УДК 681.142.2

Третий том известной монографии одного из крупнейших американских специалистов по программированию Д. Кнута (первый том вышел в издательстве "Мир" в 1976 г., второй—в 1977 г.) состоит из двух частей: "Сортировка" и "Поиск". В них подробно исследуются различные алгоритмы внутренней и внешней сортировки, изучаются методы поиска информации в таблицах на основе сравнение или преобразования ключей, даются оценки эффективности предлагаемых алгоритмов. Книга снабжена большим количеством задач и примеров разной степени трудности, существенно дополняющих основной текст.

От других руководств по программированию книга выгодно отличается строгостью изложения и широким применением математического аппарата. Вместе с тем она доступна студентам первого курса. Знакомство с двумя первыми томами желательно, но не обязательно. Каждый, кто хочет научиться квалифицированно программировать, найдет в ней много полезного.

Рассчитана на широкий круг программистов.

Редакция литературы по математическим наукам

К $\frac{20204-022}{041(01)-78}$22 − 78

ПРЕДИСЛОВИЕ РЕДАКТОРОВ ПЕРЕВОДА

Д. Э. Кнут хорошо знаком советскому читателю по переводам двух первых томов его обширной монографии "Искусство программирования для ЭВМ" и не нуждается в аттестации. Настоящая книга представляет собой третий том и посвящена алгоритмам сортировки и поиска информации.

Исторически зарождение методов машинной сортировки можно отнести еще к прошлому столетию, и за столь длительное время многие специалисты успели испробовать свои силы в этой области. Написано немало отчетов, статей, монографий. И даже в этих условиях книга Д. Кнута стала событием. По существу это энциклопедия, в которой можно найти любую справку, касающуюся алгоритмов, методов их оценок, истории вопроса и нерешенных проблем.

Нет нужды говорить о важности самой области. Практически сортировка и поиск в той или иной мере присутствуют во всех приложениях; в частности, при обработке больших объемов данных эффективность именно этих операций определяет эффективность, а иногда и работоспособность всей системы. Поэтому, как справедливо отмечает автор, книга адресована не только системным программистам, занимающимся разработкой программ сортировки и поиска информации. Можно сказать, что достаточно четкие представления об этой области нужны при решении любой задачи на ЭВМ как обязательные элементы искусства программирования.

Кроме теоретической и практической ценности, книга имеет большое методическое значение. Многие авторы и преподаватели смогут извлечь из нее новые и полезные сведения не только по существу рассматриваемых вопросов, но и по способу их изложения. Автору мастерски удается "расслоить" весь материал таким образом, что книгу можно использовать практически на любом уровне знакомства с предметом и при различной общей математической подготовленности читателя.

Перевод выполнен по изданию 1973 г. (первая редакция) с внесением многих (около 700) исправлений и добавлений, любезно предоставленных автором. Разделы с 5.1 по 5.3.2 пере- ведены. Н. И. Вьюковой; разделы с 5.3.3 по 5.5 и предисловие— А. Б. Ходулевым; главу 6 перевел В. А. Галатенко.

Ю. М. Баяковский
В. С. Штаркман

ПРЕДИСЛОВИЕ

Кулинария стала искусством, высокой наукой;
повара теперь—благородные люди.
Тит Ливий, АЬ Urbe Condita, XXXIX.vi
(Роберт Бертон, Anatomy of Melancholy, 1.2.2.2)[1]

Материал этой книги логически продолжает материал по информационным структурам, изложенный в гл. 2, поскольку здесь к уже рассмотренным концепциям структур добавляется понятие линейно упорядоченных данных. Подзаголовок "Сортировка и поиск" может привести к мысли, что эта книга предназначена лишь для системных программистов, занимающихся построением универсальных программ сортировки или связанных с выборкой информации. Однако в действительности предмет сортировки и поиска дает нам прекрасную основу для обсуждения широкого класса важных общих вопросов:

Как находить хорошие алгоритмы?

Как улучшать данные алгоритмы и программы?

Как исследовать эффективность алгоритмов математически?

Как разумно выбрать один из нескольких алгоритмов для решения конкретной задачи?

В каком смысле можно доказать, что некоторые алгоритмы являются "наилучшими из возможных"?

Как теория вычислений согласуется с практическими соображениями?

Как эффективно использовать различные виды внешней памяти—ленты, барабаны, диски—для больших баз данных? Я думаю, что на самом деле в контексте сортировки и поиска встречается практически *любой* важный аспект программирования.

Настоящий том состоит из гл. 5 и 6 монографии. В гл. 5 рассматривается сортировка (упорядочение); это очень большая тема, она разбита на две главные части—внутреннюю и внешнюю сортировку. В эту главу входят также дополнительные разделы, развивающие вспомогательную теорию перестановок (§ 5.1) и теорию оптимальных алгоритмов сортировки (§ 5.3). В гл. 6 мы имеем дело с поиском определенного элемента в таблице или файле; содержимое этой главы подразделяется на методы последовательного поиска, методы поиска со сравнением ключей, поиска с использованием свойств цифр, поиска с помощью "хеширования"; затем рассматривается более сложная задача выборки по вторичным ключам. Обе главы поразительно тесно переплетаются между собой, между их предметами имеются близкие аналогии. В

дополнение к гл. 2 рассматриваются два важных вида информационных структур, а именно приоритетные очереди (п. 5.2.3) и линейные списки, представляемые посредством сбалансированных деревьев (п. 6.2.3).

Читателю, не знакомому с первым томом этой монографии, рекомендуется обращаться к указателю обозначений (приложение В), так как некоторые из встречающихся в книге обозначений не являются общепринятыми.

Эта книга без большей части математического материала была использована мной в качестве учебника по второму курсу лекций "Структуры данных" для студентов младших и средних курсов. Математические части этой книги, особенно § 5.1, п.5.2.2, § 6.3 и 6.4, могли бы составить учебник по анализу алгоритмов для студентов средних и старших курсов. Кроме того, на основе п. 4.3.3, 4.6.3, 4.6.4, § 5.3 и п. 5.4.4 можно построить курс лекций "Сложность вычислений" для старшекурсников.

Быстрое развитие информатики и вычислительных наук задержало выход в свет этой книги почтя на три года, поскольку очень многие аспекты сортировки и поиска подвергались детальной разработке. Я очень благодарен Национальному научному фонду, Отделению военно-морских исследований, Институту обороны, фирмам IBM и Norges Almemitenskapelige Forskningsrad за постоянную поддержку моих исследований.

В подготовке этого тома к печати мне оказали помощь многие лица, особенно Эдвард А. Бендер, Кларк Э. Крэйн, Дэвид Э. Фергюсон, Роберт У. Флойд, Рональд Л. Грэхем, Леонидас Гюиба, Джон Хопкрофт, Ричард М. Карп, Гэри Д. Кнотт, Рудольф А. Крутар, Шень Линь, Воган Р. Пратт, Стефан О. Райе, Ричард П; Стэнли, Я. А. ван дер Пул и Джон У. Ренч мл., а также студенты Стэнфорда и Беркли, которым пришлось искать ошибки в рукописи.

Осло, Норвегия, *Д. Э. Кнут* сентябрь 1972

> Писатель пользуется известными привилегиями, в благодетельности которых, надеюсь, нет никаких оснований сомневаться. Так, встретив у меня непонятное место, читатель должен предположить, что под ним кроется нечто весьма полезное и глубокомысленное[2]).
>
> (Джонатан Свифт, Сказка бочки, предисловие, 1704)

ЗАМЕЧАНИЯ ОБ УПРАЖНЕНИЯХ

Упражнения, помещенные в книгах настоящей серии, предназначены как для самостоятельной проработки, так и для семинарских занятий. Трудно, если не невозможно изучить предмет, только читая теорию и не применяя полученную информацию для решения специальных задач и тем самым не заставляя себя обдумывать то, что было прочитано. Кроме того, мы лучше всего заучиваем то, что сами открываем для себя. Поэтому упражнения образуют важную часть данной работы; были предприняты определенные попытки, чтобы отобрать упражнения, в которых бы содержалось как можно больше информации и которые было бы интересно решать.

Во многих книгах легкие упражнения даются вперемешку с исключительно трудными. Зачастую это очень неудобно, так как перед тем, как приступать к решению задачи, читатель обязательно должен представлять себе, сколько времени уйдет у него на это решение (иначе он может разве только просмотреть все задачи). Классическим примером здесь является книга Ричарда Беллмана "Динамическое программирование"; это важная пионерская работа, в которой в конце каждой главы под рубрикой "Упражнения и исследовательские проблемы" дается целый ряд задач, где наряду с глубокими еще нерешенными проблемами встречаются исключительно тривиальные вопросы. Говорят, что однажды кто-то спросил д-ра Беллмана, как отличить упражнения от исследовательских проблем, и тот ответил: "Если вы можете решить задачу, это—упражнение; в противном случае это—проблема".

Можно привести много доводов в пользу того, что в книге типа этой должны быть как исследовательские проблемы, так и очень простые упражнения, и для того чтобы читателю не приходилось ломать голову над тем, какая задача легкая, а какая трудная, мы ввели "оценки", которые указывают степень трудности каждого упражнения. Эти оценки имеют следующее значение:

Оценка	Объяснение
00	Чрезвычайно легкое упражнение, на которое можно ответить сразу же, если понят материал текста, и которое почти всегда можно решить "в уме".
10	Простая задача, которая заставляет задуматься над прочитанным материалом, но не представляет никаких особых трудностей. На решение такой задачи требуется не больше одной минуты; в процессе решения могут понадобиться карандаш и бумага.

20 Задача средней трудности, позволяющая проверить, насколько хорошо понят текст. На то чтобы дать исчерпывающий ответ, требуется примерно 15–20 минут.

30 Задача умеренной трудности и/или сложности, для удовлетворительного решения которой требуется больше двух часов.

40 Очень трудная или трудоемкая задача, которую, вероятно, следует включить в план практических занятий. Предполагается, что студент может решить такую задачу, но для этого ему потребуется значительный отрезок времени; задача решается нетривиальным образом.

50 Исследовательская проблема, которая (насколько это было известно автору в момент написания) еще не получила удовлетворительного решения. Если читатель найдет решение этой задачи, его настоятельно просят опубликовать его; кроме того, автор данной книги будет очень признателен, если ему сообщат решение как можно быстрее (при условии, что оно правильно).

Интерполируя по этой "логарифмической" шкале, можно прикинуть, что означает любая промежуточная оценка. Например, оценка 17 говорит о том, что данное упражнение чуть легче, чем упражнение средней трудности. Задача с оценкой 50, если она будет решена каким-либо читателем, в следующих изданиях данной книги может иметь уже оценку 45.

Автор честно старался давать объективные оценки, но тому, кто составляет задачи, трудно предвидеть, насколько трудными эти задачи окажутся для кого-то другого; к тому же у каждого человека существует определенный тип задач, которые он решает быстрее. Надеюсь, что выставленные мной оценки дают правильное представление о степени трудности задач, но в общем их нужно воспринимать как ориентировочные, а не абсолютные.

Эта книга написана для читателей самых разных степеней математической подготовки и искушенности, поэтому некоторые упражнения предназначены только для читателей с математическим уклоном. Если в каком-либо упражнении математические понятия или результаты используются более широко, чем это необходимо для тех, кого в первую очередь интересует программирование алгоритмов, то перед оценкой такого упражнения ставится буква "M". Если для решения упражнения требуется знание высшей математики в большем объеме, чем это дано в настоящей книге, то ставятся буквы "ВМ". Пометка "ВМ" отнюдь не является свидетельством того, что данное упражнение трудное.

Перед некоторыми упражнениями стоит стрелка ">"; это означает, что данное упражнение особенно поучительно и его рекомендуется обязательно выполнить. Само собой разумеется, никто не ожидает, что читатель (или студент) будет решать все задачи, потому-то наиболее полезные из них и выделены. Это совсем не значит, что другие задачи не стоит решать! Каждый читатель должен по крайней мере попытаться решить все задачи с оценкой 10 и ниже; стрелки же помогут выбрать, какие задачи с более высокими оценками следует решить в первую очередь.

К большинству упражнений приведены ответы; они помещены в специальном разделе в конце книги. Пользуйтесь ими мудро; в ответ смотрите только после того, как вы приложили достаточно усилий, чтобы решить задачу самостоятельно, или же если для решения данной задачи у вас нет времени. Если получен собственный ответ, либо если вы действительно пытались решить задачу, только в этом случае ответ, помещенный в книге, будет поучительным и полезным. Как правило, ответы к задачам излагаются очень кратко, схематично, так как предполагается, что читатель уже честно пытался решить задачу собственными силами. Иногда в приведенном решении дается меньше информации, чем спрашивалось, чаще—наоборот. Вполне возможно, что полученный вами ответ окажется лучше ответа, помещенного в книге, или вы найдете ошибку в этом ответе; в таком случае автор был бы очень обязан, если бы вы как можно скорее подробно сообщили ему об этом. В последующих изданиях настоящей книги будет помещено уже исправленное решение вместе с именем его автора.

Сводка условных обозначений

> Рекомендуется
M С математическим уклоном
ВМ Требует знания "высшей математики"
00 Требует немедленного ответа
10 Простое (на одну минуту)
20 Средней трудности (на четверть часа)
30 Повышенной трудности.
40 Для "матпрактикума"
50 Исследовательская проблема

Упражнения

>1. [00] Что означает пометка "M20"?

2. [10] Какое значение для читателя имеют упражнения, помещаемые в учебниках?

3. [M50] Докажите, что если n—целое число, $n > 2$, то уравнение $x^n + y^n = z^n$ неразрешимо в целых положительных числах x, y,z.

5. Сортировка

Нет дела более трудного по замыслу, более сомнительного по успеху, более опасного при осуществлении, чем вводить новые порядки.

Никколо Макьявелли, "Государь" (1513)

"Но мы не успеем, просмотреть все номера автомобилей",— возразил Дрейк. "А нам и не нужно этого делать. Пол. Мы просто расположим их по порядку и поищем одинаковые".

Перри Мейсон[1]. Из "The Case of Angry Mourner" (1951)

Сортировка деревьев с использованием ЭВМ.

При таком новом, "машинном подходе" к изучению природы вы получите возможность быстро распознавать более 260 различных деревьев США, Аляски, Канады, включая пальмы, деревья пустынь и прочую экзотику. Чтобы определить породу дерева, достаточно просто вставить спицу.

Каталог "Edmund Scientific Company" (1964)

В этой главе мы изучим вопрос, который часто возникает в программировании: переразмещение элементов в возрастающем или убывающем порядке. Представьте, насколько трудно было бы пользоваться словарем, если бы слова в нем не располагались в алфавитном порядке. Точно так же от порядка, в котором хранятся элементы в памяти ЭВМ, во многом зависит скорость и простота алгоритмов, предназначенных для их обработки.

Хотя в словарях слово "сортировка" (sorting) определяется как "распределение, отбор по сортам; деление на категории, сорта, разряды", программисты традиционно используют это слово в гораздо более узком смысле, обозначая им сортировку предметов в возрастающем или убывающем порядке. Этот процесс, пожалуй, следовало бы назвать не сортировкой, а *упорядочением* (ordering), но использование этого слова привело бы к путанице из-за перегруженности значениями слова "порядок". Рассмотрим, например, следующее предложение: "Так как только два наших лентопротяжных устройства в порядке, меня призвали к порядку и обязали в срочном порядке заказать еще несколько устройств, чтобы можно было упорядочивать данные разного порядка на несколько порядков быстрее[2]. В математической терминологии это слово также изобилует значениями (порядок группы, порядок перестановки, порядок точки ветвления, отношение порядка и т. п.). Итак, слово "порядок" приводит к хаосу.

В качестве обозначения для процесса упорядочения предлагалось также слово "ранжирование"[3], но оно во многих случаях, по-видимому, не вполне отражает суть дела, особенно если присутствуют равные элементы, и, кроме того, иногда не согласуется с другими терминами. Конечно, слово "сортировка" и само имеет довольно много значений[4], но оно прочно вошло в программистский жаргон. Поэтому мы без дальнейших извинений будем использовать слово "сортировка" в узком смысле "сортировка по порядку".

Вот некоторые из наиболее важных применений сортировки:

a) Решение задачи "группировки", когда нужно собрать вместе все элементы с одинаковым значением некоторого признака. Допустим, имеется 10000 элементов, расположенных в случайном порядке, причем значения многих из них равны; и предположим, нам нужно переупорядочить файл так, чтобы элементы с равными значениями занимали соседние позиции в файле. Это, по существу, задача "сортировки" в широком смысле слова, и она легко может быть решена путем сортировки файла в узком смысле слова, а именно расположением элементов в неубывающем порядке $v_1 \le v_2 \le \ldots \le$

[2] В оригинале "Since only two of our tape drives were in working order I was ordered to order more tape units in short order, in order to order the data several orders of magnitude faster".— *Прим. перев.*

[3] В оригинале "sequencing".—*Прим. перев.*

[4] Это в большей степени относится к английскому слову "sorting". Здесь автор приводят пример: "He was sort of out sorts after sorting that sort of data". (Он был как будто не в духе после сортировки такого сорта денных), который в русском переводе не столь выразителен.— *Прим. перев.*

v_{10000}. Эффективностью, которая может быть достигнута в этой процедуре, и объясняется изменение первоначального смысла слова "сортировка".

b) Если два или более файла отсортировать в одном и том же порядке, то можно отыскать в них все общие элементы за один последовательный просмотр всех файлов, без возвратов. Это тот самый принцип, которым воспользовался Перри Мейсон для раскрытия дела об убийстве (см. эпиграфы к этой главе). Оказывается, что, как правило, гораздо экономнее просматривать список последовательно, а не перескакивая с места на место случайным образом, если только список не настолько мал, что он целиком помещается в оперативной памяти. Сортировка позволяет использовать последовательный доступ к большим файлам в качестве приемлемой замены прямой адресации.

c) Как мы увидим в гл. 6, сортировка помогает и при поиске, с ее помощью можно сделать выдачи ЭВМ более удобными для человеческого восприятия. В самом деле, листинг (напечатанный машиной документ), отсортированный в алфавитном порядке, зачастую выглядит весьма внушительно, даже если соответствующие числовые данные были рассчитаны неверно.

Хотя сортировка традиционно и большей частью использовалась для обработки коммерческих данных, на самом деле она является инструментом, полезным в самых разных ситуациях, и поэтому о нем не следует забывать; В упр. 2.3.2–17 мы обсудили ее применение для упрощения алгебраических формул. Упражнения, приведенные ниже, иллюстрируют разнообразие типичных применений сортировки.

Одной из первых крупных систем программного обеспечения, продемонстрировавших богатые возможности сортировки, был компилятор Larc Scientific Compiler, разработанный фирмой Computer Sciences Corporation в 1960 г. В этом оптимизирующем компиляторе для расширенного ФОРТРАНа сортировка использовалась весьма интенсивно, так что различные алгоритмы компиляции работали с относящимися к ним частями исходной программы, расположенными в удобной последовательности. При первом просмотре осуществлялся лексический анализ, т. е. выделение в исходной программе лексических единиц (лексем), каждая из которых соответствует либо идентификатору (имени переменной), либо константе, либо оператору и т. д. Каждая лексема получала несколько порядковых номеров. В результате сортировки по именам и соответствующим порядковым номерам все использования данного идентификатора оказывались собранными вместе. "Определяющие вхождения", специфицирующие идентификатор как имя функции, параметр или многомерную переменную, получали меньшие номера, поэтому они оказывались первыми в последовательности лексем, отвечающих этому идентификатору. Тем самым облегчалась проверка правильности употребления идентификаторов, распределение памяти с учетом деклараций эквивалентности и т. д. Собранная таким образом информация о каждом идентификаторе присоединялась к соответствующей лексеме. Поэтому не было необходимости хранить в оперативной памяти "таблицу символов", содержащую сведения об идентификаторах. После такой обработки лексемы снова сортировались по другому порядковому номеру; в результате в программе, по существу, восстанавливался первоначальный порядок, если не считать того, что арифметические выражения оказывались записанными в более удобной, "польской префиксной" форме. Сортировка использовалась и на последующих фазах компиляции—для облегчения оптимизации циклов, включения в листинг сообщений об ошибках и т. д. Короче говоря, компилятор был устроен так, что всю обработку файлов, хранящихся на барабанах, фактически можно было вести последовательно. Поэтому-то данные и снабжались такими порядковыми номерами, которые позволяли упорядочивать эти данные различными удобными способами.

Другое, более очевидное применение сортировки возникает при редактировании файлов, где каждая строка снабжена ключом. Пока пользователь вносит с клавиатуры изменения и добавления, необязательно держать весь файл в оперативной памяти. Все изменяемые строки можно позднее отсортировать (а они и так обычно в основном упорядочены) и слить с исходным файлом. Это дает возможность разумно использовать память в режиме мультипрограммирования. [Ср. с C. C. Foster, *Comp. J.*, **11** (1968), 134–137].

Поставщики вычислительных машин считают, что в среднем более 25% машинного времени систематически тратится на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Из этой статистики можно заключить, что либо (i) сортировка имеет много важных применений, либо (ii) ею часто пользуются без нужды, либо (iii) применяются в основном неэффективные алгоритмы сортировки. По-видимому, каждое из трех предположений содержит долю истины. Во всяком случае ясно, что сортировка заслуживает серьезного изучения с точки зрения ее практического использования.

Но даже если бы сортировка была почти бесполезна, нашлась бы масса других причин заняться ею! Изобретательные алгоритмы сортировки говорят о том, что она и сама по себе интересна как объект исследования. В этой области существует множество увлекательных нерешенных задач наряду с весьма немногими уже решенными.

Рассматривая вопрос в более широком плане, мы обнаружим, что алгоритмы сортировки представляют собой интересный частный пример того, как следует подходить к решению проблем программи-

рования вообще. Мы познакомимся со многими важными принципами манипулирования со структурами данных и проследим за эволюцией различных методов сортировки, причем читателю часто будет предоставляться возможность самому "открывать" те же идеи, как будто бы до него никто с подобными задачами не сталкивался. Обобщение этих частных методов позволит нам в значительной степени овладеть теми способами мышления, которые помогут создавать добротные алгоритмы для решения других проблем, связанных с ЭВМ.

Методы сортировки служат великолепной иллюстрацией идей *анализа алгоритмов*, т. е. идей, позволяющих оценивать рабочие характеристики алгоритмов, а значит, разумно выбирать среди. казалось бы, равноценных методов. Читатели, имеющие склонность к математике, найдут в этой главе немало способов оценки скорости работы алгоритмов и методов решения сложных рекуррентных соотношений. С другой стороны, изложение построено так, что читатели, не имеющие такой склонности, могут безболезненно пропускать выкладки.

Прежде чем двигаться дальше, необходимо более четко определить задачу и ввести соответствующую терминологию. Пусть надо упорядочить N элементов

$$R_1, R_2, \ldots, R_N.$$

Назовем их *записями*, а всю совокупность N записей назовем *файлом*. Каждая запись R_j имеет *ключ* K_j, который и управляет процессом сортировки. Помимо ключа, запись может содержать дополнительную, "сопутствующую информацию", которая не влияет на сортировку, но всегда остается в этой записи.

Отношение порядка "$<$" на множестве ключей вводится таким образом, чтобы для любых трех значений ключей a, b, c выполнялись следующие условия:

i) справедливо одно и только одно из соотношений $a < b$, $a = b$, $b < a$ (закон трихотомии);

ii) если $a < b$ и $b < c$, то $a < c$ (закон транзитивности).

Эти два свойства определяют математическое понятие *линейного упорядочения*, называемого еще *совершенным упорядочением*. Любое множество с отношением "$<$", удовлетворяющим свойствам (i) и (ii), поддается сортировке большинством методов, описанных в этой главе, хотя некоторые из них годятся только для числовых и буквенных ключей с обычным отношением порядка.

Задача сортировки—найти такую перестановку записей p(1) p(2) ... p(N), после которой ключи расположились бы в неубывающем порядке:

$$K_{\mathrm{p}(1)} \le K_{p(2)} \le \cdots \le K_{p(n)}. \tag{1}$$

Сортировка называется *устойчивой*, если она удовлетворяет дополнительному условию, что записи с одинаковыми ключами остаются в прежнем порядке, т. е.

$$p(i) < p(j), \qquad \text{если } K_{p(i)} = K_{p(j)} \text{ и } i < j. \tag{2}$$

В ряде случаев может потребоваться физически перемещать записи в памяти так, чтобы их ключи, были упорядочены; в других случаях достаточно создать вспомогательную таблицу, которая некоторым образом описывает перестановку и обеспечивает доступ к записям в соответствии с порядком их ключей.

Некоторые методы сортировки предполагают существование величин "∞" и "$-\infty$" или одной из них. Величина "∞" считается больше, а величина "$-\infty$" меньше любого ключа:

$$-\infty < K_j < \infty, \qquad 1 \le j \le N. \tag{3}$$

Эти величины используются в качестве искусственных ключей, а также как граничные признаки. Равенство в (3), вообще говоря, исключено. Если же оно тем не менее допускается, алгоритмы можно модифицировать так, чтобы они все-таки работали, хотя нередко при этом их изящество и эффективность отчасти утрачиваются.

Обычно сортировку подразделяют на два класса: *внутреннюю*, когда все записи хранятся в быстрой оперативной памяти, и *внешнюю*, когда все они там не помещаются. При внутренней сортировке имеются более гибкие возможности для построения структур данных и доступа к ним, внешняя же показывает, как поступать в условиях сильно ограниченного доступа.

Достаточно хороший общий алгоритм затрачивает на сортировку N записей время порядка $N \log N$; при этом требуется около $\log N$ "проходов" по данным. Как мы увидим в п. 5.3.1, это минимальное время. Так, если удвоить число записей, то и время при прочих равных условиях возрастет немногим более чем вдвое. (На самом деле, когда N стремится к ∞, время"растет как $N(\log N)^2$, если все ключи различны, так как и размеры ключей увеличиваются с ростом N; но практически N всегда остается ограниченным.)

Упражнения

(ПЕРВАЯ ЧАСТЬ)

1. [M20] Докажите, что из законов трихотомии и транзитивности вытекает *единственность* перестановки p(1), p(2), ..., p(N), если сортировка устойчива.
2. [21] Пусть каждая запись R. некоторого файла имеет *два* ключа: "большой ключ" K_j и "малый ключ" k_j, причем оба множества ключей линейно упорядочены. Тогда можно обычным способом ввести "лексикографический порядок" на множестве пар ключей (K, k):

$$(K_i, k_i) < (K_j, k_j), \qquad \text{если } K_i < K_j \text{ или если } K_i = K_j \text{ и } k_i < k_j.$$

Некто (назовем его мистер A) отсортировал этот файл сначала по большим ключам, получив n групп записей с одинаковыми большими ключами в каждой группе:

$$K_{p(1)} = \cdots = K_{p(i_1)} < K_{p(i_1+1)} = \cdots = K_{p(i_2)} < \cdots < K_{p(i_{n-1}+1)} = \cdots = K_{p(i_n)},$$

где $i_n = N$. Затем каждую из n групп $R_{p(i_{j-1}+1)}, \ldots, R_{p(i_j)}$ он отсортировал по малым ключам.

Тот же исходный файл взял мистер B и отсортировал его сначала по малым ключам, а потом получившийся файл отсортировал по большим ключам.

Взяв тот же исходный файл, мистер C отсортировал его один раз, пользуясь лексикографическим порядком между парами ключей (K_j, k_j).

Получилось. ли у всех троих одно и то же?

3. [M25] Пусть на множестве $K_1, \ldots, K_N$ определено отношение $<$, для которого закон трихотомии выполняется, а закон транзитивности— *нет*. Докажите, что и в этом случае возможна устойчивая сортировка записей, т. е. такая, что выполняются условия (1) и (2); на самом деле существуют по крайней мере три расположения записей, удовлетворяющих этим условиям!
4. [15] Мистер Тупица (программист) захотел узнать, находится ли в ячейке `A` машины `MIX` число, большее числа из ячейки `B`, меньшее или же равное ему. Он написал

```
LDAA
SUBB
```

а потом проверил, какой результат получился в `rA`: положительное число, отрицательное или нуль. Какую серьезную ошибку он допустил и как должен был поступить?

5. [17] Напишите `MIX`-подпрограмму для сравнения ключей, занимающих несколько слов, исходя из следующих условий:

Вызов:	`JMP COMPARE`
Состояние при входе:	`rI1` $= n$; `CONTENTS`$(A+k) = a_k$, `CONTENTS`$(B+k) = b_k$ при $1 \le k \le n$; предполагается, что $n \ge 1$.
Состояние при выходе:	`CI` $= +1$, если $(a_n, \ldots, a_1) > (b_n, \ldots, b_1)$; `CI` $= 0$, если $(a_n, \ldots, a_1) = (b_n, \ldots, b_1)$; `CI` $= -1$, если $(a_n, \ldots, a_1) < (b_n, \ldots, b_1)$; `rX` и `rI1`, возможно, изменились.

Здесь отношение $(a_n, \ldots, a_1) < (b_n, \ldots, b_1)$ обозначает лексикографическое упорядочение слева направо, т. е. существует индекс j, такой, что $a_k = b_k$ при $n \ge k > j$, но $a_j < b_j$.

6. [30] В ячейках `A` и `B` содержатся соответственно числа a и b. Покажите, что можно написать `MIX`-программу, которая бы вычисляла $\min(a, b)$ и записывала результат в ячейку `C`, *не пользуясь командами перехода.* (*Предостережение:* поскольку арифметическое переполнение невозможно обнаружить без команд перехода, разумно так построить программу, чтобы переполнение не могло возникнуть ни при каких значениях a и b)
7. [M27] Какова вероятность того, что после сортировки в неубывающем порядке `N` независимых равномерно распределенных на отрезке $[0, 1]$ случайных величин r-е от начала число окажется $\le x$?

Упражнения

УПРАЖНЕНИЯ (ВТОРАЯ ЧАСТЬ) В каждом из этих упражнений поставлена задача, с которой может столкнуться программист. Предложите "хорошее" решение задачи, предполагая, что имеется сравнительно небольшая оперативная память и около полудюжины, лентопротяжных устройств (этого количества достаточно для сортировки).

1. [75] Имеется лента, на которой записан миллион слов данных. Как определить, сколько на этой ленте различных слов?

2. [18] Вообразите себя в роли Управления внутренних доходов Министерства финансов США. Вы получаете миллионы "информационных" карточек от организаций о том, сколько, денег они выплатили различным лицам, и миллионы "налоговых" карточек от различных лиц об их доходах. Как бы вы стали отыскивать людей, которые сообщили не обо всех своих доходах?
3. [M25] *(Транспонирование матрицы.)* Имеется магнитная лента, содержащая миллион слов, которые представляют собой элементы 1000×1000-матрицы, записанные по строкам: $a_{1,1}\ a_{1,2}\ \ldots a_{1,1000}\ a_{2,1}\ \ldots a_{2,1000}\ \ldots a_{1000,1000}$. Ваша задача—получить ленту, на которой элементы этой матрицы были бы записаны по столбцам: $a_{1,1}\ a_{2,1}\ \ldots a_{1000,1}\ a_{1,2} \ldots a_{1,2} \ldots a_{1000,2}\ \ldots a_{1000,1000}$ (Постарайтесь сделать не более десяти просмотров данных.)
4. [M26] В вашем распоряжении довольно большой файл из N слов. Как бы вы его "перетасовали" случайным образом?

>5. [24] В неком университете работает около 1000 преподавателей и имеется 500 комитетов. Считается, что каждый преподаватель является членом по крайней мере двух комитетов. Вам нужно подготовить с помощью машины удобочитаемые списки членов всех комитетов. Вы располагаете колодой из приблизительно 1500 перфокарт, сложенных произвольным образом и содержащих следующую информацию:

Членские карточки: колонка 1—пробел; колонки 2–18—фамилия с последующими пробелами; колонки 19–20—инициалы; колонки 21–23—номер первого Комитета; колонки 24–26—номер второго комитета; . . . ; колонки 78–80— номер двадцатого комитета (если нужно) или пробелы.

Комитетские карточки: колонка 1—"*"; колонки 2–77—название комитета; колонки 78–80—номер комитета. Как вы должны действовать? (Опишите свой метод достаточно подробно.)

6. [20] Вы работаете с двумя вычислительными системами, в которых по-разному упорядочены литеры (буквы и цифры). Как заставить первую ЭВМ сортировать файлы с буквенно-цифровой информацией, предназначенные для использования на второй ЭВМ?
7. [18] Имеется довольно большой список людей, родившихся в США, с указанием штата, в котором они родились. Как подсчитать число людей, родившихся в каждом штате? (Предполагается, что ни один человек не указан в списке более одного раза.)
8. [20] Чтобы облегчить внесение изменений в большие программы, написанные на ФОРТРАНе, вы хотите написать программу, выпечатывающую таблицу "перекрестных ссылок". Входными данными для нее служит программа на ФОРТРАНе, а в результате получается листинг исходной программы, снабженный указателем всех случаев употребления каждого идентификатора (т. е. имени) в программе. Как написать такую программу?
9. [33] (Сортировка каталожных карточек.) Способы составления алфавитных каталогов в разных библиотеках несколько отличаются друг от друга. В следующем "алфавитном" списке содержатся рекомендации, взятые из правил регистрации и хранения каталожных карточек Американской библиотечной ассоциации (Чикаго, 1942):

Текст карточки	**Замечания**
R. Accademia nazionale dei Lincei, Rome	В названиях иностранных (кроме британских) учреждений слово "royalty" (королевский) игнорируется
1812; ein historischer roman.	Achtzehnhundert zwölf
Bibliothèque d'histoire révolutionnaire.	Во французском тексте апостроф рассматривается как пробел
Bibliothèque des curiosités.	Надстрочные знаки игнорируются
Brown, Mrs. J. Crosby	Указание положения (Mrs.) игнорируется
Brown, John	Фамилии с датами следуют за фамилиями без дат . . .
Brown, John, mathematician	. . . которые упорядочиваются по
Brown, John, of Boston	описательным словам
Brown, John, 1715–1766	Одинаковые фамилии упорядочиваются по датам рождения
BROWN, JOHN, 1715–1766	Работы "о нем" идут после его работ
Brown, John, d. 1811	Иногда год рождения определяется приблизительно
Brown, Dr. John, 1810–1882	Указание положения (Dr.) игнорируется
Brown-Williams, Reginald Makepeace	Дефис рассматривается как пробел
Brown America.	Названия книг идут после составных фамилий
Brown & Dallison's Nevada directory.	& в английском тексте превращается в "and"
Brownjohn, Alan	

Den', Vladimir Éduardovich, 1867–	Апостроф в именах игнорируется
The den.	Артикль в начале текста игнорируется
Den lieben süssen mädeln.	. . . если существительное стоит в именительном падеже
Dix, Morgan, 1827–1908	Фамилии идут раньше других слов
1812 ouverture.	Dix-huit cent douze
Le XIXe siècle français.	Dix-neuvième
The 1847 issue of U. S. stamps.	Eighteen forty-seven
1812 overture.	Eighteen twelve
I am a mathematician,	(by Norbert Weiner)
IBM journal of research and development.	Аббревиатуры рассматриваются как ряд однобуквенных слов
ha-I ha-eḥad.	Артикль в начале текста игнорируется
Ia; a love story.	Знаки препинания в названиях игнорируются
International Business Machines Corporation	
al-Khuwārizmī, Muḥammad ibn Mūsā, *fl.* 813–846	Начальное "al-" в арабских именах игнорируется
Labour; a magazine for all workers.	Заменяется на "Labor"
Labor research association	
Labour, *see* Labor	Ссылка на другую карточку в картотеке
McCall's cookbook	Апостроф в английском тексте игнорируется
McCarthy, John, 1927–	Mc = Mac
Machine-independent computer programming.	Дефис рассматривается как пробел
MacMahon, Maj. Percy Alexander, 1854–1929	Указание положения (Maj) игнорируется
Mrs. Dalloway.	"Mrs. "= "Mistress"
Mistress of mistresses.	
Royal society of London	
St. PetersburgerZeitung.	"St."= "Saint" даже в немецком тексте
Saint-Saëns, Camille, 1835–1921	Дефис рассматривается как пробел
Ste. Anne des Monts, Quebec	Sainte
Seminumerical algorithms.	
Uncle Tom's cabin.	
U.S. Bureau of the census.	"U.S." = "United States"
Vandermonde, Alexander Théophile, 1735–1796	
Van Valkenburg, Mac Elwyn, 1921–	Пробел после префикса в фамилиях игнорируется
Von Neumann, John, 1903–1957	
The whole art of legerdemain.	
Who's afraid of Virginia Woolf?	Апостроф в английском тексте игнорируется
Wijngaarden, Adriaan van, 1916–	Фамилия никогда не начинается с малой буквы

(У большинства из этих правил есть исключения; кроме того, существует много других правил, которые здесь не упомянуты.)

Предположим, вам пришлось сортировать большое количество таких карточек с помощью вычислительной машины и впоследствии обслуживать огромную картотеку, причем у вас нет возможности изменить уже сложившиеся порядки заполнения карточек. Как бы вы организовали информацию, чтобы упростить операции включения новых карточек и сортировки?

10. [M21] *(Дискретные логарифмы.)* Пусть известно, что p—(довольно большое) простое число, а a—первообразный корень по модулю p. Следовательно, для любого b в диапазоне $1 \le b < p$ существует единственное n, такое, что $a^n \bmod p = b$, $1 \le n <$ p. Как по заданному b найти n менее чем за $O(n)$ шагов? [Указание. Пусть $m = \lceil\sqrt{p}\rceil$. Попытайтесь решить уравнение $a^{mn_1} \equiv ba^{-n_2} \pmod p$ при $0 \le n_1, n_2 < m$.]
11. [M25] (Э. Т. Паркер.) Эйлер выдвинул предположение, что уравнение

$$u^6 + v^6 + w^6 + \mathrm{x}^6 + y^6 = z^6$$

не имеет решений (за исключением тривиальных) среди целых неотрицательных чисел u, v, w, x, y, z, когда по крайней мере четыре переменные равны нулю. Помимо этого, он предполагал, что уравнение

$$x_1^n + \cdots + x_{n-1}^n = x_n^n$$

не имеет нетривиальных решений при $n \geq 3$, но это предположение было опровергнуто: с помощью вычислительной машины найдено тождество $27^5 + 84^5 + 110^5 + 133^5 = 144^5$; см. Л. Дж. Лэндер, Т. Р. Паркин и Дж. Л. Селфридж, *Math. Comp.*, **21** (1967), 446–459. Придумайте, как можно было бы использовать сортировку для поиска примеров, опровергающих предположение Эйлера при $n = 6$.

>12. [24] Файл содержит большое количество 30-разрядных двоичных слов: $x_1, \ldots x_N$. Придумайте хороший способ нахождения в нем всех *дополнительных* пар (x_i, x_j). (Два слова называются дополнительными, если второе содержит 0 во всех разрядах, в которых были 1 в первом слове, и обратно; таким образом, они дополнительны тогда и только тогда, когда их сумма равна $(11\ldots1)_2$, если они рассматриваются как двоичные числа.)

>13. [25] Имеется файл. содержащий 1000 30-разрядных двоичных слов $x_1, \ldots, x_{1000}$. Как бы вы стали составлять список всех пар (x_i, x_j), таких, что x_i отличается от x_j не более чем в двух разрядах?

14. [22] Как бы вы поступили при отыскании всех пятибуквенных анаграмм, таких, как **CARET, CARTE, CATER, CRATE, REACT, TRACE; CRUEL, LUCRE, ULCER; DOWRY, ROWDY, WORDY?** [Если бы вы, скажем, захотели узнать, существуют ли в английском яаыке наборы из десяти или более анаграмм, кроме замечательной серии:

APERS, ASPER,PARES,PARSE,PEARS, PRASE, RAPES, REAPS, SPARE, SPEAR,

к которой можно добавить еще французское слово **APRES.**]

15. [M28] Пусть даны описания весьма большого числа направленных графов. Каким путем можно сгруппировать *изоморфные* графы? (Два направленных графа называются изоморфными, если существует взаимно однозначное соответствие между их вершинами и взаимно однозначное соответствие между их дугами, причем эти соответствия сохраняют инцидентность вершин и дуг.)

16. [30] В некоторой группе из 4096 человек каждый имеет около 100 знакомых. Был составлен список всех пар людей, знакомых между собой. (Это отношение симметрично, т. е. если x знаком с y, то и y знаком с x. Поэтому список содержит примерно 200000 пар.) Придумайте алгоритм, который по заданному k выдавал бы все *клики* из k человек. (Клика — это группа людей, в которой все между собой знакомы.) Предполагается, что слишком больших клик не бывает.

17. [30] Три миллиона человек с различными именами были уложены рядом непрерывной цепочкой от Нью-Йорка до Калифорнии. Каждому из них дали листок бумаги, на котором он написал свое имя и имя своего ближайшего западного соседа. Человек, находившийся в самой западной точке цепи, не понял, что ему делать, и выкинул свой листок. Остальные 2999999 листков собрали в большую корзину и отправили в Национальный архив, в Вашингтон, округ Колумбия. Там содержимое корзины тщательно перетасовали и записали на магнитные ленты.

Специалист по теории информации определил, что имеется достаточно информации для восстановления списка людей в исходном порядке. Специалист по программированию нашел способ сделать это менее чем за 1000 просмотров лент с данными, используя лишь последовательный доступ к файлам на лентах и небольшое количество памяти с произвольным доступом. Как ему это удалось?

[Другими словами, как, имея расположенные произвольным образом пары (x_i, x_{i+1}), $1 \leq i < N$, где все x_i различны, получить последовательность $x_1, x_2, \ldots, x_N$, применяя лишь методы последовательной обработки данных, пригодные для работы с магнитными лентами? Это задача сортировки в случае, когда трудно определить, какой из двух ключей предшествует другому; мы уже поднимали этот вопрос в упр. 2.2.3–25.]

5.1. * КОМБИНАТОРНЫЕ СВОЙСТВА ПЕРЕСТАНОВОК

Перестановкой конечного множества называется некоторое расположение его элементов в ряд. Перестановки особенно важны при изучении алгоритмов сортировки, так как они служат для представления неупорядоченных исходных данных. Чтобы исследовать эффективность различных методов сортировки, нужно уметь подсчитывать число перестановок, которые вынуждают повторять некоторый шаг алгоритма определенное число раз.

Мы, конечно, уже не раз встречались с перестановками в гл. 1, 2 и 3. Например, в п. 1.2.5 обсуждались два основных теоретических метода построения $n!$ перестановок из n объектов; в п. 1.3.3 проанализированы некоторые алгоритмы, связанные с циклической структурой и мультипликативными свойствами перестановок; в п. 3.3.2 изучены их отрезки монотонности. Цель настоящего параграфа—изучить некоторые другие свойства перестановок и рассмотреть общий случай, когда допускается наличие одинаковых элементов. Попутно мы узнаем многое о комбинаторной математике.

Свойства перестановок настолько красивы, что представляют и самостоятельный интерес. Удобно будет дать их систематическое изложение в одном месте, а не разбрасывать материал по всей главе. Читателям, не имеющим склонности к математике, а также тем, кто жаждет поскорее добраться до самих методов сортировки, рекомендуется перейти сразу к § 5.2, потому что настоящий параграф *непосредственного* отношения к сортировке почти не имеет.

5.1.1. *Инверсии

Пусть $a_1\ a_2\ \ldots\ a_n$ —перестановка множества $\{1, 2, \ldots, n\}$. Если $i < j$, а $a_i > a_j$, то пара (a_i, a_j) называется инверсией перестановки; например, перестановка 3 1 4 2 имеет три инверсии: $(3, 1)$, $(3, 2)$ и $(4, 2)$. Каждая инверсия—это пара элементов, "нарушающих порядок"; следовательно, единственная перестановка, не содержащая инверсий,—это отсортированная перестановка $1\ 2\ \ldots n$. Такая связь с сортировкой и есть главная причина нашего интереса к инверсиям, хотя это понятие уже использовалось нами при анализе алгоритма динамического распределения памяти (см. упр. 2.2.2–9).

Понятие инверсии ввел Г. Крамер в 1750 г. [Intr. à l'Analyse des Lignes Courbes algébriques (Geneva, 1750), 657–659; ср. с Томас Мюир, Theory of Determinants, 1 (1906), 11–14] в связи со своим замечательным правилом решения линейных уравнений. В сущности, он определил детерминант $n \times n$-матрицы следующим образом:

$$\det \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nn} \end{pmatrix} = \sum (-1)^{I(a_1 a_2 \ldots a_n)} x_{1a_1} x_{2a_2} \ldots x_{na_n},$$

где сумма берется по всем перестановкам $a_1\ a_2\ \ldots\ a_n$, а $I(a_1 a_2 \ldots a_n)$—число инверсий в перестановке.

Таблицей инверсий перестановки $a_1\ a_2\ \ldots\ a_n$ называется последовательность чисел $b_1\ b_2\ \ldots\ b_n$, где b_j—число элементов, *бóльших* j и расположенных *левее* j. (Другими словами, b_j—число инверсий, у которых второй элемент равен j.) Например, таблицей инверсий перестановки

$$5\ 9\ 1\ 8\ 2\ 6\ 4\ 7\ 3 \tag{1}$$

будет

$$2\ 3\ 6\ 4\ 0\ 2\ 2\ 1\ 0, \tag{2}$$

поскольку 5 и 9 расположены левее 1; 5, 9, 8—левее 2 и т.д., всего 20 инверсий. По определению

$$0 \le b_1 \le n-1, 0 \le b_2 \le n-2, \ldots, 0 \le b_{n-1} \le 1, b_n = 0. \tag{3}$$

Пожалуй, наиболее важный факт, касающийся перестановок, и установленный Маршаллом Холлом, это то, что *таблица инверсий единственным образом определяет соответствующую перестановку.* [См. Proc. Symp. Applied Math., **6** (American Math. Society, 1956), 203.] Из любой таблицы инверсий $b_1\ b_2\ \ldots\ b_n$, удовлетворяющей условиям (3), можно однозначно восстановить перестановку, которая порождает данную таблицу, путем последовательного определения относительного расположения элементов n, $n-1$, $\ldots$, 1 (в этом порядке). Например, перестановку, соответствующую (2), можно построить следующим образом: выпишем число 9; так как $b_8 = 1$, то 8 стоит правее 9. Поскольку $b_7 = 2$, то 7 стоит правее 8 и 9. Так как $b_6 = 2$, то 6 стоит правее двух уже выписанных нами чисел; таким образом, имеем

$$9\ 8\ 6\ 7.$$

Припишем теперь 5 слева, потому что $b_5 = 0$; помещаем 4 вслед за четырьмя из уже записанных чисел, 3—после шести выписанных чисел (т. е. в правый конец) и получаем

$$5\ 9\ 8\ 6\ 4\ 7\ 3.$$

Вставив аналогичным образом 2 и 1, придем к (1).

Такое соответствие важно, потому что часто можно заменить задачу, сформулированную в терминах перестановок, эквивалентной ей задачей, сформулированной в терминах таблиц инверсий, которая, возможно, решается проще. Рассмотрим, например, самый простой вопрос: сколько существует перестановок множества $\{1, 2, \ldots, n\}$? Ответ должен быть равен числу всевозможных таблиц инверсий, а их легко пересчитать, так как b_1 можно выбрать n различными способами, b_2 можно независимо от $b-1$ выбрать $n-1$ способами, $\ldots$, b_n—одним способом; итого $n(n-1)\ldots 1 = n!$ различных таблиц инверсий. Таблицы инверсий пересчитать легче, потому что b независимы, в то время как a должны быть все различны.

В п. 1:2.10 мы исследовали задачу о числе локальных максимумов перестановки, если читать ее справа налево; иными словами, требовалось подсчитать количество элементов, каждый из которых больше любого из следующих после него. (Например, правосторонние максимумы в (1)—это 9, 8, 7 и 3.) Оно равно количеству индексов j, таких, что $b_j = n - j$. Так как b_1 принимает значение $n-1$ с вероятностью $1/n$, b_2 (независимо) принимает значение $n-2$ с вероятностью $1/(n-1)$ и т.д., то из рассмотрения инверсий ясно, что среднее число правосторонних максимумов равно

$$\frac{1}{n} + \frac{1}{n-1} + \cdots + 1 = H_n.$$

Аналогичным способом легко получить и соответствующую производящую функцию.

Другие применения таблиц инверсий встретятся далее в этой главе в связи с конкретными алгоритмами сортировки.

Ясно, что если поменять местами *соседние* элементы перестановки, то общее число инверсий увеличится или уменьшится на единицу. На рис. 1 показаны 24 перестановки множества $\{1, 2, 3, 4\}$; линиями соединены перестановки, отличающиеся друг от друга положением двух соседних элементов; двигаясь вдоль линии вниз, мы .увеличиваем число инверсий на единицу. Следовательно, число инверсий в перестановке п равно длине нисходящего пути из 1 2 3 4 в n на рис. 1; все такие пути должны иметь одинаковую длину.

Заметим, что эту диаграмму можно рассматривать как трехмерное твердое тело—"усеченный октаэдр", имеющий 8 шестиугольных и 6 квадратных граней. Это один из равномерных многогранников, которые обсуждал Архимед (см. упр.,10).

Picture: Рис. 1. Усеченный октаэдр, на котором показано изменение числа инверсий, когда меняются местами два соседних элемента перестановки;

Не следует путать "инверсии" перестановок с обратными перестановками. Вспомним, что перестановку можно записывать в виде двух строк

$$\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a_1 & a_2 & a_3 & \dots & a_n \end{pmatrix}; \tag{4}$$

обратной к этой перестановке называется перестановка a'_1, a'_2, ... a'_n, которая получается, если в (4) поменять местами строки, а затем упорядочить столбцы в возрастающем порядке по верхним элементам:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ 1 & 2 & 3 & \dots & n \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a'_1 & a'_2 & a'_3 & \dots & a'_n \end{pmatrix}; \tag{5}$$

Например, обратной к перестановке 5 9 1 8 2 6 4 7 3 будет перестановка 3 5 9 7 1 6 8 4 2, так как

$$\begin{pmatrix} 5 & 9 & 1 & 8 & 2 & 6 & 4 & 7 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 9 & 7 & 1 & 6 & 8 & 4 & 2 \end{pmatrix}.$$

Можно дать другое определение обратной перестановки: $a'_j = k$ тогда и только тогда, когда $a_k = j$.

Обратную перестановку впервые ввел Х. А. Роте [в K.F. Hindenburg(ed.)., Sammlung combinatorisch-analytischer Abhandlungen, 2 (Leipzig, 1800), 263–305]; он заметил интересную связь между обратными перестановками и инверсиями: *обратная перестановка содержит ровно столько же инверсий, сколько исходная*. Роте дал не самое простое доказательство этого факта, но оно поучительно и притом довольно красиво. Построим таблицу размера $n \times n$ и поставим точки в j-й клетке i-й строки, если $a_i = j$. После этого расставим крестики во всех клетках, снизу от которых (в том же столбце) и справа (в той же строке) есть точки. Например, для 5 9 1 8 2 6 4 7 3 диаграмма будет такой:

Количество крестиков равно числу инверсий, так как нетрудно видеть, что b_j равно числу крестиков в j-м столбце. Если мы теперь транспонируем диаграмму (поменяв ролями строки и столбцы), то получим диаграмму для обратной по отношению к исходной перестановки; значит, число крестиков (число инверсий) одинаково в обоих случаях. Роте использовал этот факт для доказательства того, что детерминант матрицы не меняется при транспонировании.

Для анализа некоторых алгоритмов сортировки необходимо знать число перестановок n элементов, содержащих ровно k инверсий. Обозначим это число через $I_n(k)$; в табл. 1 приведены первые несколько значений этой функции.

Таблица 1

	Перестановки с k инверсиями											
n	$I_n(0)$	$I_n(1)$	$I_n(2)$	$I_n(3)$	$I_n(4)$	$I_n(5)$	$I_n(6)$	$I_n(7)$	$I_n(8)$	$I_n(9)$	$I_n(10)$	$I_n(11)$
1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	1	2	2	1	0	0	0	0	0	0	0	0
4	1	3	5	6	5	3	1	0	0	0	0	0
5	1	4	9	15	20	22	20	15	9	4	1	0
6	1	5	14	29	49	71	90	101	101	90	71	49

Из рассмотрения таблицы инверсий $b_1\ b_2\ \ldots b_n$ ясно, что $I_k(0) = 1$, $I_n(1) = n - 1$ и что выполняется свойство симметрии:

$$I_n\left(\binom{n}{2} - k\right) = I_n(k) \tag{6}$$

Далее, так как значения b можно выбирать независимо друг от друга, то нетрудно видеть, что производящая функция

$$G_n(z) = I_n(0) + I_n(1)z + I_n(2)z^2 + \cdots \tag{7}$$

удовлетворяет соотношению $G_n(z) = (1 + z + \cdots + z^{n-1})G_{n-1}(z)$; следовательно, она имеет довольно, простой вид

$$(1 + z + \cdots + z^{n-1}) \ldots (1 + z)(1) = (1 - z^n) \ldots (l - z^2)(l - z)/(l - z)^n. \tag{8}$$

С помощью этой производящей функции можно легко продолжить табл. 1 и убедиться, что числа, расположенные под ступенчатой линией в таблице, удовлетворяют соотношению

$$I_n(k) = I_n(k - 1) + I_{n-1}(k) \qquad \text{при } k < n. \tag{9}$$

(Для чисел *над* ступенчатой линией это соотношение *не* выполняется.) Более сложные рассуждения (см. упр. 14) показывают, что на самом деле имеют место формулы

$$\begin{aligned}
I_n(2) &= \binom{n}{2} - 1, n \geq 2; \\
I_n(3) &= \binom{n+1}{3} - \binom{n}{1}, n \geq 3; \\
I_n(4) &= \binom{n+2}{4} - \binom{n+1}{2}, n \geq 4; \\
I_n(5) &= \binom{n+3}{5} - \binom{n+2}{3} + 1, n \geq 5.
\end{aligned}$$

Общая формула для $I_n(k)$ содержит около $1.6\sqrt{k}$ слагаемых:

$$\begin{aligned}
I_n(k) = &\binom{n+k-2}{k} - \binom{n+k-3}{k-2} + \binom{n+k-6}{k-5} + \binom{n+k-8}{k-7} - \cdots \\
&+ (-1)^j \left(\binom{n+k-u_j-1}{k-u_j} + \binom{n+k-u_j-j-1}{k-u_j-j} \right) + \cdots, \qquad n \geq k,
\end{aligned} \tag{10}$$

где $u_j = (3j^2 - j)/2$ — так называемое "пятиугольное число".

Разделив $G_n(z)$ на $n!$, получим производящую функцию $g_n(z)$ распределения вероятностей числа инверсий в случайной перестановке n элементов. Она равна произведению

$$g_n(z) = h_1(z)h_2(z) \ldots h_n(z), \tag{11}$$

где $h_k(z) = (1 + z + z^2 + \cdots + z^{k-1})/k$ — производящая функция равномерного распределения случайной величины, принимающей целые неотрицательные значения, меньшие k. Отсюда

$$\begin{aligned} \operatorname{mean}(g_n) &= \operatorname{mean}(h_1) + \operatorname{mean}(h_2) + \cdots + \operatorname{mean}(h_n) = \\ &= 0 + \frac{1}{2} + \cdots + \frac{n-1}{2} = \frac{n(n-1)}{4}; \qquad (12) \\ \operatorname{var}(g_n) &= \operatorname{var}(h_1) + \operatorname{var}(h_2) + \cdots + \operatorname{var}(h_n) = \\ &= 0 + \frac{1}{4} + \cdots + \frac{n^2-1}{12} = \frac{n(2n+5)(n-1)}{72} \qquad (13) \end{aligned}$$

Таким образом, среднее число инверсий довольно велико—около $\frac{1}{4}n^2$; стандартное отклонение также весьма велико—около $\frac{1}{6}n^{3/2}$.

В качестве интересного завершения изучения инверсий рассмотрим одно замечательное открытие, принадлежащее П. А. Мак-Магону [*Amer. J. Math.*, **35** (1913), 281–322]. Определим *индекс* перестановки $a_1\, a_2 \ldots a_n$ как сумму всех j, таких, что $a_j > a_{j+1}$, $1 \leq < n$. Например, индекс перестановки 5 9 1 8 2 6 4 7 3 равен $2 + 4 + 6 + 8 = 20$. Индекс случайно совпал с числом инверсий. Если составить список всех 24 перестановок множества $\{1, 2, 3, 4\}$, а именно

Перестановка	Индекс	Инверсии	Перестановка	Индекс	Инверсии
1 2 3 4	0	0	3 1 2 4	1	2
1 2 4 3	3	1	3 1 4 2	4	3
1 3 2 4	2	1	3 2 1 4	3	3
1 3 4 2	3	2	3 2 4 1	4	4
1 4 2 3	2	2	3 4 1 2	2	4
1 4 3 2	5	3	3 4 2 1	5	5
2 1 3 4	1	1	4 1 2 3	1	3
2 1 4 3	4	2	4 1 3 2	4	4
2 3 1 4	2	2	4 2 1 3	3	4
2 3 4 1	3	3	4 2 3 1	4	5
2 4 1 3	2	8	4 3 1 2	3	5
2 4 3 1	5	4	4 3 2 1	6	6

то видно, что *число перестановок, имеющих данный индекс k, равно числу перестановок, имеющих k инверсий.*

На первый взгляд этот факт может показаться почти очевидным, однако после некоторых размышлений он начинает казаться чуть ли не мистическим, и не видно никакого простого прямого его доказательства. Мак-Магон нашел следующее остроумное косвенное доказательство: пусть $J(a_1 a_2 \ldots a_n)$—индекс перестановки $a_1\, a_2 \ldots a_n$, и соответствующая производящая функция есть

$$H_n(z) = \sum z^{J(a_1 a_2 \ldots a_n)}, \qquad (14)$$

где сумма берется по всем перестановкам множества $\{1, 2, \ldots, n\}$. Мы хотели бы доказать, что $H_n(z) = G_n(z)$. Для этого определим взаимно однозначное соответствие между n-ками $(q_1, q_2, \ldots, q_n)$ неотрицательных целых чисел, с одной стороны, и упорядоченными парами n-ок

$$((a_1, a_2, \ldots, a_n), (p_1, p_2, \ldots, p_n)),$$

с другой стороны; здесь $a_1\ a_2 \ldots a_n$—перестановка множества $\{1, 2, \ldots, n\}$ и $p_1 \geq p_2 \geq \ldots \geq p_n \geq 0$.. Это соответствие будет удовлетворять условию

$$q_1 + q_2 + \cdots + q_n = J(a_1 a_2 \ldots a_n) + (p_1 + p_2 + \cdots + p_n). \qquad (15)$$

Производящая функция $\sum z^{q_1+q_2+\cdots+q_n}$, где сумма берется по всем n-кам неотрицательных целых чисел $(q_1, q_2, \ldots, q_n)$, равна $Q_n(z) = 1/(1-z)^z$; а производящая функция $\sum z^{p_1+p_2+\cdots+p_n}$, где сумма берется по всем n-кам целых чисел $(p_1, p_2, \ldots p_n)$, таких, что $p_1 \geq p_2 \geq \ldots \geq p_n \geq 0$, равна, как показано в упр. 15,

$$P_n(z) = 1/(1-z)(1-z^2)\ldots(1-z^n). \qquad (16)$$

Существование взаимно однозначного соответствия, которое удовлетворяет условию (15) и которое мы собираемся установить, доказывает равенство $Q_n(z) = H_n(z)P_n(z)$, т.е.

$$H_n(z) = Q_n(z)/P_n(z) = G_n(z).$$

Требуемое соответствие определяется с помощью алгоритма "сортировки". Начав с пустого списка, при $k = 1, 2, \ldots, n$ (в таком порядке) вставляем в этот список числа q_k следующим образом: пусть после $k-1$ шагов в списке содержатся элементы $p_1, p_2, \ldots, p_{k-1}$, где $p_1 \geq p_2 \geq \ldots \geq p_{k-1}$, и определена перестановка $a_1\ a_2 \ldots a_n$ множества $\{n, n-1, \ldots, n-k+2\}$. Пусть j—единственное целое число, такое, что $p_j > q_k \geq p_{j+1}$; если $q_k \geq p_1$, то полагаем $j = 0$, а если $p_{k-1} > q_k$, то полагаем $j = k-1$. Вставим теперь q_k в список между p_j и p_{j+1}, а целое число $(n-k+1)$—в перестановку между a_j, и a_{j+1}. Проделав это для всех k, получим перестановку $a_1\ a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$ и n-ку чисел $(p_1, p_2, \ldots, p_n)$, таких, что $p_1 \geq p_2 \geq \ldots \geq p_n \geq 0$ и

$$p_j > p_{j+1}, \qquad \text{если } a_j > a_{j+1}.$$

Наконец, для $1 \leq j < n$ вычтем единицу из всех чисел $p_1, \ldots, p_j$ при всех j, таких, что $a_j > a_{j+1}$. Полученная пара $((\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_n), (p_1, p_2, \ldots, p_n))$ удовлетворяет условию (15).

Пусть, например, $n = 6$ и $(q_1, \ldots, q_6) = (3, 1, 4, 0, 0, 1)$. Построение происходит следующим образом:

k	$p_1 \ldots p_k$	$a_1 \ldots a_k$
1	3	6
2	3 1	6 5
3	4 3 1	4 6 5
4	4 3 1 0	4 6 5 3
5	4 3 1 0 0	4 6 5 2 3
6	4 3 1 1 0 0	4 6 1 5 2 3

После заключительной корректировки получаем $(p_1, \ldots, p_6) = (2, 1, 0, 0, 0, 0)$.

Нетрудно проверить, что этот процесс обратим; таким образом, требуемое соответствие установлено и теорема Мак-Магона доказана. Аналогичное взаимно однозначное соответствие встретится нам в п. 5.1.4.

Упражнения

1. [10] Какова таблица инверсий для перестановки 2 7 1 8 4 5 9 3 6? Какой перестановке соответствует таблица инверсий 5 0 1 2 1 2 0 0?
2. [M15] Решением задачи Иосифа, сформулированной в упр. 1.3.2–22, является перестановка множества $\{1, 2, \ldots n\}$; решение для приведенного там примера $(n = 8, m = 4)$—перестановка 5 4 6 1 3 8 7 2. Соответствующая этой перестановке таблица инверсий—3 6 3 1 0 0 1 0. Найдите простое рекуррентное соотношение для элементов $b_1\ b_2 \ldots b_n$ таблицы инверсий в общей задаче Иосифа для n человек, если казнят каждого m-го человека.
3. [18] Пусть перестановке $a_1\ a_2 \ldots a_n$ соответствует таблица инверсий $b_1\ b_2 \ldots b_n$; какой перестановке $\bar{a}_1\ \bar{a}_2 \ldots \bar{a}_n$ соответствует таблица инверсий

$$(n-1-b_1)(n-2-b_2)\ldots(0-b_n)?$$

>4. [20] Придумайте алгоритм, годный для реализации на ЭВМ, который по данной таблице инверсий $b_1\ b_2 \ldots b_n$, удовлетворяющей условиям (3), строил бы соответствующую перестановку $a_1\ a_2 \ldots a_n$. [*Указание:* вспомните методы работы со связанной памятью.]
5. [35] Для выполнения на типичной ЭВМ алгоритм из упр. 4 требует времени, приблизительно пропорционального n^2; можно ли создать алгоритм, время работы которого было бы существенно меньше n^2?
>6. [26] Придумайте алгоритм вычисления таблицы инверсий $b_1\ b_2 \ldots b_n$, соответствующей данной перестановке $a_1\ a_2 \ldots a_n$ множества $\{1, 2, \ldots, \mathrm{n}\}$,, время работы которого на типичной ЭВМ было бы порядка $n \log n$.
7. [20] Помимо таблицы $b_1\ b_2 \ldots b_n$, определенной в этом пункте, можно определить некоторые другие типы таблиц инверсий, соответствующих данной перестановке $a_1\ a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$. В этом упражнении мы рассмотрим три других типа таблиц инверсий, которые возникают в приложениях.

Пусть c_j—число инверсий, *первая* компонента которых равна j, т. е. число элементов, меньших j и расположенных *правее* j. [Перестановке (1) соответствует таблица 0 0 0 1 4 2 1 5 7; ясно, что$0 \leq c_j < j$.] Пусть $B_j = b_{a_j}$ и $C_j = c_{a_j}$.

Покажите, что при $1 \leq j \leq n$ справедливы неравенства $0 \leq B_j < j$ и $0 \leq C_j \leq n-j$; покажите также, что перестановку $a_1\ a_2 \ldots a_n$ можно однозначно определить, если задана или таблица $c_1\ c_2 \ldots c_n$, или $B_1\ B_2 \ldots B_n$, или $C_1\ C_2 \ldots C_n$.

8. [M24] Сохраним обозначения упр. 7; пусть $a'_1\ a'_2\ \ldots a'_n$—перестановка, обратная к $a_1\ a_2\ \ldots a_n$ и пусть соответствующие ей таблицы инверсий— $b'_1\ b'_2\ \ldots b'_n$, $c'_1\ c'_2\ \ldots c'_n$, $B'_1\ B'_2\ \ldots B'_n$ и $C'_1\ C'_2\ \ldots C'_n$. Найдите как можно больше интересных соотношений между a_j, b_j, c_j, B_j, C_j, a'_j, b'_j, c'_j, B'_j, C'_j.

>9. [M21] Докажите, что в обозначениях упр. 7 перестановка $a_1\ a_2\ \ldots a_n$ обратна самой себе тогда и только тогда, когда $b_j = C_j$ при $1 \le j \le n$.

10. [BM20] Рассмотрите рис. 1 как многогранник в трехмерном пространстве. Чему равен диаметр усеченного октаэдра (расстояние между вершинами 1234 и 4321), если все ребра имеют единичную длину?

11. [M25] (a) Пусть $\pi = a_1\ a_2 \ldots a_n$—перестановка множества $\{1, 2, \ldots, n\}$, $E(x) = \{(a_i, a_j) | i < j, a_i > a_j\}$—множество ее инверсий, а
$$\overline{E}(\pi) = \{(a_i, a_j) | i > j, a_i > a_j\}$$
—множество ее ""неинверсий". Докажите, что $E(\pi)$ и $\overline{E}(\pi)$ транзитивны. [Множество S упорядоченных пар называется *транзитивным*, если для любых (a, b) и (b, c), принадлежащих S, пара (a, c) также принадлежит S.] (b) Обратно, пусть E—любое транзитивное подмножество множества $T = \{(x, y) | 1 \le y < x \le n\}$, дополнение которого $T \backslash E$ транзитивно. Докажите, что существует перестановка π, такая, что $E(\pi) = E$.

12. [M28] Используя обозначения предыдущего упражнения, докажите, что если π_1 и π_2—перестановки, а E—минимальное транзитивное множество, содержащее $E(\pi_1) \cup E(\pi_2)$, то $\overline{E}$—тоже транзитивное множество. [Следовательно, если мы будем говорить, что π_1 находится "над" π_2, когда $E(\pi_1) \subseteq E(\pi_2)$, то определена решетка перестановок; существует единственная "самая низкая" перестановка, находящаяся "над" двумя данными перестановками. Диаграмма решетки при $n = 4$ представлена на рис. 1.]

13. [M23] Известно, что в разложении определителя половина членов выписывается со знаком +, а половина—со знаком −. Другими словами, при $n \ge 2$ перестановок с *четным* числом инверсий ровно столько же, сколько с *нечетным*. Покажите, что вообще при $n \ge m$ количество перестановок с числом инверсий, конгруэнтным $t \bmod m$, равно $n!/m$, независимо от того, каково целое число t.

14. [M24] *(Ф. Франклин.)* Разбиение числа n на k различных частей— это представление n в виде суммы $n = p_1 + p_2 + \cdots + p_k$, где $p_1 > p_2 > \ldots > p_k > 0$. Например, разбиения числа 7 на различные части таковы: 7, $6 + 1$, $5 + 2$,

Picture: Рис. 2. Соответствие Франклина между разбиениями на различные части.

$4+3$, $4+2+1$. Пусть $f_k(n)$—число разбиений n на k различных частей. Докажите, что $\sum_k (-1)^k f_k(n) = 0$, если только n не представляется в виде $(3j^2 \pm j)/2$ при некотором неотрицательном целом j; в этом случае сумма равна $(-1)^j$. Например, для $n = 7$ сумма равна $-1 + 3 - 1 = 1$, потому что $7 = (3 \cdot 2^2 + 2)/2$. [*Указание.* Представьте разбиения в виде массива точек, в i-й строке которого имеется p_i точек, $1 \le i \le k$. Найдите наименьшее j, такое, что $p_{j+1} < p_j - 1$, и обведите крайние правые точки первых j строк. Если $j < p_k$, то эти j точек можно, как правило изъять из массива, повернуть на 45° и поместить в новую, $(k + 1)$-ю строку. С другой стороны, если $j \ge p_k$, то обычно можно изъять из массива k-ю строку точек, повернуть ее на 45° и поместить справа от обведенных точек (рис. 2). В результате этого процесса в большинстве случаев разбиения с четным числом строк и разбиения с нечетным числом строк группируются в пары, таким образом, в сумме надо учитывать только непарные разбиения.]

Замечание. В качестве следствия получаем формулу Эйлера
$$(1 - z)(1 - z^2)(1 - z^3)\ldots = 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + \cdots = \sum_{-\infty < j < \infty} (-1)^j z^{(3j^2 + j)/2}.$$

Так как производящая функция для обычных разбиений (не обязательно на различные части) равна $\sum p(n) z^n = 1/(1 - z)(1 - z^2)(1 - z^3)\ldots$, то получаем неочевидное рекуррентное соотношение для числа разбиений $p(n) = p(n - 1) + p(n - 2) - p(n - 5) - p(n - 7) + p(n + 12) + p(n + 15) - \cdots$.

15. [M29] Докажите, что (16)—производящая функция для числа разбиений на не более чем n частей, т. е. докажите, что коэффициент при z^m в $1/(1 - z)(1 - z^2)\ldots(1 - z^n)$ равен числу способов представить m в виде суммы $p_1 + p_2 + \cdots + p_n$, где $p_1 \ge p_2 \ge \ldots p_n \ge 0$. [*Указание.* Нарисуйте точки, как в упр. 14, и покажите, что существует взаимно однозначное соответствие между n-ками чисел $(p_1, p_2, \ldots, p_n)$, такими, что $p_1 \ge p_2 \ge \ldots \ge p_n \ge 0$, и последовательностями $(P_1, P_2, P_3, \ldots)$, такими, что $n \ge P_1 \ge P_2 \ge P_3 \ge \ldots \ge 0$, обладающее тем свойством, что $p_1 + p_2 + \cdots + p_n = P_1 + P_2 + P_3 + \cdots$. Иными словами, разбиениям на не более чем n частей соответствуют разбиения на части, не превосходящие n.]

16. [M25](Л. Эйлер.) Докажите следующие тождества, интерпретируя обе части соотношений в терминах разбиений:

$$\prod_{k\ge 0}\frac{1}{(1-q^kz)}=\frac{1}{(1-z)(1-qz)(1-q^2z)\dots}=$$
$$=1+\frac{z}{1-q}+\frac{z^2}{(1-q)(1-q^2)}+\cdots=\sum_{n\ge 0}z^n/\prod_{1\le k\le n}(1-q^k).$$
$$\prod_{k\ge 0}(1+q^kz)=(1+z)(1+qz)(1+q^2z)\dots=$$
$$=1+\frac{z}{1-q}+\frac{z^2q}{(1-q)(1-q^2)}+\cdots=$$
$$=\sum_{n\ge 0}z^nq^{n(n-1)/2}/\prod_{1\le k\le n}(1-q^k).$$

17. [20] Каковы 24 четверки (q_1,q_2,q_3,q_4), для которых в соответствии Мак-Магона, определенном в конце этого пункта, $(p_1,p_2,p_3,p_4)=(0,0,0,0)$?
18. [M30] (Т. Хиббард, *CACM*, **6** (1963), 210.) Пусть $n>0$, и предположим, что последовательность n-битовых целых чисел X_0, ..., X_{2^n-1} длины 2^n получена случайным образом, причем каждый бит каждого числа независимо принимает значение 1 с вероятностью p. Рассмотрим последовательность $X_0\oplus 0$, $X_1\oplus 1$, ..., $X_{2^n-1}\oplus(2^n-1)$, где $\oplus$—операция "исключающее или" над бинарными представлениями. Так, если $p=0$, то последовательность будет 0, 1, ..., 2^n-1, а если $p=1$, то она будет 2^n-1, ..., 1, 0; если же $p=\frac{1}{2}$, то каждый элемент последовательности—случайное число между 0 и 2^n-1. Вообще же при разных p это хороший способ получения последовательности случайных целых чисел со смещенным числом инверсий, в то время как распределение элементов последовательности, рассматриваемой как единое целое, равномерно.
Определите среднее число инверсий в такой последовательности как функцию от вероятности p.
19. [M36] (Д. Фоата.) Дайте прямое доказательство теоремы Мак-Магона об индексах: найдите точное взаимно однозначное соответствие, которое переводит перестановку n элементов, имеющую индекс k, в перестановку, имеющую k инверсий и тот же самый крайний правый элемент.
20. [M43] Следующее знаменитое тождество, принадлежащее Якоби [Fundamenta Nova Theoriæ Functionum Ellipticorum (1829), § 64], лежит в основе многих замечательных соотношений, содержащих эллиптические функции:

$$\prod_{k\ge 1}(1-u^kv^{k-1})(1-u^{k-1}v^k)(1-u^kv^k)=$$
$$=(1-u)(1-v)(1-uv)(1-u^2v)(1-uv^2)(1-u^2v^2)\dots=$$
$$=1-(u+v)+(u^3v+uv^3)-(u^6v^3+u^3v^6)+\cdots=$$
$$=1+\sum_{n\ge 1}(-1)^n(u^{(n+1)n/2}v^{(n-1)n/2}+u^{(n-1)n/2}v^{(n+1)n/2}).$$

Если, например, положить $u=z$, $v=z^2$, то получится формула Эйлера из упр. 14. Если положить $z=\sqrt{u/v}$, $q=\sqrt{uv}$, то получим

$$\prod_{k\ge 1}(1-q^{2k-1}z)(1-q^{2k-1}z^{-1}(1-q^{2k})=\sum_{-\infty<n<\infty}(-1)^nz^nq^{n^2}.$$

Существует ли комбинаторное доказательство тождества Якоби, аналогичное доказательству Франклина для частного случая упр. 14? (Таким образом, нужно рассмотреть "комплексные разбиения"

$$m+ni=(p_1+q_1i)+(p_2+q_2i)+\cdots+(p_k+q_ki),$$

где p_j+q_ji—различные ненулевые комплексные числа; p_j, q_j—неотрицательные целые числа, причем $|p_j-q_j|\le 1$. Согласно тождеству Якоби, число таких представлений с четными k равно числу представлений с нечетными k, если только m и n не являются соседними треугольными числами!) Какими еще замечательными свойствами обладают комплексные разбиения?

>21. [M25] (Г. Д. Кнотт.) Покажите, что перестановку $a_1\dots a_n$ можно получить с помощью стека в смысле упр. 2.2.1–5 или 2.3.1–6 тогда и только тогда, когда $C_j\le C_{j+1}+1$ при $1\le j<n$ (см. обозначения в упр. 7).

22. [M28] (К. Мейер.) Мы знаем, что если m и n—взаимно простые числа, то последовательность $(m \bmod n)\ (2m \bmod n) \ldots ((n-1)m \bmod n)$ представляет собой перестановку множества $\{1, 2, \ldots, n-1\}$. Покажите, что число инверсий в этой перестановке можно выразить через суммы Дедекинда (ср. с п. 3.3.3).

5.1.2. *Перестановки мультимножества

До сих пор мы рассматривали перестановки *множества* элементов; это частный случай перестановок *мультимножества*. (Мультимножество—это то же самое, что и множество, но в нем могут содержаться одинаковые элементы. Некоторые основные свойства мультимножеств обсуждались в п. 4.6.3.)

Рассмотрим, например, мультимножество

$$M = \{a, a, a, b, b, c, d, d, d, d\}, \tag{1}$$

в котором содержится 3 элемента a, 2 элемента b, 1 элемент c и 4 элемента d. Повторения элементов можно указать и другим способом:

$$M = \{3 \cdot a, 2 \cdot b, c, 4 \cdot d\}. \tag{2}$$

Перестановка мультимножества — это некоторое расположение его элементов в ряд, например

$$c\ a\ b\ d\ d\ a\ b\ d\ a\ d.$$

С другой стороны, такую последовательность можно назвать цепочкой букв, содержащей 3 буквы a, 2 буквы b, 1 букву c и 4 буквы d.

Сколько существует перестановок мультимножества M? Если бы мы рассматривали все элементы M как различные, обозначив их a_1, a_2, a_3, b_1, b_2, c_1, d_1, d_2, d_3, d_4, то получили бы $10! = 3\ 628\ 800$ перестановок, но после отбрасывания индексов многие из них оказались бы одинаковыми. Фактически каждая перестановка M встретилась бы ровно $3!2!1!4! = 288$ раз, поскольку в любой перестановке M индексы при буквах a можно расставить 31 способами, при b (независимо)—2! способами, при c—одним способом, а при d—соответственно 4! способами. Поэтому число перестановок M равно

$$\frac{10!}{3!2!1!4!} = 12\ 600.$$

В применении к общему случаю те же рассуждения доказывают, что число перестановок любого мультимножества равно мультиномиальному коэффициенту

$$\binom{n}{n_1, n_2, \ldots} = \frac{n!}{n_1! n_2! \ldots},$$

где n_1—число элементов первого типа, n_2—число элементов второго типа и т. д., а $n = n_1 + n_2 + \cdots$—общее число элементов.

Количество перестановок множества было известно еще в древние времена. В древнееврейской Книге Творения (около 100 г. н. э.)[5], наиболее раннем литературном произведении иудейского философского мистицизма, даны верные значения первых семи факториалов, после чего говорится: "Продолжай и получишь числа, которые уста не могут произнести, а ухо не может воспринять." [Sefer Yezirah, ed. by R. Mordecai Atia (Jerusalem: Sh. Monson, 1962), стих 52 (стр. 107–108); ср. также с Solomon Gandz, Studies in Hebrew Astronomy and Mathematics (New York: Ktav, 1970), 494–496. Книга Творения была основана на считавшихся важными отношениях между семью планетами, семью согласными звуками с двойным произношением, семью отверстиями в голове человека и семью днями сотворения мира.] Это первый известный в истории подсчет числа перестановок. Второй встречается в индийском классическом произведении Ануйогадвара-сутра (около 500 г. н. э.), правило 97, где приводится формула числа перестановок шести элементов, которые не расположены ни в возрастающем, ни в убывающем порядке:

$$6 \times 5 \times 4 \times 3 \times 2 \times 1 - 2.$$

[См. G. Chakravarti, *Bull. Calcutta Math. Soc.*, **24** (1932), 79–88. Ануйогадвара-сутра—одна из книг канонов джайнизма, религиозной секты, распространенной в Индии.]

Соответствующее правило для мультимножеств впервые, по- видимому, встречается в книге Лилавати, написанной Бхаскарой Ахарьей (ок. 1150 г.), разд. 270–271. У Бхасжары это правило сформулировано весьма сжато и проиллюстрировано лишь двумя простыми примерами $\{2, 2, 1, 1\}$ и $\{4, 8, 5, 5, 5\}$. В

[5] Книга Творения (Йоцира)—одна из основополагающих книг каббалистики.— *Прим. перев.*

результате в английском переводе это правило не сформулировано корректно, впрочем, имеются некоторые сомнения относительно того, понимал ли сам Бхаскара, о чем он говорил. Вслед за этим правилом Бхаскара приводит интересную формулу

$$\frac{(4+8+5+5+5)\times 120\times 11111}{5\times 6}$$

для суммы 20 чисел $48\,555 + 45855 + \cdots$.

Верное правило для нахождения числа перестановок в случае, когда только один элемент может повторяться, найдено независимо немецким ученым иезуитом Атанасиусом Кирхером в его многотомном труде о музыке Musurgia Universalis (Rome, 1650), том 2, стр. 5–7. Кирхера интересовал вопрос о количестве мелодий, которые можно создать из данного набора нот; для этого он придумал то, что называл "музарифметикой". На стр. 18–21 своего труда он дает верное значение числа перестановок мультимножества $\{m\cdot C, n\cdot D\}$ при нескольких значениях m и n, хотя описал он свой метод вычислений лишь для случая $n = 1$.

Общее правило (3) появилось позже в книге Жана Престэ Elémens de Mathématiques (Paris, 1675), стр. 351–352, которая содержит одно из первых изложений комбинаторной математики, написанных в западной Европе. Престэ верно сформулировал правило для случая произвольного мультимножества, но проиллюстрировал его лишь простым примером $\{a, a, b, b, c, c\}$. Он особо отметил, что деление на *сумму* факториалов, которое он считал естественным обобщением правила Кирхера, было бы ошибкой. Несколько лет спустя Джон Валлис в своей книге Treatise of Algebra (Oxford, 1685), том 2, стр. 117–118, обсудил это правило несколько более подробно.

В 1965 г. Доминик Фоата ввел одно интересное понятие, так называемое "соединительное произведение"[6], которое позволило распространить многие известные результаты, касающиеся обычных перестановок, на общий случаи перестановок мультимножества. [См. Publ. Inst. Statistique, Univ. Paris, **14** (1965), 81–241. а также Lecture Notes in Math., **85** (Springer, 1969).] Предполагая, что элементы мультимножества каким-то способом линейно упорядочены, можно рассмотреть *двустрочное обозначение*, например

$$\begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}.$$

Здесь верхняя строка содержит элементы M в неубывающем порядке, и нижняя—это сама перестановка. *Соединительное произведение* $\alpha\top\beta$ двух перестановок мультимножеств α и β—это перестановка, которая получается, если (a) взять двустрочные обозначения для α и β, (b) записать соответствующие строки в одну и (c) отсортировать столбцы так, чтобы элементы верхней строки расположились в неубывающем порядке. Сортировка должна быть устойчивой в том смысле, что взаимное расположение элементов нижней строки сохраняется, если соответствующие элементы верхней строки равны. Например, $c\ a\ d\ a\ b \top b\ d\ d\ a\ d = c\ a\ b\ d\ d\ a\ b\ d\ a\ d$, так как

$$\begin{pmatrix} a & a & b & c & d \\ c & a & d & a & b \end{pmatrix} \top \begin{pmatrix} a & b & d & d & d \\ b & d & d & a & d \end{pmatrix} = \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}. \tag{5}$$

Нетрудно видеть, что операция соединительного произведения ассоциативна, т. е.

$$\alpha\top\beta\top\gamma = \alpha\top(\beta\top\gamma), \tag{6}$$

и что она подчиняется законам сокращения

$$\text{если } \pi\top\alpha = \pi\top\beta, \text{ то } \alpha = \beta,$$
$$\text{если } \alpha\top\pi = \beta\top\pi, \text{ то } \alpha = \beta.$$

Существует "единичный элемент"

$$\alpha\top\varepsilon = \varepsilon\top\alpha = \alpha, \tag{8}$$

где ε—пустая перестановка, "расположение в ряд" элементов пустого множества. Закон коммутативности, вообще говоря, не выполняется (см. упр. 2), тем не менее

$$\alpha\top\beta = \beta\top\alpha, \qquad \text{если } \alpha \text{ и } \beta \text{ не содержат общих букв.} \tag{9}$$

[6] В оригинале—"intercalation product".—*Прим. перев.*

Аналогичным способом и понятие *цикла* можно распространить на случай, когда элементы могут повторяться. Будем записывать в виде

$$(x_1\ x_2\ \ldots\ x_n) \tag{10}$$

перестановку, двустрочное представление которой получается путем устойчивой сортировки столбцов

$$\begin{pmatrix} x_1 & x_2 & \ldots & x_n \\ x_2 & x_3 & \ldots & x_1 \end{pmatrix} \tag{11}$$

по верхним элементам. Например,

$$(d\ \ b\ \ d\ \ d\ \ a\ \ c\ \ a\ \ a\ \ b\ \ d) = \begin{pmatrix} d & b & d & d & a & c & a & a & b & d \\ b & d & d & a & c & a & a & b & d & d \end{pmatrix} = \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}$$

так что перестановка (4) фактически является циклом. Мы могли бы описать этот цикл словесно, сказав что-нибудь вроде "d переходит в b, переходит в d, переходит в d, переходит в ... переходит в d и возвращается обратно". Заметим, что эти обобщенные циклы не обладают всеми свойствами обычных циклов; $(x_1\ x_2\ \ldots\ x_n)$ не обязательно то же самое, что и $(x_2\ \ldots\ x_n\ x_1)$.

В п. 1.3.3 мы выяснили, что каждую перестановку множества можно единственным с точностью до порядка сомножителей образом представить в виде произведения непересекающихся циклов, где произведение перестановок определяется законом композиции. Легко видеть, что *произведение непересекающихся циклов—то же самое, что их соединительное произведение;* это наводит на мысль о том, что можно будет обобщить полученные ранее результаты, если найти единственное (в каком-то смысле) представление и для произвольной перестановки мультимножества в виде соединительного произведения циклов. В действительности существуют по крайней мере два естественных способа сделать это, и каждый из них имеет важные приложения.

Равенство (5) дает один способ представления $c\ a\ b\ d\ d\ a\ b\ d\ a\ \ d$ в виде соединительного произведения более коротких перестановок; рассмотрим общую задачу о нахождении всех разложений $\pi = \alpha \top \beta$ данной перестановки π. Для исследования этого вопроса полезно рассмотреть конкретную перестановку, скажем

$$\pi = \begin{pmatrix} a & a & b & b & b & b & b & c & c & c & d & d & d & d & d \\ d & b & c & b & c & a & c & d & a & d & d & b & b & b & d \end{pmatrix}, \tag{12}$$

Если можно записать π в виде $\alpha \top \beta$, где α содержит по крайней мере одну букву a, то самое левое a в верхней строке двустрочного представления α должно оказаться над d, значит, перестановка α должна содержать по крайней мере одну букву d. Если взглянуть теперь на самое левое d в верхней строке α, то увидим точно так же, что оно должно оказаться над d, значит, в α должны содержаться по меньшей мере *две* буквы d. Посмотрев на второе d, видим, что α содержит также b. Одно-единственное предположение о том, что α есть левый сомножитель π, содержащий букву a, приводит к такому промежуточному результату:

$$\begin{pmatrix} a & & b & & d & d & \\ & \ldots & & \ldots & & & \ldots \\ d & & & & d & b & \end{pmatrix}. \tag{13}$$

Продолжая рассуждать точно так же и далее, обнаружим, что буква b в верхней строке (13) должна оказаться над c и т. д. В конце концов этот процесс вновь приведет нас к букве a, и мы сможем, если захотим, отождествить ее с первой буквой a. Только что проведенное рассуждение, по существу, доказывает,

что любой левый сомножитель в разложении перестановки (12), содержащий a, имеет вид $(d\ d\ b\ c\ d\ b\ b\ c\ a) \top \alpha'$, где α'— некоторая перестановка. (Удобно записывать a не в начале, а в конце цикла; это допустимо, поскольку буква a только одна.) Аналогично, если бы мы предположили, что α содержит букву b, то вывели бы, что $\alpha = (c\ d\ d\ b) \top \alpha''$, где α''—некоторая перестановка.

В общем случае эти рассуждения показывают, что *если есть какое-нибудь разложение* $\alpha \top \beta = \pi$, *где* α *содержит данную букву* y, *то существует единственный цикл вида*

$$(x_1\ \ldots\ x_n\ y), \qquad n \geq 0, x_1, \ldots, x_n \neq y, \tag{14}$$

который является левым сомножителем в разложении перестановки α. Такой цикл легко отыскать, зная π и y; это самый короткий левый сомножитель в разложении перестановки π, содержащий букву y. Одно из следствий этого наблюдения дает

Теорема A. *Пусть элементы мультимножества M линейно упорядочены отношением "$<$". Каждая перестановка π мультимножества M имеет единственное представление в виде соединительного произведения*

$$\pi = (x_{11} \ldots x_{1n_1}y_1) \top (x_{21} \ldots x_{2n_2}y_2) \top \ldots (x_{t1} \ldots x_{tn_t}y_t), \quad t \geq 0, \tag{15}$$

удовлетворяющее следующим двум условиям:

$$\begin{gathered} y_1 \leq y_2 \leq \ldots \leq y_t; \\ y_i < x_{ij} \text{ при } 1 \leq j \leq n_i,\ 1 \leq i \leq t. \end{gathered} \tag{16}$$

(Иными словами, в каждом цикле последний элемент меньше любого другого, и последние элементы циклов образуют неубывающую последовательность.)

Доказательство При $\pi = \varepsilon$ получим требуемое разложение, положив $t = 0$. В противном случае пусть y_1—минимальный элемент π; определим $(x_{11} \ldots x_{1n_1}y_1)$—самый короткий левый сомножитель разложения π, содержащий y_1, как в рассмотренном примере. Теперь $\pi = (x_{11} \ldots x_{1n_1}y_1) \top \rho$, где ρ—некоторая перестановка; применив индукцию по длине перестановки, можем написать

$$\rho = (x_{21} \ldots x_{2n_2}y_2) \top \ldots \top (x_{t1} \ldots x_{tn_t}y_t), t \geq 1,$$

где условия (16) выполнены. Тем самым доказано существование такого разложения.

Докажем единственность разложения (15), удовлетворяющего условиям (16). Ясно, что $t = 0$ тогда и только тогда, когда π— пустая перестановка ε. При $t > 0$ из (16) следует, что y_1—минимальный элемент перестановки π и что $(x_{11} \ldots x_{1n_1}y_1$—самый короткий левый сомножитель, содержащий y_1. Поэтому $(x_{11} \ldots x_{1n_1}y_1)$ определяется однозначно; доказательство единственности такого представления завершается применением индукции и законов сокращения (7). ∎

Например, "каноническое" разложение перестановки (12), удовлетворяющее данным условиям, таково:

$$(d\ d\ b\ c\ d\ b\ b\ c\ a) \top (b\ a) \top (c\ d\ b) \top (d), \tag{17}$$

если $a < b < c < d$.

Важно отметить, что на самом деле в этом определении можно отбросить скобки и знаки операции $\top$, и это не приведет к неоднозначности! Каждый цикл заканчивается появлением наименьшего из оставшихся элементов. Таким образом, наше построение связывает с исходной перестановкой

$$\pi' = d\ d\ b\ c\ d\ b\ b\ c\ a\ b\ a\ c\ d\ b\ d$$

перестановку

$$\pi = d\ b\ c\ b\ c\ a\ c\ d\ a\ d\ d\ b\ b\ b\ d.$$

Если в двустрочном представлении π содержится столбец вида $\binom{y}{x}$, где $x < y$, то в связанной с π перестановке присутствует соответствующая пара соседних элементов $\ldots y\ x \ldots$. Так, в нашем примере π содержит три столбца вида $\binom{d}{b}$, а в π' трижды встречается пара $d\ b$. Вообще из этого построения вытекает замечательная

Теорема B. *Пусть M—мультимножество. Существует взаимно однозначное соответствие между перестановками M, такое, что если π соответствует π', то выполняются следующие условия:*

a) крайний левый элемент π' равен крайнему левому элементу π;
b) для всех пар участвующих в перестановке элементов (x, y),

таких, что $x < y$, число вхождений столбца $\binom{y}{x}$ в двустрочное представление перестановки π равно числу случаев, когда в перестановке π' элемент x следует непосредственно за y.

Если M—множество, то это, по существу, "нестандартное соответствие", обсуждавшееся в конце п. 1.3.3, с незначительными изменениями. Более общий результат теоремы B полезен при подсчете числа перестановок специальных типов, поскольку часто проще решить задачу с ограничениями, наложенными на двустрочное представление, чем эквивалентную задачу с ограничениями на пары соседних элементов.

П. А. Мак-Магон рассмотрел задачи этого типа в своей выдающееся книге Combinatory Analysis (Cambridge Univ. Press, 1915), том 1, стр. 168–186. Он дал конструктивное доказательство теоремы B в частном случае, когда M содержит элементы лишь двух различных типов, скажем a и b, его построение для этого случая, по существу, совпадает с приведенным здесь, но представлено в совершенно ином виде.

Для случая трех различных элементов a, b, c Мак-Магон дал сложное неконструктивное доказательство теоремы B; общий случай впервые доказал Фоата в 1965 г.

В качестве нетривиального примера применения теоремы B найдем число цепочек букв a, b, c, содержащих ровно

$$\begin{aligned} A &\text{ вхождений буквы } a;\\ B &\text{ вхождений буквы } b;\\ C &\text{ вхождений буквы } c;\\ k &\text{ вхождений пары стоящих рядом букв } c\ a;\\ l &\text{ вхождений пары стоящих рядом букв } c\ b;\\ m &\text{ вхождений пары стоящих рядом букв } b\ a; \end{aligned} \tag{18}$$

Из теоремы следует, что это то же самое, что найти число двустрочных массивов вида

$$\begin{array}{c} \overbrace{\begin{pmatrix} a & \dots & a \\ \sqcup & \dots & \sqcup \end{pmatrix}}^{A} \overbrace{\begin{pmatrix} b & \dots & b \\ \sqcup & \dots & \sqcup \end{pmatrix}}^{B} \overbrace{\begin{pmatrix} c & \dots & c \\ \sqcup & \dots & \sqcup \end{pmatrix}}^{C} \\ \underbrace{\qquad\qquad}_{A-k-m \text{ букв } a} \underbrace{\qquad\qquad}_{m \text{ букв } a} \underbrace{\qquad\qquad}_{k \text{ букв } a} \\ \underbrace{\qquad\qquad\qquad\qquad}_{B-l \text{ букв } b} \underbrace{\qquad\qquad}_{l \text{ букв } b} \\ \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{C \text{ букв } c} \end{array} \tag{19}$$

Буквы a можно расположить во второй строке

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k} \qquad \text{способами;}$$

после этого буквы b можно разместить в оставшихся позициях

$$\binom{B+k}{B-l}\binom{C-k}{l} \qquad \text{способами.}$$

Остальные свободные места нужно заполнить буквами c; следовательно, искомое число равно

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l}. \tag{20}$$

Вернемся к вопросу о нахождении всех разложений данной перестановки. Существует ли такой объект, как "простая" перестановка, которая не разлагается на множители, отличные от нее самой и ε? Обсуждение, предшествующее теореме A, немедленно приводит к выводу о том, что *перестановка будет простой тогда и только тогда, когда она есть цикл без повторяющихся элементов,* так как, если перестановка является таким циклом, наше рассуждение доказывает, что не существует левых множителей, кроме ε и самого цикла. Если же перестановка содержит повторяющийся элемент y, то всегда можно выделить нетривиальный цикл в качестве левого сомножителя, в котором элемент y встречается всего однажды.

Если перестановка не простая, то ее можно разлагать на все меньшие и меньшие части, пока не будет получено произведение простых перестановок. Можно даже показать, что такое разложение единственно с точностью до порядка записи коммутирующих сомножителей.

Теорема C. *Каждую перестановку мультимножества можно записать в виде произведения*

$$\sigma_1 \top \sigma_2 \top \dots \top \sigma_t, \quad t \geq 0, \tag{21}$$

где σ_j—циклы, не содержащие повторяющихся элементов. Это представление единственно в том смысле, что любые два таких представления одной и той же перестановки можно преобразовать одно в другое, последовательно меняя местами соседние непересекающиеся циклы.

Термин "непересекающиеся циклы" относится к циклам, не имеющим общих элементов. В качестве примера можно проверить, что перестановка

$$\begin{pmatrix} a & a & b & b & c & c & d \\ b & a & a & c & d & b & c \end{pmatrix}$$

разлагается на множители ровно пятью способами:

$$\begin{aligned}(a\ b)\top(a)\top(c\ d)\top(b\ c) &= (a\ b)\top(c\ d)\top(a)\top(b\ c) = \\ &= (a\ b)\top(c\ d)\top(b\ c)\top(a) = \\ &= (c\ d)\top(a\ b)\top(a)\top(b\ c) = \\ &= (c\ d)\top(a\ b)\top(b\ c)\top(a).\end{aligned} \tag{22}$$

Доказательство Нужно установить, что выполняется сформулированное в теореме свойство единственности. Применим индукцию по длине перестановки; тогда достаточно доказать, что если ρ и σ—два различных цикла, не содержащие повторяющихся элементов, и

$$\rho\top\alpha = \sigma\top\beta,$$

то ρ и σ—непересекающиеся циклы, и

$$\alpha = \sigma\top\theta, \quad \beta = \rho\top\theta,$$

где θ—некоторая перестановка.

Пусть y—произвольный элемент цикла ρ, тогда у любого левого сомножителя в разложении $\sigma\top\beta$, содержащего этот элемент y, будет левый сомножитель ρ. Значит, если ρ и σ имеют общий элемент, то цикл σ должен быть кратен ρ; следовательно, $\sigma = \rho$ (так как они простые), что противоречит нашему предположению. Следовательно, цикл, содержащий y и не имеющий общих элементов с σ, должен быть левым сомножителем в разложении β. Применив законы сокращения (7), завершим доказательство. ∎

В качестве иллюстрации теоремы C рассмотрим перестановки мультимножества $M = \{A\cdot a, B\cdot b, C\cdot c\}$, состоящего из A элементов a, B элементов b и C элементов c. Пусть $N(A,B,C,m)$— число перестановок мультимножества M, двустрочное представление которых *не содержит* столбцов вида $\binom{a}{a}$, $\binom{b}{b}$, $\binom{c}{c}$ и содержит ровно m столбцов вида $\binom{a}{b}$. Отсюда следует, что имеется ровно $A-m$ столбцов вида $\binom{a}{c}$, $B-m$ столбцов вида $\binom{c}{b}$, $C-B+m$ столбцов вида $\binom{c}{a}$, $C-A+m$ столбцов вида $\binom{b}{c}$ и $A+B-C-m$ столбцов вида $\binom{b}{a}$; следовательно,

$$N(A,B,C,m) = \binom{A}{m}\binom{B}{C-A+m}\binom{C}{B-m}. \tag{23}$$

Теорема C предлагает другой способ для подсчета этих перестановок: коль скоро столбцы $\binom{a}{a}$, $\binom{b}{b}$, $\binom{c}{c}$ исключены, то в разложении перестановки единственно возможны такие простые множители:

$$(a\ b), (a\ c), (b\ c), (a\ b\ c), (a\ c\ b). \tag{24}$$

Каждая пара этих циклов имеет хотя бы одну общую букву, значит, разложение единственно. Если цикл $(a\ b\ c)$ встречается в разложении k раз, то из нашего предыдущего предположения следует, что $(a\ b)$ встречается $m-k$ раз, $(b\ c)$ встречается $C-A+m-k$ раз, $(a\ c)$ встречается $C-B+m-k$ раз и $(a\ c\ b)$ встречается $A+B-C-2m+k$ раз. Следовательно, $N(A,B,C,m)$ равно числу перестановок этих циклов (мультиномиальному коэффициенту), просуммированному по всем значениям k:

$$\begin{aligned}N(A,B,C,m) &= \sum_k \frac{(C+m-k)!}{(m-k)!(C-A+m-k)!(C-B+m-k)!k!(A+B-C-2m+k)!} = \\ &= \sum_k \binom{m}{k}\binom{A}{m}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A}.\end{aligned} \tag{25}$$

Сравнивая это с (23), обнаруживаем, что должно выполняться тождество

$$\sum_k \binom{m}{k}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A} = \binom{B}{C-A+m}\binom{C}{B-m}. \tag{26}$$

Оказывается, с этим тождеством мы встречались в упр. 1.2.6-31:

$$\sum_j \binom{M-R+S}{j}\binom{N+R-S}{N-j}\binom{R+j}{M+N} = \binom{R}{M}\binom{S}{N}, \tag{27}$$

где $M = A + B - C - m$, $N = C - B + m$, $R = B$, $S = C$, а $j = C - B + m - k$.

Аналогично можно подсчитать число перестановок мультимножества $\{ A \cdot a, B \cdot b, C \cdot c, D \cdot d \}$, если количество столбцов различных типов в них задано следующим образом:

$$\begin{array}{lcccccccc} \text{Столбец:} & \dfrac{a}{d} & \dfrac{a}{b} & \dfrac{b}{a} & \dfrac{b}{c} & \dfrac{c}{b} & \dfrac{c}{d} & \dfrac{d}{a} & \dfrac{d}{c} \\ \text{Количество:} & r & A-r & q & B-q & B-A+r & D-r & A-q & D-A+q \end{array} \tag{28}$$

(Здесь $A + C = B + D$.) Возможными циклами в разложении такой перестановки на простые множители будут

$$\begin{array}{lccccccc} \text{Цикл:} & (a\,b) & (b\,c) & (c\,d) & (d\,a) & (a\,b\,c\,d) & (d\,c\,b\,a) \\ \text{Количество:} & A-r-s & B-q-s & D-r-s & A-q-s & s & q-A+r+s \end{array} \tag{29}$$

при некотором s (см. упр. 12). В этом случае циклы $(a\,b)$ и $(c\,d)$ коммутируют, так же как и циклы $(b\,c)$ и $(d\,a)$, поэтому необходимо подсчитать число различных разложений на простые множители. Оказывается (см. упр. 10), всегда существует единственное разложение, такое, что цикл $(a\,b)$ никогда не следует непосредственно за $(c\,d)$, а $(b\,c)$ не встречается сразу после $(d\,a)$. Отсюда, пользуясь результатом упр. 13, получаем тождество

$$\sum_{s,t} \binom{B}{t} \binom{A-q-s}{A-r-s-t} \binom{B+D-r-s-t}{B-q-s} \times \frac{D!}{(D-r-s)!(A-q-s)!s!(q-A+r+s)!} = \\ = \binom{A}{r} \binom{B+D-A}{D-r} \binom{B}{q} \binom{D}{A-q}.$$

Вынося из обеих частей множитель $\binom{D}{A-q}$ и слегка упрощая факториалы, приходим к сложному на вид пятипараметрическому тождеству биномиальных коэффициентов:

$$\sum_{s,t} \binom{B}{t} \binom{A-r-t}{s} \binom{B+D-r-s-t}{D+q-r-t} \times \binom{D-A+q}{D-r-s} \binom{A-q}{r+t-q} = \binom{A}{r} \binom{B+D-A}{D-r} \binom{B}{q}. \tag{30}$$

Пользуясь тождеством (27), можно выполнить суммирование по s, а получившаяся сумма по t легко вычисляется. Таким образом, . после всей проделанной работы нам не посчастливилось обнаружить какое-либо тождество, которое мы бы еще не умели выводить. Но мы по крайней мере научились подсчитывать число перестановок определенного вида двумя различными способами, а эти методы подсчета—хорошая подготовка к решению задач, которые еще впереди.

Упражнения

1. [M05] *Да или нет?* Пусть M_1 и M_2—мультимножества. Если α—перестановка M_1, а β—перестановка M_2, то $\alpha \top \beta$—перестановка $M_1 \cup M_2$.
2. [10] Соединительное произведение перестановок $c\,a\,d\,a\,b$ и $b\,d\,d\,a\,d$ вычислено в (5); найдите соединительное произведение $b\,d\,d\,a\,d \top c\,a\,d\,a\,b$, которое получается, если сомножители поменять местами.
3. [M13] Верно ли утверждение, обратное (9)? Иначе говоря, если перестановки α и β коммутативны относительно операции соединительного произведения, то следует ли из этого, что они не содержат общих букв?
4. [M11] Каноническое разложение перестановки (12) в смысле теоремы A при $a < b < c < d$ задается формулой (17). Найдите соответствующее каноническое разложение в случае, когда $d < c < b < a$.
5. [M23] В условии (b) теоремы B требуется, чтобы $x < y$; что будет, если ослабить это требование, заменив его на $x \le y$?
6. [M15] Сколько существует цепочек, состоящих ровно из m букв a и n букв b, таких, что ровно k букв b стоят непосредственно перед буквами a и нет никаких других букв, кроме a и b?
7. [M21] Сколько цепочек из букв a, b, c, удовлетворяющих условиям (18), начинается с буквы a? с буквы b? с буквы c?

>8. [20] Найдите все разложения перестановки (12) на два множителя $\alpha \top \beta$.

9. [33] Напишите программы для ЭВМ, которые бы производили разложения заданной перестановки мультимножества, описанные в теоремах A и C.

>10. [M30] *Да или нет?* Согласно теореме C, разложение на простые множители не вполне однозначно, тем не менее можно следующим образом обеспечить единственность. "Существует линейное упорядочение $\prec$ на множестве простых перестановок, такое, что каждая перестановка мультимножества

имеет единственное разложение $\sigma_1 \top \sigma_2 \top \ldots \top \sigma_n$ на простые множители, удовлетворяющее условию $\sigma_i \preceq \sigma_{i+1}$, если σ_i коммутирует с σ_{i+1}, при $1 \le i < n$".

11. [M26] Пусть σ_1, σ_2, $\ldots$, σ_t—циклы без повторяющихся элементов. Определим частичное упорядочение $\prec$ на множестве t элементов $\{ x_1, \ldots, x_t \}$, полагая $x_i \prec x_j$, если $i < j$ и σ_i имеет по крайней мере одну общую букву с σ_j. Докажите следующую связь между теоремой C и понятием "топологической сортировки" (п. 2.2.3): *число различных разложений перестановки $\sigma_1 \top \sigma_2 \top \ldots \top \sigma_t$ на простые множители равно количеству способов топологически отсортировать данное частичное упорядочение.* (Например, в соответствии с (22) существует пять способов топологически отсортировать упорядочение $x_1 \prec x_3$, $x_2 \prec x_4$, $x_1 \prec x_4$.) Обратно, если на множестве из t элементов задано какое-то частичное упорядочение, то существует множество циклов

$$\{ \sigma_1, \sigma_2, \ldots, \sigma_t \},$$

которое определяет это частичное упорядочение указанным способом.

12. [M16] Покажите, что соотношения (29) являются следствиями предположений (28).
13. [M21] Докажите, что число перестановок мультимножества $\{ A \cdot a, B \cdot b, C \cdot c, D \cdot d, E \cdot e, F \cdot f \}$, не содержащих пар стоящих рядом букв $c\,a$ и $d\,b$, равно

$$\sum_t \binom{D}{A-t} \binom{A+B+E+F}{t} \binom{A+B+C+E+F-t}{B} \binom{C+D+E+F}{C, D, E, F}.$$

14. [M30] Один из способов определить перестановку π^{-1}, "обратную" перестановке π, который подсказан нам другими определениями этого пункта, это поменять местами строки двустрочного представления π и затем выполнить устойчивую сортировку столбцов, так чтобы элементы верхней строки расположились в неубывающем порядке. Например, если $a < b < c < d$, то из этого определения следует, что $c\,a\,b\,d\,d\,a\,b\,d\,a\,d^{-1} = a\,c\,d\,a\,d\,a\,b\,b\,d\,d$.

Исследуйте свойства этой операции обращения; имеется ли, например, какая-нибудь простая связь между ней и соединительным произведением? Можно ли подсчитать число перестановок, таких, что $\pi = \pi^{-1}$?

>15. [M25] Докажите, что перестановка $a_1 \ldots a_n$ мультимножества

$$\{ n_1 \cdot x_1, n_2 \cdot x_2, \ldots, n_m \cdot x_m \},$$

где $x_1 < x_2 < \ldots < x_n$ и $n_1 + n_2 + \cdots + n_m = n$, является циклом тогда и только тогда, когда направленный граф с вершинами $\{ x_1, x_2, \ldots, x_m \}$ и дугами из x_j в $a_{n_1+\cdots+n_j}$ содержит ровно один ориентированный цикл. В этом случае число способов представить перестановку в виде цикла равно длине этого ориентированного цикла. Например, направленным графом, соответствующим перестановке

$$\begin{pmatrix} a & a & a & b & b & c & c & c & d & d \\ d & c & b & a & c & a & a & b & d & c \end{pmatrix}, \text{ будет}$$

Picture: p. 48

а два способа представить перестановку в виде цикла—это $(b\,a\,d\,d\,c\,a\,c\,a\,b\,c)$ и $(c\,a\,d\,d\,c\,a\,c\,b\,a\,b)$.

16. [M35] В предыдущем пункте, формула (8), мы нашли производящую функцию для *инверсий* перестановок в частном случае, когда в перестановке участвуют элементы множества. Покажите, что в общем случае перестановок *мультимножества* производящая функция для инверсий равна "z-мультиномиальному коэффициенту"

$$\binom{n}{n_1, n_2, \ldots}_z = \frac{n!_z}{n_1!_z n_2!_z \ldots}, \qquad \text{где } m!_z = (1-z)(1-z^2)\ldots(1-z^m).$$

[Ср. с (3); z-биномиальные коэффициенты определены в формуле (1.2.6-37).]

17. [M24] Пользуясь производящей функцией, найденной в упр. 16, вычислите среднее значение и дисперсию для числа инверсий в случайной перестановке мультимножества.
18. [M30] (П. А. Мак-Магон.) *Индекс* перестановки $a_1\,a_2 \ldots a_n$ был определен в предыдущем пункте; мы доказали, что число перестановок данного множества, имеющих данный индекс k, равно числу перестановок, имеющих k инверсий. Верно ли это для перестановок заданного мультимножества?
19. [BM28] Определим *функцию Мебиуса* $\mu(\pi)$ перестановки π: она равна 0, если π содержит повторяющиеся элементы, и $(-1)^k$ в противном случае, если π—произведение k простых перестановок. (Ср. с определением обычной функции Мебиуса, упр. 4.5.2-10.) (a) Докажите, что если $\pi \ne \varepsilon$, то

$$\sum \mu(\lambda) = 0,$$

где сумма берется по всем перестановкам λ, являющимся левыми сомножителями в разложении π (т. е. $\lambda \top \rho = \pi$, где ρ—некоторая перестановка) (b) Докажите, что если $x_1 < x_2 < \ldots < x_m$ и $\pi = x_{i_1} x_{i_2} \ldots x_{i_n}$, где $1 \le i_l \le m$ при $1 \le j \le n$, то

$$\mu(\pi) = (-1)^m \varepsilon(i_1 i_2 \ldots i_n), \qquad \text{где } \varepsilon(i_1 i_2 \ldots i_n) = \operatorname{sign} \textstyle\prod_{1 \le j < k \le n} (i_k - i_j).$$

20. [BM33] (Д. Фоата.) Пусть (a_{ij})—произвольная матрица действительных чисел. Пользуясь обозначениями упр. 19 (b), определим $\nu(\pi) = a_{i_1 j_1} \ldots a_{i_n j_n}$, если двустрочное представление перестановки π таково:

$$\begin{pmatrix} x_{i_1} & x_{i_2} & \ldots & x_{i_n} \\ x_{j_1} & x_{j_2} & \ldots & x_{j_n} \end{pmatrix}.$$

Эта функция полезна при вычислении производящих функций для перестановок мультимножества, потому что $\sum \nu(\pi)$, где сумма берется по всем перестановкам π мультимножества

$$\{ n_1 \cdot x_1, \ldots, n_m \cdot x_m \},$$

будет производящей функцией для числа перестановок, удовлетворяющих некоторым ограничениям. Например, если положить $a_{ij} = z$ при $i = j$ и $a_{ij} = 1$ при $i \ne j$, то $\sum \nu(\pi)$—производящая функция для числа "неподвижных точек" (столбцов, в которых верхний и нижний элементы равны). Чтобы можно было исследовать $\sum \nu(\pi)$ для всех мультимножеств одновременно, рассмотрим функцию

$$G = \sum \pi \nu(\pi),$$

где сумма берется по всем π из множества $\{ x_1, \ldots, x_m \}^*$ всех перестановок мультимножеств, содержащих элементы $x_1, \ldots, x_m$, и посмотрим на коэффициенты при $x_1^{n_1} \ldots x_m^{nm}$ в G.

В этой формуле для G будем рассматривать π как произведение переменных x. Например, при $m = 2$ имеем

$$\begin{aligned} G &= 1 + x_1\nu(x_1) + x_2\nu(x_2) + x_1x_1\nu(x_1x_1) + x_1x_2\nu(x_1x_2) + x_2x_1\nu(x_2x_1) + x_2x_2\nu(x_2x_2) + \ldots = \\ &= 1 + x_1a_{11} + x_2a_{22} + x_1^2a_{11}^2 + x_1x_2a_{11}a_{22} + x_1x_2a_{21}a_{12} + x_2^2a_{22}^2 + \ldots . \end{aligned}$$

Таким образом, коэффициент при $x_1^{n_1} \ldots x_m^{n_m}$ в G равен сумме $\sum \nu(\pi)$ по всем перестановкам π мультимножества $\{ n_1 \cdot x_1, \ldots, n_m \cdot x_m \}$. Нетрудно видеть, что этот коэффициент равен также коэффициенту при $x_1^{n_1} \ldots x_m^{n_m}$ в выражении

$$(a_{11}x_1 + \cdots + a_{1m}x_m)^{n_1} (a_{21}x_1 + \cdots + a_{2m}x_m)^{n_2} \ldots (a_{m_1}x_1 + \cdots + a_{mm}x_m)^{n_m}.$$

Цель этого упражнения—доказать то, что П. А. Мак-Магон в своей книге Combinatory Analysis (1915), том 1, разд. 3, назвал "основной теоремой", а именно формулу

$$G = 1/D, \qquad \text{где } D = \det \begin{pmatrix} 1 - a_{11}x_1 & -a_{12}x_2 & \ldots & -a_{1m}x_m \\ -a_{21}x_1 & 1 - a_{22}x_2 & \ldots & -a_{2m}x_m \\ \vdots & & & \vdots \\ -a_{m1}x_1 & -a_{m2}x_2 & \ldots & 1 - a_{mm}x_m \end{pmatrix}.$$

Например, если $a_{ij} = 1$ при всех i, j, то эта формула дает

$$G = 1/(1 - (x_1 + x_2 + \cdots + x_m)),$$

и коэффициент при $x_1^{n_1} \ldots x_m^{n_m}$ оказывается равным $(n_1 + \cdots + n_m)!/n_1! \ldots n_m!$, как ему и положено.

Для доказательства основной теоремы Мак-Магона покажите, что (a) $\nu(\pi \top \rho) = \nu(\pi)\nu(\rho)$; (b) в обозначениях упр. 19 $D = \sum \pi \mu(\pi) \nu(\pi)$, где сумма берется по всем перестановкам π из множества

$$\{ x_1, \ldots, x_m \}^*,$$

и, наконец, (c) $D \cdot G = 1$.

5.1.3. *Отрезки

В п. 3.3.2 мы рассмотрели "неубывающие отрезки" как один из методов, позволяющих проверить случайность последовательности. Если поместить вертикальные черточки с обоих концов перестановки $a_1 a_2 \ldots a_n$,

а также между a_j и a_{j+1}, когда $a_j > a_{j+1}$, то *отрезками* будут называться сегменты, ограниченные парами черточек. Например, в перестановке

$$|3\ 5\ 7\ |\ 1\ 6\ 8\ 9\ |\ 4\ 2$$

—четыре отрезка. Мы нашли среднее число отрезков длины k в случайной перестановке множества $\{1, 2, \ldots, n\}$, а также ковариацию числа отрезков длины j и длины k. Отрезки важны при изучении алгоритмов сортировки, так как они представляют упорядоченные сегменты данных. Поэтому-то мы теперь вновь вернемся к вопросу об отрезках. Обозначим через

$$\left\langle {n \atop k} \right\rangle \tag{1}$$

число перестановок множества $\{1, 2, \ldots, n\}$, имеющих ровно k возрастающих отрезков. Такие числа $\left\langle {n \atop k} \right\rangle$ возникают и в других контекстах; их обычно называют *числами Эйлера,* потому что Эйлер обсудил их в своей знаменитой книге Institutiones calculi differentialis (St. Petersburg, 1755), 485–487 [Euler, *Opera Omnia,* (1) **10** (1913), 373–375]; их следует отличать от "эйлеровых чисел", о которых идет речь в упр. 5.1.4-22.

Из любой данной перестановки множества $\{1, 2, \ldots, n-1\}$ можно образовать n новых перестановок, вставляя элемент n во все возможные места. Если в исходной перестановке содержалось k отрезков, то ровно k новых перестановок будут иметь k отрезков; в остальных $n-k$ перестановках будет по $k+1$ отрезков, поскольку всякий раз, когда n вставляется не в конец уже существующего отрезка, число отрезков увеличивается на единицу. Например, среди шести перестановок, полученных из перестановки 3 1 2 4 5,

$$6\,3\,1\,2\,4\,5,\quad 3\,6\,1\,2\,4\,5,\quad 3\,1\,6\,2\,4\,5,$$
$$3\,1\,2\,6\,4\,5,\quad 3\,1\,2\,4\,6\,5,\quad 3\,1\,2\,4\,5\,6;$$

все, кроме второй и последней, содержат по три отрезка вместо исходных двух. Отсюда имеем рекуррентное соотношение

$$\left\langle {n \atop k} \right\rangle = k\left\langle {n-1 \atop k} \right\rangle + (n-k+1)\left\langle {n-1 \atop k-1} \right\rangle, \qquad \text{где } n \text{ целое, } n \geq 1;\ k \text{ целое.} \tag{2}$$

Условимся, что

$$\left\langle {0 \atop k} \right\rangle = \delta_{1k}, \tag{3}$$

т. е. будем считать, что в пустой перестановке содержится один отрезок. Читатель, возможно, найдет небезынтересным сравнить (2) с рекуррентным соотношением для чисел Стирлинга [формулы (1.2.6-42)]. В табл. 1 приведены числа Эйлера для малых n.

В табл. 1 можно заметить некоторые закономерности. По определению имеем

$$\left\langle {n \atop 0} \right\rangle + \left\langle {n \atop 1} \right\rangle + \cdots + \left\langle {n \atop n} \right\rangle = n!; \tag{4}$$
$$\left\langle {n \atop 0} \right\rangle = 0, \quad \left\langle {n \atop 1} \right\rangle = 1; \tag{5}$$
$$\left\langle {n \atop n} \right\rangle = 1 \qquad \text{при } n \geq 1. \tag{6}$$

Таблица 1

Числа Эйлера

n	$\left\langle {n \atop 0} \right\rangle$	$\left\langle {n \atop 1} \right\rangle$	$\left\langle {n \atop 2} \right\rangle$	$\left\langle {n \atop 3} \right\rangle$	$\left\langle {n \atop 4} \right\rangle$	$\left\langle {n \atop 5} \right\rangle$	$\left\langle {n \atop 6} \right\rangle$	$\left\langle {n \atop 7} \right\rangle$
0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0
3	0	1	4	1	0	0	0	0
4	0	1	11	11	1	0	0	0
5	0	1	26	66	26	1	0	0
6	0	1	57	302	302	57	1	0
7	0	1	120	1191	2416	1191	120	1

Выполняется также свойство симметрии

$$\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n+1-k} \right\rangle, \qquad n \geq 1, \tag{7}$$

которое вытекает из того факта, что каждой перестановке $a_1\,a_2\,\ldots\,a_n$, содержащей k отрезков, соответствует перестановка $a_n\,\ldots\,a_2\,a_1$, содержащая $n+1-k$ отрезков.

Другое важное свойство чисел Эйлера выражается формулой

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{m+k-1}{n} = m^n, \qquad n \geq 0, \tag{8}$$

которую можно доказать, используя понятие сортировки. Рассмотрим m^n последовательностей $a_1\,a_2\ldots a_n$, где $1 \leq a_i \leq m$. Любую такую последовательность можно устойчиво отсортировать таким образом, чтобы элементы расположились в неубывающем порядке:

$$a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_n}, \tag{9}$$

где $i_1\,i_2\,\ldots\,i_n$—однозначно определенная перестановка множества $\{1, 2, \ldots, n\}$, такая, что $i_j < i_{j+1}$, если $a_{i_j} = a_{i_{j+1}}$; другими словами, из $i_j > i_{j+1}$ следует $a_{i_j} < a_{i_{j+1}}$. Покажем, что если в перестановке $i_1\,i_2\,\ldots\,i_n$ содержится k отрезков, то число соответствующих ей последовательностей $a_1\,a_2\ldots a_n$ равно $\binom{m+n-k}{n}$; тем самым будет доказана формула (8), если заменить k на $n+1-k$.

Пусть, например, $n = 9$, $i_1\,i_2\ldots i_n = 3\,5\,7\,1\,6\,8\,9\,4\,2$ и требуется подсчитать число последовательностей $a_1 a_2 \ldots a_9$, таких, что

$$1 \leq a_3 \leq a_5 \leq a_7 < a_1 \leq a_6 \leq a_8 \leq a_9 < a_4 < a_2 \leq m; \tag{10}$$

оно равно числу последовательностей $b_1\,b_2\ldots b_9$, таких, что

$$1 \leq b_1 < b_2 < b_3 < b_4 < b_5 < b_6 < b_7 < b_8 < b_9 \leq m+5,$$

так как можно положить $b_1 = a_3$, $b_2 = a_5 + 1$, $b_3 = a_7 + 2$, $b_4 = a_1 + 2$, $b_5 = a_6 + 3$ и т. д. Число способов, которыми можно выбрать элементы b, равно просто-напросто числу способов выбрать 9 предметов из $m+5$, т. е. $\binom{m+5}{9}$; аналогичное доказательство годится для произвольных n и k и любой перестановки $i_1\,i_2\ldots i_n$ с k отрезками.

Так как в обеих частях равенства (8) стоят полиномы от m, то вместо m можно подставить любое действительное число, получив интересное выражение степеней через последовательные биномиальные коэффициенты:

$$x^n = \left\langle {n \atop 1} \right\rangle \binom{x}{n} + \left\langle {n \atop 2} \right\rangle \binom{x+1}{n} + \cdots + \left\langle {n \atop n} \right\rangle \binom{x+n-1}{n}, \qquad n \geq 1. \tag{11}$$

Например,

$$x^3 = \binom{x}{3} + 4\binom{x+1}{3} + \binom{x+2}{3}.$$

В основном благодаря именно этому свойству числа Эйлера весьма полезны при изучении дискретной математики. Положив в (11) $x = 1$, докажем еще раз, что

$$\left\langle {n \atop n} \right\rangle = 1,$$

поскольку биномиальные коэффициенты обращаются в 0 во всех слагаемых, кроме последнего. Положив $x = 2$, получим

$$\left\langle {n \atop n-1} \right\rangle = 2^n - n - 1, \qquad n \geq 1. \tag{12}$$

Подставив $x = 3, 4, \ldots$, убедимся, что все числа $\left\langle {n \atop k} \right\rangle$ полностью определяются соотношением (11), и придем к формуле, впервые найденной Эйлером:

$$\begin{aligned} \left\langle {n \atop k} \right\rangle &= k^n - (k-1)^n \binom{n+1}{1} + (k-2)^n \binom{n+1}{2} - \cdots + (-1)^k 0^n \binom{n+1}{k} = \\ &= \sum_{0 \leq j \leq k} (-1)^j (k-j)^n \binom{n+1}{j}, \qquad n \geq 0,\ k \geq 0. \end{aligned} \tag{13}$$

Изучим теперь производящую функцию для отрезков. Если положить

$$g_n(z) = \sum_k \left\langle {n \atop k} \right\rangle z^k/n!, \tag{14}$$

то коэффициент при z^k равен вероятности того, что случайная перестановка множества $\{1, 2, \ldots, n\}$ содержит ровно k отрезков. Поскольку k отрезков в перестановке столь же вероятны, как и $n+1-k$, то среднее число отрезков должно равняться $\frac{1}{2}(n+1)$, и, следовательно, $g_n'(1) = \frac{1}{2}(n+1)$. В упр. 2(b) показано, что имеет место простая формула для *всех* производных функции $g_n(z)$ в точке $z = 1$:

$$g_n^{(m)}(1) = \left\{ {n+1 \atop n+1-m} \right\} \Big/ \binom{n}{m}, \qquad n \geq m. \tag{15}$$

Так, в частности, дисперсия $g_n''(1) + g_n'(1) - g_n'(1)^2$ равна $(n+1)/12$ при $n \geq 2$, что указывает на довольно устойчивое распределение около среднего значения. (Мы нашли эту же величину в упр. 3.3.2-18; там она называлась $\operatorname{covar}(R_1', R_1')$.) Функция $g_n(z)$—полином, поэтому с помощью формулы (15) можно разложить ее в ряд Тейлора

$$\begin{aligned} g_n(z) &= \frac{1}{n!} \sum_{0 \leq k \leq n} (z-1)^{n-k} k! \left\{ {n+1 \atop k+1} \right\} = \\ &= \frac{1}{n!} \sum_{0 \leq k \leq n} z^{k+1} (1-z)^{n-k} k! \left\{ {n+1 \atop k+1} \right\}. \end{aligned} \tag{16}$$

Второе равенство следует из первого, так как по свойству симметрии (7)

$$g_n(z) = z^{n+1} g_n(1/z), \qquad n \geq 1. \tag{17}$$

Из рекуррентного соотношения для чисел Стирлинга

$$\left\{ {n+1 \atop k+1} \right\} = (k+1) \left\{ {n \atop k+1} \right\} + \left\{ {n \atop k} \right\}$$

получаются два чуть более простых представления:

$$\begin{aligned} g_n(z) &= \frac{1}{n!} \sum_{0 \leq k \leq n} z(z-1)^{n-k} k! \left\{ {n \atop k} \right\} = \\ &= \frac{1}{n!} \sum_{0 \leq k \leq n} z^k (1-z)^{n-k} k! \left\{ {n \atop k} \right\}. \end{aligned} \tag{18}$$

Производящая функция от двух переменных[7]

$$g(z, x) = \sum_{n \geq 0} g_n(z) x^n = \sum_{k,n \geq 0} \left\langle {n \atop k} \right\rangle z^k x^n \Big/ n! \tag{19}$$

равна, следовательно,

$$z \sum_{k,n \geq 0} \frac{((z-1)x)^n}{(z-1)^k} \left\{ {n \atop k} \right\} \frac{k!}{n!} = z \sum_{k \geq 0} \left(\frac{e^{(z-1)x} - 1}{z-1} \right)^k = \frac{z(1-z)}{e^{(z-1)x} - z}; \tag{20}$$

это еще одно соотношение, обсуждавшееся Эйлером.

Другие свойства чисел Эйлера можно найти в обзорной статье Л. Карлица [*Math. Magazine,* **33** (1959), 247–260]; см. также Дж. Риордан, Введение в комбинаторный анализ, ИЛ, М., 1963, стр. 50, 253, 254.

Рассмотрим теперь длину отрезков; какова в среднем длина отрезка? В п. 3.3.2 мы уже изучили математическое ожидание числа отрезков данной длины; средняя длина отрезка равна примерно 2 в согласии с тем фактом, что в случайной перестановке длины n содержится в среднем $\frac{1}{2}(n+1)$ отрезков.

[7] В оригинале "super generating function". Соответствующего термина в отечественной литературе не существует.—*Прим. ред.*

Применительно к алгоритмам сортировки полезна несколько отличная точка зрения; рассмотрим длину k-го слева отрезка перестановки при $k = 1, 2, \ldots$.

Какова, например, длина первого (крайнего слева) отрезка случайной перестановки $a_1\, a_2 \ldots a_n$? Его длина всегда ≥ 1; она ≥ 2 ровно в половине случаев (а именно, если $a_1 < a_2$). Его длина ≥ 3 ровно для $1/6$ случаев (если $a_1 < a_2 < a_3$). Вообще его длина $\geq m$ с вероятностью $q_m = 1/m!$ при $1 \leq m \leq n$. Следовательно, вероятность того, что длина этого отрезка равна в точности m, есть

$$\begin{aligned} p_m &= q_m - q_{m+1} = 1/m! - 1/(m+1)! \qquad \text{при } 1 \leq m < n; \\ p_n &= 1/n!. \end{aligned} \tag{21}$$

Средняя длина равна, таким образом,

$$\begin{aligned} p_1 + 2p_2 + \cdots + np_n = \\ &= (q_1 - q_2) + 2(q_2 - q_3) + \cdots + (n-1)(q_{n-1} - q_n) + nq_n = \\ &= q_1 + q_2 + \cdots + q_n = \\ &= \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!}. \end{aligned} \tag{22}$$

Предел при $n \to \infty$ равен $e - 1 = 1.71828\ldots$, а для конечных n это значение равно $e - 1 - \delta_n$, где δ_n довольно мало:

$$\delta_n = \frac{1}{(n+1)!}\left(1 + \frac{1}{n+2} + \frac{1}{(n+2)(n+3)} + \cdots\right) < \frac{e-1}{(n+1)!}.$$

Поэтому для практических целей удобно рассмотреть отрезки случайной *бесконечной* последовательности различных чисел

$$a_1, a_2, a_3, \ldots;$$

под "случайностью" последовательности мы здесь подразумеваем то, что все $n!$ возможных взаимных расположении первых n элементов последовательности равновероятны. Так что средняя длина первого отрезка случайной бесконечной последовательности равна $e - 1$.

Несколько усовершенствовав этот метод, можно установить среднюю длину k-го отрезка случайной последовательности. Пусть q_{km}—вероятность того, что общая длина первых k отрезков $\geq m$; тогда q_{km} равно величине $1/m!$, умноженной на число перестановок множества $\{1, 2, \ldots, m\}$, содержащих не более k отрезков:

$$q_{km} = \left(\left\langle {m \atop 1} \right\rangle + \cdots + \left\langle {m \atop k} \right\rangle\right) \Big/ m!. \tag{23}$$

Вероятность того, что общая длина первых k отрезков равна m, есть $q_{km} - q_{k(m+1)}$. Следовательно, обозначив через L_k среднюю длину k-го отрезка, находим, что

$$\begin{aligned} L_1 + \cdots + L_k &= (\text{средняя общая длина первых } k \text{ отрезков}) = \\ &= (q_{k1} - q_{k2}) = 2(q_{k2} - q_{k3}) + 3(q_{k3} - q_{k4}) + \cdots = \\ &= q_{k1} + q_{k2} + q_{k3} + \cdots. \end{aligned}$$

Вычитая $L_1 + \cdots + L_{k-1}$ и используя значения q_{km}, из (23) получаем нужную нам формулу:

$$L_k = \frac{1}{1!}\left\langle {1 \atop k} \right\rangle + \frac{1}{2!}\left\langle {2 \atop k} \right\rangle + \frac{1}{3!}\left\langle {3 \atop k} \right\rangle + \cdots = \sum_{m \geq 1} \left\langle {m \atop k} \right\rangle \frac{1}{m!}. \tag{24}$$

Поскольку $\left\langle {0 \atop k} \right\rangle = 0$ при $k \neq 1$, значение L_k оказывается равным коэффициенту при z^k в производящей функции $g(z, 1) - z$. (см. (19)); таким образом, имеем

$$L(z) = \sum_{k \geq 0} L_k z^k = \frac{z(1-z)}{e^{z-1} - z} - z. \tag{25}$$

Из формулы Эйлера (13) получим представление L_k в виде полинома от e:

$$\begin{aligned} L_k &= \sum_{m \geq 0} \sum_{0 \leq j \leq k} (-1)^{k-j} \binom{m+1}{k-j} \frac{j^m}{m!} = \\ &= \sum_{0 \leq j \leq k} (-1)^{k-j} \sum_{m \geq 0} \binom{m}{k-j} \frac{j^m}{m!} + \sum_{0 \leq j \leq k} (-1)^{k-j} \sum_{m \geq 0} \binom{m}{k-j-1} \frac{j^m}{m!} = \\ &= \sum_{0 \leq j \leq k} \frac{(-1)^{k-j} j^{k-j}}{(k-j)!} \sum_{n \geq 0} \frac{j^n}{n!} + \sum_{0 \leq j \leq k} \frac{(-1)^{k-j} j^{(k-j-1)}}{(k-j-1)!} \sum_{n \geq 0} \frac{j^n}{n!} = \\ &= k \sum_{0 \leq j \leq k} \frac{e^j (-1)^{k-j} j^{k-j-1}}{(k-j)!}. \end{aligned} \tag{26}$$

Эту формулу для L_k впервые получила Б. Дж. Гэсснер [*CACM*, **10** (1967), 89–93]. Имеем, в частности,

$$\begin{aligned} L_1 &= e-1 && \approx 1.71828\ldots; \\ L_2 &= e^2-2e && \approx 1.95249\ldots; \\ L_3 &= e^3-3e^2+\frac{3}{2}e && \approx 1.99579\ldots. \end{aligned}$$

Итак, следует ожидать, что второй отрезок будет длиннее первого, а третий отрезок будет в среднем еще длиннее! На первый взгляд это может показаться странным, но после минутного размышления становится ясно, что, поскольку первый элемент второго отрезка будет скорее всего малым числом (именно это служит причиной окончания первого отрезка), у второго отрезка больше шансов оказаться длиннее, чем у первого. Тенденция такова, что первый элемент третьего отрезка будет даже меньше, чем первый элемент второго отрезка.

Числа L_k важны в теории сортировки посредством выбора с замещением (п. 5.4.1), поэтому интересно подробно изучить их значения. В табл. 2, вычисленной Дж. У. Ренчем (мл.), приведены первые 18 значений L_k с точностью до 15 десятичных знаков. Рассуждения, приведенные в предыдущем абзаце, могут вызвать подозрение, что $L_{k+1} > L_k$; на самом же деле значения колеблются, то возрастая, то убывая. Заметим, что L_k быстро приближаются к предельному значению 2; весьма примечательно то, что эти нормированные полиномы от трансцендентного числа e так быстро сходятся к рациональному числу 2! Полиномы (26) представляют некоторый интерес и с точки зрения численного анализа, будучи прекрасным примером потери значащих цифр при вычитании почти равных чисел; используя 19-значную арифметику с плавающей точкой, Гэсснер пришла к неверному заключению о том, что $L_{12} > 2$, а Ренч отметил, что 42-значная арифметика

Таблица 2

Средние длины отрезков

k	L_k	k	L_k
1	1.71828 18284 59045+	10	2.00000 00012 05997+
2	1.95249 24420 12560−	11	2.00000 00001 93672+
3	1.99579 13690 84285−	12	1.99999 99999 99909+
4	2.00003 88504 76806−	13	1.99999 99999 97022−
5	2.00005 75785 89716+	14	1.99999 99999 99719+
6	2.00000 50727 55710−	15	2.00000 00000 00019+
7	1.99999 96401 44022+	16	2.00000 00000 00006+
8	1.99999 98889 04744+	17	2.00000 00000 00000+
9	1.99999 99948 43434−	18	2.00000 00000 00000−

с плавающей точкой дает L_{28} лишь с точностью до 29 значащих цифр.

Асимптотическое поведение L_k можно определить, исходя из простых положений теории функций комплексного переменного.

Picture: Рис. 3 Корни уравнения $e^{z-1} = z$. Пунктирная линия соответствует уравнению $e^{x-1}\cos y = x$, сплошная линия—уравнению $e^{x-1}\sin y = y$.

Знаменатель в (25) обращается в нуль лишь при $e^{z-1} = z$, т. е. (полагая $z = x + iy$) когда

$$e^{x-1}\cos y = x \text{ и } e^{x-1}\sin y = y. \tag{27}$$

На рис. 3, где нанесены оба графика этих уравнений, видно, что они пересекаются в точках $z = z_0$, z_1, $\bar{z}_1$, z_2, $\bar{z}_2$, $\ldots$; здесь $z_0 = 1$,

$$z_1 = (3.08884\,30156\,13044-) + (7.46148\,92856\,54255-)i \tag{28}$$

и при больших k мнимая часть $\Im(z_{k+1})$ равна приблизительно $\Im(z_k) + 2\pi$. Так как

$$\lim_{z\to z_k}\left(\frac{1-z}{e^{z-1}-z}\right)(z-z_k) = -1 \qquad \text{при } k > 0$$

и этот предел равен -2 при $k = 0$, то функция

$$R_m(z) = L(z) + \frac{2}{z-z_0} + \frac{z_1}{z-z_1} + \frac{\bar{z}_1}{z-\bar{z}_1} + \frac{z_2}{z-z_2} + \frac{\bar{z}_2}{z-\bar{z}_2} + \cdots + \frac{z_m}{z-z_m} + \frac{\bar{z}_m}{z-\bar{z}_m}$$

не имеет особенностей в комплексной плоскости при $|z| < |z_{m+1}|$. Значит, $R_m(z)$ можно разложить в степенной ряд $\sum_k \rho_k z^k$, который сходится абсолютно при $|z| < |z_{m+1}|$; отсюда следует, что $\rho_k M^k \to 0$ при $k \to \infty$, где $M = |z_{m+1}| - \varepsilon$. Коэффициентами для $L(z)$ служат коэффициенты разложения функции

$$\frac{2}{1-z} + \frac{1}{1-z/z_1} + \frac{1}{z/\bar{z}_1} + \cdots + \frac{1}{z-z/z_m} + \frac{1}{z-z/\bar{z}_m} + R_m z,$$

а именно

$$L_n = 2 + 2r_1^{-n}\cos n\theta_1 + 2r_2^{-n}\cos n\theta_2 + \cdots + 2r_m^{-n}\cos n\theta_m + O(r_{m+1}^{-n}), \tag{29}$$

если положить

$$z_k = r_k e^{i\theta_k}. \tag{30}$$

Отсюда можно проследить асимптотическое поведение L_n. Имеем

$$\begin{aligned} r_1 &= 8.07556\,64528\,89526-, \\ \theta_1 &= 1.17830\,3978\,4\,74668+; \\ r_2 = 14.35457-, &\qquad \theta_2 = 1.31269-; \\ r_3 = 20.62073+, &\qquad \theta_3 = 1.37428-; \\ r_4 = 26.88795+, &\qquad \theta_4 = 1.41050-; \end{aligned} \tag{31}$$

таким образом, главный вклад в $L_n - 2$ дают r_1 и θ_1, и ряд (29) сходится очень быстро. Приведенные здесь значения r_1 и θ_1 найдены Дж. У. Ренчем (мл.) Дальнейший анализ [W. W. Hooker, *CACM*, **12** (1969), 411–413] показывает, что $R_m(z) \to -z$ при $m \to \infty$; следовательно, ряд $2\sum_{k\ge 0} z_k^{-n}\cos n\theta_k$ действительно *сходится* к L_n при $n > 1$.

Можно провести более тщательное исследование вероятностей, чтобы полностью определить распределение вероятностей для длины k-го отрезка и для общей длины первых k отрезков (см. упр. 9, 10, 11), Оказывается сумма $L_1 + \cdots + L_k$ асимптотически приближается к $2k - 1/3$.

В заключение этого пункта рассмотрим свойства отрезков в случае, когда в перестановке допускаются одинаковые элементы. Бесчисленные пасьянсы, которым посвящал свои досуги знаменитый американский астроном 19-го века Саймон Ньюкомб, имеют непосредственное отношение к интересующему нас вопросу. Он брал колоду карт и складывал их в одну стопку до тех пор, пока они шли в неубывающем порядке по старшинству; как только следующая карта оказывалась младше предыдущей, он начинал новую стопку. Он хотел найти вероятность того, что в результате вся колода окажется разложенной в заданное количество стопок.

Задача Саймона Ньюкомба состоит, следовательно, в нахождении распределения вероятностей для отрезков случайной перестановки мультимножества. В общем случае ответ довольно сложен (см. упр. 12), хотя мы уже видели, как решать задачу в частном случае, когда все карты различны по старшинству. Мы удовлетворимся здесь выводом формулы для *среднего* числа стопок в этом пасьянсе.

Пусть имеется m различных типов карт и каждая встречается ровно p раз. Например, в обычной колоде для бриджа $m = 13$, а $p = 4$, если пренебрегать различием масти. Замечательную симметрию обнаружил в этом случае П. А. Мак-Магон [Combinatory Analysis (Cambridge, 1915), том 1, стр. 212–213]: число перестановок с $k+1$ отрезками равно числу перестановок с $mp-p-k+1$ отрезками. Это соотношение легко проверить при $p = 1$ (формула (7)), однако при $p > 1$ оно кажется довольно неожиданным.

Можно доказать это свойство симметрии, установив взаимно однозначное соответствие между перестановками, такое, что каждой перестановке с $k+1$ отрезками соответствует другая, с $mp-p-k+1$ отрезками. Мы настойчиво рекомендуем читателю самому попробовать найти такое соответствие, прежде чем двигаться дальше.

Какого-нибудь очень простого соответствия на ум не приходит; доказательство Мак-Магона основано на производящих функциях, а не на комбинаторном построении. Однако установленное Фоатой соответствие (теорема 5.1.2B) позволяет упростить задачу, так как там утверждается существование взаимно однозначного соответствия между перестановками с $k+1$ отрезками и перестановками, в двустрочном представлении которых содержится ровно k столбцов $\binom{y}{x}$, таких, что $x < y$.

Пусть дано мультимножество $\{\, p\cdot 1, p\cdot 2, \ldots, p\cdot m \,\}$; рассмотрим перестановку с двустрочным обозначением

$$\begin{pmatrix} 1 & \ldots & 1 & 2 & \ldots & 2 & \ldots & m & \ldots & m \\ x_{11} & \ldots & x_{1p} & x_{21} & \ldots & x_{2p} & \ldots & x_{m1} & \ldots & x_{mp} \end{pmatrix}. \tag{32}$$

Сопоставим с ней другую перестановку того же мультимножества:

$$\begin{pmatrix} 1 & \ldots & 1 & 2 & \ldots & 2 & \ldots & m & \ldots & m \\ x'_{11} & \ldots & x'_{1p} & x'_{m1} & \ldots & x'_{mp} & \ldots & x'_{21} & \ldots & x'_{2p} \end{pmatrix}, \tag{33}$$

где $x' = m + 1 - x$. Если перестановка (32) содержит k столбцов вида $\binom{y}{x}$, таких, что $x < y$, то (33) содержит $(m-1)p - k$ таких столбцов, потому что необходимо рассмотреть лишь случай $y > 1$, а неравенство $x < y$ эквивалентно $x' \geq m + 2 - y$. Поскольку (32) соответствует перестановке с $k+1$ отрезками, а (33)—перестановке с $mp - p - k + 1$ отрезками и преобразование, переводящее (32) в (33), обратимо (оно же переводит (33) в (32)), то тем самым доказано условие симметрии Мак-Магона. Пример этого построения содержится в упр. 14.

В силу свойства симметрии среднее число отрезков в случайной перестановке должно равняться $\frac{1}{2}((k+1)+(mp-p-k+1)) = 1+\frac{1}{2}p(m-1)$. Например, для стандартной колоды среднее число стопок в пасьянсе Ньюкомба равно 25 (так что раскладывание этого пасьянса вряд ли покажется столь уж увлекательным занятием).

На самом деле, используя весьма простое рассуждение, можно определить среднее число отрезков в общем случае для *любого* данного мультимножества $\{ n_1 \cdot x_1, n_2 \cdot x_2, \ \ldots, n_m \cdot x_m \}$, где все x различны. Пусть $n = n_1 + n_2 + \cdots + n_m$, и представим себе, что все перестановки $a_1\, a_2 \ldots a_n$ этого мультимножества записаны в явном виде; посмотрим, как часто a_i оказывается больше a_{i+1} при каждом фиксированном значении i, $1 \leq i < n$. Число случаев, когда $a_i > a_{i+1}$, равно в точности половине числа случаев, когда $a_i \neq a_{i+1}$, и нетрудно видеть, что $a_i = a_{i+1} = x_j$ ровно в $Nn_j(n_j-1)/n(n-1)$ случаях, где N—общее число перестановок. Следовательно, $a_i = a_{i+1}$ ровно в

$$\frac{N}{n(n-1)}(n_1(n_1-1) + \cdots + n_m(n_m-1)) = \frac{N}{n(n-1)}(n_1^2 + \cdots + n_m^2 - n)$$

случаях, а $a_i > a_{i+1}$ ровно в

$$\frac{N}{2n(n-1)}(n^2 - (n_1^2 + \cdots + n_m^2))$$

случаях. Суммируя по i и прибавляя N, потому что в каждой перестановке один отрезок кончается элементом a_n, получим общее число отрезков во всех N перестановках:

$$N\left(\frac{n}{2} - \frac{1}{2n}(n_1^2 + \cdots + n_m^2) + 1\right). \tag{34}$$

Поделив на N, получим искомое среднее число отрезков.

Так как отрезки важны при изучении "порядковых статистик", имеется весьма обширная литература, посвященная им, в том числе и некоторым другим типам отрезков, не рассмотренным здесь. Дополнительную информацию можно найти в книге Ф. Н. Дэвид и Д. Э. Бартона Combinatorial Chance (London: Griffin, 1962), гл. 10, и в обзорной статье Д. Э. Бартона и К. Л. Мэллоуза [*Annals of Math. Statistics,* **36** (1965), 236–260]. Дальнейшие связи между числами Эйлера и перестановками рассматриваются в работе Д. Фоаты и М. П. Шюценберже Théorie Géométrique des Polynômes Eulériens (Lecture Notes in Math., 138 (Berlin: Springer, 1970), 94 стр.).

Упражнения

1. [M26] Выведите формулу Эйлера (13).

>2. [M22] (a) Попытайтесь дальше развить идею, использованную в тексте при доказательстве тождества (8): рассмотрите последовательности $a_1\, a_2 \ldots a_n$, содержащие ровно q различных элементов, и докажите, что

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k-1}{n-q} = \left\{ {n \atop q} \right\} q!.$$

(b) Используя это тождество, докажите, что

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k}{m} = \left\{ {n+1 \atop n+1-m} \right\} (n-m)! \qquad \text{при } n \geq m.$$

3. [BM25] Вычислите сумму $\sum_k \left\langle {n \atop k} \right\rangle (-1)^k$.
4. [M21] Чему равна сумма

$$\sum_k (-1)^k \left\{ {n \atop k} \right\} k! \binom{n-k}{m}?$$

5. [M20] Найдите значение $\left\langle {p \atop k} \right\rangle \bmod p$, если p—простое число.

>6. [M21] Мистер Тупица заметил, что из формул (4) и (13) можно получить

$$n! = \sum_{k\ge 0} \left\langle {n \atop k} \right\rangle = \sum_{k\ge 0}\sum_{j\ge 0} (-1)^{k-j} \binom{n+1}{k-j} j^n.$$

Произведя суммирование сначала по k, затем по j, он обнаружил, что $\sum_{k\ge 0}(-1)^{k-j}\binom{n+1}{k-j} = 0$ при всех $j \ge 0$, отсюда $n! = 0$ при любом $n \ge 0$. Не допустил ли он какой-нибудь ошибки?

7. [BM40] Является ли распределение вероятностей для отрезков, задаваемое формулой (14), асимптотически нормальным? (Ср. с упр. 1.2.10-13.)

8. [M24] (П. А. Мак-Магон) Покажите, что вероятность того, что длина первого отрезка достаточно длинной перестановки есть l_1, длина второго есть l_2, ..., а длина k-го отрезка $\ge l_k$, равна

$$\det \begin{pmatrix} 1/l_1! & 1/(l_1+l_2)! & 1/(l_1+l_2+l_3)! & \dots & 1/(l_1+l_2+l_3+\cdots+l_k)! \\ 1 & 1/l_2! & 1/(l_2+l_3)! & \dots & 1/(l_2+l_3+\cdots+l_k)! \\ 0 & 1 & 1/l_3! & \dots & 1/(l_3+\cdots+l_k)! \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & 1/l_k! \end{pmatrix}.$$

9. [M30] Пусть $h_k(z) = \sum p_{km} z^m$, где p_{km}—вероятность того, что общая длина первых k отрезков (бесконечной) случайной последовательности равна m. Найдите "простые" выражения для $h_1(z)$, $h_2(z)$ и для производящих функций $h(z,x) = \sum_k h_k(z)x^k$ от двух переменных.

10. [BM30] Определите асимптотическое поведение среднего значения и дисперсии распределения $h_k(z)$ из предыдущего упражнения при больших k.

11. [M40] Пусть $H_k(z) = \sum p_{km} z^m$, где p_{km}—вероятность того, что длина k-го отрезка в случайной (бесконечной) последовательности равна m. Выразите $H_1(z)$, $H_2(z)$ и производящую функцию $H(z,x) = \sum_k H_k(z)x^k$ от двух переменных через известные функции.

12. [M33] (П.А. Мак-Магон.) Обобщите формулу (13) на случай перестановок мультимножества, доказав, что число перестановок мультимножества $\{ n_1\cdot 1, n_2\cdot 2, \dots, n_m\cdot m \}$, имеющих ровно k отрезков, равно

$$\sum_{0\le j\le k} (-1)^i \binom{n+1}{j} \binom{n_1-1+k-j}{n_1} \binom{n_2-1+k-j}{n_2} \cdots \binom{n_m-1+k-j}{n_m},$$

где $n = n_1 + n_2 + \cdots + n_m$.

13. [05] Каким будет среднее число стопок в пасьянсе Ньюкомба, если пользоваться обычной колодой для бриджа (из 52 карт), игнорируя старшинство карт, но считая, что

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit ?$$

14. [M17] Перестановка 3 1 1 1 2 3 1 4 2 3 3 4 2 2 4 4 содержит 5 отрезков; найдите с помощью приведенного в тексте построения для условия симметрии Мак-Магона соответствующую перестановку с 9-ю отрезками.

>15. [M21] *(Перемежающиеся отрезки.)* В классической литературе 19-го века по комбинаторному анализу не изучался вопрос об отрезках в перестановках, которые рассматриваем мы, но было несколько статей, посвященных попеременно возрастающим и убывающим "отрезкам". Так, считалось, что перестановка 5 3 2 4 7 6 1 8 содержит 4 отрезка

$$5\,3\,2, \quad 2\,4\,7, \quad 7\,6\,1, \quad 1\,8.$$

(Первый отрезок будет возрастающим или убывающим в зависимости от того, $a_1 < a_2$ или $a_1 > a_2$; таким образом, перестановки $a_1\,a_2 \dots a_n$, $a_n \dots a_2\,a_1$ и $(n+1-a_1)\,(n+1-a_2)\dots(n+1-a_n)$ все содержат одинаковое число перемежающихся отрезков.) Максимальное число отрезков этого типа в перестановке n элементов равно $n-1$.

Найдите среднее число перемежающихся отрезков в случайной перестановке множества $\{1, 2, \dots, n\}$. [*Указание:* разберите вывод формулы (34).]

16. [M30] Продолжим предыдущее упражнение. Пусть $\left\langle\left\langle {n \atop k} \right\rangle\right\rangle$—число перестановок множества $\{1, 2, \dots, n\}$, которые имеют ровно k перемежающихся отрезков. Найдите рекуррентное соотношение, с помощью которого можно вычислить таблицу значений $\left\langle\left\langle {n \atop k} \right\rangle\right\rangle$; найдите также соответствующее рекуррентное соотношение для производящей функции $G_n(z) = \sum_k \left\langle\left\langle {n \atop k} \right\rangle\right\rangle z^k/n!$. Используя это последнее рекуррентное соотношение, найдите простую формулу для *дисперсии* числа перемежающихся отрезков в случайной перестановке множества $\{1, 2, \dots, n\}$.

17. [M25] Существует всего 2^n последовательностей $a_1\,a_2\,a_n$, где каждый элемент a_j—либо 0, либо 1. Сколько среди них последовательностей, содержащих ровно k отрезков (т. е. содержащих ровно $k-1$ элементов a_j, таких, что $a_j > a_{j+1}$)?
18. [M27] Существует всего $n!$ последовательностей $a_1\,a_2\,\ldots\,a_n$, где каждый элемент a_j—целое число, лежащее в диапазоне $0 \le a_j \le n-j$; сколько среди них последовательностей, содержащих ровно k отрезков (т. е. содержащих ровно $k-1$ элементов a_j, таких, что $a_j > a_{j+1}$)?

>19. [M26] (Дж. Риордан.) (a) Сколькими способами можно расположить n неатакующих ладей (т. е. никакие две не должны находиться на одной вертикали

Picture: Рис. 4. Неатакующие ладьи на шахматной доске при заданном числе ладей ниже главной диагонали.

или горизонтали) на шахматной доске размера $n \times n$ так, чтобы ровно k из них находились на заданной стороне от главной диагонали? (b) Сколькими способами можно расположить k неатакующих ладей на заданной стороне от главной диагонали шахматной доски размера $n \times n$?

Например, на рис. 4 показан один из 15619 способов расположить восемь неатакующих ладей на обычной шахматной доске с тремя ладьями на незаштрихованном участке ниже главной диагонали, а также один из 1050 способов расположить три неатакующие ладьи на треугольной доске.

>20. [M21] Говорят, что перестановка требует k *чтений,* если ее нужно просмотреть k раз, слева направо, чтобы прочитать все элементы в неубывающем порядке. Например, перестановка

4 9 1 8 2 5 3 6 7

требует четырех чтений: при первом чтении получаем 1, 2, 3; при втором—4, 5, 6, 7; затем 8; затем 9. Найдите связь между отрезками и чтениями.

21. [M22] Если перестановка $a_1\,a_2\,\ldots\,a_n$ множества $\{1, 2, \ldots, n\}$ содержит k отрезков и требует j чтений в смысле упр. 20, то что можно сказать о перестановке $a_n\,\ldots\,a_2\,a_1$?
22. [M26] (Л. Карлиц, Д. П. Розель и Р. А. Скоувилл.) Покажите, что не существует перестановки множества $\{1, 2, \ldots, n\}$ с $n+1-r$ отрезками, требующей s чтений, если $rs < n$, однако такая перестановка существует, если $n \ge n+1-r \ge s \ge 1$, $rs \ge n$.
23. [BM42] (Вальтер Вейссблюм.) "Удлиненные отрезки" перестановки $a_1\,a_2\,\ldots\,a_n$ получаются, если вставлять вертикальные черточки в тех местах, где нарушается установившаяся монотонность; удлиненные отрезки бывают как возрастающими, так и убывающими в зависимости от того, в каком порядке расположены первые два элемента, так что длина каждого удлиненного отрезка (кроме, возможно, последнего) ≥ 2. Например, перестановка 7 5|6 2|3 8 9|1 4 содержит четыре удлиненных отрезка. Найдите средние длины первых двух удлиненных отрезков бесконечной перестановки и докажите, что в пределе длина удлиненного отрезка равна

$$\left(1 + \operatorname{ctg}\frac{1}{2}\right) \Big/ \left(3 - \operatorname{ctg}\frac{1}{2}\right) \approx 2.4202.$$

24. [M30] Выразите в виде функции от p среднее число отрезков в последовательностях, полученных методом, который описан в упр. 5.1.1-18.

5.1.4. *Табло и инволюции

В заключение нашего обзора комбинаторных свойств перестановок обсудим некоторые замечательные отношения, связывающие перестановки с массивами целых чисел, называемыми табло. *Табло Янга формы* $(n_1, n_2, \ldots, n_m)$, где $n_1 \ge n_2 \ge \ldots \ge n_m \ge 0$, это расположение $n_1 + n_2 + \cdots + n_m$ различных целых чисел в массив строк, выровненных по левому краю, где в i-й строке содержится n_i элементов; при этом в каждой строке элементы возрастают слева направо, а элементы каждого столбца возрастают сверху вниз. Например,

1	2	5	9	10	15
3	6	7	13		
4	8	12	14		
11					

(1)

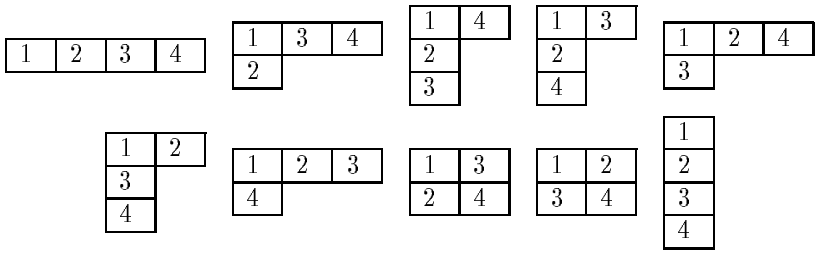
—табло Янга формы $(6, 4, 4, 1)$. Такие расположения ввел Альфред Янг в 1900 г. в качестве вспомогательного средства при изучении матричного представления перестановок. [См., например, D. E. Rutherford, Substitutional Analysis (New York: Hafiier, 1968)]. Для краткости мы будем вместо "табло Янга" говорить просто "табло".

Инволюция—это перестановка, обратная самой себе. Например, существует 10 инволюций множества $\{1,2,3,4\}$:

$$\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix}\begin{pmatrix}1&2&3&4\\2&1&3&4\end{pmatrix}\begin{pmatrix}1&2&3&4\\3&2&1&4\end{pmatrix}\begin{pmatrix}1&2&3&4\\4&2&3&1\end{pmatrix}\begin{pmatrix}1&2&3&4\\1&3&2&4\end{pmatrix}$$
$$\begin{pmatrix}1&2&3&4\\1&4&3&2\end{pmatrix}\begin{pmatrix}1&2&3&4\\1&2&4&3\end{pmatrix}\begin{pmatrix}1&2&3&4\\2&1&4&3\end{pmatrix}\begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix}\begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix} \tag{2}$$

Термин "инволюция" первоначально использовался в классических задачах геометрии; инволюции в общем смысле, рассматриваемые здесь, были впервые изучены Х. А. Роте, когда он ввел понятие обратной перестановки (см. п. 5.1.1).

Может показаться странным, что мы рассматриваем табло и инволюции вместе, но существует удивительная связь между этими понятиями, не имеющими, казалось бы, друг к другу никакого отношения: *число инволюций множества* $\{1,2,\ \ldots,n\}$ *равно числу табло, которые можно сформировать из элементов* $\{1,2,\ \ldots,n\}$. Например, из $\{1,2,3,4\}$ можно сформировать ровно 10 табло



(3)

что соответствует 10 инволюциям (2).

Эта связь между инволюциями и табло отнюдь не очевидна, и, по-видимому, не существует простого способа ее доказать. Доказательство, которое мы обсудим, включает интересный алгоритм построения табло, обладающий и некоторыми другими неожиданными свойствами; он основан на особой процедуре вставки в табло новых элементов.

Предположим, например, что нужно вставить элемент 8 в табло

1	3	5	9	12	16
2	6	10	15		
4	13	14			
11					
17					

(4)

Метод, которым мы будем пользоваться, состоит в том, чтобы сначала поместить 8 в 1-ю строку, в клетку, ранее занимаемую 9, поскольку 9—наименьший из элементов этой строки, больших 8. Элемент 9 "вытесняется" вниз, в строку 2, где он замещает 10. Затем 10 "вытесняет" 13 из 3-й строки в 4-ю и, поскольку в 4-й строке нет элемента больше 13, процесс завершается добавлением 13 в правый конец строки 4. Наше табло преобразовалось в

1	3	5	8	12	16
2	6	9	15		
4	10	14			
11	13				
17					

(5)

Точное описание этого процесса с доказательством того, что он всегда сохраняет свойства табло, содержится в алгоритме I.

Алгоритм I. *(Вставка в табло.)* Пусть $P=(P_{ij})$—табло целых положительных чисел, а x—целое положительное число, не содержащееся в P. Этот алгоритм преобразует P в другое табло, содержащее x наряду с исходными элементами P. Новое табло имеет ту же форму, что и старое, с той лишь разницей, что на пересечении строки s и столбца t появился новый элемент, где s и t—величины, определяемые алгоритмом.

(В этом алгоритме замечания в круглых скобках предназначены для доказательства его правильности, поскольку по индукции легко проверить, что эти замечания справедливы и что массив P продолжает оставаться табло на протяжении всего процесса. Для удобства будем предполагать, что табло ограничено

нулями сверху и слева и символами ∞ справа и снизу, так что элемент P_{ij} определен при всех $i, j \geq 0$. Если ввести отношение

$$a \lesssim b \qquad \text{тогда и только тогда, когда } a < b \text{ или } a = b = \infty, \tag{6}$$

то неравенства для табло можно выразить в удобной форме:

$$\begin{gathered} P_{ij} = 0, \qquad \text{если } i = 0 \text{ или } j = 0; \\ P_{ij} \lesssim P_{i(j+1)} \text{ и } P_{ij} \lesssim P_{(i+1)j} \qquad \text{при всех } i, j \geq 0. \end{gathered} \tag{7}$$

Утверждение "$x \notin P$" означает, что либо $x = \infty$, либо x не равно P_{ij} ни при каких $i, j \geq 0$.)

I1 [Ввести x.] Установить $i \leftarrow 1$, $x_1 \leftarrow x$, а j установить равным наименьшему значению, такому, что $P_{1j} = \infty$.

I2 [Найти x_{i+1}.] (В этот момент $P_{(i-1)j} < x_i < P_{ij}$ и $x_i \notin P$.) Если $x_i < P_{i(j-1)}$, то уменьшить j на 1 и повторить этот шаг. В противном случае установить $x_{i+1} \leftarrow P_{ij}$ и $r_i \leftarrow j$.

I3 [Заменить на x_i.] (Теперь $P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$ и $r_i = j$.) Установить $P_{ij} \leftarrow x_i$.

I4 [$x_{i+1} = \infty$?] (Теперь $P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j-1)}$, $P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$, $r_i = j$ и $x_{i+1} \notin P$.) Если $x_{i+1} \neq \infty$, то увеличить i на 1 и вернуться к шагу **I2**.

I5 [Определить s, t.] Установить $s \leftarrow i$, $t \leftarrow j$ и закончить работу алгоритма. (К этому моменту выполняются условия

$$P_{st} \neq \infty, P_{(s+1)t} = P_{s(t+1)} = \infty.) \tag{8}$$

∎

Заметим, что этот алгоритм определяет "последовательность вытеснении"

$$x = x_1 < x_2 < \ldots < x_s < x_{s+1} = \infty, \tag{9}$$

а также вспомогательную последовательность индексов столбцов

$$r_1 \geq r_2 \geq \ldots \geq r_s = t; \tag{10}$$

при этом элемент P_{ir_i} меняет свое значение с x_{i+1} на x_i, $1 \leq i \leq s$. Например, когда в табло (4) вставлялся элемент 8, последовательностью вытеснений была 8, 9, 10, 13, ∞, а вспомогательной последовательностью — 4, 3, 2, 2. Мы могли бы переформулировать алгоритм так, чтобы он расходовал меньше временной памяти (необходимо хранить только текущие значения j, x_i, x_{i+1}); последовательности (9) и (10) были введены с той лишь целью, чтобы можно было доказать некоторые интересные факты, касающиеся этого алгоритма.

Основное свойство алгоритма I, которым мы не преминем воспользоваться, состоит в том, что алгоритм можно "прокрутить" в обратном направлении: по заданным s и t, определенным в шаге I5, можно преобразовать P к исходному виду, найдя и удалив элемент x, который был вставлен. Рассмотрим, например, (5), и пусть нам известно, что элемент 13 занимает позицию, которая раньше была пуста. Тогда элемент 13, должно быть, был вытеснен вниз из 3-й строки числом 10, потому что 10—наибольший элемент в этой строке, меньший 13; аналогично, элемент 10, по-видимому, был вытеснен из 2-й строки числом 9, которое в свою очередь было вытеснено из 1-й строки числом 8. Таким образом, от (5) можно вернуться обратно к (4). В следующем алгоритме подробно описан этот процесс.

Алгоритм D. *(Удаление из табло.)* По заданному табло P и целым положительным числам s и t, удовлетворяющим условиям (8), этот алгоритм преобразует P в другое табло почти такой же формы, но с ∞ на пересечении столбца t и строки s. Определенный алгоритмом элемент x удаляется из табло P.

(Как и в алгоритме I, пояснения в круглых скобках включены для облегчения доказательства того, что массив P продолжает оставаться табло на протяжении всего процесса.)

D1 [Ввести s, t.] Установить $j \leftarrow t$, $i \leftarrow s$, $x_{s+1} \leftarrow \infty$.

D2 [Найти x_i.] (В этот момент $P_{ij} < x_{i+1} \lesssim P_{(i+1)j}$ и $x_{i+1} \notin P$.) Если $P_{i(j+1)} < x_{i+1}$, то увеличить j на 1 и повторить этот шаг. В противном случае установить $x_i \leftarrow P_{ij}$ и $r_i \leftarrow j$.

D3 [Заменить на x_{i+1}.] (Теперь $P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$ и $r_i = j$.) Установить $P_{ij} \leftarrow x_{i+1}$.

D4 [$i = 1$?] (Теперь $P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}$, $P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$ и $r_i = j$.) Если $i > 1$, то уменьшить t на 1 и вернуться к шагу **D2**.

D5 [Определить x.] Установить $x \leftarrow x_1$ и закончить работу алгоритма. (Теперь $0 < x < \infty$.) ∎

Пояснения к алгоритмам I и D, заключенные в круглые скобки, не только полезны для доказательства того факта, что эти алгоритмы сохраняют структуру табло, но они позволяют убедиться, что *алгоритмы I и D взаимно обратны.* Если сначала выполнить алгоритм I с данным табло P и некоторым целым положительным числом $x \notin P$, то он вставит x в P и определит целые положительные числа s, t, удовлетворяющие условиям (8). Алгоритм D, примененный к полученному результату, восстановит значения x и первоначальный вид P. Обратно, если сначала выполнить алгоритм D с данным табло P и некоторыми целыми положительными числами s, t, удовлетворяющими условиям (8), то он модифицирует P, удалив некоторое целое положительное число x. Алгоритм I, примененный к полученному результату, восстановит значения s, t и первоначальный вид P. Причина заключается в том, что содержание пояснений к шагам I3 и D4 совпадает так же, как и к шагам I4 и D3; они однозначно определяют значение j. Следовательно, вспомогательные последовательности (9), (10) одинаковы, в обоих случаях.

Теперь мы подготовлены к доказательству основного свойства табло.

Теорема A. *Существует взаимно однозначное соответствие между множеством всех перестановок множества* $\{1, 2, \ldots, n\}$ *и множеством всех упорядоченных пар табло* (P, Q), *где* P *и* Q*—табло одинаковой формы из элементов* $\{1, 2, \ldots, n\}$.

(Пример к этой теореме содержится в приведенном ниже доказательстве.)

Доказательство Удобнее доказать несколько более общий результат. По произвольному двустрочному массиву

$$\begin{pmatrix} q_1 & q_2 & \ldots & q_n \\ p_1 & p_2 & \ldots & p_n \end{pmatrix}, \qquad \begin{array}{l} q_1 < q_2 < \ldots < q_n, \\ p_1, p_2, \ldots, p_n \text{ все разные}, \end{array} \tag{11}$$

построим соответствующие два табло P и Q, где P состоит из элементов $\{p_1, p_2, \ldots, p_n\}$, а Q—из элементов $\{q_1, q_2, \ldots, q_n\}$, причем P и Q имеют одинаковую форму.

Пусть P и Q вначале пусты. При $i = 1, 2, \ldots, n$ (именно в таком порядке) проделаем следующую операцию: вставим P_i в табло P при помощи алгоритма I; затем установим $Q_{st} \leftarrow q_l$, где s и t определяют вновь заполненную позицию в P.

Например, если задана перестановка $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$, действуем следующим образом:

	P	Q
Вставка 7:	$\begin{array}{\|c\|}\hline 7 \\ \hline \end{array}$	$\begin{array}{\|c\|}\hline 1 \\ \hline \end{array}$
Вставка 2:	$\begin{array}{\|c\|}\hline 2 \\ \hline 7 \\ \hline \end{array}$	$\begin{array}{\|c\|}\hline 1 \\ \hline 3 \\ \hline \end{array}$
Вставка 9:	$\begin{array}{\|c\|c\|}\hline 2 & 9 \\ \hline 7 \\ \cline{1-1} \end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 \\ \cline{1-1} \end{array}$
Вставка 5:	$\begin{array}{\|c\|c\|}\hline 2 & 5 \\ \hline 7 & 9 \\ \hline \end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline \end{array}$
Вставка 3:	$\begin{array}{\|c\|c\|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1} \end{array}$	$\begin{array}{\|c\|c\|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1} \end{array}$

(12)

Следовательно, пара табло (P, Q), соответствующая перестановке $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$, такова:

$$P = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1} \end{array}, \qquad Q = \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1} \end{array}. \tag{13}$$

Из построения ясно, что P и Q всегда имеют одинаковую форму. Кроме того, поскольку элементы всегда добавляются на границу Q и в возрастающем порядке, то Q—табло.

Обратно, если заданы два табло одинаковой формы, то соответствующий двустрочный массив (11) можно построить так. Пусть

$$q_1 < q_2 < \ldots < q_n$$

—элементы Q. Пусть при $i = n, \ldots, 2, 1$ (именно в таком порядке) p_i—элемент x, который удаляется из P по алгоритму D с использованием значений s, t, таких, что $Q_{st} = q_i$.

Например, если применить это построение к табло (13) и производить вычисления, обратные (12), до тех пор, пока P не исчерпается, то получится массив $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$.

Поскольку алгоритмы I и D взаимно обратны, то взаимно обратны и описанные здесь два построения; таким образом, требуемое взаимно однозначное соответствие установлено. ∎

Соответствие, определенное в доказательстве теоремы A, обладает множеством поразительных свойств, и теперь мы приступим к выводу некоторых из них. Убедительная просьба к читателю, прежде чем двигаться дальше, испытать себя на примерах из упр. 1, чтобы освоиться с этими построениями.

Как только элемент вытеснен из строки 1 в строку 2, он уже больше не влияет на строку 1; кроме того, строки 2, 3, ... строятся из последовательности "вытесненных" элементов точно так же, как строки 1, 2, ... строятся из исходной перестановки. Эти факты наводят на мысль о том, что на построение в теореме A можно взглянуть иначе, обращая внимание лишь на первые строки P и Q. Например, перестановка $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$ вызывает следующие действия над строкой 1 [ср. с (12)]:

$$\begin{array}{lll} 1\text{: Вставить } 7, & \text{установить} & Q_{11} \leftarrow 1. \\ 3\text{: Вставить } 2, & \text{вытеснить } 7. & \\ 5\text{: Вставить } 9, & \text{установить} & Q_{12} \leftarrow 5. \\ 6\text{: Вставить } 5, & \text{вытеснить } 9. & \\ 8\text{: Вставить } 3, & \text{вытеснить } 5. & \end{array} \tag{14}$$

Таким образом, первая строка P—это 2 3, а первая строка Q—это 1 5. Кроме того, остальные строки P и Q составляют табло, соответствующие "вытесненному" двустрочному массиву

$$\begin{pmatrix} 3 & 6 & 8 \\ 7 & 9 & 5 \end{pmatrix}, \tag{15}$$

Чтобы понять, как строится строка 1, можно изучить элементы, попадающие в данный столбец этой строки. Будем говорить, что пара (q_i, p_i) принадлежит классу t относительно двустрочного массива

$$\begin{pmatrix} q_1 & q_2 & \ldots & q_n \\ p_1 & p_2 & \ldots & p_n \end{pmatrix}, \qquad \begin{array}{l} q_1 < q_2 < \ldots < q_n, \\ p_1, p_2, \ldots, p_n \text{ все разные,} \end{array} \tag{16}$$

если $p_i = P_{1t}$ после применения алгоритма I последовательно к p_1, p_2, ..., p_i, начиная с пустого табло P. (Напомним, что алгоритм I всегда вставляет данный элемент в 1-ю строку.)

Легко видеть, что (q_i, p_i) принадлежит классу 1 тогда и только тогда, когда p_i имеет $(i-1)$ инверсий, т. е. $p_i = \min\{ p_1, p_2, \ldots, p_i \}$—"левосторонний минимум". Если в массиве (16) вычеркнуть столбцы класса 1, то получится другой двустрочный массив:

$$\begin{pmatrix} q'_1 & q'_2 & \ldots & q'_m \\ p'_1 & p'_2 & \ldots & p'_m \end{pmatrix} \tag{17}$$

такой, что пара (q, p) принадлежит классу t относительно (17) тогда и только тогда, когда она принадлежит классу $t+1$ относительно массива (16). Операция перехода от (16) к (17) соответствует удалению крайней левой позиции строки 1. Это дает систематический способ определения классов. Например, в перестановке $\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$ левосторонними минимумами являются элементы 7 и 2, так что класс 1—это $\{ (1,7), (3,2) \}$; в оставшемся массиве $\begin{pmatrix} 5 & 6 & 8 \\ 9 & 5 & 3 \end{pmatrix}$ все элементы минимальны, так что класс 2—это $\{ (5,9), (6,5), (8,3) \}$. В "вытесненном" массиве (15) класс 1—это $\{ (3,7), (8,5) \}$, а класс 2—$\{ (6,9) \}$.

Для любого фиксированного t элементы класса t можно пометить

$$(q_{i_1}, p_{i_1}), \ldots, (q_{i_k}, p_{i_k}),$$

так чтобы выполнялись неравенства

$$\begin{aligned} q_{i_1} < q_{i_2} < \ldots < q_{i_k}, \\ p_{i_1} > p_{i_2} > \ldots > p_{i_k}, \end{aligned} \tag{18}$$

поскольку при работе алгоритма вставки позиция табло P_{1t} принимает убывающую последовательность значений p_{i_1}, ..., p_{i_k}. В конце построения

$$p_{1t} = p_{i_k}, \quad Q_{1t} = q_{i_1}, \tag{19}$$

а вытесненный двустрочный массив, которым определяются строки 2, 3, ... табло P и Q, содержит столбцы

$$\begin{pmatrix} q_{i_2} & q_{i_3} & \dots & q_{i_k} \\ p_{i_1} & p_{i_2} & \dots & p_{i_{k-1}} \end{pmatrix}, \tag{20}$$

а также другие столбцы, аналогичным образом полученные из других классов.

Эти наблюдения приводят к простому методу вычисления P и Q вручную (см. упр. 3), а также предоставляют средства для доказательства одного весьма неожиданного результата.

Теорема В. *Если в построении из теоремы А перестановка*

$$\begin{pmatrix} 1 & 2 & \dots & n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}$$

соответствует табло (P, Q), *то обратная ей перестановка соответствует табло* (Q, P).

Это довольно удивительный факт, потому что в теореме А табло P и Q формируются совершенно разными способами, и обратная перестановка получается в результате весьма причудливой перетасовки столбцов двустрочного массива.

Доказательство Предположим, у нас есть двустрочный массив (16); поменяв местами его строки и отсортировав столбцы так, чтобы элементы новой верхней строки расположились в неубывающем порядке, получим "обратный" массив

$$\begin{aligned} \begin{pmatrix} q_1 & q_2 & \dots & q_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}^{-1} &= \begin{pmatrix} p_1 & p_2 & \dots & p_n \\ q_1 & q_2 & \dots & q_n \end{pmatrix} = \\ &= \begin{pmatrix} p'_1 & p'_2 & \dots & p'_n \\ q'_1 & q'_2 & \dots & q'_n \end{pmatrix}, \qquad \begin{matrix} p'_1 < p'_2 < \dots < p'_n; \\ q'_1, q'_2, \dots, q'_n \text{ все разные.} \end{matrix} \end{aligned} \tag{21}$$

Покажем, что эта операция соответствует замене (P, Q) на (Q, P) в построении из теоремы А.

В упр. 2 наши замечания об определении классов переформулированы таким образом, что класс, к которому относится пара (q_i, p_i), не зависит от того факта, что элементы q_1, q_2, ..., q_n расположены в возрастающем порядке. Поскольку результирующие условия симметричны относительно p и q, то операция (21) не нарушает структуру классов; если (q, p) принадлежит классу t относительно (16), то (p, q) принадлежит классу t относительно (21). Поэтому, если разместить элементы этого последнего класса t так, чтобы

$$\begin{aligned} p_{i_k} < \dots < p_{i_2} < p_{i_1}, \\ q_{i_k} > \dots > q_{i_2} > q_{i_1}, \end{aligned} \tag{22}$$

[ср. с (18)], то получим

$$P_{1t} = q_{i_1}, Q_{1t} = p_{i_k}, \tag{23}$$

как в (19), а столбцы

$$\begin{pmatrix} p_{i_{k-1}} & \dots & p_{i_2} & p_{i_1} \\ q_{i_k} & \dots & q_{i_3} & q_{i_2} \end{pmatrix} \tag{24}$$

войдут в вытесненный массив, как в (20). Следовательно, первые строки P и Q меняются местами. Кроме того, вытесненный двустрочный массив для (21) является обратным по отношению к вытесненному двустрочному массиву для (16), так что доказательство завершается применением индукции по числу строк в табло. ∎

Следствие. *Количество табло, которые можно сформировать из элементов* $\{1, 2, \dots, n\}$, *равно количеству инволюций множества* $\{1, 2, \dots, n\}$.

Доказательство Если π—инволюция, соответствующая паре табло (P, Q), то $\pi = \pi^{-1}$ соответствует паре (Q, P). Следовательно, $P = Q$. Обратно, если π—какая-либо перестановка, соответствующая паре (P, P), то π^{-1} тоже соответствует паре (P, P); отсюда $\pi = \pi^{-1}$. Таким образом, существует взаимно однозначное соответствие между инволюциями π и табло P. ∎

Ясно, что элемент в левом верхнем углу табло всегда наименьший. Это наводит на мысль о возможном способе сортировки множества чисел. Сначала можно составить из них табло, многократно применяя алгоритм I, в результате наименьший элемент окажется в углу. Затем этот наименьший элемент удаляется, а остальные элементы переразмещаются так, чтобы образовалось другое табло; потом удаляется новый минимальный элемент и т.д.

Поэтому давайте посмотрим, что происходит, когда мы удаляем угловой элемент из табло

1	3	5	8	12	16
2	6	9	15		
4	10	14			
11	13				
17					

(25)

После удаления 1 на освободившееся место необходимо поставить 2. Затем можно поднять 4 на место двойки, однако 11 нельзя поднять на место 4, но на это место можно подвинуть 10, а потом 13 на место 10. В общем случае приходим к следующей процедуре.

Алгоритм S. *(Удаление углового элемента.)* Этот алгоритм удаляет элемент из левого верхнего угла табло P и перемещает остальные элементы так, чтобы сохранились свойства табло. Используются те же обозначения, что и в алгоритмах I и D.

S1 [Начальная установка.] Установить $r \leftarrow 1$, $s \leftarrow 1$.

S2 [Конец?] Если $P_{rs} = \infty$, то процесс завершен.

S3 [Сравнить.] Если $P_{(r+1)s} \lesssim P_{r(s+1)}$, то перейти к шагу **S5**. (Сравниваем элементы справа и снизу от свободного места и передвигаем меньший из них.)

S4 [Подвинуть влево.] Установить $P_{rs} \leftarrow P_{r(s+1)}$, $s \leftarrow s+1$ и вернуться к **S3**.

S5 [Подвинуть вверх.] Установить $P_{rs} \leftarrow P_{(r+1)s}$, $r \leftarrow r+1$ и вернуться к **S2**. ∎

Легко доказать, что после удаления углового элемента с помощью алгоритма S, P—по-прежнему табло (см. упр. 10). Таким образом, применяя алгоритм S до тех пор, пока P не исчерпается, можно прочитать его элементы в возрастающем порядке. К сожалению, этот алгоритм сортировки оказывается не столь эффективным, как другие алгоритмы, которые нам еще предстоит рассмотреть. Минимальное время его работы пропорционально $n^{1.5}$; аналогичные алгоритмы, использующие вместо табло деревья, затрачивают время порядка $n \log n$.

Алгоритм S, хотя и не приводит к особенно эффективному алгоритму сортировки, обладает некоторыми очень интересными свойствами.

Теорема C. *(М. П. Шюценберже.) Если P—табло, построенное, как в теореме A, из перестановки $a_1\, a_2, \ldots\, a_n$, и $a_i = \min\{\, a_1, a_2, \ldots, a_n \,\}$, то алгоритм S преобразует P в табло, соответствующее перестановке $a_1\ \ldots\ a_{i-1}\, a_{i+1}\ \ldots\ a_n$.* ∎

Доказательство См. упр. 13. ∎

Давайте после применения алгоритма S поместим на вновь освободившееся место удаленный элемент в скобках, указав таким образом, что на самом деле он не является частью табло. Применив, например, эту процедуру к табло (25), мы получили бы

2	3	5	8	12	16
4	6	9	15		
10	13	14			
11	(1)				
17					

а еще два применения приводят к

4	5	8	12	16	(2)
6	9	14	15		
10	13	(3)			
11	(1)				
17					

Продолжая до тех пор, пока все элементы не окажутся в скобках, и убрав скобки, получим конфигурацию

17	15	14	13	11	2
16	10	6	4		
12	5	3			
9	1				
8					

(26)

имеющую ту же форму, что и исходное табло (25). Эту конфигурацию можно назвать *двойственным табло,* потому что она похожа на табло с той лишь разницей, что применяется "двойственное отношение порядка" ($<$ и $>$ поменялись ролями). Обозначим двойственное табло, полученное из P таким способом, через P^S.

Табло P определяется из P^S единственным образом. В самом деле, исходное табло можно получить из P^S при помощи того же самого алгоритма (с обратным отношением порядка, поскольку P^S—двойственное табло). Например, применение к (26) двух шагов этого алгоритма дает

15	14	13	11	2	(16)
12	10	6	4		
9	5	3			
8	1				
(17)					

и в конце концов опять получается табло (25). Этот замечательный результат—одно из следствий нашей следующей теоремы.

Теорема D. *(К. Шенстед, М. П. Шюценберже.) Пусть*

$$\begin{pmatrix} q_1 & q_2 & \dots & q_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix} \tag{27}$$

—двустрочный массив, соответствующий паре табло (P, Q).

a) *Если пользоваться двойственным (обратным) отношением порядка для* q, *но не для* p, *то двустрочный массив*

$$\begin{pmatrix} q_n & \dots & q_2 & q_1 \\ p_n & \dots & p_2 & p_1 \end{pmatrix} \tag{28}$$

соответствует паре $(P^T, (Q^S)^T)$.

(Как обычно, через T обозначена операция транспонирования; P^T—табло, а $(Q^S)^T$—двойственное табло, поскольку элементы q расположены в обратном порядке.)

b) *Если пользоваться двойственным отношением порядка для* p, *но не для* q, *то двустрочный массив (37) соответствует паре* $((P^S)^T, Q^T)$.

c) *Если пользоваться двойственным отношением порядка как для* p, *так и для* q, *то двустрочный массив (28) соответствует паре* (P^S, Q^S).

Доказательство Не известно простого доказательства этой теоремы. То, что случай (a) соответствует паре (P^T, X), где X—некоторое двойственное табло, доказано в упр. 6; следовательно, по теореме B, случай (b) соответствует паре (Y, Q^T), где Y—некоторое двойственное табло той же формы, что и P^T.

Пусть $p_i = \min\{p_1, \dots, p_n\}$; так как p_i—"наибольший" элемент при двойственном отношении порядка, то он окажется на границе Y и не вытесняет никаких элементов при построении из теоремы A. Таким образом, если последовательно вставлять элементы $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$, применяя двойственное отношение порядка, то получится $Y - \{p_i\}$, т.е. Y, из которого удален элемент p_i. По теореме C, если последовательно вставлять элементы $p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n$, применяя обычное отношение порядка, построим табло $d(P)$, которое получается путем применения к P алгоритма S. Индукция по n дает $Y - \{p_i\} = (d(P)^S)^T$. Но поскольку

$$(P^S)^T - \{p_i\} = (d(P)^S)^T \tag{29}$$

по определению операции S, а Y имеет ту же форму, что и $(P^S)^T$, то должно иметь место равенство $Y = (P^S)^T$.

Тем самым доказано утверждение (b); (a) получается применением теоремы B. Последовательное применение (a) и (b) показывает, что случай (c) соответствует паре $(((P^T)^S)^T, ((Q^S)^T)^T)$, а это равно (P^S, Q^S), так как $(P^S)^T = (P^T)^S$ вследствие симметрии операции S по отношению к строкам и столбцам. ∎

Эта теорема, в частности, устанавливает два удивительных факта, касающихся алгоритма вставки в табло. Если в результате последовательной вставки различных элементов $p_1, \dots, p_n$ в пустое табло получается табло P, то в результате вставки этих элементов в обратном порядке—$p_n, \dots, p_1$, получится *транспонированное* табло P^T. Если же мы не только станем вставлять элементы в порядке $p_n, \dots, p_1$, но и поменяем ролями $<$ и $>$, а также 0 и ∞, то получим двойственное табло P^S. Настоятельно рекомендуем читателю испытать эти процессы на нескольких простых примерах. Необычная природа этих совпадений

может вызвать подозрение о вмешательстве каких-то колдовских сил. До сих пор не известно какого-либо простого объяснения подобных явлений; кажется, не существует простого способа доказать даже то, что случай (c) соответствует табло той же *формы*, что P и Q.

Соответствие, устанавливаемое теоремой A, найдено Ж. Робинсоном [*American J. Math.*, **60** (1938), 745–760, Sec. 5] в несколько иной и довольно туманной форме как часть решения весьма сложной задачи из теории групп. Нетрудно проверить, что его построение в сущности идентично приведенному здесь. Он сформулировал теорему B без доказательства. Много лет спустя К. Шенстед независимо заново открыл это соответствие, которое он сформулировал по существу в той же форме, какую использовали мы [*Canadian J. Math.*, **13** (1961), 179–191]. Он также доказал "P"-часть теоремы D (a). М. П. Шюценберже [*Math. Scand.*, **12** (1963), 117–128] доказал теорему B и "Q"-часть теоремы D (a), из которой следуют (b) и (c). Это соответствие можно распространить и на перестановки мультимножеств; случай, когда $p_1, \ldots, p_n$ не обязательно различны, рассмотрел Шенстед, а "максимальное" обобщение на случай, когда и p, и q могут содержать повторяющиеся элементы, исследовано Кнутом [*Pacific J. Math.*, **34** (1970), 709–727].

Обратимся теперь к родственному вопросу: *сколько табло, составленных из элементов* $\{1, 2, \ldots, n\}$, *имеют данную форму* $(n_1, n_2, \ldots, n_m)$, *где* $n_1+n_2+\cdots+n_m = n$? Обозначим это число через $f(n_1, n_2, \ldots, n_m)$; оно должно удовлетворять соотношениям

$$f(n_1, n_2, \ldots, n_m) = 0, \qquad \text{если не выполнено условие } n_1 \ge n_2 \ge \ldots \ge n_m \ge 0; \tag{30}$$

$$f(n_1, n_2, \ldots, n_m, 0) = f(n_1, n_2, \ldots, n_m); \tag{31}$$

$$f(n_1, n_2, \ldots, n_m) = f(n_1 - 1, n_2, \ldots, n_m) + f(n_1, n_2 - 1, \ldots, n_m) + \cdots + f(n_1, n_2, \ldots, n_m - 1), \quad \text{если } n_1 \ge n_2 \ge \ldots \ge n_m \ge 1. \tag{32}$$

Последнее рекуррентное соотношение вытекает из того факта, что при удалении наибольшего элемента табло всегда снова получается табло; например, количество табло формы $(6, 4, 4, 1)$ равно $f(5, 4, 4, 1) + f(6, 3, 4, 1)+f(6, 4, 3, 1)+f(6, 4, 4, 0) = f(5, 4, 4, 1)+f(6, 4, 3, 1)+f(6, 4, 4)$, потому что всякое табло формы $(6, 4, 4, 1)$ из элементов $\{1, 2, \ldots, 15\}$ получается в результате вставки элемента 15 в подходящее место в табло формы $(5, 4, 4, 1)$, $(6, 4, 3, 1)$ или $(6, 4, 4)$. Изобразим это на схеме

Picture: p. 80, (33)

Функция $f(n_1, n_2, \ldots, n_m)$, удовлетворяющая таким соотношениям, имеет довольно простой вид:

$$f(n_1, n_2, \ldots, n_m) = \frac{\Delta(n_1 + m - 1, n_2 + m - 2, \ldots, n_m) n!}{(n_1 + m - 1)!(n_2 + m - 2)! \ldots n_m!} \qquad \text{при } n_1 + m - 1 \ge n_2 + m - 2 \ge \ldots \ge n_m, \tag{34}$$

где через Δ обозначен детерминант

$$\Delta(x_1, x_2, \ldots, x_m) = \det \begin{pmatrix} x_1^{m-1} & x_2^{m-1} & \ldots & x_m^{m-1} \\ \vdots & & & \vdots \\ x_1^2 & x_2^2 & & x_m^2 \\ x_1 & x_2 & & x_m \\ 1 & 1 & \ldots & 1 \end{pmatrix} = \prod_{1 \le i < j \le m} (x_i - x_j) \tag{35}$$

Формулу (34) вывел Ф. Фробениус [Sitzungsberichte Preuss. Akad. der Wissenchaften (Berlin, 1900), 516–534, Sec. 3], изучая эквивалентную задачу теории групп; он использовал довольно глубокую аргументацию, опирающуюся на теорию групп. Комбинаторное доказательство независимо нашел Мак-Магон [*Philosophical Trans.*, **A-209** (London, 1909), 153–175]. Эту формулу можно доказать по индукции, так как (30) и (31) доказываются без труда, а формула (32) получается, если положить $y = -1$ в тождестве упр. 17.

Из теоремы A в связи с этой формулой для числа табло вытекает замечательное тождество. Взяв сумму по всевозможным формам табло, получим

$$\begin{aligned} n! &= \sum_{\substack{k_1 \ge \ldots \ge k_n \ge 0 \\ k_1 + \cdots + k_n = n}} f(k_1, \ldots, k_n)^2 = \\ &= n!^2 \sum_{\substack{k_1 \ge \ldots \ge k_n \ge 0 \\ k_1 + \cdots + k_n = n}} \frac{\Delta(k_1 + n - 1, \ldots, k_n)^2}{(k_1 + n - 1)!^2 \ldots k_n!^2} = \\ &= n!^2 \sum_{\substack{q_1 > q_2 > \ldots > q_n \ge 0 \\ q_1 + \cdots + q_n = (n+1)n/2}} \frac{\Delta(q_1, \ldots, q_n)^2}{q_1!^2 \ldots q_n!^2}; \end{aligned}$$

отсюда

$$\sum_{\substack{q_1+\cdots+q_n=(n+1)n/2\\ q_1\ldots q_n\ge 0}} \frac{\Delta(q_1,\ldots,q_n)^2}{q_1!^2\ldots q_n!^2}=1. \tag{36}$$

В последней сумме отсутствуют неравенства $q_1>q_2>\ldots>q_n$, потому что слагаемые—симметричные относительно q функции, обращающиеся в 0 при $q_i=q_j$. Аналогичное тождество появляется в упр. 24.

Формулу числа табло можно выразить несколько более интересным способом, если ввести понятие "уголков". В *уголок*, соответствующий

Picture: Рис 5. Уголки и длины уголков.

клетке табло, входит сама эта клетка плюс все клетки, лежащие снизу и справа от нее. Например, заштрихованный участок на рис. 5—уголок, соответствующий клетке $(2,3)$ строки 2 и столбца 3; он состоит из шести клеток. В каждой клетке на рис. 5 записана длина соответствующего ей уголка.

Если табло имеет форму $(n_1,n_2,\ \ldots,n_m)$, где $n_m\ge 1$, то длина самого длинного уголка равна n_1+m-1. Дальнейшее исследование длин уголков показывает, что строка 1 содержит все длины n_1+m-1, $n_1+m-2,\ \ldots,\ 1$, *кроме* $(n_1+m-1)-(n_m)$, $(n_1+m-1)--(n_{m-1}+1),\ \ldots,\ (n_1+m-1)-(n_2+m-2)$. Например, на рис. 5 длины уголков в 1-й строке суть 12, 11, 10, ..., 1, за исключением 10, 9, 6, 3, 2; эти исключения соответствуют пяти несуществующим уголкам, начинающимся в несуществующих клетках $(6,3)$, $(5,3)$, $(4,5)$, $(3,7)$, $(2,7)$ и заканчивающимся в клетке $(1,7)$. Аналогично j-я строка содержит все длины уголков $n_j+m-j,\ \ldots,\ 1$, кроме $(n_j+m-j)-(n_m),\ \ldots,\ (n_j-m-j)-(n_{j+1}-m-j-1)$. Отсюда следует, что произведение длин всех уголков равно

$$(n_1+m-1)!\ldots(n_m)!/\Delta(n_1+m-1,\ \ldots,n_m).$$

Это выражение входит в формулу (34); отсюда следует

Теорема H. *(Дж. С. Фрэйм, Ж. Робинсон, Р. М. Тролл.) Количество табло данной формы, составленных из элементов* $\{1,2,\ \ldots,n\}$, *равно* $n!$, *деленному на произведение длин уголков.* ∎

Такой простой результат заслуживает и простого доказательства; однако (как и для большинства других фактов, касающихся табло) более прямого доказательства не известно. Каждый элемент табло—минимальный в своем уголке; если заполнить клетки табло случайным образом, то вероятность того, что в клетке (i,j) окажется минимальный элемент соответствующего уголка, есть величина, обратная длине уголка. Перемножение этих вероятностей по всем i, j дает теорему H. Однако такое рассуждение ошибочно, поскольку эти вероятности отнюдь не являются независимыми. Все известные доказательства теоремы H основаны на одном неубедительном индуктивном рассуждении, которое на самом деле не объясняет, почему же теорема верна (так как в нем совершенно не используются свойства уголков).

Существует интересная связь между теоремой H и перечислением деревьев, которое рассматривалось в гл. 2. Мы видели, что бинарным деревьям с n узлами соответствуют перестановки, которые можно получить с помощью стека, и что таким перестановкам соответствуют последовательности $a_1\,a_2\,\ldots\,a_{2n}$ литер S и X, такие, что количество литер S никогда не бывает меньше количества литер X, если читать последовательность слева направо. (См. упр. 2.3.1-6 и 2.2.1-3.) Таким последовательностям естественным образом сопоставляются табло формы (n,n); в 1-ю строку помещаются индексы i, такие, что $a_i=S$, а во 2-ю строку—индексы, при которых $a_i=X$. Например, последовательности

$$S\ S\ S\ X\ X\ S\ S\ X\ X\ S\ X\ X$$

соответствует табло

1	2	3	6	7	10
4	5	8	9	11	12

(37)

Условие, налагаемое на столбцы, удовлетворяется в таком табло в том и только том случае, когда при чтении слева направо число литер X никогда не превышает числа литер S. По теореме H количество всевозможных табло формы (n,n) равно

$$(2n)!/(n+1)!n!;$$

следовательно, таково же и число бинарных деревьев с n узлами (что согласуется с формулой (2.3.4.4-13)). Более того, если воспользоваться табло формы (n,m) при $n\ge m$, то при помощи этого рассуждения можно решить и более общую "задачу о баллотировке", рассмотренную в ответе к упр. 2.2.1-4. Таким образом, теорема H в качестве простых частных случаев включает в себя некоторые весьма сложные задачи о перечислении.

Всякому табло A формы (n, n) из элементов $\{1, 2, \ldots, 2n\}$ соответствуют два табло (P, Q) одинаковой формы. Следующий способ построения такого соответствия предложен Мак-Магоном [Combinatory Analysis, **1** (1915), 130–131]. Пусть P состоит из элементов $\{1, \ldots, n\}$, расположенных, как в A, а Q получается, если взять остальные элементы A, повернуть всю конфигурацию на 180° и заменить $n+1$, $n+2$, $\ldots$, $2n$ на соответственно n, $n-1$, $\ldots$, 1. Например, табло (37) распадается на

1	2	3	6
4	5		

и

		7	10
8	9	11	12

;

после поворота и перенумерации имеем

$$P = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 3 & 6 \\ \hline 4 & 5 \\ \cline{1-2}\end{array}, \quad Q = \begin{array}{|c|c|c|c|}\hline 1 & 2 & 4 & 5 \\ \hline 3 & 6 \\ \cline{1-2}\end{array}. \tag{38}$$

Наоборот, каждой паре табло одинаковой формы, состоящих иа n элементов и из не более двух строк, соответствует табло формы (n, n). Следовательно (из упр. 7), *число перестановок $a_1\, a_2 \ldots a_n$ множества $\{1, 2, \ldots, n\}$, не содержащих убывающих подпоследовательностей $a_i > a_j > a_k$ при $i < j < k$, равно числу бинарных деревьев с n углами.* Интересное взаимно однозначное соответствие между такими перестановками и бинарными деревьями, установленное более прямым способом, чем тот окольный путь посредством алгоритма I. которым воспользовались мы, нашел Д. Ротом [*Inf. Proc. Letters,* **4** (1975), 58–61]; аналогично, существует довольно прямое соответствие между бинарными деревьями и перестановками, не содержащими подпоследовательностей $a_j > a_k > a_i$ и $i < j < k$ (см. упр. 2.2.1-5).

Число способов заполнить табло формы $(6, 4, 4, 1)$, очевидно, равно числу способов пометить вершины направленного графа

Picture: p.84

числами $\{1, 2, \ldots, 15\}$ так, чтобы метка вершины u была меньше метки вершины v, если $u \to v$. Иначе говоря, оно равно числу способов топологически отсортировать частичное упорядочение (39) в смысле п. 2.2.3.

В общем случае можно задать тот же вопрос для произвольного направленного графа, не содержащего ориентированных циклов. Было бы хорошо, если бы существовала какая-нибудь простая формула, обобщающая теорему H на случай произвольного графа; однако не все графы обладают такими приятными свойствами, как графы, соответствующие табло. Другие классы направленных графов, для которых задача о разметке вершин имеет простое решение, частично обсуждаются в упражнениях в конце этого пункта. Там также имеются упражнения, в которых показано, что в ряде случаев направленных графов *не* существует простой формулы, соответствующей теореме H. Например, число способов пометить вершины не всегда является делителем $n!$.

Завершим наши исследования подсчетом общего числа табло, которые можно сформировать из n различных элементов. Обозначим это число через t_n. По следствию из теоремы B t_n равно числу инволюций множества $\{1, 2, \ldots, n\}$. Перестановка обратна самой себе тогда и только тогда, когда ее представление с помощью циклов состоит только из единичных циклов (неподвижных точек) и циклов из двух элементов (транспозиций). Поскольку в t_{n-1} из t_n инволюций (n)—неподвижная точка, а в t_{n-2} из них (j, n)—цикл из двух элементов при некотором фиксированном $j < n$, то получаем формулу

$$t_n = t_{n-1} + (n-1)t_{n-2}, \tag{40}$$

которую вывел Роте в 1800 г. для получении таблицы значений t_n при малых n.

Вычислим t_n другим способом. Пусть имеется k циклов из двух элементов и $n-2k$ единичных циклов. Существует $\binom{n}{2k}$ способов выбрать неподвижные точки, и мультиномиальный коэффициент $(2k)!/(2!)^k$ равен числу способов организовать остальные элементы в k различимых транспозиций. Поделив на $k!$, чтобы сделать эти транспозиции неразличимыми, получим

$$t_n = \sum_{k \geq 0} t_n(k), \quad t_n(k) = \frac{n!}{(n-2k)!2^k k!}. \tag{41}$$

К сожалению, насколько это известно, такую сумму уже нельзя упростить, поэтому, для того чтобы лучше понять поведение величины t_n, мы применим два косвенных подхода:

a) Можно найти производящую функцию

$$\sum_n t_n z^n / n! = e^{z + z^2/2}; \tag{42}$$

см. упр. 25.

b) Можно определить асимптотическое поведение величины t_n. Это поучительная задача, ее решение содержит несколько общих методов, которые будут полезны и в связи с другими вопросами; поэтому мы заключим этот пункт анализом асимптотического поведения t_n.

Первый шаг в анализе асимптотического поведения (41) состоит в выделении слагаемых, дающих основной вклад в сумму. Поскольку

$$\frac{t_n(k+1)}{t_n(k)} = \frac{(n-2k)(n-2k-1)}{2(k+1)}, \tag{43}$$

можно видеть, что слагаемые постепенно возрастают, когда k меняется от $k=0$ до k, равного приблизительно $\frac{1}{2}(n-\sqrt{n})$, а затем убывают до нуля, когда k достигает значения, равного примерно $\frac{1}{2}n$. Ясно, что основной вклад дают члены в окрестности $k=\frac{1}{2}(n-\sqrt{n})$. Для анализа, однако, удобнее, когда основной вклад достигается при значении 0, поэтому представим k в виде

$$k = \frac{1}{2}(n-\sqrt{n}) + x \tag{44}$$

и исследуем величину $t_n(k)$ как функцию от x.

Чтобы избавиться от факториалов в $t_n(k)$, полезно воспользоваться формулой Стирлинга (1.2.11.2-18). Для этой цели удобно (как мы вскоре увидим) ограничить область изменения x промежутком

$$-(n^{\varepsilon+1/4}) \le x \le n^{\varepsilon+1/4}, \tag{45}$$

где ε равно, скажем, 0.001, так что можно включить слагаемое погрешности. После довольно трудоемких вычислений, которые на самом деле должны были быть проделаны вычислительной машиной, получается формула

$$t_n(k) = \exp\left(\frac{1}{2}n\ln n - \frac{1}{2}n + \sqrt{n} - \frac{1}{4}\ln n - 2x^2/\sqrt{n} - \right.$$
$$\left. -\frac{1}{4} - \frac{1}{2}\ln\pi - \frac{4}{3}x^3/n + 2x/\sqrt{n} + \frac{1}{3}/\sqrt{n} - \frac{4}{3}x^4/n\sqrt{n} + O(n^{5\varepsilon-3/4})\right). \tag{46}$$

Ограничение (45) на x оправдывается тем, что можно положить $x = \pm n^{\varepsilon+1/4}$ для получения верхней оценки всех отброшенных слагаемых, именно

$$e^{-2n^{2\varepsilon}} \cdot \exp\left(\frac{1}{2}n\ln n - \frac{1}{2}n + \sqrt{n} - \frac{1}{4}\ln n - \frac{1}{4} - \frac{1}{2}\ln\pi + O(n^{3\varepsilon-1/4})\right), \tag{47}$$

а если умножить это на n, то получится верхняя оценка суммы исключенных слагаемых. Верхняя оценка имеет меньший порядок, чем те слагаемые, которые мы вычислим для x в ограниченном промежутке (45), благодаря множителю $\exp(-2n^{2\varepsilon})$, который много меньше любого полинома от n. Очевидно, можно отбросить в сумме множитель

$$\exp\left(\frac{1}{2}n\ln n - \frac{1}{2}n + \sqrt{n} - \frac{1}{4}\ln n - \frac{1}{4} - \frac{1}{2}\ln\pi + \frac{1}{3}/\sqrt{n}\right), \tag{48}$$

после чего нам останется только просуммировать

$$\exp\left(-2x^2/\sqrt{n} - \frac{4}{3}x^3/n + 2x/\sqrt{n} - \frac{4}{3}x^4/n\sqrt{n} + O(n^{5\varepsilon-3/4})\right) =$$
$$= \exp\left(\frac{-2x^2}{\sqrt{n}}\right)\left(1 - \frac{4}{3}\frac{x^3}{n} + \frac{8}{9}\frac{x^6}{n^2}\right) \times \left(1 + 2\frac{x}{\sqrt{n}} + 2\frac{x^2}{n}\right)\left(1 - \frac{4}{3}\frac{x^4}{n\sqrt{n}}\right)(1 + O(n^{9\varepsilon-3/4})) \tag{49}$$

по всем $x = \alpha, \alpha+1, \ldots, \beta-2, \beta-1$, где $-\alpha$ и β (не обязательно целые) равны приблизительно $n^{\varepsilon+1/4}$. Если сместить интервал суммирования, то формулу суммирования Эйлера (1.2.11.2-10) можно записать в виде

$$\sum_{\alpha \le x < \beta} f(x) = \int_\alpha^\beta f(x)\,dx - \frac{1}{2}f(x)\Big|_\alpha^\beta + \frac{1}{2}B_2 \cdot \frac{f'(x)}{1!}\Big|_\alpha^\beta + \cdots + \frac{1}{m+1}B_{m+1} \cdot \frac{f^{(m)}(x)}{m!}\Big|_\alpha^\beta + R_{m+1}. \tag{50}$$

Здесь $|R_m| \le (4/(2\pi)^m) \int_\alpha^\beta |f^{(m)}(x)\,dx$. Полагая $f(x) = x^t \times \exp(-2^2/\sqrt{n})^\alpha)$, где t—фиксированное неотрицательное целое число, найдем, что $\int_\alpha^\beta f(x)\,dx$ отличается от $\int_{-\infty}^{\infty} f(x)\,dx$ на $O(\exp(-2\pi^\varepsilon))$, так что можно взять $\alpha = -\infty$, $\beta = \infty$. В этом случае формула суммирования Эйлера даст асимптотический ряд для $\sum f(x)$ при $n \to \infty$, поскольку

$$f^{(m)}(x) = n^{(t-m)/4} g^{(m)}(n^{-1/4}x), \quad g(y) = y^i e^{-2y^2}, \tag{51}$$

и $g(y)$—хорошая функция, не зависящая от n; отсюда видно, что $R_m = O(n^{(t+1-m)/4})$. Так как $f^{(m)}(-\infty) = f^m(\infty) = 0$, мы доказали, что

$$\sum_{\alpha \le x < \beta} f(x) = \int_{-\infty}^{\infty} f(x)\,dx + O(n^{-m}) \qquad \text{для любого } m \ge 0; \tag{52}$$

на самом деле при этом конкретном выборе $f(x)$ существен только интеграл! Интеграл нетрудно вычислить (см. упр. 26); таким образом, мы можем выполнить умножение и сложение в формуле (49) и получить

$$\sqrt{\frac{\pi}{2}} n^{1/4} \left(1 - \frac{1}{24} n^{-1/4} + O(n^{-1/2})\right);$$
$$t_n = \frac{1}{\sqrt{2}} n^{n/2} e^{-n/2+\sqrt{n}-1/4} \left(1 + \frac{7}{24} n^{-1/2} + O(n^{-3/4})\right). \tag{53}$$

В действительности слагаемые в $O(n^{-3/4})$ содержат еще экспоненту от 9ε, но из наших выкладок ясно, что величина 9ε должна исчезнуть, если произвести вычисления с большей точностью. В принципе этот метод можно усилить и вместо $O(n^{-3/4})$ получить $O(n^{-k})$ при любом k. Этот асимптотический ряд для t_n впервые нашли (другим способом) Мозер и Уаймэн [*Canadian. J. Math.*, **7** (1955), 159–168]. Другие примеры и усиления метода, примененного для вывода формулы (53), см. в конце п. 5.2.2.

Упражнения

1. [16] Какие табло (P, Q) соответствуют двустрочному массиву

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 4 & 9 & 5 & 7 & 1 & 2 & 8 & 3 \end{pmatrix}$$

в построении из теоремы A? Какой двустрочный массив соответствует паре табло

$$B = \begin{array}{|c|c|c|} \hline 1 & 4 & 7 \\ \hline 2 & 8 \\ \cline{1-2} 5 & 9 \\ \cline{1-2} \end{array}, \qquad C = \begin{array}{|c|c|c|} \hline 1 & 3 & 7 \\ \hline 4 & 5 \\ \cline{1-2} 8 & 9 \\ \cline{1-2} \end{array} ?$$

2. [M21] Докажите, что (q, p) принадлежит классу t относительно массива (16) тогда и только тогда, когда t равно максимальному числу индексов $i_1, \ldots, i_t$, таких, что

$$p_{i_1} < p_{i_2} < \ldots < p_{i_t} = p, \qquad q_{i_1} < q_{i_2} < \ldots < q_{i_t} = q.$$

>3. [M24] Покажите, что соответствие в теореме A можно также установить путем построения такой таблицы:

Строка 0	1	3	5	6	8
Строка 1	7	2	9	5	3
Строка 2	∞	7	∞	9	5
Строка 3		∞		∞	7
Строка 4					∞

Здесь в строках 0 и 1 записан данный двустрочный массив. При $k \ge 1$ строка $k+1$ образуется из строки k следующим образом:

a) Установить $p \leftarrow \infty$.
b) Пусть столбец j—самый левый столбец, в котором строка k содержит целое число $< p$, а соответствующее место в строке $k+1$ не заполнено. Если такого столбца нет и $p = \infty$, то строка $k+1$ заполнена; если такого столбца нет и $p < \infty$, то возвратиться к (a).

c) Вставить p в столбец j в строке $k+1$, потом установить p равным элементу столбца j и строки k и вернуться к (b).

После того как таблица таким образом построена, строка k табло P составляется из тех целых чисел строки k этой таблицы, которые отсутствуют в строке $(k+1)$; строка k табло Q строится из тех целых чисел строки 0, которые находятся в столбцах, содержащих ∞ в строке $(k+1)$.

4. [M26] (М. П. Шюценберже). Пусть π—инволюция с k неподвижными точками. Покажите, что табло, соответствующее π в доказательстве следствия из теоремы B, содержит ровно k столбцов нечетной длины.

>5. [M30] Пусть $a_1 \ldots a_{j-1}\, a_j \ldots a_n$—перестановка различных элементов и $1 < j \le n$. Перестановка $a_1 \ldots a_{j-2}\, a_j\, a_{j-1}\, a_{j+1}$ получающаяся из исходной, если поменять местами a_{j-1} и a_j, называется "допустимой", если либо

i) $j \ge 3$ и a_{j-2} лежит между a_{j-1} и a_j, либо
ii) $j < n$ и a_{j+1} лежит между a_{j-1} и a_j.

Например, в перестановке 1 5 4 6 8 3 7 можно произвести ровно три допустимые замены: можно поменять местами 1 и 5, поскольку $1 < 4 < 5$; можно поменять местами 8 и 3, поскольку $3 < 6 < 8$ (или $3 < 7 < 8$); однако нельзя менять местами 5 и 4 или 3 и 7.

a) Докажите, что при допустимой замене не меняется табло P, которое получается из перестановки путем последовательной вставки элементов a_1, a_2, ..., a_n в первоначально пустое табло.

b) Обратно, покажите, что любые две перестановки, соответствующие одному и тому же табло P, можно преобразовать одну в другую последовательностью одной или более допустимых замен. [*Указание:* покажите, что если табло P имеет форму $(n_1, n_2, \ldots, n_m)$, то любую соответствующую ему перестановку при помощи последовательности допустимых замен можно преобразовать в "каноническую перестановку" $P_{m1} \ldots P_{mn_m} \ldots P_{21} \ldots P_{2n_2} P_{11} \ldots P_{1n_1}$.]

>6. [M22] Пусть P—табло, соответствующее перестановке $a_1\, a_2 \ldots a_n$; с помощью упр. 5 докажите, что табло P^T соответствует перестановке $a_n \ldots a_2\, a_1$.

7. [M 20] (К. Шенстед). Пусть P—табло, соответствующее перестановке $a_1\, a_2 \ldots a_n$. Докажите, что число *столбцов* в P равно длине c максимальной возрастающей подпоследовательности $a_{i_1} < a_{i_2} < \ldots < a_{i_c}$, где $i_1 < i_2 < \ldots < i_c$; число *строк* в P равно длине r максимальной убывающей подпоследовательности $a_{j_1} > a_{j_2} > \ldots > a_{j_r}$, где $j_1 < j_2 < \ldots < j_r$.

8. [M18] (П. Эрдеш, Г. Секереш). Докажите, что в любой перестановке, состоящей из более чем n^2 элементов, имеется монотонная подпоследовательность длины более n; однако существуют перестановки n^2 элементов, не содержащие монотонных подпоследовательностей длины более n. [*Указание:* см. предыдущее упражнение.]

9. [M24] Продолжая упр. 8, найдите "простую" формулу точного числа перестановок множества $\{1, 2, \ldots, n^2\}$, не содержащих монотонных подпоследовательностей длины более чем n.

10. [M20] Докажите, что массив P является табло по окончании работы алгоритма S, если он был табло до этого.

11. [20] Можно ли восстановить исходный вид табло P по окончании работы алгоритма S, если известны только значения r и s?

12. [M24] Сколько раз выполняется шаг S3, если алгоритм S многократно применяется для исключения всех элементов из табло P формы $(n_1, n_2, \ldots, n_m)$? Каково минимальное значение этой величины по всем формам, таким, что $n_1 + n_2 + \cdots + n_m = n$?

13. [M28] Докажите теорему C.

14. [M43] Найдите более прямое доказательство части (c) теоремы D.

15. [M20] Сколько перестановок мультимножества $\{l \cdot a, m \cdot b, n \cdot c\}$ обладают тем свойством, что если читать перестановку слева направо, то число прочитанных букв c никогда не превышает числа букв b, которое в свою очередь не превышает числа букв a? (Например, перестановка $a\, a\, b\, c\, a\, b\, b\, c\, a\, c\, a$ обладает этим свойством.)

16. [M08] Сколькими способами можно топологически отсортировать частичное упорядочение, представляемое графом (39)?

17. [BM25] Пусть

$$g(x_1, x_2, \ldots, x_n; y) = x_1 \Delta(x_1 + y, x_2, \ldots, x_n) + x_2 \Delta(x_1, x_2 + y, \ldots, x_n) + \cdots + x_n \Delta(x_1, x_2, \ldots, x_n + y).$$

Докажите, что

$$g(x_1, x_2, \ldots, x_n; y) = \left(x_1 + x_2 + \cdots + x_n + \binom{n}{2} y\right) \Delta(x_1, x_2, \ldots, x_n).$$

[*Указание:* полином g однородный (все слагаемые имеют одинаковую суммарную степень) и антисимметричный по x (знак g изменится, если поменять местами x_i и x_j).]

18. [BM30] Обобщая упр. 17, вычислите при $m \geq 0$ сумму

$$x_1^m \Delta(x_1 + y, x_2, \ldots, x_n) + \Delta x_2^m (x_1, x_2 + y, \ldots, x_n) + \cdots + x_n^m \Delta(x_1, x_2, \ldots, x_n + y).$$

19. [M40] Найдите формулу для определения числа способов, которыми можно заполнить массив, подобный табло, но без первых двух клеток в первой строке; например, массив такой формы:

Picture: 1. p.90

(Элементы в строках и столбцах должны располагаться в возрастающем порядке, как в обычных табло.)

Иначе говоря, сколько из $f(n_1, n_2, \ldots, n_m)$ табло формы $(n_1, n_2, \ldots, n_m)$, составленных из элементов $\{1, 2, \ldots, n_1 + \cdots + n_m\}$, содержат элементы 1 и 2 в первой строке?

>20. [M24] Докажите, что число способов пометить узлы данного бинарного дерева элементами $\{1, 2, \ldots, n\}$ так, чтобы метка каждого узла была меньше метки любого из его потомков, равно частному от деления $n!$ на произведение "длин поддеревьев", т. е. количеств узлов в каждом поддереве. (Ср. с теоремой H.). Например, число способов пометить узлы дерева

Picture: 1. p.90

равно $10!/10 \cdot 4 \cdot 5 \cdot 1 \cdot 2 \cdot 3 \cdot 1 \cdot 1 \cdot 1 \cdot 1 = 9 \cdot 8 \cdot 7 \cdot 6$.

21. [BM3I] (P. M. Тролл). Пусть числа $n_1 > n_2 > \ldots > n_m$ описывают форму "сдвинутого табло", в котором строка $i + 1$ начинается на одну позицию, правее, чем строка i; например, сдвинутое табло формы $(7, 5, 4, 1)$ изображено на диаграмме

Picture: 3. p.90

Докажите, что число способов заполнить сдвинутое табло формы $(n_1, n_2, \ldots, n_m)$ числами $1, 2, \ldots, n = n_1 + n_2 + \cdots n_m$ так, чтобы числа во всех строках и столбцах располагались в возрастающем порядке, равно частному от деления $n!$ на произведение "длин обобщенных уголков"; на рисунке заштрихован обобщенный уголок длины 11, соответствующий клетке строки 1 и столбца 2. (Уголки в левой части массива, имеющей вид "перевернутой лестницы", имеют форму буквы U, повернутой на 90°, а не буквы L.) Итак, существует

$$17!/12 \cdot 11 \cdot 8 \cdot 7 \cdot 5 \cdot 4 \cdot 1 \cdot 9 \cdot 6 \cdot 5 \cdot 3 \cdot 2 \cdot 5 \cdot 4 \cdot 2 \cdot 1 \cdot 1$$

способов заполнить изображенную выше форму так, чтобы элементы во всех строках и столбцах располагались в возрастающем порядке.

>22. [BM30] (Д. Андрэ). Чему равно число A_n способов заполнить числами $\{1, 2, \ldots, n\}$ массив из n ячеек

Picture: p.91

так, чтобы во всех строках и столбцах они располагались в возрастающем порядке? Найдите производящую функцию $g(z) = \sum A_n z^n / n!$

23. [M39] Сколькими способами можно заполнить массив формы $(n_1, n_2, \ldots, n_m)$ элементами множества $\{1, 2, \ldots, N\}$, *если допускаются одинаковые элементы,* причем в строках элементы должны располагаться в неубывающем порядке, а в столбцах—в строго возрастающем? Например, простую форму из m строк $(1, 1, \ldots, 1)$ можно заполнить $\binom{N}{m}$ способами; форму из одной строки m можно заполнить $\binom{m+N-1}{m}$ способами; форму $(2, 2)$ можно заполнить $\frac{1}{3}\binom{N+1}{2}\binom{N}{2}$ способами.

24. [M28] Докажите, что

$$\sum_{\substack{q_1+\cdots+q_n=t \\ 0\leq q_1, \ldots, q_n \leq m}} \binom{m}{q_1} \ldots \binom{m}{q_n} \Delta(q_1, \ldots, q_n)^2 =$$

$$= n! \binom{nm - (n^2 - n)}{t - \frac{1}{2}(n^2 - n)} \binom{m}{n-1} \binom{m}{n-2} \ldots \binom{m}{0} \Delta(n-1, \ldots, 0)^2.$$

[*Указания:* докажите, что $\Delta(k_1 + n - 1,\ \ldots, k_n) = \Delta(m - k_n + n - 1,\ \ldots, m - k_1)$; разложите табло формы $n \times (m - n + 1)$ способом, аналогичным (38), и преобразуйте сумму, как при выводе тождества (36).]

25. [M20] Почему (42) является производящей функцией для инволюций?
26. [BM21] Вычислите $\int_{-\infty}^{\infty} x^t \exp(-2x^2/\sqrt{n})\, dx$ при неотрицательном целом t.
27. [M24] Пусть Q—табло Янга из элементов $\{1, 2,\ \ldots, n\}$, и пусть элемент t находится в строке r_i и столбце c_i. Мы говорим, что i "выше" j, если $r_i < r_j$.

    a) Докажите, что при $1 \le i < n$ элемент i выше $i + 1$ тогда и только тогда, когда $c_i \ge c_{i+1}$.
    b) Пусть Q такое, что (P, Q) соответствуют перестановке

$$\begin{pmatrix} 1 & 2 & \ldots & n \\ a_1 & a_2 & \ldots & a_n \end{pmatrix}$$

    Докажите, что i выше $i + 1$ тогда и только тогда, когда $a_i > a_{i+1}$. (Следовательно, можно найти число отрезков перестановки, зная только Q. Этот результат получен Шюценберже.)
    c) Докажите, что при $1 \le i < n$ элемент i выше $i + 1$ в Q тогда и только тогда, когда $i + 1$ выше i в Q^S.

28. [M47] Каково асимптотическое поведение средней длины максимальной возрастающей последовательности в случайной перестановке множества $\{1, 2,\ \ldots, n\}$? (Это средняя длина первой строки в соответствии из теоремы A. Обширные таблицы, вычисленные Р. М. Баером и П. Броком [*Math. Comp.*, **22** (1968), 385–410], в связи с тем, что они назвали "естественной сортировкой", позволяют предположить, что среднее l_n равно примерно $2\sqrt{n}$; Л. Шепп и Б. Логан доказали, что $\liminf_{n\to\infty} l_n/\sqrt{n} \ge 2$ (в печати).
29. [M50] Исследуйте трехмерные массивы, с тем чтобы понять, какие свойства двумерных табло можно обобщить.
30. [M42] (М. П. Шюценберже). Покажите, что операция перехода от P к P^S—частный случай операции, которую можно связать с любым конечным частично упорядоченным множеством, а не только с табло. Пометьте элементы частично упорядоченного множества целыми числами $\{1, 2,\ \ldots, n\}$ так, чтобы эта система меток была согласована с частичным упорядочением. Найдите двойственную систему меток, аналогичную (26), путем последовательного удаления меток 1, 2, ..., передвигая при этом другие метки способом, подобным алгоритму S, и помещая метки (1), (2), ... на освободившиеся места. Покажите, что эта операция, если ее многократно применять к двойственной системе меток с обратным отношением порядка для чисел, дает исходную систему меток; исследуйте другие свойства этой операции
31. [BM30] Пусть x_n—число способов разместить n взаимно неатакующих ладей на шахматной доске размера $n \times n$ таким образом, что расположение не меняется при отражении доски относительно одной из главных диагоналей и при повороте на 180°. Найдите асимптотическое поведение x_n.

5.2. Внутренняя сортировка

Начнем обсуждение хорошего "сортирования" с маленького эксперимента: как бы вы решили следующую задачу программирования?

> "Ячейки памяти $\mathtt{R}+1$, $\mathtt{R}+2$, $\mathtt{R}+3$, $\mathtt{R}+4$ и $\mathtt{R}+5$ содержат пять чисел. Напишите программу, которая переразмещает, если нужно, эти числа так, чтобы они расположились в возрастающем порядке."

(Если вы уже знакомы с какими-либо методами сортировки, постарайтесь, пожалуйста, на минуту забыть о них; вообразите, что вы решаете такую задачу впервые, не имея никаких предварительных знаний о том, как к ней подступиться.)

Прежде чем читать дальше, настоятельно просим вас найти хоть какое-нибудь решение этой задачи.

. . . . . .

Время, затраченное на решение приведенной выше задачи, окупится с лихвой, когда вы продолжите чтение этой главы. Возможно, ваше решение окажется одним из следующих:

A. *Сортировка вставками.* Элементы просматриваются по одному, и каждый новый элемент вставляется в подходящее место среди ранее упорядоченных элементов. (Именно таким способом игроки в бридж упорядочивают свои карты, беря по одной.)

B. *Обменная сортировка.* Если два элемента расположены не по порядку, то они меняются местами. Этот процесс повторяется до тех пор, пока элементы не будут упорядочены. C. *Сортировка посредством выбора.* Сначала выделяется наименьший (или. быть может, наибольший) элемент и каким-либо образом отделяется от остальных, затем выбирается наименьший (наибольший) из оставшихся и т. д.

D. *Сортировка подсчетом.* Каждый элемент сравнивается со всеми остальными; окончательное положение элемента определяется подсчетом числа меньших ключей.

E. *Специальная сортировка*, которая хороша для пяти элементов, указанных в задаче, но не поддается простому обобщению на. случай большего числа элементов. F. *Ленивое решение.* Вы не откликнулись на наше предложение и отказались решать задачу. Жаль, но теперь, когда вы прочли так много, ваш шанс упущен.

G. *Новая суперсортировка*, которая представляет собой определенное усовершенствование известных методов. (Пожалуйста, немедленно сообщите об этом автору.)

Если бы эта задача была сформулирована, скажем, для 1000 элементов, а не для 5, то вы могли бы открыть и некоторые более тонкие методы, о которых будет упомянуто ниже. В любом случае, приступая к новой задаче, разумно найти какую-нибудь очевидную процедуру, а затем попытаться улучшить ее. Случаи A, B и C приводят к важным классам методов сортировки, которые представляют собой усовершенствования сформулированных выше простых идей.

Было изобретено множество различных алгоритмов сортировки, и в этой книге рассматривается около 25 из них. Это пугающее количестве методов на самом деле лишь малая толика всех алгоритмов, придуманных до сих пор; многие, уже устаревшие методы мы вовсе не будем рассматривать или упомянем лишь вскользь. Почему же так много методов сортировки? Для программирования это частный случай вопроса: "Почему существует так много методов x?", где x пробегает множество всех задач. Ответ состоит в том, что каждый метод имеет свои преимущества и недостатки, поэтому он оказывается эффективнее других при некоторых конфигурациях данных и аппаратуры. К сожалению, неизвестен "наилучший" способ сортировки; существует *много* наилучших методов в зависимости от того, что сортируется, на какой машине, для какой цели. Говоря словами Редьярда Киплинга, "существует 9 и еще 60 способов сложить песню племени, и каждый из них в отдельности хорош".

Полезно изучить характеристики каждого метода сортировки, чтобы можно было производить разумный выбор для конкретных приложений. К счастью, задача изучения этих алгоритмов не столь уж громоздка, поскольку между ними существуют интересные взаимосвязи.

В начале этой главы мы ввели основную терминологию и обозначения, которые и будем использовать при изучении сортировки. Записи

$$R_1, R_2, \ldots, R_N \tag{1}$$

должны быть отсортированы в неубывающем порядке по своим ключам K_1, K_2, ..., K_N, по существу, путем нахождения перестановки $p(1)\ p(2)\ \ldots p(N)$, такой, что

$$K_{p(1)} \le K_{p(2)} \le \ldots \le K_{p(N)}. \tag{2}$$

В этом параграфе рассматривается *внутренняя сортировка*, когда число записей, подлежащих сортировке, достаточно мало, так что весь процесс можно провести в оперативной памяти ЭВМ.

В одних случаях может понадобиться физически переразместить записи в памяти так, чтобы их ключи были упорядочены; в других можно обойтись вспомогательной таблицей некоторого

Picture: Рис. 6. Сортировка таблицы адресов.

вида, которая определяет перестановку. Если записи и/или ключи занимают несколько слов памяти, то часто лучше составить новую таблицу адресов (ссылок), которые указывают на записи, и работать с этими адресами, не перемещая громоздкие записи. Такой метод называется *сортировкой таблицы адресов*

Picture: Рис. 7. Сортировка списка.

(рис 6.). Если ключи короткие, а сопутствующая информация в записях велика, то для повышения скорости ключи можно вынести в таблицу адресов; это называется *сортировкой ключей.* Другие схемы сортировки используют вспомогательное поле связи, которое включается в каждую запись. Связи обрабатываются таким образом, что в результате все записи оказываются связанными в линейный список, в котором каждая связь указывает на следующую по порядку запись. Это называется *сортировкой списка* (рис. 7).

После сортировки таблицы адресов или сортировки списка можно по желанию расположить записи в неубывающем порядке. Для этого имеется несколько способов, требующих дополнительной памяти для

хранения всего одной записи (см. упр. с 10 по 12); или же можно просто переместить записи в новую область памяти, если она может вместить все эти записи. Последний способ обычно вдвое быстрее первого, но требует почти в два раза больше памяти. Во многих приложениях вовсе не обязательно перемещать записи, так как поля связи, как правило, вполне приемлемы для операций с последовательной адресацией.

Все методы сортировки, которые мы исследуем "досконально", будут проиллюстрированы четырьмя способами: посредством

a) словесного описания алгоритма,
b) блок-схемы,
c) программы для машины `MIX`,
d) примера применения этого метода сортировки к заданному множеству чисел.

[В тех примерах, где это уместно, будет обрабатываться множество из 16 чисел, которые автор, пользуясь набором десятичных игральных костей, выбрал случайным образом 19 марта 1963 г.; ср. с упр. 3.1–1 (c).]

Из соображений удобства программы для машины `MIX` будут, как правило, написаны в предположении, что ключ числовой и что он помещается в одном слове; иногда мы даже будем ограничивать значения ключей так, чтобы они занимали лишь часть слова. Отношением порядка $<$ будет обычное арифметическое отношение порядка, а записи будут состоять из одного ключа, без сопутствующей информации. В этих предположениях программы получаются короче, проще для понимания, и не представляет труда распространить их на общий случай (например, применяя сортировку таблиц адресов). Вместе с `MIX`-программами приводится анализ времени выполнения соответствующего алгоритма сортировки.

Сортировка подсчетом. Чтобы проиллюстрировать способ, которым мы будем изучать методы внутренней сортировки, рассмотрим идею "подсчета", упомянутую в начале этого параграфа. Этот простой метод основан на том, что j-й ключ в окончательно упорядоченной последовательности превышает ровно $(j-1)$ из остальных ключей. Иначе говоря, если известно, что некоторый ключ превышает ровно 27 других, то после сортировки соответствующая запись должна занять 28-е место. Таким образом, идея состоит в том, чтобы сравнить попарно все ключи и подсчитать, сколько из них меньше каждого отдельного ключа.

Очевидный способ выполнить сравнения—

$$((\text{сравнить } K_j \text{ с } K_i) \text{ при } 1 \le j \le N) \text{ при } 1 \le i \le N,$$

но легко видеть, что более половины этих действий излишни, поскольку не нужно сравнивать ключ сам с собой и после сравнения K_a с K_b уже не надо сравнивать K_b с K_a. Поэтому достаточно

$$((\text{сравнить } K_j \text{ с } K_i) \text{ при } 1 \le j \le i) \text{ при } 1 < i \le N.$$

Итак, приходим к следующему алгоритму.

Алгоритм C. *(Сравнение и подсчет.)* Этот алгоритм сортирует записи $R_1, \ldots, R_N$ по ключам $K_1, \ldots, K_N$, используя для подсчета числа ключей, меньших данного, вспомогательную таблицу `COUNT`[1], ..., `COUNT`[N]. После завершения алгоритма величина `COUNT`[j] + 1 определяет окончательное положение записи R_j.

C1 [Сбросить счетчики.] Установить `COUNT`[1], ..., `COUNT`[N] равными нулю.
C2 [Цикл по i.] Выполнить шаг **C3** при $i = N, N-1, \ldots, 2$; затем завершить работу алгоритма.
C3 [Цикл по j.] Выполнить шаг **C4** при $j = i-1, i-2, \ldots, 1$.
C4 [Сравнить K_i, K_j.] Если $K_i < K_j$, то увеличить `COUNT`[j] на 1; в противном случае увеличить `COUNT`[i] на 1. ∎

Заметим, что в этом алгоритме записи не перемещаются. Он аналогичен сортировке таблицы адресов, поскольку таблица `COUNT` определяет конечное расположение записей; но эти методы несколько различаются, потому что `COUNT`[j] указывает то место, куда нужно переслать запись R_j, а не ту запись, которую надо переслать на место R_j. (Таким образом, таблица `COUNT` определяет перестановку, обратную $p(1) \ldots p(n)$; см. п. 5.1.1.)

В рассуждении, предшествующем этому алгоритму, мы не учитывали, что ключи могут быть равными. Это, вообще говоря, серьезное упущение, потому что если бы равным ключам соответствовали равные счетчики, то заключительное перемещение записей было бы довольно сложным. К счастью, как показано в упр. 2, алгоритм C дает верный результат независимо от числа равных ключей.

Picture: Рис. 8. Алгоритм C: сравнение и подсчет.

Таблица 1

Сортировка подсчетом (алгоритм C)

ключи:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
COUNT(нач.):	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COUNT($i = N$):	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	12
COUNT($i = N-1$):	0	0	0	0	2	0	2	0	0	0	0	0	0	0	13	12
COUNT($i = N-2$):	0	0	0	0	3	0	3	0	0	0	0	0	0	11	13	12
COUNT($i = N-3$):	0	0	0	0	4	0	4	0	1	0	0	0	9	11	13	12
COUNT($i = N-4$):	0	0	1	0	5	0	5	0	2	0	0	7	9	11	13	12
COUNT($i = N-5$):	1	0	2	0	6	1	6	1	3	1	2	7	9	11	13	12
. . . . .																
COUNT($i = 2$):	6	1	8	0	15	3	14	4	10	5	2	7	9	11	13	12

Программа C. *(Сравнение и подсчет.)* В следующей реализации алгоритма C для машины MIX предполагается, что запись R_j находится в ячейке INPUT+j, а COUNT[j]—в ячейке COUNT+j, где $1 \le j \le N$; rI1 $\equiv i$; rI2 $\equiv j$; rA $\equiv K_i \equiv R_i$; rX $\equiv$ COUNT[i].

```
START ENT1  N          1      C1 . Сбросить счетчики.
      STZ   COUNT,1    N      COUNT[i] ← 0.
      DEC1  1          N
      J1P   *-2        N      N ≥ i > 0.
      ENT1  N          1      C2. Цикл по i.
      JMP   1F         1
2H    LDA   INPUT,1    N-1
      LDX   COUNT,1    N-1
3H    CMPA  INPUT,2    A      C4. Сравнить K_i, K_j.
      JGE   4F         A      Переход, если K_i ≥ K_j.
      LD3   COUNT,2    B      COUNT[j]
      INC3  1          B      +1
      ST3   COUNT,2    B      → COUNT[j]
      JMP   5F         B
4H    INCX  1          A-B    COUNT[i] + 1 → COUNT[i].
5H    DEC2  1          A      C3.Цикл по j.
      J2P   3B         A
      STX   COUNT,1    N-1
      DEC1  1          N-1
1H    ENT2  -1,1       N      N ≥ i > j > 0.
      J2P   2B         N
```

Время работы этой программы равно $13N + 6A + 5B - 4$ единиц, где N—число записей, A—число способов выбрать 2 предмета из N, т. е. $\binom{N}{2} = (N^2 - N)/2$, а B— число пар индексов, таких, что $j < i$, и $K_j > K_i$. Таким образом, B—*число инверсий* перестановки $K_1, \ldots, K_N$; эта величина подробно анализировалась в п. 5.1.1, и было найдено [формулы (5.1.1–12,13)], что для неравных ключей, расположенных в случайном порядке,

$$B = \left(\min 0, \operatorname{ave} \frac{(N^2 - N}{4}, \max \frac{(N^2 - N)}{2}, \operatorname{dev} \frac{\sqrt{N(N-1)(N+2.5)}}{6} \right).$$

Следовательно, выполнение программы C занимает от $3N^2 + 10N - 4$ до $5.5N^2 + 7.5N - 4$ единиц времени, а среднее время работы находится посередине между этими двумя границами. Например, для данных табл. 1 имеем $N = 16$, $A = 120$, $B = 41$, значит, программа C отсортирует их за время $1129u$. Модификацию программы C, обладающую несколько иными временными характеристиками, см. в упр. 5.

Множитель N^2, которым определяется время работы, свидетельствует о том, что алгоритм C не дает эффективного способа сортировки, когда N велико; при удвоении числа записей время увеличивается в четыре раза. Поскольку этот метод требует сравнения всех пар ключей (K_i, K_j), то нет очевидного

способа исключить зависимость от N^2, тем не менее мы увидим дальше в этой главе, что, пользуясь "разделением и обменом", можно снизить порядок среднего времени работы до $N \log N$. Алгоритм C интересен для нас не эффективностью, а главным образом своей простотой; его описание служит примером того стиля, в котором будут описаны более сложные (и более эффективные) методы.

Существует другая разновидность сортировки посредством подсчета, которая *действительно* весьма важна с точки зрения эффективности; она применима в основном в том случае, когда имеется много равных ключей и все они попадают в диапазон $u \le K_j \le v$, где значение $(v-u)$ невелико. Эти предположения представляются весьма ограничительными, но на,самом деле мы увидим немало приложений этой идеи; например, если применить этот метод лишь к старшим цифрам ключей, а не ко всем ключам целиком, то файл окажется частично отсортированным, и будет уже сравнительно просто довести дело до конца.

Чтобы понять принцип, предположим, что все ключи лежат между 1 и 100. При первом просмотре файла можно подсчитать, сколько имеется ключей, равных 1, 2, ..., 100, а при втором просмотре можно уже располагать записи в соответствующих местах области вывода. В следующем алгоритме все это описано более подробно.

Picture: Рис. 9. Алгоритм D: распределяющий подсчет.

Алгоритм D. *(Распределяющий подсчет.)* Этот алгоритм в предположении, что все ключи—целые числа в диапазоне $u \le K_j \le v$ при $1 \le j \le N$, сортирует записи $R_1, \ldots, R_N$, используя вспомогательную таблицу `COUNT`$[u], \ldots,$ `COUNT`$[v]$. В конце работы алгоритма все записи в требуемом порядке переносятся в область вывода $S_1, \ldots, S_N$.

D1 [Сбросить счетчики.] Установить `COUNT`$[u], \ldots,$ `COUNT`$[v]$ равными нулю.

D2 [Цикл по j.] Выполнить шаг **D3** при $1 \le j \le N$, затем перейти к шагу **D4**.

D3 [Увеличить `COUNT`$[K_j]$.] Увеличить значение `COUNT`$[K_j]$ на 1.

D4 [Суммирование.] (К этому моменту значение `COUNT`$[i]$ есть число ключей, равных i.) Установить `COUNT`$[i] \leftarrow COUNT[i] + COUNT[i-l]$ при $i = u+l, u+2, \ldots, v$.

D5 [Цикл по j.] (К этому моменту значение `COUNT`$[i]$ есть число ключей, меньших или равных i, в частности `COUNT`$[v] = N$.) Выполнить шаг **D6** при $j = N, N-1, \ldots, 1$ и завершить работу алгоритма.

D6[Вывод R_j.] Установить $i \leftarrow$ `COUNT`$[K_j]$, $S_i \leftarrow R_i$, и `COUNT`$[K_j] \leftarrow i-1$. ∎

Пример применения этого алгоритма приведен в упр. 6; программу для машины `MIX` можно найти в упр. 9. При сформулированных выше условиях это очень быстрая процедура .сортировки.

Сортировка посредством сравнения и подсчета, как, в алгоритме C, впервые упоминается в работе Э. Х. Фрэнда [*JACM*,**3** (1965), 152],. хотя он и не заявил о ней как о своем собственном изобретении. Распределяющий подсчет, как в алгоритме D, впервые разработан Х. Сьювордом в 1954 г. и использовался при поразрядной сортировке, которую мы обсудим позже (см. п. 5.2.5); этот метод также -был опубликован под названием "Mathsort" в работе W. Feurzig, *CACM*, **3** (I960), 601.

Упражнения

1. [15] Будет ли работать алгоритм C, если в шаге C2 значение C будет изменяться от 2 до N, а не от N до 2? Что произойдет, если в шаге C3 значение j будет изменяться от 1 до $i-1$?
2. [21] Покажите, что алгоритм C работает правильно и при наличии одинаковых ключей. Если $K_j = K_i$ и $j < i$, то где окажется в конце концов R_j—до или после R_i?
3. [21] Будет ли алгоритм C работать правильно, если в шаге C4 заменить проверку "$K_i < K_j$" на "$K_i \le K_j$"?
4. [16] Напишите `MIX`-программу, которая завершит сортировку, начатую программой C; ваша программа должна поместить ключи в ячейки `OUTPUT` $+1, \ldots,$ `OUTPUT` $+N$ в возрастающем порядке. Сколько времени затратит на это ваша программа?
5. [22] Улучшится ли программа C в результате следующих изменений?

Ввести новую строку 8а:	`INCX 0, 2`
Изменить строку 10:	`JGE 5F`
Изменить строку 14:	`DECX 1`
Исключить строку 15.	

6. [18] Промоделируйте вручную работу алгоритма D, показывая промежуточные результаты, получающиеся при сортировке 16 записей

5T, 0C, 5U, 00, 9, 1N, 8S, 2R, 6A, 4A, 1G, 5L, 6T, 61, 70, 7N.

Здесь цифры—это ключи, а буквы—сопутствующая информация в записях.

7. [13] Является ли алгоритм D алгоритмом "устойчивой" сортировки?
7. [15] Будет ли алгоритм D работать правильно, если в шаге D5 значение j будет изменяться от 1 до N, а не от N до 1?
9. [23] Напишите `MIX`-программу для алгоритма D, аналогичную программе C и упр. 4. Выразите время работы вашей программы в виде функции от N и $(v-u)$.
10. [25] Предложите эффективный алгоритм, который бы заменял N величин $(R_1, \ldots, R_N)$ соответственно на $(R_{p(1)}, \ldots, R_{p(1)})$), если даны значения $R_1, \ldots, R_N$ и перестановка $p(1)\ldots p(N)$ множества $\{1, \ldots, N\}$. Постарайтесь не использовать излишнего пространства памяти. (Эта задача возникает, когда требуется переразместить в памяти записи после сортировки таблицы адресов, не расходуя память на $2N$ записей.)
11. [M27] Напишите `MIX`-программу для алгоритма из упр. 10 и проанализируйте ее. эффективность.

>12. [25] Предложите эффективный алгоритм переразмещения в памяти записей $R_1, \ldots, R_N$ в отсортированном порядке после завершения сортировки списка (рис. 7). Постарайтесь не использовать излишнего пространства памяти.

>13. [27] Алгоритму D требуется пространство для $2N$ записей $R_1, \ldots R_N$ и $S_1, \ldots, S_N$. Покажите, что можно обойтись пространством для N записей $R_1, \ldots, R_N$, если вместо шагов D5 и D6 подставить новую упорядочивающую процедуру. (Таким образом, задача состоит в том, чтобы разработать алгоритм, который бы переразмещал записи $R_1, \ldots, R_N$. основываясь на значениях

$$\texttt{COUNT}[u], \ldots \texttt{COUNT}[v]$$

после выполнения шага D4, не используя дополнительной памяти; это, по существу, обобщение задачи, рассмотренной в упр. 10.)

5.2.1. Сортировка вставками

Одно из важных семейств-методов сортировки основано на упомянутом в начале § 5.2 способе, которым пользуются игроки в бридж; предполагается, что перед рассмотрением записи R_j предыдущие записи R_1, $\ldots$, R_{j-1} уже упорядочены, и R_j вставляется в соответствующее место. На основе этой схемы возможны несколько любопытных вариаций.

Простые вставки. Простейшая сортировка вставками относится к наиболее очевидным. Пусть $1 < j \le N$ и записи $R_1, \ldots, R_{j-1}$ уже размещены так, что

$$K_1 \le K_2 \le \ldots \le K_{j-1}.$$

(Напомним, что в этой главе через K_j обозначается ключ записи R_j.) Будем сравнивать по очереди K_j с K_{j-1}, K_{j-2}, $\ldots$до тех пор, пока не обнаружим, что запись R_j следует вставить между R_i и R_{i+1}; тогда подвинем записи $R_{i+1}, \ldots, R_{j-1}$ на одно место вверх и поместим новую запись в позицию $i+1$. Удобно совмещать операции сравнения и перемещения, перемежая их друг с другом, как показано в следующем алгоритме; поскольку запись R_j как бы "проникает на положенный ей уровень", этот способ часто называют *просеиванием*, или *погружением*.

Алгоритм S. *(Сортировка простыми вставками.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$.

S1 [Цикл. по j.] Выполнить шаги от **S2** до **S5** при $j = 2, 3, \ldots, N$ и после этого завершить алгоритм.

Picture: Рис. 10. Алгоритм S: простые вставки.

S2 [Установить i, K, R.] Установить $i \leftarrow j-1$, $K \leftarrow K_j$, $R \leftarrow R_j$.(В последующих шагах мы попытаемся вставить запись R в нужное место, сравнивая K с K_i при убывающих значениях i.)

S3 [Сравнить K с K_i.] Если $K \ge K_i$, то перейти к шагу **S5**. (Мы нашли искомое место для записи R.)

S4 [Переместить R_i, уменьшить i.] Установить $R_{i+1} \leftarrow R_i$, $i \leftarrow i-1$. Если $i > 0$, то вернуться к шагу **S3**. (Если $i = 0$, то K—наименьший из рассмотренных до сих пор ключей, а, значит, запись R должна занять первую позицию.)

S5 [R на место R_{i+1}.] Установить $R_{i+1} \leftarrow R$.

В табл. 1 показано применение алгоритма S к шестнадцати числам, взятым нами для примеров. Этот метод чрезвычайно просто реализуется на вычислительной машине; фактически следующая `MIX`-программа—самая короткая из всех приемлемых программ сортировки в этой книге.

Программа S. *(Сортировка простыми вставками)* . Записи, которые надо отсортировать, находятся в ячейках `INPUT` $+ 1$, $\ldots$, `INPUT` $+ N$; они сортируются в той же области по ключу, занимающему одно слово целиком. Здесь `rI1` $\equiv j - N$; `rI2` $\equiv i$, `rA` $\equiv R \equiv K$; предполагается, что $N \ge 2$.

```
START  ENT1  2-N              1          S1.Цикл по j. j ← 2.
2H     LDA   INPUT+N,1        N − 1      S2. Установить i, K, R
       ENT2  N-1,1            N − 1      i ← j − 1.
3H     CMPA  INPUT,2    B + N − 1 − A    S3. Сравнить K, K_i
       JGE   5F         B + N − 1 − A    К S5, если K ≥ K_i.
4H     LDX   INPUT,2          B          S4. Переместить R_i, уменьшить i
       STX   INPUT+1,2        B          R_{i+1} ← R_i.
       DEC2  1                B          i ← i − 1.
       J2P   3B               B          К S3, если i > 0.
5H     STA   INPUT+1,2        N − 1      S5. R на место R_i + 1.
       INC1  1                N − 1
       J1NP  2B               N − 1      2 ≤ j ≤ N.
```

Время работы этой программы равно $9B + 10N - 3A - 9$ единиц, где N—число сортируемых записей, A—число случаев, когда в шаге S4 значение i убывает до 0, а B—число перемещений. Ясно, что A равно числу случаев, когда $K_j < (K_1, \ldots, K_{j-1})$ при $1 < j \le N$, т. е. это число левосторонних минимумов—величина, которая была тщательно исследована в п. 1.2.10. Немного подумав, нетрудно понять, что B—тоже известная величина: число перемещений при фиксированном j равно числу инверсий ключа K_j, так что B равно числу инверсий перестановки K_1, K_2, ..., K_N. Следовательно, согласно формулам (1.2.10–16) и (5.1.1–12, 13),

$$A = (\min 0, \operatorname{ave} H_N - 1, \max N - 1, \operatorname{dev} \sqrt{H_n - H_n^{(2)}});$$
$$B = (\min 0, \operatorname{ave}(N^2 - N)/4, \max(N^2 - N)/2, \operatorname{dev} \sqrt{N(N-1)(N+2.5)/6}),$$

а среднее время работы программы S в предположении, что ключи различны и расположены в случайном порядке, равно $(2.25N^2 + 7.75N - 3H_N - 6)u$. В упр. 33 показано, как можно чуть-чуть уменьшить это время.

Приведенные в качестве примера в табл. 1 данные содержат 16 элементов; имеются два левосторонних минимума, 087 и 061, и, как мы видели в предыдущем пункте, 41 инверсия. Следовательно, $N = 16$, $A = 2$, $B = 41$, а общее время сортировки равно $514u$.

Бинарные вставки и двухпутевые вставки. Когда при сортировке простыми вставками обрабатывается j-я запись, ее ключ сравнивается в среднем примерно с $j/2$ ранее отсортированными ключами;

Таблица 1

Пример применения простых вставок

```
^503  :087
 087  503^ :512
^087  503  512  :061
 061  087  503  512^ :908
 061  087^ 503  512  908  :170
 061  087  170  503  512^ 908  :897
 . . . . . .

 061  087  154  170  275  426  503  509  512  612  653  677^ 765  897  908  :703
 061  087  154  170  275  426  503  509  512  612  653  677  703  765  897  908
```

поэтому общее число сравнений равно приблизительно $(1 + 2 + \cdots + N)/2 \approx N^2/4$, а это очень много, даже если N умеренно велико. В п. 6.2.1 мы изучим методы "бинарного поиска", которые указывают, куда вставлять j-й элемент после приблизительно $\log_2 j$ соответствующим образом выбранных сравнений. Например, если вставляется 64-я запись, можно сначала сравнить ключ K_{64} с K_{32}; затем, если он меньше, сравниваем его с K_{16}, если больше—с K_{48} и т. д., так что место для R_{64} будет найдено после всего лишь шести сравнений. Общее число сравнений для N вставляемых элементов равно приблизительно $N \log_2 N$, что существенно лучше, чем $N^2/4$; в п. 6.2.1 показано, что соответствующая программа не обязательно намного сложнее, чем программа для простых вставок. Этот метод называется *бинарными вставками*. Он упоминался Джоном Мочли еще в 1946 г., в первой публикации по машинной сортировке.

Неприятность состоит в том, что бинарные вставки решают задачу только наполовину: после того, как мы нашли, куда вставлять запись R_j, все равно нужно подвинуть примерно $j/2$ ранее отсортированных записей, чтобы освободить место для R_j так что общее время работы остается, по существу,

пропорциональным N_2. Некоторые вычислительные машины (например, IBM 705) имеют встроенные инструкции "переброски", выполняющие операции перемещения с большой скоростью, но с ростом N зависимость от N^2 в конце концов начинает преобладать, Например, анализ, проведенный Х. Нэгдером [*CACM*, **3** (1960), 618–620], показывает, что не следует рекомендовать бинарные вставки при сортировке более $N = 128$ записей на машине IBM 705, если каждая запись состоит из 80 литер; аналогичный анализ применим и к другим машинам.

Таблица 2

Двухпутевые вставки

				-503			
			087	503-			
			-087	503	512		
		061	087	503	512-		
		061	087-	503	512	908	
	061	087	170	503	512	908	
	061	087	170-	503	512	897	908
061	087	170	276	503	512	897	908

Разумеется, изобретательный программист может придумать какие-нибудь способы, позволяющие сократить число необходимых перемещений; первый такой прием, предложенный в начале 50-х годов, проиллюстрирован в табл. 2. Здесь первый - элемент помещается в середину области вывода, и место для последующих элементов освобождается при помощи сдвигов влево или вправо, туда, куда удобнее. Таким образом удается сэкономить примерно половину времени работы по сравнению с простыми вставками за счет некоторого усложнения программы. Можно применять этот метод, используя не больше памяти, чем требуется для N записей (см. упр. 6), но мы не станем дольше задерживаться на таких "двухпутевых" вставках, так как были разработаны гораздо более интересные методы.

Метод Шелла. Для алгоритма сортировки, который каждый раз перемещает запись только на одну позицию, среднее время работы будет в лучшем случае пропорционально N^2, потому что в процессе сортировки каждая запись должна пройти в среднем через $N/3$ позиций (см. упр. 7). Поэтому, если мы хотим получить метод, существенно превосходящий по скорости простые вставки, то необходим некоторый механизм, с помощью которого записи могли бы перемещаться большими скачками, а не короткими шажками.

Такой метод предложен в 1959 г. Дональдом Л. Шеллом [*CACM*, **2** (July, 1959), 30–32]; мы будем называть его *сортировкой с убывающим шагом*. В табл. 3 проиллюстрирована общая идея, лежащая в основе этого метода. Сначала делим 16 записей на 8 групп по две записи в каждой группе: (R_1, R_9), (R_2, R_{10}), ..., (R_8, R_{16}). В результате сортировки каждой группы записей по отдельности приходим ко второй строке табл. 3,

Picture: Таблица 3

это называется "первым просмотром". Заметим, что элементы 154 и 512 поменялись местами, а 908 и 897 переместились вправо. Разделим теперь записи на четыре группы по четыре в каждой: (R_1, R_5, R_9, R_{13}), ..., $(R_4, R_8, R_{12}, R_{16})$—и опять отсортируем каждую группу в отдельности; этот второй просмотр приводит к третьей строке таблицы. При третьем просмотре сортируются две группы по восемь записей; процесс завершается четвертым просмотром, во время которого сортируются все 16 записей. В каждом из этих промежуточных процессов сортировки участвуют либо сравнительно короткие файлы, либо уже сравнительно хорошо упорядоченные файлы, поэтому на каждом этапе можно пользоваться простыми вставками; записи довольно быстро достигают своего конечного положения.

Последовательность шагов 8, 4, 2, 1 не следует считать незыблемой, можно пользоваться *любой* последовательностью h_t, h_{t-1}, ..., h_1, в которой последний шаг h_1 равен 1. Например, в табл. 4 будет показана сортировка тех же данных с шагами 7, 5, 3, 1. Одни последовательности оказываются гораздо лучше других; мы обсудим выбор последовательностей шагов позже.

Алгоритм D. *(Сортировка с убывающим шагом.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$. Для управления процессом сортировки используется вспомогательная последовательность шагов h_t, h_{t-1}, ..., h_1, где $h_1 = l$; правильно выбрав эти приращения, можно значительно сократить время сортировки. При $t = 1$ этот алгоритм сводится к алгоритму S.

D1 [Цикл по s.] Выполнить шаг **D2** при $s = t, t-1, \ldots, 1$, после чего завершить работу алгоритма.

D2 [Цикл по j.] Установить $h \leftarrow h_s$ и выполнить шаги **D3**, ... **D6** при $h < j \leq N$. (Для сортировки элементов, отстоящих друг от друга на h позиций, воспользуемся простыми вставками и в результате получим $K_i \leq K_{i+h}$ при $1 \leq i \leq N-h$. Шаги **D3**, ..., **D6**, по существу, такие же, как соответственно S2, ..., S5 в алгоритме S.)

D3 [Установить i, K, R.] Установить $i \leftarrow j-h$, $K \leftarrow K_j$, $R \leftarrow R_j$.

D4 [Сравнить K, K_i.] Если $K \geq K_i$, то перейти к шагу **D6**.

D5 [Переместить R_i, уменьшить i.] Установить $R_{i+h} \leftarrow R_i$, затем $i \leftarrow i-h$. Если $i > 0$, то возвратиться к шагу **D4**.

D6 [R на место R_{i+h}.] Установить $R_{i+h} \leftarrow R$. ∎

Соответствующая MIX-программа не намного длиннее, чем наша программа для простых вставок. Строки 08–19 этой программы перенесены из программы S в более общий контекст алгоритма D.

Программа D. *(Сортировка с убывающим шагом.)* Предполагается, что шаги сортировки хранятся во вспомогательной таблице и h_s находится в ячейке H + s; все шаги сортировки меньше N. Содержимое регистров: rI1 $\equiv j-N$; rI2 $\equiv i$; rA $\equiv R \equiv K$; rI3 $\equiv s$; rI4 $\equiv h$. Заметим, что эта программа сама себя изменяет. Это сделано для того, чтобы добиться более эффективного выполнения внутреннего цикла.

```
START  ENT3  T              1              D1. Цикл по s. s ← t.
1H     LD4   H,3            T              D2. Цикл по j.h ← h_s.
       ENT1  INPUT,4        T              Изменить адреса в трех
       ST1   6F(0:2)        T              инструкциях в
       ST1   7F(0:2)        T              основном цикле.
       ENN1  -N, 4          T              rI1 ← N − h.
       ST1   4F(0:2)        T
       ENT1  1-N, 4         T              j ← h + 1.
2H     LDA   INPUT+N,1      NT − S         D3. Установить i, K, R.
4H     ENT2  N-H,1          NT − S         i ← j − h. (Изменяемая инструкция)
5H     CMPA  INPUT,2        B + NT − S − A  D4. Сравнить K, K_i.
       JOE   7F             B + NT − S − A  К шагу D6, если K ≥ K_i.
       LDX   INPUT,2        B              D5. Переместить R_i, уменьшить i.
6H     STX   INPUT+H,2      B              R_{i+h} ← R_i. (Изменяемая инструкция)
       DEC2  0,4            B              i ← i − h.
       J2P   5B             B              К шагу D4, если i > 0.
7H     STA   INPUT+H,2      NT − S         D6. R на место R_{i+h}. (Изменяемая инструкция)
8H     INC1  1              NT − S         j ← j + 1.
       J1NP  2B             NT − S         К D3, если j ≤ N.
       DEC3  1              T
       J3P   1B             T              t ≥ s ≥ 1.
```

∎

***Анализ метода Шелла.** Чтобы выбрать хорошую последовательность шагов сортировки для алгоритма D, нужно проанализировать время работы как функцию от этих шагов. Это приводит к очень красивым, но еще не до конца решенным математическим задачам; никому до сих пор не удалось найти наилучшую возможную последовательность шагов для больших N. Тем не менее известно довольно много интересных фактов о поведении сортировки методом Шелла с убывающим шагом, и мы здесь их кратко изложим; подробности можно найти в приведенных ниже упражнениях. [Читателям, не имеющим склонности к математике, лучше пропустить следующие несколько страниц, до формул (8).]

Счетчики частот выполнения в программе D показывают, что на время выполнения влияют пять факторов: размер файла N, число просмотров (т.е. число шагов) $T = t$, сумма шагов

$$S = h_1 + \cdots + h_t,$$

число сравнений $B + NT - S - A$ и число перемещений B. Как и при анализе программы S, здесь A равно числу левосторонних минимумов, встречающихся при промежуточных операциях сортировки, а B равно числу инверсий в подфайлах. Основным фактором, от которого зависит время работы, является величина B, поэтому на нее мы и обратим главным образом свое внимание. При анализе будет предполагаться, что ключи различны и первоначально расположены в случайном порядке.

Назовем операцию шага D2 "h-сортировкой". Тогда сортировка методом Шелла состоит из h_t-сортировки, за которой следует h_{t-1}-сортировка, ..., за которой следует h_1-сортировка. Файл, в котором $K_i \leq K_{i+h}$ при $1 \leq i \leq N-h$, будем называть h-упорядоченным.

Рассмотрим сначала простейшее обобщение простых вставок, когда имеются всего два шага $h_2 = 2$ и $h_1 = 1$. Во время второго просмотра имеем 2-упорядоченную последовательность ключей $K_1\ K_2 \ldots K_N$. Легко видеть, что число перестановок $a_1\ a_2\ \ldots a_n$ множества $\{1, 2, \ldots, \text{п}\}$, таких, что $a_i \le a_{i+2}$ при $l \le i \le n-2$, равно

$$\binom{n}{\lfloor n/2 \rfloor},$$

так как существует всего одна 2-упорядоченная перестановка для каждого выбора $\lfloor n/2 \rfloor$ элементов, расположенных в четных позициях $a_2 a_4, \ldots$, тогда остальные $\lceil n/2 \rceil$ элементов попадают в позиции с нечетными номерами. После 2-сортировки случайного файла с одинаковой вероятностью может получиться любая 2-упорядоченная перестановка. Каково среднее число инверсий во всех таких перестановках?

Picture: Рис. 11. Соответствие между 2-упорядочением и путями на решетке. Курсивом набраны веса, соответствующие числу инверсий в 2-упорядоченной перестановке.

Пусть A_n—суммарное число инверсий во всех 2-упорядоченных перестановках множества $\{1, 2, \ldots, n\}$. Ясно, что $A_1 = 0$, $A_2 = 1$, $A_3 = 2$, а из рассмотрения шести случаев

1 3 2 4 1 2 3 4 1 2 4 3 2 1 3 4 2 1 4 3 3 1 4 2

находим, что $A_4 = 1+0+1+1+2+3 = 8$. Чтобы исследовать A_n в общем случае, рассмотрим решетчатую диаграмму на рис. 11 для $n = 15$. В такой диаграмме 2-упорядоченную перестановку можно представить в виде пути из верхней левой угловой точки $(0, 0)$ в нижнюю правую угловую точку $(\lceil n/2 \rceil, \lfloor n/2 \rfloor)$, если делать k-й шаг пути вправо или вниз в соответствии с тем, находится ли k в четной или нечетной позиции перестановки. Этим правилом определяется взаимно однозначное соответствие между 2-упорядоченными перестановками и n-шаговыми путями из одного угла решетчатой диаграммы в другой. Например, изображенный на рисунке путь соответствует перестановке

$$2\ 1\ 3\ 4\ 6\ 5\ 7\ 10\ 8\ 11\ 9\ 12\ 14\ 13\ 15. \tag{1}$$

Далее, вертикальным отрезкам пути можно приписать "веса", как показано на диаграмме; отрезку, ведущему из точки (i, j) в точку $(i+1, j)$ приписывается вес $|i-j|$. Немного подумав, читатель убедится в том, что сумма этих весов вдоль каждого пути равна числу инверсий в соответствующей перестановке (см. упр. 12). Так, например, перестановка (1) содержит $1+0+1+0+1+2+1 = 6$ инверсий.

Если $a \le a'$ и $b \le b'$, то число допустимых путей из (a, b) в (a', b') равно числу способов перемешать $a'-a$ вертикальных отрезков с $b'-b$ горизонтальными, а именно

$$\binom{a'-a+b'-b}{a'-a}.$$

Следовательно, число перестановок, для которых соответствующие пути проходят через вертикальный отрезок из (i, j) в $(i+1, j)$, равно

$$\binom{i+j}{i}\binom{n-i-j-1}{\lfloor n/2 \rfloor - j}.$$

Умножая это значение на вес данного отрезка и суммируя по всем отрезкам, получаем

$$\begin{aligned} A_{2n} &= \sum_{\substack{0 \le i \le n \\ 0 \le j \le n}} |i-j| \binom{i+j}{i}\binom{2n-i-j-1}{n-j}; \\ A_{2n+1} &= \sum_{\substack{0 \le i \le n \\ 0 \le j \le n}} |i-j| \binom{i+j}{i}\binom{2n-i-j}{n-j}; \end{aligned} \tag{2}$$

Знаки абсолютной величины в этих суммах несколько усложняют вычисления, но в упр. 14 показано, что величина A_n имеет удивительно простой вид: $\lfloor n/2 \rfloor 2^{n-2}$. Следовательно, среднее число инверсий в случайной 2-упорядоченной перестановке равно

$$\lfloor n/2 \rfloor 2^{n-2} / \binom{n}{\lfloor n/2 \rfloor}.$$

По формуле Стирлинга эта величина асимптотически приближается к $\sqrt{\pi/128}n^{3/2} \approx 0.15n^{3/2}$. Как легко видеть, максимальное число инверсий равно

$$\binom{\lfloor n/2 \rfloor + 1}{2} \approx \frac{1}{8}n^2.$$

Полезно исследовать распределение числа инверсий более тщательно, рассмотрев производящие функции

$$\begin{aligned} h_1(z) = 1, \quad h_2(z) = 1 + z, \\ h_3(z) = 1 + 2z, \quad h_4(z) = 1 + 3z + z^2 + z^3, \ldots, \end{aligned} \tag{3}$$

как в упр. 15. Таким образом, найдем, что стандартное отклонение тоже пропорционально $n^{3/2}$, так что число инверсий не слишком устойчиво распределено около своего среднего значения. Рассмотрим теперь общий двухпроходный случай алгоритма D, когда шаги сортировки равны h и 1.

Теорема Н. *Среднее число инверсий в h-упорядоченной перестановке множества $\{1, 2, \ldots, n\}$ равно*

$$f(n,h) = \frac{2^{2q-1}q!q!}{(2q+1)!}\left(\binom{h}{2}q(q+1) + \binom{r}{2}(q+1) - \frac{1}{2}\binom{h-r}{2}q\right), \tag{4}$$

где $q = \lfloor n/h \rfloor$, $r = n \bmod h$. Эта теорема принадлежит Дугласу Ханту [Bachelor's thesis, Princeton University (April, 1967)]. Заметим, что формула справедлива и при $h \geq n : f(n,h) = \frac{1}{2}\binom{n}{2}$.

Доказательство В h-упорядоченной перестановке содержится r упорядоченных подпоследовательностей длины $q+1$ и $h-r$ подпоследовательностей длины q. Каждая инверсия образуется из элементов двух различных подпоследовательностей, а каждая пара различных упорядоченных подпоследовательностей в случайной h-упорядоченной перестановке определяет случайную 2-упорядоченную перестановку. Поэтому среднее число инверсий равно сумме средних значений числа инверсий во всех парах различных подпоследовательностей, а именно

$$\binom{r}{2}\frac{A_{2q+2}}{\binom{2q+2}{q+1}} + r(h-r)\frac{A_{2q+1}}{\binom{2q+1}{q}} + \binom{h-r}{2}\frac{A_{2q}}{\binom{2q}{q}} = f(n,h). \blacksquare$$

Следствие. *Если последовательность приращений удовлетворяет условию*

$$h_{s+1} \bmod h_s = 0 \qquad \text{при } t > s \geq 1, \tag{5}$$

то среднее число операций перемещения в алгоритме D равно

$$\sum_{t \geq s \geq 1} (r_s f(q_s + 1, h_{s+1}/h_s) + (h_s - r_s) f(q_s, h_{s+1}/h_s)), \tag{6}$$

где $r_s = N \bmod h_s$, $q_s = \lfloor N/h_s \rfloor$, $h_{t+1} = N$, а функция f определяется формулой (4).

Доказательство Процесс h_s-сортировки состоит из сортировки простыми вставками r_s (h_{s+1}/h_s)-упорядоченных подфайлов длины $q_s + 1$ и $(h_s - r_s)$ таких подфайлов длины q_s. Поскольку мы предполагаем, что исходная перестановка случайна и все ее элементы различны, то из условий делимости следует, что каждый из подфайлов—"случайная" (h_{s+1}/h_s)-упорядоченная перестановка в том смысле, что все (h_{s+1}/h_s)-упорядоченные перестановки равновероятны. $\blacksquare$

Условие (5) этого следствия всегда выполняется для *двухпроходной* сортировки методом Шелла, когда шаги равны соответственно h и 1. Пусть $q = \lfloor N/h \rfloor$, а $r = N \bmod h$, тогда среднее значение величины B в программе D равно

$$rf(q+1, N) + (h-r)f(q, N) + f(N, h) = \frac{r}{2}\binom{q+1}{2} + \frac{h-r}{2}\binom{q}{2} + f(N, h).$$

В первом приближении функция $f(n,h)$ равна $(\sqrt{\pi}/8)n^{3/2}h^{1/2}$; ср. с гладкой кривой на рис. 12. Следовательно, время работы

Picture: Рис. 12. Среднее число инверсий $f(n,h)$ в h-упорядоченном файле из n элементов для случая $n = 64$.

двухпроходной программы D примерно пропорционально $2N^2/h + \sqrt{\pi N^3 h}$. Поэтому наилучшее значение h равно приблизительно $\sqrt[3]{16N/\pi} \approx 1.72\sqrt[3]{N}$; при таком выборе h среднее время работы пропорционально $N^{5/3}$.

Таким образом, даже применяя метод Шелла с всего двумя шагами, можно существенно сократить время по сравнению с простыми вставками, с $O(N^2)$ до $O(N^{1.667})$. Ясно, что можно добиться лучших результатов, если использовать больше шагов. В упр. 18 обсуждается оптимальный выбор $h_t, \ldots, h_1$ при фиксированном t в случае, когда значения h ограничены условием делимости; время работы при больших N сокращается до $O(N^{1.5+\varepsilon/2}$ где $\varepsilon = 1/(2^t - 1)$. Если мы пользуемся приведенными выше формулами, барьер $N^{1.5}$ преодолеть невозможно, потому что при последнем просмотре в сумму инверсий всегда вносится вклад

$$f(N, h_2) \approx (\sqrt{\pi}/8)N^{3/2}h_2^{1/2}.$$

Но интуиция подсказывает, что, если шаги $h_t, \ldots, h_1$ *не будут* удовлетворять условию делимости (5), можно достичь большего. Например, при последовательном выполнении 8-, 4- и 2-сортировок невозможно взаимодействие между ключами в четных и нечетных позициях; поэтому на долю заключительной 1-сортировки останется $O(N^{3/2})$ инверсий. В то же время при последовательном выполнении 7-, 5- и 3-сортировок файл перетряхивается так, что при заключительной 1-сортировке не может встретиться более $2N$ инверсий! (См. упр. 26.) На самом деле наблюдается удивительное явление.

Теорема K. *После h-сортировки k-упорядоченный файл остается k-упорядоченным.*

Таким образом, файл, который был сначала 7-отсортирован, а потом 5-отсортирован, становится одновременно. 7- и 5-упорядоченным. А если мы подвергнем его 3-сортировке, то полученный файл будет 7-, 5- и 3-упорядочен. Примеры проявления этого замечательного свойства можно найти в табл. 4.

Picture: Таблица 4 Сортировка с убывающим шагом (7, 5, 3, 1)

Доказательство В упр. 20 показано, что эта теорема вытекает из следующей леммы:

Лемма L. *Пусть m, n, r—неотрицательные целые числа, а $x_1, \ldots, x_{m+r}$ и $y_1, \ldots, y_{n+r}$—произвольные последовательности чисел, такие, что*

$$y_1 \le x_{m+1}, y_2 \le x_{m+2}, \ldots, y_r \le x_{m+r}. \tag{7}$$

Если последовательности x и y отсортировать независимо, так что $x_1 \le \ldots \le x_{m+r}$ и $y_1 \le \ldots \le y_{n+r}$, то соотношения (7) останутся в силе.

Доказательство Известно, что все, кроме, быть может, m элементов последовательности x, превосходят (т. е. больше или равны) некоторые элементы последовательности y, причем различные элементы x превосходят различные элементы y. Пусть $1 \le j \le r$. Так как после сортировки элемент x_{m+j} превосходит $m + j$ элементов x, то он превосходит по крайней мере j элементов y, а значит, он превосходит j *наименьших* элементов y. Следовательно, после сортировки имеем $x_{m+j} \ge y_j$. ∎ ∎

Из теоремы K видно, что при сортировке желательно пользоваться взаимно простыми значениями шагов, однако непосредственно из нее не следуют точные оценки числа перемещений, выполняемых алгоритмом D. Так как число перестановок множества $\{1, 2, \ldots, n\}$, одновременно h- и k-упорядоченных, не всегда является делителем $n!$, то понятно, что теорема K объясняет далеко не все; в результате k- и h-сортировок некоторые k- и h-упорядоченные файлы получаются чаще других. Более того, не существует очевидного способа отыскать "наихудший случай" для алгоритма D при произвольной последовательности шагов сортировки $h_t, \ldots, h_1$. Поэтому до сих пор все попытки анализа этого алгоритма в общем случае были тщетны; по существу, все, что нам известно,—это приближенное асимптотическое поведение максимального времени работы в некоторых случаях.

Теорема P. *Если $h_s = 2^s - 1$ при $1 \le s \le t = \lfloor \log_2 N \rfloor$, то время работы алгоритма D есть $O(N^{3/2})$*

Доказательство Достаточно найти оценку B_s числа перемещений при s-м просмотре, такую, чтобы $B_t + \cdots + B_1 = O(N^{3/2})$. Для первых $t/2$ просмотров при $t \le s \ge t/2$ можно воспользоваться очевидной оценкой $B_s = O(h_s(N/h_s)^2)$, а для последующих просмотров можно применить результат упр. 23: $B_s = O(N h_{s+2} h_{s+1}/h_s)$. Следовательно, $B_t + \cdots + B_1 = O(N(2 + 2^2 + \cdots + 2^{t/2} + +2^{t/2} + \cdots + 2)) = O(N^{3/2})$. ∎

Эта теорема принадлежит А. А. Папернову и Г. В. Стасевичу [*Проблемы передачи информации*, 1,3 (1965), 81–98]. Она дает верхнюю оценку *максимального* времени работы алгоритма, а не

Таблица 5

Анализ алгоритма D при $N = 8$

Шаги			A_{ave}	B_{ave}	S	T	время машины MIX
		1	1.718	14.000	1	1	$204.85u$
	2	1	2.667	9.657	3	2	$235.91u$
	3	1	2.917	9.100	4	2	$220.16u$
	4	1	3.083	10.000	5	2	$217.75u$
	5	1	2.601	10.000	6	2	$210.00u$
	6	1	2.135	10.667	7	3	$206.60u$
	7	1	1.718	12.000	8	2	$208.85u$
4	2	1	3.500	8.324	7	3	$272.32u$
5	3	1	3.301	8.167	9	3	$251.60u$
3	2	1	3.320	7.829	6	3	$278.50u$

просто оценку *среднего* времени работы. Этот результат нетривиален, поскольку максимальное время работы в случае, когда приращения h удовлетворяют условию делимости (5), имеет порядок N^2, а в упр. 24 доказано, что показатель $3/2$ уменьшить нельзя.

Интересное улучшение по сравнению с теоремой P обнаружил в 1969 г. Воган Пратт. *Если все шаги сортировки выбираются из множества чисел вида* 2^p3^q, *меньших* N, *то время работы алгоритма D будет порядка* $N(\log N)^2$. В этом случае также можно внести в алгоритм несколько существенных упрощений. К сожалению, метод Пратта требует сравнительно большого числа просмотров, так что это не лучший способ выбора шагов, если только N не очень велико; см. упр. 30 и 31.

Рассмотрим *общее* время работы программы D, именно $(9B + 10NT + 13T - 10S - 3A + 1)u$. В табл. 5 показано среднее время работы для различных последовательностей шагов при $N = 8$. Каждый элемент таблицы можно вычислить с помощью формул, приведенных выше или в упр. 19, за исключением случаев, когда шаги равны 5 3 1 и 3 2 1; для этих двух случаев было проведено тщательное исследование всех 8! перестановок. Заметим, что при таком малом значении N в общем времени работы преобладают вспомогательные операции, поэтому наилучшие результаты получаются при $t = 1$; следовательно, при $N = 8$ лучше всего пользоваться простыми вставками. (Среднее время работы программы S при $N = 8$ равно всего $191.85u$.) Любопытно, что наилучший результат в двухпроходном алгоритме достигается при $h_2 = 6$, поскольку большая величина 5 оказывается важнее малой величины B. Аналогично три шага 3 2 1 минимизируют среднее число перемещений, но это не самая лучшая последовательность для трех просмотров. Быть может, интересно привести некоторые "наихудшие" перестановки, максимизирующие число перемещений, так как общий способ построения таких перестановок до сих пор не известен:

$$h_3 = 5,\quad h_2 = 3,\quad h_1 = 1: \quad 8\ \ 5\ \ 2\ \ 6\ \ 3\ \ 7\ \ 4\ \ 1 \quad \text{(19 перемещений)}$$
$$h_3 = 3,\quad h_2 = 2,\quad h_1 = 1: \quad 8\ \ 3\ \ 5\ \ 7\ \ 2\ \ 4\ \ 6\ \ 1 \quad \text{(17 перемещений)}$$

С ростом N наблюдается несколько иная картина. В табл. 6 показаны приближенные значения числа перемещений для различных последовательностей шагов при $N = 1000$. Первые несколько последовательностей удовлетворяют условию делимости (5), так что можно воспользоваться формулой (6); для получения средних значений при других последовательностях шагов применялись эмпирические тесты. Были сгенерированы пять файлов по 1000 случайных элементов, и каждый файл сортировался с каждой последовательностью шагов.

Эти данные позволяют выявить некоторые характеристики, но поведение алгоритма D все еще остается неясным. Шелл первоначально предполагал использовать шаги $\lfloor N/2\rfloor$, $\lfloor N/4\rfloor$, $\lfloor N/8\rfloor$, ...но это нежелательно, если двоичное представление числа N содержат длинные цепочки нулей. Лазарус и Фрэнк [*CACM*, **3** (1960), 20–22] предложили использовать, по существу, ту же последовательность, но добавляя 1 там, где это необходимо, чтобы сделать все шаги нечетными. Хиббард [*CACM*, **6** (1963), 206–213] предложил шаги вида $2^k - 1$; Папернов и Стасевич предложили последовательность $2^k + 1$. Среди других естественных последовательностей, использованных для. получения табл. 6,—последовательности $(2^k - (-1)^k)/3$, $(3^k - 1)/2$ и числа Фибоначчи.

Минимальное число перемещений 6600 наблюдается для шагов вида $2^k + 1$, но важно понимать, что надо учитывать не только число перемещений, хотя именно оно асимптотически доминирует в общем времени работы. Так как время работы программы D равно $9B + 10NT + \cdots$ единиц, ясно, что экономия одного просмотра примерно эквивалентна сокращению числа перемещений на $\frac{10}{9}N$; мы готовы добавить 1100 перемещений, если за счет этого удастся сэкономить один просмотр. Поэтому представляется неразумным начинать с h_t, большего, чем, скажем, $N/3$, поскольку большой шаг не убавит числа последующих перемещений настолько, чтобы оправдать первый просмотр.

Большое число экспериментов с алгоритмом D провели Джеймс Петерсон и Дэвид Л. Рассел в Стэнфордском университете в 1971 г. Они обнаружили, что для среднего числа перемещений B хорошим

Picture: Таблица 6.

приближением при $100 \le N \le 60\,000$ служат следующие формулы:

$1.09N^{1.27}$ или $.30N(\ln N)^2 - 1.35N\ln N$ для последовательности шагов $2^k + 1, \ldots, 9, 5, 3, 1$;
$1.22N^{1.26}$ или $.29N(\ln N)^2 - 1.26N\ln N$ для последовательности шагов $2^k - 1, \ldots, 15, 7, 3, 1$;
$1.12N^{1.28}$ или $.36N(\ln N)^2 - 1.73N\ln N$ для последовательности шагов $(2^k \pm 1)/3, \ldots, 11, 5, 3, 1$;
$1.66N^{1.25}$ или $.33N(\ln N)^2 - 1.26N\ln N$ для последовательности шагов $(3^k - 1)/2, \ldots, 40, 13, 4, 1$.

Например, при $N = 20\,000$ для всех этих типов шагов получаем соответственно $B \approx 31\,000, 33\,000, 35\,000, 39\,000$. Табл. 7 дает

Таблица 7
Количества перемещений по просмотрам (примеры для $N = 20000$)

h_s	B_s	h_s	B_s	h_s	B_s
4095	19460	4097	19550	3280	25210
2047	15115	2049	14944	1093	28324
1023	15869	1025	15731	364	35477
511	18891	513	18548	121	47158
255	22306	257	21827	40	62110
127	27400	129	27814	13	88524
63	35053	65	33751	4	74599
31	34677	33	34303	1	34666
15	51054	17	46044		
7	40382	9	35817		
3	24044	5	19961		
1	16789	3	9628		
		1	13277		

типичную картину того, как распределяются перемещения по просмотрам в трех из этих экспериментов. Любопытно, что *оба* вида функций $\alpha N(\ln N)^2 + \beta N\ln N$ и βN^α, кажется, довольно хорошо согласуются с наблюдаемыми данными, хотя степенная функция была существенно лучше при меньших значениях N. Дальнейшие эксперименты были выполнены для последовательности шагов $2^k - 1$ со значением N, достигающим 250 000; сорок пять испытаний с $N = 250\,000$ дали значение $B \approx 7\,900\,000$ при наблюдаемом стандартном отклонении 50 000. "Наиболее подходящими" формулами для диапазона $100 \le N \le 250\,000$ оказались соответственно $1.21N^{1.26}$ и $.39N(\ln N)^2 - 2.33N\ln N$. Так как коэффициенты в представлении степенной функцией остались почти такими же, в то время как коэффициенты в логарифмическом представлении довольно резко изменились, то разумно предположить, что именно степенная функция описывает истинное асимптотическое поведение метода Шелла.

Эти эмпирические данные ни коим образом не исчерпывают всех возможностей, и мы не получили оснований для решительных заключений о том, какие же последовательности шагов являются наилучшими для алгоритма D. Поскольку приращения вида $(3^k - 1)/2$ не увеличивают существенно числа перемещений и поскольку для них требуется лишь примерно $5/8$ числа просмотров, необходимых для шагов других типов, то, очевидно, *разумно выбирать последовательность шагов следующим образом:*

$$\textit{Принять } h_t = 1,\ h_{s+1} = 3h_s + 1 \textit{ и остановиться на } h_t, \textit{ когда } h_{t+2} \ge N. \tag{8}$$

Вставки в список. Оставим теперь метод Шелла и рассмотрим другие пути усовершенствования простых вставок. Среди общих способов улучшения алгоритма один из самых важных основывается на тщательном анализе структур данных, поскольку реорганизация структур данных, позволяющая избежать ненужных операций, часто дает существенный эффект. Дальнейшее обсуждение этой общей идеи можно найти в § 2.4, в котором изучается довольно сложный алгоритм. Посмотрим, как она применяется к такому нехитрому алгоритму, как простые вставки. Какова наиболее подходящая структура данных для алгоритма S?

Сортировка простыми вставками состоит из двух основных операций:

i) просмотра упорядоченного файла для нахождения наибольшего ключа, меньшего или равного данному ключу;

ii) вставки новой записи в определенное место упорядоченного файла.

Файл—это, очевидно, линейный список, и алгоритм S обрабатывает его, используя последовательное распределение (п. 2.2.2); поэтому для выполнения каждой операции вставки необходимо переместить примерно половину записей. С другой стороны, нам известно, что для вставок идеально подходит связанное распределение (п. 2.2.3), так как при этом требуется изменить лишь несколько связей; другая операция—последовательный просмотр—при связанном распределении почти так же проста, как и при последовательном. Поскольку списки всегда просматриваются в одном и том же направлении, то достаточно иметь списки с одной связью. Таким образом, приходим к выводу, что "правильная" структура данных для метода простых вставок—линейные списки с одной связью. Удобно также изменить алгоритм S, чтобы список просматривался в возрастающем порядке.

Алгоритм L. *(Вставки в список.)* Предполагается, что записи $R_1, \ldots, R_N$ содержат ключи $K_1, \ldots, K_N$ и "поля связи" $L_1, \ldots, L_N$, в которых могут храниться числа от 0 до N; имеется также еще одно поле связи L_0 в некоторой искусственной записи R_0 в начале файла. Алгоритм устанавливает поля связи так, что записи оказываются связанными в возрастающем порядке. Так, если $p(1)\ldots p(N)$—"устойчивая" перестановка, такая, что $K_{p(1)} \le \ldots \le K_{p(N)}$, то в результате применения алгоритма получим

$$L_0 = p(1); L_{p(i)} = p(i+1) \text{ при } 1 \le i < N; L_{p(N)} = 0. \tag{9}$$

L1 [Цикл по j.] Установить $L_0 \leftarrow N$, $L_N \leftarrow 0$. (L_0 служит "головой" списка, а 0—пустой связью; следовательно, список, по существу, циклический.) Выполнить шаги от **L2** до **L5** при $j = N-1, N-2, \ldots, 1$ и завершить работу алгоритма.

L2 [Установить p, q, K.] Установить $p \leftarrow L_0$, $q \leftarrow 0$, $K \leftarrow K_j$. (В последующих шагах мы вставим запись R_j в нужное место в связанном списке путем сравнения ключа K с предыдущими ключами в возрастающем порядке. Переменные p и q служат указателями на текущее место в списке, причем $p = L_q$, так что q всегда на один шаг отстает от p.)

L3 [Сравнить K, K_p.] Если $K \le K_p$, то перейти к шагу **L5**. (Мы нашли искомое положение записи R в списке между R_q и R_p.)

L4 [Продвинуть p, q.] Установить $q \leftarrow p$, $p \leftarrow L_q$. Если $p > 0$, то возвратиться к шагу **L3**. (Если $p = 0$, то K—наибольший ключ, обнаруженный до сих пор; следовательно, запись R должна попасть в конец списка, между R_q и R_0.)

L5 [Вставить в список.] Установить $L_q \leftarrow j$, $L_j \leftarrow p$. ∎

Таблица 8

Пример применения алгоритма вставок в список

j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
K_i:	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
L_j:	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0
L_j:	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0	15
L_j:	14	—	—	—	—	—	—	—	—	—	—	—	—	—	16	0	15

Этот алгоритм важен не только как простой метод сортировки, но и потому, что он часто встречается как часть других алгоритмов обработки списков. В табл. 8 показаны первые несколько шагов сортировки шестнадцати чисел, выбранных нами для примеров.

Программа L. *(Вставки в список.)* Предполагается, что ключ K_j хранится в поле `INPUT` + $j(0:3)$, а L_j—в поле `INPUT` + $j(4:5)$. Значения регистров: `rI1` $\equiv j$; `rI2` $\equiv p$; `rI3` $\equiv= q$; `rA`$(0:3) \equiv K$.

```
KEY    EQU   0:3
LINK   EQU   4:5
START  ENT1  N                    1          L1. Цикл по j, j ← N.
       ST1   INPUT(LINK)          1          L0 ← N.
       STZ   INPUT+N(LINK)        1          LN ← 0.
       JMP   6F                   1          Переход к уменьшению j.
2H     LD2   INPUT(LINK)          N-1        L2. Установить p, q, K. p ← L0.
       ENT3  0                    N-1        q ← 0.
       LDA   INPUT,1              N-1        K ← Kj.
3H     CMPA  INPUT,2(KEY)         B+N-1-A    L3. Сравнить K, Kp.
```

```
        JLE   5F              B+N-1-A  К L5, если K ≤ K_p.
4H      ENT3  0,2                  B   L4. Продвинуть p, q. q ← p.
        LD2   INPUT,3(LINK)        B   p ← L_q.
        J2P   3B                   B   К L3, если p > 0.
5H      ST1   INPUT,3(LINK)      N-1   L5. Вставить в список. L_q ← j.
        ST2   INPUT,1(LINK)      N-1   L_j ← p.
6H      DEC1  1                    N
        J1P   2B                   N   N > j ≥ 1.
```

Время работы этой программы равно $7B+14N-3A-6$ единиц, где N—длина файла, A—число правосторонних максимумов, а B—число инверсий в исходной перестановке. (Ср. с анализом программы S. Заметьте, что программа L не перемещает записи в памяти; это можно сделать, как в упр. 5.2-12, затратив дополнительно $20N$ единиц времени.) Программа S требует $(9B+10N-3A-9)u$, а так как B равно примерно $\frac{1}{4}N^2$, то нетрудно видеть, что за счет дополнительного пространства памяти, выделенного под поля связи, удалось сэкономить примерно 22% времени работы. Аккуратно программируя, можно сберечь еще 22% (см. упр. 33), но время работы все же останется пропорциональным N^2.

Подведем итог сделанному до сих пор. Мы начали с алгоритма S—простого и естественного алгоритма сортировки, который выполняет около $\frac{1}{4}N^2$ сравнений и около $\frac{1}{4}N^2$ перемещений. Мы усовершенствовали его, рассмотрев бинарные вставки, при которых выполняется около $N\log_2 N$ сравнений и $\frac{1}{4}N^2$ перемещений. Несколько изменив структуру данных, применив "двухпутевые вставки", сумели сократить число перемещений до $\frac{1}{8}N^2$. При сортировке методом Шелла "с убывающим шагом" число сравнений и перемещений снижается примерно до $N^{1.3}$ для тех значений N, которые встречаются на практике; при $N\to\infty$ это число можно сократить до порядка $N(\log_2 N)^2$. Дальнейшее стремление улучшить алгоритм—путем применения связанной структуры данных—привело нас к вставкам в список, при которых выполняется около $\frac{1}{4}N^2$ сравнений, 0 перемещений и $2N$ изменений связей.

Можно ли соединить лучшие свойства этих методов, сократив число сравнений до порядка $N\log N$, как при бинарных вставках, и исключив при этом перемещения данных, как при вставках

Picture: Рис. 13. Пример схемы Уилера для вставок в дерево.

в список? Ответ утвердительный: это достигается переходом к древовидной структуре. Такая возможность была впервые исследована около 1957 г. Д. Дж. Уилером, который предложил использовать двухпутевые вставки до тех пор, пока не появится необходимость перемещать данные. Тогда вместо того, чтобы их перемещать, вставляется указатель на новую область памяти, и тот же самый метод рекуррентно применяется ко всем элементам, которые нужно вставить в эту новую область памяти. Оригинальный метод Уилера [см. A. S. Douglas, *Comp. J.*, **2** (1959), 5] представляет собой сложную комбинацию последовательной и связанной памяти с узлами переменного размера; для наших шестнадцати чисел было бы сформировано дерево, показанное на рис. 13. Аналогичную, но более простую схему вставки в дерево с использованием бинарных деревьев независимо разработал около 1958 г. К.М. Бернес-Ли [см. *Comp. J.*, **3** (1960), 174, 184]. Сам этот метод и его модернизации весьма важны как для сортировки, так и для поиска, поэтому подробно они обсуждаются в гл. 6.

Еще один путь улучшить простые вставки—попытаться вставлять несколько элементов одновременно. Если, например, имеется файл из 1000 элементов и 998 из них уже отсортированы, то алгоритм S выполнит еще два просмотра файла (вставив сначала R_{999}, а потом R_{1000}). Очевидно, можно сэкономить время, если сначала сравнить ключи K_{999} с K_{1000} чтобы выяснить, который из них больше, а потом вставить их *оба* за один просмотр файла. Комбинированная операция такого рода требует около $(2/3)N$ сравнений и перемещений (ср. с упр. 3.4.2–5) вместо двух просмотров, примерно по $N/2$ сравнений и перемещений каждый.

Иначе говоря, обычно бывает полезно "группировать" операции, которые требуют длительного поиска, чтобы можно было выполнить несколько операций вместе. Если довести эту идею до ее естественного завершения, то мы заново откроем для себя сортировку посредством слияния, настолько важную, что ей посвящен отдельный пункт.

Сортировка с вычислением адреса. Теперь уж мы, несомненно, исчерпали все возможные способы усовершенствовать метод простых вставок, но давайте подумаем еще! Представьте себе, что вам дали чистый лист бумаги и собираются диктовать какие-то слова. Ваша задача—записать их в алфавитном порядке и вернуть листок с отсортированным списком слов. Услышав слово на букву А, вы будете стремиться записать его ближе к верхнему краю страницы, тогда как слово на букву Я будет, по-видимому, помещено ближе к нижнему краю страницы и т. д. Аналогичный способ применяется при расстановке книг на полке по фамилиям авторов, если книги берутся с пола в случайном порядке: ставя

книгу на полку, вы оцениваете ее конечное положение, сокращая таким образом число необходимых сравнений и перемещений. (Эффективность процесса повышается, если на полке имеется немного больше места, чем это абсолютно необходимо.) Такой метод машинной сортировки впервые предложили Исаак и Синглтон, [*JACM*, **3** (1956), 169–174]; он получил дальнейшее развитие в работе Кронмэла и Тартара [Proc. ACM Nat'l Conf., **21** (1966), 331–337].

Сортировка с вычислением адреса обычно требует дополнительного пространства памяти либо для того, чтобы оставить достаточно свободного места и не делать много лишних перемещений, либо для хранения вспомогательных таблиц, которые бы позволяли учитывать неравномерность распределения ключей. (См. сортировку распределяющим подсчетом (алгоритм 5.2D), рый выполняет около $\frac{1}{4}N^2$ сравнений и около $\frac{1}{4}N^2$ перемещений. Мы усовершенствовали его, рассмотрев бинарные вставки, при которых выполняется около $N \log_2 N$ сравнений и $\frac{1}{4}N^2$ перемещений. Несколько изменив структуру данных, применив "двухпутевые вставки", сумели сократить число перемещений до $\frac{1}{8}N^2$. При сортировке методом Шелла "с убывающим шагом" число сравнений и перемещений снижается примерно до $N^{1.3}$ для тех значений N, которые встречаются на практике; при $N \to \infty$ это число можно сократить до порядка $N(\log_2 N)^2$. Дальнейшее стремление улучшить алгоритм—путем применения связанной структуры данных—привело нас к вставкам в список, при которых выполняется около $\frac{1}{4}N^2$ сравнений, 0 перемещений и $2N$ изменений связей.

Можно ли соединить лучшие свойства этих методов, сократив число сравнений до порядка $N \log N$, как при бинарных вставках, и исключив при этом перемещения данных, как при вставках

Picture: Рис. 13. Пример схемы Уилера для вставок в дерево.

в список? Ответ утвердительный: это достигается переходом к древовидной структуре. Такая возможность была впервые исследована около 1957 г. Д. Дж. Уилером, который предложил использовать двухпутевые вставки до тех пор, пока не появится необходимость перемещать данные. Тогда вместо того, чтобы их перемещать, вставляется указатель на новую область памяти, и тот же самый метод рекуррентно применяется ко всем элементам, которые нужно вставить в эту новую область памяти. Оригинальный метод Уилера [см. A. S. Douglas, *Comp. J.*, **2** (1959), 5] представляет собой сложную комбинацию последовательной и связанной памяти с узлами переменного размера; для наших шестнадцати чисел было бы сформировано дерево, показанное на рис. 13. Аналогичную, но более простую схему вставки в дерево с использованием бинарных деревьев независимо разработал около 1958 г. К.М. Бернес-Ли [см. *Comp. J.*, **3** (1960), 174, 184]. Сам этот метод и его модернизации весьма важны как для сортировки, так и для поиска, поэтому подробно они обсуждаются в гл. 6.

Еще один путь улучшить простые вставки—попытаться вставлять несколько элементов одновременно. Если, например, имеется файл из 1000 элементов и 998 из них уже отсортированы, то алгоритм S выполнит еще два просмотра файла (вставив сначала R_{999}, а потом R_{1000}). Очевидно, можно сэкономить время, если сначала сравнить ключи K_{999} с K_{1000} чтобы выяснить, который из них больше, а потом вставить их *оба* за один просмотр файла. Комбинированная операция такого рода требует около $(2/3)N$ сравнений и перемещений (ср. с упр. 3.4.2–5) вместо двух просмотров, примерно по $N/2$ сравнений и перемещений каждый.

Иначе говоря, обычно бывает полезно "группировать" операции, которые требуют длительного поиска, чтобы можно было выполнить несколько операций вместе. Если довести эту идею до ее естественного завершения, то мы заново откроем для себя сортировку посредством слияния, настолько важную, что ей посвящен отдельный пункт.

Сортировка с вычислением адреса. Теперь уж мы, несомненно, исчерпали все возможные способы усовершенствовать метод простых вставок, но давайте подумаем еще! Представьте себе, что вам дали чистый лист бумаги и собираются диктовать какие-то слова. Ваша задача—записать их в алфавитном порядке и вернуть листок с отсортированным списком слов. Услышав слово на букву А, вы будете стремиться записать его ближе к верхнему краю страницы, тогда как слово на букву Я будет, по-видимому, помещено ближе к нижнему краю страницы и т. д. Аналогичный способ применяется при расстановке книг на полке по фамилиям авторов, если книги берутся с пола в случайном порядке: ставя книгу на полку, вы оцениваете ее конечное положение, сокращая таким образом число необходимых сравнений и перемещений. (Эффективность процесса повышается, если на полке имеется немного больше места, чем это абсолютно необходимо.) Такой метод машинной сортировки впервые предложили Исаак и Синглтон, [*JACM*, **3** (1956), 169–174]; он получил дальнейшее развитие в работе Кронмэла и Тартара [Proc. ACM Nat'l Conf., **21** (1966), 331–337].

Сортировка с вычислением адреса обычно требует дополнительного пространства памяти либо для того, чтобы оставить достаточно свободного места и не делать много лишних перемещений, либо для хранения вспомогательных таблиц, которые бы позволяли учитывать неравномерность распределения ключей. (См. сортировку распределяющим подсчетом (алгоритм 5.2D), которая является разновидностью

сортировки с вычислением адреса.) Дополнительное пространство будет, по-видимому, использоваться наилучшим образом, если отвести его под поля связи, как в методе вставок в список. К тому же отпадает необходимость выделять отдельные области для ввода и вывода; все операции можно выполнить в одной и той же области памяти.

Основная идея—так обобщить метод вставок в список, чтобы располагать не одним списком, а *несколькими.* Каждый список содержит ключи из определенного диапазона. Мы делаем важное предположение о том, что ключи распределены довольно равномерно и не "скапливаются" хаотически в каких-то диапазонах. Множество всех возможных значений ключей разбивается на M отрезков и предполагается, что данный ключ попадает в данный отрезок с вероятностью $1/M$. Отводим дополнительную память под головы M списков, а каждый список строим, как при простых вставках в список.

Нет необходимости приводить здесь этот алгоритм со всеми подробностями. Достаточно вначале установить все головы списков равными Λ, а для каждого вновь поступившего элемента предварительно решить, в какой из M отрезков он попадает, после чего вставить его в соответствующий список, как в алгоритме L.

Чтобы проиллюстрировать этот метод в действии, предположим, что наши 16 ключей разделены на $M = 4$ диапазона 0–249, 250–499, 500–749, 750–999. В процессе сортировки получаются следующие конфигурации:

Списки	Пришло 4 элемента	Пришло 8 элементов	Пришло 12 элементов	Конечное состояние
1:	061, 087	061, 087, 170	061, 087, 154, 170	061, 087, 154, 170
2:		275	275, 426	275, 426
3:	503, 512	503, 512	503, 509, 512, 653	503, 509, 512, 612 653, 677, 703
4:		897, 908	897, 908	765, 897, 908

Благодаря применению связанной памяти не возникает проблемы распределения памяти при использовании списков переменной длины.

Программа M. *(Вставки в несколько списков.)* В этой программе делаются те же предположения, что и в программе L, но ключи должны быть *неотрицательными;* так что

$$0 \le \texttt{KEY} < (\texttt{BYTESIZE})^3.$$

Этот диапазон делится в программе на `M` равных частей путем умножения каждого ключа на подходящую константу. Головы списков хранятся в ячейках `HEAD` + 1, ..., `HEAD` + `M`.

```
KEY     EQU   1:3
LINK    EQU   4:5
START   ENT2  M                 1
        STZ   HEAD,2            M       HEAD[p] ← Λ.
        DEC2  1                 M
        J2P   *-2               M       M ≥ p ≥ 1.
        ENT1  N                 1       j ← N.
2H      LDA   INPUT.1(KEY)      N
        MUL   =M(1:3)=          N       rA ← ⌊M × K_j/BYTESIZE^3⌋.
        STA   *+1(1:2)          N
        ENT3  0                 N       q ← rA.
        INC3  HEAD+1-INPUT      N       q ← LOC(HEAD[q]).
        LDA   INPUT, 1          N       K ← K_j.
        JMP   4F                N       Установить p.
3H      CMPA  INPUT,2(KEY)      B+N-A
        JLE   5F                B+N-A   Вставить, если K ≤ K_p.
        ENT3  0,2               B       q ← p.
4H      LD2   INPUT,3(LINK)     B+N     p ← LINK(q).
        J2P   3B                B+N     Переход, если не конец списка.
5H      ST1   INPUT,3(LINK)     N       LINK(q) ← LOC(R_j).
        ST2   INPUT,1(LINK)     N       LINK(LOC(R_j)) ← p.
6H      DEC1  1                 N
        J1P   2B                N       N ≥ j ≥ 1.
```

Эта программа написана для произвольного значения M, но лучше зафиксировать M, положив его равным некоторому удобному значению; можно, например, положить $M = \texttt{BYTESIZE}$, тогда головы списков можно опустошить одной-единственной командой `MOVE`, а последовательность команд 08–11, реализующих умножение, заменить командой `LD3 INPUT,1(1:1)`. Наиболее заметное отличие программы M от программы L состоит в том, что в программе M нужно рассматривать случай пустого списка, когда не надо делать сравнений.

Сколько же времени мы экономим, имея M списков вместо одного? Общее время работы программы M равно $7B+31N-3A+4M+2$ единиц, где M—число списков, N—число сортируемых записей, A и B равны соответственно числу правосторонних максимумов и числу инверсий среди ключей, принадлежащих каждому списку. (При анализе этого алгоритма в отличие от всех других анализов в данном пункте мы считаем крайний правый элемент непустой перестановки правосторонним максимумом, а не игнорируем его.) Мы уже изучили величины A и B при $M = 1$, когда их средние значения равны соответственно H_N и $\frac{1}{2}\binom{N}{2}$. Согласно предположению о распределении ключей, вероятность того, что данный список в конце сортировки будет содержать ровно n элементов, есть "биномиальная" вероятность

$$\binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n}. \tag{10}$$

Поэтому средние значения величин A и B в общем случаев равны

$$A_{ave} = M\sum_n \binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} H_n; \tag{11}$$

$$B_{ave} = M\sum_n \binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} \binom{n}{2}. \tag{12}$$

По теореме 1.2.7A

$$\sum_n \binom{N}{n}(M-1)^{-n}H_n = \left(1-\frac{1}{M}\right)^{-N}(H_N - \ln M) + \varepsilon, \qquad 0 < \varepsilon = \sum_{n>N}\frac{1}{n}\left(1-\frac{1}{M}\right)^{n-N} < \frac{M-1}{N+1};$$

следовательно,

$$A_{ave} = M(H_N - \ln M) + \delta, \qquad 0 < \delta < \frac{M^2}{N+1}\left(1-\frac{1}{M}\right)^{N+1}. \tag{13}$$

(Эта формула практически бесполезна, если $M \approx N$. Более подробно асимптотическое поведение величины A_{ave} при $M = N/\alpha$ обсуждается в упр. 5.2.2-57.) Сумму (12) легко вычислить с помощью тождества

$$\binom{N}{n}\binom{n}{2} = \binom{N}{2}\binom{N-2}{n-2},$$

которое представляет собой частный случай тождества (1.2.6-20); получаем

$$B_{ave} = \frac{1}{2M}\binom{N}{2}. \tag{14}$$

(Стандартное отклонение для величины B см. в упр. 37.) Следовательно, общее время работы программы M при фиксированном M и $N \to \infty$ равно

$$\begin{aligned} \min\quad & 31N + M + 2, \\ \text{ave}\quad & 1.75N^2/M + 31N - 3MH_N + 3M\ln M + 4M - 3 - 1.75N/M + 2, \\ \max\quad & 3.50N^2 + 24.5N + 4M + 2. \end{aligned} \tag{15}$$

Заметим, что если M не слишком велико, *то среднее время работы сокращается в M раз.* При $M = 10$ сортировка будет примерно в 10 раз быстрее, чем при $M = 1$! Однако максимальное время работы гораздо больше среднего. Таким образом, подтверждается необходимость выполнения предположения о равномерности распределения ключей, так как наихудший случай имеет место, когда все ключи попадают в один список.

Если положить $M = N$, то среднее время работы будет примерно $34.36N$, при $M = \frac{1}{2}N$ оно несколько больше, равно приблизительно $34.52N$, а при $M = N/10$ оно равно приблизительно $48.04N$. (Заметим, что $10N$ из этих единиц времени машины `MIX` тратятся на одну лишь команду умножения!) *Мы получили метод сортировки с временем работы порядка N при условии, что ключи довольно равномерно распределены в области изменения.*

Упражнения

1. [10] Является ли алгоритм S алгоритмом "устойчивой" сортировки?
2. [11] Будет ли алгоритм S правильно сортировать числа, если в шаге S3 отношение "$K \geq K_i$" заменить на "$K > K_i$"?

>3. [30] Является ли программа S самой короткой программой сортировки для машины `MIX`, или можно написать более короткую программу, которая бы давала тот же результат?

>4. [M20] Найдите минимальное и максимальное время работы программы S как функцию от N.

>5. [M27] Найдите производящую функцию $g_N(z) = \sum_{k\geq 0} p_{Nk} z^k$ для общего времени работы программы S, где p_{Nk}—вероятность того, что на выполнение программы S уйдет ровно k единиц времени при заданной исходной случайной перестановке множества $\{1, 2, \ldots, N\}$. Вычислите также стандартное отклонение времени работы для данного значения N.

6. [33] Для двухпутевых вставок, проиллюстрированных в табл. 2, по-видимому, необходимо, помимо области ввода, содержащей N записей, иметь область вывода, в которой может храниться $2N + 1$ записей. Покажите, что можно выполнять двухпутевые вставки, имея как для ввода, так и для вывода пространство, достаточное для хранения всего $N + 1$ записей.
7. [M20] Пусть $a_1\, a_2 \ldots a_n$—случайная перестановка множества $\{1, 2, \ldots, n\}$; каково среднее значение величины $|a_1 - 1| + |a_2 - 2| + \cdots + |a_n - n|$? (Оно равно произведению n на среднее чистое расстояние, на которое перемещается запись в процессе сортировки.)
8. [10] Является ля алгоритм D алгоритмом "устойчивой" сортировки?
9. [20] Какие значения A и B и какое общее время работы программы D соответствуют табл. 3 и 4? Проанализируйте достоинства метода Шелла по сравнению с простыми вставками в этом случае.

>10. [22] В случае, когда $K_j \geq K_{j-h}$, в шаге D3 алгоритм D предписывает выполнение множества ненужных действий. Покажите, как можно изменить программу D, чтобы избежать этих избыточных вычислений, и обсудите преимущества такого изменения.

11. [M10] Какой путь на решетке (аналогичной представленной на рис. 11) соответствует перестановке 1 2 5 3 7 4 8 6 9 11 10 12?
12. [M20] Докажите, что сумма вертикальных весов пути на решетке равна числу инверсий соответствующей 2-упорядоченной перестановки.

>13. [M22] Поясните, как нужно приписать веса *горизонтальным* отрезкам вместо вертикальных, чтобы сумма горизонтальных весов пути на решетке равнялась числу инверсий в соответствующей 2-упорядоченной перестановке.

14. [M24] (a) Покажите, что для сумм, определяемых равенством (2), $A_{2n+1} = 2A_{2n}$. (b) Если положить $r = -s - 1$, $t = 1$, то общее тождество из упр. 1.2.6-26 упрощается до

$$\sum_k \binom{2k+s}{k} z^k = \frac{1}{\sqrt{1-4z}} \left(\frac{1-\sqrt{1-4z}}{2z}\right)^s.$$

Рассмотрев сумму $\sum_n A_{2n} z^n$ покажите, что

$$A_{2n} = n \cdot 4^{n-1}.$$

>15. [BM33] Пусть $g_n(z)$, $\bar{g}_n(z)$, $h(z)$, $\bar{h}_n(z)$ равны $\sum z^{\text{общий вес пути}}$, где сумма берется по всем путям длины $2n$ на решетке из $(0, 0)$ в (n, n), удовлетворяющим некоторым ограничениям на вершины, через которые эти пути проходят: для $h_n(z)$ нет ограничений, для $g_n(z)$ пути не должны проходить через вершины (i, j), такие, что $i > j$; $\bar{h}_n(z)$ и $\bar{g}_n(z)$ определяются аналогично, но не допускаются также и вершины (i, i) при $0 < i < n$. Таким образом,

$$g_0(z) = 1, \quad g_1(z) = z, \quad g_2(z) = z^3 + z^2; \quad \bar{g}_1(z) = z, \quad \bar{g}_2(z) = z^3;$$
$$h_0(z) = 1, \quad h_1(z) = z + 1, \quad h_2(z) = z^3 + z^2 + 3z + 1;$$
$$\bar{h}_1(z) = z + 1, \quad \bar{h}_2(z) = z^3 + z.$$

Найдите рекуррентные соотношения, определяющие эти функции, и воспользуйтесь этими рекуррентными соотношениями для доказательства равенства

$$h_n''(1) + h_n'(1) = \frac{7n^3 + 4n^2 + 4n}{30} \binom{2n}{n}.$$

(Отсюда легко находится точная формула дисперсии числа инверсий в случайной 2-упорядоченной перестановке множества $\{1, 2, \ldots, 2n\}$; она асимптотически приближается к $\left(\frac{7}{30} - \frac{\pi}{16}\right) n^3$.)

16. [M24] Найдите формулу максимального числа инверсий в h-упорядоченной перестановке множества $\{1, 2, \ldots, n\}$. Каково максимально возможное число перемещений, выполняемых алгоритмом D, если шаги сортировки удовлетворяют условию делимости (5)?
17. [M21] Покажите, что если $N = 2^t$ и $h_s = 2^{s-1}$ при $t \geq s \geq 1$, то существует единственная перестановка множества $\{1, 2, \ldots, N\}$, максимизирующая число перемещений, выполняемых алгоритмом D. Найдите простой способ описания этой перестановки.
18. [BM24] При больших значениях N сумму (6) можно оценить величиной

$$\frac{1}{4}\frac{N^2}{h_t} + \frac{\sqrt{\pi}}{8}\left(\frac{N^{3/2}h_t^{1/2}}{h_{t-1}} + \cdots + \frac{N^{3/2}h_2^{1/2}}{h_1}\right).$$

Каковы действительные числа $h_t, \ldots, h_1$, минимизирующие это выражение, если N и t фиксированы, а $h_1 = 1$?

>19. [M25] Каково среднее значение величины A в анализе времени работы программы D, если шаги сортировки удовлетворяют условию делимости (5)?
20. [M20] Покажите, что теорема K следует из леммы L.
21. [M25] Пусть h и k—взаимно простые целые положительные числа. Покажите, что наибольшее целое число, которое нельзя представить в виде $ah+bk$, где a и b—*неотрицательные* целые числа, равно $hk-h-k$. Следовательно, если файл является одновременно h-упорядоченным и k-упорядоченным, то $K_i \leq K_j$ при $j - i \geq (h-1)(k-1)$.
22. [M30] Докажите, что любое целое число $\geq 2^h(2^h - 1)$ можно представить в виде $a_0(2^h - 1) + a_1(2^{h+1} - 1) + a_2(2^{h+2} - 1) + \ldots$, где a_j—неотрицательные целые числа; но число $2^h(2^h - 1) - 1$ нельзя представить в таком виде. Более того, существует ровно $2^{h-1}(2^h + h - 3)$ целых положительных чисел, которые нельзя представить в таком виде.

Найдите аналогичные формулы для случая, когда в этом представлении выражение $2^k - 1$ заменено на $2^k + 1$.

>23. [M22] Докажите, что если h_{s+2} и h_{s+1} взаимно просты, то число перемещений, выполняемых алгоритмом D при просмотре с шагом h_s, есть $O(Nh_{s+2}h_{s+1}/h_s)$. (*Указание:* см. упр. 21.)
24. [M42] Докажите, что теорема P—наилучшая из возможных теорем в том смысле, что показатель 3/2 нельзя уменьшить.

>25. [M22] Сколько существует перестановок множества $\{1, 2, \ldots, n\}$, являющихся одновременно 3-упорядоченными и 2-упорядоченными? Каково максимальное число инверсий в такой перестановке? Чему равно суммарное число инверсий во всех таких перестановках?
26. [M35] Может ли 3-, 5- и 7-упорядоченный файл из N элементов содержать более N инверсий?
27. [M50] Предложите эффективный алгоритм построения перестановок, которые требуют максимального числа операций перемещения в алгоритме D при заданных N, h_t, h_{t-1}, $\ldots$, h_1.
28. [15] Какая из последовательностей шагов, указанных в табл. 6, наилучшая для программы D с точки зрения суммарного времени работы?
29. [M42] Найдите для $N = 1000$ и различных значений t эмпирические значения $h_t, \ldots, h_1$, которые, быть может, минимизируют среднее число операций перемещения в алгоритме D в том смысле, что если изменить одну из величин h, не меняя остальных, то среднее число перемещений возрастет.
30. [M23] (В. Пратт.) Покажите, что если множество шагов есть $\{2^p3^q \mid 2^p3^q < N\}$, то число просмотров равно примерно $\frac{1}{2}(\log_2 N)(\log_3 N)$ и на каждый просмотр приходится не более $N/2$ операций перемещения. В действительности при любом просмотре, если $K_{j-h} > K_j$, то всегда K_{j-3h}, $K_{j-2h} >\leq K_j < K_{j-h} \leq K_{j+h}$, K_{j+2h}, так что можно просто поменять местами K_{j-h} и K_j и увеличить j на $2h$, сэкономив в алгоритме D 2 сравнения. (*Указание:* см. упр. 25.)

>31. [25] Напишите `MIX`-программу для алгоритма сортировки, предложенного В. Праттом (упр. 30). Выразите ее время работы через величины A, B, S, T, N, аналогичные соответствующим величинам в программе D.
32. [10] Каково будет окончательное содержимое связей $L_0L_1 \ldots L_{16}$, если провести до конца сортировку списка вставками в табл. 8?

>33. [25] Найдите способ усовершенствовать программу L, чтобы ее время работы определялось величиной $5B$, а не $7B$, где B—число инверсий.
34. [M10] Проверьте формулу (10).
35. [18] Предположим, что размер байта машины `MIX` равен 100 и шестнадцать ключей в табл. 8 равны на самом деле 503 000, 087 000, 512 000, $\ldots$, 703 000. Определите время работы программ M и L с этими данными при $M = 4$.

36. [BM16] Если программа выполняется приблизительно за $A/M + B$ единиц времени и использует $C + M$ ячеек памяти, то при каком выборе M достигается оптимальное значение произведения пространства на время?
37. [M25] Пусть $g_n(z)$—производящая функция для числа инверсий в случайной перестановке n элементов (ср. с формулой (5.1.1-11)). Пусть $g_{NM}(z)$—соответствующая производящая функция для величины B в программе M. Покажите, что

$$\sum_{N\geq 0} g_{NM}(z) M^N w^N/N! = \left(\sum_{n\geq 0} g_n(z) w^n/n!\right)^M,$$

и воспользуйтесь этой формулой для вывода дисперсии величины B.
38. [BM23] (Р. М. Карп.) Пусть $F(x)$—некоторая функция распределения вероятностей, причем $F(0) = 0$ и $F(1) = 1$. Докажите, что если ключи K_1, K_2, ..., K_N независимо выбираются случайным образом из этого распределения и $M = cN$, где c—константа, а $N \to \infty$, то время работы программы M есть $O(N)$, если только F—достаточно гладкая функция. (Ключ K вставляется в список j, если $\lfloor MK \rfloor = j - 1$; это случается с вероятностью $F(j/M) - F((j-1)/M)$. В тексте рассматривался лишь случай $F(x) = x$, $0 \leq x \leq 1$.)

Упражнения

1. [M25] *(Беспорядок в библиотеке.)* Небрежные читатели часто ставят книги на полки в библиотеке не на свое место. Один из способов измерить степень беспорядка в библиотеке—посмотреть, какое минимальное, количество раз пришлось бы брать книгу с одного места и вставлять ее в другое место до тех пор, пока книги не окажутся расположенными в правильном порядке.

Пусть $\pi = a_1\ a_2\ \ldots\ a_n$—перестановка множества $\{1, 2, \ldots, n\}$. "Операция удаления-вставки" заменяет π на

$$a_1\ a_2\ \ldots\ a_{i-1}\ \ldots\ a_j\ a_i\ a_{j+1}\ \ldots\ a_n$$

или на

$$a_1\ \ldots\ a_j\ a_i\ a_{j+1}\ \ldots\ a_{i-1}\ a_{i+1}\ \ldots\ a_n$$

при некоторых i и j. Пусть $\mathrm{dis}(\pi)$—минимальное число операций удаления-вставки, необходимое для упорядочения перестановки π. Можно ли выразить $\mathrm{dis}(\pi)$ через более простые характеристики π?
2. [40] Проведите эксперименты со следующей модификацией сортировки с убывающим шагом, имеющей целью ускорение "внутреннего цикла": для $s = t, t-1, \ldots, 2$ и для $j = h_s + 1, h_s + 2, \ldots, N$ поменять местами $R_{j-h_s} \leftrightarrow R_j$, если $K_{j-h_s} > K_j$. (Таким образом, результат обменов не распространяется; не производится сравнений $K_{j-h_s} : K_{j-2h_s}$. Записи не обязательно будут h_s-отсортированы, но эти предварительные просмотры способствуют сокращению числа инверсий.) Сортировка завершается применением простых вставок.

5.2.2. Обменная сортировка

Мы подошли теперь ко второму из семейств алгоритмов сортировки, упомянутых в самом начале § 5.2,—к методам "обменов" или "транспозиций", предусматривающих систематический обмен местами между элементами пар, в которых нарушается упорядоченность, до тех пор, пока таких пар не останется.

Процесс простых вставок (алгоритм 5.2.1S) можно рассматривать как обменную сортировку: мы берем новую запись R_j и, по существу, меняем местами с соседями слева до тех пор, пока она не займет нужного места. Таким образом, классификация методов сортировки по таким семействам, как "вставки", "обмены", "выбор" и т. д., не всегда четко определена. В этом пункте мы рассмотрим четыре типа методов сортировки, для которых обмены являются основной характеристикой: *обменную сортировку с выбором* ("метод пузырька"), *обменную сортировку со слиянием* (параллельную сортировку Бэтчера), *обменную сортировку с разделением* ("быструю сортировку" Хоара), *поразрядную обменную сортировку*.

Метод пузырька. Пожалуй, наиболее очевидней способ обменной сортировки это сравнивать K_1 с K_2, меняя местами R_1 и R_2, если их ключи не упорядочены, затем проделать то же самое с R_2 и R_3, R_3 и R_4 и т. д. При выполнении этой последовательности операций записи с большими ключами будут продвигаться вправо, и на самом деле запись с наибольшим ключом займет положение R_N. При многократном выполнении этого процесса соответствующие записи попадут в позиции R_{N-1}, R_{N-2} и т. д., так что в конце концов все записи будут упорядочены.

На рис. 14 показано действие этого метода сортировки на шестнадцати ключах 503 087 512 ...703. Файл чисел удобно

Picture: Рис. 14. Сортировка методом пузырька в действии.

представлять не горизонтально, а вертикально, чтобы запись R_N была сверху, а R_1—снизу. Метод назван "методом пузырька", потому что большие элементы, подобно пузырькам, "всплывают" на соответствующую позицию в противоположность "методу погружения" (т. е. методу простых вставок), в котором элементы погружаются на соответствующий уровень. Метод пузырька известен также под более прозаическими именами, такими, как "обменная сортировка с выбором" или метод "распространения". Нетрудно видеть, что после каждого просмотра файла все записи, начиная с самой последней, которая участвовала в обмене, и выше, должны занять свой окончательные позиции,так что их не нужно проверять при последующих просмотрах. На рис. 14 такого рода продвижения алгоритма отмечены черточками. Заметим, например, что после четвертого просмотра стало известно, что еще пять записей заняли свои окончательные позиции. При последнем, просмотре вообще не было произведено обменов. Теперь, сделав эти наблюдения, мы готовы сформулировать алгоритм.

Алгоритм В. *(Метод пузырька.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$.

B1 [Начальная установка BOUND.] Установить $\texttt{BOUND} \leftarrow N$. (BOUND—индекс самого верхнего элемента, о котором еще не известно, занял ли он уже свою окончательную позицию; таким образом, мы отмечаем, что к этому моменту еще ничего не известно.)

B2 [Цикл по j.] Установить $t \leftarrow 0$. Выполнить шаг **B3** при $j = 1, 2, \ldots, \texttt{BOUND} - 1$. Затем перейти к шагу **B4**. (Если $\texttt{BOUND} = 1$, то сразу перейти к **B4**.)

B3 [Сравнение/обмен $R_j : R_{j+1}$][8]. Если $K_j > K_{j+1}$, то поменять местами $R_j \leftrightarrow R_{j+1}$ и установить $t \leftarrow j$.

B4 [Были ли обмены?] Если $t = 0$, то завершить работу алгоритма. В противном случае установить $\texttt{BOUND} \leftarrow t$ и возвратиться к шагу **B2**. ∎

Picture: Рис. 15. Блок-схема сортировки методом пузырька.

Программа В. *(Метод пузырька.)* Как и в предыдущих MIX-программах этой главы, мы предполагаем, что сортируемые элементы находятся в ячейках INPUT + 1, ..., INPUT + N; регистры: rI1 $\equiv t$; rI2 $\equiv j$.

```
START   ENT1  N              1      B1. Начальная установка BOUND. t ← N.
1H      ST1   BOUND (1:2)    A      BOUND ← t.
        ENT2  1              A      B2. Цикл. по j.
        ENT1  0              A      t ← 0.
        JMP   BOUND          A      Выход, если j ≥ BOUND.
3H      LDA   INPUT, 2       C      B3. Сравнение/обмен R_j : R_{j+1}.
        CMPA  INPUT+1,2      C
        JLE   2F             C      Если K_j ≤ K_j + 1, то без обмена.
        LDX   INPUT+1,2      B      R_{j+1}
        STX   INPUT,2        B      → R_j.
        STA   INPUT+1,2      B      (прежнее R_j) → R_{j+1}
        ENT1  0,2            B      t ← j.
9H      INC2  1              C      j ← j + 1.
BOUND   ENTX  -*,2           A+C    rX ← j − BOUND. (Изменяемая инструкция)
        JXN   3B             A+C    1 ≤ j < BOUND.
4H      J1P   1B             A      B4. Были ли обмены? К B2, если t > 0.
```

∎

Анализ метода пузырька. Очень полезно исследовать время работы алгоритма В. Оно определяется тремя величинами: числом просмотров A, числом обменов B и числом сравнений C. Если исходные ключи различны и расположены в случайном порядке, то можно предположить, что они образуют случайную перестановку множества $\{1, 2, \ldots, n\}$. Понятие таблицы инверсий (п. 5.1.1) приводит к простому способу описания действия каждого просмотра при сортировке методом пузырька.

Теорема I. *Пусть $a_1\ a_2\ \ldots\ a_n$—перестановка множества $\{1, 2, \ldots, n\}$, а $b_1\ b_2\ \ldots b_n$—соответствующая таблица инверсий. Если в результате очередного просмотра при сортировке методом пузырька (алгоритм*

[8])

B) перестановка $a_1\ a_2\ \ldots a_n$ *преобразуется в* $a'_1\ a'_2\ \ldots a'_n$, *то соответствующая таблица инверсий* $b'_1\ b'_2\ \ldots b'_n$ *получается из* $b_1\ b_2\ \ldots b_n$ *уменьшением на единицу каждого нулевого элемента.*

Доказательство Если перед a_i имеется больший элемент, то a_i поменяется местами с наибольшим из предшествующих элементов, так что b_i уменьшится на единицу. С другой стороны, если перед a_i нет элемента, большего a_i, то a_i никогда не поменяется местами с большим элементом, так что b_{a_i} останется 0. ∎

Итак, можно разобраться в том, что происходит в процессе сортировки методом пузырька, изучая последовательность таблиц инверсий между просмотрами. Вот как выглядят, например, таблицы инверсий для рис. 14:

$$\begin{array}{lcccccccccccccccc}
 & 3 & 1 & 8 & 3 & 4 & 5 & 0 & 4 & 0 & 3 & 2 & 2 & 3 & 2 & 1 & 0 \\
\text{Просмотр 1} & & & & & & & & & & & & & & & & \\
 & 2 & 0 & 7 & 2 & 3 & 4 & 0 & 3 & 0 & 2 & 1 & 1 & 2 & 1 & 0 & 0 \\
\text{Просмотр 2} & & & & & & & & & & & & & & & & \\
 & 1 & 0 & 6 & 1 & 2 & 3 & 0 & 2 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{Просмотр 3} & & & & & & & & & & & & & & & & \\
 & 0 & 0 & 5 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array} \tag{1}$$

и т. д. Поэтому, если $b_1\, b_2\, \ldots\, b_n$—таблица инверсий исходной перестановки, то должны выполняться равенства

$$A = 1 + \max(b_1, b_2, \ldots, b_n); \tag{2}$$

$$B = b_1 + b_2 + \cdots + b_n; \tag{3}$$

$$C = c_1 + c_2 + \cdots + c_A, \tag{4}$$

где c_j—значение `BOUND` $- 1$ перед началом j-го просмотра. Используя таблицы инверсий, запишем

$$c_j = \max\{\, b_i + i \mid b_i \geq j - 1 \,\} - j \tag{5}$$

(см. упр. 5). Следовательно, в примере (1) $A = 9$, $B = 41$, $C = 15 + 14 + 13 + 12 + 7 + 5 + 4 + 3 + 2 = 75$. Общее время сортировки на машине `MIX` для рис. 14 равно $1030u$.

Распределение величины B (суммарное число инверсий в случайной перестановке) нам уже хорошо известно; таким образом, остается проанализировать величины A и C.

Вероятность того, что $A \leq k$, равна произведению $1/n!$ на число таблиц инверсий, не содержащих компонент $\geq k$, именно $k^{n-k}k!$ при $1 \leq k \leq n$. Следовательно, вероятность того, что потребуется ровно k просмотров, равна

$$A_k = \frac{1}{n!}(k^{n-k}k! - (k-1)^{n-k+1}(k-1)!). \tag{6}$$

Теперь можно вычислить среднее значение $\sum kA_k$, производя суммирование по частям, получаем

$$A_{ave} = n + 1 \sum_{0 \leq k \leq n} -\frac{k^{n-k}k!}{n!} = n + 1 + P(n), \tag{7}$$

где $P(n)$—функция, асимптотическое поведение которой, как было показано в п. 1.2.11, описывается формулой $\sqrt{\pi n/2} - \frac{2}{3} + O(1/\sqrt{n})$. Формула (7) была найдена без доказательства Э. Х. Фрэндом [*JACM*, **3** (1956), 150]; доказательство в своей докторской диссертации привел Говард Б. Демут [(Stanford University: October, 1956), 64–68]. Стандартное отклонение величины A см. в упр. 7.

Суммарное число сравнений C исследовать несколько сложнее, и мы рассмотрим только C_{ave}. Пусть $f_j(k)$—число таких таблиц инверсий $b_1\ \ldots\ b_n$ (n фиксировано), что при $1 \leq i \leq n$ либо $b_i < j-1$, либо $b_i + i - j \leq k$; тогда

$$f_j(k) = (j+k)!(j-1)^{n-j-k} \qquad \text{при } 0 \leq k \leq n-j. \tag{8}$$

(См. упр. 8.) Среднее значение c_j в (5) равно $\sum k(f_j(k) - f_j(k-1)))/n!$; суммируя по частям, а затем по j, получаем формулу

$$C_{ave} = \binom{n+1}{2} - \frac{1}{n!} \sum_{\substack{1 \leq j \leq n \\ 0 \leq k \leq n-j}} f_j(k) = \binom{n+1}{2} - \frac{1}{n!} \sum_{0 \leq r < s \leq n} s!r^{n-s}. \tag{9}$$

Определить асимптотическое значение здесь не так просто, и мы вернемся к нему в конце этого пункта.

Чтобы подвести итог нашего анализа метода пузырька, запишем формулы, выведенные выше (а также ниже), следующим образом:

$$A = (\min 1, \operatorname{ave} N - \sqrt{\pi N/2} + O(1), \max N); \tag{10}$$

$$B = (\min 0, \operatorname{ave} \frac{1}{4}(N^2 - N), \max \frac{1}{2}(N^2 - N)); \tag{11}$$

$$C = \left(\min N - 1, \operatorname{ave} \frac{1}{2}(N^2 - N \ln N - (\gamma + \ln 2 - 1)N) + O(\sqrt{N}), \max \frac{1}{2}(N^2 - N)\right). \tag{12}$$

Во всех случаях минимум достигается, когда исходный файл уже упорядочен, а максимум—когда записи расположены в обратном порядке; таким образом, время работы для машины `MIX` равно $8A+7B+8C+1 = (\min 8N + 1, \operatorname{ave} 5.75N^2 + O(N \ln N), \max 7.5N^2 + 0.5N + 1)$.

Усовершенствования метода пузырька. Мы потратили много усилий на анализ метода пузырька, и, хотя способы, применявшиеся при вычислениях, поучительны, результаты разочаровывают, поскольку они говорят о том, что метод пузырька вовсе не так уж хорош. По сравнению с простыми вставками (алгоритм 5.2.1S) метод пузырька описывается более сложной программой и требует примерно в 2 раза больше времени!

Можно предложить несколько путей улучшения метода пузырька. Например, на рис. 14 первое сравнение в 4-м просмотре излишне, так же как и первые два сравнения в 5-м просмотре и первые три в 6-м и 7-м просмотрах. Заметим также, что за один просмотр элемент не может переместиться более чем на одну позицию влево; так что если наименьший элемент вначале находился в правом конце, то мы вынуждены выполнить максимальное число сравнений. Это наводит на мысль о "шейкер-сортировке", когда файл просматривается попеременно в обоих направлениях (рис. 16). При таком подходе среднее число сравнений несколько сокращается. К. Э. Айверсон [A Programming Language (Wiley, 1962), 218–219] сделал интересное в этом отношении наблюдение: если j—такой индекс, что R_j и R_{j+1} не меняются местами при двух последовательных просмотрах

Picture: Рис. 16. "Шейкер-сортировка".

в противоположных направлениях, то записи R_j и R_{j+1} должны занимать свои окончательные позиции, и они могут не участвовать в последующих сравнениях. Например, просматривая перестановку 4 3 2 1 8 6 9 7 5 слева направо, получаем 3 2 1 4 6 8 7 5 9: записи R_4 и R_5 не поменялись местами. При просмотре последней перестановки справа налево R_4 все еще меньше (новой) записи R_5; следовательно, можно сразу же сделать вывод о том, что записи R_4 и R_5 могут и не участвовать ни в одном из последующих, сравнений.

Однако ни одно из этих усовершенствований не приводит к лучшему алгоритму, чем алгоритм сортировки простыми вставками, а мы уже знаем, что даже он не годится при больших N. Другая идея состоит в том, чтобы избежать большинства обменов. Так как большая часть элементов во время обменов просто сдвигается на один шаг влево, то можно было бы достичь того же эффекта, иначе рассматривая массив—сместив начало отсчета! Но полученный алгоритм не превосходит метода простого *выбора* (алгоритм 5.2.3S), который мы изучим позже.

Короче говоря, метод пузырька, кажется, не обладает никакими достоинствами, за которые его можно было бы порекомендовать, если не считать легко запоминающегося названия и интересных теоретических задач, к которым он приводит.

Параллельная сортировка Бэтчера. Если мы хотим получить алгоритм обменной сортировки, время работы которого имеет порядок, меньший N^2, то необходимо подбирать для сравнений некоторые пары *несоседних* ключей (K_i, K_j); иначе придется

Picture: Рис. 17. Алгоритм M.

выполнить столько обменов, сколько инверсий имеется в исходной перестановке, а среднее число инверсий равно $(N^2 - N)/4$. В 1964 г. К. Э. Бэтчер [Proc. AFIPS Spring Joint Computer Conference, **32** (1968), 307–314] открыл один искусный способ запрограммировать последовательность сравнений, предназначенную для отыскания возможных обменов. Его метод далеко не очевиден. В самом деле, только для того, чтобы показать его справедливость, требуется весьма сложное доказательство, так как выполняется, относительно мало сравнений. Мы обсудим два доказательства, одно в этом пункте и одно в п. 5.3.4.

Схема сортировки Бэтчера несколько напоминает сортировку Шелла, но сравнения выполняются по-новому, так что распространение обменов не обязательно. Так, можно сравнить табл. 1 с табл. 5.2.1-3. Сортировка Бэтчера действует как 8-сортировка, 4-сортировка, 2-сортировка и 1-сортировка, но сравнения не перекрываются. Так как в алгоритме Бэтчера, по существу, происходит слияние пар отсортированных подпоследовательностей, то его можно назвать "обменной сортировкой со слиянием".

Алгоритм М. *(Обменная сортировка со слиянием.)* Записи$R_1, \ldots, R_N$ переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$. Предполагается, что $N \ge 2$.

M1 [Начальная установка p.] Установить $p \leftarrow 2^{t-1}$, где $t = \lceil \log_2 N \rceil$—наименьшее целое число, такое, что $2^t \ge N$. (Шаги **M2**, ..., **M5** будут выполняться с $p = 2^{t-1}, 2^{t-2}, \ldots, 1$.)

M2 [Начальная установка q, r, d.] Установить $q \leftarrow 2^{t-1}$, $r \leftarrow 0$, $d \leftarrow p$.

M3 [Цикл по t.] Для всех t, таких, что $0 \le i < N - d$ и $i \wedge p = r$, выполнить шаг **M4**. Затем перейти к шагу **M5**. (Здесь через $i \wedge p$ обозначена операция "логическое и" над двоичными представлениями чисел i и p; все биты результата равны 0, кроме тех, для которых в соответствующих позициях i и p находятся единичные биты. Так, $13 \wedge 21 = (1101)_2 \wedge (10101)_2 = (00101)_2 = 5$. К этому моменту d—нечетное кратное p (т. е. частное от деления d на p нечетно), а p—степень двойки, так что $i \wedge p \ne (i + d) \wedge p$; отсюда следует, что действия шага **M4** можно выполнять при всех нужных значениях i в любом порядке или даже одновременно.)

M4 [Сравнение/обмен $R_{i+1} : R_{i+d+1}$.] Если $K_{i+1} > K_{i+d+1}$, то поменять местами $R_{i+1} \leftrightarrow R_{i+d+1}$.

M5 [Цикл по q.] Если $q \ne p$, то установить $d \leftarrow q - p$, $q \leftarrow q/2$, $r \leftarrow p$ и возвратиться к шагу **M3**.

M6 [Цикл по p.] (К этому моменту перестановка $K_1 K_2 \ldots K_N$ будет p-упорядочена.) Установить $p \leftarrow \lfloor p/2 \rfloor$. Если $p > 0$, то возвратиться к шагу **M2**. ∎

В табл. 1 этот метод проиллюстрирован при $N = 16$. Заметим, что алгоритм сортирует N элементов, по существу, путем независимой сортировки подфайлов R_1, R_3, R_5, ... и R_2, R_4, R_6, ..., после чего выполняются шаги M2, ..., M5 с $p = 1$, чтобы слить две отсортированные последовательности.

Чтобы доказать, что магическая последовательность сравнений/обменов, описанная в алгоритме М, действительно сортирует любой файл $R_1\, R_2 \ldots R_N$, мы должны показать только, что в результате выполнения шагов от M2 до M5 с $p = 1$ будет слит любой 2-упорядоченный файл $R_1\, R_2 \ldots R_N$. Для этой цели можно воспользоваться методом решеточных диаграмм из п. 5.2.1 (см. рис. 11); каждая 2-упорядоченная перестановка множества $\{1, 2, \ldots, N\}$ соответствует на решетке единственному пути из вершины $(0, 0)$ в $(\lceil N/2 \rceil, \lfloor N/2 \rfloor)$. На рис. 18(a) показан

Picture: Таблица 1. p.139

пример при $N = 16$, соответствующий перестановке 1 3 2 4 10 5 11 6 13 7 14 8 15 9 16 12. Действие шага M3 при $p = 1$, $q = 2^{t-1}$, $r = 0$, $d = 1$ состоит в сравнении (и, возможно, обмене) записей $R_1 : R_2$, $R_3 : R_4$ и т. д. Этой операции соответствует простое преобразование решеточного пути: "перегиб" относительно диагонали, если необходимо, так, чтобы путь нигде не проходил выше диагонали. (См. рис. 18(b) и доказательство в упр. 10.) Действие последующих повторений шага M3 с $p = r = 1$ и $d = 2^{t-1} - 1$, $2^{t-2} - 1$, ..., 1 состоит в сравнении/обмене записей $R_2 : R_{2+d}$, $R_4 : R_{4+d}$ и т. д., и опять имеется простая геометрическая интерпретация: путь "перегибается" относительно прямой, расположенной на $(d + l)/2$ единиц ниже диагонали (см. рис. 18(c) и (d)). В конце концов приходим к пути, изображенному на рис. 18(e), который соответствует полностью отсортированной перестановке. На этом "геометрическое доказательство" справедливости алгоритма Бэтчера завершается; этот алгоритм можно было бы назвать сортировкой посредством перегибов!

MIX-программа для алгоритма М приведена в упр. 12. К сожалению, количество вспомогательных операций, необходимых для управления последовательностью сравнений, весьма велико, так что программа менее эффективна, чем другие методы, которые мы разбирали. Однако алгоритм обладает одним важным компенсирующим качеством: все сравнения/обмены, определяемые данной итерацией шага М3, можно выполнять *одновременно* на ЭВМ или логических схемах, которые допускают параллельные

Picture: Рис. 18. Геометрическая интерпретация метода Бэтчера, $N = 16$.

вычисления. С такими параллельными операциями сортировка выполняется за $\frac{1}{2}\lceil \log_2 N \rceil (\lceil log_2 N \rceil + 1)$ шагов, и это один из самых быстрых известных общих методов. Например, *1024 элемента можно отсортировать методом Бэтчера всего за 55 параллельных шагов*. Его ближайшим соперником является метод Пратта (см. упр. 5.2.1–30), который затрачивает 40 или 73 шага в зависимости от того, как считать: если мы готовы допустить перекрытие сравнений до тех пор, пока не потребуется выполнять перекрывающиеся обмены, то для сортировки 1024 элементов методом Пратта требуется всего 40 циклов сравнения/обмена. Дальнейшие пояснения см. в п. 5.3.4.

"Быстрая сортировка". В методе Бэтчера последовательность сравнений предопределена: каждый раз сравниваются одни и те же пары ключей независимо от того, что мы могли узнать о файле из предыдущих сравнений. Это утверждение в большой мере справедливо и применительно к методу пузырька, хотя

алгоритм B и использует в ограниченной степени полученные сведения, с тем чтобы сократить количество работы в правом конце файла. Обратимся теперь к совсем иной стратегии, при которой используется результат каждого сравнения, чтобы определить, какие ключи сравнивать следующими. Такая стратегия не годится для параллельных вычислений, но она может оказаться плодотворной для вычислительных машин, работающих последовательно.

Итак, рассмотрим. следующую схему сравнений/обменов. Имеются два указателя i и j, причем вначале $i = l$, а $j = N$. Сравним $K_i : K_j$, и если обмен не требуется, то уменьшим j на единицу и повторим этот процесс. После первого обмена увеличим i на единицу и будем продолжать сравнения, увеличивая i, пока не произойдет еще один обмен. Тогда опять уменьшим j и т. д.; будем "сжигать свечку с обоих концов", пока не станет $i = j$. Посмотрим, например, что произойдет с нашим файлом из шестнадцати чисел:

Дано:	**503**	087	512	061	908	170	897	275	653	426	154	509	612	677	765	**703**
																уменьшить j
1-йобмен	**154**	087	512	061	908	170	897	275	653	426	**503**	509	612	677	765	703
																увеличить i
2-й обмен	154	087	**503**	061	908	170	897	275	653	426	**512**	509	612	677	765	703
																уменьшить j
3-й обмен	154	087	**426**	061	908	170	897	275	653	**503**	512	509	612	677	765	703
																увеличить i
4-й обмен	154	087	426	061	**503**	170	897	275	653	**908**	512	509	612	677	765	703
																уменьшить j
5-й обмен	154	087	426	061	**275**	170	897	**503**	653	908	512	509	612	677	765	703
																увеличить i
6-й обмен	154	087	426	061	275	170	**503**	**897**	653	808	512	509	612	677	765	703
																уменьшить j

(Чтобы выявить состояние i и j, ключи K_i и K_j напечатаны жирным шрифтом.) Заметим, что в каждом сравнении этого примера участвует ключ 503; вообще в каждом сравнении будет участвовать исходный ключ K_1, потому что он будет продолжать обмениваться местами с другими ключами каждый раз, когда мы переключаем направление. К моменту, когда $i = j$, исходная запись R_1 займет свою окончательную позицию, потому что, как нетрудно видеть, слева от нее не будет больших ключей, а справа—меньших. Исходный файл окажется разделен таким образом, что первоначальная задача сведется к двум более простым: сортировке файла $R_1 \ldots R_{i-1}$ и (независимо) сортировке файла $R_{i+1} \ldots R_N$. К каждому из этих подфайлов можно применить тот же самый метод.

В табл. 2 показано, как выбранный нами для примеров файл полностью сортируется при помощи этого метода за 11 стадий. В скобки заключены подфайлы, которые еще предстоит отсортировать; в машине эти подфайлы можно представлять двумя переменными l и r (границы рассматриваемого в данный момент файла) и стеком дополнительных пар (l_k, r_k). Каждый раз, когда файл подразделяется, мы помещаем в стек *больший* из подфайлов и начинаем обрабатывать оставшийся, и так до тех пор, пока не придем к тривиально коротким файлам; как показано в упр. 20, такая процедура гарантирует, что в стеке никогда не будет находиться более, чем примерно $\log_2 N$ элементов.

Таблица 2

"Быстрая сортировка"

																(l, r)	Стек
[503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703]	(1, 16)	—
[154	087	426	061	275	170]	503	[897	653	908	512	509	612	677	765	703]	(1, 6)	(8, 16)
[061	087]	154	[426	275	170]	503	[897	653	908	512	509	612	677	765	703]	(1, 2)	(4, 6)(8, 16)
061	087	154	[426	275	170]	503	[897	653	908	512	509	612	677	765	703]	(4, 6)	(8, 16)
061	087	154	[170	275]	426	503	[897	653	908	512	509	612	677	765	703]	(4, 5)	(8, 16)
061	087	154	170	275	426	503	[897	653	908	512	509	612	677	765	703]	(8, 16)	—
061	087	154	170	275	426	503	[703	653	765	512	509	612	677]	897	908	(8, 14)	—
061	087	154	170	275	426	503	[677	653	612	512	509]	703	765	897	908	(8, 12)	—
061	087	154	170	275	426	503	[509	653	612	512]	677	703	765	897	908	(8, 11)	—
061	087	154	170	275	426	503	509	[653	612	512]	677	703	765	897	908	(9, 11)	—
061	087	154	170	275	426	503	509	[512	612]	653	677	703	765	897	908	(9, 10)	—
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	—	—

Только что описанную процедуру сортировки можно назвать обменной сортировкой *с разделением;* она принадлежит Ч. Э. Р. Хоару, интереснейшая статья которого [*Comp. J.*, **5** (1962), 10–15]—одно из наиболее

исчерпывающих из когда-либо опубликованных сообщений об этом методе. Хоар окрестил свой метод "quicksort" ("быстрая сортировка"), и это название вполне соответствует действительности, так как, например, весь процесс сортировки, показанный в табл. 2, требует всего 48 сравнений (меньше любого другого встречавшегося уже метода, за исключением бинарных вставок, требующих 47 сравнений).

Picture: Рис 19. Обменная сортировка с разделением ("быстрая сортировка").

Во всех сравнениях на данной стадии участвует один и тот же ключ, так что его можно хранить в регистре. Кроме того, количество перемещений данных весьма умеренно: при вычислениях табл. 2 произведено всего 17 обменов, причем большинство из них—просто "полуобмены" (простые пересылки), так как один из ключей все время остается в регистре, и его не нужно записывать до самого конца стадии.

Вспомогательные операции (требуемые для управления стеком и переменными i, j) не сложны, но из-за них процедура быстрой сортировки посредством разделений пригодна в основном для больших значений N; поэтому короткие подфайлы желательно сортировать особом образом, как это делается в следующем алгоритме.

Алгоритм Q. *(Обменная сортировка с разделением.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$. Нужен вспомогательный стек для хранения не более чем $\log_2 N$ элементов. Этот алгоритм соответствует процедуре "быстрой сортировки" посредством разделений, приведенной выше, с небольшими изменениями в целях повышения эффективности:

a) Предполагается наличие искусственных ключей $K_0 = -\infty$ и $K_{N+1} = +\infty$, таких, что

$$K_0 \le K_i \le K_{N+1} \qquad \text{при } 1 \le i \le N. \tag{13}$$

(Равенство допускается.)

b) Подфайлы, состоящие из M и менее элементов, сортируются простыми вставками, где $M \ge 1$—параметр, который выбирается, как описано ниже.

c) На некоторых стадиях делается одно или два дополнительных сравнения (допускается перекрытие указателей i, j), чтобы основные циклы сравнения выполнялись настолько быстро, насколько это возможно.

d) Записи с одинаковыми ключами меняются местами, хотя это не является строго необходимым. (Эта идея, принадлежащая Р. К. Синглтону, способствует разделению подфайлов почти пополам, если имеются равные ключи; см. упр. 18.)

Q1 [Начальная установка.] Опустошить стек и установить $l \leftarrow 1$; $r \leftarrow N$.

Q2 [Начать новую стадию.] (Нам хотелось бы отсортировать файл $R_l, \ldots, R_r$; из самого существа алгоритма вытекает, что $r \ge l-1$, $K_{l-1} \le K_i \le K_{r+1}$ при $l \le i \le r$.) Если $r - l < M$, то перейти к шагу **Q8**. В противном случае установить $i \leftarrow l$, $j \leftarrow r$, $K \leftarrow K_l$, $R \leftarrow R_l$. (Обсуждение наилучшего выбора R и K приведено ниже.)

Q3 [Сравнить $K : K_j$.] Если $K < K_j$, то уменьшить j на 1 и повторить этот шаг.

Q4 [Переслать R на место R_i.] (К этому моменту K_i—ключ $\ge K$, находящийся не на своем месте, и $K \ge K_j$.) Если $j \le i$, то установить $R_i \leftarrow R$ и перейти к шагу Q7. В противном случае установить $R_i \leftarrow R_j$ и увеличить i на 1.

Q5 [Сравнить $K_i : K$.] Если $K_i < K$, то увеличить i на 1 и повторить этот шаг.

Q6 [Переслать R на место R_j.] (К этому моменту K_j—ключ $\le K$, находящийся не на своем месте, и $K \le K_i$.) Если $j \le i$, то установить $R_j \leftarrow R$ и $i \leftarrow j$. В противном случае установить $R_j \leftarrow R_i$, уменьшить j на 1 и перейти к шагу **Q3**.

Q7 [Поместить в стек.] (Теперь подфайл $R_l \ldots R_i \ldots R_r$ разделен так, что $K_k \le K_i$ при $l \le k \le i$ и $K_i \le K_k$ при $i \le k \le r$.) Если $r - i \ge i - l$, то поместить в стек $(i+1, r)$ и установить $r \leftarrow i-1$. В противном случае поместить в стек $(l, i-1)$ и установить $l \leftarrow i+1$. (Каждый элемент стека (a, b)—это заявка на сортировку подфайла $R_a \ldots R_b$ в одной из последующих стадий.) Вернуться к шагу **Q2**.

Q8 [Сортировка простыми вставками.] Для $j = l+1, l+2, \ldots$ до $j > r$ выполнять следующие операции: установить $K \leftarrow K_j$, $R \leftarrow R_j$, $i \leftarrow j-1$; затем установить $R_{i+1} \leftarrow R_i$, $i \leftarrow i-1$ нуль или более раз до тех пор, пока не выполнится условие $K_i \le K$; затем установить $R_{i+1} \leftarrow R$. (Это, по существу, алгоритм 5.2.1S, примененный к подфайлу из M или менее элементов.)

Q9 [Взять из стека.] Если стек пуст, то сортировка завершена; в противном случае взять верхний элемент стека (l', r'), установить $l \leftarrow l'$, $r \leftarrow r'$ и возвратиться к шагу **Q2**. ∎

Соответствующая `MIX`-программа довольно велика, но не сложна; на самом деле большая часть команд относится к шагу Q7, в котором проводятся весьма простые манипуляции с переменными.

Программа Q. *(Обменная сортировка с разделением.)* Записи, которые предстоит отсортировать, находятся в ячейках `INPUT` + 1, ..., `INPUT` + N; предполагается, что в ячейках `INPUT` и `INPUT` + N + 1 содержатся значения, соответственно минимально, и максимально допустимые в машине `MIX`. Стек располагается в ячейках `STACK` + 1, `STACK` + 2, ...; точное число ячеек, которое необходимо отвести под стек, обсуждается в упр. 20. Значения регистров: `rI1` $\equiv l$, `rI2` $\equiv r$, `rI3` $\equiv i$, `rI4` $\equiv j$, `rI6` $\equiv$ размер стека, `rA` $\equiv K \equiv R$.

```
A      EQU   2:3                Первая компонента элемента стека.
B      EQU   4:5                Вторая компонента элемента стека.
START  ENT1  1          1       Q1. Начальная установка. l ← 1.
       ENT2  N          1       r ← N.
       ENT6  0          1       Опустошить стек.
2H     ENTX  0,2        2A+1    Q2. Начать новую стадию.
       DECX  M,1        2A+1    rX ← r − l − M.
       JXN   8F         2A+1    К шагу Q8, если размер подфайла ≤ M.
       ENT3  0,1        A       i ← l.
       ENT4  0,2        A       j ← r.
       LDA   INPUT,3    A       K ← K_i.
       JMP   3F         A       К шагу Q3.
0H     LDX   INPUT,3    B
       STX   INPUT,4    B       R_j ← R_i.
       DEC4  1          C'−A    j ← j − 1.
3H     CMPA  INPUT,4    C'      Q3. Сравнить K : K_j.
       JL    *-2        C'      Если <, то уменьшить j и повторить.
4H     ENTX  0,3        B+A     Q4. Переслать R на место R_i.
       DECX  0,4        B+A
       JXNN  7F         B+A     К шагу Q7, если i ≥ j.
       LDX   INPUT,4    B+X
       STX   INPUT,3    B+X     R_i ← R_j.
       INC3  1          C''     i ← i + 1.
5H     CMPA  INPUT,3    C''     Q5. Сравнить K_i : K.
       JG    *-2        C''     Если <, то увеличить i и повторить.
6H     ENTX  0,3        B+X     Q6. Переслать R на место R_i.
       DECX  0,4        B+X
       JXN   0B         B+X     К шагу Q3, если i < j.
       ENT3  0,4        X       i ← j.
7H     STA   INPUT,3    A       Q7. Поместить в стек. R_i ← R.
       ENTA  0,2        A
       DECA  0,3        A
       DECA  0,3        A
       INCA  0,1        A
       INC6  1          A       rI6 ← rI6 + 1.
       JANN  1F         A       Переход, если r − i ≥ i − l.
       ST1   STACK,6(A) A'
       DEC3  1          A'
       ST3   STACK,6(B) A'      (l, i − 1)стек.
       ENT1  2,3        A'      l ← i + 1.
       JMP   2B         A'      К шагу Q2.
1H     INC3  1          A − A'
       ST3   STACK,6(A) A − A'
       ST2   STACK,6(B) A − A'  (i + 1, r)стек.
       ENT2  -2,3       A − A'  r ← i − 1.
       JMP   2B         A − A'  К шагу Q2.
8H     DEC2  0,1        A+1     Q8. Сортировка простыми вставками.
       J2NP  9F         A+1     К шагу Q9, если r ≤ l.
       ENT4  1,1        A+1−L   j ← l + 1.
1H     LDA   INPUT,4    D       R ← R_j.
       ENT3  -1,4       D       i ← j − 1.
       JMP   *+4        D       Переход и цикл.
```

```
        LDX   INPUT,3        E
        STX   INPUT+1,3      E      R_{i+1} ← R_i.
        DEC3  1              E      i ← i − 1.
        CMPA  INPUT,3        D+E
        JL    *-4            D+E    Переход, если K < K_i.
        STA   INPUT+1,3      D      R_{i+1} ← R.
        INC4  1              D      j ← j + 1.
        DEC2  1              D
        J2P   1B             D      Повторить r − l раз.
9H      LD1   STACK,6(A)     A+1    Q9. Взять из стека.
        LD2   STACK,6(B)     A+1    (l, r) ⇐ стек.
        DEC6  1              A+1    rI6 ← rI6 − 1.
        J6NN  2B             A+1    К шагу Q2, если стек не был пуст.
```

■

Анализ "быстрой сортировки". Информацию о времени выполнения, приведенную вместе с программой Q, нетрудно вывести из закона Кирхгофа, или закона сохранения (см. п. 1.3.3), и из того факта, что все помещенное в стек в конце концов оттуда извлекается. Общее время работы зависит от следующих величин:

$$\begin{aligned}
A &= \text{число стадий, на которых данный подфайл содержит более } M \text{ элементов;}\\
B &= \text{число присваиваний } R_j \leftarrow R_i \text{ в шаге Q6;}\\
C' &= \text{число выполнений шага Q3;}\\
C'' &= \text{число выполнений шага Q5;}\\
D &= \text{число присваиваний } R \leftarrow R_j \text{ в шаге Q8;}\\
E &= \text{число присваиваний } R_{i+1} \leftarrow R_i \text{ в шаге Q8;}\\
L &= \text{число случаев, когда в шаге Q8 } r \le l;\\
X &= \text{число операций } i \leftarrow j \text{ в шаге Q6.}
\end{aligned} \tag{14}$$

Мы не будем анализировать величины C' и C'' независимо, а рассмотрим лишь

$$C = C' + C'' = \text{общее число сравнений,} \tag{15}$$

так как время работы для C' и C'' одинаково. Изучив семь величин: A, B, C, D, E, L и X, мы сможем разумно выбрать параметр M, которым определяется "порог" между сортировкой простыми вставками и разделением. (Если читатель не математик, он может пропустить все вплоть до формул (25).)

Как и при анализе большинства других алгоритмов в этой главе, мы будем предполагать, что сортируемые ключи различны;. в упр. 18 показано, что наличие равных ключей не снижает существенно эффективности алгоритма Q; в действительности присутствие равных ключей, по-видимому, даже способствует ее увеличению. Так как метод зависит только от относительного расположения ключей, то можно предполагать также, что это просто числа $\{1, 2, \ldots, N\}$, расположенные в некотором порядке.

Когда мы впервые достигаем шага Q2, сортировка сводится к простым вставкам, если $N \le M$, а это мы уже изучили; поэтому предположим, что $N > M$. Необходимо лишь разобраться в вычислениях, которые в первый раз приводят к шагу Q7; нетрудно видеть, что при разделении записи в обоих подфайлах $R_1 \ldots R_{i-1}$ и $R_{i+1} \ldots R_N$ будут расположены в случайном порядке, если только записи исходного файла были расположены в случайном порядке. Значит, вклад последующих вычислений можно определить, применив индукцию по N.

Пусть s—значение первого ключа K_1, и предположим, что ровно t из ключей $K_1, \ldots, K_s$ превосходят s. Пусть

$$h = \begin{cases} 1, & \text{если } K_s < s;\\ 0, & \text{если } K_s \ge s. \end{cases} \tag{16}$$

Иначе говоря, $h = 1$, если ключ, который вначале занимал позицию s (куда попадет ключ K_1), будет перемещен влево. Напомним, что сортируемые ключи являются целыми числами из множества $\{1, 2, \ldots, N\}$.

Если $s = 1$, то нетрудно видеть, что произойдет во время первой стадии разделения. Шаг Q3 выполнится N раз, после чего шаг Q4 приведет нас к Q7. Таким образом, первая стадия в этом случае даст такие вклады: $A = 1$, $B = 0$, $C = N$, $X = 0$. Аналогичные, но несколько более сложные рассуждения для случая $s > 1$ (см. упр. 21) показывают, что вкладами первой стадии в суммарное время выполнения в общем случае будут

$$A = 1, \quad B = t, \quad C = N + 1 - \delta_{s1}, \quad X = h \qquad \text{при } 1 < s \le N. \tag{17}$$

К этому необходимо добавить вклады доследующих стадий, во время которых упорядочиваются подфайлы соответственно из $s-1$ и $N-S$ элементов.

Если предположить, что в исходном файле записи располагались в случайном порядке, то можно выписать формулы, которыми определяются производящие функции для распределений вероятностей величин A, B, ..., X (см. упр. 22). Но для простоты мы изучим здесь только *средние значения* этих величин A_N, B_N, ..., X_N как функции от N. Рассмотрим, например, среднее число сравнений C_N, выполненных в процессе разделения. Если $N \le M$, то $C_N = 0$. При $N > M$, так как любое данное значение s встречается с вероятностью $1/N$,

$$\begin{aligned} C_N &= \frac{1}{N}\sum_{1\le s\le N}(N+1-\delta_{s1}+C_{s-1}+C_{N-s}) = \\ &= N+1-\frac{1}{N}+\frac{2}{N}\sum_{0\le k<N}C_k. \end{aligned} \tag{18}$$

Аналогичные формулы имеют место и для остальных величин A_N, B_N, ..., X_N (см. упр. 23).

Существует простой способ решения рекуррентных соотношений вида

$$x_n = f_n + \frac{2}{n}\sum_{0\le k<n} x_k \qquad \text{при } n \ge m. \tag{19}$$

На первом шаге освобождаются от знака суммирования: поскольку

$$\begin{aligned} (n+1)x_{n+1} &= (n+1)f_{n+1} + 2\sum_{0\le k\le n} x_k, \\ nx_n &= nf_n + 2\sum_{0\le k<n} x_k, \end{aligned}$$

то можно вычесть второе равенство из первого, получив

$$(n+1)x_{n+1} - nx_n = g_n + 2x_n, \qquad \text{где } g_n = (n+1)f_{n+1} - nf_n.$$

Теперь рекуррентная зависимость принимает гораздо более простой вид:

$$(n+1)x_{n+1} = (n+2)x_n + g_n \qquad \text{при } n \ge m. \tag{20}$$

Любое рекуррентное соотношение общего вида

$$a_n x_{n+1} = b_n x_n + g_n \tag{21}$$

можно свести к простой сумме, если умножить обе части на суммирующий множитель $a_0a_1\ldots a_{n-1}/b_0b_1\ldots b_n$. Получаем

$$y_{n+1} = y_n + c_n, \qquad \text{где } y_n = \frac{a_0\ldots a_{n-1}}{b_0\ldots b_{n-1}}x_n,\ c_n = \frac{a_0\ldots a_{n-1}}{b_0\ldots b_n}g_n. \tag{22}$$

В случае (20) суммирующий множитель равен просто $n!/(n+2)! = 1/(n+1)(n+2)$. Таким образом, находим, что простое соотношение

$$\frac{x_{n+1}}{n+2} = \frac{x_n}{n+1} + \frac{(n+1)f_{n+1}-nf_n}{(n+1)(n+2)}, \qquad n \ge m, \tag{23}$$

является следствием соотношения (19).

Если, например, положить $f_n = 1/n$, то получится неожиданный результат $x_n/(n+1) = x_m/(m+1)$ при любом $n \ge m$. Если положить $f_n = n+1$, то получится

$$\begin{aligned} x_n/(n+1) &= 2/(n+1) + 2/n + \cdots + 2/(m+2) + x_m/(m+1) = \\ &= 2(H_{n+1} - H_{m+1}) + x_m/(m+1) \end{aligned}$$

при любом $n \ge m$. Комбинация этих двух решений и подстановка $m = M+1$ и $x_n = 0$ при $n \le M$ дают формулу

$$\begin{aligned} C_N &= (N+1)\left(2H_{N+1} - 2H_{M+2} + 1 - \frac{1}{(M+1)(M+2)}\right) \approx \\ &\approx 2(N+1)\ln\left(\frac{N+1}{M+2}\right) \qquad \text{при } N > M. \end{aligned} \tag{24}$$

В п. 6.2.2 мы докажем, что стандартное отклонение величины C_N асимптотически равно $\sqrt{(21-2\pi^2)}/3N$; это довольно мало по сравнению с (24).

Остальные величины можно найти аналогичным способом (см. упр. 23); имеем

$$\begin{aligned} A_N &= 2(N+1)/(M+2) - 1, \\ B_N &= \frac{1}{6}(N+1)\left(2H_{N+1} - 2H_{M+2} + 1 - \frac{6}{M+2}\right) + \frac{1}{2}, \\ D_N &= (N+1)M(M-1)/(M+2)(M+1), \\ E_N &= \frac{1}{6}(N+1)M(M-1)/(M+2), \\ L_N &= 4(N+1)/(M+2)(M+1), \\ X_N &= (N+1)/(M+2) - \frac{1}{2} \qquad \text{при } N > M. \end{aligned} \tag{25}$$

Приведенное выше обсуждение показывает, что можно произвести точный анализ среднего времени выполнения весьма сложной программы, используя методы, которые мы ранее применяли лишь к более простым случаям.

Чтобы определить "наилучшее" значение M для конкретной машины, можно воспользоваться формулами (24) и (25). Программа Q для машины `MIX` требует $37A+14B+4C+12D+8E-L+8X+15$ единиц времени; это равно в среднем $\frac{1}{3}(38(N+1)H_N + (N+1)f(M)) - 19$ единиц при $N > M$, где

$$f(M) = 4M + 38H_{M+2} + 43 + \frac{84}{M+2} + \frac{48}{(M+2)(M+1)}. \tag{26}$$

Мы хотим выбрать такое значение M, при котором функция $f(M)$ достигает минимума. В данном случае

$$f(M) - f(M-1) = 4 - \frac{38}{M+2} - \frac{84}{(M+2)(M+1)} - \frac{96}{(M+2)(M+1)M},$$

и требуется найти такое значение M, чтобы $f(M) - f(M-1) \le 0$, $f(M+1) - f(M) \ge 0$; решение $M = 9$ найти нетрудно. Если $M = 9$, то при больших N среднее время выполнения программы Q равно приблизительно $12.67(N+1)\ln N - 1.92N - 14.59$.

Таким образом, программа Q работает в среднем довольно быстро; следует, кроме того, учесть, что она требует очень мало памяти. Но каков *наихудший* случай для алгоритма Q? Существуют ли какие-нибудь исходные файлы, обрабатывать которые этим алгоритмом не эффективно? Ответ несколько обескураживает: если исходный файл уже упорядочен, а именно $K_1 < K_2 < \ldots < K_N$, то каждая операция "разделения" становится почти бесполезной, так как подфайл сводится к одному элементу! Значит, в этой ситуации (которая должна быть самой простой для сортировки) быстрая сортировка превращается в отнюдь не быструю. Время ее работы становится пропорциональным N^2, а не $N \log N$ (см. упр. 25). В отличие от других алгоритмов сортировки, которые нам встречались, алгоритм Q предпочитает неупорядоченные файлы!

В упомянутой статье Хоара предложены два способа поправить ситуацию, основывающихся на выборе лучшего значения проверяемого ключа K, который управляет разделением. Одна из его рекомендаций состоит .в том, чтобы в последней части шага Q2 выбирать *случайное* целое число q между l и r; в этом шаге можно заменить инструкции "$K \leftarrow K_l$, $R \leftarrow R_l$" на

$$K \leftarrow K_q, \quad R \leftarrow R_q, \quad R_q \leftarrow R_l. \tag{27}$$

Согласно формулам (25), такие случайные целые числа придется вычислять в среднем лишь $2(N+1)/(M+2) - 1$ раз, так что дополнительное время работы несущественно, а случайный выбор—хорошая защита от опасности оказаться в наихудшей ситуации.

Второе предложение Хоара состоит в том, чтобы просмотреть небольшой участок файла и найти медиану для этой совокупности данных. Такому подходу последовал Р. К. Синглтон [*CACM*, **12** (1969), 185–187], который предложил в качестве K_q брать медиану трех значений

$$K_l, \quad K_{\lfloor (l+r)/2 \rfloor}, \quad K_r. \tag{28}$$

Процедура Синглтона сокращает число сравнений с $2N\ln N$ примерно до $\frac{12}{7}N\ln N$ (см. упр. 29). Можно показать, что. в этом случае B_N асимптотически приближается к $C_N/5$, а не к $C_N/6$, так что метод медианы несколько увеличивает время, затрачиваемое на пересылку данных, поэтому общее время

работы сокращается примерно, на 8%. (Подробный анализ см. в упр.56.) Время работы в наихудшем случае все еще порядка N^2, однако с таким медленным поведением алгоритма вряд ли когда-либо придется встретиться на практике.

У. Д. Фрэйзер и А. Ч. Мак-Келлар [*JACM*, **17** (1970), 496–507] предложили рассматривать совокупность гораздо большего объема из $2^k - 1$ записей, где k выбирается так, чтобы $2^k \approx N/\ln N$. Эту совокупность можно отсортировать обычным методом быстрой сортировки, после чего элементы вставляются среди остальных записей за k просмотров всего файла (в результате файл будет разделен на 2^k подфайлов, ограниченных элементами первоначальной совокупности). На заключительном этапе сортируются полученные подфайлы. Среднее число, сравнений, выполняемых такой процедурой "сортировки совокупности", примерно такое же, как и для метода медианы Синглтона, когда N находится в практическом диапазоне значений, но при $N \to \infty$ оно асимптотически приближается к $N \log_2 N$.

Обменная поразрядная сортировка. Мы подходим теперь к методу, совершенно отличному от всех схем сортировки, которые рассматривались прежде; в нем используется *двоичное представление* ключей, и.потому он предназначен исключительно для двоичных машин. Вместо того чтобы сравнивать между собой два ключа, в этом методе проверяется, равны ли 0 или 1 отдельные биты ключа. В других отношениях он обладает характеристиками обменной сортировки и на самом деле очень напоминает быструю сортировку. Так как он зависит от разрядов ключа, представленного в двоичной системе счисления, мы называем его "обменной поразрядной сортировкой". В общих чертах этот алгоритм можно описать следующим образом:

1) Последовательность сортируется *по старшему значащему двоичному биту* так, чтобы все ключи, начинающиеся с 0, оказались перед всеми ключами, начинающимися с 1. Для этого надо найти самый левый ключ K_i, начинающийся с 1, и самый правый ключ K_j, начинающийся с 0, после чего R_i и R_j меняются местами, и процесс повторяется, пока не станет $i > j$.
2) Пусть F_0—множество элементов, начинающихся с 0, а F_1—все остальные. Применим к F_0 поразрядную сортировку (начав теперь со *второго* бита слева, а не со старшего) до тех пор, пока множество F_0 .не будет полностью отсортировано; затем проделаем то же с F_1.

Например, в табл. 3 показано, как действует обменная поразрядная сортировка на наши 16 случайных чисел, записанных теперь в восьмеричной системе счисления. На стадии 1 показан исходный файл; после обменов по первому биту приходим ко второй стадии. На второй стадии сортируется первая группа по второму, биту, на третьей—по третьему биту. (Читатель должен мысленно преобразовать восьмеричные числа в 10-разрядные двоичные.) Когда мы после сортировки по .четвертому биту достигаем пятой стадии, то обнаруживаем, что каждая из оставшихся групп содержит всего по одному элементу,- так что эту часть файла можно больше не рассматривать. Запись "^{4}[0232 0252]" означает, что подфайл 0232 0252 еще предстоит сортировать по четвертому биту слева. В этом конкретном случае сортировка по четвертому биту не дает ничего нового; чтобы разделить элементы, необходимо добраться до пятого бита.

Весь процесс сортировки, показанный в табл. 3, выполняется за 22 стадии; это несколько больше соответствующего числа в быстрой сортировке (табл. 2). Число проверок битов 82 также велико; но мы увидим, что число проверок битов при больших 151

Picture: Таблица 3

N в действительности меньше, чем число сравнений в быстрой сортировке, в предположении о равномерном распределении ключей. Общее число обменов в табл. 3 равно 17, т. е. весьма умеренно. Заметим, что, хотя сортируются 10-битовые числа, в данном примере при проверке битов никогда не приходится идти дальше седьмого бита.

Как и при быстрой сортировке, для хранения "информации о границах" подфайлов, ожидающих сортировки, можно воспользоваться стеком. Вместо того чтобы сортировать в первую очередь наименьший из подфайлов, удобно просто продвигаться слева направо, так как размер стека в. этом случае никогда не превзойдет числа битов в сортируемых ключах. В алгоритме, приведенном ниже, элемент стека (r, b) указывает на то, что подфайл с правой границей r ожидает сортировки по биту b; левую границу можно не запоминать в стеке: она всегда задана неявно, поскольку в этой процедуре файл всегда обрабатывается слева направо.

Алгоритм R. *(Обменная поразрядная сортировка.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$. Предполагается, что все ключи—m-разрядные двоичные числа $(a_1\ a_2\ \ldots\ a_m)_2$; i-й по старшинству бит a_i называется "бит i" ключа. Требуется вспомогательный стек, вмещающий до $m - 1$ элементов. Этот алгоритм, по

существу, следует процедуре обменной поразрядной сортировки с разделениями, описанной выше; возможны некоторые усовершенствования с целью повышения эффективности (они описаны далее в тексте и в упражнениях).

R1 [Начальная установка.] Опустошить стек и установить $l \leftarrow 1$, $r \leftarrow N$, $b \leftarrow 1$.

R2 [Начать новую стадию.] (Мы хотели бы теперь отсортировать подфайл $R_l \leq \ldots \leq R_r$ по биту b; по смыслу алгоритма $l \leq r$.) Если $l = r$, то перейти к шагу **R10** (так как файл, состоящий из одного слова, уже отсортирован). В противном случае установить $i \leftarrow l$, $j \leftarrow r$.

R3 [Проверить K_i на 1.] Проверить бит b ключа K_i.. Если он равен 1, то перейти к шагу **R6**.

R4 [Увеличить i.] Увеличить i на 1. Если $i \leq j$, то возвратиться к шагу **R3**; в противном случае перейти к шагу **R8**.

R5 [Проверить K_{j+1} на 0.] Проверить бит b ключа K_{j+1}. Если он равен 0, то перейти к шагу **R7**.

R6 [Уменьшить j.] Уменьшить j на 1. Если$i \leq j$, то перейти к шагу **R5**; в противном случае перейти к шагу **R8**.

R7 [Поменять местами R_i, R_{j+1}]. Поменять местами $R_i \leftrightarrow R_{j+1}$, затем перейти к шагу **R4**.

R8 [Проверить особые случаи.] (К этому моменту стадия разделения завершена, $i = j + 1$, бит b ключей K_l, ..., K_j равен 0, а бит b ключей K_i, ..., K_r равен 1.) Увеличить b на 1. Если $b > m$, где m—общее число битов в ключах, то перейти к шагу **R10**. (Это означает, что подфайл $R_l \ldots R_r$ отсортирован. Если в файле не может быть равных ключей, то такую проверку можно не делать.) В противном случае, если $j < l$ или $j = r$, возвратиться к шагу **R2** (все просмотренные биты оказались равными соответственно 1 или 0). Если же $j = l$, то увеличить l на 1 и перейти к шагу **R2** (встретился только один бит, равный 0).

R9 [Поместить в стек.] Поместить в стек элемент (r, b), затем установить $r \leftarrow j$ и перейти к шагу **R2**.

R10 [Взять из стека.] Если стек пуст, то сортировка закончена. В противном случае установить $l \leftarrow r+1$, взять из. стека элемент (r', b'), установить $r \leftarrow r'$, $b \leftarrow b'$ и возвратиться к шагу **R2**. ∎

Программа R. *(Обменная поразрядная сортировка.)* В следующей программе для машины `MIX` используются, по существу, те же соглашения, что и в программе Q. Значения регистров: `rI1` $\equiv l - r$, `rI2` $\equiv r$, `rI3` $\equiv i$, `rI4` $= j$, `rI5` $\equiv m - b$, `rI6` = размер стека, если не считать того, что в некоторых командах (отмеченных ниже) удобно оставить `rI3` $= i - j$ или `rI4` $= j - i$. Двоичной природой поразрядной сортировки объясняется использование в этой программе команд `SRB` (двоичный сдвиг содержимого `AX` вправо), `JAE` (переход, если содержимое `A` четно) и `JAO` (переход, если содержимое `A` нечетно), определенных в п. 4.5.2. Предполагается, что $N \geq 2$.

```
START ENT6  0                 1        R1. Начальная установка. Опустошить стек.
      ENT1  1-N               1        l ← 1.
      ENT2  N                 1        r ← N.
      ENT5  M-1               1        b ← 1.
      JMP   1F                1        К R2 (опустить проверку l = r).
9H    INC6  1                 S        R9. Поместить в стек. [rI4 = l − j]
      ST2   STACK,6(A)        S
      ST5   STACK,6(B)        S        (r, b) ⇒ стек.
      ENN1  0,4               S        rI1 ← l − j.
      ENT2  -1,3              S        r ← j.
1H    ENT3  0,1               A        R2. Начать новую стадию. [rI3 = i − j]
      ENT4  0,2               A        i ← l, j ← r. [rI3 = i − j]
3H    INC3  0,4               C'       R3. Проверить K_i на 1.
      LDA   INPUT,3           C'
      SRB   0,5               C'       младший бит rA ← бит b ключа K_i.
      JAE   4F                C'       К R4, если он равен 0.
6H    DEC4  1,3               C''+X    R6. Уменьшить j ← j − l. [rI4 = j − i].
      J4N   8F                C''+X    К R8, если j < i. [rI4 = j − i]
5H    INC4  0,3               C''      R5. Проверить K_{j+1} на 0.
      LDA   INPUT+1,4         C''
      SRB   0,5               C''      младший бит rA ← бит b ключа K_{j+1}.
      JAO   6B                C''      К R6, если он равен 1.
7H    LDA   INPUT+1,4         B        R7. Поменять местами R_i, R_{j+1}.
      LDX   INPUT,3           B
      STX   INPUT+1,4         B
      STA   INPUT,3           B
```

```
4H    DEC3  -1,4            C'-X      R4. Увеличить i. i ← i+1. [rI3 = i-j]
      J3NP  3B              C'-X      К R3, если i ≤ j. [rI3 = i-j]
      INC3  0,4             A-X       rI3 ← i.
8H    J5Z   0F              A         R8. Проверить особые случаи. [rI4 = неизвестен]
      DEC5  1               A-G       К R10, если b = m, иначе b ← b-1.
      ENT4  -1,3            A-G       rI4 ← j.
      DEC4  0,2             A-G       rI4 ← j-r.
      J4Z   1B              A-G       К R2, если j = r.
      DEC4  0,1             A-G-R     rI4 ← j-l.
      J4N   1B              A-G-R     К R2, если j < l.
      J4NZ  9B              A-G-L-R   К R9, если j ≠ l.
      INC1  1               K         l ← l+1.
2H    J1NZ  1B              K+S       Переход, если l ≠ r.
0H    ENT1  1,2             S+1       R10. Взять из стека.
      LD2   STACK, 6 (A)    S+1
      DEC1  0,2             S+1
      LD5   STACK,6 (B)     S+1       Стек ⇒ (r,b)
      DEC6  1               S+1
      J6NN  2B              S+1       К R2, если стек не был пуст.
```

Время работы программы обменной поразрядной сортировки зависит от

$$\begin{aligned}
A &= \text{число стадий, в которых } l < r;\\
B &= \text{число замещений;}\\
C &= C' + C'' = \text{число проверок битов;}\\
G &= \text{число случаев, когда в шаге R8 } b > m;\\
K &= \text{число случаев, когда в шаге R8 } b \le m,\ j = l;\\
L &= \text{число случаев, когда в шаге R8 } b \le m,\ j < l;\\
R &= \text{число случаев, когда в шаге R8 } b \le m,\ j = r;\\
S &= \text{число случаев, когда что-либо помещается в стек;}\\
X &= \text{число случаев, когда в шаге R6 } j < i.
\end{aligned} \tag{29}$$

По закону Кирхгофа $S = A - G - K - L - R$, так что общее время выполнения равно $27A + 8B + 8C - 23G - 14K - 17L - 19R - X + 13$ единиц. За счет усложнения программы можно несколько ускорить циклы проверки битов (см. упр. 34). Обменную поразрядную сортировку можно также ускорить, если при достаточно малых значениях разности $r - l$ применять простые вставки, как это сделано в алгоритме Q, но мы не будем задерживаться на таких усовершенствованиях.

Для анализа времени выполнения обменной поразрядной сортировки напрашиваются два типа исходных данных:

i) Пусть $N = 2^m$ и ключи, которые надо отсортировать,— просто числа 0, 1, 2, ..., $2^m - 1$, расположенные в случайном порядке.

ii) Пусть $m = \infty$ (неограниченная точность) и ключи, которые надо отсортировать,—независимые, равномерно распределенные в промежутке $[0, 1)$ действительные числа.

Анализ случая (i) относительно прост, поэтому он оставлен читателю в качестве упражнения (см. упр. 35). Случай (ii) сравнительно сложен, поэтому он *также* оставлен для упражнений. В следующей таблице приведены грубые оценки результатов того и другого анализов:

Величина	**Случай (i)**	**Случай (ii)**
A	N	αN
B	$\frac{1}{4}N\log_2 N$	$\frac{1}{4}N\log_2 N$
C	$N\log_2 N$	$N\log_2 N$
G	$\frac{1}{2}N$	0
K	0	$\frac{1}{2}N$
L	0	$\frac{1}{2}(\alpha-1)N$
R	0	$\frac{1}{2}(\alpha-1)N$
S	$\frac{1}{2}N$	$\frac{1}{2}N$
X	$\frac{1}{2}N$	$\frac{1}{4}(\alpha-1)N$

(30)

Здесь $\alpha = 1/(\ln 2) = 1.4427$. Заметим, что среднее число обменов, проверок битов и обращений к стеку, по существу, одинаково в обоих случаях, несмотря даже на то, что в случае (ii) число стадий на 44% больше. На сортировку N элементов в случае (ii) наша `MIX`-программа затратит в среднем приблизительно $14.4N \ln N$ единиц времени. Это число можно было бы сократить примерно до $11.5N \ln N$, если воспользоваться предложением из упр. 34. Соответствующая величина для программы Q равна $12.7N \ln N$, но и ее можно уменьшить до $11.7N \ln N$, если воспользоваться предложением Синглтона и выбирать медиану из трех ключей.

В случае равномерного распределения данных обменная поразрядная сортировка занимает в среднем примерно столько же времени, сколько и быстрая сортировка; фактически для машины `MIX` она немного быстрее, чем быстрая сортировка. В упр. 53 показано, в какой мере замедляется этот процесс при неравномерном распределении. Важно отметить, что весь наш анализ основан на предположении о том. что все ключи различны. *Обменная поразрядная сортировка в том виде, как она описана выше, не очень эффективна, если имеются одинаковые ключи,* так как она проходит через несколько стадий, требующих времени, пытаясь разделить множества одинаковых ключей, пока b не станет $> m$. Один приемлемый способ исправить этот недостаток предложен в ответе к упр. 40.

Как обменная поразрядная сортировка, так и быстрая сортировка основаны на идее разделения. Записи меняются местами до тех пор, пока файл не будет разбит на две части: левый подфайл, в котором все ключи $\le K$ при некотором K, и правый подфайл, в котором все ключи $\ge K$. В быстрой сортировке в качестве K выбирается реальный ключ из файла, в то время как в обменной поразрядной сортировке, по существу, выбирается некоторый искусственный ключ на основе двоичных представлений. Что касается исторической стороны дела, то обменную поразрядную сортировку открыли П. Хильдебрандт, Г. Исбитц, Х. Райзинг и Ж. Шварц [*JACM*, **6** (1959), 156–163] примерно за год до изобретения быстрой сортировки. Возможны также и другие схемы разделения; например, Джон Маккарти предложил выбирать $K \approx \frac{1}{2}(u+v)$, если известно, что все ключи лежат в диапазоне между u и v.

Еще одну стратегию разделения предложил М. Х. ван Эмден [*CACM*, **13** (1970), 563–567]: вместо того чтобы выбирать K заранее, мы "узнаем", каково может быть хорошее значение K, следя в процессе разделения за изменением величин $K' = \max(K_l, \ldots, K_i)$ и $K'' = \min(K_j, \ldots, K_r)$. Можно увеличивать i до тех пор, пока не встретится ключ, больший K'; затем начать уменьшать j, пока не встретится ключ, меньший K'', после чего поменять их местами и/или уточнить значения K' и K''. Эмпирические тесты для этой интервальной сортировки показывают, что она требует около $1.64N \ln N = 1.14N \log_2 N$ сравнений. Это единственный метод, обсуждаемый в этой книге, для поведения которого еще не найдено адекватного теоретического объяснения.

Обобщение обменной поразрядной сортировки на случай системы счисления с основанием, большим 2, обсуждается в п. 5.2.5.

*** Асимптотические методы.** Анализ алгоритмов обменной сортировки приводит к некоторым особенно поучительным математическим задачам, которые позволяют больше узнать о способах определения асимптотического поведения функции. Например, при анализе метода пузырька [формула (9)] мы столкнулись с функцией

$$W_n = \frac{1}{n!} \sum_{0 \le r < s \le n} s! r^{n-s}. \tag{31}$$

Каково ее асимптотическое значение?

Можно действовать так же, как при исследовании числа инволюций [формула (5.1.4–41)]. Поэтому, прежде чем двигаться дальше, полезно еще раз просмотреть обсуждение в конце п. 5.1.4.

Исследование формулы (31) показывает, что вклад при $s = n$ больше, чем вклад при $s = n-1$, и т. д.; это наводит на мысль о замене s на $n - s$. Мы скоро увидим, что на самом деле удобнее всего применить подстановки $t = n - s + 1$, $m = n + 1$, в результате которых формула (31) примет вид

$$\frac{1}{m} W_{m-1} = \frac{1}{m!} \sum_{1 \le t < m} (m-t)! \sum_{0 \le r < m-t} r^{t-1}. \tag{32}$$

Для внутренней суммы существует хорошо известный асимптотический ряд, полученный из формулы суммирования Эйлера:

$$\begin{aligned} \sum_{0 \le r < N} r^{t-1} &= \frac{N^t}{t} - \frac{1}{2}(N^{t-1} - \delta_{t1}) + \frac{B_2}{2!}(t-1)(N^{t-2} - \delta_{t2}) + \cdots = \\ &= \frac{1}{t} \sum_{0 \le j \le k} \binom{t}{j} B_j (N^{t-j} - \delta_{tj}) + O(N^{t-k}) \end{aligned} \tag{33}$$

(см. упр. 1.2.11.2–4). Следовательно, наша задача сводится к изучению сумм вида

$$\frac{1}{m!}\sum_{1\le t<m}(m-t)!(m-t)^t\,t^k,\quad k\ge -1. \tag{34}$$

Как и в п. 5.1.4, можно показать, что при значениях t, бóльших $m^{1/2+\varepsilon}$, члены суммы пренебрежимо малы: $O(\exp(-n^\delta))$. Значит, можно положить $t=O(m^{1/2+\varepsilon})$ и заменить факториалы по формуле Стирлинга

$$\frac{(m-t)!(m-t)^t}{m!}=\sqrt{1-\frac{t}{m}}\exp\left(\frac{t}{12m^2}-\left(\frac{t^2}{2m}+\frac{t^3}{3m^2}+\frac{t^4}{4m^3}+\frac{t^5}{5m^4}\right)+O(m^{-2+6\varepsilon})\right).$$

Таким образом, нас интересует асимптотическое значение суммы

$$r_k(m)=\sum_{1\le t<m}e^{-t^2/2m}t^k,\quad k\ge -1. \tag{35}$$

(Суммирование здесь также можно было бы распространить на весь диапазон $1\le t<\infty$, не изменив при этом асимптотического значения суммы, поскольку, как указано выше, входящие в сумму значения при $t>m^{1/2+\varepsilon}$ пренебрежимо малы.)

Пусть $g_k(x)=x^ke^{-x^2}$ и $f_k(x)=g_k(x/\sqrt{2m})$. По формуле суммирования Эйлера при $k\ge 0$

$$\begin{aligned}\sum_{0\le t<m}f_k(t)&=\int_0^m f_k(x)\,dx+\sum_{1\le j\le p}\frac{B_j}{j!}(f_k^{(j-1)}(m)-f_k^{(j-1)}(0))+R_p,\\ R_p&=\frac{(-1)^{p+1}}{p!}\int_0^m B_p(\{x\})f_k^{(p)}(x)\,dx=\\ &\left(\frac{1}{\sqrt{2m}}\right)^p O\left(\int_0^\infty|g_k^{(p)}(y)|\,dy\right)=O(m^{-p/2});\end{aligned} \tag{36}$$

следовательно, при помощи, по существу, тех же приемов, что и раньше, можно получить асимптотический ряд для $r_k(m)$, если только $k\ge 0$. Но при $k=-1$ этот метод не годится, так как значение $f_{-1}(0)$ не определено; нельзя также просто просуммировать от 1 до m, так как остаточные члены не дают убывающих степеней m, если нижний предел равен 1. (Именно в этом суть дела, и, прежде чем двигаться дальше, читатель должен убедиться в том, что он хорошо понял задачу.)

Чтобы разрешить эту дилемму, можно положить по определению $g_{-1}(x)=(e^{-x^2}-1)/x$, $f_{-1}(x)=g_{-1}(x/\sqrt{2m})$; тогда $f_{-1}(0)=0$ и $r_{-1}(m)$ нетрудно получить из $\sum_{0\le t<m}f_{-1}(t)$. Уравнение (36) справедливо теперь и при $k=-1$, а оставшийся интеграл нам "хорошо знаком" (см. упр. 43):

$$\begin{aligned}\frac{2}{\sqrt{2m}}\int_0^m f_{-1}(x)\,dx&=2\int_0^m\frac{e^{-x^2/2m}-1}{x}\,dx=\int_0^{m/2}\frac{e^{-y}-1}{y}\,dy=\\ &=\int_0^1\frac{e^{-y}-1}{y}\,dy+\int_1^{m/2}\frac{e^{-y}}{y}\,dy-\ln(m/2)=\\ &=-\gamma-\ln(m/2)+O(e^{-m/2}).\end{aligned}$$

Теперь у нас достаточно формул и фактов для того, чтобы вывести наконец ответ, который равен, как показано в ответе к упр 44,

$$W_n=\frac{1}{2}m\ln m+\frac{1}{2}(\gamma+\ln 2)m-\frac{2}{3}\sqrt{2\pi m}+\frac{31}{36}+O(n^{-1/2}),\quad m=n+1. \tag{37}$$

Этим завершается анализ метода пузырька.

Чтобы проанализировать обменную поразрядную сортировку, необходимо знать асимптотическое поведение при $n\to\infty$ конечной суммы

$$U_n=\sum_{k\ge 2}\binom{n}{k}(-1)^k\frac{1}{2^{k-1}-1}. \tag{38}$$

Этот вопрос оказывается гораздо сложнее других задач об асимптотическом поведении, с которыми мы сталкивались до сих пор; элементарные методы разложения в степенные ряды, формула суммирования Эйлера и т. д. здесь бессильны. Следующий вывод был предложен Н. Г. де Брейном.

Чтобы избавиться в формуле (38) от подавляющего влияния больших множителей $\binom{n}{k}(-1)^k$, мы начнем с того, что перепишем сумму в виде бесконечного ряда:

$$U_n = \sum_{k\ge 2}\binom{n}{k}(-1)^k\sum_{j\ge 1}\left(\frac{1}{2^k-1}\right)^j = \sum_{j\ge 1}(2^j(1-2^{-j})^n - 2^j + n). \tag{39}$$

Если положить $x = n/2^j$, то член ряда запишется в виде

$$2^j(1-2^{-j})^n - 2^j + n = \frac{n}{x}\left(\left(1-\frac{x}{n}\right)^n - 1 + x\right).$$

При $x \le n^\varepsilon$ имеем

$$\left(1-\frac{x}{n}\right)^n = \exp\left(n\ln\left(1-\frac{x}{n}\right)\right) = \exp(-x + x^2O(n^{-1})), \tag{40}$$

а это наводит на мысль о том, чтобы приблизить (39) рядом

$$T_n = \sum_{j\ge 1}(2^je^{-n/2^j} - 2^j + n). \tag{41}$$

Чтобы оправдать это приближение, рассмотрим разность $U_n - T_n = X_n + Y_n$, где

$$\begin{aligned}
X_n &= \sum_{\substack{j\ge 1\\ 2^j<n^{1-\varepsilon}}}(2^j(1-2^{-j})^n - 2^je^{-n/2^j}) = && \text{[т.е. слагаемые при } x > n^\varepsilon\text{]}\\
&= \sum_{\substack{j\ge 1\\ 2^j<n^{1-\varepsilon}}} O(ne^{-n/2^j}) = && \text{[так как } 0 < 1-2^{-j} < e^{-2^{-j}}\text{]}\\
&= O(n\log n e^{-n^\varepsilon}) && \text{[так как имеется } O(\log n)\text{слагаемых];}\\
Y_n &= \sum_{\substack{j\ge 1\\ 2^j\ge n^{1-\varepsilon}}}(2^j(1-2^{-j})^n - 2^je^{-n/2^j}) = && \text{[слагаемые при } x \le n^{-\varepsilon}\text{]}\\
&= \sum_{\substack{j\ge 1\\ 2^j\ge n^{1-\varepsilon}}}\left(e^{-n/2^j}\left(\frac{n}{2^j}\right)O(1)\right) && \text{[по формуле (40)]}
\end{aligned}$$

Приведенное ниже рассуждение покажет, что эта последняя сумма есть $O(1)$; следовательно, $U_n - T_n = O(1)$. (См. упр. 47.)

До сих пор мы еще не использовали никаких методов, которые бы действительно отличались от применявшихся ранее, но

Picture: Рис. 20. Контуры интегрирования для тождеств с гамма-функциями.

для изучения ряда T_n потребуется новая идея, основанная на простых принципах теории функций комплексного переменного. Если x—произвольное положительное число, то

$$e^{-x} = \frac{1}{2\pi i}\int_{1/2-i\infty}^{1/2+i\infty}\Gamma(z)x^{-z}\,dz = \frac{1}{2\pi}\int_{-\infty}^{\infty}\Gamma\left(\frac{1}{2}+it\right)x^{-(1/2+it)}\,dt. \tag{42}$$

Для доказательства этого тождества рассмотрим путь интегрирования, показанный на рис. 20(а), где N, N' и M велики. Значение интеграла вдоль этого контура равно сумме вычетов внутри контура, а именно

$$\sum_{0\le k<M}x^{-k}\lim_{z\to -k}(z+k)\Gamma(z) = \sum_{0\le k<M}x^{-k}\frac{(-1)^k}{k!}.$$

Интеграл по верхней части контура есть $O\left(\int_{-\infty}^{1/2}|\Gamma(t+iN)|x^{-t}\,dt\right)$, и у нас имеется известная оценка

$$\Gamma(t+iN) = O(N^{t-1/2}e^{-\pi N/2}) \qquad \text{при } N\to\infty.$$

[Свойства гамма-функций см., например, в книге A. Erdélyi et al., Higher Transcendental Functions, 1 (New York: McGraw-Hill, 1953), гл. 1.] Поэтому интеграл по верхнему отрезку бесконечно мал: $O\left(e^{-\pi N/2}\int_{-\infty}^{1/2}(N/x)^t\,dt\right)$. Интеграл по нижнему отрезку ведет себя столь же безобидно. Для вычисления интеграла по левому отрезку контура воспользуемся тем фактом, что

$$\Gamma\left(\frac12+it-M\right)=\Gamma\left(\frac12+it\right)/\left(-M+\frac12+it\right)\ldots\left(-1+\frac12+it\right)=$$
$$=\Gamma\left(\frac12+it\right)O(1/(M-1)!).$$

Следовательно, интеграл по левой стороне есть $O(x^{M-1/2}/(M-1)!)\times\int_{-\infty}^{\infty}|\Gamma\left(\frac12+it\right)|\,dt$. Поэтому при $M,\ N,\ N'\to\infty$ уцелеет лишь интеграл по правой стороне; тем самым доказано тождество (42). В действительности тождество (42) остается в силе и в том случае, если заменить $\frac12$ любым положительным числом.

Тем же самым рассуждением можно воспользоваться для вывода других полезных соотношений, содержащих гамма-функции. Величину x^{-z} можно заменить другими функциями от z; можно также заменить другой величиной константу $\frac12$. Например,

$$\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)x^{-z}\,dz=e^{-x}-1+x,\tag{43}$$

а это—критическая величина в формуле (41) для T_n:

$$T_n=n\sum_{j\ge1}\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)(n/2^j)^{-1-z}\,dz.\tag{44}$$

Суммирование можно внести под знак интеграла, так как сходимость здесь достаточно хорошая; имеем

$$\sum_{j\le1}(n/2^j)^w=n^w\sum_{j\le1}(1/2^w)^j=n^w/(2^w-1),\qquad\text{если }\Re(w)>0,$$

так как $|2^w|=2^{\Re(w)}>1$. Поэтому

$$T_n=\frac{n}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\frac{\Gamma(z)n^{-1-z}}{2^{-1-z}-1}\,dz,\tag{45}$$

и остается оценить последний интеграл.

На этот раз интегрирование производится по контуру, который больше вытянут вправо, как изображено на рис. 20(b). Интеграл по верхнему отрезку есть $O\left(n^{1/2}e^{-\pi M/2}\int_{-3/2}^{M}N^t\,dt\right)$, если $2^{iN}\ne1$, а интеграл по нижнему отрезку также пренебрежимо мал. Интеграл по правому отрезку равен $O\left(n^{-1-M}\int_{-\infty}^{\infty}|\Gamma(M+it)|\,dt\right)$. Фиксируя M и устремляя N, N' к ∞, можно показать, что $-T_n/n$ есть $O(n^{-1-M})$ плюс сумма вычетов в области $-3/2<\Re(z)<M$. Пусть $M\to\infty$, $\Gamma(z)$ имеет простые полюсы при $z=-1$ и $z=0$, n^{-1-z} не имеет полюсов, а $1/(2^{-1-z}-1)$ имеет простые полюсы при $z=-1+2\pi ik/\ln2$.

Наибольшую трудность представляет двойной полюс в точке $z=-1$. Если $w=z+1$ мало, то можно воспользоваться известным соотношением

$$\Gamma(z+1)=\exp(-\gamma z+\zeta(2)z^2/2-\zeta(3)z^3/3+\zeta(4)z^4/4-\ldots),$$

где $\zeta(s)=1^{-s}+2^{-s}+3^{-s}+\ldots=H_\infty^{(s)}$, для вывода следующих разложений:

$$\Gamma(z)=\frac{\Gamma(w+1)}{w(w-1)}=-w^{-1}+(\gamma-1)+O(w);$$
$$n^{-1-z}=1-(\ln n)w+O(w^2);$$
$$1/(2^{-1-z}-1)=-w^{-1}/\ln2-\frac12+O(w).$$

Вычет в точке $z=-1$ равен коэффициенту при w^{-1} в произведении этих трех формул, а именно $\frac12-(\ln n+\gamma-1)/\ln2$. Прибавляя остальные вычеты, получаем формулу

$$T_n/n=\frac{\ln n+\gamma-1}{\ln2}-\frac12+f(n)+\frac{2}{n},\tag{46}$$

где $f(n)$—функция довольно необычного вида:

$$f(n) = \frac{2}{\ln 2} \sum_{k \geq 1} \Re(\Gamma(-1 - 2\pi i k / \ln 2) \exp(2\pi i k \log_2 n)). \tag{47}$$

Заметим, что $f(n) = f(2n)$. Среднее значение $f(n)$ равно 0, так как среднее значение каждого слагаемого равно 0. (Можно считать, что величина $(\log_2 n) \bmod 1$ распределена равномерно, принимая во внимание результаты о числах с плавающей точкой, полученные в п. 4.2.4.) Кроме того, поскольку $|\Gamma(-1 + it)| = |\pi/(t(1 + t^2) \sinh \pi t)|^{1/2}$, нетрудно показать, что

$$f(n) < 0.0000001725;$$

таким образом, в практических приложениях $f(n)$ можно спокойно отбросить. Что касается теории, то без $f(n)$ мы не можем получить асимптотического ряда для U_n; именно поэтому величина U_n сравнительно трудна для анализа. Итак, мы доказали, что

$$U_n = n \log_2 n + n \left(\frac{\gamma - 1}{\ln 2} - \frac{1}{2} + f(n) \right) + O(1). \tag{48}$$

Другие примеры применения этого метода гамма-функций можно найти в упр. 51–53 и в § 6.3.

Упражнения

1. [M20] Пусть $a_1 \ldots a_n$—перестановка множества $\{1, \ldots, n\}$, и пусть i и j таковы, что $i < j$ и $a_i > a_j$. Пусть $a'_1 \ldots a'_n$—перестановка, которая получается из $a_1 \ldots a_n$, если поменять местами a_i и a_j. Может ли в $a'_1 \ldots a'_n$ быть больше инверсий, чем в $a_1 \ldots a_n$?

>2. [M25] (a) Каково минимальное число обменов, необходимое для того, чтобы отсортировать перестановку 3 7 6 9 8 1 4 5? (b) Вообще пусть дана перестановка $\pi = a_1 \ldots a_n$ множества $\{1, \ldots, n\}$, и пусть $\operatorname{xch}(\pi)$—минимальное число обменов, в результате которых перестановка π будет отсортирована в возрастающем порядке. Выразите $\operatorname{xch}(\pi)$, через "более простые" характеристики перестановки π. (Ср. с упр. 5 2.1–39.)

3. [10] Является ли устойчивой сортировка методом пузырька (алгоритм B)?

4. [M23] Если в шаге B4 получится $t = 1$, то на самом деле работу алгоритма B можно сразу же заканчивать, потому что в следующем шаге B2 не выполнится никаких полезных действий. Какова вероятность того, что при сортировке случайной перестановки в шаге B4 окажется $t = 1$?

5. [M25] Пусть $b_1\ b_2\ \ldots b_n$—таблица инверсий перестановки $a_1\ a_2\ \ldots a_n$. Покажите, что после r просмотров сортировки методом пузырька значение переменной `BOUND` равно $\max\{b_i + i \mid b_i \geq r\} - r$, где $0 \leq r \leq \max(b_1, \ldots, b_n)$.

6. [M22] Пусть $a_1 \ldots a_n$—перестановка множества $\{1, \ldots, n\}$, и пусть $a'_1 \ldots a'_n$—обратная к ней перестановка. Покажите, что число просмотров, необходимых для того, чтобы отсортировать $a_1 \ldots a_n$" методом пузырька. равно $1 + \max(a'_1 - 1, a'_2 - 2, \ldots, a'_n - n)$.

7. [M28] Вычислите стандартное отклонение числа просмотров сортировки методом пузырька и выразите его через n и функцию $P(n)$. [Ср. с формулами (6) и (7).]

8. [M24] Выведите формулу (8).

9. [M48] Проанализируйте число просмотров и число сравнений в алгоритме шейкер-сортировки. (*Замечание:* частичная информация содержится в упр. 5.4.8–9.)

10. [M26] Пусть $a_1\ a_2\ \ldots\ a_n$—2-упорядоченная перестановка множества $\{1, 2, \ldots, n\}$. (a) Каковы координаты конечных точек a_i-го шага соответствующего решеточного пути (ср. с рис. 11)? (b) Докажите, что сравнение/обмен элементов a_1: a_2, a_3, a_4, ...соответствует перегибанию пути относительно диагонали, как на рис. 18(b). (c) Докажите, что сравнение/обмен элементов a_2: a_{2+d}, a_4:a_{4+d}, ... соответствует перегибанию пути относительно линии, расположенной на m единиц ниже диагонали, как на рис. 18(c), (d) и (e), если $d = 2m - l$.

>11. [M25] На какой перестановке множества $\{1, 2, \ldots, 16\}$ достигается максимум числа обменов в алгоритме Бэтчера?

12. [24] Напишите `MIX`-программу для алгоритма M, предполагая, что `MIX`—двоичная вычислительная машина с операциями `AND` и `SRB`. Сколько времени потребуется вашей программе, чтобы отсортировать шестнадцать записей из табл. 1?

13. [10] Устойчива ли сортировка Бэтчера?

14. [M21] Пусть $c(N)$—число сравнений ключей, производимых при сортировке N элементов методом Бэтчера; это равно числу выполнении шага M4. (a) Покажите, что при $t \geq 1$ $c(2^t) = 2c(2^{t-1} + (t -$

$1)2^{t-1}+1$. (b) Найдите простое выражение для $c(2^t)$ как функцию от t. (*Указание:* рассмотрите последовательность $x_t = c(2^t)/2^t$).

15. [M38] Содержание этого упражнения—анализ функции $c(N)$ из упр. 14 и нахождение формулы для $c(N)$, если $N = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}$, $e_1 > e_2 > \ldots > e_r \geq 0$. (a) Пусть $a(N) = c(N+1) - c(N)$. Докажите, что $a(2n) = a(n) + \lfloor \log_2(2n) \rfloor$, $a(2n+1) = a(n) + 1$; отсюда

$$a(N) = \binom{e_1+1}{2} - r(e_1 - 1) + (e_1 + e_2 + \cdots + e_r).$$

(b) Пусть $x(n) = a(n) - a(\lfloor n/2 \rfloor)$, и тогда $a(n) = x(n) + x(\lfloor n/2 \rfloor) + x(\lfloor n/4 \rfloor) + \cdots$. Пусть $y(n) = x(1) + x(2) + \cdots + x(n)$, и пусть $z(2n) = y(2n) - a(n)$, $z(2n+1) = y(2n+1)$. Докажите, что $c(N+1) = z(N) + 2z(\lfloor N/2 \rfloor) + 4z(\lfloor N/4 \rfloor) + \cdots$. (c) Докажите,что $y(N) = N + (\lfloor N/2 \rfloor + 1) \times (e_1 - 1) - 2^{e_1} + 2$. (d) Теперь соберите все вместе и найдите выражение $c(N)$ через показатели e_j при фиксированном значении r.

16. [BM46] Найдите асимптотическое значение среднего числа обменов в случае, когда алгоритм Бэтчера применяется к $N = 2^t$ различным элементам, расположенным в случайном порядке.

>17. [20] Где в алгоритме Q используется то, что K_0 и K_{N+1} обладают значениями, постулированными неравенствами (13)?

>18. [20] Объясните, как работает алгоритм Q в случае, когда все ключи в исходном файле равны. Что произойдет, если в шагах Q3 и Q5 заменить знаки "$<$" на "$\leq$"?

19. [15] Будет ли алгоритм Q по-прежнему работать правильно, если вместо стека (последним включается—первым исключается) воспользоваться очередью (первым включается—первым исключается)?

20. [M20] Выразите наибольшее число элементов, которые могут одновременно оказаться в стеке во время работы алгоритма Q, в виде функции от M и N.

21. [20] Объясните, почему фаза разделения в алгоритме Q требует как раз такого числа сравнений, пересылок и т. д., как это описано в (17).

22. [M25] Пусть $p_k N$—вероятность того, что величина A в (14) будет равна k, если алгоритм Q применяется к случайной перестановке множества $\{1, 2, \ldots, N\}$, и пусть $A_N(z) = \sum_k p_k N^{z^k}$—соответствующая производящая функция. Докажите, что $A_N(z) = 1$ при $N \leq M$, и $A_N(z) = z(\sum_{1 \leq s \leq N} A_{s-1}(z) A_{N-s}(z))/N$ при $N > M$. Найдите аналогичные рекуррентные соотношения, определяющие другие распределения вероятностей $B_N(z)$, $C_N(z)$, $D_N(z)$, $E_N(z)$, $L_N(z)$, $X_N(z)$.

23. [M24] Пусть A_N, B_N, D_N, E_N, L_N, X_N—средние значения соответствующих величин в (14) при сортировке случайной перестановки-множества $\{1, 2, \ldots, N\}$. Найдите для этих величин рекуррентные соотношения, аналогичные (18), затем разрешите эти соотношения и получите формулы (25).

24. [M21] Очевидно, в алгоритме Q производится несколько больше сравнений, чем это необходимо, потому что в шагах Q3 и Q5 может оказаться, что $i = j$ или даже $i > j$. Сколько сравнений G_N производилось бы в среднем, если бы исключались все сравнения при $i \geq j$?

25. [M20] Чему равны точные значения величин A, B, C, D, E, L, X для программы Q в случае, когда исходные ключи представляют собой упорядоченный набор чисел 1, 2, ..., N в предположении, что $N > M$?

>26. [M21] Постройте исходный файл, при котором программа Q работала бы еще медленнее, чем в упр. 25. (Попытайтесь найти действительно плохой случай.)

27. [M23] Какой исходный файл будет *наилучшим* для алгоритма Q? Найдите приблизительные значения величин A, B, C, D, E, X для этого случая.

28. [M26] Найдите рекуррентное соотношение, аналогичное (20), которому удовлетворяет среднее число сравнений в алгоритме Q, модифицированном Синглтоном (т. е. когда в качестве s выбирается не $s = K_1$, а медиана из $\{ K_1, K_{\lfloor (N+1)/2 \rfloor}, K_N \}$).

29. [BM40] Найдите асимптотическое значение числа сравнений в алгоритме Синглтона "медиана из трех" (см. упр. 28).

30. [25] (П. Шаклетон.) При сортировке *ключей длиной в несколько слов* многие алгоритмы все более замедляются по мере того, как файл приближается к упорядоченному состоянию, поскольку для определения правильного лексикографического порядка равных или почти равных ключей требуется сравнить несколько пар слов (см. упр. 5-5). Файлы, которые встречаются на практике, часто содержат почти равные ключи, и это явление может заметно отразиться на времени сортировки.

Такое затруднение существенно для алгоритма Q, но не для алгоритма R, поскольку там каждый раз проверяется только один бит (хотя присутствие равных и почти равных ключей может сильно увеличить время работы алгоритма R по другим причинам).

Объясните, как можно усовершенствовать алгоритм Q, чтобы избежать этого затруднения; в подфайле, о котором известно, что старшие k слов во всех ключах содержат постоянные значения, следует проверять лишь $(k+1)$-е слова ключей.

>31. [20] (Ч. Э. Р. Хоар) Предположим, что нам нужно не отсортировать файл, а лишь определить m-й наименьший по величине элемент из заданного множества n элементов. Покажите, что "быструю сортировку" можно приспособить для этой цели, избежав значительной части вычислений, необходимых для полной сортировки.

32. [M40] Найдите простое выражение "в замкнутом виде" для C_{nm}—среднего числа сравнений ключей, необходимых для отыскания m-го наименьшего из n элементов по методу упр. 31. (Для простоты можно пропускать все сравнения с $i \geq j$, как в упр. 24.) Каково асимптотическое поведение величины $C_{(2m-1)m}$—среднего числа сравнений, необходимых для нахождения медианы из $2m-1$ элементов методом Хоара?

>33. [20] Разработайте алгоритм переразмещения чисел в некоторой заданной таблице таким образом, чтобы все отрицательные значения предшествовали положительным. Элементы не нужно полностью сортировать: достаточно просто отделить отрицательные числа от положительных. Ваш алгоритм должен производить минимально возможное число обменов.

34. [20] Как можно ускорить циклы проверки битов в обменной поразрядной сортировке (шаги от R3 до R6)?

35. [M23] Проанализируйте статистические характеристики A, B, C, G, K, L, R, S и X, которые получаются при обменной поразрядной сортировке для "исходных данных типа (i)".

36. [M27] Для любой данной последовательности чисел $\langle a_n \rangle = a_0, a_1, a_2, \ldots$ определим ее *биномиальное преобразование* $\langle \hat{a} \rangle_n = \hat{a}_0, \hat{a}_1, \hat{a}_2, \ldots$ правилом

$$\hat{a}_n = \sum_k \binom{n}{k} (-1)^k a_k.$$

(a) Докажите, что $\left\langle \hat{\hat{a}}_n \right\rangle = \langle a_n \rangle$. (b) Найдите биномиальные преобразования последовательностей $\langle 1 \rangle$; $\langle n \rangle$; $\left\langle \binom{n}{m} \right\rangle$ при фиксированном m; $\langle a^n \rangle$ при фиксированном a; $\left\langle \binom{n}{m} a^n \right\rangle$ при фиксированных a и m. (c) При условии, что последовательность $\langle x_n \rangle$ удовлетворяет соотношению

$$x_n = a_n + 2^{1-n} \sum_{k \geq 2} \binom{n}{k} x_k \qquad \text{при } n \geq 2,\ x_0 = x_1 = a_0 = a_1 = 0,$$

докажите, что

$$x_n = \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{2^{k-1} \hat{a}_k}{2^{k-1} - 1} = a_n + \sum_{k \geq 2} \binom{n}{k} (-1)^k \frac{\hat{a}_k}{2^{k-1} - 1}.$$

37. [M28] Определите все такие последовательности $\langle a_n \rangle$, что $\langle \hat{a}_n \rangle = \langle a_n \rangle$ в смысле упр. 36.

>38. [M30] Найдите A_N, B_N, C_N, G_N, K_N, L_N, R_N, и X_N—средние значения величин (29)—в случае, когда поразрядной сортировке подвергаются "исходные данные типа (ii)". Выразите ваши ответы через N и функции

$$U_n = \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k}{2^{k-1} - 1}, \qquad V_n = \sum_{k \geq 2} \binom{n}{k} \frac{(-1)^k k}{2^{k-1} - 1} = n(U_n - U_{n-1}).$$

[*Указание:* см. упр. 36.]

39. [20] Результаты (30) показывают, что поразрядная обменная сортировка, примененная к случайным входным данным, требует около 1.44 стадий. Докажите, что быстрая сортировка никогда не требует более N стадий, и объясните, почему при обменной поразрядной сортировке часто бывает необходимо более N стадий.

40. [21] Объясните, как нужно изменить алгоритм R, чтобы он работал достаточно эффективно и тогда, когда сортируемые файлы содержат много равных ключей.

41. [23] Дайте точное описание алгоритма "интервальной обменной сортировки" Ван Эмдена.

42. [M43] Проанализируйте алгоритм интервальной обменной сортировки из упр. 41.

43. [BM21] Докажите, что $\int_0^1 y^{-1}(e^{-y} - 1)\, dy + \int_1^\infty y^{-1} e^{-y}\, dy = -\gamma$. [*Указание:* рассмотрите $\lim_{a \to 0+} y^{a-1}$.]

44. [BM24] Выведите формулу (37), как это предложено в тексте настоящего пункта.

45. [BM20] Объясните, почему при $x > 0$ справедливо равенство (43).

46. [BM20] Каково значение выражения $(1/2\pi i) \int_{a-i\infty}^{a+i\infty} \Gamma(z) n^{s-z}\, dz/(2^{s-z} - 1)$ при условии, что s—целое положительное число и $0 < a < s$?

47. [BM21] Докажите, что $\sum_{j \geq 1} (n/2^j) e^{-n/2^j}$—ограниченная функция от n.

48. [BM24] Найдите асимптотическое значение величины V_n, определенной в упр. 38, при помощи методов, аналогичных тем, которыми в тексте настоящего пункта исследовалась величина U_n, и получите члены до $O(1)$.

49. [BM24] Продолжите асимптотическую формулу (47) для U_n до членов порядка $O(n^{-1})$.
50. [BM24] Найдите асимптотическое значение функции

$$U_{mn} = \sum_{k\ge 2} \binom{n}{k} (-1)^k \frac{1}{m^{k-1}-1},$$

где m—произвольное фиксированное число, большее 1. (Эта величина при m целом большем 2 потребуется при исследовании обобщений обменной поразрядной сортировки, а также при изучении алгоритмов поиска в "боровой памяти" в § 6.3.)

>51. [BM28] Покажите, что для вывода асимптотического разложения $r_k(m)$ можно воспользоваться методом исследования асимптотических задач при помощи гамма-функций вместо формулы суммирования Эйлера (ср. с (35)). Это дает нам единообразный способ изучения $r_k(m)$ при всех k, не полагаясь на такие "трюки", как введение функции $g_{-1}(x) = (e^{-x^2}-1)/x$.
52. [BM35] (Н. Г. де Брейн.) Каково асимптотическое поведение суммы

$$S_n = \sum_{t\ge 1} \left(\frac{2n}{n+t}\right) d(t),$$

где $d(t)$—количество делителей числа t? (Так, $d(1) = 1$, $d(2) = d(3) = 2$, $d(4) = 3$, $d(5) = 2$ и т. д. Этот вопрос возникает в связи с анализом алгоритма прохождения дерева; упр. 2.3.1-11.) Найдите значение величины $S_n/\binom{2n}{n}$ до членов порядка $O(n^{-1})$.
53. [BM42] Проанализируйте число проверок битов и число обменов, выполняемых при обменной поразрядной сортировке, если исходными данными служат двоичные числа с бесконечной точностью из промежутка $[0, 1)$, каждый бит которых независимо принимает значение 1 с вероятностью p. (В основном тексте пункта обсуждался лишь случай $p = \frac{1}{2}$; применявшиеся методы можно обобщить на случай произвольного p.) Рассмотрите особо случай $p = 1/\phi = .61803\ldots$.
54. [BM24] (С. О. Райе.) Покажите, что U_n можно записать в виде

$$U_n = (-1)^n \frac{n!}{2\pi i} \oint_C \frac{dz}{z(z-1)\ldots(z-n)} \frac{1}{2^{z-1}-1},$$

где C—замкнутая кривая, охватывающая область около точек 2, 3, ..., n. В результате замены C на произвольно большую окружность с центром в начале координат получаем сходящийся ряд

$$U_n = n(H_{n-1}-1)/(\ln 2) - \frac{1}{2}n + 2 + \frac{2}{\ln 2} \sum_{m\ge 1} \Re(B(n+1, -1+ibm)),$$

где $b = 2\pi/(\ln 2)$ и $B(n+1, -1+ibm) = \Gamma(n+1)\Gamma(-1+ibm)/\Gamma(n+ibm) = n!/\prod_{0\le k\le n}(k-1+ibm)$.

>55. [22] Покажите, как нужно изменить программу Q, чтобы в качестве разделяющего элемента выбиралась медиана из трех ключей (28).
56. [M44] Проанализируйте в среднем поведение величин, от которых зависит время работы программы Q, если программа изменена так, что она выбирает медиану из трех элементов, как в упр. 55. [(Ср. упр. 29.)]

>57. [BM24] Найдите асимптотическое значение числа правосторонних максимумов, встречающихся при сортировке вставками в несколько списков (формула (5.2.1-11)), если $M = N/\alpha$ при фиксированном α и $N \to \infty$. Выполните разложение до членов порядка $O(N^{-1})$ и выразите ваш ответ через интегральную показательную функцию $E_1(z) = \int_z^\infty e^{-t}\, dt/t$.

Сортировка посредством выбора Еще одно важное семейство методов сортировки основано на идее многократного выбора. Вероятно, простейшая сортировка посредством выбора сводится к следующему:

i) Найти наименьший ключ; переслать соответствующую запись в область вывода и заменить ключ значением "∞" (которое по предположению больше любого реального ключа).
ii) Повторить шаг (i). На этот раз будет выбран ключ, наименьший из оставшихся, так как ранее наименьший ключ был заменен на ∞.
iii) Повторять шаг (i) до тех пор, пока не будут выбраны N записей.

Заметим, что этот метод требует наличия всех исходных элементов до начала сортировки, а элементы вывода он порождает последовательно, один за другим. Картина, по существу, противоположна методу вставок, в котором исходные элементы должны поступать последовательно, но вплоть до завершения сортировки ничего не известно об окончательном выводе.

Ряд вычислительных машин (например, машины с циклической барабанной памятью) имеет встроенную команду "найти наименьший элемент", которая выполняется с большой скоростью. На таких машинах сортировка указанным методом особенно привлекательна, если только N не слишком велико.

Описанный метод требует $N-1$ сравнений каждый раз, когда выбирается очередная запись; он также требует отдельной области вывода в памяти. Имеется очевидный способ несколько поправить ситуацию, избежав при этом использования ∞: выбранное значение можно записывать в соответствующую позицию, а запись, которая ее занимала, переносить на место выбранной. Тогда эту позицию не нужно рассматривать вновь при последующих выборах. На этой идее основан наш первый алгоритм сортировки посредством выбора.

Алгоритм S. *(Сортировка посредством простого выбора.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте. После завершения сортировки их ключи будут упорядочены: $K_1 \le \ldots \le K_N$. Сортировка основана на описанном выше методе, если не считать того, что более, удобно, оказывается, выбирать сначала *наибольший* элемент, затем второй по величине и т. д.

S1 [Цикл по j.] Выполнить шаги **S2** и **S3** при $j = N, N-1, \ldots, 2$.

S2 [Найти $\max(K_1, \ldots, K_j)$.] Найти наибольший из ключей $K_j, K_{j-1}, \ldots, K_1$; пусть это будет K_i.

S3 [Поменять местами с R_j.] Взаимозаменить записи $R_i \leftrightarrow R_j$. (Теперь записи $R_j, \ldots, R_N$ занимают свои окончательные позиции.) ∎

Picture: Рис. 21. Сортировка простым выбором.

В табл. 1 показано действие этого алгоритма на шестнадцать ключей, выбранных нами для примеров; элементы, претендующие на то, чтобы оказаться максимальными во время поиска в шаге S2, выделены жирным шрифтом.

Таблица 1

Сортировка посредством простого выбора

503	087	512	061	**908**	170	**897**	275	653	426	154	509	612	677	**765**	**703**
503	087	512	061	703	170	**897**	275	653	426	154	509	612	677	**765**	908
503	087	512	061	703	170	**765**	275	653	426	154	509	612	**677**	897	908
503	087	512	061	**703**	170	**677**	275	**653**	426	154	509	**612**	765	897	908
503	087	512	061	612	170	**677**	275	**653**	426	154	**509**	703	765	897	908
503	087	512	061	612	170	509	275	**653**	**426**	**154**	677	703	765	897	908
...															
061	**087**	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Соответствующая MIX-программа довольно проста.

Программа S. *(Сортировка посредством простого выбора.)* Как и в предыдущих программах этой главы, записи, находящиеся в ячейках от INPUT + 1 до INPUT + N, сортируются на том же месте по ключу, занимающему полное слово. Значения регистров: rA ≡ текущий максимум, rI1 ≡ $j-1$, rI2 ≡ k (индекс при поиске), rI3 ≡ i. Предполагается, что $N \ge 2$.

```
START  ENT1  N-1          1     S1. Цикл по j. j ← N.
2H     ENT2  0,1        N-1     S2. Найти max(K_1,...,K_j). k ← j-1.
       ENT3  1,1        N-1     i ← j.
       LDA   INPUT,3    N-1     rA ← K_i.
8H     CMPA  INPUT,2      A
       JGE   *+3          A     Переход, если K_i ≥ K_k.
       ENT3  0,2          B     В противном случае установить i ← k,
       LDA   INPUT, 3     B     rA ← K_i.
       DEC2  1            A     k ← k-1.
       J2P   8B           A     Повторить, если k > 0.
       LDX   INPUT+1,1  N-1     S3. Поменять местами с R_j.
       STX   INPUT,3    N-1     R_i ← R_j.
       STA   INPUT+1,1  N-1     R_j ← rA.
       DEC1  1          N-1
       J1P   2B         N-1     N ≥ j ≥ 2.
```

Время работы этой программы зависит от числа элементов N, числа сравнений A и числа "правосторонних максимумов" B. Нетрудно видеть, что независимо от значений исходных ключей

$$A = \binom{N}{2} = \frac{1}{2}N(N-1). \tag{1}$$

Следовательно, переменной является только величина B. Несмотря на всю безыскусность простого выбора, не так легко произвести точный анализ величины B. В упр. с 3 по 6 показано, что

$$B = (\min 0, \operatorname{ave}(N+1)H_n - 2N, \max\lfloor N^2/4 \rfloor); \tag{2}$$

в этом случае особенно интересным оказывается максимальное значение (стандартное отклонение величины B до сих пор не определено).

Таким образом, среднее время работы программы S равно $2.5N^2 + 3(N+1)H_N + 3.5N - 11$ единиц, т. е. она лишь немногим медленнее простых вставок (программа 5.2.1S). Интересно сравнить алгоритм S с сортировкой методом пузырька (алгоритм 5.2.2B), так как метод пузырька можно рассматривать как алгоритм выбора, в котором иногда выбирается более одного элемента за раз. По этой причине при сортировке методом пузырька производится меньше сравнений, чем при простом выборе, и она, как может показаться, предпочтительнее. Но в действительности программа 5.2.2B более чем вдвое медленнее программы S! Сортировка методом пузырька проигрывает из-за того, что в ней выполняется слишком много обменов, в то время как при сортировке простым выбором производится очень мало пересылок данных.

Усовершенствования простого выбора. Существует ли какой-нибудь способ улучшить метод выбора, используемый в алгоритме S? Возьмем к примеру поиск максимума в шаге S2: возможен ли существенно более быстрый способ нахождения максимума? Ответ на этот вопрос—*нет!*

Лемма М. *В любом алгоритме нахождения максимума из n элементов, основанном на сравнении пар элементов, необходимо выполнить по крайней мере $n-1$ сравнений.*

Доказательство Если произведено менее $n-1$ сравнений, то найдутся по крайней мере два элемента, для которых не было обнаружено ни одного элемента, превосходящего их по величине. Следовательно, мы так и не узнаем, который из этих двух элементов больше, и, значит, не сможем определить максимум. ∎

Таким образом, процесс выбора, в котором отыскивается наибольший элемент, должен состоять не менее чем из $n-1$ шагов. Означает ли это, что для всех методов сортировки, основанных на n повторных выборах, число шагов неизбежно будет порядка n^2? К счастью, лемма М применима только к *первому* шагу выбора; при последующих выборах можно использовать извлеченную ранее информацию. Например, в упр. 8 показано, что сравнительно простое изменение алгоритма S наполовину сокращает число сравнений.

Рассмотрим 16 чисел, представленных в 1-й строке в табл. 1. Один из способов сэкономить время при последующих выборах—разбить все числа на четыре группы по четыре числа. Начать можно с определения наибольшего, элемента каждой группы, а именно соответственно с ключей

$$512, 908, 653, 765;$$

тогда наибольший из этих четырех элементов 908 и будет наибольшим во всем файле. Чтобы получить второй по величине элемент, достаточно просмотреть сначала остальные три элемента группы, содержащей 908; наибольший из $\{170, 897, 275\}$ равен 897, и тогда наибольший среди

$$512, 897, 653, 765$$

это 897. Аналогично, для того чтобы получить третий по величине элемент, определяем наибольший из $\{170, 275\}$, а затем наибольший из

$$512, 275, 653, 765$$

и т. д. Каждый выбор, кроме первого, требует не более 6 дополнительных сравнений. Вообще, если N—точный квадрат, то можно разделить файл на $\sqrt{N}$ групп по $\sqrt{N}$ элементов в каждой; любой выбор, кроме первого, требует не более чем $\sqrt{N}-1$ сравнений внутри группы ранее выбранного элемента плюс $\sqrt{N}-1$ сравнений среди "лидеров групп". Этот метод получил название "квадратичный выбор"; общее время работы для него есть $O(N\sqrt{N})$, что существенно лучше, чем $O(N^2)$.

Метод квадратичного выбора впервые опубликован Э. Х. Фрэндом [*JACM*, **3** (1956), 152–154]; он указал, что эту идею можно обобщить, получив в результате метод кубического выбора, выбора четвертой степени и т. д. Например, метод кубического выбора состоит в том, чтобы разделить файл на $\sqrt[3]{N}$ больших групп, каждая из которых содержит по $\sqrt[3]{N}$ малых групп по $\sqrt[3]{N}$ записей; время работы пропорционально $N\sqrt[3]{N}$. Если развить эту идею до ее полного завершения, то мы придем к тому, что Фрэнд назвал "выбор n-й степени", основанный на структуре бинарного дерева. Время работы этого метода пропорционально $N \log N$; мы будем называть его *выбором из дерева*.

Выбор из дерева. Принципы сортировки посредством выбора из дерева будет легче воспринять, если воспользоваться аналогией с типичным "турниром с выбыванием". Рассмотрим, например, результаты соревнования по настольному теннису, показанные на рис. 22. Джим побеждает Дона, а Джо побеждает Джека; затем в следующем туре Джо выигрывает у Джима и т. д.

На рис. 22 показано, что Джо—чемпион среди восьми спортсменов, а для того, чтобы определить это, потребовалось $8-1 = 7$ матчей (т. е. сравнений). Дик вовсе не обязательно будет вторым по силе игроком: любой из спортсменов, у которых выиграл Джо, включая даже проигравшего в первом туре Джека, мог бы оказаться вторым по силе игроком. Второго игрока можно определить, заставив Джека сыграть с Джимом, а победителя этого матча—с Диком; всего два дополнительных матча требуется для определения второго по силе игрока, исходя из того соотношения сил, которое мы запомнили из предыдущих игр.

Вообще можно "вывести" игрока, находящегося в корне дерева, и заменить его чрезвычайно слабым игроком. Подстановка этого слабого игрока означает, что первоначально второй по силе спортсмен станет теперь наилучшим, и именно он окажется в корне, если вновь вычислить победителей в верхних уровнях дерева. Для этого нужно изменить лишь один путь, в дереве, так что для выбора следующего по силе игрока необходимо менее $\lceil \log_2 N \rceil$) дополнительных сравнений. Суммарное

Picture: Рис. 23. Пример сортировки посредством выбора из дерева...

время выполнения такой сортировки посредством выбора примерно пропорционально $N \log N$.

На рис. 23 сортировке посредством выбора из дерева подвергаются наши 16 чисел. Заметим, что для того, чтобы знать, куда вставлять следующий элемент "$-\infty$", необходимо помнить, откуда пришел ключ, находящийся в корне. Поэтому узлы разветвления в действительности содержат указатели или индексы, описывающие позицию ключа, а не сам ключ. Отсюда следует, что необходима память для N исходных записей, $N-1$ слов-указателей и N выводимых записей. (Разумеется, если вывод

Picture: Рис. 24. Применение корпоративной системы выдвижений к сортировке. Каждый поднимается на свой уровень некомпетентности в иерархии.

идет на ленту или на диск, то не нужно сохранять выводимые записи в оперативной памяти.)

Чтобы оценить те замечательные усовершенствования, которые мы собираемся обсудить, в этом месте следует прервать чтение до тех пор, пока вы не освоитесь с выбором из дерева хотя бы настолько, что решение упр. 10 не составит для вас никакого труда.

Одна из модификаций выбора из дерева, введенная, по существу, К. Э. Айверсоном [A Programming Language (Wiley, 1962), 223–227], устраняет необходимость указателей, следующим образом осуществляя "заглядывание вперед": когда победитель матча на нижнем уровне поднимается вверх, то на нижнем уровне его сразу же можно заменить на "$-\infty$"; когда же победитель перемещается вверх с одного разветвления на другое, то его можно заменить игроком, который в конце концов все равно должен подняться, на его прежнее место (а именно наибольшим из двух ключей, расположенных под ним). Выполнив эту операцию столько раз, сколько возможно, придем от рис. 23(а) к рис. 24.

Коль скоро дерево построено таким образом, можно продолжать сортировку не "восходящим" методом, показанным на рис. 23, а "нисходящим": выводится элемент, находящийся в корне, перемещается вверх наибольший из его потомков, перемещается вверх наибольший из потомков последнего и т. д. Процесс начинает походить не столько на турнир по настольному теннису, сколько на корпоративную систему выдвижений.

Читатель должен уяснить, что у нисходящего метода есть важное достоинство—он позволяет избежать лишних сравнений $-\infty$ с $-\infty$. (Пользуясь восходящим методом, мы на более поздних стадиях сортировки всюду натыкаемся на $-\infty$, а при нисходящем методе можно на каждой стадии заканчивать преобразование дерева сразу же после занесения $-\infty$.)

Picture: Рис. 25. Последовательное распределение памяти для полного бинарного дерева.

На рис. 23 и 24 изображены *полные бинарные деревья* с 16 концевыми узлами (ср. с п. 2.3.4.5); такие деревья удобно хранить в последовательных ячейках памяти, как показано на рис. 25. Заметим, что отцом

узла номер k является узел $\lfloor k/2 \rfloor$, а его потомками являются узлы $2k$ и $2k+1$. Отсюда вытекает еще одно преимущество нисходящего метода, поскольку зачастую значительно проще продвигаться вниз от узла k к узлам $2k$ и $2k+1$, чем вверх от узла k к узлам $k \oplus 1$ и $\lfloor k/2 \rfloor$. (Здесь через $k \oplus 1$ обозначено число $k+1$ или $k-1$ в зависимости от того, является ли k четным или нечетным.)

До сих пор в наших примерах выбора из дерева в той или иной мере предполагалось, что N есть степень 2; в действительности можно работать с произвольным значением N, так как полное бинарное дерево с N концевыми узлами нетрудно построить для любого N.

Мы подошли теперь к основному вопросу: нельзя ли в нисходящем методе обойтись совсем без "$-\infty$"? Не правда ли, было бы чудесно, если бы всю существенную информацию, имеющуюся на рис. 24, удалось расположить в ячейках 1–16 полного бинарного дерева без всяких бесполезных "дыр", содержащих $-\infty$? Поразмыслив, можно прийти к выводу, что эта цель в действительности достижима, причем не только исключается $-\infty$, но и появляется возможность сортировать N записей на том же месте без вспомогательной области вывода. Это приводит к еще одному важному алгоритму сортировки. Его автор Дж. У. Дж. Уильямс [*CACM*, **7** (1964), 347–348] окрестил свой алгоритм "пирамидальной сортировкой" ("heapsort").

Пирамидальная сортировка. Будем называть файл ключей $K_1, K_2, \ldots, K_N$ "пирамидой", если

$$K_{\lfloor j/2 \rfloor} \geq K_j \qquad \text{при } 1 \leq \lfloor j/2 \rfloor < j \leq N. \tag{3}$$

В этом случае $K_1 \geq K_2$, $K_1 \geq K_3$, $K_2 \geq K_4$ и т.д.; именно это условие выполняется на рис. 24. Из него следует, в частности, что наибольший ключ оказывается "на вершине пирамиды":

$$K_1 = \max(K_1, K_2, \ldots, K_N). \tag{4}$$

Если бы мы сумели как-нибудь преобразовать произвольный исходный файл в пирамиду, то для получения эффективного алгоритма сортировки можно было бы воспользоваться "нисходящей" процедурой выбора, подобной той, которая описана выше.

Эффективный подход к задаче построения пирамиды предложил Р. У. Флойд [*CACM*, **7** (1964), 701]. Пусть нам удалось расположить файл таким образом, что

$$K_{\lfloor j/2 \rfloor} \geq K_j \qquad \text{при } l < \lfloor j/2 \rfloor < j \leq N, \tag{5}$$

где l—некоторое число ≥ 1. (В исходном файле это условие выполняется "автоматически" для $l = \lfloor N/2 \rfloor$, поскольку ни один индекс j не удовлетворяет условию $\lfloor N/2 \rfloor < \lfloor j/2 \rfloor < j \leq N$.) Нетрудно понять, как, изменяя лишь поддерево с корнем в узле l, преобразовать файл, чтобы распространить неравенства (5) и на случай, когда $\lfloor j/2 \rfloor = l$. Следовательно, можно уменьшать l на единицу, пока в конце концов не будет достигнуто условие (3). Эти идеи Уильямса и Флойда приводят к изящному алгоритму, который заслуживает пристального изучения.

Алгоритм Н. *(Пирамидальная сортировка.)* Записи $R_1, \ldots, R_N$ переразмещаются на том же месте; после завершения сортировки их ключи будут упорядочены: $K_1 \leq \ldots \leq K_N$. Сначала файл перестраивается в пирамиду, после чего вершина пирамиды многократно исключается и записывается на свое окончательное место. Предполагается, что $N \geq 2$.

H1 [Начальная установка.] Установить $l \leftarrow \lfloor N/2 \rfloor + 1$, $r \leftarrow N$.

H2 [Уменьшить l или r.] Если. $l > 1$, то установить $l \leftarrow l-1$, $R \leftarrow R_l$, $K \leftarrow K_l$. (Если $l > 1$, это означает, что происходит процесс преобразования исходного файла в пирамиду; если же $l = 1$, то это значит, что ключи $K_1, K_2, \ldots, K_r$ уже образуют пирамиду.) В противном случае установить $R \leftarrow R_r$, $K \leftarrow K_r$, $R_r \leftarrow R_1$ и $r \leftarrow r-1$; если в результате оказалось, что $r = 1$, то установить $R_1 \leftarrow R$ и завершить работу алгоритма.

H3 [Приготовиться к "протаскиванию".] Установить $j \leftarrow l$. (К этому моменту

$$K_{\lfloor j/2 \rfloor} \geq K_j \qquad \text{при } l < \lfloor j/2 \rfloor < j \leq r, \tag{6}$$

а записи R_k, $r < k \leq N$, занимают свои окончательные места. Шаги **H3–H8** называются алгоритмом "протаскивания"; их

Picture: Рис. 26. Пирамидальная сортировка; пунктиром обведен алгоритм "протаскивания".

действие эквивалентно установке $R_l \leftarrow R$ с последующим перемещением записей $R_l, \ldots, R_r$ таким образом, чтобы условие (6) выполнялось и при $\lfloor j/2 \rfloor = l$.)

H4 [Продвинуться вниз.] Установить $i \leftarrow j$ и $j \leftarrow 2j$. (В последующих шагах $i = \lfloor j/2 \rfloor$.) Если $j < r$, то перейти к шагу **H5**; если $j = r$, то перейти к шагу **H6**; если же $j > r$, то перейти к шагу **H8**.

H5 [Найти "большего" сына.] Если $K_j < K_{j+1}$, то установить $j \leftarrow j + 1$.

H6 [Больше K?] Если $K \geq K_j$, то перейти к шагу **H8**.

H7 [Поднять его вверх.] Установить $R_i \leftarrow R_j$ и возвратиться к шагу **H4**.

H8 [Занести R.] Установить $R_i \leftarrow R$. (На этом алгоритм "протаскивания", начатый в шаге **H3**, заканчивается.) Возвратиться к шагу **H2**. ∎

Пирамидальную сортировку иногда описывают как -алгоритм; это обозначение указывает на характер изменения переменных l и r. Верхний треугольник соответствует фазе построения пирамиды, когда $r = N$, а l убывает до 1; нижний треугольник представляет фазу выбора, когда $l = 1$, а r убывает до 1. В табл. 2 показан процесс пирамидальной сортировки все тех же шестнадцати чисел. (В каждой строке изображено состояние после шага H2, скобки указывают на значения переменных l и r.)

Таблица 2

Пример пирамидальной сортировки

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r	K
503	087	512	061	908	170	897	[275	653	426	154	509	612	677	765	703]	8	16	275
503	087	512	061	908	170	[897	703	653	426	154	509	612	677	765	275]	7	16	897
503	087	512	061	908	[170	897	703	653	426	154	509	612	677	765	275]	6	16	170
503	087	512	061	[908	612	897	703	653	426	154	509	170	677	765	275]	5	16	908
503	087	512	[061	908	612	897	703	653	426	154	509	170	677	765	275]	4	16	061
503	087	[512	703	908	612	897	275	653	426	154	509	170	677	765	061]	3	16	512
503	[087	897	703	908	612	765	275	653	426	154	509	170	677	512	061]	2	16	087
[503	908	897	703	426	612	765	275	653	087	154	509	170	677	512	061]	1	16	503
[908	703	897	653	426	612	765	275	503	087	154	509	170	677	512]	908	1	15	061
[897	703	765	653	426	612	677	275	503	087	154	509	170	061]	897	908	1	14	512
[765	703	677	653	426	612	512	275	503	087	154	509	170]	765	897	908	1	13	061
[703	653	677	503	426	612	512	275	061	087	154	509]	703	765	897	908	1	12	170
[677	653	612	503	426	509	512	275	061	087	154]	677	703	765	897	908	1	11	170
[653	503	612	275	426	509	512	170	061	087]	653	677	703	765	897	908	1	10	154
[612	503	512	275	426	509	154	170	061]	612	653	677	703	765	897	908	1	9	087
[512	503	509	275	426	087	154	170]	512	612	653	677	703	765	897	908	1	8	061
[509	503	154	275	426	087	061]	509	512	612	653	677	703	765	897	908	1	7	170
[503	426	154	276	170	087]	503	509	512	612	653	677	703	765	897	908	1	6	061
[426	275	154	061	176]	426	503	509	512	612	653	677	703	765	897	908	1	5	087
[275	170	154	061]	275	426	503	509	512	612	653	677	703	765	897	908	1	4	087
[170	087	154]	170	275	426	503	509	512	612	653	677	703	765	897	908	1	3	061
[154	087]	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	2	061
[061]	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	1	061

Программа H. *(Пирамидальная сортировка.)* Записи, находящиеся в ячейках с `INPUT`+1 по `INPUT`+N, сортируются при помощи алгоритма H; регистры принимают следующие значения: `rI1` $\equiv l - 1$, `rI2` $\equiv r - 1$, `rI3` $\equiv i$, `rI4` $\equiv j$, `rI5` $\equiv r - l$, `rA` $\equiv K \equiv R$, `rX` $\equiv R_j$.

```
START  ENT1  N/2             1          H1. Начальная установка l ← ⌊N/2⌋ + 1.
       ENT2  N-1             1          r ← N.
1H     DEC1  1               ⌊N/2⌋      l ← l − 1.
       LDA   INPUT+1,1       ⌊N/2⌋      R ← R_l, K ← K_l.
3H     ENT4  1,1               P        H3. Приготовиться к "протаскиванию". j ← l.
       ENT5  0,2               P
       DEC5  0,1               P        rI5 ← r − j.
       JMP   4P                P        К шагу H4.
5H     LDX   INPUT 4     B+A−D          H5. Найти "большего" сына.
       CMPX  INPUT+1,4   B+A−D
       JOE   6F          B+A−D          Переход, если K_j ≥ K_{j+1}.
       INC4  1                 C        В противном случае установить j ← j + 1.
       DECS  1                 C
9H     LDX   INPUT,4         C+D        rX ← R_j.
```

```
6H      CMPA  INPUT,4       B + A    H6. Больше K?
        JGE   8F            B + A    К H8, если K ≥ K_j.
7H      STX   INPUT,3         B      H7. Поднять его вверх. R_i ← R_j.
4H      ENT3  0,4           B + P    H4. Продвинуться вниз.
        DEC5  0,4           B + P    rI5 ← rI5 − j.
        INC4  0,4           B + P    j ← j + j.
        J5P   5B            B + P    К H5, если j < r.
        J5Z   9B          P − A + D  К H6, если j = r.
8H      STA   INPUT,3         P      H8. Занести R. R_i ← R.
2H      J1P   1B              P      H2. Уменьшить l или r.
        LDA   INPUT+1,2     N − 1    Если l = 1, то установить R ← R_r, K ← K_r.
        LDX   INPUT+1       N − 1
        STX   INPUT+1,2     N − 1    R_r ← R_1.
        DEC2  1             N − 1    r ← r − 1.
        J2P   3B            N − 1    К H3, если r > 1.
        STA   INPUT+1         1      R_1 ← R.
```

Эта программа приблизительно лишь вдвое длиннее программы S, но при больших N она гораздо более эффективна. Ее время работы зависит от

$P = N + \lfloor N/2 \rfloor - 2 =$ число "протаскиваний";

$A =$ число протаскиваний, при которых ключ K в конце попадает во внутренний узел пирамиды;

$B =$ суммарное число ключей, просмотренных во время протаскиваний;

$C =$ число присваиваний $j \leftarrow j + 1$ в шаге H5;

$D =$ число случаев, когда в шаге H4 $j = r$.

Эти величины проанализированы ниже. Как показывает практика, они сравнительно мало отклоняются от своих средних значений:

$$\begin{aligned} A &\approx 0.349N; & B &\approx N\log_2 N - 1.87N; \\ C &\approx \tfrac{1}{2}N\log_2 N - 0.9N; & D &\approx \ln N. \end{aligned} \tag{7}$$

При $N = 1000$, например, четыре эксперимента со случайными исходными данными показали соответственно результаты $(A, B, C, D) = (371, 8055, 4056, 12), (351, 8072, 4087, 14), (341, 8094, 4017, 8), (340, 8108, 4083, 13)$. Общее время работы $7A + 14B + 4C + 20N - 2D + 15\lfloor N/2 \rfloor - 28$ равно, таким образом, в среднем примерно $16N\log_2 N + 0.2N$ единиц.

Глядя на табл. 2, трудно поверить в то, что пирамидальная сортировка так уж эффективна: большие ключи перемещаются влево прежде, чем мы успеваем отложить их вправо! Это и в самом деле странный способ сортировки при малых N. Время сортировки 16 ключей из табл. 2 равно $1068u$, тогда как обычный метод простых вставок (программа 5.2.1S) требует всего $514u$. При сортировке простым выбором (программа S) требуется $853u$.

При больших N программа H более эффективна. Напрашивается сравнение с сортировкой методом Шелла с убывающим шагом (программа 5.2.1D) и быстрой сортировкой Хоара (программа 5.2.2Q), так как во всех трех программах сортировка производится путем сравнения ключей, причем вспомогательной памяти используется мало или она не используется вовсе. При $N = 1000$ средние времена работы равны приблизительно

$160000u$ для пирамидальной сортировки;

$130000u$ для сортировки методом Шелла;

$80000u$ для быстрой сортировки.

(MIX—типичный представитель большинства современных вычислительных машин, но, разумеется, на конкретных машинах получатся несколько иные относительные величины.) С ростом N пирамидальная сортировка превзойдет по скорости метод Шелла, но асимптотическая формула $16N\log_2 N \approx 23.08N\ln N$ никогда не станет лучше выражения для быстрой сортировки $12.67N\ln N$. Модификация пирамидальной сортировки, обсуждаемая в упр. 18, ускорит процесс на многих вычислительных машинах, но даже с этим усовершенствованием пирамидальная сортировка не достигнет скорости быстрой сортировки.

С другой стороны, быстрая сортировка эффективна лишь в среднем; в наихудшем случае ее время работы пропорционально N^2. Пирамидальная же сортировка обладает тем интересным свойством, что для нее наихудший случай не намного хуже среднего. Всегда выполняются неравенства

$$A \le 1.5N, \quad B \le N\lfloor\log_2 N\rfloor, \quad C \le N\lfloor\log_2 N\rfloor; \tag{8}$$

таким образом, независимо от распределения исходных данных выполнение программы H не займет более $18N\lfloor\log_2 N\rfloor + 38N$ единиц времени. Пирамидальная сортировка—первый из рассмотренных нами до сих пор методов сортировки, время работы которого *заведомо* имеет порядок $N \log N$. Сортировка посредством слияний, которая будет обсуждаться ниже, в п. 5.2.4, тоже обладает этим свойством, но она требует больше памяти.

Наибольший из включенных—первым исключается. В гл. 2 мы видели, что линейные списки часто можно осмысленно расклассифицировать по характеру производимых над ними операций включения и исключения. *Стек* ведет себя по принципу "последним включается—первым исключается" в том смысле, что при каждом исключении удаляется самый молодой элемент списка (элемент, который был вставлен позже всех других элементов, присутствующих в данный момент в списке). Простая *очередь* ведет себя по принципу "первым включается—первым исключается" в том смысле, что при каждом исключении удаляется самый старший из имеющихся элементов. В более сложных ситуациях, таких, как моделирование лифта в п. 2.2.5, требуется список типа "наименьший из включенных—первым исключается", где при каждом исключении удаляется элемент, имеющий наименьший ключ. Такой список можно назвать *приоритетной очередью*, так как ключ каждого элемента отражает его относительную способность быстро покинуть список. Сортировка посредством выбора—частный случай приоритетной очереди, над которой производится сначала N операций вставки, а затем N операций удаления.

Приоритетные очереди возникают в самых разнообразных приложениях. Например, в некоторых численных итеративных схемах повторяется выбор элемента, имеющего наибольшее (или наименьшее) значение некоторого проверочного критерия; параметры выбранного элемента изменяются, и он снова вставляется в список с новым проверочным значением,- соответствующим новым значениям параметров. Приоритетные очереди часто используются в операционных системах при планировании заданий. Другие типичные применения приоритетных очередей упоминаются в упр. 15, 29 и 36; кроме того, много примеров встретится в последующих главах.

Как же реализовать приоритетную очередь? Один из очевидных способов—поддерживать отсортированный список элементов, упорядоченных по. ключам. Тогда включение нового элемента, по существу, сводится к задаче, рассмотренной нами при изучении сортировки вставками в п. 5.2.1. При другом, еще более очевидном способе работы с приоритетной очередью элементы в списке хранятся в произвольном порядке, и тогда для выбора нужного элемента, приходится осуществлять поиск наибольшего (или наименьшего) ключа каждый раз, когда необходимо произвести исключение. В обоих этих очевидных подходах неприятность состоит в том, что требуется порядка N шагов для выполнения либо операции вставки, либо операции удаления, если в списке содержится N элементов, т. е. при больших N эти операции занимают слишком много времени.

В своей статье о пирамидальной сортировке Уильяме указал на то, что пирамиды идеально подходят для приложений с большими приоритетными очередями, так как элемент можно вставить в пирамиду или удалить из нее за $O(\log N)$ шагов; к тому же все элементы пирамиды компактно располагаются в последовательных ячейках памяти. Фаза выбора в алгоритме H—это последовательность шагов удаления в процессе типа *наибольший из включенных—первым исключается*: чтобы исключить наибольший элемент K_1 мы удаляем его и "протаскиваем" элемент K_N в новой пирамиде из $N-1$ элементов. (Если нужен алгоритм типа *наименьший из включенных—первым исключается*, как при моделировании лифта, то, очевидно, можно изменить определение пирамиды, заменив в (3) знак "$\geq$" на "$\leq$"; для удобства мы будем рассматривать здесь лишь случай "наибольший из включенных—первым исключается".) Вообще, если требуется исключить наибольший элемент, а затем вставить новый элемент x, то можно выполнить процедуру протаскивания с $l = 1$, $r = N$ и $K = x$. Если же необходимо вставить элемент без предварительного исключения, то можно воспользоваться "восходящей" процедурой из упр. 16.

Связанное представление приоритетных очередей. Эффективный способ представления приоритетных очередей в виде связанных бинарных деревьев предложил в 1971 г. Кларк Э. Крэйн. Его метод требует наличия в каждой записи двух полей связи и короткого поля счетчика, но по сравнению с пирамидами он обладает следующими преимуществами:

1) Если с приоритетной очередью работают как со стеком, то операции включения и исключения более эффективны (они занимают фиксированное время, не зависящее от длины очереди).
2) Записи никогда не перемещаются, изменяются только указатели.
3) Можно слить две непересекающиеся приоритетные очереди, содержащие в общей сложности N элементов, в одну всего за $O(\log N)$ шагов.

Слегка видоизмененный метод Крэйна проиллюстрирован на рис. 27, на котором показан особый тип структуры бинарного дерева. Каждый узел содержит поле `KEY`, поле `DIST` и два поля связи—`LEFT` и `RIGHT`. Поле `DIST` всегда устанавливается равным длине кратчайшего пути от этого узла до концевого узла (т. е. до пустого узла Λ) дерева. Если считать, что $\texttt{DIST}(\Lambda) = 0$ и $\texttt{KEY}(\Lambda) = -\infty$, то поля `KEY` и `DIST` в этом

дереве удовлетворяют следующим соотношениям:

$$\texttt{KEY(P)} \geq \texttt{KEY(LEFT(P))}, \texttt{KEY(P)} \geq \texttt{KEY(RIGHT(P))}; \tag{9}$$

$$\texttt{DIST(P)} = 1 + \min(\texttt{DIST(LEFT(P))}, \texttt{DIST(RIGHT(P))}); \tag{10}$$

$$\texttt{DIST(LEFT(P))} \geq \texttt{DIST(RIGHT(P))}. \tag{11}$$

Соотношение (9) аналогично условию пирамиды (3) и служит гарантией того, что в корне дерева находится наибольший ключ, а соотношение (10)—это просто определение поля DIST, сформулированное выше. Соотношение (11) представляет собой интересное новшество: из него следует, что кратчайший путь к концевому узлу всегда можно получить, двигаясь вправо. Мы

Picture: Рис. 27. Приоритетная очередь, представленная в виде левостороннего дерева.

будем называть бинарное дерево с этим свойством левосторонним деревом, поскольку оно, как правило, сильно "тянется" влево.

Из этих определений ясно, что равенство $\texttt{DIST(P)} = n$ подразумевает существование по крайней мере 2^n концевых узлов ниже P; в противном случае нашелся бы более, короткий путь от P до концевого узла. Таким образом, если в левостороннем дереве имеется N узлов, то путь, ведущий из корня вниз по направлению вправо, содержит не более чем $\lfloor \log_2(N+1) \rfloor$ узлов. Новый узел можно вставить в приоритетную очередь, пройдя по этому пути (см. упр. 32); следовательно, в худшем случае необходимо всего $O(\log N)$ шагов. Наилучший случай достигается, когда дерево линейно (все связи RIGHT равны Λ), а наихудший случай достигается, когда дерево абсолютно сбалансировано.

Чтобы удалить узел из корня, нужно просто слить два его поддерева. Операция слияния двух непересекающихся левосторонних деревьев, на которые ссылаются указатели P и Q, по своей идее проста: если $\texttt{KEY(P)} \geq \texttt{KEY(Q)}$, то берем в качестве корня P и сливаем Q с правым поддеревом P; при этом изменится DIST(P), а LEFT(P) меняется местами с RIGHT(P), если это необходимо. Нетрудно составить подробное описание этого процесса (см. упр. 32).

Сравнение методов работы с приоритетными очередями. Если число узлов N мало, то для поддержания приоритетной очереди лучше всего применять один из простых методов с использованием линейных списков. Если же N велико, то, очевидно, гораздо более быстрым будет метод, время работы которого порядка $\log N$. Поэтому большие приоритетные очереди обычно представляют в виде пирамид или левосторонних деревьев. В п. 6.2.3 мы обсудим еще один способ представления линейных списков в виде *сбалансированных деревьев*, который приводит к третьему методу, пригодному для представления приоритетных очередей, с временем работы порядка $\log N$. Поэтому уместно сравнить эти три метода.

Мы видели, что операции над левосторонними деревьями в целом несколько быстрее, чем операции над пирамидами, но пирамиды занимают меньше памяти. Сбалансированные деревья занимают примерно столько же памяти, сколько левосторонние деревья (быть может, чуть меньше); операции над ними медленнее, чем над пирамидами, а программирование сложнее, но структура сбалансированных деревьев в некоторых отношениях существенно более гибкая. Работая с пирамидами, не так просто предсказать, что произойдет с элементами, если у них равные ключи; нельзя гарантировать, что элементы с равными ключами будут обрабатываться по принципу "последним включается— первым исключается" или "первым включается—первым исключается", если только ключ не расширен и не содержит дополнительного поля "порядковый номер вставки", и тогда равных ключей просто нет. С другой стороны, если применять сбалансированные деревья, можно легко оговорить твердые условия относительно равных ключей. Можно также выполнять такие действия, как "вставить x непосредственно перед (или после) y". Сбалансированные деревья симметричны, так что в любой момент можно исключить либо наибольший, либо наименьший элемент, в то время как левосторонние деревья и пирамиды должны быть так или иначе ориентированы. (См. тем не менее упр. 31, в котором

Picture: Рис. 28. Так выглядит пирамида...

показано, как строить *симметричные* пирамиды.) Сбалансированные деревья можно использовать как для поиска, так и для сортировки; и из сбалансированного дерева можно довольно быстро удалять последовательные блоки элементов. Но два сбалансированных дерева нельзя слить менее чем за $O(N)$ шагов, в то время как два левосторонних дерева можно слить всего за $O(\log N)$ шагов.

Итак, пирамиды наиболее экономны с точки зрения памяти; левосторонние деревья хороши тем, что можно быстро слить две непересекающиеся приоритетные очереди; и, если нужно, за умеренное вознаграждение можно получить ту гибкость, какую предоставляют сбалансированные деревья.

*** Анализ пирамидальной сортировки.** Алгоритм H до сих пор не был полностью проанализирован, но некоторые его свойства можно вывести без особого труда. Поэтому мы завершим этот пункт довольно подробным исследованием, касающимся пирамид.

На рис. 28 показана форма пирамиды из 26 элементов; каждый узел помечен двоичным числом, соответствующим его индексу в пирамиде. Звездочками в этой диаграмме помечены так называемые *особые узлы*, которые лежат на пути от 1 к N.

Одна из наиболее важных характеристик пирамиды—набор размеров ее поддеревьев. Например, на рис. 28 размеры поддеревьев с корнями в узлах 1,2, ..., 26 равны соответственно

$$26^*, 15, 10^*, 7, 7, 6^*, 3, 3, 3, 3, 3, 3, 2^*, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1^*. \tag{12}$$

Звездочками отмечены *особые поддеревья* с корнями в особых узлах; в упр. 20 показано, что если N имеет двоичное представление

$$N = (b_n b_{n-1} \ldots b_1 b_0)_2, \qquad n = \lfloor \log_2 N \rfloor, \tag{13}$$

то размеры особых поддеревьев равны

$$(1b_{n-1} \ldots b_1 b_0)_2, (1b_{n-2} \ldots b_1 b_0)_2, \ldots, (1b_1 b_0)_2, (1b_0)_2, (1)_2. \tag{14}$$

(Неособые поддеревья всегда абсолютно сбалансированы, так что их размеры всегда имеют вид $2^k - 1$. В упр. 21 показано, что среди неособых поддеревьев имеется ровно

$$\begin{array}{llll} \lfloor (N-1)/2 \rfloor \text{ размера } 1, & & \lfloor (N-2)/4 \rfloor \text{ размера } 3, \\ \lfloor (N-4)/8 \rfloor \text{ размера } 7, & \ldots & \lfloor (N-2^{n-1})/2^n \rfloor \text{ размера } (2^n-1), \end{array} \tag{15}$$

Например, на рис. 28 изображено двенадцать неособых поддеревьев размера 1, шесть поддеревьев размера 3, два—размера 7 и одно—размера 15.

Пусть s_l—размер поддерева с корнем l, а M_N—мультимножество $\{s_1, s_2, \ldots, s_N\}$ всех этих размеров. Используя (14) и (15), легко вычислить M_N при любом заданном N. В упр. 5.1.4–20 показано, что общее число способов построить пирамиду из целых чисел $\{1, 2, \ldots, N\}$ равно

$$N!/s_1 s_2 \ldots s_N = N!/\prod_{s \in M_N} s. \tag{16}$$

Например, число способов расположить 26 букв $\{A, B, C, \ldots, Z\}$ на рис. 28 так, чтобы по вертикали сохранялся алфавитный порядок, равно

$$26!/(26 \cdot 10 \cdot 6 \cdot 2 \cdot 1 \cdot 1^{12} \cdot 3^6 \cdot 7^2 \cdot 15^1).$$

Теперь мы в состоянии проанализировать фазу построения пирамиды в алгоритме H, т. е. вычисления, которые завершаются до того, как в шаге H2 впервые выполнится условие $l = 1$. К счастью, благодаря следующей ниже теореме анализ построения пирамиды можно свести к изучению независимых операций протаскивания.

Теорема H. *Если исходными данными для алгоритма H служит случайная перестановка множества* $\{1, 2, \ldots, N\}$, *то в фазе построения пирамиды с одинаковой вероятностью может получиться любая из* $N!/\left(\prod_{s \in M_N} s\right)$ *возможных пирамид. Более того, все* $\lfloor N/2 \rfloor$ *операций протаскивания, выполненные за время этой фазы, "равномерны" в том смысле, что по достижении шага H8 все* s_l *возможных значений переменной* i *равновероятны.*

Доказательство Применим метод, который в численном анализе называется методом "обратной задачи". Пусть в качестве одного из возможных результатов операции протаскивания задана пирамида $K_1 \ldots K_N$ с корнем в узле l; тогда ясно, что имеется всего s_l исходных конфигураций $K'_1 \ldots K'_N$ файла, которые после протаскивания дают такой результат. Все эти исходные конфигурации имеют различные значения K'_l, следовательно, рассуждая в обратном направлении, существует ровно $s_l\ s_{l+1} \ldots s_N$ исходных перестановок множества $\{1, 2, \ldots, N\}$, которые после завершения операции протаскивания в позицию l дают конфигурацию $K_1 \ldots K_N$.

Случай $l = 1$ типичен: пусть $K_1 \ldots K_N$—пирамида, и пусть $K'_1 \ldots K'_N$—файл, который преобразуется в $K_1 \ldots K_N$ в результате протаскивания при $l = 1$, $K = K'_1$. Если $K = K_i$, то должны иметь место равенства $K'_i = K_{\lfloor i/2 \rfloor}$, $K'_{\lfloor i/2 \rfloor} = K_{\lfloor i/4 \rfloor}$ и т. д., при этом $K'_j = K_j$ для всех j, не лежащих на пути от 1 к i. Обратно, при любом i в результате такого построения получается файл $K'_1 \ldots K'_N$, такой, что (a) операция протаскивания преобразует файл $K'_1 \ldots K'_N$ в $K_1 \ldots K_N$ и (b) $K_{\lfloor j/2 \rfloor} \geq K_j$ при $2 \leq \lfloor j/2 \rfloor < j \leq N$. Следовательно, возможны ровно N таких файлов $K'_1 \ldots K'_N$, и операция протаскивания равномерна. (Пример доказательства этой теоремы см. в упр. 22.)∎

Обращаясь к величинам A, B, C, D в анализе программы H, можно видеть, что равномерная операция протаскивания, производимая над поддеревом размера s, дает вклад $\lfloor s/2 \rfloor / s$ в среднее значение величины A; ее вклад в среднее значение величины B равен

$$\frac{1}{s}(0+1+1+2+\cdots+\lfloor \log_2 s \rfloor) = \frac{1}{s}\left(\sum_{1\le k\le s} \lfloor \log_2 k \rfloor\right) = \frac{1}{s}((s+1)\lfloor \log_2 s \rfloor - 2^{\lfloor \log_2 s \rfloor + 1} + 2)$$

(см. упр. 1.2.4–42); и она дает вклад $2/s$ или 0 в среднее значение величины D в зависимости от того, является ли s четным или нечетным. Несколько сложнее определить соответствующий вклад в среднее значение величины C, так что эта задача предоставляется читателю (см. упр. 26). Производя суммирование по всем операциям протаскивания, находим, что среднее значение величины A за время построения пирамиды равно

$$A'_N = \sum_{s\in M_N} \lfloor s/2 \rfloor / s; \tag{17}$$

аналогичные формулы имеют место и для B, C и D, так что можно без особого труда точно вычислить эти средние значения. В следующей таблице приведены типичные результаты:

N	A'_N	B'_N	C'_N	D'_N
99	19.18	68.35	42.95	0.00
100	19.93	69.39	42.71	1.84
999	196.16	734.66	464.53	0.00
1000	196.94	735.80	464.16	1.92
9999	1966.02	7428.18	4695.54	0.00
10000	1966.82	7429.39	4695.06	1.97
10001	1966.45	7430.07	4695.84	0.00
10002	1967.15	7430.97	4695.95	1.73

Что касается асимптотики, то в M_N можно не обращать внимания на размеры особых поддеревьев, и тогда мы найдем, например, что

$$A'_N = \frac{N}{2}\cdot\frac{0}{1} + \frac{N}{4}\cdot\frac{1}{3} + \frac{N}{8}\cdot\frac{3}{7} + \cdots + O(\log N) = \left(1 - \frac{1}{2}\alpha\right)N + O(\log N), \tag{18}$$

где

$$\alpha = \sum_{k\ge 1} \frac{1}{2^k - 1} = 1.60669\ 51524\ 15291\ 76378\ 33015\ 23190\ 92458\ 04805\text{—}. \tag{19}$$

(Это значение получил Дж. У. Ренч младший, пользуясь преобразованием ряда из упр. 27.) При больших N можно использовать приближенные формулы

$$\begin{aligned}
&A'_N \approx 0.1967N + 0.3\ (N \text{ чет.}), \quad 0.1967N - 0.3 (N \text{ нечет.});\\
&B'_N \approx 0.74403N - 1.3\ln N;\\
&C'_N \approx 0.47034N - 0.8\ln N;\\
&D'_N \approx 1.8 \pm 0.2\ (N \text{ чет.}), \quad 0 (N \text{ нечет.}).
\end{aligned} \tag{20}$$

Нетрудно определить также максимальные и минимальные значения. Для построения пирамиды требуется всего $O(N)$ шагов (ср. с упр. 23).

Этим, по существу, завершается анализ фазы построения пирамиды в алгоритме H. Анализ фазы выбора—совсем другая задача, которая еще ожидает своего решения! Пусть пирамидальная сортировка применяется к N элементам; обозначим через A''_N, B''_N, C''_N и D''_N средние значения величин A, B, C и D во время фазы выбора. Поведение алгоритма H подвержено сравнительно малым колебаниям около эмпирически установленных средних значений

$$\begin{aligned}
&A''_N \approx 0.152N;\\
&B''_N \approx N\log_2 N - 2.61N;\\
&C''_N \approx \frac{1}{2}N\log_2 N - 1.4N;\\
&D''_N \approx \ln N \pm 2;
\end{aligned} \tag{21}$$

тем не менее до сих пор не найдено правильного теоретического объяснения этим константам. Рассмотрев отдельные операции протаскивания, нетрудно вывести верхние оценки, указанные в неравенствах (8), хотя, если рассматривать алгоритм как целое, верхнюю оценку для C, по-видимому, следует уменьшить приблизительно до $\frac{1}{2}N\log_2 N$.

Упражнения

1. [10] Является ли сортировка посредством простого выбора (алгоритм S) "устойчивой"?
2. [15] Почему в алгоритме S оказывается более удобным находить сначала наибольший элемент, затем наибольший из оставшихся и т. д , вместо того чтобы находить сначала наименьший элемент, затем наименьший из оставшихся и т. д.?
3. [M21] (a) Докажите, что если алгоритм S применяется к случайной перестановке множества $\{1, 2, \ldots, N\}$, то в результате первого выполнения шагов S2 и S3 получается случайная перестановка множества $\{1, 2, \ldots, N-1\}$, за которой следует N. (Иначе говоря, файл $K_1 \ldots K_{N-1}$ с одинаковой вероятностью может быть любой перестановкой множества $\{1, 2, \ldots, N-1\}$.) (b) Следовательно, если через B_N обозначить среднее значение величины B в программе S, то при условии, что исходный файл упорядочен случайным образом, имеем $B_N = H_N - 1 + B_{N-1}$. [*Указание:* ср. со статистическими характеристиками 1.2.10–16.]

>4. [M35] В шаге S3 алгоритма S ничего не делается, если $i = j$. Не лучше ли перед выполнением шага S3 проверить условие $i = j$? Чему равно среднее число случаев выполнения условия $i = j$ в шаге S3, если исходный файл случаен?

5. [20] Чему равно значение величины B в анализе программы S для исходного файла $N \ldots 3\ 2\ 1$?
6. [M29] (a) Пусть $a_1\ a_2\ \ldots\ a_N$—перестановка множества $\{1, 2, \ldots, N\}$ с C циклами, I инверсиями и такая, что при ее сортировке с помощью программы S производится B обменов на правосторонний максимум. Докажите, что $2B \le I+N-C$. [*Указание:* см. упр. 5.2.2–1.] (b) Покажите, что $I+N-C \le \lfloor n^2/2 \rfloor$; следовательно, B не превышает $\lfloor n^2/4 \rfloor$.
7. [M46] Найдите дисперсию величины B в программе S как функцию от N, считая, что исходный файл случаен.
8. [24] Покажите, что если при поиске $\max(K_1, \ldots, K_j)$ в шаге S2 просматривать ключи слева направо: K_1, K_2, ..., K_j, а не наоборот: K_j, ..., K_2, K_1, как в программе S, то за счет этого можно было бы сократить число сравнений при следующих повторениях шага S2. Напишите `MIX`-программу, основанную на этом наблюдении.
9. [M25] Чему равно среднее число сравнений, выполняемых алгоритмом из упр. 8 для случайного исходного файла?
10. [12] Как будет выглядеть дерево, изображенное на рис. 23, после того как будут выведены 14 из 16 первоначальных элементов?
11. [10] Как будет выглядеть дерево, изображенное на рис. 24, после вывода элемента 908?
12. [M20] Сколько раз будет выполнено сравнение $-\infty$ с ∞, если применить "восходящий" метод, представленный на рис. 23, для упорядочения файла из 2^n элементов?
13. [20] (Дж. У. Дж. Уильяме.) В шаге H4 алгоритма H различаются три случая: $j < r$, $j = r$ и $j > r$. Покажите, что если $K \ge K_{r+1}$, то можно было бы так упростить шаг H4, чтобы разветвление происходило лишь по двум путям. Как надо изменить шаг H2, чтобы обеспечить в процессе пирамидальной сортировки выполнение условия $K \ge K_{r+1}$?
14. [10] Покажите, что простая очередь—частный случай приоритетной. (Объясните, какие ключи нужно присваивать элементам, чтобы процедура "наибольший из включенных—первым исключается" была эквивалентна процедуре "первым включается—первым исключается".) Является ли стек также частным случаем приоритетной очереди?

>15. [M22] (В. Э. Чартрс.) Придумайте быстрый алгоритм построения таблицы простых чисел $\le N$, в котором используется *приоритетная очередь* с целью избежать операций деления. [*Указание.* Пусть наименьший клюя в приоритетной очереди будет наименьшим нечетным непростым числом, большим, чем самое последнее нечетное число, воспринятое как кандидат в простые числа. Попытайтесь свести к минимуму число элементов в этой очереди.]

16. [20] Постройте эффективный алгоритм, который вставляет новый ключ в данную пирамиду из n элементов, порождая пирамиду из $n+1$ элементов.
17. [20] Алгоритм из упр. 16 можно использовать для построения пирамиды взамен метода "уменьшения l до 1", применяемого в алгоритме H.

    Порождают ли оба метода из одного и того же исходного файла одну и ту же пирамиду?

>18. [21] (Р. У. Флойд) Во время фазы выбора в алгоритме пирамидальной сортировки ключ K, как правило, принимает довольно малые значения, и поэтому почти при всех сравнениях в шаге H6 обна-

руживается, что $K < K_j$. Как. можно изменить алгоритм, чтобы ключ K не сравнивался с K_j в основном цикле вычислений?

19. [21] Предложите алгоритм исключения данного элемента из пирамиды размера N, порождающий пирамиду размера $N-1$.
20. [M20] Покажите, что формулы (14) задают размеры особых поддеревьев пирамиды.
21. [M24] Докажите, что формулы (15) задают размеры неособых поддеревьев пирамиды.
>22. [20] Какие перестановки множества $\{1,2,3,4,5\}$ фаза ностроения пирамиды в алгоритме H преобразует в 5 3 4 1 2?
23. [M28] (a) Докажите, что длина пути B в алгоритме протаскивания никогда не превышает $\lfloor \log_2(r/l) \rfloor$. (b) Согласно неравенствам (8), ни при каком конкретном применении алгоритма H величина B не может превзойти $N\lfloor \log_2 N \rfloor$. Найдите максимальное значение B по всевозможным исходным файлам как функцию от N. (Вы должны доказать, что существует исходный файл, на котором B принимает это максимальное значение.)
24. [M24] Выведите точную формулу стандартного отклонения величины B'_N (суммарная длина пути, пройденного по дереву во время фазы построения пирамиды в алгоритме H).
25. [M20] Чему равен средний вклад в значение величины C за время первой операции протаскивания, когда $l = 1$, а $r = N$, если $N = 2^{n+1} - 1$.
26. [M30] Решите упр. 25: (a) для $N = 26$, (b) для произвольного N.
27. [M25] (Дж. У. Ренч мл.) Докажите, что $\sum_{n\ge1} x^n/(1-x^n) = \sum_{n\ge1} x^{n^2}(1+x^n)/(1-x^n)$. [Положив $x = \frac{1}{2}$, получите очень быстро сходящийся ряд для вычисления (19).]
28. [35] Продумайте идею *тернарных пирамид*, основанных на полных тернарных, а не бинарных деревьях. Будет ли тернарная пирамидальная сортировка быстрее бинарной?
29. [26] (У. С. Браун.) Постройте алгоритм умножения многочленов или степенных рядов $(a_1x^{i_1} + a_2x^{i_2} + \cdots)(b_1x^{j_1} + b_2x^{j_2} + \cdots)$, который бы порождал коэффициенты произведения $c_1x^{i_1+j_1} + \cdots$ по порядку, по мере того как перемножаются коэффициенты исходных многочленов. [*Указание:* 'воспользуйтесь подходящей приоритетной очередью.]
30. [M48] Может ли величина C превзойти $\frac{1}{2}N\log_2 N$ при пирамидальной сортировке файла? Чему равно максимальное значение C? Чему равно минимальное значение?
31. [37] (Дж. У. Дж. Уильяме.) Покажите, что если две пирамиды подходящим образом совместить "основание к основанию", то это даст возможность поддерживать структуру, в которой в любой момент можно за $O(\log n)$ шагов исключить либо наибольший, либо наименьший элемент. (Такую структуру можно назвать *приоритетным деком*.)
32. [21] Разработайте алгоритм слияния двух непересекающихся приоритетных очередей, представленных в виде левосторонних деревьев, в одну. (В частности, если одна из данных очередей содержит всего один элемент, то ваш алгоритм будет вставлять его в другую очередь.)
>33. [15] Почему в приоритетной очереди, представленной в виде левостороннего дерева, операция удаления корня выполняется путем слияния двух поддеревьев, а не "продвижения" узлов по направлению к вершине, как в пирамиде?
34. [M47] Сколько можно построить левосторонних деревьев из N узлов, если игнорировать значения поля KEY? [Эта последовательность начинается с чисел 1, 1, 2, 4, 8, 17, 38, ...; существует ли какая-нибудь простая асимптотическая формула?]
35. [26] Если в левостороннее дерево с N узлами добавить связи UP (ср. с обсуждением деревьев с тремя связями в п. 6.2.3), то это даст возможность исключать из приоритетной очереди произвольный узел P следующим образом: слить LEFT(P) и RIGHT(P) и поместить полученное поддерево на место P, затем исправлять поля DIST у предков узла P до тех пор, пока не будет достигнут либо корень, либо узел, у которого поле DIST не меняется.

    Докажите, что при этом никогда не потребуется изменить более чем $O(\log N)$ полей DIST, несмотря даже на то, что дерево может содержать. очень длинные восходящие пути.
36. [18] *(Замещение наиболее давно использованной страницы.)* Многие операционные системы используют алгоритм следующего типа: над набором узлов допустимы две операции—"использование" узла и замещение наиболее давно "использованного" узла новым узлом. Какая структура данных облегчает нахождение наиболее давно "использованного" узла?

Сортировка слиянием *Слияние* означает объединение двух или более упорядоченных файлов в один упорядоченный файл. Можно, например, слить два подфайла—503 703 765 и 087 512 677, получив 087 503 512 677 703 765. Простой способ сделать это—сравнить два наименьших элемента, вывести наименьший из них и повторить эту процедуру.

Начав с

$$\begin{cases} 503 & 703 & 765 \\ 087 & 512 & 677 \end{cases},$$

получим

$$087 \begin{cases} 503 & 703 & 765 \\ 512 & 677 \end{cases}$$

затем

$$087\ 503 \begin{cases} 703 & 765 \\ 512 & 677 \end{cases}$$

и т. д. Необходимо позаботиться о действиях на случай, когда исчерпается один из файлов. Весь процесс подробно описан в следующем алгоритме.

Алгоритм M. *(Двухпутевое слияние.)* Этот алгоритм осуществляет слияние двух упорядоченных файлов $x_1 \le x_2 \le \ldots \le x_m$ и $y_1 \le y_2 \le \ldots \le y_n$ в один файл $z_1 \le z_2 \le \ldots \le z_{m+n}$.

M1 [Начальная установка.] Установить $i \leftarrow 1$, $j \leftarrow 1$, $k \leftarrow 1$.

M2 [Найти наименьший элемент.] Если $x_i \le y_j$, то перейти к шагу **M3**; в противном случае перейти к **M5**.

M3 [Вывести x_i.] Установить $z_k \leftarrow x_i$, $k \leftarrow k+1$, $i \leftarrow i+1$. Если $i \le m$, то возвратиться к **M2**.

M4 [Вывести $y_j, \ldots, y_n$.] Установить $(z_k, \ldots, z_{m+n}) \leftarrow (y_j, \ldots, y_n)$ и завершить работу алгоритма.

M5 [Вывести y_j.] Установить $z_k \leftarrow y_j$, $k \leftarrow k+1$, $j \leftarrow j+1$. Если $j \le n$, то возвратиться к **M2**.

M6 [Вывести $x_i, \ldots, x_m$.] Установить $(z_k, \ldots, z_{m+n}) \leftarrow (x_i, \ldots, x_m)$ и завершить работу алгоритма. ∎

Picture: Рис. 29. Слияние $x_1 \le \ldots \le x_m$ с $y_1 \le \ldots \le y_n$.

В п. 5.3.2 мы увидим, что эта простая процедура, по существу, "наилучший из возможных" способов слияния на традиционной ЭВМ, если $m \approx n$. (Но, если m гораздо меньше n, можно разработать более эффективные алгоритмы сортировки, хотя в общем случае они довольно сложны.) Алгоритм M без особой потери эффективности можно немного упростить, добавив в конец исходных файлов искусственных "стражей" $x_{m+1} = y_{n+1} = \infty$ и останавливаясь перед выводом ∞. Анализ алгоритма M см. в упр. 2.

Общий объем работы, выполняемой алгоритмом M, по существу, пропорционален $m+n$, поэтому ясно, что слияние—более простая задача, чем сортировка. Однако задачу сортировки можно свести к слияниям, сливая все более длинные подфайлы до тех пор, пока не будет отсортирован весь файл. Такой подход можно рассматривать как развитие идеи сортировки вставками: вставка нового элемента в упорядоченный файл—частный случай слияния при $n = 1$! Если нужно ускорить процесс вставок, то можно рассмотреть вставку нескольких элементов за раз, "группируя" их, а это естественным образом приводит к общей идее сортировки слиянием. С исторической точки зрения метод слияний— один из самых первых методов, предназначенных для сортировки

Picture: Таблица 1 Сортировка естественным двухпутевым слиянием

на ЭВМ; он был предложен Джоном фон Нейманом еще в 1945 г. (см. § 5.5).

Мы довольно подробно изучим слияния в § 5.4 в связи с алгоритмами внешней сортировки, а в настоящем пункте сосредоточим свое внимание на сортировке в быстрой памяти с произвольным доступом.

Таблица 1 иллюстрирует сортировку слиянием, когда "свечка сжигается с обоих концов", подобно тем процедурам просмотра элементов файла, которые применялись при быстрой сортировке, поразрядной обменной сортировке и т. д. Мы анализируем исходный файл слева и справа, двигаясь к середине. Пропустим пока первую строку и рассмотрим переход от второй строки к третьей. Слева мы видим возрастающий отрезок 503 703 765, а справа, если читать справа налево, имеем отрезок 087 512 677. Слияние этих двух последовательностей дает подфайл 087 503 512 677 703 765, который помещается слева в строке 3. Затем ключи 061 612 908 в строке 2 сливаются с 170 509 897, и результат (061 170 509 612 897 908) записывается *справа* в строке 3. Наконец, 154 275 426 653 сливается с 653 (перекрытие обнаруживается прежде, чем оно может привести к вредным последствиям), и результат записывается слева. Точно так же строка 2 получилась из исходного файла в строке 1.

Вертикальными линиями в табл. 1 отмечены границы между отрезками. Это так называемые "ступеньки вниз", где меньший элемент следует за большим. В середине файла обычно возникает двусмысленная ситуация, когда при движении с обоих концов мы прочитываем один и тот же ключ; это не приведет к осложнениям, если проявить чуточку осторожности, как в следующем алгоритме. Такой метод по традиции называется "естественным" слиянием, потому что он использует отрезки, которые "естественно" образуются в исходном файле.

Алгоритм N. *(Сортировка естественным двухпутевым слиянием.)* При сортировке записей $R_1, \ldots, R_N$ используются две области памяти, каждая из которых может содержать N записей. Для удобства

обозначим записи, находящиеся во второй области, через $R_{N+1}, \ldots, R_{2N}$, хотя в действительности запись R_{N+1} может и не примыкать непосредственно к R_N. Начальное содержимое записей $R_{N+1}, \ldots, R_{2N}$ не имеет значения. После завершения сортировки ключи будут упорядочены: $K_1 \le \ldots \le K_N$.

Picture: Рис. 30. Сортировка слиянием.

N1 [Начальная установка.] Установить $s \leftarrow 0$. (При $s = 0$ мы будем пересылать записи из области $(R_1, \ldots, R_N)$ в область $(R_{N+1}, \ldots, R_{2N})$; при $s = 1$ области по отношению к пересылкам поменяются ролями.)

N2 [Подготовка к просмотру.] Если $s = 0$, то установить $i \leftarrow 1$, $j \leftarrow N$, $k \leftarrow N + 1$, $l \leftarrow 2N$; если $s = 1$, то установить $i \leftarrow N + 1$, $j \leftarrow 2N$, $k \leftarrow 1$, $l \leftarrow N$. (Переменные i, j, k, l указывают текущие позиции во входных "файлах", откуда идет чтение, и в выходных "файлах", куда идет запись.) Установить $d \leftarrow 1$, $f \leftarrow 1$. (Переменная d определяет текущее направление вывода, f устанавливается равной 0, если необходимы дальнейшие просмотры.)

N3 [Сравнение $K_i : K_j$] Если $K_i > K_j$, перейти к шагу **N8**. Если $i = j$, установить $P_k \leftarrow R_i$ и перейти к шагу **N13**.

N4 [Пересылка R_i.] (Шаги **N4–N7** аналогичны шагам M3–M4 алгоритма M.) Установить $R_k \leftarrow R_i$, $k \leftarrow k + d$.

N5 [Ступенька вниз?] Увеличить i на 1. Затем, если $K_{i-1} \le K_i$, возвратиться к шагу **N3**.

N6 [Пересылка R_j.] Установить $R_k \leftarrow R_j$, $k \leftarrow k + d$.

N7 [Ступенька вниз?] Уменьшить j на 1. Затем, если $K_{j+1} \le K_j$, возвратиться к шагу **N6**; в противном случае перейти к шагу **N12**.

N8 [Пересылка R_j.] (Шаги **N8–N11** двойственны по отношению к шагам **N4–N7**.) Установить $R_k \leftarrow R_j$, $k \leftarrow k + d$.

N9 [Ступенька вниз?] Уменьшить j на 1. Затем, если $K_{j+1} \le K_j$, возвратиться к шагу **N3**.

N10 [Пересылка R_i.] Установить $R_k \leftarrow R_i$, $k \leftarrow k + d$.

N11 [Ступенька вниз?] Увеличить i на 1. Затем, если $K_{i-1} \le K_i$, возвратиться к шагу **N10**.

N12 [Переключение направления.] Установить $f \leftarrow 0$, $d \leftarrow -d$ и взаимозаменить $k \leftrightarrow l$. Возвратиться к шагу **N3**.

N13 [Переключение областей.] Если $f = 0$, то установить $s \leftarrow 1 - s$ и возвратиться к **N2**. В противном случае сортировка завершена; если $s = 0$, то установить $(R_1, \ldots, R_N) \leftarrow (R_{N+1}, \ldots, R_{2N})$. (Если результат можно оставить в области $(R_{N+1}, \ldots, R_{2N})$, то последнее копирование необязательно.) ∎

В этом алгоритме есть одна небольшая тонкость, которая объясняется в упр. 5.

Запрограммировать алгоритм N для машины `MIX` нетрудно, но основные сведения о его поведении можно получить и без построения всей программы. Если файл случаев, то в нем около $\frac{1}{2}N$ возрастающих отрезков, так как $K_i > K_{i+1}$ с вероятностью $\frac{1}{2}$; подробная информация о числе отрезков при несколько отличных предположениях была получена в п. 5.1.3. При каждом просмотре число отрезков сокращается вдвое (за исключением необычных случаев, таких, как ситуация, описанная в упр. 6). Таким образом, число просмотров, как правило, составляет около $\log_2 N$. При каждом просмотре мы должны переписать все N записей, и, как показано в упр. 2, бóльшая часть времени затрачивается в шагах N3, N4, N5, N8, N9. Если считать, что равные ключи встречаются с малой вероятностью, то время, затрачиваемое во внутреннем цикле, можно охарактеризовать следующим образом:

Шаг		Операции	Время
	$N3$	`CMPA, JG, JE`	$3.5u$
Либо	$N4$	`STA, INC`	$3u$
	$N5$	`INC, LDA, CMPA, JGE`	$6u$
Либо	$N8$	`STX, INC`	$3u$
	$N9$	`DEC, LDX, CMPX, JGE`	$6u$

Таким образом, при каждом просмотре на каждую запись затрачивается 12.5 единиц времени, и общее время работы асимптотически приближается к $12.5N \log_2 N$ как в среднем, так и в наихудшем случае. Это медленнее быстрой сортировки и не настолько лучше времени работы пирамидальной сортировки, чтобы оправдать вдвое больший расход памяти, так как асимптотическое время работы программы 5.2.3H равно $16N \log_2 N$.

В алгоритме N границы между отрезками полностью определяются "ступеньками вниз". Такой подход обладает тем возможным преимуществом, что исходные файлы с преобладанием возрастающего *или убывающего* расположения элементов могут обрабатываться очень быстро, но при этом замедляется основной цикл вычислений. Вместо проверок ступенек вниз можно принудительно установить длину отрезков,

считая, что все отрезки исходного файла имеют длину 1, после первого просмотра все отрезки (кроме, возможно, последнего) имеют длину 2, ..., после k-го просмотра все отрезки (кроме, возможно, последнего) имеют длину 2^k. В отличие от "естественного" слияния в алгоритме N такой способ называется *простым* двухпутевым слиянием.

Алгоритм простого двухпутевого слияния очень напоминает алгоритм N—он описывается, по существу, той же блок-схемой; тем не менее методы достаточно отличаются друг от друга, и поэтому стоит записать весь алгоритм целиком.

Алгоритм S. *(Сортировка простым двухпутевым слиянием.)* Как и в алгоритме N, при сортировке записей $R_1, \ldots, R_N$ используются две области памяти.

S1 [Начальная установка.] Установить $s \leftarrow 0$, $p \leftarrow 1$. (Смысл переменных s, i, j, k, l, d см. в алгоритме N. Здесь p—размер возрастающих отрезков, которые будут сливаться во время текущего просмотра; q и r—количества неслитых элементов в отрезках.)

S2 [Подготовка к просмотру.] Если $s = 0$, то установить $i \leftarrow 1$, $j \leftarrow N$, $k \leftarrow N$, $l \leftarrow 2N + 1$; если $s = 1$, то установить $i \leftarrow N + 1$, $j \leftarrow 2N$, $k \leftarrow 0$, $l \leftarrow N + 1$. Затем установить $d \leftarrow 1$, $q \leftarrow p$, $r \leftarrow p$.

S3 [Сравнение $K_i : K_j$.] Если $K_i > K_j$, то перейти к шагу **S8**.

S4 [Пересылка R_i] Установить $k \leftarrow k + d$, $R_k \leftarrow R_i$.

S5 [Конец отрезка?] Установить $i \leftarrow i + 1$, $q \leftarrow q - 1$. Если $q > 0$, то возвратиться к шагу **S3**.

S6 [Пересылка R_j.] Установить $k \leftarrow k + d$. Затем, если $k = l$, перейти к шагу **S13**; в противном случае установить $R_k \leftarrow R_j$.

S7 [Конец отрезка?] Установить $j \leftarrow j - 1$, $r \leftarrow r - 1$. Если $r > 0$, возвратиться к шагу **S6**; в противном случае перейти к шагу **S12**.

S8 [Пересылка R_j.] Установить $k \leftarrow k + d$, $R_k \leftarrow R_j$

S9 [Конец отрезка?] Установить $j \leftarrow j - 1$, $r \leftarrow r - 1$. Если $r > 0$, то возвратиться к шагу **S3**.

S10 [Пересылка R_i.] Установить $k \leftarrow k + d$. Затем, если $k = l$, перейти к шагу **S13**; в противном случае установить $R_k \leftarrow R_i$.

S11 [Конец отрезка?] Установить $i \leftarrow i + 1$, $q \leftarrow q - 1$. Если $q > 0$, то возвратиться к шагу **S10**.

S12 [Переключение направления.] Установить $q \leftarrow p$, $r \leftarrow p$, $d \leftarrow -d$ и взаимозаменить $k \leftrightarrow l$. Возвратиться к шагу **S3**.

S13 [Переключение областей.] Установить $p \leftarrow p + p$. Если $p < N$, то установить $s \leftarrow 1 - s$ и возвратиться к **S2**. В противном случае сортировка завершена; если $s = 0$, то установить

$$(R_1, \ldots, R_n) \leftarrow (R_{N+1}, \ldots, R_{2N}).$$

(Независимо от распределения исходного файла последнее копирование будет выполнено тогда и только тогда, когда значение $\lceil \log_2 N \rceil$ нечетно. Так что можно заранее предсказать положение отсортированного файла, и копирование, как правило, не требуется.) ∎

Таблица 2

Сортировка простым двухпутевым слиянием

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	512	677	509	908	426	897	653	275	170	154	612	061	765	087
087	503	703	765	154	170	509	908	897	653	426	275	677	612	512	061
061	087	503	512	612	677	703	765	908	897	653	509	426	275	170	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Пример работы алгоритма см. в табл. 2. Довольно удивительно, что этот метод справедлив и тогда, когда N не является степенью 2; сливаемые отрезки не все имеют длину 2^k, тем не менее никаких явных мер предосторожности на случай таких исключений не предусмотрено! (См. упр. 8.) Проверки ступенек вниз заменены уменьшением переменных q и r и проверкой на равенство нулю. Благодаря этому время работы на машине `MIX` асимптотически приближается к $11N \log_2 N$ единицам, что несколько лучше значения, которого нам удалось добиться в алгоритме N.

На практике имеет смысл комбинировать алгоритм S с простыми вставками; вместо первых четырех просмотров алгоритма S можно простыми вставками отсортировать группы, скажем, из 16 элементов, исключив таким образом довольно расточительные вспомогательные операции, связанные со слиянием коротких файлов. Как мы уже видели в случае быстрой сортировки, такое комбинирование методов не влияет/на асимптотическое время работы, но дает тем не менее, немалую выгоду.

Рассмотрим теперь алгоритмы N и S с точки зрения структур данных. Почему нам необходима память под $2N$, а не под N записей? Причина относительно проста: мы работаем с четырьмя списками переменного размера (два "входных списка" и два "выходных списка" в каждом просмотре); при этом для

каждой пары последовательно распределенных списков мы пользуемся стандартным понятием "встречного роста", обсуждавшимся в (п. 2.2.2. Но в любой момент времени половина памяти не используется, и после некоторого размышления становится ясно, что в действительности, для наших четырех списков следовало бы воспользоваться *связанным* распределением памяти. Если к каждой из N записей добавить поле связи, то все необходимое можно проделать, пользуясь алгоритмами слияния, которые производят простые манипуляции со связями и совсем не перемещают сами записи. Добавление N полей связи, как правило, выгоднее, чем добавление пространства памяти еще под N записей, а отказавшись от перемещения записей, мы можем также сэкономить и время. Итак, нам нужно рассмотреть алгоритм, подобный следующему.

Алгоритм L. *(Сортировка посредством слияния списков.)* Предполагается, что записи $R_1, \ldots, R_N$ содержат ключи $K_1, \ldots, K_N$ и "поля связи" $L_1, \ldots, L_N$, в которых могут храниться числа от $-(N+1)$ до $(N+1)$. В начале и в конце файла имеются искусственные записи R_0 и R_{N+1} с полями связи L_0 и L_{N+1} Этот алгоритм сортировки списков устанавливает поля связи таким образом, что записи оказываются связанными в возрастающем порядке. После завершения сортировки L_0 указывает на запись с наименьшим ключом; при $1 \le k \le N$ связь L_k указывает на запись, следующую за R_k, а если R_k—запись с наибольшим ключом, то $L_k = 0$. (См. формулы (5.2.1–9).)

В процессе выполнения этого алгоритма записи R_0 и R_{N+1} служат "головами" двух линейных списков, подсписки которых в данный момент сливаются. Отрицательная связь означает конец подсписка, о котором известно, что он упорядочен; нулевая связь означает конец всего списка. Предполагается, что $N \ge 2$.

Через "$|L_s| \leftarrow p$" обозначена операция "присвоить L_s значение p или $-p$, сохранив прежний знак L_s". Такая операция легко реализуется на машине `MIX`, но, к сожалению, это не так для большинства ЭВМ. Нетрудно изменить алгоритм, чтобы получить столь же эффективный метод и для большинства других машин.

L1 [Подготовить два списка.] Установить $L_0 \leftarrow 1$, $L_{N+1} \leftarrow 2$, $L_i \leftarrow -(i+2)$ при $l \le i \le N-2$ и $L_{N-1} \leftarrow L_N \leftarrow 0$. (Мы создали два списка, содержащие соответственно записи R_1, R_3, R_5, ... и R_2, R_4, R_6, ...; отрицательные связи говорят о том, что каждый упорядоченный "подсписок" состоит всего лишь из одного элемента. Другой способ выполнить этот шаг, извлекая пользу из упорядоченности, которая могла присутствовать в исходных данных, см. в упр. 12.)

L2 [Начать новый просмотр.] Установить $s \leftarrow 0$, $t \leftarrow N+1$, $p \leftarrow L_s$, $q \leftarrow L_t$. Если $q = 0$, то работа алгоритма завершена. (При каждом просмотре p и q пробегают по спискам, которые подвергаются слиянию; s обычно указывает на последнюю обработанную запись текущего подсписка, а t—на конец только что выведенного подсписка.)

L3 [Сравнить $K_p : K_q$.] Если $K_p > K_q$, то перейти к **L6**.

L4 [Продвинуть p.] Установить $|L_s| \leftarrow p$, $s \leftarrow p$, $p \leftarrow L_p$. Если $p > 0$, то возвратиться к **L3**.

L5 [Закончить подсписок.] Установить $L_s \leftarrow q$, $s \leftarrow t$. Затем установить $t \leftarrow q$ и $q \leftarrow L_q$ один или более раз, пока не станет $q \le 0$, после чего перейти к **L8**.

L6 [Продвинуть q.] (Шаги **L6** и **L7** двойственны по отношению к **L4** и **L5**.) Установить $|L_s| \leftarrow q$, $s \leftarrow q$, $q \leftarrow L_q$. Если $q > 0$, то возвратиться к **L3**.

L7 [Закончить подсписок.] Установить $L_s \leftarrow p$, $s \leftarrow t$. Затем установить $t \leftarrow p$ и $p \leftarrow L_p$ один или более раз, пока не станет $p > 0$.

L8 [Конец просмотра?] (К этому моменту $p \le 0$ и $q \le 0$, так как оба указателя продвинулись до конца соответствующих подсписков.) Установить $p \leftarrow -p$, $q \leftarrow -q$. Если $q = 0$, то установить $|L_s| \leftarrow p$, $|L_t| \leftarrow 0$, и возвратиться к **L2**; в противном случае возвратиться к **L3**. ∎

Пример работы этого алгоритма приведен в табл. 3, в которой показаны связи к моменту выполнения шага L2. По окончании работы алгоритма можно, пользуясь методом из упр. 5.2–12, переразместить записи так, чтобы их ключи были упорядочены. Можно заметить интересную аналогию между слиянием списков и сложением разреженных многочленов (см. алгоритм 2.2.4A).

Таблица 3

Сортировка посредством слияния списков

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
K_i	–	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	–
L_j	1	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13	−14	−15	−16	0	0	2
L_j	2	−6	1	−8	3	−10	5	−11	7	−13	9	12	−16	14	0	0	15	4
L_j	4	3	1	−11	2	−13	8	5	7	0	12	10	9	14	16	0	15	6
L_j	4	3	6	7	2	0	8	5	1	14	12	10	13	9	16	0	15	11
L_j	4	12	11	13	2	0	8	5	10	14	1	6	3	9	16	7	15	0

Напишем теперь MIX-программу для алгоритма L, чтобы выяснить, столь ли выгодно оперировать списками с точки зрения времени, как и с точки зрения пространства?

Программа L. *(Сортировка посредством слияния списков.)* Для удобства предполагается, что записи занимают одно слово, причем L_j хранится в поле (0 : 2), а K_j—в поле (3 : 5) ячейки INPUT + j; значения регистров: rI1 $\equiv p$, rI2 $\equiv q$, rI3 $\equiv s$, rI4 $\equiv t$ rA $\equiv R_q$; $N \geq 2$.

```
L      EQU   0:2                  Определение имен полей.
ABSL   EQU   1:2
KEY    EQU   3:5
START  ENT1  N-2            1     L1. Подготовить два списка.
       ENNA  2,1          N-2
       STA   INPUT,1(L)    N-2    L_i ← -(i+2).
       DEC1  1             N-2
       J1P   *-3           N-2    N-2 ≥ i > 0.
       ENTA  1             1
       STA   INPUT(L)      1      L_0 ← 1.
       ENTA  2             1
       STA   INPUT+N+1(L)  1      L_{N+1} ← 2.
       STZ   INPUT+ N-1(L) 1      L_{N-1} ← 0.
       STZ   INPUT +N(L)   1      L_N ← 0.
       JMP   L2            1      К L2.
L3Q    LDA   INPUT,2       C''+B' L3. Сравнить K_p : K_q.
L3P    CMPA  INPUT,1(KEY)  C
       JL    L6            C      К L6, если K_q < K_p.
L4     ST1   INPUT,3(ABSL) C'     L4. Продвинуть p. |L_s| ← p.
       ENT3  0,1           C'     s - p.
       LD1   INPUT, 1(L)   C'     p ← L_p
       J1P   L3P           C'     К L3, если p > 0.
L5     ST2   INPUT,3(L)    B'     L5. Закончить подсписок. L_s ← q.
       ENT3  0,4           B'     s ← t.
       ENT4  0,2           D'     t ← q.
       LD2   INPUT,2(L)    D'     q ← L_q.
       J2P   *-2           D'     Повторить, если q > 0.
       JMP   L8            B'     К L8
L6     ST2   INPUT,3(ABSL) C''    L6. Продвинуть q. |L_s| ← q.
       ENT3  0,2           C''    s ← q.
       LD2   INPUT,2(L)    C''    q ← L_q.
       J2P   L3Q           C''    К L3, если q > 0.
L7     ST1   INPUT,3(L)    B''    L7. Закончить подсписок. L_s ← p.
       ENT3  0,4           B''    s ← t.
       ENT4  0,1           D''    t ← p.
       LD1   INPUT, 1(L)   D''    p ← L_p.
       J1P   *-2           D''    Повторить, если p > 0.
L8     ENN1  0,1           B      L8. Конец просмотра? p ← -p.
       ENN2  0,2           B      q ← -q.
       J2NZ  L3Q           B      К L3, если q ≠ 0.
       ST1   INPUT,3(ABSL) A      |L_s| ← p.
       STZ   INPUT,4(ABSL) A      |L_t| ← 0.
L2     ENT3  0             A+1    L2. Начать новый просмотр, s ← 0.
       ENT4  N+1           A+1    t ← N+1.
       LD1   INPUT (L)     A+1    p ← L_s.
       LD2   INPUT+N+1(L)  A+1    q ← L_t.
       J2NZ  L3Q           A+1    К L3, если q ≠ 0.
```

Время работы этой программы можно оценить при помощи методов, которыми мы уже не раз пользовались (см. упр. 13, 14); в среднем оно равно приблизительно $(10N \log_2 N + 4.92N)$ единиц с небольшим стандартным отклонением порядка $\sqrt{N}$. В упр. 15 показано, что за счет некоторого удлинения программы можно сократить время примерно до $9N \log_2 N$.

Итак, в случае внутреннего слияния связанное распределение памяти имеет бесспорные преимущества перед последовательным распределением: требуется меньше памяти, и программа работает на 10–20% быстрее. Аналогичные алгоритмы опубликованы Л. Дж. Вудрамом [*IBM Systems J.*, **8** (1969), 189–203] и А. Д. Вудаллом [*Сотр. J.*, **13** (1970), 110–111].

Упражнения

1. [20] Обобщите алгоритм М на *k-путевое слияние* исходных файлов $x_{i1} \le \ldots \le x_{im_i}$ при $i = 1, 2, \ldots, k$.
2. [М24] Считая, что все $\binom{m+n}{m}$ возможных расположений m элементов x среди n элементов y равновероятны, найдите математическое ожидание и стандартное отклонение числа выполнении шага М2 в алгоритме М. Чему равны максимальное и минимальное значения этой величины?

>3. [20] *(Изменение.)* Даны записи $R_1, \ldots, R_M$ и $R'_1, \ldots, R'_N$, ключи которых различны и упорядочены, т. е. $K_1 < \ldots < K_M$ и $K'_1 < \ldots < K'_N$. Как нужно изменить алгоритм М, чтобы в результате слияния получался файл, в котором отсутствуют записи R_i первого файла, если во втором файле тоже есть запись с таким же ключом?

4. [21] В тексте отмечено, что сортировку слиянием можно рассматривать как обобщение сортировки вставками. Покажите, что метод слияний имеет непосредственное отношение и к выбору из дерева (воспользуйтесь в качестве иллюстрации рис. 23).

>5. [21] Докажите, что в шагах N6 и N10 переменные i и j не могут быть равны. (Поэтому в этих шагах проверка на случай возможного перехода к N13 необязательна.)

6. [22] Найдите такую перестановку $K_1 K_2 \ldots K_{16}$ множества $\{1, 2, \ldots, 16\}$, что

$$K_2 > K_3, K_4 > K_5, K_6 > K_7, K_8 > K_9,$$
$$K_{10} < K_{11}, K_{12} < K_{13}, K_{14} < K_{15},$$

которая тем не менее будет отсортирована при помощи алгоритма N всего за два просмотра. (Так как в искомой перестановке имеется не менее восьми отрезков, то мы могли бы ожидать, что после первого просмотра останутся по меньшей мере четыре отрезка, после второго просмотра—два отрезка, и сортировка, как правило, не завершится раньше окончания третьего просмотра. Каким же образом можно обойтись всего двумя просмотрами?)

7. [16] Найдите точную формулу, выражающую число просмотров алгоритма S в виде функции от N.
8. [22] Предполагается, что во время работы алгоритма переменные q и rг представляют длины неслитых частей отрезков, обрабатываемых в данный момент; в начале работы -как q, так и r устанавливаются равными p, в то время как отрезки не всегда имеют такую длину. Почему же алгоритм тем не менее работает правильно?
9. [24] Напишите `MIX`-программу для алгоритма S. Выразите частоту выполнения каждой команды через величины, подобные A, B', B'', C', ... в программе L.
10. [25] (Д. А. Белл.) Покажите, что простое двухпутевое слияние файлов с последовательно расположенными элементами можно выполнить, имея всего $\frac{3}{2}N$ ячеек памяти, а не $2N$, как в алгоритме S.
11. [21] Является ли алгоритм L алгоритмом "устойчивой" сортировки?

>12. [22] Измените шаг L1 алгоритма L так, чтобы двухпутевое слияние стало "естественным", извлекая пользу из наличия возрастающих отрезков в исходном файле. (В частности, если исходные данные уже упорядочены, то шаг L2 завершит работу алгоритма сразу же после выполнения измененного вами шага L1.)

>13. [М34] Проанализируйте среднее время работы программы L подобно тому, как мы анализировали другие алгоритмы в этой главе. Дайте толкование величинам A, B, B', ... и объясните, как вычислить их точные средние значения. Сколько времени затратит программа L на сортировку 16 чисел в табл. 3?

14. [М24] Пусть двоичное представление числа N—это $2^{e_1} + 2^{e_2} + \cdots + 2^{e_t}$, где $e_1 > e_2 > \ldots > e_t \ge 0$, $t \ge 1$. Докажите, что максимальное число сравнений ключей, выполняемых алгоритмом L, равно $1 - 2^{e_t} + \sum 1 \le k \le t(e_k + k - 1)2^{e_k}$.
15. [20] Если промоделировать вручную работу алгоритма L, то обнаружится, что в нем иногда выполняются лишние операции; примерно в половине случаев не нужны присваивания $|L_s| \leftarrow p$, $|L_s| \leftarrow q$ в шагах L4 и L6, поскольку мы имеем $L_s = p$ (или q) всякий раз, когда возвращаемся из шага L4 (или L6) к L3. Как улучшить программу L, избавившись от этих лишних присваивании?
16. [28] Разработайте алгоритм слияния списков, подобный алгоритму L, но основанный на трехпутевом слиянии.
17. [20] (Дж. Мак-Карти.) Пусть двоичное представление числа N такое же, как в упр. 14, и предположим, что дано N записей, организованных в t упорядоченных подфайлов, имеющих размеры соответственно

2^{e_1}, 2^{e_2}, ..., 2^{e_t}. Покажите, как можно сохранить такое состояние при добавлении $(N+1)$-й записи и $N \leftarrow N+1$. (Полученный алгоритм можно назвать "оперативной" сортировкой слиянием.)

18. [40] (М. А. Кронрод.) Можно ли отсортировать файл из N записей, содержащий всего два отрезка:

$$K_1 \le \ldots \le K_M \quad \text{и} \quad K_{M+1} \le \ldots \le K_N,$$

За $O(N)$ операций в памяти с произвольным доступом, *используя лишь небольшое дополнительное пространство памяти фиксированного размера*, не зависящего от M и N? (Все алгоритмы слияния, описанные в этом пункте, используют дополнительное пространство памяти, пропорциональное N.)

19. [26] Рассмотрим железнодорожный разъезд с n "стеками", как показано на рис. 31 при $n = 5$; такой разъезд имеет некоторое отношение к алгоритмам сортировки с n просмотрами. В упр. с 2.2.1–2 по 2.2.1–5 мы рассмотрели разъезды с одним стеком. Вы видели, что если с правого конца поступает N вагонов, то слева может появиться сравнительно небольшое количество из N всевозможных перестановок этих вагонов.

Предположим, что в разъезд с n стеками справа поступает 2^n вагонов. Докажите, что при помощи подходящей последовательности операций слева *можно получить* любую из $2^n!$ всевозможных перестановок этих вагонов. (Каждый стек достаточно велик, и при необходимости в него можно поместить все вагоны).

20. [47] В обозначениях упр. 2.2.1–4 при помощи разъездов с n стеками можно получить не более a_N^n перестановок N элементов; следовательно, для

Picture: Рис. 31. Железнодорожный разъезд с пятью "стеками".

получения всех $N!$ перестановок требуется не менее $\log N!/\log a_N \approx \log_4 N$ стеков. В упр. 19 показано, что нужно не более $\lceil \log_2 N \rceil$ стеков. Какова истинная скорость роста необходимого числа стеков при $N \to \infty$?

21. [23] (Э. Дж. Смит.) Объясните, как можно обобщить алгоритм L, чтобы он, помимо сортировки, вычислял также число *инверсий* в исходной перестановке.

Распределяющая сортировка Мы подходим теперь к интересному классу методов сортировки, который, как показано в п. 5.4.7, по существу, прямо *противоположен* слиянию. Читателям, знакомым с перфокарточным оборудованием, хорошо известна эффективная процедура, применяемая в машинах для сортировки карт и основанная на сравнении цифр ключей; ту же идею можно приспособить и для программирования. Она общеизвестна под названиями "поразрядная сортировка", "цифровая сортировка" или "карманная сортировка".

Предположим, нам нужно отсортировать колоду из 52 игральных карт. Определим упорядочение по старшинству (достоинству) карт в масти

$$\text{Т} < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < \text{В} < \text{Д} < \text{К},$$

а также по масти

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit$$

Одна карта предшествует другой, если либо (i) она младше по масти, либо (ii) масти обеих карт одинаковы, но она младше по достоинству. (Это частный случай *лексикографического* упорядочения на множестве упорядоченных пар предметов; ср. с упр. 5–2.) Таким образом,

$$\text{Т}\clubsuit < 2\clubsuit < \cdots < \text{К}\clubsuit < \text{Т}\diamondsuit < \cdots < \text{Д}\spadesuit < \text{К}\spadesuit$$

Мы могли бы отсортировать карты одним из обсуждавшихся ранее методов; люди, как правило, пользуются способом, по сути аналогичным обменной поразрядной сортировке. Естественно отсортировать карты сначала по их масти, разложив их в четыре стопки, а затем перекладывать карты внутри каждой стопки до тех пор, пока они не будут упорядочены по достоинству.

Но существует более быстрый способ! Сначала разложить карты в 13 стопок лицевой стороной вверх по их достоинству. Затем собрать все стопки вместе: снизу тузы, затем двойки, тройки и т. д. и сверху короли (лицевой стороной вверх). Перевернуть колоду рубашками вверх и снова разложить, на этот раз в четыре стопки по масти. Сложив вместе полученные стопки так, чтобы внизу были трефы, затем бубны, черви и пики, получим упорядоченную колоду карт.

Та же идея годится и для сортировки числовых и буквенных данных. Почему же она правильна? Потому что (в нашем примере с игральными картами), если две карты при последнем раскладе попали в разные стопки, то они имеют разные масти, так что карта с меньшей мастью младше. Если же карты

одной масти, то они уже находятся в нужном порядке благодаря предварительной сортировке. Иначе говоря, при втором раскладе в каждой из четырех стопок карты будут расположены в возрастающем порядке. Это доказательство можно обобщить и показать, что таким способом можно отсортировать любое множество с лексикографическим упорядочением; подробности см. в упр. 5–2 (в начале главы).

Только что описанный метод сортировки сразу не очевиден, и не ясно, кто же первый обнаружил, что он так удобен. В брошюре на 19 страницах под названием "The Inventory Simplified", опубликованной отделением фирмы IBM Tabulating Machines Company в 1923 г., представлен интересный цифровой метод вычисления сумм произведений на сортировальной машине. Пусть, например, нужно перемножить два числа, пробитых соответственно в колонках 1–10 и в колонках 23–25, и вычислить сумму таких произведений для большого числа карт. Тогда сначала можно отсортировать карты по колонке 25 и найти при помощи счетно-аналитической машины величины a_1, a_2, $\ldots a_9$, где a_k—сумма чисел из колонок 1–10 по всем карточкам, на которых в колонке 25 пробита цифра k. Затем можно отсортировать карты по колонке 24 и найти аналогичные суммы b_1, b_2, $\ldots$, b_9, а потом по колонке 23, получив величины c_1, c_2, $\ldots$, c_9. Легко видеть, что искомая сумма произведений равна

$$a_1 + 2a_2 + \cdots + 9a_9 + 10b_1 + 20b_2 + \cdots + 90b_9 + 100c_1 + 200c_2 + \cdots + 900c_9.$$

Такой перфокарточный метод табулирования естественным образом приводит к идее поразрядной сортировки "сначала-по-младшей-цифре", так что, по-видимому, она впервые стала известна операторам этих машин. Первая опубликованная ссылка на этот метод содержится в ранней работе Л. Дж. Комри, посвященной обсуждению перфокарточного оборудования [Transactions of the Office Machinery Users' Assoc., Ltd. (1929), 25–37, особенно стр. 28].

Чтобы выполнить поразрядную сортировку с помощью ЭВМ, необходимо решить, как представлять стопки. Пусть имеется M стопок; можно было бы выделить M областей памяти и пересылать каждую исходную запись в соответствующую область. Но это решение нас не удовлетворяет, потому что в каждой области должно быть достаточно места для хранения N элементов, и тогда потребуется пространство под $(M+1)N$ записей. Такая чрезмерная потребность в памяти заставляла большинство программистов отказываться от применения поразрядной сортировки на вычислительных машинах, пока Х. Сьюворд [дипломная работа, M.I.T. Digital Computer Laboratory Report R-232 (Cambridge Mass: 1954), 25–28] не показал, что того же эффекта можно добиться, имея в распоряжении пространство всего под $2N$ записей и M счетчиков. Сделав один предварительный просмотр данных, можно просто посчитать, сколько элементов попадет в каждую область; это даст нам возможность точно распределить память под стопки. Мы уже применяли эту идею при распределяющей сортировке (алгоритм 5.2D).

Итак, поразрядную сортировку можно выполнить следующим образом: сначала произвести распределяющую сортировку *по младшим цифрам ключей* (в системе счисления с основанием M), переместив записи из области ввода во вспомогательную область, затем произвести еще одну распределяющую сортировку по следующей цифре, переместив записи обратно в исходную область и т. д., до тех пор, пока после завершающего просмотра (сортировка по старшей цифре) все ключи не окажутся расположенными в нужном порядке.

Если у нас имеется десятичная машина, а ключи—12-разрядные числа и если N весьма велико, то можно выбрать $M = 1000$ (считая три десятичные цифры за одну в системе счисления с основанием 1000); независимо от величины N сортировка будет выполнена за четыре просмотра. Аналогично, если бы имелась двоичная машина, а ключи—40-битовые двоичные числа, то можно положить $M = 1024$ и также завершить сортировку за четыре просмотра. Фактически каждый просмотр состоит из трех частей (подсчет, распределение памяти, перемещение); Фрэнд [*JACM*, **3** (1956), 151] предложил комбинировать два из этих трех действий, добавив еще M ячеек: накапливать значения счетчиков для $(k+1)$-го просмотра одновременно с перемещением во время k-го просмотра.

В табл. 1 показано применение поразрядной сортировки к нашим 16 ключам при $M = 10$. При таких малых N поразрядная сортировка, как правило, не особенно полезна, так что этот маленький пример предназначен главным образом для того, чтобы продемонстрировать достаточность метода, а не его эффективность.

Таблица 1

Поразрядная сортировка

Область ввода:	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
Счетчики для младших цифр:				1	1	2	3	1	2	1	3	1	1			
Соответствующее распределение памяти:				1	2	4	7	8	10	11	14	15	16			
Вспомогательная область:	170	061	512	612	503	653	703	154	275	765	426	087	897	677	908	509
Счетчики для средних цифр:				4	2	1	0	0	2	2	3	1	1			
Соответствующее распределение памяти:				4	6	7	7	7	9	11	14	15	16			
Область ввода;	503	703	908	509	512	612	426	653	154	061	765	170	275	677	087	897
Счетчики для старших цифр:				2	2	1	0	1	3	3	2	1	1			
Соответствующее распределение памяти:				2	4	5	5	6	9	12	14	15	16			
Вспомогательная область:	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Искушенный "современный" читатель заметит, однако, что идея счетчиков для распределения памяти привязана к "старомодным" понятиям о последовательном представлении данных; нам же известно, что специально для работы с множеством таблиц переменной длины придумано *связанное* распределение. Поэтому для поразрядной сортировки естественно будет воспользоваться связанными структурами данных. Так как каждая стопка просматривается последовательно, то все, что нам нужно,— иметь при каждом элементе одну-единственную ссылку на следующий элемент. Кроме того, никогда не придется перемещать записи: достаточно скорректировать связи—и можно смело двигаться дальше по спискам. Объем необходимой памяти равен $(1+\varepsilon)N + 2\varepsilon M$ записей, где ε—пространство, занимаемое одним полем связи. Довольно интересны формальные подробности этой процедуры, поскольку они дают прекрасный пример типичных манипуляций со структурами данных, соединяющих в себе последовательное и связанное распределение памяти.

Picture: Рис. 32. Поразрядная сортировка списка.

Алгоритм R. *(Поразрядная сортировка списка.)* Предполагается, что каждая из записей R_1, ..., R_N содержит поле связи `LINK`, а ключи представляют собой последовательность из p элементов

$$(a_p, \ldots, a_2, a_1), \quad 0 \le a_i < M,$$

и отношение порядка—лексикографическое, т. е.

$$(a_p, \ldots, a_2, a_1) < (b_p, \ldots, b_2, b_1)$$

тогда и только тогда, когда существует такой индекс j, $1 \le j \le p$, что

$$a_i = b_i \qquad \text{при } i > j, \text{ но } a_j < b_j.$$

Ключи можно представлять себе, в частности, как числа, записанные в системе счисления с основанием M:

$$a_p M^{p-1} + \cdots + a_2 M + a_1,$$

и в этом случае лексикографическое отношение порядка соответствует обычному упорядочению множества неотрицательных чисел. Ключи также могут быть цепочками букв алфавита и т. д.

Во время сортировки формируются M "стопок" подобно тому, как это делается в сортировальной машине для перфокарт. Стопки фактически представляют собой очереди в смысле гл. 2, поскольку мы связываем их вместе таким образом, что они всегда просматриваются по принципу "первым включается—первым исключается". Для каждой стопки имеются две переменные-указатели: `TOP`$[i]$ и `BOTM`$[i]$, $0 \le i < M$, и, как и в гл. 2, предполагается, что

$$\texttt{LINK}(\texttt{LOC}(\texttt{BOTM}([i]))) \equiv \texttt{BOTM}[i].$$

R1 [Цикл по k.] Вначале установить `P` $\leftarrow$ `LOC`(R_N), указатель на последнюю запись. Затем выполнить шаги с **R2** по **R6** при $k = 1, 2, \ldots, p$ (шаги с **R2** по **R6** составляют один "просмотр") и завершить работу алгоритма. Переменная `P` будет указывать на запись с наименьшим ключом, `LINK(P)`— на

запись со следующим по величине ключом, LINK(LINK(P))—на следующую и т.д.; поле LINK последней записи будет равно Λ.

R2 [Опустошить стопки.] При $0 \le i < M$ установить TOP$[i] \leftarrow$ LOC(BOTM$[i]$) и BOTM$[i] \leftarrow \Lambda$.

R3 [Выделить k-ю цифру ключа.] Пусть KEY(P)—ключ записи, на которую указывает P,—равен $(a_p, \ldots, a_2, a_1)$; установить $i \leftarrow a_k$, k-я младшая цифра этого ключа.

R4 [Скорректировать связи.] Установить LINK(TOP$[i]) \leftarrow$ P, затем TOP$[i] \leftarrow$ P.

R5 [Перейти к следующей записи.] Если $k = 1$ (первый просмотр) и если P $=$ LOC(R_j) при некотором $j \ne 1$, то установить P $\leftarrow$ LOC(R_{j-1}) и возвратиться к шагу **R3**. Если $k > 1$ (не первый просмотр), то установить P $\leftarrow$ LINK(P) и возвратиться к **R3**, если P $\ne \Lambda$.

R6 [Выполнить алгоритм H.] (Теперь мы уже распределили все элементы по стопкам.) Выполнить приведенный ниже алгоритм H, который сцепляет отдельные "стопки" в один список, подготавливая их к следующему просмотру. Затем установить P $\leftarrow$ BOTM[0], указатель на первый элемент объединенного списка. (См. упр. 3.) ∎

Алгоритм H. *(Сцепление очередей.)* Из M данных очередей со связями, удовлетворяющими соглашениям алгоритма R, данный алгоритм создает одну очередь, меняя при этом не более M связей. В результате BOTM[0] указывает на первый элемент, и стопка 0 предшествует стопке 1, ..., предшествует стопке $(M-1)$.

H1 [Начальная установка.] Установить $i \leftarrow 0$.

H2 [Указатель на вершину стопки.] Установить P $\leftarrow$ TOP$[i]$.

H3 [Следующая стопка.] Увеличить i на 1. Если $i = M$, то установить LINK(P) $\leftarrow \Lambda$ и завершить работу алгоритма.

H4 [Стопка пуста?] Если BOTM$[i] = \Lambda$, то возвратиться к **H3**.

H5 [Сцепить стопки.] Установить LINK(P) $\leftarrow$ BOTM$[i]$. Возвратиться к **H2**. ∎

На рис. 33 показано содержимое стопок после каждого из трех просмотров, выполняемых при сортировке наших 16 чисел с $M = 10$. Алгоритм R очень просто запрограммировать для машины MIX, если только найти удобный, способ изменять от просмотра к просмотру действия в шагах R3 и R5. В следующей программе этого удалось добиться, не жертвуя скоростью внутреннего цикла, путем предварительной записи двух команд в тело программы. Заметим, что TOP$[i]$ и BOTM$[i]$ можно упаковать в одно слово.

Picture: Рис. 33. Поразрядная сортировка с использованием связанного распределения памяти (показано содержимое всех десяти стопок после каждого просмотра).

Программа R. *(Поразрядная сортировка списков.)* Предполагается, что исходные ключи в ячейках от INPUT $+ 1$ до INPUT $+ N$ содержат $p = 3$ компоненты (a_3, a_2, a_1), хранящиеся соответственно в полях $(1:1)$, $(2:2)$ и $(3:3)$. (Таким образом, считается, что значение M меньше или равно размеру байта машины MIX.) В поле $(4:5)$ записи хранится связь LINK. Пусть TOP$[i] \equiv$ PILES $+ i(l:2)$ и BOTM$[i] \equiv$ PILES $+ i(4:5)$ при $0 \le i < M$. Удобно указывать в связи положение относительно ячейки INPUT, так что LOC(BOTM$[i]$) $=$ PILES $+ i -$ INPUT; чтобы избежать появления отрицательных связей, нужно расположить таблицу PILES после таблицы INPUT. Значения индексных регистров: rI1 $\equiv$ P, rI2 $\equiv i$, rI3 $\equiv 3 - k$, rI4 $\equiv$ TOP$[i]$; во время работы алгоритма H rI2 $\equiv i - M$.

```
LINK  EQU   4:5
TOP   EQU   1:2
START ENT1  N                    1      R1. Цикл по k. P ← LOC(R_N).
      ENT3  2                    1      k ← 1.
2H    ENT2  M-1                  3      R2. Опустошить стопки.
      ENTA  PILES-INPUT,2       3M      LOC(BOTM[i])
      STA   PILES,2 (TOP)       3M                      → TOP[i]
      STZ   PILES,2(LINK)       3M      BOTM[i] ← Λ.
      DEC2  1                   3M
      J2NN  *-4                 3M      M > i ≥ 0.
      LDA   R3SW,3               3
      STA   3F                   3      Изменить команды
      LDA   R5SW,3               3      для k-го просмотра.
      STA   5F                   3
3H    [LD2  INPUT, 1(3:3)]              R3. Выделить k-ю цифру ключа.
```

```
4H    LD4   PILES,2 (TOP)        3N      R4. Скорректировать связи.
      ST1   INPUT,4(LINK)        3N      LINK(TOP[i]) ← P.
      ST1   PILES,2(TOP)         3N      TOP[i] ← P.
5H    [DEC1 1]                           R5. Перейти к следующей записи.
      J1NZ  3B                   3N      К R3, если просмотр закончен.
6H    ENN2  M                    3       R6. Выполнить алгоритм H.
      JMP   7F                   3       К H2 с i ← 0.
R3SW  LD2   INPUT, 1(1:1)        N       Команда для R3 при k = 3.
      LD2   INPUT, 1(2:2)        N       Команда для R3 при k = 2.
      LD2   INPUT, 1(3:3)        N       Команда для R3 при k = 1.
R5SW  LD1   INPUT, 1(LINK)       N       Команда для R5 при k = 3.
      LD1   INPUT, 1(LINK)       N       Команда для R5 при k = 2.
      DEC1  1                    N       Команда для R5 при k = 1.
9H    LDA   PILES+M, 2(LINK)     3M - 3  H4.Стопка пуста?
      JAZ   8F                   3M - 3  К H3, если BOTM[i] = Λ
      STA   INPUT, 1(LINK)       3M - 3 - E  H5. Сцепить стопки LINK(P) ← BOTM[i].
7H    LD1   PILES+M, 2(TOP)      3M - E  H2. Указатель на вершину стопки.
8H    INC2  1                    3M      H3. Следующая стопка, i ← i + 1.
      J2NZ  9B                   3M      К H4, если i ≠ M.
      STZ   INPUT, 1(LINK)       3       LINK(P) ← Λ.
      LD1   PILES (LINK)         3       P ← BOTM[0].
      DEC3  1                    3
      J3NN  2B                   3       1 ≤ k ≤ 3
```

Время работы программы R равно $32N + 48M + 38 - 4E$, где N—число исходных записей, M—основание системы счисления (число стопок), а E—число встретившихся пустых стопок. Сравнение с другими программами, построенными на основе аналогичных предположений (программы 5.2.1M, 5.2.4L), говорит явно в пользу программы R. Время работы p-проходного варианта программы R равно $(11p - 1)N + O(pM)$ единиц; критический фактор, влияющий на время работы,—внутренний цикл, который содержит пять обращений к памяти и один переход. Для типичной вычислительной машины $M = b^r$ и $p = \lceil t/r \rceil$, где t— число цифр в ключах, представленных в системе счисления с основанием b; с ростом r убывает p, так что можно воспользоваться нашими формулами для определения "наилучшего" значения r.

Единственная переменная величина в формуле времени работы—это E—число пустых стопок, обнаруженных в шаге H4. Предположим, что все M^N последовательностей цифр M-ичной системы счисления равновероятны. Из изучения "покер-теста" в п. 3.3.2D мы умеем вычислять вероятность того, что в каждом просмотре встретится ровно $M - r$ пустых стопок; она равна

$$\frac{M(M-1)\ldots(M-r+1)}{M^N} \left\{ {N \atop r} \right\},$$

где $\left\{ {N \atop r} \right\}$—число Стирлинга второго рода. Согласно упр. 5,

$$E = \big(\min \max(M-N,0)p,\ \text{ave}\, M(1-\frac{1}{M})^N p,\ \max(M-1)p\big).$$

В последние годы появляется все больше "трубопроводных", или "магистральных", вычислительных машин. Эти машины имеют несколько арифметических устройств и схему "опережения", так что обращения к памяти и вычисления могут в значительной степени совмещаться во времени; но эффективность таких машин заметно понижается при наличии условных переходов, если только эти переходы не происходят почти всегда в одном и том же направлении. Внутренний цикл поразрядной сортировки хорошо приспособлен для таких машин, поскольку это простое итеративное вычисление, типичное "пережевывание чисел". Поэтому *для магистральных машин поразрядная сортировка обычно бывает наиболее эффективным методом из всех известных методов внутренней сортировки*, при условии что N не слишком мало и ключи не слишком длинные.

Разумеется, если ключи уж очень длинные, поразрядная сортировка не так эффективна. Представьте себе, например, что нужно отсортировать перфокарты по ключу из 80 колонок; как правило, встретится очень мало пар карт, у которых бы совпали первые пять колонок, так что первые 75 просмотров выполнятся почти впустую. При анализе обменной поразрядной сортировки мы обнаружили, что вовсе не обязательно проверять много битов ключей, если просматривать их не справа налево, а слева направо. Поэтому давайте возвратимся к идее поразрядной сортировки, в которой ключи просматриваются, начиная со старших цифр (СЦ), а не с младших цифр (МЦ).

Мы уже отмечали, что СЦ-поразрядная сортировка естественным образом приходит на ум. В самом деле, нетрудно понять, почему при сортировке почты в отделениях связи пользуются именно этим методом. Большое количество писем можно отсортировать по отдельным мешкам, соответствующим географическим областям; теперь каждый мешок содержит уже меньшее количество писем, которые можно независимо сортировать по другим мешкам, соответствующим все меньшим и меньшим географическим районам. (Разумеется, прежде чем подвергать письма дальнейшей сортировке, их можно переправить поближе к месту назначения.) Этот принцип "разделяй и властвуй" весьма привлекателен, и единственная причина его непригодности для сортировки перфокарт в том, что большое количество стопок приводит к путанице. Этим же явлением объясняется относительная эффективность алгоритма R (хотя здесь сначала рассматриваются МЦ), потому что нам никогда не приходится работать более чем с M стопками и стопки приходится сцеплять всего p раз. С другой стороны, нетрудно построить СЦ-поразрядный метод с использованием связанного распределения памяти с отрицательными связями для обозначение границ между стопками, как в алгоритме 5.2.4L. (См. упр. 10.)

Пожалуй, наилучший компромиссный выход указал М. Д. Макларен [*JACM*, **13** (1966), 404–411], который предложил использовать МЦ-сортировку, как в алгоритме R, *но лишь в применении к старшим цифрам*. Это не будет полной сортировкой файла, но в результате файл становится почти упорядоченным, т. е. в нем остается очень мало инверсий, так что для завершения сортировки можно воспользоваться методом простых вставок. Наш анализ алгоритма 5.2.1M применим и к-этой ситуации; если ключи распределены равномерно, то после сортировки файла по старшим p цифрам в нем останется в среднем

$$\frac{1}{4}N(N-1)M^{-p}$$

инверсий. [См. формулу (5.2.1–14) и упр. 5.2.1–38.] Макларен вычислил среднее число обращений к памяти, приходящееся на один обрабатываемый элемент, и оказалось, что оптимальный выбор значений M и p (в предположении, что M—степень двойки, ключи равномерно распределены и $N/M^p \le 0.1$, так.что отклонения от равномерного распределения приемлемы) описывается следующей таблицей:

$N =$	100	1000	5000	10000	50000	100000
Наилучшее $M =$	32	128	256	512	1024	1024
Наилучшее $p =$	2	2	2	2	2	2
$\bar{\beta}(N) =$	19.3	18.5	18.2	18.2	18.1	18.0

Здесь $\bar{\beta}(N)$—среднее число обращений к памяти на один сортируемый элемент; эта величина ограничена при $N \to \infty$, если взять $p = 2$ и $M > \sqrt{N}$, так что среднее время сортировки есть $O(N)$, а не $O(N \log N)$. Этот метод является усовершенствованием метода вставок в несколько списков (алгоритм 5.2.1M), который, по существу, представляет собой случай $p = 1$. В упр. 12 приводится интересная процедура Макларена для окончательного переразмещения после частичной сортировки файла с использованием списков.

Если воспользоваться методами алгоритма 5.2D и упр. 5.2-13, то можно обойтись без полей связи; при этом в дополнение к памяти, занятой самими записями, потребуется всего $O(\sqrt{N})$ ячеек. Если исходные данные распределены равномерно, то среднее время сортировки пропорционально N.

Упражнения

>1. [20] Алгоритм из упр. 5.2–13 показывает, как можно выполнить распределяющую сортировку, имея пространство памяти всего под N записей (и M полей счетчиков), а не под $2N$ записей. Приводит ли эта идея к усовершенствованию алгоритма поразрядной сортировки, проиллюстрированного в табл. 1?
2. [13] Является ли алгоритм R алгоритмом "устойчивой" сортировки?
3. [15] Объясните, почему в алгоритме H переменной `BOTM`[0] присваивается значение указателя на первую запись в "объединенной" очереди, *несмотря на то что стопка 0 могла быть пустой.*
>4. [23] Во время работы алгоритма R все M стопок хранятся в виде связанных очередей (первым включается—первым исключается). Исследуйте идею связывания элементов стопок как в *стеке*. (На рис. 33 стрелки пойдут не вверх, а вниз, и таблица `BOTM` станет не нужна.) Покажите, что если сцеплять стопки в соответствующем порядке, то может получиться правильный метод сортировки. Будет ли этот алгоритм более простым или более быстрым?
5. [M24] Пусть $g_{MN}(z) = \sum p_{MNk} z^k$, где p_{MNk}— вероятность того, что после случайного просмотра поразрядной сортировки, разложившего N элементов на M стопок, получилось ровно k пустых стопок. (a) Покажите, что $g_{M,N+1}(z) = g_{MN}(z) + ((1-z)/M) g'_{MN}(z)$. (b) Найдите при помощи указанного соотношения простые выражения для математического ожидания и дисперсии этого распределения вероятностей как функций от M и N.

6. [20] Какие изменения необходимо внести в программу R, чтобы она сортировала не трехбайтовые ключи, а восьмибайтовые? Считается, что старшие байты ключа K_i хранятся в ячейке $\texttt{KEY}+i(1:5)$, а три младших байта, как и раньше,—в ячейке $\texttt{INPUT}+i(1:3)$. Каково время работы программы с этими изменениями?
7. [20] Обсудите, в чем состоит сходство и отличие алгоритма R и алгоритма обменной поразрядной сортировки (алгоритм 5.2.2R).
>8. [20] В алгоритмах поразрядной сортировки, обсуждавшихся в тексте, предполагалось, что все сортируемые ключи неотрицательны. Какие изменения следует внести в эти алгоритмы в том случае, когда ключами могут быть и отрицательные числа, представленные в *дополнительном или обратном* коде?
9. [20] (Продолжение упр. 8.) Какие изменения нужно внести в эти алгоритмы в случае, когда ключами являются числа, представленные в виде абсолютной величины со знаком?
10. [30] Сконструируйте алгоритм поразрядной сортировки "сначала-по-старшей-цифре", использующий связаннее распределение. (Так как размер подфайлов все уменьшается, то разумно уменьшить M, а для сортировки коротких файлов применить не поразрядную сортировку.)
11. [16] Перестановка шестнадцати исходных чисел, показанная в табл. 1, содержит вначале 41 инверсию. После завершения сортировки инверсий, разумеется, нет совсем. Сколько инверсий осталось бы в файле, если бы мы пропустили первый просмотр, а выполнили бы поразрядную сортировку лишь по цифрам десятков и сотен? Сколько инверсий останется, если пропустить как первый, так и второй просмотры?
12. [24] (М. Д. Макларен.) Предположим, алгоритм R применили только к p старшим цифрам реальных ключей; тогда файл, если читать его по порядку, указанному связями, почти отсортирован, но ключи, у которых старшие p цифр совпадают, могут быть неупорядочены. Придумайте алгоритм переразмещения записей на том же месте так, чтобы ключи расположились по порядку: $K_1 \le K_2 \le \ldots \le K_N$. [*Указание:* частный случай, когда файл полностью отсортирован, можно найти в ответе к упр. 5.2–12, его можно скомбинировать с простыми вставками без потери эффективности, так как в файле осталось мало инверсий.]
13. [40] Реализуйте метод внутренней сортировки, предложенный в тексте в конце этого пункта, получив программу сортировки случайных данных за $O(N)$ единиц времени, требующую всего $O(N)$ дополнительных ячеек памяти.
14. [22] Последовательность игральных карт можно отсортировать в возрастающем порядке: `Т 2 ... В Д К` от верхней карты к нижней, за два просмотра, раскладывая карты каждый раз лишь в две стопки: разложите карты лицевой стороной вниз в две стопки, содержащие соответственно `Т 2 9 3 10` и `4 В 5 6 Д К 7 8` (от нижней карты к верхней); затем положите вторую стопку поверх первой, поверните колоду лицевой стороной вверх и разложите в две стопки `Т 2 3 4 5 6 7 8` и `9 10 В Д К`. Соедините эти две стопки и поверните их лицевой стороной вверх. Колода отсортирована.

Докажите, что приведенную выше последовательность карт нельзя отсортировать в *убывающем* порядке: `К Д В ... 2 Т`, от верхней карты к нижней, за два просмотра, даже если разрешено раскладывать карты в три стопки. (Сдавать карты, нужно всегда сверху колоды, поворачивая их при раздаче рубашкой вверх. На рисунке верхняя карта колоды изображена справа, а нижняя—слева.)

15. [M25] Рассмотрите задачу из упр. 14 в случае, когда карты раздаются лицевой стороной вверх, а не вниз. Таким образом, один просмотр можно потратить на преобразование возрастающего порядка в убывающий. Сколько нужно просмотров?

> Как только появится аналитическая машина, она, безусловно, определит дальнейший путь развития науки. Всякий раз, когда с ее помощью будет найден какой-либо результат, тут же возникнет вопрос: нельзя ли тот же результат получить на этой машине за кратчайшее время?
>
> Чарльз Бэббидж (1864)

ОПТИМАЛЬНАЯ СОРТИРОВКА

Теперь, когда мы проанализировали такое множество методов внутренней сортировки, пришло время обратиться к более общему вопросу: *какой метод внутренней сортировки наилучший?* Существует ли такой верхний предел скорости сортировки, которого бы не мог достичь ни один программист, как бы искусен он ни был?

Разумеется, наилучшего возможного способа сортировки *нет*; мы должны точно определить, что понимать под словом "наилучший", но не существует наилучшего возможного способа определить слово

"наилучший". Аналогичные вопросы об оптимальности алгоритмов мы обсуждали в п. 4.3.3, 4.6.3 и 4.6.4, где рассматривалось умножение с высокой точностью и вычисление полиномов. В каждом случае, для того чтобы выполнялись условия "достаточности", т. е. чтобы задача стала разрешимой, необходимо было сформулировать довольно простое определение алгоритма, "наилучшего из возможных". И в каждом случае перед нами вставали интереснейшие задачи, настолько сложные, что они до сих пор полностью не решены. Так же обстоит дело и с сортировкой: были получены некоторые интересные результаты, но осталось еще много интригующих вопросов, на которые до сих пор нет ответов.

Изучение внутреннего механизма методов сортировки обычно было направлено на минимизацию числа сравнений ключей при сортировке n элементов, или слиянии m элементов с n элементами, или выборе t-го наибольшего элемента из неупорядоченного набора n элементов. В п. 5.3.1, 5.3.2 и 5.3.3 эти вопросы обсуждаются в общем случае; в п. 5.3.4 рассматриваются аналогичные вопросы с интересным ограничением: последовательность сравнений должна быть, по существу, заранее фиксирована. Некоторые другие типы интересных теоретических вопросов, связанных с оптимальной сортировкой, можно найти в упражнениях к п. 5.3.4 и в обсуждении внешней сортировки в п. 5.4.4.

Сортировка с минимальным числом сравнений

Очевидно, минимальное число сравнений ключей, необходимое для сортировки n элементов, равно *нулю*, поскольку, как мы видели, существуют методы поразрядной сортировки, в которых вообще не выполняется сравнений. В самом деле, можно написать `MIX`-программы, способные сортировать и не содержащие тем не менее ни одной команды условного перехода! (См. упр. 5-6 в начале этой главы.) Мы также встречались с несколькими методами сортировки, которые, по существу, были основаны на сравнении ключей, но время работы которых на деле определялось другими факторами, такими, как перемещение данных, вспомогательные операции и т. д.

Поэтому ясно, что подсчет числа сравнений—не единственный способ измерить эффективность метода сортировки. Однако в любом случае небезынтересно провести тщательное исследование числа сравнений, поскольку теоретическое изучение этого вопроса позволит нам с пользой для дела проникнуть во внутреннюю природу процессов сортировки, а также поможет отточить мастерство для решения более практических задач, которые могут встать перед нами в будущем.

Чтобы исключить поразрядную сортировку, где совсем не выполняется сравнений, ограничимся обсуждением методов сортировки, основанных только на абстрактном линейном отношении порядка "$<$" между ключами, рассмотренном в начале этой главы. Для простоты мы также ограничим свое обсуждение случаем *различных* ключей, а это значит, что при любом сравнении ключей K_i и K_j возможны лишь два исхода: либо $K_i < K_j$, либо $K_i > K_j$. (Распространение этой теории на общий случай, когда допускаются равные ключи, см. в упр. от 3 до 12.)

Задачу сортировки посредством сравнений можно сформулировать также другими эквивалентными способами. Если есть n грузов и весы с двумя чашами, то каково минимальное число взвешиваний, необходимое для того, чтобы расположить грузы по порядку в соответствии с весом, если в каждой чаше весов помещается только один груз? Или же, если в некотором турнире участвуют n игроков, то каково наименьшее число игр, достаточное для того, чтобы распределить места между соревнующимися в предположении, что силы игроков можно линейно упорядочить (ничейные результаты не допускаются).

Методы сортировки n элементов, удовлетворяющие указанным ограничениям, можно представить посредством структуры расширенного бинарного дерева, такого, как показано на рис. 34. Каждый *внутренний узел* (изображенный в виде кружочка) содержит два индекса "$i : j$" и означает сравнение ключей K_i и K_j. Левое поддерево этого узла соответствует последующим сравнениям, которые необходимо выполнить, если $K_i < K_j$, а правое поддерево—тем действиям, которые необходимо предпринять в случае $K_i > K_j$. Каждый *внешний узел* дерева (изображенный в виде прямоугольника) содержит перестановку $a_1\ a_2\ \ldots a_n$

Picture: Рис. 34. Дерево сравнений для сортировки трех элементов.

множества $\{1, 2, \ldots, n\}$, обозначающую тот факт, что было установлено упорядочение

$$K_{a_1} < K_{a_2} < \ldots < K_{a_n}.$$

(Если взглянуть на путь от корня к этому внешнему узлу, то каждое из $n-1$ соотношений $K_{a_i} < K_{a_{i+1}}$, где $1 \le i < n$, будет результатом некоторого сравнения $a_i : a_{i+1}$ или $a_{i+1} : a_i$ на этом пути.)

Так, на рис. 34 представлен метод сортировки, согласно которому нужно сначала сравнить K_1 с K_2; если $K_1 > K_2$, то продолжать (двигаясь по правому поддереву) сравнивать K_2 с K_3, а затем, если $K_2 < K_3$, сравнить K_1 с K_3; наконец, если $K_1 > K_3$, становится ясно, что $K_2 < K_3 < K_1$. Реальный алгоритм сортировки обычно будет также перемещать ключи по файлу, но нас интересуют только сравнения, поэтому

мы игнорируем все перемещения данных. При сравнении K_i с K_j в этом дереве всегда имеются в виду *исходные* ключи K_i и K_j, а не те ключи, которые могли занять i-ю и j-ю позиции в файле в результате перемещения записей.

Возможны и избыточные сравнения; например, на рис. 35 нет необходимости сравнивать $3:1$, поскольку из неравенств $K_1 < K_2$ и $K_2 < K_3$ следует $K_1 < K_3$. Никакая перестановка не может соответствовать левому поддереву узла $< 3:1 >$ на рис. 35, так что эта часть алгоритма никогда не будет выполняться! Поскольку нас интересует минимальное число сравнений, то можно считать, что избыточных сравнений не производится; с этого момента мы будем иметь дело со структурой расширенного бинарного дерева, в котором каждому внешнему узлу соответствует некоторая перестановка. Все перестановки исходных ключей возможны, и каждая перестановка определяет единственный путь от корня к внешнему узлу; отсюда вытекает, что *в дереве сравнений для сортировки n элементов без избыточных сравнений имеется ровно $n!$ внешних узлов.*

Picture: Рис. 35. Пример избыточного сравнения.

Оптимизация в наихудшем случае. Первая естественным образом возникающая задача—найти деревья сравнений, минимизирующие *максимальное* число выполняемых сравнений. (Позже мы рассмотрим *среднее* число сравнений.)

Пусть $S(n)$—минимальное число сравнений, достаточное для сортировки n элементов. Если все внутренние узлы в дереве сравнений располагаются на уровнях $< k$, то очевидно, что в дереве не может быть более 2^k узлов. Следовательно, полагая $k = S(n)$, имеем

$$n! \le 2^{S(n)}.$$

Поскольку $S(n)$—целое число, то можно записать эту формулу иначе, получив нижнюю оценку:

$$S(n) \ge \lceil \log_2 n! \rceil. \tag{1}$$

Заметим, что по формуле Стирлинга

$$\lceil \log_2 n! \rceil = n \log_2 n - n/(\ln 2) + \frac{1}{2} \log_2 n + O(1); \tag{2}$$

следовательно, необходимо выполнить около $n \log_2 n$ сравнений.

Соотношение (1) часто называют "теоретико-информационной нижней оценкой", поскольку специалист в области теории информации сказал бы, что в процессе сортировки проверяется $(\log_2 n!)$ "битов информации"; каждое сравнение дает не более одного "бита информации". Такие деревья, как на рис. 34, называют также "вопросниками" ("questionnaires"), а их математические свойства исследованы в книге Клода Пикара Théorie des questionnaires (Paris: Gauthier-Villars, 1965). Из всех рассмотренных нами методов сортировки три метода требуют меньше всего сравнений: бинарные вставки (ср. с п. 5.2.1), выбор из дерева (ср. с п. 5.2.3) и простое двухпутевое слияние, как оно описано в алгоритме 5.2.4L. Нетрудно видеть, что максимальное число сравнений для метода бинарных вставок равно

$$B(n) = \sum_{1 \le k \le n} \lceil \log_2 k \rceil = n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} + 1 \tag{3}$$

(ср. с упр. 1.2.4-42), а максимальное число сравнений для алгоритма 5.2.4L приведено в упр. 5.2.4-14. Оказывается (см. п. 5.3.3), что для выбора из дерева верхняя оценка числа сравнений либо такая же, как для бинарных вставок, либо такая же, как для двухпутевого слияния, в зависимости от того, как строится дерево. Во всех трех случаях имеем асимптотическое значение $n \log_2 n$; объединяя верхнюю и нижнюю оценки для $S(n)$, докажем, что

$$\lim_{n \to \infty} \frac{S(n)}{n \log_2 n} = 1. \tag{4}$$

Таким образом, мы получили приближенную формулу для $S(n)$, однако желательно иметь более точную информацию. В следующей таблице приведены значения указанных выше величин при малых n:

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \log_2 n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$B(n) =$	0	1	3	6	8	11	14	17	21	25	29	33	37	41	45	49	54
$L(n) =$	0	1	3	5	9	11	14	17	25	27	30	33	38	41	45	49	65

Здесь $B(n)$ и $L(n)$ относятся соответственно к бинарным вставкам и слиянию списков. Можно показать, что $B(n) \le L(n)$ при любом n (см. упр. 2).

Как видно из приведенной выше таблицы, $S(4) = 5$, но $S(5)$ может равняться либо 7, либо 8. В результате снова приходим к задаче, поставленной в начале § 5.2. Каков наилучший способ сортировки пяти элементов? Возможна ли сортировка пяти элементов при помощи всего семи сравнений?

Такая сортировка возможна, но сам способ найти не так просто. Начинаем так же, как при сортировке четырех элементов посредством слияний, сравнивая сперва $K_1 : K_2$, затем $K_3 : K_4$, а затем наибольшие элементы обеих пар. Эти сравнения порождают конфигурацию, которую можно изобразить диаграммой

Picture: Рис. стр. 222

показывающей, что $a < b < d$ и $c < d$. (Для представления известных отношений порядка между элементами удобно воспользоваться направленными графами, такими, как этот, где неравенство $x < y$ считается известным тогда и только тогда, когда на графе есть путь от x к y.) Теперь вставляем пятый элемент $K_5 = e$ в соответствующее место среди $\{ a, b, d \}$; для этого требуются всего два сравнения, поскольку можно сравнить его сначала с b, а затем с a или d. Таким образом, остается одна из четырех возможностей:

Picture: p.223

и в каждом случае достаточно еще двух сравнений, чтобы вставить c в цепочку остальных элементов, меньших d. Такой способ сортировки пяти элементов впервые обнаружил Г. Б. Демут [Ph. D. thesis, Stanford University (Oct., 1956). 41–43].

Сортировка вставками и слиянием. Изящное обобщение изложенного выше метода принадлежит Лестеру Форду мл. и Селмеру М. Джонсону. Поскольку оно объединяет некоторые особенности двух способов сортировки: посредством слияний и посредством вставок, то мы назовем этот метод *сортировкой вставками и слиянием.* Рассмотрим, например, задачу сортировки 21 элемента. Начать можно со сравнений десяти пар ключей $K_1 : K_2$, $K_3 : K_4$, ..., $K_{19} : K_{20}$, затем следует отсортировать вставками и слиянием большие элементы пар. В результате получим конфигурацию

Picture: 223.2

аналогичную (5). Следующий шаг состоит в том, чтобы вставить элемент b_3 в последовательность $\{ b_1, a_1, a_2 \}$, а затем b_2—в последовательность остальных элементов, меньших a_2, приходим к конфигурации

Picture: 223.3

Назовем верхние элементы *главной цепочкой.* Элемент b_5 можно вставить в главную цепочку за три сравнения (сравнив его сначала с c_4, затем с c_2 или c_6 и т. д.); затем еще за три сравнения можно переместить в главную цепочку b_4, что приводит к конфигурации

Picture: 224.1

Следующий шаг решающий; ясно ли вам, что делать дальше? При помощи всего четырех сравнений вставляем b_{11} (а *не* b_7) в главную цепочку. После этого элементы b_{10}, b_9, b_8, b_7, b_6 (именно в таком порядке) можно вставить в нужное место в главной цепочке не более чем за четыре сравнения каждый.

Аккуратный подсчет числа требуемых сравнений показывает, что 21 элемент можно отсортировать не более чем за $10 + 22 + 2 + 2 + 3 + 3 + 4 + 4 + 4 + 4 + 4 + 4 = 66$ шагов. Поскольку

$$2^{65} < 21! < 2^{66},$$

ясно также, что и в любом другом случае необходимо не менее 66 сравнений; следовательно,

$$S(21) = 66. \tag{10}$$

(При сортировке бинарными вставками понадобилось бы 74 сравнения.)

В общем случае сортировка вставками и слиянием для n элементов выглядит следующим образом:

i) Произвести сравнения $\lfloor n/2 \rfloor$ непересекающихся пар элементов. (Если n нечетно, то один элемент не участвует в сравнениях.)

ii) Отсортировать $\lfloor n/2 \rfloor$ бóльших элементов пар, найденных в шаге (i), вставками и слиянием.

iii) Для элементов введем обозначения a_1, $a-2$, ..., $a_{\lfloor n/2\rfloor}$, b_1, b_2, ..., $b_{\lceil n/2\rceil}$, как в (7), где $a_1 \le a_2 \le \ldots \le a_{\lfloor n/2\rfloor}$ и $b_i \le a_i$ при $1 \le i \le \lfloor n/2\rfloor$; назовем b_1 и все элементы a "главной цепочкой". Не трогая элементов b_j при $j > \lceil n/2\rceil$, вставить бинарными вставками в главную цепочку остальные элементы b в следующем порядке:

$$b_3, b_2; b_5, b_4; b_{11}, b_{10}, \ \ldots, b_6; b_{t_k}, b_{t_k-1}, \ \ldots, b_{t_{k-1}+1}; \ \ldots . \tag{11}$$

Нам хотелось бы определить последовательность $(t_1, t_2, t_3, t_4, \ \ldots) = (1, 3, 5, 11, \ \ldots)$, участвующую в (11), таким образом, чтобы каждый из элементов b_{t_k}, b_{t_k-1}, ..., $b_{t_{k-1}+1}$ можно было вставить в главную цепочку не более, чем за k сравнений. Обобщая (7), (8) и (9), получим диаграмму

Picture: 224.2

где главная цепочка до a_{t_k-1} включительно содержит $2t_{k-1} + (t_k - t_{k-1} - 1)$ элементов. Это число должно быть меньше 2^k; для нас лучше всего положить его равным $2^k - 1$, и тогда

$$t_{k-1} + t_k = 2^k. \tag{12}$$

Поскольку $t_1 = 1$, то для удобства можно положить $t_0 = 1$; тогда, суммируя геометрическую прогрессию, найдем

$$\begin{aligned} t_k = 2^k - t_{k-1} = 2^k - 2^{k-1} + t_{k-2} = \ldots \\ \ldots = 2^k - 2^{k-1} + \cdots + (-1)^k 2^0 = (2^{k+1} + (-1)^k)/3. \end{aligned} \tag{13}$$

(Любопытно, что точно такая же последовательность возникла при изучении алгоритма вычисления наибольшего общего делителя двух целых чисел; ср. с упр. 4.5.2-27.)

Пусть $F(n)$—число сравнений, необходимых для сортировки n элементов вставками и слиянием. Ясно, что

$$F(n) = \lfloor n/2\rfloor + F(\lfloor n/2\rfloor) + G(\lceil n/2\rceil), \tag{14}$$

где функция G описывает количество работы, выполняемой в шаге (iii). Если $t_{k-1} \le m \le t_k$, то, суммируя по частям, получаем

$$\begin{aligned} G(m) &= \sum_{1\le j<k} j(t_j - t_{j-1}) + k(m - t_{k-1}) = \\ &= km - (t_0 + t_1 + \cdots + t_{k-1}). \end{aligned} \tag{15}$$

Положим

$$w_k = t_0 + t_1 + \cdots + t_{k-1} = \lfloor 2^{k+1}/3\rfloor, \tag{16}$$

и тогда $(w_0, w_1, w_2, w_3, w_4, \ \ldots) = (0, 1, 2, 5, 10, 21, \ \ldots)$. В упр. 13 показано, что

$$F(n) - F(n-1) = k \qquad \text{тогда и только тогда, когда } w - k < n \le w_{k+1}, \tag{17}$$

а последнее условие эквивалентно неравенствам

$$\begin{gathered} \frac{2^{k+1}}{3} < n \le \frac{2^{k+2}}{3}, \\ k + 1 < \log_2(3n) \le k + 2; \end{gathered}$$

следовательно,

$$F(n) - F(n-1) = \left\lceil \log_2 \left(\frac{3}{4}n\right)\right\rceil. \tag{18}$$

(Этой формулой мы обязаны А. Адьяну [Ph. D. thesis, Univ. of Minnesota (1969), 38–42].) Отсюда вытекает, что функция F выражается удивительно простой формулой:

$$F(n) = \sum_{1\le k\le n} \left\lceil \log_2 \left(\frac{3}{4}k\right)\right\rceil, \tag{19}$$

которая очень похожа на соответствующую формулу (3) для бинарных вставок. В "замкнутом виде" эту сумму можно найти в упр. 14.

Воспользовавшись (19), нетрудно построить таблицу значений функции $F(n)$; имеем

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil \log_2 n! \rceil =$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$F(n) =$	0	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50

$n =$	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
$\lceil \log_2 n! \rceil =$	53	57	62	66	70	75	80	84	89	94	98	103	108	113	118	123
$F(n) =$	54	58	62	66	71	76	81	86	91	96	101	106	111	116	121	126

Заметим, что при $1 \le n \le 11$ и при $20 \le n \le 21$ $F(n) = \lceil \log_2 n! \rceil$; таким образом, при этих значениях сортировка вставками и слиянием оптимальна:

$$S(n) = \lceil \log_2 n! \rceil = F(n) \qquad \text{при } 1 \le n \le 11 \text{ и } 20 \le n \le 21. \tag{20}$$

Задачу нахождения функции $S(n)$ поставил Гуго Штейнгауз во втором издании своей классической книги Mathematical Snapshots (Oxford University Press, 1950), 38–39[0]). Он описал бинарные вставки, которые являются наилучшим способом сортировки n элементов при условии, что n-й элемент не рассматривается до тех пор, пока не отсортированы первые $n-1$ элементов; и он сделал предположение о том, что метод бинарных вставок оптимален и в общем случае. Несколько лет спустя [*Calcutta Math. Soc. Golden Jubilee Commemoration,* 2 (1959), 323–327] он сообщил, что двое его коллег, С. Трибула и С. Пин, "недавно" опровергли его предположение и нашли значения $S(n)$ при $n \le 11$. Вероятно, Трибула и Пин независимо пришли к сортировке вставками и слиянием, по которой вскоре появилась публикация Л. Р. Форда и С. М. Джонсона [*AMM,* **66** (1950), 387–389].

После изобретения сортировки вставками и слиянием первым неизвестным значением функции $S(n)$ стало $S(12)$. Из табл. 1 видно, что число 12! довольно близко к 2^29, поэтому существование 29-шаговой процедуры сортировки 12 элементов весьма маловероятно. Для решения этого вопроса Марком Уэлсом был предпринят исчерпывающий поиск (занявший на вычислительной машине Маниак II около 60 ч.), который показал, что $S(12) = 30$ [Proc. IFIP Congress 65, 2 (1965), 497–498]. Итак, процедура вставок и слияний оказывается оптимальной и при $n = 12$.

Таблица 1

Значения факториалов в двоичной системе счисления

$$1 = 1!$$
$$10 = 2!$$
$$110 = 3!$$
$$11000 = 4!$$
$$1111000 = 5!$$
$$1011010000 = 6!$$
$$1001110110000 = 7!$$
$$1001110110000000 = 8!$$
$$1011000100110000000 = 9!$$
$$110111010111110000000 0 = 10!$$
$$100110000100010101000000000 = 11!$$
$$111001000110011111110000000000 = 12!$$
$$101110011001010001100110000000000 = 13!$$
$$101000100110111111011001010000000000 = 14!$$
$$100110000011101110111011101101100000000000 = 15!$$
$$1001100000111011101110111010110000000000000000 = 16!$$

***Более подробный анализ.** Чтобы более тщательно исследовать функцию $S(n)$, внимательно изучим диаграммы частичного упорядочения, такие, как (5). Полученные после нескольких сравнений сведения можно представить в виде направленного графа. Этот направленный граф в силу транзитивности отношения $<$ не содержит циклов; следовательно, его всегда можно изобразить таким образом, чтобы все дуги шли слева направо; поэтому удобнее удалить из диаграммы стрелки. В результате диаграмма (5) преобразуется в

Picture: p.227

[0] Есть перевод первого издания этой книги: Математический калейдоскоп, М., Гостехиздат, 1949.— *Прим. перев.*

Пусть G—такой направленный граф; обозначим через $T(G)$ число перестановок, согласующихся с G, т. е. число способов пометить вершины графа G целыми числами $\{1, 2, \ldots, n\}$ так, чтобы число в вершине x было меньше числа в вершине y, если дуга $x \to y$ принадлежит G. Пример перестановки, согласующейся с (21): $a = 1$, $b = 4$, $c = 2$, $d = 5$, $e = 3$. Мы изучили функцию $T(G)$ для различных G в п. 5.1.4, где было замечено, что $T(G)$ есть число способов "топологически отсортировать" граф G.

Пусть G—граф из n элементов, который можно получить после k сравнений; определим *эффективность* графа G функцией

$$E(G) = \frac{n!}{2^k T(G)}. \tag{22}$$

(Эта идея принадлежит Франку Хуану и Шень Линю.) Строго говоря, эффективность не есть функция лишь самого графа G—она зависит от того пути, которым мы пришли к G в процессе сортировки, однако нам удобно допустить эту маленькую неточность. Выполнив еще одно сравнение над элементами i и j, получим два графа G_1, и G_2: один—для случая $K_i < K_j$ и другой—для случая $K_i > K_j$. Ясно, что

$$T(G) = T(G_1) + T(G_2).$$

Если $T(G_1) \geq T(G_2)$, то имеем

$$T(G) \leq 2T(G_1),$$
$$E(G_1) = \frac{n!}{2^{k+1}T(G_1)} = \frac{E(G)T(G)}{2T(G_1)} \leq E(G). \tag{23}$$

Следовательно, каждое сравнение приводит к графу меньшей или равной эффективности; нельзя увеличить эффективность за счет дополнительных сравнений.

Заметим, что если G совсем не содержит дуг, то $k = 0$ и $T(G) = n!$, т. е. начальная эффективность равна 1. Если же граф G представляет окончательный результат сортировки, то G выглядит как отрезок прямой, и $T(G) = 1$. Так, например, если нам нужно построить процедуру сортировки, которая бы сортировала пять элементов за семь или менее сравнений, то необходимо получить линейный граф

Picture: 228.1

эффективность которого равна $5!/(2^7 \times 1) = 120/128 = 15/16$. Отсюда следует, что все графы, возникающие в процессе сортировки, должны иметь эффективность $\geq \frac{15}{16}$; если бы появился какой-нибудь граф меньшей эффективности, то по крайней мере один из его потомков тоже имел бы меньшую эффективность, и мы бы в конце концов пришли к линейному графу с эффективностью $< \frac{15}{16}$. В общем случае это рассуждение показывает, что все графы, соответствующие узлам дерева для некоторой процедуры сортировки n элементов, должны иметь эффективность $\geq n!/2^l$, где $l + 1$—число уровней в дереве. Это еще один способ доказательства неравенства $S(n) \geq \lceil \log_2 n! \rceil$, хотя такое рассуждение на самом деле не сильно отличается от приведенного выше.

Граф (21) имеет эффективность 1, поскольку $T(G) = 15$ и граф G был получен за три сравнения. Чтобы выяснить, какие вершины должны участвовать в следующем сравнении, можно построить *матрицу сравнений*

$$C(G) = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 15 & 10 & 15 & 11 \\ 0 & 0 & 5 & 15 & 7 \\ 5 & 10 & 0 & 15 & 9 \\ 0 & 0 & 0 & 0 & 3 \\ 4 & 8 & 6 & 12 & 0 \end{pmatrix} \end{array}, \tag{24}$$

где C_{ij} есть $T(G_1)$ для графа G_1, полученного из G путем добавления дуги $i \to j$. Если мы, например, сравним K_c с K_e, то 15 перестановок, согласующихся с G, распадутся на две группы: $C_{ec} = 6$, в которых $K_e < K_c$, и $C_{ce} = 9$, в которых $K_c < K_e$. Последний граф имел бы эффективность $15/(2 \times 9) = \frac{5}{6} < \frac{15}{16}$, так что это сравнение не может привести к семишаговой процедуре сортировки. Следующим сравнением, для того чтобы сохранить эффективность $\geq \frac{15}{16}$, *обязано* быть $K_b : K_e$.

Понятие "эффективность" особенно полезно при рассмотрении "связных компонент" графов. Возьмем, например, граф

Picture: p.229.1

он состоит из двух компонент

Picture: p.229.2

где ни одна дуга не соединяет G' и G''; следовательно, он был образован путем нескольких сравнений вершин только G' и независимо нескольких сравнений вершин только G''. В общем случае предположим, что граф $G = G' + G''$ не содержит дуг, связывающих G' и G'', где G' и G'' имеют соответственно n' и n'' вершин; легко видеть, что

$$T(G) = \binom{n'+n''}{n'} T(G')T(G''), \tag{25}$$

поскольку каждая перестановка, согласующаяся с G, получается путем выбора n' элементов, которые считаются принадлежащими графу G', и последующего составления перестановки, согласующейся с G', и независимо, перестановки, согласующейся с G''. Пусть внутри G' выполнено k' сравнений, а внутри G''—соответственно k'' сравнений; получаем основной результат

$$E(G) = \frac{(n'+n'')!}{2^{k'+k''}T(G)} = \frac{n'!}{2^{k'}T(G')} \cdot \frac{n''!}{2^{k''}T(G'')} = E(G')E(G''), \tag{26}$$

показывающий, что между эффективностью графа и эффективностями его компонент существует простая связь. Поэтому мы можем ограничить наше рассмотрение графами, имеющими только одну компоненту.

Теперь предположим, что G' и G''—однокомпонентные графы, и мы хотим связать их вместе, сравнив вершину x графа G' с вершиной y графа G''. Нужно выяснить, насколько эффективным получится граф. Для этой цели нам необходима функция, которую можно обозначить через

$$\left(\begin{matrix} p \\ m \end{matrix} < \begin{matrix} q \\ n \end{matrix}\right), \tag{27}$$

равная по определению числу перестановок, согласующихся с графом

Picture: p.230

Таким образом, $\left(\begin{smallmatrix} p \\ m \end{smallmatrix} < \begin{smallmatrix} q \\ n \end{smallmatrix}\right)$ есть произведение $\binom{m+n}{m}$ на вероятность того, что p-й наименьший элемент из множества m чисел меньше q-го наименьшего элемента из независимо выбранного множества n чисел. В упр. 17 показано, что функцию $\left(\begin{smallmatrix} p \\ m \end{smallmatrix} < \begin{smallmatrix} q \\ n \end{smallmatrix}\right)$ можно выразить через биномиальные коэффициенты двумя способами:

$$\left(\begin{matrix} p \\ m \end{matrix} < \begin{matrix} q \\ n \end{matrix}\right) = \sum_{0 \le k < q} \binom{m-p+n-k}{m-p} \binom{p-1+1}{p-1} = \sum_{p \le j \le m} \binom{n-q+m-j}{n-q} \binom{q-1+j}{q-1}. \tag{29}$$

(Между прочим, алгебраически никоим образом не очевидно, что эти две суммы произведений биномиальных коэффициентов должны быть равны.) Имеем также формулы

$$\left(\begin{matrix} p \\ m \end{matrix} < \begin{matrix} q \\ n \end{matrix}\right) + \left(\begin{matrix} q \\ n \end{matrix} < \begin{matrix} p \\ m \end{matrix}\right) = \binom{m+n}{m}, \tag{30}$$

$$\left(\begin{matrix} q \\ n \end{matrix} < \begin{matrix} p \\ m \end{matrix}\right) = \left(\begin{matrix} m+1-p \\ m \end{matrix} < \begin{matrix} n+1-q \\ n \end{matrix}\right). \tag{31}$$

Для ясности рассмотрим два графа

Picture: p.231

Нетрудно показать простым перечислением, что $T(G') = 42$ и $T(G'') = 5$; так что если G—граф с 11 вершинами, содержащий G' и G'' в качестве своих компонент, то по формуле (25) $T(G) = \binom{11}{4} \cdot 42 \cdot 5 = 69\,300$. Это число перестановок слишком внушительно, чтобы их можно было выписать и, таким образом, выяснить, в скольких из них $x_i < y_j$ для всех i и j. Однако это вычисление менее чем за час можно проделать вручную следующим образом. Построим матрицы $A(G')$ и $A(G'')$, где A_{ik}—число перестановок, согласующихся с G' (или G''), в которых x_i (или y_i) равно k. Тогда число перестановок, согласующихся с G, в которых x_i меньше y_j, есть сумма по всем p и q, $1 \le p \le 7$, $1 \le q \le 4$, произведений (ip)-го элемента матрицы $A(G')$ на $\left(\begin{smallmatrix} p \\ 7 \end{smallmatrix} < \begin{smallmatrix} q \\ 4 \end{smallmatrix}\right)$ и на (jq)-й элемент матрицы $A(G'')$. Иначе говоря, нужно вычислить произведение матриц $A(G') \cdot L \cdot A(G'')^T$, где $L_{pq} = \left(\begin{smallmatrix} p \\ 7 \end{smallmatrix} < \begin{smallmatrix} q \\ 4 \end{smallmatrix}\right)$. Оно равно

$$\begin{pmatrix} 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 18 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \end{pmatrix} \begin{pmatrix} 210 & 294 & 322 & 329 \\ 126 & 238 & 301 & 325 \\ 70 & 175 & 265 & 315 \\ 35 & 115 & 215 & 295 \\ 15 & 65 & 155 & 260 \\ 5 & 29 & 92 & 204 \\ 1 & 8 & 36 & 120 \end{pmatrix} \begin{pmatrix} 2 & 3 & 0 & 0 \\ 2 & 2 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 48169 & 42042 & 66858 & 64031 \\ 22825 & 16005 & 53295 & 46475 \\ 48169 & 42042 & 66858 & 64031 \\ 22110 & 14850 & 54450 & 47190 \\ 5269 & 2442 & 27258 & 21131 \\ 22825 & 16005 & 53295 & 46475 \\ 5269 & 2442 & 27258 & 21131 \end{pmatrix}.$$

Таким образом, "наилучший" способ соединить G' с G''—это сравнить x_1 с y_2; в 42 042 случаях получим $x_1 < y_2$ и в $69\ 300 - 42\ 042 = 27\ 258$ случаях—$x_1 > y_2$. (В силу симметрии, по существу, к тем же результатам привели бы сравнения x_3 с y_2, x_5 с y_3 или x_7 с y_3.) Эффективность полученного графа для $x_1 < y_2$ равна

$$\frac{69\ 300}{84\ 084} E(G')E(G''),$$

т. е. не особенно высока; следовательно, по-видимому, вообще не стоит ни в одном методе сортировки применять соединение G' с G''! Смысл этого примера в том, что мы смогли принять такое решение, не производя непомерно больших вычислений. Этими идеями можно воспользоваться также для независимого подтверждения принадлежащего Марку Уэлсу доказательства того, что $S(12) = 30$. Начав с графа, содержащего одну вершину, мы можем повторять попытки добавления сравнений к одному из наших графов G или к паре компонент графа G' и G'' с таким расчетом, чтобы оба полученных графа содержали 12

Picture: Рис. 36. Некоторые графы и их эффективности, полученные в начале длинного доказательства неравенства $S(12) > 29$.

или менее вершин и обладали эффективностью $\geq 12!/2^{29} \approx 0.89221$. Всякий раз, как это оказывается возможным, мы выбираем граф с меньшей эффективностью и добавляем его к нашему множеству, если только он не является изоморфным одному из уже включенных в множество графов (или двойственным к нему, т. е. получается обращением отношения порядка). Если оба полученных графа имеют одинаковую эффективность, то произвольным образом выбирается один из них. Первые 24 графа, полученные таким способом, изображены на рис. 36, где приведены также их эффективности.

При помощи вычислительной машины было построено ровно 1594 графа, прежде чем этот процесс завершился. Поскольку граф не был получен, можно сделать вывод о том, что $S(12) > 29$. Весьма правдоподобно, что и для доказательства неравенства $S(22) > 70$ можно произвести аналогичный эксперимент за вполне разумное время, поскольку $22!/2^{70} \approx 0.952$—это чрезвычайно высокая эффективность для сортировки за 70 шагов. (Из 1594 найденных графов с 12 или менее вершинами всего 92 имеют столь высокую эффективность.)

Промежуточные результаты дают веские основания предположить, что $S(13) = 33$, и, следовательно, сортировка вставками и слиянием не оптимальна при $n = 13$. Но до сих пор никому не удалось обнаружить *ни одного* такого значения n, что $S(n) < F(n)$.

Наверняка можно доказать, что $S(16) < F(16)$, поскольку $F(16)$—это как раз такое число сравнений, какое требуется, чтобы сначала отсортировать 10 элементов за $S(10)$ шагов, а затем посредством бинарных вставок вставить по одному остальные шесть элементов. Непременно должен существовать более хороший способ!

Среднее число сравнений. До сих пор мы рассматривали процедуры, наилучшие в том смысле, что они не плохи в наихудшем случае; мы искали "минимаксные" процедуры, минимизирующие *максимальное* число сравнений. Поищем теперь "минисредние" процедуры, минимизирующие *среднее* число сравнений в предположении, что входные данные случайны, т. е. все перестановки равновероятны.

Рассмотрим еще раз изображенное на рис. 34 представление процедуры сортировки в виде дерева. Среднее число сравнении по всем перестановкам для этого дерева равно

$$\frac{2+3+3+3+3+2}{6} = 2\frac{2}{3}.$$

В общем случае среднее число сравнений для метода сортировки есть *длина внешнего пути дерева,* деленная на n. (Напомним, что длина внешнего пути—это сумма всех расстояний от корня до каждого из внешних узлов; см. п. 2.3.4.5.) Из обсуждения в п. 2.3.4.5 легко видеть, что минимум длины внешнего пути достигается на таком бинарном дереве с N внешними узлами, у которого имеется $2^q - N$ внешних узлов на уровне $q-1$ и $2N - 2^q$ на уровне q, где $q = \lceil \log_2 N \rceil$. (Корень находится на нулевом уровне.) Таким образом, минимальная длина внешнего пути равна

$$(q-1)(2^q - N) + q(2N - 2^q) = (q+1)N - 2^q. \tag{33}$$

Имеется еще один интересный способ охарактеризовать минимальную длину пути: *расширенное бинарное дерево имеет минимальную длину внешнего пути тогда и только тогда, когда существует такое число l, что все внешние узлы находятся на уровнях l и $l+1$.* (См. упр. 20.)

Если положить $q = \log_2 N + \theta$, где $0 \leq \theta < 1$, то формула минимальной длины внешнего пути примет вид

$$N(\log_2 N + 1 + \theta - 2^\theta). \tag{34}$$

График функции $1+\theta-2^\theta$ изображен на рис. 37; при $0<\theta<1$ она принимает положительные, но очень малые значения, не превышающие

$$1-(1+\ln\ln 2)/(\ln 2) = 0.08607\ 13320\ 55934+. \tag{35}$$

Picture: Рис. 37. Функция $1+\theta-2^\theta$.

Таким образом, минимальная возможная средняя длина пути, которая получается в результате деления (34) на N, не может быть меньше $\log_2 N$ и больше $\log_2 N + 0.0861$. (Этот результат впервые получил Э. Глисон в неопубликованной заметке (1956).)

Если теперь положим $N=n!$, то получим нижнюю оценку среднего числа сравнений по всем схемам сортировки. Заметим, что оценка равна $\log_2 n! + O(1) = n\log_2 n - n/(\ln 2) + O(\log n)$.

Пусть $\bar F(n)$—среднее число сравнений, выполняемых алгоритмом сортировки вставками и слиянием; имеем

$n=$	1	2	3	4	5	6	7	8
Нижняя оценка$(33)=$	0	2	16	112	832	6896	62368	619904
$n!\bar F(n)=$	0	2	16	112	832	6912	62784	623232

Итак, сортировка вставками и слиянием оптимальна в обоих смыслах при $n\ge 5$, однако при $n=6$ при таком методе выполняется в среднем $6912/720 = 9.6$ сравнения вместо возможных, согласно нижней оценке, $6896/720 = 9.577777\ldots$ сравнения. Немного подумав, нетрудно понять, почему это так: некоторые "удачные" перестановки шести элементов сортируются методом вставок и слияний всего за восемь сравнений, и тогда дерево сравнений имеет внешние узлы на трех, а не на двух уровнях. Из-за этого увеличивается суммарная длина пути. В упр. 24 показано, что можно построить процедуру сортировки шести элементов, требующую всегда девять или десять сравнений; следовательно, этот метод превосходит метод вставок и слияний в среднем и не хуже него в худшем случае.

И. Сезари [thesis (Paris, 1968), p. 37] доказал, что при $n=7$ не существует метода сортировки, при котором бы достигалась нижняя оценка 62368 длины внешнего пути. (Используя результат упр. 22, этот факт можно доказать вручную.) С другой стороны, он построил процедуры, для которых достигается нижняя оценка (33) при $n=9$ или 10. Вообще же задача минимизации среднего числа сравнений гораздо сложнее нахождения функции $S(n)$. Вполне даже возможно, что при некоторых n все методы, минимизирующие *среднее* число сравнений, в худшем случае требуют *более* $S(n)$ сравнений.

Упражнения

1. [20] Нарисуйте деревья сравнений для сортировки четырех элементов методами (a) бинарных вставок; (b) простого двухпутевого слияния. Каковы длины внешних путей для этих деревьев?
2. [M24] Докажите, что $B(n)\le L(n)$, и найдите все значения n, при которых имеет место равенство.
3. [M22] Если допускаются равные ключи, то при сортировке трех элементов возможны 13 исходов:

$$\begin{gathered}K_1=K_2=K_3,\quad K_1=K_2<K_3,\quad K_1=K_3<K_2,\\ K_2=K_3<K_1,\quad K_1<K_2=K_3,\quad K_2<K_1=K_3,\\ K_3<K_1=K_2,\quad K_1<K_2<K_3,\quad K_1<K_3<K_2,\\ K_2<K_1<K_3,\quad K_2<K_3<K_1,\quad K_3<K_1<K_2,\quad K_3<K_2<K_1.\end{gathered}$$

   Обозначим через P_n число возможных исходов при сортировке n элементов, если допускаются равные ключи, так что $(P_0,P_1,P_2,P_3,P_4,P_5,\ \ldots) = (1,1,3,13,75,541,\ \ldots)$. Докажите, что производящая функция $P(z)=\sum_{n\ge 0}P_n z^n/n!$ равна $1/(2-e^z)$. *Указание:* покажите, что

$$P_n=\sum_{k>0}\binom{n}{k}P_{n-k}\qquad\text{при } n>0.$$

4. [BM27] (О. А. Гросс.) Найдите предел последовательности чисел P_n из упр. 3 при $n\to\infty$. [*Возможное указание:* рассмотрите частичное разложение в дробь $\operatorname{ctg} z$.]
5. [16] Если допускаются равные ключи, то каждое сравнение может иметь не два, а три результата: $K_i<K_j$, $K_i=K_j$, $K_i>K_j$. В этой общей ситуации алгоритмы сортировки можно представлять в виде расширенных *тернарных* деревьев, в которых каждый внутренний узел $i:j$ имеет три поддерева: левое, среднее и правое, соответствующие трем возможным исходам сравнения.

   Нарисуйте расширенное тернарное дерево, определяющее алгоритм сортировки для $n=3$, если допускаются равные ключи. В вашем дереве должно быть 13 внешних узлов, соответствующих 13 возможным исходам, перечисленным в упр. 3.

>6. [M22] Пусть $S'(n)$—минимальное число сравнений, необходимых для сортировки n элементов и выявления всех равенств между ключами, если каждое сравнение имеет три возможных результата, как в упр. 5. Нетрудно обобщить "теоретико-информационное" рассуждение, приведенное в тексте, и показать, что $S'(n) \geq \lceil \log_3 P_n \rceil$, где P_n—функция, изученная в упр. 3 и 4; докажите, что на самом деле $S'(n) = S(n)$.

7. [20] Нарисуйте расширенное тернарное дерево в смысле упр. 5 для сортировки четырех элементов, если известно, что все ключи равны либо 0, либо 1. (Так, например, если $K_1 < K_2$ и $K_3 < K_4$, то понятно, что $K_1 = K_3$ и $K_2 = K_4$!) Добейтесь минимального числа сравнений в среднем, считая, что все 2^4 возможных исходных файлов равновероятны.

8. [26] Нарисуйте расширенное тернарное дерево, как в упр. 7, для сортировки четырех элементов, если известно, что все ключи равны либо -1, либо 0, либо $+1$. Добейтесь минимального числа сравнений в среднем, считая, что все 3^4 возможных исходных файлов равновероятны.

9. [M20] Каково минимальное число сравнений в наихудшем случае при сортировке n элементов, как в упр. 7, когда известно, что все элементы равны либо 0, либо 1?

>10. [M25] Каково минимальное *среднее* число сравнений при сортировке n элементов, как в упр. 7, когда известно, что все ключи равны либо 0, либо 1? Результат представьте в виде функции от n.

11. [BM25] Пусть $S_m(n)$—минимальное число сравнений, необходимое в наихудшем случае для сортировки n элементов, как в упр. 5, если известно, что все ключи принадлежат множеству $\{ 1, 2, \ldots, m \}$. [Таким образом, согласно упр. 6, $S_n(n) = S(n)$.] Докажите, что $S_m(n)$ стремится к $n \log_2 m$ при фиксированном m и $n \to \infty$.

>12. [M25] (У. Г. Бурисиус, около 1954 г.) Предположим, что равные ключи могут встречаться, но мы хотим просто отсортировать элементы $\{ K_1, K_2, \ldots, K_n \}$ так, чтобы определить перестановку $a_1\, a_2\, \ldots a_n$, такую, что $K_{a_1} \leq K_{a_2} \leq \ldots \leq K_{a_n}$; нам не важно, имеет ли место равенство между элементами K_{a_i} и $K_{a_{i+1}}$.

Будем говорить, что дерево сравнений *сильно* сортирует последовательность ключей, если оно сортирует эту последовательность в вышеуказанном смысле, независимо от того, какой путь выбирать в узлах $i : j$, для которых $K_i = K_j$. (Дерево бинарное, а не тернарное.)

a) Докажите, что дерево, не содержащее избыточных сравнений, сильно сортирует любую последовательность ключей тогда и только тогда, когда оно сортирует любую последовательность различных ключей.

b) Докажите, что дерево сравнений сильно сортирует любую последовательность ключей тогда и только тогда, когда оно сильно сортирует любую последовательность нулей и единиц.

13. [M28] Докажите утверждение (17).

14. [M24] Выразите сумму (19) в "замкнутом виде".

15. [M21] Определите асимптотическое поведение функции $B(n)$ и $F(n)$ с точностью до $O(\log n)$. [*Указание:* покажите, что в обоих случаях коэффициент при n содержит функцию, изображенную на рис. 37.]

16. [BM26] (Ф. Хуан и Ш. Линь.) Докажите, что при $n \geq 22$ выполняется неравенство $F(n) > \lceil \log_2 n! \rceil$.

17. [M20] Докажите тождество (29).

18. [20] Если бы процедура, начало которой изображено на рис. 36, породила граф

Picture: p.236

с эффективностью $12!/2^{29}$, то было ли бы тем самым доказано, что $S(12) = 29$?

19. [40] Проведите эксперименты со следующим эвристическим правилом решения относительно того, какую пару ключей сравнивать следующей при конструировании дерева сравнений. Пусть на каждой стадии сортировки ключей $\{ K_1, \ldots, K_n \}$ число ключей, о которых на основании выполненных до сих пор сравнений известно, что они $\leq K_i$, обозначается через u_i, а число ключей, о которых известно, что они $\geq K_i$, обозначается через v_i, $1 \leq i \leq n$.

Перенумеруем ключи так, чтобы последовательность u_i/v_i стала возрастающей: $u1/v_1 \leq u_2/v_2 \leq \ldots \leq u_n/v_n$. Теперь сравним $K_i : K_{i+1}$, где i—индекс, минимизирующий выражение $|u_i v_{i+1} - u_{i+1} v_i|$. (Хотя этот метод использует гораздо меньше информации, чем содержится в полной матрице сравнений, подобной (24), он, как оказывается, во многих случаях дает оптимальные результаты.)

>20. [M26] Докажите, что расширенное бинарное дерево имеет минимальную длину внешнего пути тогда и только тогда, когда существует такое число l, что все внешние узлы находятся на уровнях l и $l+1$ (или, быть может, только на уровне l).

21. [M21] *Высотой* расширенного бинарного дерева называется максимальный номер уровня, на котором есть внешние узлы. Пусть x—внутренний узел расширенного бинарного дерева; обозначим через $t(x)$ число внешних узлов-потомков узла x, а через $l(x)$ корень левого поддерева узла x. Если x—внешний

узел, то положим $t(x) = 1$. Докажите, что расширенное бинарное дерево имеет минимальную высоту среди всех бинарных деревьев с тем же числом узлов тогда и только тогда, когда для всех его внутренних узлов x выполняется неравенство

$$|t(x) - 2t(l(x))| \le 2^{\lceil \log_2 t(x) \rceil} - t(x).$$

22. [M24] Продолжение упр. 21. Докажите, что бинарное дерево имеет минимальную длину внешнего пути среди всех бинарных деревьев с тем же числом узлов тогда и только тогда, когда для всех его внутренних узлов x выполняются неравенства

$$|t(x) - 2t(l(x))| \le 2^{\lceil \log_2 t(x) \rceil} - t(x) \text{ и } |t(x) - 2t(l(x))| \le t(x) - 2^{\lfloor \log_2 t(x) \rfloor}.$$

[Так, например, если $t(x) = 67$, то должно быть $t(l(x)) = 32$, 33, 34 или 35. Если нужно просто минимизировать высоту дерева, то, согласно предыдущему упражнению, достаточно, чтобы $3 \le t(l(x)) \le 64$.]

23. [10] В тексте доказано [см. формулу (34)], что среднее число сравнений, выполняемых любым методом сортировки n элементов, не может быть меньше $\lceil \log_2 n! \rceil \approx n \log_2 n$. Однако при сортировке вставками в несколько списков (алгоритм 5.2.1M) затрачивается в среднем всего $O(n)$ единиц времени. Чем это объясняется?
24. [27] (К. Пикар.) Постройте такое дерево сортировки для шести элементов, чтобы все его внешние узлы располагались на уровнях 10 и 11.
25. [11] Если бы существовала процедура сортировки семи элементов, на которой достигался минимум среднего числа сравнений, вычисляемый при помощи формулы (34), то сколько внешних узлов было бы на уровне 13 соответствующего дерева?
26. [M42] Найдите процедуру сортировки для семи элементов, минимизирующую среднее число выполняемых сравнений.

>27. [20] Пусть известно, что конфигурации ($K_1 < K_2 < K_3$, $K_1 < K_3 < K_2$, $K_2 < K_1 < K_3$, $K_2 < K_3 < K_1$, $K_3 < K_1 < K_2$, $K_3 < K_2 < K_1$) встречаются с вероятностями соответственно (.01, .25, .01, .24, .25, .24). Найдите дерево сравнений, которое бы сортировало такие три элемента с наименьшим средним числом сравнений.

28. [40] Напишите MIX-программу, которая сортирует 5 однословных ключей за минимально возможное время, после чего останавливается. (См. основные правила в начале § 5.2.)
29. [M25] (С. М. Чэйз.) Пусть $a_1\, a_2 \ldots a_n$—перестановка множества $\{1, 2, \ldots, n\}$. Докажите, что любой алгоритм, который распознает, является ли данная перестановка четной или нечетной (т. е. содержит ли она четное или нечетное число инверсий), и основанный исключительно на сравнениях элементов a, должен выполнить не менее $n \log_2 n$ сравнений, хотя он имеет всего два возможных исхода.
30. [M23] *(Оптимальная обменная сортировка.)* Любой алгоритм обменной сортировки в смысле определения, данного в п. 5.2.2, можно представить в виде *дерева сравнений-обменов,* а именно в виде структуры бинарного дерева, внутренние узлы которого имеют вид $< i : j >$, где $i < j$, и интерпретируются следующим образом: "если $K_i \le K_j$, то продвинуться по левой ветви дерева; если $K_i > K_j$, то поменять местами записи i и j и продвинуться по правой ветви дерева" По достижении внешнего узла должны выполняться условия $K_1 \le K_2 \le \ldots \le K_n$. Таким образом, дерево сравнений-обменов отличается от дерева сравнений тем, что оно описывает не только операции сравнения, но и операции перемещения данных.

Обозначим через $S_e(n)$ минимальное число сравнений-обменов, необходимых в наихудшем случае для сортировки элементов при помощи дерева сравнений-обменов. Докажите, что $S_e(n) \le S(n) + n - 1$.

31. [M38] Продолжение упр. 30. Докажите, что $S_e(5) = 8$.
32. [M42] Продолжение упр. 31. Исследуйте значения функции $S_e(n)$ при малых $n > 5$.
33. [M30] (Т. Н. Хиббард.) *Вещественнозначным деревом поиска* порядка x с разрешением δ называется расширенное бинарное дерево, каждый узел которого содержит неотрицательное действительное значение, такое, что (i) значение в любом внешнем узле $\le \delta$; (ii) значение в любом внутреннем узле $\le$ суммы значений двух его сыновей; (iii) значение в корне равно x. *Длина взвешенного пути* такого дерева определяется как сумма по всем внешним узлам номеров уровней этих узлов, умноженных на содержащиеся в них значения.

Докажите, что вещественнозначное дерево поиска порядка x с разрешением 1 имеет минимальную среди всех таких деревьев того же порядка и с тем же разрешением длину взвешенного пути тогда и только тогда, когда в (ii) имеет место равенство и для всех пар значений x_0 и x_1, принадлежащих узлам-братьям, выполняются следующие условия: (iv) не существует целого числа $k \ge 0$, такого, что $x_0 < 2^k < x_1$ или $x_1 < 2^k < x_0$; (v) $\lceil x_0 \rceil - x_0 + \lceil x_1 \rceil - x_1 < 1$. (В частности, если x—целое число, то из условия (v) следует, что все значения в дереве—целые, а условие (iv) эквивалентно результату упр. 22.)

Докажите также, что соответствующая длина взвешенного пути равна $x\lceil\log_2 x\rceil + \lceil x\rceil - 2^{\lceil\log_2 x\rceil}$.

34. [M50] Определите точные значения функции $S(n)$ для бесконечного множества аргументов n.

***Слияние с минимальным числом сравнений** Рассмотрим теперь вопрос, имеющий отношение к предыдущему пункту: каков наилучший способ слияния упорядоченного множества m элементов с упорядоченным множеством n элементов? Обозначим элементы сливаемых множеств через

$$\begin{aligned} A_1 < A_2 < \ldots < A_m, \\ B_1 < B_2 < \ldots < B_n. \end{aligned} \tag{1}$$

Как и в п. 5.3.1, будем предполагать, что все $m+n$ элементов различны. Элементы A среди элементов B могут располагаться $\binom{m+n}{m}$ способами; таким образом, из рассуждения, которым мы воспользовались в задаче о сортировке, следует, что необходимо выполнить по крайней мере

$$\left\lceil \log_2 \binom{m+n}{m} \right\rceil \tag{2}$$

сравнений. Если положить $m = \alpha n$ и устремить n к ∞, оставив α неизменным, то по формуле Стирлинга

$$\log_2 \binom{\alpha n + n}{\alpha n} = n\left((1+\alpha)\log_2(1+\alpha) - \alpha\log_2\alpha\right) - \frac{1}{2}\log_2 n + O(1). \tag{3}$$

Обычная процедура слияния (алгоритм 5.2.4M) выполняет в худшем случае $m+n-1$ сравнений.

Обозначим через $M(m,n)$ функцию, аналогичную $S(n)$, а именно минимальное число сравнений, заведомо достаточное для слияния m элементов с n элементами. Из только что сделанного наблюдения следует, что

$$\left\lceil \log_2 \binom{m+n}{m} \right\rceil \le M(m,n) \le m+n-1 \qquad \text{при всех } m,n \ge 1. \tag{4}$$

Формула (3) показывает, насколько далеко могут отстоять друг от друга нижняя и верхняя оценки. При $\alpha = 1$ (т. е. $m = n$) нижняя оценка равна $2n - \frac{1}{2}\log_2 n + O(1)$, так что обе оценки—величины одного порядка, но разность между ними может быть сколь угодно велика. При $\alpha = 0.5$ (т. е. $m = \frac{1}{2}n$) нижняя оценка равна

$$\frac{3}{2}n\left(\log_2 3 - \frac{2}{3}\right) + O(\log n),$$

что составляет примерно $\log_2 3 - \frac{2}{3} \approx 0.918$ от верхней оценки. С убыванием α разница между верхней и нижней оценками все увеличивается, поскольку стандартный алгоритм слияния разработан главным образом для файлов с $m \approx n$.

При $m = n$ задача о слиянии имеет весьма простое решение; неверной оказывается не верхняя, а *нижняя* оценка (4). Следующую теорему независимо доказали Р. Л. Грэхем и Р. М. Карп примерно в 1968 г.

Теорема М. *При $m \ge 1$ справедливо равенство $M(m,m) = 2m-1$.*

Доказательство Рассмотрим какой-нибудь алгоритм, который осуществляет слияние элементов $A_1 < \ldots < A_m$ с $B_1 < \ldots < B_m$. При сравнении элементов $A_i : B_j$ выберем ветвь $A_i < B_j$, если $i < j$, и ветвь $A_i > B_j$, если $i \ge j$. Слияние должно завершиться конфигурацией

$$B_1 < A_1 < B_2 < A_2 < \ldots < B_m < A_m, \tag{5}$$

поскольку она согласуется со всеми выбранными ветвями. И каждое из $2m-1$ сравнений $B_1 : A_1$, $A_1 : B_2$, $B_2 : A_2$, $\ldots$, $B_m : A_m$ должно быть выполнено явно, иначе нашлись бы по меньшей мере две конфигурации, не противоречащие известным фактам. Если бы мы, например, не сравнили A_1 с B_2, то конфигурация

$$B_1 < B_2 < A_1 < A_2 < \ldots < B_m < A_m$$

была бы неотличима от (5). ∎

Простая модификация этого доказательства дает аналогичную формулу

$$M(m,m+1) = 2m \qquad \text{при } m \ge 0. \tag{6}$$

Нахождение нижних оценок. Теорема M показывает, что "теоретико-информационная" нижняя оценка (2) может сколь угодно далеко отстоять от истинной нижней границы; таким образом, метод доказательства теоремы M дает нам еще один способ нахождения нижних оценок. Такой метод доказательства часто рассматривается как порождение *дьявола,* некоего злого духа, который пытается принудить алгоритм работать как можно медленнее. Когда алгоритм слияния решает сравнить элементы $A_i : B_j$, дьявол так определяет судьбу сравнения, что вынуждает алгоритм избрать наиболее трудный путь. Если бы мы смогли придумать подходящего дьявола, то смогли бы убедиться в том, что всякий правильный алгоритм слияния должен выполнить довольно много сравнений. (Некоторые называют его не дьяволом, а "оракулом" или "демоном"; однако желательно избегать таких терминов в этом контексте, поскольку слово "оракул" имеет совсем другое значение в теории рекурсивных функций, а слово "демон" употребляется в другом смысле в терминологии, связанной с искусственным интеллектом.)

Мы будем использовать *дьяволов с ограниченными возможностями,* произвол которых лимитирован заранее заданными результатами некоторых сравнений. В методе слияния, находящемся под воздействием дьявола с ограниченными возможностями, ограничения считаются неизвестными и поэтому выполняются все необходимые сравнения даже в том случае, когда их результаты предопределены. Например, в доказательстве теоремы M мы ограничили все результаты сравнений условием (5), тем не менее в алгоритме слияния нельзя воспользоваться этим обстоятельством, чтобы избежать хотя бы одного сравнения.

Ограничения, которые мы будем использовать в следующем обсуждении, относятся к левому и правому концам файлов. Левые ограничения обозначаются символами

$\cdot$ (нет ограничения слева),
$\backslash$ (результаты всех сравнений не должны противоречить условию $A_1 < B_1$),
$/$ (результаты всех сравнений не должны противоречить условию $A_1 > B_1$).

Правые ограничения обозначаются символами

$\cdot$ (нет ограничения справа),
$\backslash$ (результаты всех сравнений не должны противоречить условию $A_m < B_n$),
$/$ (результаты всех сравнений не должны противоречить условию $A_m > B_n$).

Существует девять типов дьяволов, обозначаемых символами $\nabla M \phi$, где ∇—левое ограничение, а ϕ—правое. Например, дьявол "$\backslash M \backslash$" должен говорить, что $A_1 < B_j$ и $A_i < B_n$; дьявол "$.M.$" не подчиняется никаким ограничениям. При некоторых малых значениях m и n дьяволы с ограниченными возможностями некоторых типов могут не существовать; при $m = 1$, очевидно, не может быть дьявола "$\backslash M /$".

Займемся теперь построением весьма сложного, но чрезвычайно коварного дьявола для слияний. Он не всегда порождает оптимальные результаты, но дает нижние оценки, которые охватывают множество интересных случаев. Предположим, заданы m и n, а также левые и правые ограничения ∇ и ϕ, и пусть дьявола спрашивают, который из двух элементов A_i и B_j больше. Дьявол может, вообще говоря, применить шесть стратегий сведения задачи к случаю меньшего значения $m + n$:

Стратегия $A(k,l)$ *для* $i \le k \le m$ *и* $1 \le l \le j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{ A_1, \ldots, A_k \}$ с $\{ B_1, \ldots, B_{l-1} \}$ и $\{ A_{k+1}, \ldots, A_m \}$ с $\{ B_l, \ldots, B_n \}$. Тогда последующие сравнения $A_p : B_q$ дадут результаты: $A_p < B_q$, если $p \le k$ и $q \ge l$, и $A_p > B_q$, если $p > k$ и $q < l$; они будут управляться дьяволом $(k, l-1, \nabla, .)$, если $p \le k$ и $q < l$, и дьяволом $(m-k, n+1-l, ., \phi)$, если $p > k$ и $q \ge l$.

Стратегия $B(k,l)$ *для* $i \le k \le m$ *и* $1 \le l < j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{ A_1, \ldots, A_k \}$ с $\{ B_1, \ldots, B_l \}$ и $\{ A_{k+1}, \ldots, A_m \}$ с $\{ B_l, \ldots, B_n \}$ при условии $A_k < B_l < A_{k+1}$. (Заметим, что B_l участвует в обоих списках, подлежащих слиянию. Условие $A_k < B_l < A_{k+1}$ обеспечивает такое положение, при котором слияние одной пары файлов не может дать никакой информации, которая бы помогла при слиянии другой пары.) Тогда последующие сравнения $A_p : B_q$ дадут результаты: $A_p < B_q$, если $p \le k$ и $q \ge l$, и $A_p > B_q$, если $p > k$ и $q \le l$; они будут управляться дьяволом $(k, l, \nabla, \backslash)$, если $p \le k$ и $q \le l$, и дьяволом $(m-k, n+1-l, /, \phi)$, если $p > k$ и $q \le l$.

Стратегия $C(k,l)$ *для* $i < k \le m$ *и* $1 \le l \le j$. Ответить, что $A_i < B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{ A_1, \ldots, A_k \}$ с $\{ B_1, \ldots, B_{l-1} \}$ и $\{ A_k, \ldots, A_m \}$ с $\{ B_l, \ldots, B_n \}$ при условии $B_{l-1} < A_k < B_l$. (Аналогично стратегии B, но файлы A и B меняются ролями.)

Стратегия $A'(k,l)$ *для* $1 \le k \le i$ *и* $j \le l \le n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{ A_1, \ldots, A_{k-1} \}$ с $\{ B_1, \ldots, B_l \}$ и $\{ A_k, \ldots, A_m \}$ с $\{ B_{l+1}, \ldots, B_n \}$. (Аналогично стратегии A.)

Стратегия $B'(k,l)$ *для* $1 \le k \le i$ *и* $j < l \le n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{ A_1, \ldots, A_{k-1} \}$ с $\{ B_1, \ldots, B_l \}$ и $\{ A_k, \ldots, A_m \}$ с $\{ B_l, \ldots, B_n \}$ при условии $A_{k-1} < B_l < A_k$. (Аналогично стратегии B.)

Стратегия $C'(k,l)$ *для* $1 \le k \le i$ *и* $j \le l \le n$. Ответить, что $A_i > B_j$, и потребовать, чтобы последующие операции осуществляли слияния $\{A_1, \ldots, A_k\}$ с $\{B_1, \ldots, B_l\}$ и $\{A_k, \ldots, A_m\}$ с $\{B_{l+1}, \ldots, B_n\}$ при условии $B_l < A_k < B_{l+1}$. (Аналогично стратегии C.)

Из-за налагаемых ограничений приведенные выше стратегии не могут применяться в некоторых случаях, которые мы здесь приводим:

Стратегия	Не должна применяться, если
$A(k,1), B(k,1), C(k,1)$	$\nabla = /$
$A'(1,l), B'(1,l), C'(1,l)$	$\nabla = \backslash$
$A(m,l), B(m,l), C(m,l)$	$\phi = /$
$A'(k,n), B'(k,n), C'(k,n)$	$\phi = \backslash$

Обозначим через $\nabla M\phi(m,n)$ максимальную нижнюю оценку, которую можно получить при помощи дьявола из описанного выше класса. Если первое сравнение есть $A_i : B_j$, то каждая стратегия, если она применима, дает неравенства, связывающие эти девять функций, а именно

$$
\begin{aligned}
A(k,l) &: \nabla M\phi(m,n) \ge 1 + \nabla M.(k,l-1) + .M\phi(m-k, n+1-l);\\
B(k,l) &: \nabla M\phi(m,n) \ge 1 + \nabla M\backslash(k,l) + /M\phi(m-k, n+1-l);\\
C(k,l) &: \nabla M\phi(m,n) \ge 1 + \nabla M/(k,l-1) + \backslash M\phi(m+1-k, n+1-l);\\
A'(k,l) &: \nabla M\phi(m,n) \ge 1 + \nabla M.(k-1,l) + .M\phi(m+1-k, n-l);\\
B'(k,l) &: \nabla M\phi(m,n) \ge 1 + \nabla M\backslash(k-1,l) + /M\phi(m+1-k, n+1-l);\\
C'(k,l) &: \nabla M\phi(m,n) \ge 1 + \nabla M/(k,l) + \backslash M\phi(m+1-k, n-l).
\end{aligned}
$$

При фиксированных i и j дьявол примет ту стратегию, которая максимизирует нижнюю оценку, задаваемую правыми частями неравенств; таким образом, $\nabla M_\phi(m,n)$ есть минимум этих нижних оценок по всем $1 \le i \le m$ и $1 \le j \le n$. Если m или n равны нулю, то и значение функции $\nabla M_\phi(m,n)$ равно 0.

Пусть, например, $m = 2$ и $n = 3$, а наш дьявол не ограничен. Если первым выполняется сравнение $A_1 : B_1$, то дьявол может принять стратегию $A'(1,1)$, в результате чего потребуются еще $.M.(0,1) + .M.(2,2) = 3$ сравнения. Если первым выполняется сравнение $A_1 : B_3$, то он может выбрать стратегию $B(1,2)$, тогда потребуются еще $.M\backslash(1,2) + /M.(1,2) = 4$ сравнения. Независимо от того, каково было первое сравнение, дьявол гарантирует выполнение еще по крайней мере 3 сравнений. Следовательно, $.M.(2,3) = 4$.

Не так просто выполнить эти вычисления вручную, но при помощи ЭВМ можно довольно быстро получить таблицы функций $\nabla M\phi$. Эти функции обладают некоторыми очевидными свойствами симметрии

$$/M.(m,n) = .M\backslash(m,n) = \backslash M.(n,m) = .M/(n,m), \tag{7}$$

позволяющими свести наши девять функций всего лишь к четырем:

$$.M.(m,n), \quad /M.(m,n), \quad /M\backslash(m,n) \text{ и } /M/(m,n).$$

В табл. 1 приведены значения для всех $m, n \le 10$. Наш дьявол для слияний определен таким образом, что

$$.M.(m,n) \le M(m,n) \qquad \text{при всех } m,n \ge 0. \tag{8}$$

Это соотношение содержит в качестве частного случая теорему M, поскольку при $|m-n| \le 1$ наш дьявол изберет простую стратегию этой теоремы.

Таблица 1

Нижние оценки для слияний, выполненных при участии "дьявола"

$.M.(m,n)$

	1	2	3	4	5	6	7	8	9	10	n
1	1	2	2	3	3	3	3	4	4	4	
2	2	3	4	5	5	6	6	6	7	7	
3	2	4	5	6	7	7	8	8	9	9	
4	3	5	6	7	8	9	10	10	11	11	
5	3	5	7	8	9	10	11	12	12	13	
6	3	6	7	9	10	11	12	13	14	15	
7	3	6	8	10	11	12	13	14	15	16	
8	4	6	8	10	12	13	14	15	16	17	
9	4	7	9	11	12	14	15	16	17	18	
10	4	7	9	11	13	15	16	17	18	19	
m											

$/M.(m,n)$

	1	2	3	4	5	6	7	8	9	10	n
1	1	2	2	3	3	3	3	4	4	4	
2	1	3	4	4	5	5	6	6	7	7	
3	1	3	5	6	7	7	8	8	9	9	
4	1	4	5	7	8	9	9	10	10	11	
5	1	4	6	8	9	10	11	12	12	13	
6	1	4	6	8	10	11	12	13	14	14	
7	1	4	7	9	10	12	13	14	15	16	
8	1	5	7	9	11	13	14	15	16	17	
9	1	5	8	10	11	13	15	16	17	18	
10	1	5	8	10	12	14	15	17	18	19	
m											

$/M\backslash(m,n)$

	1	2	3	4	5	6	7	8	9	10	n
1	$-\infty$	2	2	3	3	3	3	4	4	4	
2	$-\infty$	2	4	4	5	5	6	6	7	7	
3	$-\infty$	2	4	6	6	7	8	8	8	9	
4	$-\infty$	2	5	6	8	8	9	10	10	11	
5	$-\infty$	2	5	7	8	10	10	11	12	13	
6	$-\infty$	2	5	7	9	10	12	13	14	14	
7	$-\infty$	2	5	8	10	11	12	14	15	16	
8	$-\infty$	2	6	8	10	12	13	15	16	17	
9	$-\infty$	2	6	9	10	12	14	16	17	18	
10	$-\infty$	2	6	9	11	13	15	16	18	19	
n	1	2	3	4	5	6	7	8	9	10	

$/M/(m,n)$

	1	2	3	4	5	6	7	8	9	10	n
1	1	1	1	1	1	1	1	1	1	1	
2	1	3	3	4	4	4	4	5	55	2	
3	1	3	5	5	6	6	7	7	8	8	
4	1	4	5	7	7	8	9	9	9	10	
5	1	4	6	7	9	9	10	11	11	12	
6	1	4	6	8	9	11	11	12	13	14	
7	1	4	7	9	10	11	13	14	15	15	
8	1	5	7	9	11	12	14	15	16	17	
9	1	5	8	9	11	13	15	16	17	18	
10	1	5	8	10	12	14	15	17	18	19	
m											

Рассмотрим теперь несколько простых соотношений, которым удовлетворяет функция M:

$$M(m,n) = M(n,m), \tag{9}$$

$$M(m,n) \le M(m,n+1), \tag{10}$$

$$M(k+m,n) \le M(k,n) + M(m,n), \tag{11}$$

$$M(m,n) \le \max(M(m,n-1)+1, M(m-1,n)+1), \tag{12}$$

$$M(m,ni) \le \max(M(m,n-2)+1, M(m-1,n)+2) \qquad \text{при } m \ge 1,\ n \le 2. \tag{13}$$

Соотношение (12) следует из обычной процедуры слияния, если начать со сравнения элементов $A_1 : B_1$. Соотношение (13) выводится аналогично, только в первую очередь сравниваются $A_1 : B_2$; если $A_1 > B_2$, то нужно еще $M(m,n-2)$ сравнений, если же $A_1 < B_2$, то можно вставить A_1 в соответствующее место и слить $\{ A_2, \ldots, A_m \}$ с $\{ B_1, \ldots, B_n \}$. Нетрудно видеть, что неравенства (12) и (13) можно обобщить:

$$M(m,n) \le \max(M(m,n-k)+1, \quad M(m-1,n)+1+\lceil \log_2 k \rceil) \qquad \text{при } m \ge 1,\ n \ge k, \tag{14}$$

выполнив сначала сравнение $A_1 : B_k$, и применив бинарный поиск в случае $A_1 < B_k$.

Оказывается, $M(m,n) = .M.(m,n)$ при всех $m,n \le 10$, так что в табл. 1 в действительности приведены оптимальные значения для слияний. Это можно доказать, применяя соотношения (9)–(14), а также специальные построения для $(m,n) = (2,8)$, $(3,6)$ и $(5,9)$, которые приводятся в упр. 8, 9 и 10.

С другой стороны, такой дьявол не всегда дает наилучшую возможную нижнюю оценку; простейший пример: $m = 3$, $n = 11$, когда $.M.(3,11) = 9$, а $M(3,11) = 10$. Чтобы понять, где же в этом случае наш дьявол "промахнулся", нужно изучить мотивы, на которых основаны его решения; при дальнейшем тщательном исследовании обнаруживается, что если $(i,j) \ne (2,6)$, то дьявол может отыскать стратегии, требующие 10 сравнений; если же $(i,j) = (2,6)$, то ни одна стратегия не превосходит стратегии $A(2,4)$, которая приводит к нижней оценке $1 + .M.(2,3) + .M.(1,8) = 9$. Необходимо, но не достаточно, чтобы процесс заканчивался слиянием $\{ A_1, A_2 \}$ с $\{ B_1, B_2, B_3 \}$ и $\{ A_3 \}$ с $\{ B_4, \ldots, B_{11} \}$; таким образом, нижняя оценка в этом случае оказывается не точной.

Аналогично можно показать, что $.M.(2,38) = 10$, в то время как $M(2,38) = 11$; значит, наш дьявол не достаточно искусен, чтобы справиться со случаем $m = 2$. Однако существует бесконечный класс значений, для которых он работает безупречно.

Теорема К.

$$M(m, m+2) = 2m+1 \qquad \textit{при } m \geq 2,$$
$$M(m, m+3) = 2m+2 \qquad \textit{при } m \geq 4,$$
$$M(m, m+4) = 2m+3 \qquad \textit{при } m \geq 6.$$

Доказательство На самом деле мы можем доказывать эти соотношения, заменив M на $.M.$; при малых от эти результаты были получены с помощью ЭВМ, поэтому можно предполагать, что m достаточно велико. Можно также считать, что первое сравнение есть $A_i : B_j$, где $i \leq \lceil m/2 \rceil$. Если $j \leq i$, то воспользуемся стратегией $A'(i, i)$; тогда получим

$$.M.(m, m+d) \geq 1 + .M.(i-1, i) + .M.(m+1-i, m+d-i) = 2m + d - 1,$$

применив индукцию по d, $d \leq 4$. Если $j > i$, то воспользуемся стратегией $A(i, i+1)$; применив индукцию по m, получим

$$.M.(m, m+d) \geq 1 + .M.(i, i) + .M.(m-i, m+d-i) = 2m + d - 1.$$

∎

Первые два утверждения теоремы К получили Ф. Хуан и Ш. Линь в 1969 г. Это доказательство дает основания предположить, что $M(m, m+d) = 2m + d - 1$ при всех достаточно больших m, где d фиксировано. (Ср. с упр. 6.)

Верхние оценки. Рассмотрим теперь *верхние* оценки функции $M(m, n)$; хорошие верхние оценки соответствуют эффективным алгоритмам слияния.

При $m = 1$ задача слияния эквивалентна задаче вставки, когда имеется $n+1$ мест между элементами $B_1, \ldots, B_n$, куда может попасть элемент A_1. В этом случае нетрудно видеть, что *любое* расширенное бинарное дерево с $n+1$ внешними узлами есть дерево для некоторого метода слияния! (См. упр. 2.) Следовательно, можно выбрать оптимальное бинарное дерево, реализовав теоретико-информационную нижнюю оценку

$$1 + \lfloor \log_2 n \rfloor = M(1, n) = \lceil \log_2(n+1) \rceil. \tag{15}$$

Разумеется, бинарный поиск (п. 6.2.1)—простейший способ. позволяющий достичь этого значения.

Случай $m = 2$ чрезвычайно интересен, но он гораздо сложнее. Его полностью исследовали Р. Л. Грэхем, Ф. К. Хуан и Ш. Линь (см. упр. 11, 12, 13); имеем

$$M(2, n) = \left\lceil \log_2 \frac{7}{12}(n+1) \right\rceil + \left\lceil \log_2 \frac{14}{17}(n+1) \right\rceil. \tag{16}$$

Мы видели, что при $m = n$ оптимальна обычная процедура слияния, а при $m = 1$ оптимальна довольно сильно отличающаяся от нее процедура бинарного поиска. Нам же нужен некоторый промежуточный метод, объединяющий в себе лучшие черты алгоритмов обычного слияния и бинарного поиска. Формула (14) наводит на мысль о следующем алгоритме, которым мы обязаны Ф. К. Хуану и Ш. Линю [*SIAM J. Computing,* **1** (1972), 31–39].

Алгоритм Н. *(Бинарное слияние.)*

H1 Если m или n равно 0, то остановиться.
Если $m \leq n$, то установить $t \leftarrow \lfloor \log_2(n/m) \rfloor$.
Если $m > n$, установить $t \leftarrow \lfloor \log_2(m/n) \rfloor$ и перейти к **Н4**.

H2 Сравнить $A_m : B_{n+1-2^t}$. Если A_m меньше, то установить $n \leftarrow n - 2^t$ и возвратиться к шагу **Н1**.

H3 Воспользовавшись методом бинарного поиска (который требует еще ровно t сравнений), вставить A_m в соответствующее место среди $\{ B_{n+1-2^t}, \ldots, B_n \}$. Если k—максимальный индекс, такой, что $B_k < A_m$, то установить $m \leftarrow m - 1$ и $n \leftarrow k$. Возвратиться к **Н1**.

H4 (Шаги **Н4** и **Н5** подобны шагам **Н2** и **Н3**, но переменные m, n, A, B меняются ролями.). Если $B_n < A_{m+1-2^t}$, то установить $m \leftarrow m - 2^t$ и возвратиться к шагу **Н1**.

H5 Вставить B_n в соответствующее место среди элементов A. Если k—максимальный индекс, такой, что $A_k < B_n$, то установить $m \leftarrow k$ и $n \leftarrow n - 1$. Возвратиться к шагу **Н1**. ∎

В качестве примера работы этого алгоритма в табл. 2 показан процесс, слияния трех ключей $\{ 087, 503, 512 \}$ с тринадцатью ключами $\{ 061, 154, \ldots, 908 \}$; в этом примере выполняется восемь сравнений.

Таблица 2

Пример применения метода бинарного слияния

A (сравниваются выделенные элементы)			B													Вывод			
{087	503	512}	{061	154	170	275	426	509	612	653	677	703	765	897	908}				
{087	603	512}	{061	151	170	275	426	509	612	653	677}				{703	765	897	908}	
{087	503	512}	{061	154	170	275	426	509	612}				{653	677	703	765	897	908}	
{087	503	512}	{061	154	170	275	426	509	612}			{653	677	703	765	897	908}		
{087	503}		{061	154	170	275	426	509}			{512	612	653	677	703	765	897	908}	
{087	503}		{061	154	170	275	426	509}			{512	612	653	677	793	765	897	908}	
{087}			{061	154	170	275	426}		{503	509	512	612	653	677	703	765	897	908}	
{087}			{061}		{154	170	275	426	503	509	512	612	653	677	703	765	897	908}	
			{061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908}	

Пусть $H(m,n)$—максимальное число сравнений, выполняемых алгоритмом Хуана и Линя. Чтобы вычислить $H(m,n)$, можно предположить, что $k=n$ в шаге H3 и $k=m$ в шаге H5, поскольку при помощи индукции по n нетрудно доказать, что $H(m,n)\le H(m,n+1)$ при всех $n\ge m$. Таким образом, при $m\le n$ имеем

$$H(m,n)=\max(H(m,n-2^t)+1, H(m-1,n)+t+1) \qquad \text{при } 2^t m\le n<2^{t+1}m. \tag{17}$$

Заменим n на $2n+\varepsilon$ ($\varepsilon=0$ или 1) и получим

$$H(m,2n+\varepsilon)=\max(H(m,2n+\varepsilon-2^{t+1})+1, H(m-1,2n+\varepsilon)+t+2) \qquad \text{при } 2^t m\le n<2^{t+1}m.$$

Отсюда вытекает, если применить индукцию по n, что

$$H(m,2n+\varepsilon)=H(m,n)+m \qquad \text{при } m\le n,\ \varepsilon=0 \text{ или } 1. \tag{18}$$

Легко видеть также, что $H(m,n)=m+n-1$, если $m\le n<2m$. Следовательно, многократное применение формулы (18) даст нам общую формулу

$$H(m,n)=m+\lceil n/2^t\rceil-1+tm \qquad \text{при } m\le n,\ t=\lceil\log_2(n/m)\rceil. \tag{19}$$

Полагая $m=\alpha n$ и $\theta=\log_2(n/m)-t$, найдем

$$H(\alpha n,n)=\alpha n(1+2^\theta-\theta-\log_2\alpha)+O(1) \tag{20}$$

при $n\to\infty$. Из формул (5.3.1-35) известно, что $1.9139<1+2^\theta-\theta\le 2$. Следовательно, (20) можно сравнить с теоретико-информационной нижней оценкой (3). Хуан и Линь доказали (см. упр. 17), что

$$H(m,n)<\left\lceil\log_2\binom{m+n}{m}\right\rceil+\min(m,n). \tag{21}$$

Алгоритм слияния Хуана—Линя, который можно было бы назвать алгоритмом "бинарного слияния", не всегда оптимален, но обладает тем неоценимым достоинством, что его довольно легко запрограммировать. Он сводится к "децентрированному бинарному поиску" при $m=1$ и к обычной процедуре слияния при $m\approx n$, так что это золотая середина между двумя указанными методами. Кроме того, во многих случаях он *является* оптимальным (см. упр. 16).

Формула (18) наводит на мысль о том, что и сама функция M, быть может, удовлетворяет неравенству

$$M(m,n)\le M(m,\lfloor n/2\rfloor)+m. \tag{22}$$

Это и в самом деле так (см. упр. 19). Таблицы значений функции $M(m,n)$ позволяют предположить, что, возможно, имеют место и другие соотношения, такие, как

$$M(m+1,n)\ge 1+M(m,n)\ge M(m,n+1) \qquad \text{при } m\le n; \tag{23}$$

$$M(m+1,n+1)\ge 2+M(m,n). \tag{24}$$

но в настоящее время не известно никаких доказательств этих неравенств.

Упражнения

1. [15] Найдите одно любопытное соотношение, которое связывает функцию $M(m,n)$ и функцию S, определенную в п. 5.3.1. [*Указание:* рассмотрите $S(m+n)$.]
>2. [22] При $m = l$ любой алгоритм слияния, не содержащий избыточных сравнений, определяет расширенное бинарное дерево с $\binom{m+n}{m} = n+1$ внешними узлами. Докажите, что верно и обратное, т. е. каждому расширенному бинарному дереву соответствует некоторый алгоритм слияния с $m = 1$.
3. [M24] Докажите, что $.M.(1,n) = M(1,n)$ при всех n.
4. [M44] Справедливо ли неравенство $.M.(m,n) \geq \lceil \log_2 \binom{m+n}{m} \rceil$ при всех m и n?
5. [M30] Докажите, что $.M.(m,n) \leq .M\backslash(m,n+1)$.
6. [M26] Сформулированное выше доказательство теоремы K требует проверки большого числа случаев с привлечением ЭВМ. Каким образом можно резко сократить число таких случаев?
7. [21] Докажите неравенство (11).
>8. [24] Докажите, что $M(2,8) \leq 6$. Для этого придумайте такой алгоритм слияния двух элементов с восемью другими, который бы выполнял не более шести сравнений.
9. [27] Докажите, что три элемента можно слить с шестью элементами не более чем за семь шагов.
10. [33] Докажите, что пять элементов можно слить с девятью не более чем за двенадцать шагов. [*Указание:* опыт введения дьявола подсказывает, что начать нужно со сравнения $A_1 : B_2$, затем, если $A_1 < B_2$, попытаться сравнить $A_5 : B_8$.]
11. [M40] (Ф. Хуан, Ш. Линь.) Пусть $g_{2k} = \lfloor 2^k \cdot \frac{17}{14} \rfloor$, $g_{2k+1} = \lfloor 2^k \cdot \frac{12}{7} \rfloor$ при $k \geq 0$, так что $(g_0, g_1, g_2, \ldots) = (1,1,2,3,4,6,9,13,19,27,38,54,77, \ldots)$. Докажите, что для слияния двух элементов с g_t элементами требуется в худшем случае более чем t сравнений, однако слить два элемента с $g_t - 1$ можно не более чем за t шагов. [*Указание:* покажите, что если $n \geq g_{t-1}$, и нужно слить $\{A_1, A_2\}$ с $\{B_1, B_2, \ldots, B_n\}$ за t сравнений, то наилучшее первое сравнение—это $A_2 : B_{g_{t-1}}$.]
12. [M21] Пусть $R_n(i,j)$—наименьшее число сравнений, необходимое для сортировки различных объектов $\{\alpha, \beta, X_1, X_2, \ldots, X_n\}$, если заданы соотношения

$$\alpha < \beta, \quad X_1 < X_2 < \ldots < X_n, \quad \alpha < X_{i+1}, \quad \beta > X_{n-j}.$$

[Условие $\alpha < X_{i+1}$ или $\beta > X_{n-j}$ теряет смысл, если $i \geq n$ или $j \geq n$. Поэтому $R_n(n,n) = M(2,n)$.] Ясно, что $R_n(0,0) = 0$. Докажите, что

$$R_n(i,j) = 1 + \min\left(\min_{1\leq k\leq i} \max(R_n(k-1,j), R_{n-k}(i-k,j)), \min_{1\leq k\leq j} \max(R_n(i,k-1), R_{n-k}(i,j-k)) \right)$$

при $0 \leq i \leq n$, $0 \leq j \leq n$, $i + j > 0$.
13. [M42] (Р. Л. Грэхем). Покажите, что решение рекуррентного соотношения из упр. 12 можно выразить следующим образом. Определим функцию $G(x)$ при $0 < x < \infty$ такими правилами:

$$G(x) = \begin{cases} 1, & 0 < x \leq \frac{5}{7}; \\ \frac{1}{2} + \frac{1}{8}G(8x-5), & \frac{5}{7} < x \leq \frac{3}{4}; \\ \frac{1}{2}G(2x-1), & \frac{3}{4} < x \leq 1; \\ 0, & 1 \leq x \leq \infty. \end{cases}$$

(См. рис. 38.) Поскольку $R_n(i,j) = R_n(j,i)$ и так как $R_n(0,j) = M(1,j)$, то можно считать, что $1 \leq i \leq j \leq n$. Пусть $p = \lfloor \log_2 i \rfloor$, $q = \lfloor \log_2 j \rfloor$, $r = \lfloor \log_2 n \rfloor$ и $t = n - 2^r + 1$. Тогда

$$R_n(i,j) = p + q + S_n(i,j) + T_n(i,j),$$

где функции S_n и T_n принимают значения 0 или 1:

$$S_n(i,j) = 1 \text{ тогда и только тогда, когда } q < r \text{ или } (i - 2^p \geq u \text{ и } j - 2^r \geq u);$$
$$T_n(i,j) = 1 \text{ тогда и только тогда, когда } p < r \text{ или } \left(t > \tfrac{6}{7}2^{r-2} \text{ и } i - 2^r \geq v\right),$$

где $u = 2^p G(t/2^p)$ и $v = 2^{r-2}G(t/2^{r-2})$.

(Это, быть может, самое сложное рекуррентное соотношение из всех, которые когда-либо будут решены!)
14. [46] (Хуан и Линь.) Пусть $h_{3k} = 2^k + 2^{k-1} - 1$, $h_{3k+1} = g_{2k} + g_{2k-3} + 2^{k-2}$, $h_{3k+2} = 2g_{2k}$ при $k \geq 2$, за исключением $h_8 = 9$, и начальные значения подобраны так, что $(h_0, h_1, h_2, \ldots) = (1,1,2,2,3,4,5,7,9,11,14,18,23,29,38,4$ Здесь g_k—функция, которая была определена в упр. 11. Докажите (или опровергните), что $M(3,h_t) > t$, $M(3,h_t - 1) \leq t$ при всех t.

15. [12] В шаге H1 алгоритма бинарного слияния может потребоваться вычисление значения $\lfloor \log_2(n/m) \rfloor$. Как можно легко вычислить это значение, не применяя операций деления и взятия логарифма?

Picture: Рис. 38. Функция Грэхема (см. упр. 13).

16. [18] При каких m и n, $1 \le m \le n \le 10$, оптимален алгоритм бинарного слияния Хуана и Линя?
17. [M25] Докажите неравенство (21). [*Указание:* это неравенство не очень жесткое.]
18. [M40] Исследуйте *среднее* число сравнений, выполняемых алгоритмом бинарного слияния.
>19. [23] Докажите, что функция M удовлетворяет неравенству (22).
20. [20] Покажите, что если $M(m, n+1) \le M(m+1, n)$ при всех $m \le n$, то $M(m, n+1) \le 1 + M(m, n)$ при всех $m \le n$.
21. [M47] Докажите или опровергните соотношения (23), (24).
22. [M50] Исследуйте минимальное *среднее* число сравнений, необходимых для слияния m элементов с n элементами.
23. [BM30] (Э. Рейнгольд.) Пусть $\{ A_1, \ldots, A_n \}$ и $\{ B_1, \ldots, B_n \}$—множества, содержащие по n элементов каждое. Рассмотрите алгоритм, который пытается проверить наличие равенства между множествами исключительно путем сравнений на равенство элементов этих множеств. Таким образом, алгоритм задает вопросы типа "$A_i = B_j$?" при некоторых i и j и выбирает дальнейший путь вычислений в зависимости от того, был ли ответ положительным или отрицательным.

Определив подходящего дьявола, докажите, что любой такой алгоритм в наихудшем для себя случае вынужден выполнить не менее $\frac{1}{2}n(n+1)$ сравнений.

*** Выбор с минимальным числом сравнений** При поиске наилучших возможных процедур для выбора t-го элемента в порядке убывания из n элементов мы встречаемся с классом задач, подобных рассмотренным в предыдущем пункте. История этого вопроса восходит к занимательному (хотя и серьезному) очерку преподобного Ч. Л. Доджсона о турнирах по теннису, появившемуся в St. James's Gazette 1 августа 1883 г. (стр. 5–6). Доджсон, который, разумеется, более известен как Льюис Кэррол, рассматривал несправедливые правила, по которым присуждались (и до сих пор присуждаются) призы в турнирах по теннису. Рассмотрим, например, рис. 39, где показан типичный турнир "с выбыванием" между 32 игроками, помеченными 01, 02, ..., 32. В финале игрок 01 одерживает победу над игроком 05, поэтому ясно, что игрок 01—чемпион и заслужил первый приз. Несправедливость проявляется в том, что игрок 05 обычно получает второй приз, хотя он может и не быть вторым игроком. Выиграть второй приз можно, даже если играешь хуже половины игроков турнира. В самом деле, как заметил Доджсон, второй игрок выигрывает второй приз в том и только том случае, если первоначально он и чемпион находились в противоположных половинах турнира; для 2^n игроков это происходит с вероятностью $2^{n-1}/(2^n - 1)$, так что почти в половине случаев второй приз получает не тот игрок! Если проигравшие в полуфинале (игроки 25 и 17 на рис. 39) соревнуются за третий приз, то весьма маловероятно, что третий игрок получит третий приз.

Поэтому Доджсон решил найти такой турнир, который правильно определяет второго и третьего игроков в предположении транзитивности. (Иначе говоря, если игрок A побеждает игрока B, а B побеждает C, то можно считать, что игрок A победит C). Он придумал процедуру, в которой проигравшим дают сыграть еще несколько игр, пока не станет определенно известно, что они хуже других трех игроков. Пример схемы Доджсона приводится на рис. 40, изображающем дополнительный турнир, который следует провести вместе с турниром, показанным на рис. 39. Делается попытка организовать встречи игроков, у которых до сих пор были равные результаты, и исключить матчи между игроками, побежденными одним и тем же человеком. Например, игрок 16 проигрывает 11, а игрок 13 проигрывает 12 в первом туре; после того как игрок 16 проигрывает 13 во втором туре, 16 исключается, так как теперь известно, что он хуже, чем 11, 12 и 13. В третьем туре мы не позволяем номеру 19 играть с 21, так как они оба были побеждены

Picture: Рис. 39. Турнир 32 игроков с выбыванием.

Picture: Рис. 40. Теннисный турнир Льюиса Кэррола (в дополнение к турниру рис. 39).

игроком 18 и мы не могли бы автоматически исключить проигравшего во встрече 19 с 21.

Было бы приятно сообщить, что турнир Льюиса Кэррола оказался оптимальным, но, к сожалению, это не так. Из записи в его дневнике от 23 июля 1883 г. явствует, что он составил этот очерк примерно за шесть часов, и чувствовал, что, "поскольку теннисный сезон приближается к концу, очерк следует написать побыстрее, не слишком увлекаясь качеством". В его процедуре делается больше сравнений, чем необходимо, и она не сформулирована достаточно четко, чтобы квалифицировать ее как алгоритм. С

другой стороны, в ней имеются некоторые очень интересные аспекты, если судить с точки зрения параллельных вычислений. Она также представляется отличным расписанием теннисного турнира, поскольку Кэррол включил в нее несколько драматических эффектов; например, он определил, что два финалиста должны пропустить пятый тур и сыграть "длинный" матч в турах 6 и 7. Однако организаторы турниров, по-видимому, сочли это предложение излишне логичным, и потому система Кэррола, скорее всего, никогда не испытывалась. Вместо этого практикуется метод "рассеивания" более сильных игроков, чтобы они попали в разные части дерева.

На математическом семинаре в 1929–1930 г. Гуго Штейнгауз поставил задачу нахождения минимального числа теннисных матчей, требуемых для определения первого и второго игроков в турнире, если имеется $n > 2$ игроков. Ю. Шрейер [*Mathesis Polska*, **7** (1932), 154–160] привел процедуру, требующую самое большее $n - 2 + \lceil \log_2 n \rceil$ матчей, использовав, по существу, тот же метод, что и первые две стадии процесса, который мы назвали сортировкой посредством выбора из дерева (см. п. 5.2.3, рис. 23), однако не выполняя дополнительных сравнений, содержащих $-\infty$. Шрейер также утверждал, что $n - 2 + \lceil \log_2 n \rceil$— наилучшее возможное значение; но его доказательство было ошибочным, как и еще одна попытка доказательства, предпринятая Е. Слупецки [*Colloquium Mathematician*, **2** (1951), 286–290]. Прошло 32 года, прежде чем С. С. Кислицыным было опубликовано правильное, хотя и очень сложное доказательство [*Сибирский математический журнал*, **5** (1964), 557–564]. Пусть $V_t(n)$ для $1 \le t \le n$ обозначает минимальное число сравнений, требуемых для определения t-го в порядке убывания элемента из n элементов, и пусть $W_t(n)$ равно наименьшему числу сравнений, необходимых для определения наибольшего, второго, ..., t-го элементов всех сразу. Из соображений симметрии имеем

$$V_t(n) = V_{n+1-t}(n); \tag{1}$$

очевидно также, что

$$V_1(n) = W_1(n), \tag{2}$$
$$V_t(n) \le W_t(n), \tag{3}$$
$$W_n(n) = W_{n-1}(n) = S(n). \tag{4}$$

В п. 5.2.3 мы видели, что

$$V_1(n) = n - 1. \tag{5}$$

Есть удивительно простое доказательство этого факта, поскольку каждый участник турнира, кроме чемпиона, должен проиграть по крайней мере одну игру! Обобщая эту идею и используя "дьявола", мы можем без особого труда доказать теорему Шрейера—Кислицына.

Теорема S. *При $n \ge 2$ справедливо равенство $V_2(n) = W_2(n) = n - 2 + \lceil \log_2 n \rceil$.*

Доказательство Предположим, что в турнире, где с помощью некоторой данной процедуры должен определиться второй игрок, участвуют n игроков, и пусть a_j—число игроков, проигравших j или больше матчей. Общее число сыгранных матчей будет тогда равно $a_1 + a_2 + a_3 + \ldots$. Мы не можем определить второго игрока, не выявив заодно и чемпиона (см. упр. 2), поэтому из предыдущих рассуждений $a_1 = n-1$. Для завершения доказательства покажем, что всегда существует последовательность результатов матчей, которая приводит к $a_2 \ge \lceil \log_2 n \rceil - 1$.

Предположим, что к концу турнира чемпион сыграл p игр и победил p игроков; одним из них был второй игрок, а остальные должны проиграть по крайней мере еще по одному разу, поэтому $a_2 \ge p - 1$. Итак, мы можем закончить доказательство, построив дьявола, предопределяющего результаты игр таким образом, чтобы чемпиону пришлось сыграть по крайней мере еще с $\lceil \log_2 n \rceil$ другими участниками турнира.

Пусть дьявол считает, что игрок A лучше, чем B, если A ранее не проигрывал, а B хотя бы однажды проиграл, или если оба не проигрывали, но B выиграл к этому моменту меньше матчей, чем A. При других обстоятельствах дьявол может принимать произвольное решение, не противоречащее некоторому частичному упорядочению.

Рассмотрим результаты завершенного турнира, матчи которого предопределялись таким дьяволом. Мы скажем, что "A превосходит B" тогда и только тогда, когда $A = B$ или A превосходит игрока, который первым победил B. (Только первое поражение игрока существенно в этом отношении, последующие его игры игнорируются. В соответствии с устройством дьявола любой игрок, *первым* победивший какого-то, ни в одной из предыдущих встреч не должен иметь поражений. Отсюда следует, что игрок, который выиграл свои первые p матчей, превосходит на основании этих p игр не более 2^p игроков. (Если $p = 0$, это очевидно, если же $p > 0$, то p-й матч был сыгран против игрока, который либо ранее потерпел поражение,

либо превосходит не более 2^{p-1} игроков.) Чемпион превосходит всех, поэтому он должен был сыграть не менее $\lceil \log_2 n \rceil$ матчей. ∎

Таким образом, задача нахождения второго в порядке убывания элемента полностью решена в минимаксном смысле. В упр. 6 показано, что можно дать простую формулу для минимального числа сравнений, необходимых для выявления второго элемента множества, если известно произвольное частичное упорядочение элементов.

А если $t > 2$? В упомянутой статье Кислицын пошел дальше. Он рассмотрел большие значения t, доказав, что

$$W_t(n) \le n - t + \sum_{n+1-t<j\le n} \lceil \log_2 j \rceil \qquad \text{при } n \ge t. \tag{6}$$

Мы видели, что при $t = 1$ и $t = 2$ эта формула представляет собой равенство; при $t = 3$ она может быть слегка улучшена (см. упр. 21).

Мы докажем теорему Кислицына, показав, что первые t стадий *выбора из дерева* требуют не более $n - t + \sum_{n+1-t<j\le n} \lceil \log_2 j \rceil$ сравнений (исключая все сравнения, содержащие $-\infty$). Интересно, что правая часть (6) равна $B(n)$, когда $t = n - 1$ и $t = n$ [см. формулу (5.3.1-3)]; следовательно, выбор из дерева и бинарные вставки приводят к одной и той же верхней оценке для задачи сортировки, хотя это совершенно различные методы.

Пусть α—расширенное бинарное дерево с n внешними узлами и π—перестановка множества $\{ 1, 2, \ldots, n \}$. Поместим элементы перестановки π во внешние узлы слева направо в симметричном порядке и заполним внутренние узлы в соответствии с правилами турнира с выбыванием как при выборе из дерева. Повторное применение операции выбора к результирующему дереву определяет последовательность $c_{n-1}c_{n-2}\ \ldots c_1$, где c_j есть число сравнений, требуемых, чтобы. перенести элемент j в корень дерева, после того как элемент $j + 1$ был заменен на $-\infty$. Например, если α—дерево

Picture: p.256

а $\pi = 5\ 3\ 1\ 4\ 2$,то мы получаем последовательные деревья

Picture: p. 257.1

Если же $\pi = 3\ 1\ 5\ 4\ 2$, то последовательность $c_4c_3c_2c_1$ будет иной, именно 2 1 1 0. Легко видеть, что c_1 всегда есть 0.

Пусть $\mu(\alpha, \pi)$—мультимножество $\{ c_{n-1}, c_{n-2}, \ \ldots, c_1 \}$, определяемое α и π. Если

Picture: p.257.2

и если элементы 1 и 2 не содержатся оба либо в α', либо в α'', то легко видеть, что

$$\mu(\alpha, \pi) = (\mu(\alpha', \pi') + 1) \uplus (\mu(\alpha'', \pi'') + 1) \uplus \{ 0 \} \tag{8}$$

для подходящих перестановок π' и π'', где $(\mu+1)$ обозначает мультимножество, получаемое прибавлением 1 к каждому элементу μ. С другой стороны, если и элемент 1, и элемент 2 находятся в α', имеем

$$\mu(\alpha, \pi) = (\mu(\alpha', \pi') + \varepsilon) \uplus (\mu(\alpha'', \pi'') + 1) \uplus \{ 0 \},$$

где $(\mu + \varepsilon)$ обозначает какое-нибудь мультимножество, получаемое прибавлением 1 к некоторым элементам μ и 0 к остальным. Аналогичная формула справедлива, если элементы 1 и 2 находятся в α''. Будем говорить, что мультимножество μ_1 мажорирует μ_2, если μ_1 и μ_2 содержат равное число элементов и k-й в порядке убывания элемент μ_1 больше или равен k-го в порядке убывания элемента μ_2 для всех k. Определим $\mu(\alpha)$ как мажоранту $\mu(\alpha, \pi)$ по всем перестановкам π в том смысле, что $\mu(\alpha)$ мажорирует $\mu(\alpha, \pi)$ при всех π и $\mu(\alpha) = \mu(\alpha, \pi)$ при некотором π. Приведенные выше формулы показывают, что

Picture: p.257.3

следовательно, $\mu(\alpha)$ *есть мультимножество всех расстояний от корня до внутренних узлов* α.

Если читатель уследил за ходом наших рассуждении, ему должно быть ясно, что теперь мы готовы доказать теорему Кислицына (6), поскольку $W_t(n)$ меньше или равно $n - 1$ плюс $t - 1$ наибольших элементов $\mu(\alpha)$, где α—любое дерево, используемое при сортировке посредством выбора из дерева. Можно выбрать в качестве α полное бинарное дерево с n внешними узлами (см. п. 2.3.4.5); в этом случае

$$\begin{aligned} \mu(\alpha) &= \{ \lfloor \log_2 1 \rfloor, \lfloor \log_2 2 \rfloor, \ldots, \lfloor \log_2 (n-1) \rfloor \} = \\ &= \{ \lceil \log_2 2 \rceil - 1, \lceil \log_2 3 \rceil - 1, \ldots, \lceil \log_2 n \rceil - 1 \}. \end{aligned} \tag{10}$$

Мы получим формулу (6), если рассмотрим $t-1$ наибольших элементов этого мультимножества.

Теорема Кислицына дает хорошую верхнюю оценку для $W_t(n)$; Кислицын отметил, что $V_3(5) = 6 < W_3(5) = 7$, но не смог найти в общем случае лучшую оценку для $V_t(n)$. Это было сделано А. Адьяном и М. Собелем, которые использовали *выбор с замещением* вместо выбора из дерева (см. п. 5.4.1). Выведенная ими формула [Univ. of Minnesota, Dept. of statistics, report 121 (May, 1969)]

$$V_t(n) \le n - t + (t-1)\lceil \log_2(n+2-t) \rceil, \qquad n \ge t, \tag{11}$$

отличается от (6) тем, что каждый элемент суммы в (6) заменен на наименьший элемент.

Теорему Адьяна и Собеля можно доказать, воспользовавшись следующим построением. Сначала образуем бинарное дерево для турнира с выбыванием $n-t+2$ элементов. (Это требует $n-t+1$ сравнений.) Наибольший элемент превосходит $n-t+1$ других элементов, поэтому он не может быть t-м в порядке убывания. Заменим его во внешнем узле дерева на один из $t-2$ элементов, оставшихся в резерве, и найдем наибольший из получившихся $n-t+2$ элементов; это требует не более $\lceil \log_2(n+2-t) \rceil$ сравнений, поскольку придется заново вычислить только один путь в дереве. Повторим эту операцию всего $t-2$ раз, по одному разу для каждого элемента из резерва. Наконец, заменим текущий наибольший элемент на $-\infty$ и определим наибольший из оставшихся $n+1-t$ элементов; для этого потребуется не более $\lceil \log_2(n+2-t) \rceil - 1$ сравнений, и t-й в порядке убывания элемент исходного множества попадет в корень дерева. Суммирование сравнений дает (11).

Разумеется, мы должны заменить t на $n+1-t$ в правой части соотношения (11), если $n+1-t$ дает лучшее значение (как при $n = 6$, $t = 3$). Как ни странно, но эта формула дает для $V_7(13)$ меньшую оценку, чем для $V_6(13)$. Верхняя оценка в (11) точна для $n \le 6$, но когда n и t становятся большими, можно получить значительно лучшие оценки для $V_t(n)$.

Например, можно использовать следующий изящный метод (принадлежащий Дэвиду Дорену), чтобы показать, что $V_4(8) \le 12$. Обозначим элементы через $X_1, \ldots, X_8$; сначала сравним $X_1 : X_2$, $X_3 : X_4$ и двух победителей, проделаем то же для $X_5 : X_6$, $X_7 : X_8$, и их победителей. Переименуем элементы так, чтобы получить $X_1 < X_2 < X_4 > X_3$, $X_5 < X_6 < X_8 > X_7$, затем сравним $X_2 : X_6$; в силу симметрии положим $X < 2 < X_6$, поэтому имеем конфигурацию

Picture: p.259.1

(Теперь X_1 и X_8 вышли из игры, и мы должны найти третий в порядке убывания элемент из $\{ X_2, \ldots, X_7 \}$.) Сравним $X_2 : X_7$, и отбросим меньший; в худшем случае получим $X_2 < X_7$, и нам надо найти третий в порядке убывания элемент из

Picture: p.259.2

Это можно сделать еще за $V_3(5) - 2 = 4$ шага, так как процедура для $t = 3$ и $n = 5$ в (11) начинается со сравнения двух непересекающихся пар элементов.

Можно использовать другие трюки подобного вида, чтобы получить результаты, показанные в табл. 1; до сих пор не видно никакого общего метода, и значения в таблице для $n \ge 8$ могут претерпеть изменения. Заметим, что $V_3(10) \le 14$, поэтому (11) не всегда дает равенство для $t = 3$. Тот факт, что $V_4(7) = 10$, показывает, что соотношение (11) приводит к ошибке на 2 уже при $n = 7$.

Неплохую нижнюю оценку в задаче выбора получил Дэвид Киркпатрик (Ph. D. thesis, Univ. of Toronto, 1974]. Построенный им дьявол доказывает, что

$$V_t(n) > n + t - 3 + \sum_{0 \le j \le t-2} \lceil \log_2((n+2-t)/(t+j)) \rceil, \qquad n \ge 2t-1. \tag{12}$$

Киркпатрик точно установил поведение функции $V_t(n)$ при $t = 3$, доказав, что $V_3(n) = n + \lceil \log_2((n-1)/2.5) \rceil + \lceil \log_2((n-1)/4) \rceil$ при всех $n \ge 50$ (ср. с упр. 22).

Линейный метод. Если n нечетно и $t = \lceil n/2 \rceil$, то t-й в порядке убывания (и t-й в порядке возрастания) элемент называется медианой. В соответствии с (11) мы можем найти медиану n

Таблица 1

Наилучшие из известных верхних оценок для $V_t(n)$

n	$V_1(n)$	$V_2(n)$	$V_3(n)$	$V_4(n)$	$V_5(n)$	$V_6(n)$	$V_7(n)$	$V_8(n)$	$V_9(n)$	$V_{10}(n)$
1	0									
2	1	1								
3	2	3	2							
4	3	4	4	3						
5	4	6	6	6	4					
6	5	7	8	8	7	5				
7	6	8	10	10*	10	8	6			
8	7	9	11	12	12	11	9	7		
9	8	11	12	14	15*	14	12	11	8	
10	9	12	14*	15	17	17	15	14*	12	9

*) В этих случаях упр. 10–12 дают построения, позволяющие улучшить (11).

элементов за $\approx \frac{1}{2} n \log_2 n$ сравнений, но это лишь приблизительно вдвое быстрее сортировки, хотя нам нужна значительно меньшая информация. В течение нескольких лет объединенные усилия ряда исследователей были направлены на улучшение формулы (11) для больших значений t и n; наконец, в 1971 г. Мануэль Блум открыл метод, требующий только $O(n \log\log n)$ шагов. Подход Блума к этой задаче дал толчок к развитию нового класса методов, который привел к следующему построению, принадлежащему Р. Райвесту и Р. Тарьяну.

Теорема L. *Если $n > 32$, то $V_t(n) \le 15n - 163$ при $1 \le t \le n$.*

Доказательство Когда n мало, теорема тривиальна, так как $V_t(n) \le S(n) \le 10n \le 15n - 163$ для $32 < n \le 2^{10}$. Добавив самое большее 13 фиктивных элементов "$-\infty$", можно считать, что $n = 7(2q+1)$ при некотором целом $q \ge 73$. Теперь для выбора t-го в порядке убывания элемента воспользуемся следующим методом.

Шаг 1. Разобьем элементы на $2q+1$ групп по 7 элементов в каждой и отсортируем каждую группу. Это потребует не более $13(2q+1)$ сравнений.

Шаг 2. Найдем медиану из $2q+1$ медиан, полученных на шаге 1, и обозначим ее x. Проведя индукцию по q, замечаем, что это требует не более $V_{q+1}(2q+1) \le 30q - 148$ сравнений.

Шаг 3. Теперь $n-1$ элементов, отличных от x, разбиваются на три множества (рис. 41):

$4q+3$ элементов, о которых известно, что они больше x (область B);
$4q+3$ элементов, о которых известно, что они меньше x (область C);
$6q$ элементов, отношение которых к x неизвестно (области A, D).

Выполнив дополнительно $4q$ сравнений, мы можем в точности сказать, какие элементы из областей A и D меньше x. (Сначала мы сравниваем x со средним элементом каждой тройки.)

Picture: Рис. 41. Алгоритм выбора Райвеста и Тарьяна ($q = 4$).

Шаг 4. Теперь при некотором r мы нашли r элементов, больших x, и $n-1-r$ элементов, меньших x. Если $t = r+1$, то x и будет ответом; если $t < r+1$, то нам нужно найти t-й элемент в порядке убывания из r бóльших элементов; и если $t > r+1$, то нам нужно найти $(t-1-r)$-й элемент в порядке убывания из $n-1-r$ меньших элементов. Суть дела в том, что r и $n-1-r$ оба меньше или равны $10q+3$ (размер областей A и D плюс B или C). Индукцией по q выводим, что этот шаг требует не более $15(10q+3) - 163$ сравнений.

Общее число сравнений оказывается не больше

$$13(2q+1) + 30q - 148 + 4q + 15(10q+3) - 163 = 15(14q-6) - 163.$$

Так как мы начали с не менее $14q-6$ элементов, доказательство завершено. ∎

Метод, использованный в этом доказательстве, не вполне совершенный, поскольку на шаге 4 теряется значительная информация. Тщательные улучшения, проделанные В. Праттом, Р. Райвестом и Р. Тарьяном, показывают, что константу 15 можно уменьшить до 5.43.

Среднее число. Вместо минимизации *максимального* числа сравнений можно искать алгоритм, который минимизирует *среднее* число сравнений, предполагая, что порядок случаен. Как обычно, эта задача значительно труднее, и она все еще не решена даже в случае $t = 2$. Клод Пикар упомянул эту задачу в своей

Picture: Рис. 42. Процедура, которая выбирает второй элемент из $\{ X_1, X_2, X_3, X_4, X_5, X_6 \}$, используя в среднем $6\frac{1}{2}$ сравнений. Каждая "симметричная" ветвь идентична своему брату, однако имена переставлены соответствующим образом. Во внешних узлах записано $j\ k$, если известно, что X_j—второй, а X_k—наибольший элемент; число перестановок, приводящих к этому узлу, записано непосредственно под ним.

книге Théorie des Questionnaires (1965), широкое исследование было предпринято Милтоном Собелем [Univ. of Minnesota, Dept. of Statistics, report 113 (November, 1968)].

Собель построил процедуру, изображенную на рис. 42, которая находит второй в порядке убывания элемент из шести элементов, в среднем используя только $6\frac{1}{2}$ сравнений. В худшем случае требуется 8 сравнений, и это хуже, чем $V_2(6) = 7$; но все известные процедуры для этой задачи, требующие не более 7 сравнений, используют в среднем по крайней мере $6\frac{2}{3}$ сравнений. Таким образом, вероятно, никакая процедура нахождения второго из шести элементов не будет оптимальной одновременно и как минимаксная, и как минимизирующая среднее число сравнений.

Пусть $\bar{V}(n)$ обозначает минимальное среднее число сравнений, необходимых для определения t-го элемента в порядке убывания из n элементов. В следующей таблице показаны наилучшие известные верхние оценки для $\bar{V}_2(n)$, вычисленные Собелем:

$$\begin{array}{rccccccccc} n = & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \bar{V}_2(n) \le & 1 & 2\frac{2}{2} & 4 & 5\frac{4}{4} & 6\frac{1}{1} & 7\frac{17}{17} & 9 & 10\frac{1}{1} & 11\frac{4}{4} \end{array} \tag{13}$$

Собель предположил, что

$$\bar{V}_2(n) \ge n - 2 + \lfloor 2 \log_2 n \rfloor / 2. \tag{14}$$

Р. У. Флойд в 1970 г. обнаружил, что медиана n элементов в среднем может быть найдена всего за $\frac{3}{2}n + O\left(n^{\frac{2}{3}} \log n\right)$ сравнений. (См. упр. 13.) Фактически он доказал, что

$$\bar{V}_t(n) \le n + t + f(n), \qquad \text{где } \lim_{n\to\infty} f(n)/n = 0. \tag{15}$$

Предполагается, что этот результат является наилучшей асимптотической формулой, однако никакой удовлетворительной нижней оценки все еще не найдено.

Упражнения

1. [15] Почему в турнире Льюиса Кэррола (рис. 39 и 40) игрок 13 выбывает, несмотря на то что он выиграл свой матч в третьем туре?
>2. [M25] Докажите, что после того, как мы нашли с помощью последовательности сравнений t-й элемент в порядке убывания из n элементов, мы также знаем, какие $t - 1$ элементов больше него и какие $n - t$ элементов меньше.
3. [M21] Докажите, что $V_t(n) \ge V_t(n-1) + 1$ при $1 \le t \le n$.
4. [M20] Докажите, что $W_t(n) \ge \left\lceil \log_2 n^{\underline{t}} \right\rceil$, где $n^{\underline{t}} = n(n-1)\ldots(n+1-t)$.
5. [10] Докажите, что $W_3(n) \le V_3(n) + 1$.
>6. [M26] (Р. У. Флойд.) Дано n различных элементов $\{ X_1, \ldots, X_n \}$ и отношения $X_i < X_j$ для некоторых пар (i, j). Мы хотим найти второй в порядке убывания элемент. Если известно, что $X_i < X_j$ и $X_i < X_k$ при $j \ne k$, то X_i не может быть вторым элементом, поэтому его можно исключить. В результате отношения будут иметь вид

Picture: p.264

а именно образуется m групп элементов, которые можно представить вектором $(l_1, l_2, \ldots, l_m)$; j-я группа содержит $l_j + 1$ элементов, про один из которых известно, что он больше остальных. Например, изображенная конфигурация может быть описана вектором $(3, 2, 4, 0, 1)$; если ни одного отношения не известно, то имеем вектор из n нулей.

Пусть $f(l_1, l_2, \ldots, l_m)$—минимальное число сравнений, нужных для определения второго элемента такого частично упорядоченного множества. Докажите, что

$$f(l_1, l_2, \ldots, l_m) = m - 2 + \left\lceil \log_2(2^{l_1} + 2^{l_2} + \cdots + 2^{l_m}) \right\rceil.$$

[*Указание.* Покажите, что наилучшая стратегий всегда состоит в том, чтобы сравнивать наибольшие элементы двух самых маленьких групп, пока не сведем от к единице; используйте индукцию по $l_1 + l_2 + \cdots + l_m + 2m$.]

7. [M20] Докажите (8).
8. [M21] Формула Кислицына (6) основана на сортировке посредством выбора из дерева, использующей полное бинарное дерево с n внешними узлами. Может ли выбор из дерева, основанный на некотором *другом* дереве, дать лучшую оценку для каких-нибудь t и n?

>9. [20] Нарисуйте дерево сравнений для нахождения медианы пяти элементов не более чем за шесть шагов, используя метод выбора с замещением Адьяна и Собеля [см. (11)].

10. [35] Покажите, что медиана семи элементов может быть найдена не более чем за 10 шагов.
11. [28] (Адьян и Собель.) Расширив метод Дорена, сформулированный в тексте, покажите, что медиана девяти элементов может быть найдена не более чем за 15 шагов.
12. [21] (Адьян и Собель.) Докажите, что $V_3(n) \le V_3(n-1) + 2$. [*Указание:* начните с удаления наименьшего из $\{ X_1, X_2, X_3, X_4 \}$.]

>13. [BM28] (Р. У. Флойд.) Покажите, что если начать с нахождения медианы $\{ X_1, \ldots, X_{n^{2/3}} \}$, используя рекурсивно определенный метод, то можно найти медиану $\{ X_1, \ldots, X_n \}$, выполнив в среднем $\frac{3}{2}n + O(n^{2/3} \log n)$ сравнений.

>14. [20] (М. Собель.) Покажите, что, использовав не более пяти сравнений, можно найти два наибольшие элемента из $\{ X_1, X_2, X_3, X_4, X_5 \}$, если не важен их взаимный порядок.

15. [22] (И. Пол.) Предположим, что нас интересует минимизация пространства, а не времени. Какое минимальное число слов памяти требуется для вычисления t-го из n элементов, если каждый элемент занимает одно слово и элементы вводятся в особый регистр по одному?

>16. [25] (И. Пол.) Покажите, что мы можем найти одновременно максимум и минимум множества из n элементов, использовав не более $\lceil \frac{3}{2}n \rceil - 2$ сравнений, и это число не может быть уменьшено. [*Указание.* Любая стадия такого алгоритма может быть представлена четверкой (a, b, c, d), где a элементов вообще не сравнивались, b элементов выигрывали, но никогда не проигрывали, c проигрывали, но никогда не выигрывали, d как выигрывали, так и проигрывали. Постройте подходящего дьявола.]

17. [20] (Р. У. Флойд.) Покажите, что можно выбрать k наибольших и l наименьших элементов множества из n элементов, использовав не более $\lceil \frac{3}{2}n \rceil - k - l + \sum_{n+1-k<j\le n} \lceil \log_2 j \rceil + \sum_{n+1-l<j\le n} \lceil \log_2 j \rceil$ сравнений.
18. [M20] Если бы в доказательстве теоремы L были использованы группы размера 5, а не 7, то какая бы получилась теорема?
19. [M44] Найдите точное значение $\bar{V}_2(6)$. Может ли оно достигаться в процедуре, никогда не выполняющей более семи сравнений?
20. [M47] Докажите (или опровергните) предположение Собеля (13).
21. [25] (С. Лин.) Докажите, что $W_3(2^k + 2) \le 2^k + 2k$, если $k \ge 3$.
22. [24] (Дэвид Г. Киркпатрик.) Покажите, что в случае $4 \cdot 2^k < n - 1 < 5 \cdot 2^k$ верхняя оценка (11) для $V_3(n)$ может быть следующим образом уменьшена на 1: (i) Образуйте четыре "дерева с выбыванием" размера 2^k. (ii) Найдите минимальный из четырех максимумов и удалите все 2^k элементов соответствующего дерева. (iii) Используя накопленную информацию, постройте одно дерево с выбыванием размера $n - 1 - 2^k$. (iv) Продолжайте, как при доказательстве (11).
23. [M49] Каково асимптотическое значение $V_{\lceil n/2 \rceil}(n)$ при $n \to \infty$?
24. [M48] Каково асимптотическое значение $\bar{V}_{\lceil n/2 \rceil}(n)$ при $n \to \infty$?

Сети сортировки В настоящем пункте мы будем изучать класс методов сортировки, удовлетворяющих некоторому ограничению. Интерес к таким методам объясняется в основном приложениями и солидной теоретической основой. Это новое ограничение требует, чтобы *последовательность сравнений не зависела от предыстории*[1] в том смысле, что если мы сравниваем K_i и K_j, то последующие сравнения для случая $K_i < K_j$ в точности те же, что и для случая $K_i > K_j$, однако i и j меняются ролями. На рис. 43(a) изображено дерево сравнений, в котором это условие выполнено. (Заметим, что на каждом уровне производится одинаковое число сравнений, поэтому после m сравнений имеется 2^m результатов; так как $n!$ не является степенью 2, то некоторые сравнения будут излишними в том смысле, что одно из их поддеревьев никогда не встречается на практике. Иными словами, на некоторых ветвях дерева приходится выполнять больше сравнений, чем необходимо, чтобы сортировка была правильной на всех соответствующих ветвях.)

Так как каждый путь такого дерева сверху донизу определяет все дерево, то подобную схему сортировки проще изображать в виде *сети,* как на рис. 43(b). Прямоугольники в такой сети представляют

[1] В первой редакции книги автор называл такую последовательность однородной.—*Прим. ред.*

"компараторные модули", имеющие два входа (изображенные линиями, входящими в модуль сверху)

Picture: Рис. 43. Дерево сравнений, в котором не учитывается предыстория, (a) и соответствующая сеть (b).

и два выхода (изображенные линиями, выходящими вниз); левый выход есть меньший из двух входов, а правый выход—больший из них. Элемент K_1' в нижней части сети есть наименьший из $\{K_1, K_2, K_3, K_4\}$, K_2'—второй в порядке возрастания и т. д. Нетрудно доказать, что любая сеть сортировки соответствует дереву сравнений, обладающему свойством независимости от предыстории (в указанном выше смысле), и что любое такое дерево соответствует сети компараторных модулей.

Между прочим заметим, что с инженерной точки зрения компараторный модуль довольно легко изготовить. Предположим, например, что по линиям связи в модуль поступают двоичные числа по одному биту в единицу времени, начиная со старшего. Каждый компараторный модуль имеет три состояния и функционирует следующим образом:

Момент t			Момент $(t+1)$		
Состояние	Входы		Состояние	Выходы	
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	2	0	1
0	1	1	0	1	1
1	x	y	1	x	y
2	x	y	2	y	x

Первоначально все модули находятся в состоянии 0 и выдают 00. Модуль переходит в состояние 1 или 2, как только его входы станут различными. Числа, которые в момент времени t начали поступать сверху в сеть, соответствующую рис. 43(b), начнут в момент $t+3$ выводиться снизу в отсортированном

Picture: Рис. 44. Еще один способ представления сортировки последовательности $\langle 4, 1, 3, 2\rangle$ посредством сети, изображенной на рис. 43.

порядке, если включить соответствующий задерживающий элемент в линиях K_1' и K_4'.

Для разработки теории сетей сортировки удобно изображать их несколько иным способом (рис. 44). На этом рисунке числа поступают слева, а компараторные модули изображены вертикальными соединениями между двумя прямыми; каждый компаратор вызывает, если необходимо, перестановку своих входов таким образом, что после прохождения компаратора большее число оказывается на нижней линии. В правой части диаграммы все числа упорядочены сверху вниз.

Ранее, изучая оптимальную сортировку, мы уделяли основное внимание минимизации числа сравнений, почти (или совсем)

Picture: Рис. 45. Получение $(n+1)$-элементного сортировщика из n-элементного: (a)—вставка; (b)—выбор.

не учитывая перемещение данных или сложность структуры решений метода сортировки. В этом отношении сети сортировки имеют некоторое преимущество, так как данные могут храниться в n ячейках, а структура решений "прямолинейна"; нет необходимости запоминать результаты предыдущих сравнений—план неизменен и фиксирован заранее. Еще одним важным

Picture: Рис. 46. Сетевые аналоги элементарных схем внутренней сортировки, полученные многократным применением операции, представленной на рис. 45: (a)—простая вставка; (b)—метод пузырька.

преимуществом сетей сортировки является то, что часть операций можно совместить, если выполнять их одновременно (на подходящей машине). Например, пять шагов на рис. 43 и 44 сокращаются до трех, если допустить одновременные неперекрывающиеся сравнения, так как можно объединить первые два и следующие два шага; позднее в данном пункте мы используем это свойство сетей сортировки. Таким образом, сети сортировки могут быть очень полезны, хотя возможность построения эффективной сети сортировки n элементов при больших n вовсе не очевидна; возможно, мы обнаружим, что для поддержания однородной структуры решений требуется много дополнительных сравнений.

Имеется два простых способа построения сети сортировки. для $n+1$ элементов, если дана сеть для n элементов: с использованием либо принципа *вставки,* либо принципа *выбора.* На

Picture: Рис. 47. При параллельном выполнении операций простая вставка совпадает с методом пузырька!

рис. 45(a) показано, как $(n+1)$-й элемент может быть вставлен на нужное место после того, как первые n элементов отсортированы, а на рис. 45(b) показано, как можно выбрать наибольший элемент, прежде чем перейти к сортировке остальных. Многократное применение рис. 45(a) дает сетевой аналог простых вставок (алгоритм 5.2.1S), а многократное применение рис. 45(b) приводит к сетевому аналогу метода пузырька (алгоритм 5.2.2B). На рис. 46 изображены соответствующие сети для шести элементов. Интересно заметить, что если сжать каждую сеть, чтобы обеспечить одновременные операции, то оба метода сведутся к одной и той же "треугольной" процедуре с $(2n-3)$ стадиями (рис. 47).

Легко доказать, что сети, представленные на рис. 43 и 44, будут сортировать любое множество из четырех чисел, поскольку первые четыре компаратора направляют наименьший и наибольший элементы на положенные им места, а последний компаратор располагает в требуемом порядке остальные два элемента. Однако не всегда так легко сказать, будет ли данная сеть сортировать все возможные входные последовательности; например, сети

Picture: p.269

являются правильными четырехэлементиыми сетями сортировки, но доказательство их правильности нетривиально. Было бы достаточно проверить каждую n-элементную сеть на всех $n!$ перестановках n различных чисел, но фактически мы можем обойтись значительно меньшим количеством проверок.

Теорема Z. *(Принцип нулей и единиц.) Если сеть с n входами сортирует в неубывающем порядке все 2^n последовательностей из 0 и 1, то она будет сортировать в неубывающем порядке любую последовательность n чисел.*

Доказательство (Это частный случай теоремы Бурисиуса, упр. 5.3.1-12.) Если $f(x)$—любая монотонная функция, для которой $f(x) \le f(y)$ при $x \le y$, и если данная сеть преобразует $\langle x_1, \ldots, x_n\rangle$ в $\langle y_1, \ldots, y_n\rangle$, то, как нетрудно видеть, эта сеть преобразует $\langle f(x_1), \ldots, f(x_n)\rangle$ в $\langle f(y_1), \ldots, f(y_n)\rangle$. Если $y_i > y_{i+1}$ при некотором i, то рассмотрим монотонную функцию f, которая для всех чисел $< y_i$ принимает значение 0, а для всех чисел $\le y_1$—значение 1. Эта функция определяет последовательность нулей и единиц $\langle f(x_1), \ldots, f(x_n)\rangle$, которая не сортируется данной сетью. Следовательно, если все последовательности 0 и 1 поддаются сортировке, то будем иметь $y_i \le y_{i+1}$ для всех $1 \le i < n$. ∎

Принцип нулей и единиц довольно полезен для построения сетей сортировки. В качестве нетривиального примера получим обобщенный вариант "обменной сортировки со слиянием" Бэтчера (алгоритм 5.2.2M). Идея состоит в том, чтобы сортировать $m+n$ элементов, "сортируя первые m и последние n элементов независимо и затем применяя к результату (m,n)-*сеть слияния.* Построить (m,n)-сеть слияния можно по индукции:

a) Если $m=0$ или $n=0$, то сеть пустая. Если $m=n=1$, то сеть состоит из единственного компараторного модуля.

b) Если $mn>1$, то обозначим сливаемые последовательности $\langle x_1, \ldots, x_m\rangle$ и $\langle y_1, \ldots, y_n\rangle$. Сольем "нечетные последовательности" $\langle x_1, x_3, \ldots, x_{2\lceil m/2\rceil-1}\rangle$ и $\langle y_1, y_3, \ldots, y_{2\lceil n/2\rceil-1}\rangle$ и получим отсортированный результат $\langle v_1, v_2, \ldots, v_{\lceil m/2\rceil+\lceil n/2\rceil}\rangle$; сольем "четные последовательности" $\langle x_2, x_4, \ldots, x_{2\lfloor m/2\rfloor}\rangle$ и $\langle y_2, y_4, \ldots, y_{2\lfloor n/2\rfloor}\rangle$ и получим отсортированный результат $\langle w_1, w_2, \ldots, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}\rangle$. И наконец, применим операции сравнения-обмена

$$w_1 : v_2, w_2 : v_3, w_3 : v_4, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor} : v^* \tag{1}$$

к последовательности

$$\langle v_1, w_1, v_2, w_2, v_3, w_3, \ldots, v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}, w_{\lfloor m/2\rfloor+\lfloor n/2\rfloor}, v^*, v^{**}\rangle; \tag{2}$$

результат будет отсортирован. (!) (Здесь $v^* = v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+1}$ не существует, если m и n оба четные, и $v^{**} = v_{\lfloor m/2\rfloor+\lfloor n/2\rfloor+2}$ существует, лишь если m и n оба нечетные; общее число компараторных модулей, указанных в (1), равно $\lfloor (m+n)-1)/2\rfloor$.)

Назовем (m,n)-сеть слияния Бэтчера *четно-нечетным слиянием.* Построенное в соответствии с этими принципами $(4,7)$-слияние показано на рис. 48.

Picture: Рис. 48. Четно-нечетное слияние для $m=4$ и $n=7$.

Чтобы доказать, что эта очень странная процедура действительно работает при $mn>1$, воспользуемся принципом нулей и единиц и проверим ее на всех последовательностях 0 и 1. После начальных

m-сортировки и n-сортировки последовательность $\langle x_1,\ \ldots, x_m\rangle$ будет состоять из k нулей, за которыми следуют $m-k$ единиц, а последовательность $\langle y_1,\ \ldots, y_n\rangle$—из l нулей с последующими $n-l$ единицами при некоторых k и l. Следовательно, последовательность $\langle v_1, v_2,\ \ldots\rangle$ будет состоять из $\lceil k/2\rceil + \lceil l/2\rceil$ нулей с последующими единицами, а $\langle w_1, w_2,\ \ldots\rangle$—из $\lfloor k/2\rfloor + \lfloor l/2\rfloor$ нулей с последующими единицами. Решающим моментом доказательства является то, что

$$(\lceil k/2\rceil + \lceil l/2\rceil) - (\lfloor k/2\rfloor + \lfloor l/2\rfloor) = 0, 1 \text{ или } 2. \tag{3}$$

Если эта разность равна 0 или 1, то последовательность (2) уже упорядочена, а если она равна 2, то одна из операций сравнения-обмена в (1) ставит все на свои места. Доказательство завершено. (Заметим, что принцип нулей и единиц сводит $\binom{m+n}{m}$ случаев в задаче слияния всего лишь к $(m+1)(n+1)$, каждый из которых изображается двумя параметрами k и l.) Пусть $C(m,n)$—число компараторных модулей, используемых при четно-нечетном слиянии m и n элементов, не считая начальных m- и n-сортировок; имеем

$$C(m,n) = \begin{cases} mn, & \text{если } mn \le 1; \\ C(\lceil m/2\rceil, \lceil n/2\rceil) + C(\lfloor m/2\rfloor, \lfloor n/2\rfloor) + \lfloor (m+n-1)/2\rfloor, & \text{если } mn > 1. \end{cases} \tag{4}$$

В общем случае это не слишком простая функция от m и n, однако, заметив, что $C(1,n) = n$ и что

$$C(m+1, n+1) - C(m,n) == 1 + C(\lfloor m/2\rfloor + 1, \lfloor n/2\rfloor + 1) - C(\lfloor m/2\rfloor, \lfloor n/2\rfloor), \qquad \text{если } mn \ge 1,$$

мы можем вывести соотношение

$$C(m+1, n+1) - C(m,n) = t + 2 + \lfloor n/2^{t+1}\rfloor, \qquad \text{если } n \ge m \ge 1 \text{ и } t = \lfloor \log_2 m\rfloor. \tag{5}$$

Следовательно,

$$C(m, m+r) = B(m) + m + R_m(r) \qquad \text{при } m \ge 0,\ r \ge 0, \tag{6}$$

где $B(m)$ есть функция "бинарной вставки" $\sum_{1\le k\le m}\lceil \log_2 k\rceil$ из соотношения (5.3.1-3), а $R_m(r)$ обозначает сумму первых от членов ряда

$$\left\lfloor \frac{r+0}{1}\right\rfloor + \left\lfloor \frac{r+1}{2}\right\rfloor + \left\lfloor \frac{r+2}{4}\right\rfloor + \left\lfloor \frac{r+3}{4}\right\rfloor + \left\lfloor \frac{r+4}{8}\right\rfloor + \cdots + \left\lfloor \frac{r+j}{2^{\lfloor \log_2 j\rfloor+1}}\right\rfloor + \cdots . \tag{7}$$

Если же $r = 0$, получаем важный частный случай

$$C(m,m) = B(m) + m. \tag{8}$$

Кроме того, если $t = \lceil \log_2 m\rceil$, то

$$\begin{aligned} R_m(r + 2^t) &= R_m(r) + 1\cdot 2^{t-1} + 2\cdot 2^{t-2} + \cdots + 2^{t-1}\cdot 2^0 + m = \\ &= R_m(r) + m + t\cdot 2^{t-1}. \end{aligned}$$

Следовательно, $C(m, n+2^t) - C(m,n)$ имеет простой вид и

$$C(m,n) = \left(\frac{t}{2} + \frac{m}{2^t}\right) n + O(1) \qquad \text{при фиксированном } m,\ n\to\infty,\ t = \lceil \log_2 m\rceil; \tag{9}$$

член $O(1)$ становится в конце концов периодической функцией от n с длиной периода 2^t. Асимптотически при $n\to\infty$ величина $C(n,n)$ равна $n\log_2 n$ из (8) и упр. 5.3.1-15.

Сети с минимальным числом сравнений. Пусть $\hat S(n)$—минимальное число сравнений, требуемых в сети сортировки для n элементов; ясно, что $\hat S(n) \ge S(n)$, где $S(n)$—минимальное число сравнений, необходимое для сортировки без всяких ограничений (см. п. 5.3.1). Мы видели, что $\hat S(4) = 5 = S(4)$, поэтому новое ограничение не вызывает потери эффективности при $n = 4$; но уже при $n = 5$ оказывается, что $\hat S(5) = 9$, в то время как $S(5) = 7$. Задача определения $\hat S(n)$ кажется еще более трудной, чем задача определения $S(n)$; до сих пор неизвестно даже асимптотическое поведение $\hat S(n)$.

Интересно проследить историю этой задачи, так как на каждый новый шаг приходилось затрачивать определенные усилия. Сети сортировки были впервые исследованы П. Н. Армстронгом, Р. Дж. Нельсоном и Д. Дж. О'Коннором около 1954 г. [см. U. S. Patent 3029413]; они показали, что $\hat S(n+1) \le \hat S(n) + n$. Как сказано в их патентной заявке, "приложив старания, можно сконструировать экономичные n-входные сортирующие переключатели, используя уменьшенное число двухвходных сортирующих переключателей";

они привели примеры конструкций для $4 \le n \le 8$, использовав соответственно 5, 9, 12, 18 и 19 компараторов. Работая далее над этой задачей, Нельсон совместно с Р. Ч. Бозе еще до 1960 г. разработали общую процедуру для построения сетей сортировки, показывающую, что $\hat{S}(2^n) \le 3^n - 2^n$ при всех n, поэтому $\hat{S}(n) = O(n^{\log_2 3}) = O(n^{1.585})$. Бозе и Нельсон опубликовали свой интересный метод в *JACM,* **9** (1962), 282–296, высказав предположение, что это наилучший возможный результат; Т. Н. Хиббард [*JACM,* **10** (1963), 142–150] нашел аналогичный, но несколько более простой метод, в котором используется такое же число сравнений, подкрепив тем самым это предположение.

В 1964 г. Р. У. Флойд и Д. Э. Кнут использовали новый подход к этой задаче, приведший их к асимптотической оценке вида $\hat{S}(n) = O(n^{1+c/\sqrt{\log n}})$. Работая независимо, К. Э. Бэтчер открыл описанную выше общую стратегию слияния; используя число компараторов, определяемое как

$$c(1) = 0, c(n) = c(\lceil n/2 \rceil) + c(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil, \lfloor n/2 \rfloor) \qquad \text{при } n \ge 2, \tag{10}$$

он доказал, что (см. упр. 5.2.2-14)

$$c(2^t) = (t^2 - t + 4)2^{t-2} - 1,$$

и отсюда вывел, что $\hat{S}(n) = O(n(\log n)^2)$. Как Бэтчер, так и Флойд с Кнутом опубликовали свои конструкции лишь через некоторое время [*Notices of the Amer. Math. Soc.,* **14** (1967), 283; Proc. AFIPS Spring Joint Computer Conference, **32** (1968), 307–314].

Кое-кому удалось сократить число компараторов, используемых в конструкции слияния с обменами, предложенной Бэтчером; в следующей таблице показаны даилучшие из известных в настоящее время верхних оценок для $\hat{S}(n)$:

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$c(n) =$	0	1	3	5	9	12	16	19	26	31	37	41	48	53	59	63
$\hat{S}(n) \le$	0	1	3	5	9	12	16	19	25	29	35	39	46	51	56	60

(11)

Так как $\hat{S}(n) < c(n)$ при $8 < n \le 16$, то обменная сортировка со слиянием неоптимальна при всех $n > 8$. Если $n \le 8$, то такая сортировка эквивалентна по количеству компараторов конструкции Бозе и Нельсона. Флойд и Кнут доказали в 1964–1966 гг., что указанные значения $\hat{S}(n)$ *точны* при $n < 8$ [см. A Survey of Combinatorial Theory (North-Holland, 1973), 163–172]; значения $\hat{S}(n)$ при $n > 8$ до сих пор неизвестны.

Конструкции, приводящие к указанным выше значениям для $\hat{S}(n)$, изображены на рис. 49. Сеть при $n = 9$, основанная на интересном трехпутевом слиянии, была найдена Р. У. Флойдом в 1964 г.; установить ее справедливость можно при помощи общего принципа, описанного в упр. 27. Сеть при $n = 10$ в 1969 г. построил А. Ваксман; он рассматривал входы как перестановки множества $\{1, 2, \ldots, 10\}$ и пытался, сохраняя некоторую симметрию, насколько возможно уменьшить число значений, которые могут появляться в каждой строке на данной стадии.

В 1969 г. Г. Шапиро нашел сеть сортировки 16 элементов с 62 компараторами, и это было весьма неожиданно, поскольку метод Бэтчера (63 сравнения), казалось, использует все возможности, если n является степенью 2. М. У. Грин вскоре после того, как он ознакомился с конструкцией Шапиро, поверг всех в еще большее изумление, найдя сортировку с 60 сравнениями, показанную на рис. 49. Первая часть конструкции Грина довольно проста для понимания; после того как выполнены 32 операции сравнения-обмена слева от пунктирной линии, все прямые можно так пометить 16 подмножествами $\{a, b, c, d\}$, чтобы про прямую, помеченную s, было известно, что она содержит числа, меньшие или равные содержимому прямой, помеченной t, всякий раз, когда s есть подмножество t. Состояние сортировки в этот момент обсуждается более подробно в упр. 32. Однако сравнения, выполняемые на последующих уровнях, становятся совершенно загадочными, и до сих пор никто не знает, как обобщить эту конструкцию, чтобы получить столь же эффективные сети для больших значений n.

Шапиро и Грин открыли также изображенную на рис. 49 сеть для $n = 12$. Хорошие сети для $n = 11$, 13, 14 и 15 можно получить, удалив нижнюю прямую сети для $n + 1$ вместе со всеми компараторами, подсоединенными к этой линии.

Picture: Рис. 49. Эффективные сети сортировки.

Наилучшие известные к настоящему моменту сети для $n \to \infty$ см. в докторской диссертации Д. Ван Воориса (Stanford University, 1971); его сети требуют асимптотически $\frac{1}{4}n(\log_2 n)^2 - \alpha n \log_2 n$ компараторов, где $\alpha = \frac{1}{4} + \frac{1}{6}\sum_{k\ge 0} 2^{-2^k-k} \approx 0.356\,852$.

Сети с минимальным временем. В физических реализациях сетей сортировки и на параллельных ЭВМ можно выполнять непересекающиеся операции сравнения-обмена одновременно, поэтому кажется естественным попытаться минимизировать время задержки. По некотором размышлении заключаем, что

время задержки сети сортировки равно максимальному числу компараторов, прилегающих к какому-либо "пути" через сеть, если определить путь как траекторию любого движения слева направо,

Picture: Рис. 50. Выполнение каждого сравнения в наиболее ранний из возможных моментов.

возможно, с переходом с одной прямой на другую через компараторы. У каждого компаратора мы можем поставить порядковый номер, указывающий самый ранний момент, когда может быть выполнено сравнение; этот номер на единицу больше, чем максимальный номер у компараторов, предшествующих данному. (См. рис. 50(a); в части (b) этого рисунка показана та же сеть, перерисованная так, чтобы каждое сравнение выполнялось в наиболее ранний возможный момент.)

В описанной выше сети Бэтчера для четно-нечетного слияния затрачивается $T_b(m,n)$ единиц времени, где $T_b(m,0) = T_b(0,n) = 0$, $T_B(1,1) = 1$ и

$$T_B(m,n) = 1 + \max(T_B(\lfloor m/2\rfloor, \lfloor n/2\rfloor), T_B(\lceil m/2\rceil, \lceil n/2\rceil)) \qquad \text{для } mn \geq 2.$$

Используя эти соотношения, можно доказать по индукции, что $T_B(m,n+1) \geq T_B(m,n)$; следовательно, $T_B(m,n) = 1 + T_B(\lceil m/2\rceil, \lceil n/2\rceil)$ для $mn \geq 2$, а отсюда заключаем, что

$$T_B(m,n) = 1 + \lceil \log_2 \max(m,n)\rceil \qquad \text{для } mn \geq 1. \tag{12}$$

Таким образом, как показано в упр. 5, метод сортировки Бэтчера имеет время задержки

$$\binom{1 + \lceil \log_2 n\rceil}{2}. \tag{13}$$

Пусть $\hat{T}(n)$—минимальное время задержки, достижимое в любой сети сортировки n элементов. Некоторые из описанных выше сетей можно улучшить, не использовав дополнительных компараторов, так, чтобы они имели меньшее время задержки, как

Picture: Рис. 51. Сети сортировки, которые необыкновенно быстры, если сравнения выполняются параллельно.

показано на рис. 51 для $n = 6$ и $n = 9$, а в упр. 7—для $n = 10$. Можно получить еще меньшее время задержки, если добавить один или два дополнительных модуля, как показывают сети для $n = 10$, 12 и 16 на рис. 51. Эти построения приводят к следую- щим верхним оценкам для $T(n)$ при умеренных значениях n:

$n =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\hat{T}(n) \leq$	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9

(14)

Известно, что приведенные здесь значения точны при $n \leq 8$ (см. упр. 4). Сети на рис. 51 заслуживают тщательного изучения, поскольку вовсе не очевидно, что они годятся для сортировки; эти сети были открыты в 1969–1971 гг. Г. Шапиро ($n = 6$, 9, 12) и Д. Ван Ворисом ($n = 10$, 16).

Сети слияния. Пусть $\hat{M}(m,n)$ обозначает минимальное число компараторов, необходимых для сети, которая сливает m элементов $x_1 \leq \ldots \leq x_m$ с n элементами $y_1 \leq \ldots \leq y_n$, образуя отсортированную последовательность $z_1 \leq \ldots \leq z_{m+n}$. К настоящему времени не открыто ни одной сети слияния, которая была бы лучше четно-нечетного слияния, описанного выше; следовательно, функция $C(m,n)$ в (6) представляет наилучшую известную верхнюю оценку для $\hat{M}(m,n)$.

Р. У. Флойд обнаружил интересный способ, позволяющий определить *нижние* оценки в этой задаче слияния.

Теорема F. *При всех $n \geq 1$ справедливо неравенство $\hat{M}(2n,2n) \geq 2\hat{M}(n,n) + n$.*

Доказательство Рассмотрим сеть с $\hat{M}(2n,2n)$ компараторными модулями, способную сортировать все входные последовательности $\langle z_1, \ldots, z_{4n}\rangle$, такие, что $z_1 \leq z_3 \leq \ldots \leq z_{4n-1}$ и $z_2 \leq z_4 \leq \ldots \leq z_{4n}$. Мы можем считать, что каждый модуль заменяет (z_i, z_j) на $(\min(z_i,z_j), \max(z_i,z_j))$ при некоторых $i < j$ (упр. 16). Итак, компараторы можно разделить на три класса:

a) $i \leq 2n$ и $j \leq 2n$;
b) $i > 2n$ и $j > 2n$;
c) $i \leq 2n$ и $j > 2n$.

Класс (a) должен содержать по крайней мере $\hat{M}(n,n)$ компараторов, так как z_{2n+1}, z_{2n+2}, ..., z_{4n} могут уже находиться на своих местах, когда слияние начинается; аналогично в классе (b) должно быть по

крайней мере $\hat{M}(n,n)$ компараторов. Кроме того, как показывает входная последовательность $\langle 0,1,0,1,\ \ldots,0,1\rangle$, класс (c) содержит не менее n компараторов, так как n нулей должны переместиться из $\{\, z_{2n+1},\ \ldots,z_{4n}\,\}$ в $\{\, z_1,\ \ldots,z_{2n}\,\}$. ∎

Многократное применение теоремы F доказывает, что $\hat{M}(2^m,2^m)\ge\frac{1}{2}(m+2)2^m$; следовательно, $\hat{M}(n,n)\ge\frac{1}{2}n\log_2 n+O(n)$. Слияние *без* сетевого ограничения требует лишь $M(n,n)=2n-1$ сравнений; таким образом, мы доказали, что сетевое слияние сложнее по существу, чем слияние вообще. Четно-нечетное слияние показывает, что $\hat{M}(n,n)\le C(n,n)=n\log_2 n+O(n)$, поэтому асимптотическое поведение $\hat{M}(n,n)$ известно с точностью до множителя 2. (Точные значения $\hat{M}(n,n)$ известны для $n\le 5$; см. упр. 9.) А. К. Яо и Ф. Ф. Яо доказали, что $\hat{M}(2,n)=C(2,n)=\lceil\frac{3}{2}n\rceil$ и $\hat{M}(m,n)\ge\frac{1}{2}n\log_2(m+1)$ при $m<n$ [*JACM*, будет опубликовано].

Битонная сортировка. Если допустимы одновременные сравнения, то, как видно из формулы 12, при четно-нечетном слиянии для $1\le m\le n$ возникает задержка на $\lceil\log_2(2n)\rceil$ единиц времени. Бэтчер нашел другой тип сети слияния, называемой *битонным сортировщикам,* для которого время задержки снижается до $\lceil\log_2(m+n)\rceil$, но он требует больше компараторных модулей.

Последовательность $\langle z_1,\ \ldots,z_p\rangle$ из p чисел будем называть *битонной,* если $z_1\ge\ldots\ge z_k\le\ldots\le z_p$ для некоторого k, $1\le k\le p$ (сравните это с обычным определением "монотонных" последовательностей). Битонный сортировщик порядка p—это компараторная сеть, способная сортировать в неубывающем порядке любую битонную последовательность длины p. Задача слияния $x_1\le\ldots\le x_m$ с $y_1\le\ldots\le y_n$ является частным случаем задачи битонной сортировки, так как слияние можно осуществить, применив к последовательности $\langle x_m,\ \ldots,x_1,y_1,\ \ldots,\ y_n\rangle$ битонный сортировщик порядка $m+n$.

Заметим, что если последовательность $\langle z_1,\ \ldots,z_p\rangle$ битонная, то таковыми же являются и все ее подпоследовательности. Вскоре после того, как Бэтчер открыл сети четно-нечетного слияния, он обнаружил, что аналогичным образом можно построить битонный сортировщик порядка p, сначала независимо сортируя битонные подпоследовательности $\langle z_1,z_3,z_5,\ \ldots\rangle$ и $\langle z_2,z_4,z_6,\ \ldots\rangle$, а затем выполняя сравнения-обмены $z_1:z_2,\ z_3:z_4,\ \ldots$. (Доказательство см. в упр. 10.) Если соответствующее число компараторных модулей обозначить через $C'(p)$, то будем иметь

$$C'(p)=C'(\lceil p/2\rceil)+C'(\lfloor p/2\rfloor)+\lfloor p/2\rfloor\qquad\text{при } p\ge 2. \tag{15}$$

а время задержки, очевидно, равно $\lceil\log_2 p\rceil$. На рис. 52 показан битонный сортировщик порядка 7, построенный этим способом; он может быть использован и как $(3,4)$, и как $(2,5)$-сеть слияния с задержкой в три единицы; четно-нечетное слияние для $m=2$ и $n=5$ имеет на один компаратор меньше, но на один уровень задержки больше.

Битонный сортировщик Бэтчера порядка 2^k особенно интересен; он состоит из k уровней по 2^{k-1} компараторов в каждом. Занумеруем входные прямые $z_0,\ z_1,\ \ldots,\ z_{2^k-1}$; при этом элемент 2, сравнивается с z_j на уровне l тогда и только тогда, когда двоичные представления i и j различаются только в l-м бите слева. Эта простая структура приводит к параллельной сети сортировки, которая так же быстра, как обменная сортировка

Picture: Рис. 52. Битонный сортировщик Бэтчера порядка 7.

со слиянием (алгоритм 5.2.2M), но значительно проще для реализации. (См. упр. 11 и 13.)

Если $m=n$, то нетрудно видеть, что и четно-нечетное слияние, и битонный сортировщик Бэтчера обеспечивают абсолютный минимум времени задержки, достижимого в любой сети слияния.

Picture: Рис. 53. Один элемент сливается с шестью другими с разветвлением, чтобы достичь минимально возможного времени задержки.

В (n,n)-сети слияния n-й по величине выход (считая от наименьшего) должен зависеть от всех $2n$ входов, и если его можно вычислить за l шагов, то он будет зависеть не более, чем от 2^l входов; поэтому $2^l\ge 2n$, $l\ge\lceil\log_2(2n)\rceil$.

Если $m<n$, то n-й выход (m,n)-сети слияния зависит от $2m+1$ входов (ср. с упр. 29), поэтому те же рассуждения дают в этом случае минимальное время задержки для слияния $\lceil\log_2(2m+l)\rceil$. Бэтчер показал [Report GER-14122 (Akron, Ohio: Goodyear Aerospace Corporation, 1968)], что это минимальное время задержки достигается, если в сети допускается разветвление, т. е. такое разбиение линий, что одно и то же число в одно и то же время используется несколькими модулями. В качестве примера на рис. 53 изображена (для $n=6$) сеть, которая сливает один элемент с n другими всего с двумя уровнями задержки. Конечно, сети с разветвлением не соответствуют нашим соглашениям; довольно легко понять, что любая $(1,n)$-сеть слияния без разветвления должна иметь время задержки $\log_2(n+1)$ или более. (См. упр. 14.)

Сети выбора. Сети можно применить также и к задаче п. 5.3.3. Пусть $\hat{U}_t(n)$ обозначает минимальное число компараторов, необходимых в сети, которая перемещает t наибольших из n различных входов на t определенных выходных прямых, они могут располагаться на этих выходных прямых в произвольном порядке. Пусть $\hat{V}_t(n)$ обозначает минимальное количество компараторов, нужное для перемещения $t-$го в порядке убывания из n различных входов на определенную выходную прямую, и пусть $\hat{W}_t(n)$ обозначает минимальное число компараторов, требуемых для перемещения t наибольших из n различных входов на определенные t выходных прямых в неубывающем порядке. Нетрудно видеть (см. упр. 17), что

$$\hat{U}_t(n) \le \hat{V}_t(n) \le \hat{W}_t(n). \tag{16}$$

Сначала предположим, что имеется $2t$ элементов $\langle x_1, \ldots, x_{2t}\rangle$ и мы хотим выбрать t наибольших; В. Е. Алексеев [*Кибернетика,* **5**, 5 (1969), 99–103] заметил, что это может быть выполнено, если сначала отсортировать $\langle x_1, \ldots, x_t\rangle$ и $\langle x_{t+1}, \ldots, x_{2t}\rangle$, а затем сравнить и поменять местами

$$x_1 : x_{2t}, x_2 : x_{2t-1}, \ldots, x_t : x_{t+1}. \tag{17}$$

Так как ни в одной из этих пар не может содержаться более одного из наибольших t элементов (почему?), то процедура Алексеева должна выбрать t наибольших элементов.

Если нам нужно выбрать t наибольших из nt элементов, то мы можем применить эту процедуру $n-1$ раз (исключая каждый раз t элементов); следовательно,

$$\hat{U}_t(nt) \le (n-1)(2\hat{S}(t)+t). \tag{18}$$

Алексеев также получил интересную *нижнюю* оценку для задачи выбора.

Теорема А. $\hat{U}_t(n) \ge (n-t)\lceil\log_2(t+1)\rceil$.

Доказательство Удобнее рассматривать эквивалентную задачу выбора *наименьших* t элементов. Около каждой прямой компараторной сети можно выписать числа (l, u), как показано на рис. 54, где l и u обозначают соответственно минимальное и

Picture: Рис. 54. Отделение четырех наибольших от четырех наименьших. (Числа над прямыми используются в доказательстве теоремы А.)

максимальное значения, которые могут появиться в этом месте, если входом служит перестановка $\{1, 2, \ldots, n\}$. Пусть l_i и l_j—нижние оценки на прямых i и j перед сравнением $x_i : x_j$, и пусть l_i' и l_j'—соответствующие нижние оценки после этого

Picture: Рис. 55. Иная интерпретация сети, изображенной на рис. 54.

сравнения. Очевидно, что $l_i' = \min(l_i, l_j)$, а в упр. 24 доказывается (неочевидное) соотношение

$$l_j' \le l_i + l_j. \tag{19}$$

Теперь дадим другую интерпретацию действия сети (рис. 55); предположим, что на всех входных прямых содержится нуль, а каждый "компаратор" помещает теперь на верхнюю прямую меньший из его входов, а на нижнюю прямую—больший вход *плюс один.* Получающиеся числа $\langle m_1, m_2, \ldots, m_n\rangle$ обладают свойством

$$2^{m_i} \ge l_i \tag{20}$$

в любом месте сети, так как это свойство первоначально справедливо и сохраняется каждым компаратором в силу (19). Кроме того, окончательное значение

$$m_1 + m_2 + \cdots + m_n$$

равно общему числу компараторов в сети, так как каждый компаратор добавляет к этой сумме единицу.

Если сеть выбирает наименьшие t чисел, то $n-t$ из чисел l_i больше или равны $t+1$; следовательно, $n-t$ из чисел m_i должны быть $\ge \lceil\log_2(t+1)\rceil$. ∎

Нижняя оценка в теореме А оказывается точной, если $t = 1$ или $t = 2$ (см. упр. 19). В табл. 1 даны значения $\hat{U}_t(n)$, $\hat{V}_t(n)$ и $\hat{W}_t(n)$ для небольших t и n.

Таблица 1

Сравнения, необходимые для сетей выбора ($\hat{U}_t(n)$, $\hat{V}_t(n)$, $\hat{W}_t(n)$)

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
$n=1$	$(0,0,0)$					
$n=2$	$(1,1,1)$	$(0,1,1)$				
$n=3$	$(2,2,2)$	$(2,3,3)$	$(0,2,3)$			
$n=4$	$(3,3,3)$	$(4,5,5)$	$(3,5,5)$	$(0,3,5)$		
$n=5$	$(4,4,4)$	$(6,7,7)$	$(6,7,8)$	$(4,7,9)$	$(0,4,9)$	
$n=6$	$(5,5,5)$	$(8,9,9)$	$(8,10,10)$	$(8,10,12)$	$(5,9,12)$	$(0,5,12)$

Упражнения

(ПЕРВАЯ ЧАСТЬ) Далее в нескольких упражнениях дано более глубокое развитие теории сетей сортировки, поэтому будет удобно ввести некоторые обозначения. Вместо модуля сравнения-обмена будем писать $[i:j]$. Сеть с n входами и r компараторными модулями запишем как $[i_1:j_1]\,[i_2:j_2]\ldots[i_r:j_r]$, где все i и j меньше или равны n; для краткости будем называть ее n-сетью. Сеть называется *стандартной,* если $i_q < j_q$ для $1 \le q \le r$. Так, например, рис. 44 описывает стандартную 4-сеть, обозначаемую последовательностью компараторов $[1:2]\,[3:4]\,[1:3]\,[2:4]\,[2:3]$.

Наши соглашения в тексте об изображении диаграмм сетей позволяют рисовать только стандартные сети; все компараторы $[i:j]$ изображаются прямой от i к j, где $i<j$. Если нужно нарисовать нестандартную сеть, то можно использовать *стрелку* от i к j, указывающую, что большее число направляется к острию стрелки. Например, на рис. 56 изображена нестандартная сеть для 16 элементов с компараторами $[1:2]\,[4:3]\,[5:6]\,[8:7]$ и т. д. В упр. 11 доказывается, что это сеть сортировки. Если $x = \langle x_1, \ldots, x_n\rangle$ есть n-вектор, а α есть n-сеть, то используем обозначение $x\alpha$ для вектора чисел $\langle (x\alpha)_1, \ldots, (x\alpha)_n\rangle$, порожденных сетью. Положим также для краткости $a \vee b = \max(a,b)$, $a \wedge b = \min(a,b)$, $\bar{a} = 1-a$; тогда $(x[i:j])_i = x_i \wedge x_j$, $(x[i:j])_j = x_i \vee x_j$ и $(x[i:j])_k = x_k$ для $i \ne k \ne j$. Будем говорить, что α является *сетью сортировки,* тогда и только тогда, когда $(x\alpha)_i \le (x\alpha)_{i+1}$ для $1 \le i < n$ и всех x.

Символ $e^{(i)}$ обозначает вектор, у которого в позиции i находится 1, а в остальных местах 0; таким образом, $(e^{(i)})_j = \delta_{ij}$. Символ D_n обозначает множество всех 2^n n-местных векторов из 0 и 1, а P_n обозначает множество всех $n!$ векторов, являющихся перестановками $\{1, 2, \ldots, n\}$. Мы будем использовать обозначения $x \wedge y$ и $x \vee y$ для векторов $\langle x_1 \wedge y_1, \ldots, x_n \wedge y_n\rangle$ и $\langle x_1 \vee y_1, \ldots, x_n \vee y_n\rangle$ и будем писать $x \le y$, если $x_i \le y_i$ при всех i. Таким образом, $x \le y$ тогда и только тогда, когда $x \vee y = y$, и тогда и только тогда, когда

Picture: Рис. 56. Нестандартная сеть, основанная на битонной сортировке.

$x \wedge y = x$. Если x и y лежат в D_n, то будем говорить, что x *покрывает* y, если $x = y \vee e^{(i)} \ne y$ при некотором i. Наконец, для всех x в D_n пусть $\nu(x)$ будет числом единиц в x, а $\zeta(x)$—числом нулей; таким образом, $\nu(x) + \zeta(x) = n$.

1. [20] Нарисуйте сеть четно-нечетного слияния для $m=3$ и $n=5$.
2. [22] Покажите, что алгоритму сортировки В. Пратта (см. упр. 5.2.1-30) соответствует сеть сортировки n элементов, имеющая приблизительно $(\log_2 n) \times (\log_3 n)$ уровней задержки. Нарисуйте такую сеть для $n = 12$.
3. [M20] (К. Э. Бэтчер.) Найдите простое соотношение между $C(m, m-1)$ и $C(m,m)$.
>4. [M23] Докажите, что $\hat{T}(6) = 5$.
5. [M21] Докажите, что выражение (13) действительно определяет время задержки для сети сортировки, описанной соотношениями (10).
6. [28] Пусть $T(n)$ будет минимальным числом стадий, требуемых для сортировки с *одновременным выполнением непересекающихся сравнений* (без сетевого ограничения); каждое такое множество сравнений может быть представлено узлом, содержащим множество пар $i_1:j_1$, $i_2:j_2$, $\ldots$, $i_r:j_r$, где все i_1, j_1, i_2, j_2, $\ldots$, i_r, j_r различны; от этого узла отходит вниз 2^r ветвей, соответствующих случаям

$$\langle K_{i_1} < K_{j_1}, K_{i_2} < K_{j_2}, \ldots, K_{i_r} < K_{j_r}\rangle;$$
$$\langle K_{i_1} > K_{j_1}, K_{i_2} < K_{j_2}, \ldots, K_{i_r} < K_{j_r}\rangle \text{ и т. д.}$$

Докажите, что $T(5) = T(6) = 5$.

7. [25] Покажите, что если последний компаратор сети для $n = 10$ на рис. 49 поместить непосредственно перед вторым и третьим с конца компараторами, то сеть по-прежнему будет сортировать.

8. [M20] Докажите, что $\hat{M}(m_1+m_2, n_1+n_2) \geq \hat{M}(m_1, n_1)+\hat{M}(m_2, n_2)+\min(m_1, n_2)$ при $m_1, m_2, n_1, n2 \geq 0$.
9. [M25] (Р. У. Флойд.) Докажите, что $\hat{M}(3,3) = 6$, $\hat{M}(4,4) = 9$, $\hat{M}(5,5) = 13$.
10. [M22] Докажите, что битонный сортировщик Бэтчера, как он определен в тексте перед (15), действительно работает. [*Указание.* Достаточно доказать, что будут сортироваться все последовательности, состоящие из k единиц, за которыми следуют l нулей, за которыми следуют $n-k-l$ единиц.]
11. [M23] Докажите, что битонный сортировщик Бэтчера порядка 2^p будет сортировать не только последовательности $\langle z_0, z_1, \ldots, z_{2^p-1}\rangle$, для которых $z_0 \geq \ldots \geq z_k \leq \ldots \leq z_{2^p-1}$, но также и все последовательности, для которых $z_0 \leq \ldots \leq z_k \geq \ldots \geq z_{2^p-1}$. [Как следствие этого, сеть на рис. 56 будет сортировать 16 элементов, так как каждая стадия состоит из битонных сортировщиков или обращенных битонных сортировщиков, применяемых к последовательностям, которые были отсортированы в противоположных направлениях.]
12. [M20] Докажите или опровергните следующее утверждение: если x и y—битонные последовательности равной длины, то последовательности $x \vee y$ и $x \wedge y$ также битонные.
>13. [24] (Х. С. Стоун). Покажите, что сеть сортировки для 2^t элементов можно построить по схеме, проиллюстрированной для $t = 4$ на рис. 57. Каждый из t^2 шагов этой схемы состоит из "идеального тасования" первых 2^{t-1} элементов с последними 2^{t-1}, за которым следуют операции, выполняемые одновременно над 2^{t-1} парами соседних элементов. Каждая из этих операций обозначена либо "0" (нет операции), либо "+" (стандартный компараторный модуль), либо "−" (обращенный компараторный модуль). Сортировка протекает в t стадий по t шагов каждая; на последней стадии все операции суть "+". В течение стадии s при $s < t$ мы выполняем $t-s$ шагов, где все операции суть "0", а затем выполняем s шагов, где на каждом $q-$м шаге поочередно выполняются 2^{q-1} операций "+" и затем 2^{q-1} операций "−" при $q = 1, 2, \ldots, s$.

[Заметим, что эта схема сортировки может быть выполнена весьма простым устройством, реализующим один шаг "тасования и операций" и передающим выход обратно на вход. Первые три шага на рис. 57 можно, конечно, устранить, они оставлены, лишь чтобы сделать схему понятнее. Стоун замечает, что тот же принцип "тасования/операции" встречается в некоторых других алгоритмах, таких, как быстрое преобразование Фурье (см. п. 4.6.4).]

14. [M20] Докажите, что любая $(1, n)$-сеть слияния без разветвления должна иметь не менее $\lceil \log_2(n+1) \rceil$ уровней задержки.
15. [20] Найдите нестандартную сеть сортировки четырех элементов, содержащую только пять компараторных модулей.
16. [M22] Докажите, что следующий алгоритм преобразует любую сеть сортировки $[i_1 : j_1] \cdots [i_r : j_r]$ в стандартную сеть сортировки:

    T1. Пусть q— наименьший индекс, такой, что $i_q > j_q$. Если таких индексов нет, то остановиться.
    T2. Заменить все вхождения i_q на j_q и все вхождения j_q на i_q во всех компараторах $[i_s : j_s]$ для $q \leq s \leq r$. Вернуться к шагу T1.

Например, сеть $[4:1]\,[3:2]\,[1:3]\,[2:4]\,[1:2]\,[3:4]$ преобразуется сначала в $[1:4]\,[3:2]\,[4:3]\,[2:1]\,[4:2]\,[3:1]$, затем в $[1:4]\,[2:3]\,[4:2]\,[3:1]\,[4:3]\,[2:1]$, затем

Picture: Рис. 57. Сортировка 16 элементов с "идеальным тасованием".

в $[1:4]\,[2:3]\,[2:4]\,[3:1]\,[2:3]\,[4:1]$ и т. д., пока не получится стандартная сеть $[1:4]\,[2:3]\,[2:4]\,[1:3]\,[1:2]\,[3:4]$.

17. [M25] Пусть D_{tn} будет множеством всех $\binom{n}{t}$ последовательностей $\langle x_1, \ldots, x_n\rangle$ из нулей и единиц, имеющих ровно t единиц. Докажите, что $\hat{U}_t(n)$ равно минимальному числу компараторов, которые необходимы в сети, сортирующей все элементы D_{tn}; что $\hat{V}_t(n)$ равно минимальному числу компараторов, нужных для сортировки $D_{tn} \cup D_{(t-1)n}$; и что $\hat{W}_t(n)$ равно минимальному числу компараторов, нужных для сортировки $\bigcup_{0\leq k\leq t} D_{kn}$.
>18. [M20] Докажите, что сеть, которая определяет медиану $2t-1$ элементов, требует не менее $(t-1)\lceil \log_2(t+1)\rceil + \lceil \log_2 t\rceil$ компараторных модулей. [*Указание:* см. доказательство теоремы A.]
19. [M22] Докажите, что $\hat{U}_2(n) = 2n-4$ и $\hat{V}_2(n) = 2n-3$ для всех $n \geq 2$.
20. [24] Докажите, что $\hat{V}_3(5) = 7$.
21. [M15] Пусть α—любая n-сеть, а x и y—два n-вектора. Докажите, что из $x \leq y$ следует $x\alpha \leq y\alpha$.
22. [M15] Докажите, что если x и y суть n-векторы действительных чисел, то $x \cdots y \leq (x\alpha)\cdot(y\alpha)$. (Здесь $x \cdot y$—скалярное произведение $x_1y_1 + \cdots + x_ny_n$.)
23. [M17]. Пусть α есть n-сеть. Докажите, что существует перестановка $p \in P_n$, такая, что $(p\alpha)_i = j$ тогда и только тогда, когда в D_n найдутся векторы x, y, такие, что x покрывает y, $(x\alpha)_i = 1$, $(y\alpha)_i = 0$ и $\zeta(y) = j$.

>24. [M21] (В. Е. Алексеев.) Пусть α есть n-сеть; введем обозначения $l_k = \min\{\,(p\alpha)_k \mid p \in P_n\,\}$, $u_k = \max\{\,(p\alpha)_k \mid p \in P_n\,\}$ при $1 \le k \le n$ для нижней и верхней границ диапазона значений, которые могут появляться на прямой k выхода. Пусть l'_k и u'_k—аналогично определенные величины для сети $\alpha' = \alpha[i:j]$. Докажите, что $l'_i = l_i \wedge l_j$, $l'_j \le l_i + l_j$, $u'_i \ge u_+ u_j - (n+1)$, $u'_j = u_i \vee u_j$. [*Указание:* для данных векторов x и y из D_n, таких, что $(x\alpha)_i = (y\alpha)_j = 0$, $\zeta(x) = l_i$, $\zeta(y) = l_j$, найдите вектор z из D_n, такой, что $(z\alpha')_j = 0$, $\zeta(z) \le l_i + l_j$.]

25. [M30] Пусть l_k и u_k определены, как в упр. 24. Докажите, что множество $\{\,(p\alpha)_k \mid p \in P_n\,\}$ содержит все целые числа между l_k и u_k включительно.

26. [M24] (Р. У. Флойд.) Пусть α есть n-сеть. Докажите, что множество $D_n\alpha = \{\,x\alpha \mid x \in D_n\,\}$ может быть определено, из множества $P_n\alpha = \{\,p\alpha \mid p \in P_n\,\}$ и, обратно, $P_n\alpha$ может быть определено из $D_n\alpha$.

>27. [M20] Пусть x и y—векторы, и пусть $x\alpha$ и $y\alpha$—отсортированные векторы. Докажите, что $(x\alpha)_i \le (y\alpha)_j$ тогда и только тогда, когда для любой совокупности j элементов из y можно найти совокупность i элементов из x, такую, что любой элемент, взятый из x, меньше некоторого элемента, взятого из y, или равен ему. Используйте этот принцип для доказательства того, что *если отсортировать строки любой матрицы, а затем отсортировать столбцы, то строки останутся упорядоченными.*

>28. [M20] Следующая диаграмма показывает, как записать формулы для содержимого всех линий сети сортировки через ее входы:

Picture: p.287

Используя законы коммутативности $x \wedge y = y \wedge x$, $x \vee y = y \vee x$, законы ассоциативности $x \wedge (y \wedge z) = (x \wedge y) \wedge z$, $x \vee (y \vee z) = (x \vee y) \vee z$, законы дистрибутивности $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$, $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$, законы поглощения $x \wedge (x \vee y) = x \vee (x \wedge y) = x$ и законы идемпотентности $x \wedge x = x \vee x = x$, мы можем свести формулы в правой части этой сети соответственно к $(a \wedge b \wedge c \wedge d)$, $(a \wedge b \wedge c) \vee (a \wedge b \wedge d) \vee (a \wedge c \wedge d) \vee (b \wedge c \wedge d)$, $(a \wedge b) \vee (a \wedge c) \vee (a \wedge d) \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$, $a \vee b \vee c \vee d$. Докажите, что в общем случае t-й в порядке убывания элемент из $\{\,x_1, \ldots, x_n\,\}$ дается "элементарной симметрической функцией"

$$\sigma_t(x_1, \ldots, x_n) = \bigvee \{\, x_{i_1} \wedge x_{i_2} \wedge \ldots \wedge x_{i_t} \mid 1 \le i_1 < i_2 < \ldots < i_t \le n \,\}.$$

[Здесь $\binom{n}{t}$ членов объединяются операцией $\vee$ вместе. Таким образом, задача нахождения сети сортировки минимальной стоимости эквивалентна задаче вычисления элементарных симметрических функций с минимальным числом схем "и/или", где на каждом шаге две величины ϕ и ψ заменяются на $\phi \wedge \psi$ и $\phi \vee \psi$.]

29. [M20] Дано, что $x_1 \le x_2 \le x_3$ и $y_1 \le y_2 \le y_3 \le y_4 \le y_5$ и что $z_1 \le z_2 \le \ldots \le z_8$—результат слияния x с y. Найдите выражения для каждого z через x и y, используя операторы $\wedge$ и $\vee$.

30. [BM24] Докажите, что любая формула, содержащая $\wedge$, $\vee$ и независимые переменные $\{\,x_1, \ldots, x_n\,\}$, может быть приведена с использованием тождеств из упр. 28 к "канонической" форме $\tau_1 \vee \tau_2 \vee \ldots \vee \tau_k$, здесь $k \ge 1$ и каждый τ_i имеет вид $\wedge\{\,x_j \mid j \in S_i\,\}$, где S_i—подмножество $\{\,1, 2, \ldots, n\,\}$ и никакое множество S_i не включается в S_j, если $i \ne j$. Докажите также, что две такие канонические формы равны для всех $x_1, \ldots, x_n$ тогда и только тогда, когда они идентичны (с точностью до порядка).

31. [M24] (Р. Дедекинд, 1897.) Пусть δ_n—число различных канонических форм от $x_1, \ldots, x_n$ в смысле упр. 30 Так, $\delta_1 = l$, $\delta_2 = 4$ и $\delta_3 = 18$. Чему равно δ_4?

32. [M28] (М. У. Грин.) Пусть $G_1 = \{\,00, 01, 11\,\}$; определим G_{n+1} как множество всех цепочек $\theta\phi\psi\omega$, таких, что θ, ϕ, ψ, ω имеют длину 2^{n+1} и $\theta\phi$, $\psi\omega$, $\theta\psi$ и $\phi\omega$ принадлежат G_n. Пусть α—сеть, состоящая из четырех первых уровней 16-сортировщика, изображенного на рис. 48. Покажите, что $D_{16}\alpha = G_4$, и докажите, что это множество имеет в точности $\delta_4 + 2$ элементов. (См. упр. 31.)

>33. [M22] Не все δ_n функций от $\langle x_1, \ldots, x_n\rangle$ из упр. 31 могут встретиться в компараторных сетях. А именно докажите, что функция $(x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_4)$ не может быть результатом никакой компараторной сети от $\langle x_1, \ldots, x_n\rangle$.

34. [23] Является ли следующая сеть сетью сортировки?

Picture: p.288

34. [20] Докажите, что в любой *стандартной сети* сортировки должен по крайней мере один раз встретиться каждый из компараторов $[i : i+1]$ при $1 \le i < n$.

>36. [22] Сеть на рис. 47 содержит только *кратчайшие* сравнения $[i : i+1]$; будем называть такие сети *примитивными,* (a) Докажите, что примитиваая сеть сортировки для n элементов должна иметь не менее $\binom{n}{2}$ компараторов. [*Указание:* рассмотрите инверсии перестановки.] (b) (Р. У. Флойд, 1964.)

Пусть α—примитивная сеть для n элементов, а x—вектор, такой, что $(x\alpha)_i > (x\alpha)_j$ при некоторых $i < j$. Докажите, что $(y\alpha)_i > (y\alpha)_j$, где y—вектор $\langle n, n-1, \ldots, 1\rangle$. (c) В качестве следствия (b) докажите, что примитивная сеть является сетью сортировки тогда и только тогда, когда она сортирует единственный вектор $\langle n, n-1, \ldots, 1\rangle$!

37. [M22] *Четно-нечетная сортировка с транспозициями* для n чисел, $n \geq 3$, это n-уровневая сеть с $\frac{1}{2}n(n-1)$ компараторами, напоминающая кирпичную кладку (рис. 58). (Если n четно, имеются две возможности.) Такую сортировку особенно легко реализовать аппаратурно, так как попеременно выполняются только два вида действий. Докажите, что такая сеть действительно будет правильной сетью сортировки. [*Указание:* см. упр. 36.]

Picture: Рис. 58. Четно-нечетная сортировка с транспозициями.

38. [29] Можно дать другую интерпретацию сетям сортировки, считая, что на каждой линии находится мультимножество из m чисел, а не одно число; при этой интерпретации операция $[i : j]$ заменяет x_i и x_j соответственно на $x_i \operatorname{upup} x_j$ и $x_i \operatorname{dndn} x_j$—наименьшие m и наибольшие m из $2m$ чисел $x_i \uplus x_j$. (Рис. 59 иллюстрирует это при $m = 2$.) Если a и b суть мультимножества, содержащие m чисел каждое, то будем говорите, что $a \operatorname{lfff} b$ тогда

Picture: Рис. 59. Другая интерпретация сети сортировки, представленной на рис. 44: каждый компараторный модуль выполняет операцию слияния.

и только тогда, когда $a \operatorname{upup} b = a$ (или, эквивалентно, $a \operatorname{dndn} b = b$; наибольший элемент a меньше или равен наименьшему элементу b). Таким образом, $a \operatorname{upup} b \operatorname{lfff} a \operatorname{dndn} b$.

Пусть α есть n-сеть, а $x = \langle x_1, \ldots, x_n\rangle$—вектор, в котором каждая компонента x_i—мультимножество из m элементов. Докажите, что если $(x\alpha)_i$ не $\operatorname{lfff}(x\alpha)_j$ в описанной интерпретации, то в D_n найдется вектор y, такой, что $(y\alpha)_i = 1$ и $(y\alpha)_j = 0$. [Следовательно, сеть сортировки n элементов превращается в сеть сортировки mn элементов, если заменить сравнения m-путевыми слияниями. На рис. 60 изображен восьмиэлементный сортировщик, построенный из четырехэлементного с использованием этого наблюдения.]

>39. [M23] Покажите, что в обозначениях упр. 38 $(x \operatorname{upup} y) \operatorname{upup} z = x \operatorname{upup}(y \operatorname{upup} z)$ и $(x \operatorname{dndn} y) \operatorname{dndn} z = x \operatorname{dndn}(y \operatorname{dndn} z)$, однако $(x \operatorname{dndn} y) \operatorname{upup} z$) *не* всегда равно $(x \operatorname{upup} z) \operatorname{dndn}(y \operatorname{upup} z)$ и $(x \operatorname{upup} y) \operatorname{dndn}(x \operatorname{upup} z) \operatorname{dndn}(y \operatorname{u}$ *не* всегда равно средним m элементам $x \uplus y \uplus z$. Найдите правильную формулу для этих средних элементов, использовав в ней x, y, z, а также операции upup и dndn.

40. [M25] (Р. Л. Грэхем.) Компаратор $[i : j]$ называется избыточным в сети $\alpha_1[i : j]\alpha_2$, если либо $(x\alpha_1)_i \leq (x\alpha_1)_j$ для всех векторов x, либо $(x\alpha_1)_i \geq (x\alpha_1)_j$ для всех векторов x. Докажите, что если α является сетью с r неизбыточными компараторами, то найдутся по крайней мере r различных упорядоченных

Picture: Рис. 60. 8-сортировщик, построенный из 4-сортировщика с использованием слияния.

пар (i, j) различных индексов, таких, что $(x\alpha)_i \leq (x\alpha)_j$ для всех векторов x. (Следовательно, сеть без избыточных компараторов содержит не более $\binom{n}{2}$ модулей.)

>41. [M27] (В. Е. Алексеев.) Пусть $\alpha = [i_1 : j_1] \ldots [i_r : j_r]$ есть n-сеть; для $1 \leq s \leq r$ определим $\alpha^s = [i'_1 : j'_1] \ldots [i'_{s-1} : j'_{s-1}] [i_s : j_s] \ldots [i_r : j_r]$, где i'_k и j'_k получены из i_k и j_k заменой i_s на j_s и j_s на i_s везде, где они встречаются. Например, если $\alpha = [1 : 2] [3 : 4] [1 : 3] [2 : 4] [2 : 3]$, то $\alpha^4 = [1 : 4] [3 : 2] [1 : 3] [2 : 4] [2 : 3]$.

a) Докажите, что $D_n\alpha = D_n(\alpha^s)$.
b) Докажите, что $(\alpha^s)^t = (\alpha^t)^s$.
c) *Сопряжением* α является любая сеть вида $(\ldots((\alpha^{s_1})^{s_2})\ldots)^{s_k}$. Докажите, что α имеет не более 2^{r-1} сопряжений.
d) Пусть $g_\alpha(x) = 1$, если $x \in D_n\alpha$, и $g_\alpha(x) = 0$, если $x \notin D_n\alpha$, и пусть

$$f_\alpha(x) = (\bar{x}_{i_1} \vee x_{j_1}) \wedge \ldots \wedge (\bar{x}_{i_r} \vee x_{j_r}).$$

Докажите, что $g_\alpha(x) = \bigvee\{ f_{\alpha'}(x) \mid \alpha' \text{ есть сопряжение } \alpha \}$.
e) Пусть G_α—направленный граф с вершинами $\{1, \ldots, n\}$ и дугами $i_s \to j_s$ для $1 \leq s \leq r$. Докажите, что а является сетью сортировки тогда и только тогда, когда для всех ее сопряжений α' в $G_{\alpha'}$ имеется ориентированный путь от i до $i + 1$ для $1 \leq i < n$. [Это довольно интересное условие, поскольку G_α не зависит от порядка компараторов в α.]

>42. [25] (Д. Ван Ворис.) Докажите, что $\hat{S}(n) \geq \hat{S}(n-1) + \lceil \log_2 n \rceil$.

43. [23] *Перестановочной сетью* называется последовательность модулей $[i_1 : j_1] \ldots [i_r : j_r]$, где каждый модуль $[i : j]$ может устанавливаться извне в одно из двух состояний: либо он передает свои входы без изменений, либо меняет местами x_i и x_j (независимо от значений x_i и x_j), и последовательность модулей должна быть такой, что на выходе можно получить любую перестановку входов при соответствующей установке модулей. Любая сеть сортировки является, очевидно, перестановочной сетью, но обратное неверно. Найдите перестановочную сеть для пяти элементов, имеющую только восемь модулей.
44. [46] Изучите свойства сетей сортировки, построенных из m-сортировщиков вместо 2-сортировщиков. (Например, Г. Шапиро построил сеть для сортировки 16 элементов, использовав четырнадцать 4-сортировщиков. Наилучшее ли это решение? Существует ли для всех m эффективный способ сортировки m^2 элементов с помощью модулей, выполняющих m-сортировку?)
45. [48] Найдите, (m, n)-сеть слияния с числом компараторов, меньшим $C(m, n)$, или докажите, что такой сети не существует.
46. [48] Найдите (m, n)-сеть слияния меньше, чем с $\lceil \log_2(m + n) \rceil$ уровнями задержки, или докажите, что ее не существует.
47. [48] Изучите класс схем сортировки, которые могут быть реализованы в виде схем с идеальным тасованием, как на рис. 57, но с другим расположением операций "0", "+" и "−".
48. [BM49] Исследуйте свойства операций upup и dndn, определенных в упр. 38. Можно ли охарактеризовать все тождества в этой алгебре каким-либо изящным способом или вывести все их из конечного набора тождеств? В этом отношении такие тождества, как

$$x \operatorname{upup} x \operatorname{upup} x = x \operatorname{upup} x \text{ или } x \operatorname{upup}(x \operatorname{dndn}(x \operatorname{upup}(x \operatorname{dndn} y))) = x \operatorname{upup}(x \operatorname{dndn} y),$$

которые имеют место только для $m \leq 2$, представляют относительно небольшой интерес; рассматривайте лишь тождества, справедливые при *всех* m.
49. [M49] Каково асимптотическое поведение функции $T(n)$, определенной в упр. 6? Может ли быть $T(n) < \hat{T}(n)$ при каком-нибудь n?
50. [50] Найдите точное значение $\hat{S}(n)$ для какого-либо $n > 8$.
51. [M50] Докажите, что асимптотическое значение $\hat{S}(n)$ не есть $O(n \log n)$.

УПРАЖНЕНИЯ, (ВТОРАЯ ЧАСТЬ)

Следующие упражнения имеют дело с различными типами оптимальных задач, касающихся сортировки. Первые несколько задач основаны на "интересном "многоголовочном" обобщении метода пузырька, предложенном Ф. Н. Армстронгом и Р. Дж. Нельсоном еще в 1954 г. [См. U. S. Patents 3029413, 3034102.] Пусть $1 = h_1 < h_2 < \ldots < h_m = n$—возрастающая последовательность целых чисел; будем называть ее "последовательностью головок" длины m с диапазоном n. Она будет использоваться при определении методов сортировки специального вида. Сортировка записей $R_1 \ldots R_N$ осуществляется за несколько проходов, а каждый проход состоит из $N + n - 1$ шагов. На шаге j при $j = 1 - n, 2 - n, \ldots, N - 1$ рассматриваются записи $R_{j+h[1]}, R_{j+h[2]}, \ldots, R_{j+h[m]}$, которые в случае необходимости переставляются так, чтобы их ключи оказались упорядоченными. (Мы говорим, что $R_{j+h[1]} \ldots R_{j+h[m]}$ находятся "под головками чтения-записи". Если $j + h[k]$ либо < 1, либо $> N$, то запись $R_{j+h[k]}$ не рассматривается, иначе говоря, ключи K_0, K_{-1}, K_{-2}, ... считаются равными $-\infty$, а K_{N+1}, K_{N+2}, ...—равными $+\infty$. Поэтому при $j \leq -h[m-1]$ или $j > N - h[2]$ шаг j тривиален.)

Например, в следующей таблице показан один проход сортировки при $m = 3$, $N = 9$ и $h_1 = 1$, $h_2 = 2$, $h_3 = 4$:

	K_{-2}	K_{-1}	K_0	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}
$j = -3$	_	_		3	1	4	5	9	2	6	8	7			
$j = -2$		_	_	3	1	4	5	9	2	6	8	7			
$j = -1$			_	3	1	4	5	9	2	6	8	7			
$j = 0$				1	3	4	5	9	2	6	8	7			
$j = 1$				1	3	4	5	9	2	6	8	7			
$j = 2$				1	3	2	4	9	5	6	8	7			
$j = 3$				1	3	2	4	6	5	9	8	7			
$j = 4$				1	3	2	4	5	6	9	8	7			
$j = 5$				1	3	2	4	5	6	7	8	9			
$j = 6$				1	3	2	4	5	6	7	8	9	_		
$j = 7$				1	3	2	4	5	6	7	8	9		_	
$j = 8$				1	3	2	4	5	6	7	8	9	_		_

Заметим, что, если $m = 2$, $h_1 = 1$ и $h_2 = 2$, этот "многоголовочный" метод сводится к методу пузырька (алгоритм 5.2.2B).

52. [21] (Джеймс Дугунди.) Докажите, что если $h[k+1] = h[k] + 1$ при некотором k, $1 \le k < m$, то многоголовочный сортировщик, определенный выше, отсортирует любой входной файл за конечное число проходов. Но если $h[k+1] \ge h[k] + 2$ при $1 \le k < m$, то может случиться, что входная последовательность *никогда* не станет упорядоченной.

>53. [50] (Армстронг и Нельсон.) Пусть $h[k+1] \le h[k] + k$ при $1 \le k \le m$ и $N \ge n - 1$. Докажите, что в течение первого прохода наибольшие $n-1$ элементов всегда займут свои окончательные места. [*Указание:* используйте принцип нулей и единиц; докажите, что если сортируется последовательность из нулей и единиц, причем единиц меньше n, то все головки могут читать 1 лишь в том случае, когда все нули лежат слева от головок.]

Докажите, что если головки удовлетворяют сформулированным условиям, то сортировка будет закончена не более, чем за $\lceil (N-1)/(n-1) \rceil$ проходов. Существует ли входной файл, для которого необходимо ровно столько проходов?

54. [26] Докажите, что при $n = N$ первый проход поместит наименьший ключ в позицию R_1 тогда и только тогда, когда $h[k+1] \le 2h[k]$, $1 \le k < m$.

55. [34] (Дж. Хопкрофт.) "Совершенным сортировщиком" N элементов называется многоголовочный сортировщик, который всегда заканчивает работу за один проход. Упражнение 53 доказывает, что последовательность

$$\langle h_1, h_2, h_3, h_4, \ \ldots, h_m \rangle = \left\langle 1, 2, 4, 7, \ \ldots, 1 + \binom{m}{2} \right\rangle .$$

образует совершенный сортировщик для $N = \binom{m}{2}$ элементов, используя $m = (\sqrt{8N-7}+l)/2$ головок. Например, последовательность головок $\langle 1, 2, 4, 7, 11, 16, 22 \rangle$ является совершенным сортировщиком для 22 элементов.

Докажите, что последовательность головок $\langle 1, 2, 4, 7, 11, 16, 23 \rangle$ на самом деле будет совершенным сортировщиком для 23 элементов.

56. [49] Определите при заданном m наибольшее N, для которого существует совершенный сортировщик с m головками. Верно ли, что $N = O(m^2)$?

57. [23] (В. Пратт.) Если каждая головка h_k находится в положении 2^{k-1} для $1 \le k \le m$, то сколько проходов потребуется для сортировки последовательности нулей и единиц $z_1\, z_2 \ldots z_{2^m-1}$, где $z_j = 0$ тогда и только тогда, когда j является степенью 2?

58. [24] (Однородная сортировка.) В дереве на рис. 34 в п. 5.3.1 сравнение $2:3$ выполняется в обоих ветвях уровня 1; а в каждой ветви уровня 2 выполняется сравнение $1:3$, если только оно не является избыточным. В общем случае мы можем рассмотреть класс алгоритмов сортировки, однородных именно в этом смысле, предполагая, что $M = \binom{N}{2}$ пар $\{ (a,b) \mid 1 \le a < b \le N \}$ выстроены в последовательность

$$(a_1, b_2)\,(a_2, b_2) \ldots (a_M, b_M);$$

мы можем последовательно выполнять те из сравнений $K_{a_1} : K_{b_1}$, $K_{a_2} : K_{b_2}$, ..., результат которых еще не известен. Каждая из $M!$ расстановок пар (a,b) определяет алгоритм однородной сортировки. Идея однородной сортировки принадлежит Х. Л. Бьюсу [*JACM*,**17** (1970), 482–495], в работе которого были предложены ближайшие несколько упражнений.

Для формального определения однородной сортировки удобно воспользоваться теорией графов. Пусть G — направленный граф с вершинами $\{ 1, 2, \ldots, N \}$ и без дуг. Для $i = 1, 2, \ldots, M$ мы добавляем дуги к G следующим образом:

Случай 1. В G имеется путь от a_i к b_i. Добавить к G дугу $a_i \to b_i$.

Случай 2. В G имеется путь от b_i к a_i. Добавить к G дугу $b_i \to a_i$.

Случай 3. В G нет пути ни от a_i к b_i, ни от b_i к a_i. Сравнить $K_{a_i} : K_{b_i}$; затем, если $K_{a_i} \le K_{b_i}$, добавить к G дугу $a_i \to b_i$, если же $K_{a_i} > K_{b_i}$, то добавить дугу $b_i \to a_i$.

Нас интересует главным образом число сравнений ключей, выполняемых алгоритмом однородной сортировки, а не механизм, с помощью которого действительно устраняются избыточные сравнения; граф G не обязательно строить в явном виде—здесь он используется только для определения однородной сортировки.

Будем также рассматривать *ограниченную однородную сортировку,* при которой в указанных выше случаях 1–3 учитываются только пути длины 2. (Алгоритм ограниченной сортировки может выполнять некоторые избыточные сравнения, но, как показывает упр. 59, анализ ограниченного случая несколько проще.)

Докажите, что алгоритм ограниченной однородной сортировки совпадает с алгоритмом однородной сортировки, когда последовательность пар лексикографически упорядочена:

$$(1,2)\,(1,3)\,(1,4)\ldots(1,N)\,(2,3)\,(2,4)\ldots(2,N)\ldots(N-1,N).$$

Покажите, что на самом деле оба алгоритма эквивалентны "быстрой сортировке" (алгоритм 5.2.2Q), если все ключи различны и избыточные сравнения быстрой сортировки устранены, как в упр. 5.2.2-24. (Не обращайте внимания на порядок, в котором действительно выполняются сравнения в быстрой сортировке; учитывайте только, какие пары ключей сравниваются.)

59. [M38] Для заданной, как в упр. 58, последовательности пар $(a_1,b_1)\ \ldots(a_M,b_M)$ пусть c_i будет числом пар (j,k), таких, что $j<k<i$ и (a_i,b_i), (a_j,b_j), (a_k,b_k) образуют треугольник. (a) Докажите, что среднее число сравнений, выполняемых алгоритмом ограниченной однородной сортировки, равно $\sum_{1\le i\le M} 2/(c_i+2)$. (b) Используйте результат (a) и упр. 58, чтобы определить среднее число неизбыточных сравнений, выполняемых быстрой сортировкой. (c) Следующая последовательность пар навеяна сортировкой слиянием (но не эквивалентна ей):

$$(1,2)\,(3,4)\,(5,6)\ldots(1,3)\,(1,4)\,(2,3)\,(2,4)\,(5,7)\ldots(1,5)\,(1,6)\,(1,7)\,(1,8)\,(2,5)\ \ldots$$

При однородном методе, основанном на этой последовательности, будет выполняться в среднем больше или меньше сравнений, чем при быстрой сортировке?

60. [M29] Быстрая сортировка производит в наихудшем случае $\binom{N}{2}$ сравнений. Верно ли, что все алгоритмы ограниченной однородной сортировки (в смысле упр. 57) выполняют $\binom{N}{2}$ сравнений в наихудшем случае?

Picture: Рис. 61. Устройство, для которого стратегия метода пузырька является оптимальной.

61. [M48] (Х. Л. Бьюс.) Верно ли, что быстрая сортировка имеет минимальное среднее число сравнений среди всех алгоритмов (ограниченной) однородной сортировки?

62. [25] Докторская диссертация Говарда Б. Демута "Electronic Data Sorting" (Stanford University: October 1956) была, вероятно, первой работой, в которой сколько-нибудь детально рассматривались вопросы сложности вычислений. Демут рассмотрел несколько абстрактных моделей устройств для сортировки и нашел нижние и верхние оценки среднего и максимального времени выполнения, достижимого в каждой модели. Простейшая его модель—"циклическая нереверсивная память" (рис. 61)—станет темой этого упражнения.

Рассмотрим машину, которая сортирует $R_1\ R_2\ \ldots\ R_N$ за ряд проходов, где каждый проход состоит из следующих $N+1$ шагов:

Шаг 1. Установить $R \leftarrow R_1$i. (R—это внутренний регистр машины.)
Шаг i. ($1<i\le N$): либо (a) установить $R_{i-1}\leftarrow R$, $R\leftarrow R_i$;
либо (b) установить $R_{i-1}\leftarrow R_i$, оставляя R неизменным.
Шаг $N+1$. Установить $R_N\leftarrow R$.

Задача состоит в том, чтобы найти такой метод выбора между альтернативами (a) и (b), чтобы минимизировать число проходов, нужных для сортировки.

Докажите, что метод "пузырька" оптимален для этой модели. Другими словами, покажите, что при стратегии, которая выбирает альтернативу (a), если $R\le R_i$, и альтернативу (b), если $R>R_i$, достигается минимальное число проходов.

ВНЕШНЯЯ СОРТИРОВКА

Пришло время заняться интересными задачами, возникающими в том случае, когда число сортируемых записей превышает объем быстродействующего оперативного запоминающего устройства. Внешняя сортировка в корне отлична от внутренней (хотя в обоих случаях необходимо расположить записи данного файла в неубывающем порядке), и объясняется это тем, что время доступа к файлам на внешних носителях нас жесточайшим образом лимитирует. Структура данных должна быть такой, чтобы сравнительно медленные периферийные запоминающие устройства (ленты, диски, барабаны и т. д.) могли справиться с потребностями алгоритма сортировки. Поэтому большинство изученных до сих пор методов внутренней сортировки (вставка, обмен, выбор) фактически бесполезно для внешней сортировки; необходимо рассмотреть всю проблему заново.

Предположим, например, что предназначенный для сортировки файл состоит из 5000 записей $R_1\ R_2\ \ldots\ R_{5000}$ длиной по 20 слов (хотя ключи K_i не обязательно такой длины). Как быть, если во внутренней памяти данной машины помещается одновременно только 1000 из этих записей?

Сразу напрашивается такое решение: начать с сортировки каждого из пяти подфайлов $R_1 \ldots R_{1000}$, $R_{1001} \ldots R_{2000}$, $\ldots$, $R_{4001} \ldots R_{5000}$ по отдельности и затем слить полученные подфайлы. К счастью, слияние оперирует только очень простыми структурами данных, именно линейными списками, пройти которые можно последовательным образом, как стеки или очереди. Поэтому для слияния годятся самые дешевые внешние запоминающие устройства.

Только что описанный процесс—внутренняя сортировка с последующим "внешним слиянием"—весьма популярен, и наше изучение внешней сортировки сведется в основном к вариациям на эту тему.

Возрастающие последовательности записей, получаемые на начальной фазе внутренней сортировки, в литературе о сортировке часто называются *цепочками;* эта терминология довольно широко распространена, но, к сожалению, она противоречит еще более распространенному использованию термина "цепочка" в других разделах вычислительной науки, где он означает *произвольную* последовательность символов. При изучении переста- новок уже было дано вполне подходящее название для упорядоченных сегментов файла, которые мы договорились называть возрастающими отрезками или просто *отрезками.* В соответствии с этим будем использовать слово "отрезки" для обозначения упорядоченных частей файла. Таким образом, использование понятий "цепочки отрезков" и "отрезки цепочек" не приведет ни к каким недоразумениям.

Рассмотрим сначала процесс внешней сортировки, использующей в качестве вспомогательной памяти *магнитные ленты.* Вероятно, простейшим и наиболее привлекательным способом слияния с применением лент служит сбалансированное двухпутевое слияние, в основе которого лежит идея, использовавшаяся ранее в алгоритмах 5.2.4N, S и L. В процессе слияния нам потребуются четыре "рабочие ленты". На протяжении первой фазы возрастающие отрезки, получаемые при внутренней сортировке, помещаются поочередно на ленты 1 и 2 до тех пор, пока не исчерпаются исходные данные. Затем ленты 1 и 2 перематываем к началу и сливаем отрезки, находящиеся на этих лентах, получая новые отрезки, вдвое длиннее исходных. Эти новые отрезки записываются по мере их формирования попеременно на ленты 3 и 4. (Если на ленте 1 на один отрезок больше, чем на ленте 2, то предполагается, что лента 2 содержит дополнительный "фиктивный" отрезок длины 0.) Затем все ленты перематываются к началу и содержимое лент 3 и 4 сливается в удвоенные по длине отрезки, записываемые поочередно на ленты 1 и 2. Процесс продолжается (при этом длина отрезков каждый раз удваивается) до тех пор, пока не останется один отрезок (а именно весь упорядоченный файл). Если после внутренней сортировки было получено S отрезков, причем $2^{k-1} < S \le 2^k$, то процедура сбалансированного двухпутевого слияния произведет ровно $k = \lceil \log_2 S \rceil$ проходов по всем данным.

Например, в рассмотренной выше ситуации, когда требуется упорядочить 5000 записей, а объем внутренней памяти составляет 1000 записей, мы имеем $S = 5$. Начальная распределительная фаза процесса сортировки поместит пять отрезков на ленты следующим образом:

$$\begin{aligned} &\textit{Лента 1} \quad R_1 \ldots R_{1000}; R_{2001} \ldots R_{3000}; R_{4001} \ldots R_{5000}. \\ &\textit{Лента 2} \quad R_{1001} \ldots R_{2000}; R_{3001} \ldots R_{4000}. \\ &\textit{Лента 3} \quad \text{(пустая)} \\ &\textit{Лента 4} \quad \text{(пустая)} \end{aligned} \tag{1}$$

После первого прохода слияния на лентах 3 и 4 получатся более длинные отрезки, чем на лентах 1 и 2:

$$\begin{aligned} &\textit{Лента 3} \quad R_1 \ldots R_{2000}; R_{4001} \ldots R_{5000}. \\ &\textit{Лента 4} \quad R_{2001} \ldots R_{4000}. \end{aligned} \tag{2}$$

В конец ленты 2 неявно добавляется фиктивный отрезок, так что отрезок $R_{4001} \ldots R_{5000}$ просто копируется на ленту 3. После перемотки всех лент к началу следующий проход по данным приведет к такому результату:

$$\begin{aligned} &\textit{Лента 1} \quad R_1 \ldots R_{4000}. \\ &\textit{Лента 2} \quad R_{4001} \ldots R_{5000}. \end{aligned} \tag{3}$$

(Отрезок $R_{4001} \ldots R_{5000}$ снова копируется, но если бы мы начали с 8000 записей, то в этот момент лента 2 содержала бы $R_{4001} \ldots R_{8000}$.) Наконец, после еще одной перемотки на ленте 3 окажется отрезок $R_1 \ldots R_{5000}$, и сортировка закончится.

Сбалансированное слияние легко обобщается на случай T лент для любого $T \ge 3$. Выберем произвольное число P, такое, что $1 \le P < T$, и разделим T лент на два "банка": P лент в левом банке и $T - P$ лент в правом банке. Распределим исходные отрезки как можно равномернее по P лентам левого "банка", затем выполним P-путевое слияние слева направо, после этого—$(T - P)$-путевое слияние справа налево и т. д., пока сортировка не завершится. Обычно значение P лучше всего выбирать равным $\lceil T/2 \rceil$ (см. упр. 3, 4).

При $T = 4$, $P = 2$ имеем частный случай—сбалансированное двухпутевое слияние. Вновь рассмотрим предыдущий пример, используя большее количество лент; положим $T = 6$ и $P = 3$. Начальное распределение теперь будет таким:

$$\begin{array}{ll} \textit{Лента 1} & R_1 \ldots R_{1000}; R_{3001} \ldots R_{4000}. \\ \textit{Лента 2} & R_{1001} \ldots R_{2000}; R_{4001} \ldots R_{5000}. \\ \textit{Лента 3} & R_{2001} \ldots R_{3000}. \end{array} \tag{4}$$

Первый проход слияния приведет к

$$\begin{array}{ll} \textit{Лента 4} & R_1 \ldots R_{3000}. \\ \textit{Лента 5} & R_{3001} \ldots R_{5000}. \\ \textit{Лента 6} & (\text{пустая}) \end{array} \tag{5}$$

(Предполагается, что на ленте 3 помещен фиктивный отрезок.) На втором проходе слияния работа завершается и отрезки $R_1 \ldots R_{5000}$ помещаются на ленту 1. Этот частный случай $T = 6$ эквивалентен $T = 5$, так как шестая лента используется лишь при $S \geq 7$.

Трехпутевое слияние затрачивает фактически несколько больше времени центрального процессора, чем двухпутевое, но оно обычно пренебрежимо мало по сравнению с временем, необходимым для чтения, записи и перемотки ленты; мы довольно хорошо оценим время выполнения сортировки, если примем во внимание только суммарную величину перемещений лент. В предыдущем примере ((4) и (5)) требуются только два прохода по данным в сравнении с тремя проходами при $T = 4$. Таким образом, слияние при $T = 6$ займет около двух третей времени по отношению к предыдущему случаю.

Сбалансированное слияние кажется очень простым и естественным. Но если приглядеться внимательнее, то сразу видно, что это не наилучший способ в разобранных выше частных случаях. Вместо того чтобы переходить от (1) к (2) и перематывать все ленты, нам следовало остановить первое слияние, когда ленты 3 и 4 содержали соответственно $R_1 \ldots R_{2000}$ и $R_{2001} \ldots R_{4000}$, а лента 1 была готова к считыванию $R_{4001} \ldots R_{5000}$. Затем ленты 2, 3, 4 могли быть перемотаны к началу, и сортировка завершилась бы трехпутевым слиянием на ленту 2. Общее число записей, прочитанных с ленты в ходе этой процедуры, составило бы $4000 + 5000 = 9000$ против $5000 + 5000 + 5000 = 15000$ в сбалансированной схеме. Сообразительная машина могла бы постичь и это!

Имея пять отрезков и четыре ленты, можно поступить еще лучше, распределив отрезки следующим образом:

$$\begin{array}{ll} \textit{Лента 1} & R_1 \ldots R_{1000}; R_{3001} \ldots R_{4000}. \\ \textit{Лента 2} & R_{1001} \ldots R_{2000}; R_{4001} \ldots R_{5000}. \\ \textit{Лента 3} & R_{2001} \ldots R_{3000}. \\ \textit{Лента 4} & (\text{пустая}). \end{array}$$

Теперь, выполнив трехпутевое слияние на ленту 4, затем перемотку лент 3 и 4 с последующим трехпутевым слиянием на ленту 3, можно было бы завершить сортировку, прочитав всего $3000 + 5000 = 8000$ записей.

Наконец, если бы мы имели шесть лент, то могли бы, конечно, записать исходные отрезки на ленты 1–5 и закончить сортировку за один проход, выполнив пятипутевое слияние на ленту 6. Рассмотрение этих случаев показывает, что простое сбалансированное слияние не является наилучшим, и было бы интересно поискать улучшенные схемы слияния.

В последующих пунктах этой главы внешняя сортировка исследуется более глубоко. В п. 5.4.1 рассматривается фаза внутренней сортировки, порождающая начальные отрезки; особый интерес представляет техника "выбора с замещением", которая использует порядок, присутствующий в большинстве данных, чтобы породить длинные отрезки, значительно превосходящие емкость внутренней памяти. В п. 5.4.1 обсуждаются также, структуры данных, удобные для целей многопутевого слияния.

Важнейшие схемы слияния обсуждаются в п. 5.4.2—5.4.5. Пока мы не вступим в единоборство с грубой действительностью настоящих лент и реальных сортируемых данных, для нас лучше, изучая характеристики этих схем, иметь весьма наивное представление о ленточной сортировке. Например, можно с легкой душой полагать (как мы делали до сих пор), что первоначальные исходные записи появляются волшебным образом в течение первой распределительной фазы; на самом деле они вероятно, будут занимать одну из наших лент и, быть может, даже целиком заполнят несколько бобин, так как лента не бесконечна! Лучше всего пренебречь подобными техническими деталями до тех пор, пока не будет достигнуто "академическое" понимание классических схем слияния. Затем в п. 5.4.6 мы "вернемся на землю", рассмотрев практические ограничения, которые сильно влияют на выбор схемы слияния. В п. 5.4.6 сравниваются основные схемы слияния из п. 5.4.2—5.4.5 с использованием множества разнообразных предположений, которые встречаются на практике.

Иные подходы к проблеме внешней сортировки, не основанные на слиянии, обсуждаются в п. 5.4.7 и 5.4.8. Наш обзор внешней сортировки заканчивается в п. 5.4.9, где рассматривается важная проблема сортировки на такой памяти, как диски и барабаны.

Упражнения

1. [15] В тексте внешняя сортировка рассматривается после внутренней. Почему нельзя вообще покончить с фазой внутренней сортировки, просто сливая записи во все более и более длинные отрезки прямо с самого начала?
2. [10] Каким будет содержимое лент (аналогичное (1)–(3)), если записи $R_1 \ldots R_{5000}$ сортируются с помощью 3-ленточного сбалансированного метода. при $P = 2$? Сравните этот случай со слиянием на 4 лентах; сколько проходов по всем данным будет сделано после первоначального распределения отрезков?
3. [20] Покажите, что сбалансированное $(P, T-P)$-путевое слияние, примененное к S начальным отрезкам, производит $2k$ проходов, если $P^k(T-P)^{k-1} < S \le P^k(T-P)^k$, и производит $2k+l$ проходов, если $P^k(T-P)^k < S \le P^{k+1}(T-P)^k$.

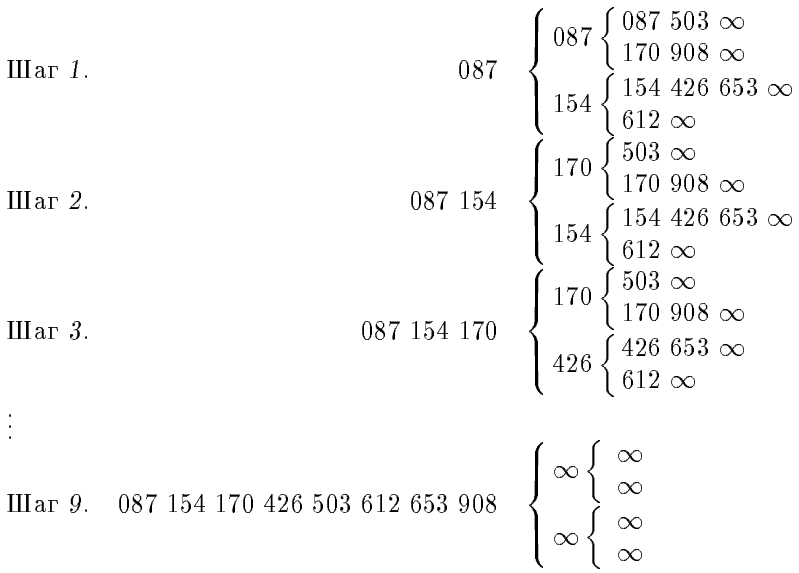
   Дайте простые формулы для вычисления (a) точного числа проходов как функции от S при $T = 2P$, (b) приближенного числа проходов при $S \to \infty$ для любых P и T.
4. [BM15] При каком значении P, $1 \le P < T$, значение $P(T-P)$ максимально?

Многопутевое слияние и выбор с замещением

В п. 5.2.4 мы изучали методы внутренней сортировки, основанные на двухпутевом слиянии—процессе объединения двух упорядоченных последовательностей в одну. Нетрудно расширить это до понятия P-путевого слияния, когда P входных отрезков объединяются в один выходной.

Пусть имеется P возрастающих отрезков, т. е. последовательностей записей, ключи которых расположены в неубывающем порядке. Очевидным способом их слияния будет следующий: посмотреть на первые записи каждого отрезка и выбрать из них ту, которой соответствует минимальный ключ; эта запись передается на выход и исключается из входных данных, затем процесс повторяется.

Пока P не слишком велико, этот выбор удобно осуществлять, просто выполняя $P-1$ сравнений для нахождения наименьшего из текущих ключей. Но если, скажем, $P \ge 8$, то можно сократить работу, используя *дерево выбора,* как описано в п. 5.2.3; тогда каждый раз требуется только $\approx \log_2 P$ сравнений (после начального формирования дерева). Рассмотрим, например, четырехпутевое слияние с двухуровневым деревом выбора:



В этом примере в конце каждого отрезка помещен добавочный ключ ∞, чтобы слияние заканчивалось естественно. Так как внешнее слияние обычно имеет дело с очень длинными отрезками, то эта добавочная запись с ключом ∞ не увеличит существенно длину данных или объем работы при слиянии; кроме того; такая "концевая" запись часто служит удобным способом разделения записей файла.

В рассматриваемом процессе каждый шаг, кроме первого, состоит из замещения наименьшего элемента следующим элементом из этого же отрезка и изменения соответствующего пути в дереве выбора. Так, на шаге 2 изменяется 3 узла, которые содержали 087 на шаге 1; на шаге 3 изменяется 3 узла, содержавшие 154 на шаге 2, и т. д. Процесс замещения в дереве выбора одного ключа другим называется *выбором с замещением.*

Мы можем по-разному рассматривать приведенное четырехпутевое слияние. С одной точки зрения оно эквивалентно трем двухпутевым слияниям, выполняемым совместно, как сопрограммы; каждый узел дерева выбора изображает одну из последовательностей, используемых в процессах слияния. Дерево выбора, по существу, используется как приоритетная очередь с дисциплиной "наименьший из включенных первым исключается".

Picture: Рис. 62. Турнир, при котором выбирается наименьший ключ; используется полное бинарное дерево, узлы которого занумерованы номерами от 1 до 23.

Так же как в п. 5.2.3, мы могли бы для реализации приоритетной очереди использовать не дерево выбора, а пирамиду. (Пирамиду, конечно, надо было бы организовать таким образом, чтобы на вершине появлялся *наименьший* элемент, а не наибольший, обратив для этого порядок соотношения (5.2.3-3).) Так как пирамида имеет переменный размер, можно избежать использования ключа ∞; слияние заканчивается, когда пирамида становится пустой. С другой стороны, приложения внешней сортировки обычно имеют дело со сравнительно длинными записями и ключами, так что в пирамиду будут записаны вместо самих ключей указатели на них; мы увидим ниже, что деревья выбора могут настолько удобно изображаться с помощью указателей, что они, вероятно, в данной ситуации лучше пирамид.

Дерево "проигравших". На рис. 62 изображено полное бинарное дерево с 12 внешними (квадратными) узлами и 11 внутренними (круглыми); во внешних узлах записаны ключи, во внутренних—"победители", если дерево рассматривать как турнир для выбора

Picture: Рис. 63. Тот же турнир, что и на рис. 62, но показаны проигравшие, а не победители; чемпион находится на самом верху.

наименьшего ключа. Числа меньшего размера над каждым узлом показывают традиционный способ распределения последовательных позиций памяти для полного бинарного дерева.

Чтобы определить новое состояние дерева выбора, изображенного на рис. 62, когда наименьший ключ 061 будет заменен другим ключом, нужно рассмотреть только ключи 512, 087 и 154 и никакие другие. Если интерпретировать дерево как турнир, последние три ключа представляют собой проигравших в матчах с игроком 061. Это наводит на мысль, что мы в действительности должны записать во внутренние узлы *проигравшего* в каждом матче, а не победителя, тогда информация, необходимая для изменения дерева, будет легкодоступной.

На рис. 63 изображено то же дерево, что и на рис. 62, но вместо победителей в нем представлены проигравшие. Дополнительный узел с номером "0" добавлен на вершине дерева для указания чемпиона турнира. Заметим, что каждый ключ, кроме чемпиона, является проигравшим ровно один раз (см. п. 5.3.3); таким образом, каждый ключ появляется один раз во внешнем узле и один раз во внутреннем.

На практике внешним узлам в нижней части рис. 63 будут соответствовать весьма длинные записи, расположенные в памяти ЭВМ, а внутренним узлам—указатели на эти записи. Заметим, что P-путевое слияние требует ровно P внешних и P внутренних узлов по одному в соседних группах, что наводит на мысль о ряде эффективных методов распределения памяти. Нетрудно видеть, каким образом можно использовать дерево проигравших для выбора с замещением. Более детально мы обсудим этот алгоритм в настоящем пункте чуть позже.

Получение начальных отрезков посредством выбора с замещением. Техника выбора с замещением может использоваться также на первой фазе внешней сортировки, если фактически выполнить P-путевое слияние входных данных с самими собой. В этом случае P выбирается достаточно большим, чтобы заполнить, по существу, всю внутреннюю память. Каждая запись при выводе замещается очередной записью из исходных данных. Если у этой новой записи ключ меньше, чем у выведенной записи, то мы не включаем ее в текущий отрезок; в противном случае мы обычным образом включаем ее в дерево выбора, так что она образует часть отрезка, порождаемого в данный момент. Таким образом, каждый отрезок может содержать больше P записей, хотя в любой момент в дереве выбора находится не более P записей. Таблица 1 иллюстрирует этот процесс для $P = 4$ (числа, заключенные в скобки, ожидают включения в следующий отрезок).

Этот важный метод формирования начальных отрезков был впервые описан Гарольдом К. Сьювордом [Master's Thesis, Digital Computer Laboratory Report R-232 (Mass. Inst. of Technology, 1954), 29–30], который привел соображения в пользу того, что при применении к случайным данным отрезки, видимо, будут содержать более $1.5P$ записей. Ту же идею предложил примерно в 1950 г. А. И. Думи в связи со специальным устройством для сортировки, разрабатываемым Engineering Research Associates, но не опубликовал ее. Само название "выбор с замещением" (replacement selecting) было придумано Э. Х. Фрэндом [*JACM*, **3** (1956), 154], который заметил, что "ожидаемая длина порождаемой последовательности не поддается

точной формулировке, но

Таблица 1

Пример четырехпутевого выбора с замещением

Содержимое памяти				Вывод
503	087	512	061	061
603	087	512	908	087
503	170	512	908	170
503	897	512	908	503
(275)	897	512	908	512
(275)	897	653	908	653
(275)	897	(426)	908	897
(275)	(154)	(426)	908	908
(275)	(154)	(426)	(509)	(конец отрезка)
275	154	426	509	154
275	612	426	509	275
и т.д.				

эксперименты позволяют предположить, что разумной оценкой будет $2P$".

Э. Ф. Мур предложил изящный способ доказательства того, что ожидаемая длина отрезка равна $2P$, проведя аналогию со

Picture: Рис. 64. Вечный снегоочиститель в своем нескончаемом цикле.

снегоочистителем, движущимся по кругу [U. S. Patent 2983904 (1961), cols. 3–4]. Рассмотрим ситуацию, изображенную на рис. 64: на кольцевую дорогу равномерно падают снежинки, а один снегоочиститель непрерывно убирает снег. Снег исчезает из системы, как только он сбрасывается за пределы дороги. Точки дороги обозначаются вещественными числами x, $0 \le x < 1$; снежинка, падающая в точку x, соответствует входной записи с ключом x, а снегоочиститель представляет собой вывод процесса выбора с замещением.

Скорость снегоочистителя обратно пропорциональна весу снега, который встречается на его пути, и ситуация вполне уравновешена, так что общее количество снежинок на дороге в точности равно P. Каждый раз, когда снегоочиститель проходит точку 0, на выходе формируется новый отрезок.

Picture: Рис. 65. Поперечное сечение, показывающее переменный вес снега перед снегоочистителем, когда система находится в установившемся режиме.

Интуитивно ясно, что система, поработав некоторое время, выйдет на установившийся режим, при котором снегоочиститель будет двигаться с постоянной скоростью (в силу круговой симметрии дороги). Это означает, что в точке, где находится снегоочиститель, снег имеет постоянный вес, а впереди снегоочистителя этот вес линейно уменьшается, как показано на рис. 65. Отсюда следует, что количество снега, удаляемого за один оборот (а именно длина отрезка), вдвое превосходит количество снега P, присутствующего в любой момент.

Во многих коммерческих приложениях входные данные *нельзя* считать полностью случайными—в них уже существует определенный порядок; следовательно, отрезки, порождаемые выбором с замещением, имеют тенденцию содержать даже больше, чем $2P$ записей. В дальнейшем мы увидим, что время, нужное для внешней сортировки слиянием, в значительной степени зависит от количества отрезков, порождаемых начальной распределительной фазой, так что выбор с замещением становится особенно привлекательным; другие способы внутренней сортировки порождали бы примерно вдвое больше начальных отрезков, поскольку размеры памяти ограничены.

Теперь детально рассмотрим процесс создания начальных отрезков выбором с замещением. Следующий алгоритм принадлежит Джону Р. Уолтерсу, Джеймсу Пэйнтеру и Мартину Залку, которые использовали его в программе сортировки посредством слияния, для ЭВМ Philco 2000 в 1958 г. Он включает в себя прекрасный способ начального формирования дерева выбора и разделения записей, принадлежащих разным отрезкам, а также вывода последнего отрезка по единообразной, сравнительно простой логике. (Надлежащая обработка последнего отрезка, порожденного выбором с замещением, оказывается довольно сложной; блок, осуществляющий эту обработку, часто бывает камнем преткновения для программиста.) Основная идея заключается в том, чтобы рассматривать каждый ключ как пару (S, K), где K—первоначальный ключ, а S—номер отрезка, которому принадлежит данная запись. Мы получим выходную последовательность, порожденную выбором с замещением, если лексикографически упорядочим эти расширенные ключи (S считается старше K).

Приводимый ниже алгоритм использует для представления дерева выбора структуру данных, состоящую из P узлов; предполагается, что j-й узел $X[j]$ содержит c слов, начинающихся с $\texttt{LOC}(X[j]]) = L_0 + cj$ при $0 \le j < P$, он представляет как внутренний узел с номером j, так и внешний узел с номером $P + j$ (рис. 63). В каждом узле имеется несколько полей:

`KEY` = ключ, находящийся в данном внешнем узле;
`RECORD` = запись, находящаяся в данном внешнем узле (включая `KEY` как подполе);
`LOSER` = указатель на "проигравшего" в данном внутреннем узле;
`RN` = номер отрезка, куда включена запись, на которую указывает `LOSER`;
`FE` = указатель на внутренний узел, расположенный в дереве выбора выше данного внешнего узла;
`FI` = указатель на внутренний узел, расположенный в дереве выбора выше данного внутреннего узла.

Например, при $P = 12$ внутренний узел с номером 5 и внешний узел с номером 17 на рис. 63 оба будут представлены узлом $X[5]$ с полями $\texttt{KEY} = 170$, $\texttt{LOSER} = L_0 + 9c$ (адрес внешнего узла с номером 21), $\texttt{FE} = L_0 + 8c$, $\texttt{FI} = L_0 + 2c$.

Значения в полях `FE` и `FI` являются константами; таким образом, строго говоря, нет необходимости хранить их в явном виде. Однако иногда на начальной фазе внешней сортировки для быстрой работы с устройствами ввода/вывода выгоднее хранить эту избыточную информацию, чем вычислять ее каждый раз заново.

Алгоритм R. *(Выбор с замещением.)* Этот алгоритм читает записи последовательно из входного файла и записывает их последовательно в выходной файл, производя `RMAX` отрезков длины P или больше (за исключением последнего отрезка). Имеется $P \ge 2$ узлов $X[0], \ldots, X[P-1]$ с полями, описанными выше.

Picture: Рис. 66. Получение начальных отрезков посредством выбора с замещением.

R1 [Начальная установка.] Установить $\texttt{RMAX} \leftarrow 0$, $\texttt{RC} \leftarrow 0$, $\texttt{LASTKEY} \leftarrow \infty$, $\texttt{Q} \leftarrow \texttt{LOC}(X[0])$ и $\texttt{RQ} \leftarrow 0$. (`RC`—номер текущего отрезка, а `LASTKEY`—ключ последней выведенной записи. Начальное значение `LASTKEY` должно быть больше любого возможного ключа; ср. с упр. 8.) Для $0 \le j < P$ начальное содержимое $X[j]$ установить следующим образом (где $\texttt{J} = \texttt{LOC}(X[j])$):

$$\texttt{LOSER(J)} \leftarrow j; \texttt{RN(J)} \leftarrow 0;$$
$$\texttt{FE(J)} \leftarrow \texttt{LOC}(X[\lfloor (P+j)/2 \rfloor]); \texttt{FI(J)} \leftarrow \texttt{LOC}(X[\lfloor j/2 \rfloor]).$$

(Установка `LOSER(J)` и `RN(J)` есть искусственный способ образовать начальное дерево, рассматривая фиктивный отрезок с номером 0, который никогда не выводится. Это— некий трюк; см. упр. 10.)

R2 [Конец отрезка?] Если $\texttt{RQ} = \texttt{RC}$, то перейти к шагу **R3**. (В противном случае $\texttt{RQ} = \texttt{RC} + 1$, и мы только что закончили отрезок с номером `RC`; в этом месте следует выполнить те специальные действия, которые нужны в соответствии со схемой слияния для последующих этапов сортировки.) Если $\texttt{RQ} > \texttt{RMAX}$, то алгоритм завершен; в противном случае установить $\texttt{RC} \leftarrow \texttt{RQ}$.

R3 [Вывод вершины дерева.] (Сейчас `Q` указывает на "чемпиона", и `RQ`—номер его отрезка.) Если $\texttt{RQ} \neq 0$, то вывести `RECORD(Q)` и установить $\texttt{LASTKEY} \leftarrow \texttt{KEY(Q)}$.

R4 [Ввод новой записи.] Если входной файл исчерпан, установить $\texttt{RQ} \leftarrow \texttt{RMAX} + 1$ и перейти к шагу **R5**. В противном случае поместить новую запись из входного файла в `RECORD(Q)`. Если $\texttt{KEY(Q)} < \texttt{LASTKEY}$ (т. е. эта запись не принадлежит текущему отрезку), то $\texttt{RQ} \leftarrow \texttt{RQ} + 1$, и теперь, если $\texttt{RQ} > \texttt{RMAX}$, установить $\texttt{RMAX} \leftarrow \texttt{RQ}$.

R5 [Подготовка к изменению.] (Сейчас `Q` указывает на новую запись с номером отрезка `RQ`.) Установить $\texttt{T} \leftarrow \texttt{FE(Q)}$. (`T`—переменный указатель, который будет двигаться по дереву.)

R6 [Установка нового проигравшего.] Если $\texttt{RN(T)} < \texttt{RQ}$ или если $\texttt{RN(T)} = \texttt{RQ}$ и $\texttt{KEY(LOSER(T))} < \texttt{KEY(Q)}$. то поменять местами $\texttt{LOSER(T)} \leftrightarrow \texttt{Q}$, $\texttt{RN(T)} \leftrightarrow \texttt{RQ}$. (В переменных `Q` и `RQ` запоминается текущий победитель и номер его отрезка.)

R7 [Сдвиг вверх.] Если $\texttt{T} = \texttt{LOC}(X[1])$, то вернуться к шагу **R2**, в противном случае $\texttt{T} \leftarrow \texttt{FI(T)}$ и вернуться к **R6**. ∎

В алгоритме R говорится о вводе и выводе записей по одной, тогда как практически оказывается лучше читать и записывать относительно большие блоки записей. Следовательно, на самом деле за кулисами прячутся буферы ввода и вывода; их присутствие в памяти приводит к уменьшению значения P. Это будет пояснено в п. 5.4.6.

Э. Х. Фрэнд [*JACM*, bf 3 (1956), 154] предложил следующее обобщение метода выбора с замещением. В тех случаях, когда вводимый ключ меньше, чем LASTKEY (так что он не попадет в текущий отрезок), но больше или равен последнему ключу, действительно записанному на ленту (так что его фактически можно было бы поместить в текущий отрезок), вставлять этот ключ внутрь буфера вывода. Кроме того, некоторые ЭВМ умеют выполнять "чтение вразброс" и "запись со сборкой", т. е. вводить информацию во внутреннюю память не обязательно в последовательные ячейки, а "вразброс" и выводить ее, собирая из разных мест. Это позволяет совмещать память для буферов с памятью для дерева выбора.

***Преобразование отрезков с задержкой.** Р. Дж. Динсмор [*CACM*, **8** (1965), 48] предложил интересное усовершенствование выбора с замещением, использующее понятие, которое будем называть *степенью свободы*. Как мы видели, каждый блок записей, находящийся на ленте в составе отрезка, содержит записи в неубывающем порядке, так что первый элемент наименьший, а последний наибольший. В обычном процессе выбора с замещением наименьший элемент каждого блока в некотором отрезке всегда не меньше, чем наибольший элемент в предыдущем блоке этого отрезка; это соответствует "1 степени свободы". Динсмор предложил ослабить это условие до "m степеней свободы"; новое условие не требует, чтобы наименьший элемент каждого блока был не меньше, чем наибольший элемент предыдущего блока, но он *не должен быть меньше, чем наибольшие элементы каких-то m предыдущих блоков того же отрезка*. Записи в отдельном блоке упорядочены, как и ранее, но соседние блоки не обязаны быть взаимно упорядоченными.

Предположим, например, что блоки содержат только по две записи; следующая последовательность блоков является отрезком с тремя степенями свободы:

$$|08\ 50|06\ 90|17\ 27|42\ 67|51\ 89| \tag{1}$$

Следующий блок, который может быть частью этого отрезка, должен начинаться с элемента, не меньшего, чем третий по порядку элемент множества $\{50, 90, 27, 67, 89\}$, считая от наибольшего, т. е. не меньше 67. Последовательность (1) не является отрезком с двумя степенями свободы, так как 17 меньше, чем 50 и 90.

Отрезок с m степенями свободы в процессе чтения в следующей фазе сортировки может быть преобразован таким образом, что для всех практических целей он будет отрезком в обычном смысле. Начнем с чтения первых m блоков в m буферов и будем производить m-путевое слияние их; когда один из буферов исчерпается, поместим в него $(m+1)$-й блок и т. д. Таким образом, мы можем восстановить отрезок в виде одной упорядоченной последовательности, так как первое слово каждого вновь считываемого блока должно быть больше или равно последнему слову только что исчерпанного блока (если оно не было меньше, чем наибольшие элементы каких-либо m блоков, предшествующих ему). Этот метод преобразования отрезка, в сущности, является m-путевым слиянием, использующим только одно ленточное устройство для всех входных блоков!

Процедура преобразования действует как сопрограмма, к которой обращаются каждый раз, когда нужно получить одну очередную запись отрезка. Мы можем преобразовывать различные отрезки с различных ленточных устройств и с различными степенями свободы и сливать получающиеся отрезки—все в одно и то же время. Это, по существу, подобно тому, как если бы мы четырехпутевое слияние, рассмотренное в начале этого пункта, представили себе как несколько двухпутевых слияний, происходящих одновременно.

Эта остроумная идея до сих пор не проанализирована до конца. Имеются некоторые предварительные результаты, показывающие, что, когда P велико по сравнению с размером блока, длина отрезка при $m = 2$ приблизительно равна $2.1P$, она равна $2.3P$ при $m = 4$ и $2.5P$ при $m = 8$. Такое увеличение, быть может, недостаточно, чтобы оправдать усложнение алгоритма. С другой стороны, метод может оказаться выгодным, если на. протяжении второго этапа сортировки есть место для довольно большого числа буферов.

***Натуральный выбор.** Другой путь увеличения длины отрезков, порождаемых выбором с замещением, был исследован У. Д. Фрэйзером и Ч. К. Уоном. Их идея состоит в том, чтобы следовать алгоритму R, но, когда на шаге R4 KEY(Q) < LASTKEY, новая запись RECORD(Q) не остается в дереве,, а выводится в некоторый внешний *резервуар* и читается новая запись. Этот процесс продолжается до тех пор, пока в резервуаре не окажется определенное количество записей P'; тогда остаток текущего отрезка выводится из дерева, и элементы резервуара используются в качестве исходных данных для следующего отрезка.

Этот метод должен порождать более длинные отрезки, чем выбор с замещением, поскольку он "обходит" вновь поступающие "мертвые" записи, вместо того чтобы позволять им заполнять дерево; но ему требуется дополнительное время на обмен с резервуаром. Когда $P' > P$, некоторые записи могут оказываться в резервуаре дважды, но при $P' \leq P$ такого случиться не может.

Фрэйзер и Уон, проведя обширные эмпирические испытания своего метода, заметили, что, когда P достаточно велико (скажем, $P \geq 32$) и $P' = P$, средняя длина отрезка для случайных данных оказывается равной eP, где $e \approx 2.718$—основание натуральных логарифмов. Это явление, а также тот факт, что метод был получен как эволюционное развитие простого выбора с замещением, послужили для них непосредственным основанием назвать свой метод *натуральным выбором.*

Можно доказать "натуральный" закон для длины отрезка, вновь воспользовавшись аналогией со снегоочистителем на рис. 64 и применив элементарный математический анализ. Пусть L обозначает длину пути, а $x(t)$—положение снегоочистителя в момент t при $0 \leq t \leq T$. Предположим, что в момент T резервуар заполняется; в этот момент падение снега временно прекращается, пока снегоочиститель возвращается в исходное положение (счищая P снежинок, оставшихся на его пути). Ситуация такая же, как и ранее, только "условия равновесия" другие—вместо P снежинок на всей дороге в любой момент времени мы имеем P снежинок перед снегоочистителем и резервуар (за снегоочистителем), заполняющийся до уровня в P снежинок. В течение интервала времени dt снегоочиститель продвигается на dx, если выводятся $h(x,t)dx$ записей, где $h(x,t)$—толщина слоя снега в момент времени t в точке $x = x(t)$, измеряемая в соответствующих единицах; следовательно, $h(x,t) = h(x,0) + Kt$ для всех x. Так как число записей в памяти остается постоянным, то $h(x,t)dx$ есть также число записей, вводимых *перед* снегоочистителем, а именно $Kdt(L-x)$, где K—скорость падения снега (рис. 67). Таким образом,

$$\frac{dx}{dt} = \frac{K(L-x)}{h(x,t)}. \tag{2}$$

К счастью, оказывается, что $h(x,t)$—константа, и она равна KT при всех $x = x(t)$ и $0 \leq t \leq T$, так как снег падает равномерно в точку $x(t)$ в течение $T-t$ единиц времени после того, как снегоочиститель проходит эту точку, плюс t единиц времени перед тем, как он вернется. Иными словами, снегоочиститель видит перед собой все время одинаковый слой снега на протяжении всего пути, если допустить, что достигнут установившийся режим, когда этот путь все время один и тот же. Следовательно, общее количество счищаемого снега (длина отрезка) есть KTL,

Picture: Рис. 67. Вводится и выводится равное количество снега; за время dt снегоочиститель перемещается на dx.

а количество снега в памяти есть количество снега, счищаемого после момента T, а именно $KT(L-x(T))$.

Решением уравнения (2) при условии, что $x(0) = 0$, будет

$$x(t) = L(1 - e^{-t/T}). \tag{3}$$

Следовательно, $P = KTLe^{-1} =$ (длина отрезка)$/e$—это как раз то, что мы и хотели доказать.

В упр. 21–23 показано, что этот анализ можно распространить на случай произвольного P'; например, когда $P' = 2P$, средняя длина отрезка оказывается равной $e^{\theta}(e-\theta)P$, где $\theta = \frac{1}{2}(e - \sqrt{e^2-4})$,—результат, который вряд ли можно было предположить заранее! В табл. 2 приводится зависимость между длиной отрезка и размером резервуара; с помощью этой таблицы можно оценить полезность натурального выбора для конкретной машины в той или иной ситуации.

***Анализ выбора с замещением.** Вернемся теперь к случаю выбора с замещением без вспомогательного резервуара. Аналогия со снегоочистителем дает довольно хорошую оценку средней длины отрезков, получаемых при выборе с замещением "в пределе", тем не менее можно получить значительно более точную информацию об алгоритме R, применяя факты об отрезках в перестановках, изученных нами в п. 5.1.3. Для удобства будем считать, что входной файл является последовательностью (произвольной длины) независимых случайных действительных чисел, расположенных между 0 и 1.

Таблица 2

Длина отрезков при натуральном выборе

Размер резервуара	Длина отрезка	Параметр	Размер резервуара	Длина отрезка	Параметр
$1.00000P$	$2.71828P$	1.00000	$0.38629P$	$2.00000P$	0.69315
$2.00000P$	$3.53487P$	1.43867	$1.30432P$	$3.00000P$	1.15881
$3.00000P$	$4.16220P$	1.74773	$2.72294P$	$4.00000P$	1.66862
$4.00000P$	$4.69445P$	2.01212	$4.63853P$	$5.00000P$	2.16714
$5.00000P$	$5.16369P$	2.24038	$21.72222P$	$10.00000P$	4.66667
$10.00000P$	$7.00877P$	3.17122	$5.29143P$	$5.29143P$	2.31329

"Параметр" $k+\theta$ определен в упр. 22

Пусть

$$g_P(z_1, z_2, \ldots, z_k) = \sum_{l_1, l_2, \ldots, l_k \geq 0} a_P(l_1, l_2, \ldots, l_k) z_1^{l_1} z_2^{l_2} \ldots z_k^{l_k}$$

—производящая функция для длины отрезка, полученного при P-путевом выборе с замещением, примененном к. такому файлу, где $a_P(l_1, l_2, \ldots, l_k)$ есть вероятность того, что первый отрезок имеет длину l_1, второй—длину l_2, ..., k-й имеет длину l_k. Будем основываться на следующей "теореме независимости", Так как она сводит наш анализ к случаю $P = 1$.

Теорема К. $g_P(z_1, z_2, \ldots, z_k) = g_1(z_1, z_2, \ldots, z_k)^P$.

Доказательство Пусть исходные ключи суть $X_1, X_2, X_3, \ldots$. Алгоритм R разделяет их на P подпоследовательностей в соответствии с тем, в какой внешний узел дерева они попадают; подпоследовательность, содержащая X_n, определяется значениями $X_1, \ldots, X_{n-1}$. Таким образом, эти подпоследовательности являются независимыми последовательностями независимых случайных чисел, расположенных между 0 и 1. Кроме того, выход выбора с замещением в точности совпадает с результатом P-путевого слияния, если его произвести над этими подпоследовательностями; некоторый элемент принадлежит j-му отрезку подпоследовательности тогда и только тогда, когда он принадлежит j-му отрезку, полученному при выборе с замещением (так как на шаге R4 ключи `LASTKEY` и `KEY(Q)` принадлежат одной подпоследовательности).

Иначе говоря, можно считать, что алгоритм R применяется к P случайным независимым исходным файлам и что шаг R4 читает следующую запись из файла, соответствующего внешнему узлу `Q`; в этом смысле рассматриваемый алгоритм эквивалентен P-путевому слиянию, где концы отрезков отмечаются убыванием элементов.

Таким образом, на выходе будут отрезки длин $(l_1, \ldots, l_k)$ тогда и только тогда, когда подпоследовательности состоят из отрезков длин $(l_{11}, \ldots, l_{1k})$, ..., $(l_{P1}, \ldots, l_{Pk})$ соответственно; где l_{ij}—некоторые неотрицательные целые числа, удовлетворяющие соотношению $\sum_{1 \leq i \leq P} l_{ij} = l_j$ при $1 \leq j \leq k$. Отсюда следует, что

$$a_P(l_1, \ldots, l_k) = \sum_{\substack{l_{11}+\cdots+l_{P1}=l_1 \\ \vdots \\ l_{1k}+\cdots+l_{Pk}=l_k}} a_1(l_{11}, \ldots, l_{1k}) \ldots a_1(l_{P1}, \ldots, l_{Pk}),$$

что эквивалентно искомому результату. ∎

В п. 5.1.3 мы изучили среднее значение L_k—длины k-го отрезка при $P = 1$ (эти значения приведены в табл. 5.1.3-2). Из теоремы К следует, что средняя длина k-го отрезка при любом P в P раз больше средней длины при $P = 1$, она равна $L_k P$; дисперсия также в P раз больше, так что стандартное отклонение длины отрезка пропорционально $\sqrt{P}$. Эти результаты были впервые получены Б. Дж. Гэсснер около 1958 г.

Таким образом, первый отрезок, полученный для случайных данных алгоритмом R, будет иметь длину, приближенно равную $(e-1)P \approx 1.718P$ записей, второй—приближенно $(e^2-2e)P \approx 1.952P$, третий—$1.996P$; длина следующих отрезков будет очень близка к $2P$, пока мы не дойдем до последних двух отрезков (см. упр. 14). Стандартное отклонение длины большинства этих отрезков приближенно равно $\sqrt{(4e-10)P} \approx 0.934\sqrt{P}$ [*CACM*, **6** (1963), 685–687].

Кроме этого, согласно упр. 5.1.3-10, *суммарная* длина первых k отрезков будет довольно близка к $\left(2k - \frac{1}{3}\right) P$ со стандартным отклонением $\left(\left(\frac{2}{3}k + \frac{2}{9}\right) P\right)^{1/2}$. Производящие функции $g_1(z, z, \ldots, z)$ и $g_1(1, \ldots, 1, z)$ выводятся в упр. 5.1.3-9 и 11.

В приведенном выше анализе предполагалось, что исходный файл бесконечно длинный, но доказательство теоремы К показывает, что точно такая же вероятность $a_P(l_1, \ldots, l_k)$ получилась бы в случае любой случайной исходной последовательности, содержащей по крайней мере $l_1 + \cdots + l_k + P$ элементов. Следовательно, полученные результаты применимы для файла размера, скажем, $N > (2K+1)P$ в силу малой величины стандартного отклонения.

Мы познакомимся с рядом применений, в которых схема слияния требует, чтобы некоторые отрезки были возрастающими, а некоторые—убывающими. Поскольку остаток, накапливающийся в памяти у конца возрастающего отрезка, имеет тенденцию содержать числа, в среднем меньшие, чем случайные данные, то изменение направления упорядочения уменьшает среднюю длину отрезков. Рассмотрим, например, снегоочиститель, который должен выполнять разворот каждый раз, как он достигает конца прямой дороги; он будет очень быстро передвигаться по только что очищенному участку. В случае изменяемого направления длина отрезков для случайных данных изменяется между $1.5P$ и $2P$ (см. упр. 24).

Упражнения

1. [10] Каким будет шаг 4 в примере четырех путевого слияния в начале этого пункта?
2. [12] Какие изменения произошли бы в дереве рис. 63, если бы ключ 061 был заменен ключом 612?
3. [16] (Э. Ф. Мур.) Что получится в результате применения четырехпутевого выбора с замещением к последовательным словам следующего предложения[2]

   `fourscore and seven years ago our fathers brought forth on this continent a new nation conceived in liberty and dedicated to the proposition that all men are created equal.`

(Используйте обычный алфавитный порядок, рассматривая каждое слово как один ключ.)

4. [16] Примените четырехпутевой *натуральный* выбор к предложению из упр. 3, используя резервуар емкости 4.
5. [00] Верно ли, что выбор с замещением, использующий дерево, работает, только если P есть степень двойки или сумма двух степеней двойки?
6. [15] В алгоритме R указывается, что P должно быть ≥ 2; какие относительно небольшие изменения надо сделать в этом алгоритме, чтобы он правильно работал для всех $P \geq 1$?
7. [17] Что делает алгоритм R в случае отсутствия исходной информации?
8. [20] Алгоритм R использует искусственный ключ "∞", который должен быть больше любого возможного ключа. Покажите, что если бы какой-нибудь реальный ключ оказался равным ∞, то алгоритм мог бы ошибиться, и объясните, как изменить алгоритм в случае, когда реализация "настоящей" бесконечности неудобна.

>9. [23] Как вы изменили бы алгоритм R, чтобы он выводил некоторые заданные отрезки (определяемые `RC`) в возрастающем порядке, а другие в убывающем?

10. [26] Начальная установка указателей `LOSER` на шаге R1 обычно не соответствует никакому действительному турниру, так как внешний узел $P + j$ может не лежать в поддереве с вершиной во внутреннем узле j. Объясните, почему алгоритм R все равно работает. [*Указание.* Будет ли работать алгоритм R, если множеству $\{\,$`LOSER(LOC`$(X[0]))$, $\ldots$, `LOSER(LOC`$(X[P-1]))\,\}$ присваивается на шаге R1 *произвольная* перестановка множества $\{\,$`LOC`$(X[0])$, $\ldots$, `LOC`$(X[P-1])\,\}$?]
11. [M25] Верно ли, что для случайных исходных данных вероятность того, что `KEY(Q)` $<$ `LASTKEY` на шаге R4, приближенно равна $1/2$?
12. [M46] Проведите детальное исследование того, сколько раз выполняется каждая часть алгоритма R; например, как часто выполняется перестановка на шаге R6?
13. [13] Почему второй отрезок, полученный при выборе с замещением, обычно длиннее первого?

>14. [HM25] Используйте аналогию со снегоочистителем, чтобы оценить среднюю длину двух *последних* отрезков, полученных при выборе с замещением, примененном к длинной последовательности исходных данных.

15. [20] Верно ли, что последний отрезок, полученный при выборе с замещением, никогда не содержит более P записей? Обсудите ваш ответ.
16. [M26] Найдите "простое" необходимое и достаточное условие того, что файл $R_1\ R_2\ \ldots\ R_N$ будет полностью упорядочен за один проход P-путевого выбора с замещением. Какова вероятность этого события как функция P и N, если исходными данными служит случайная перестановка множества $\{\,1, 2, \ldots, N\,\}$?
17. [20] Что получается в результате работы алгоритма R, когда исходные ключи представляют собой невозрастающую последовательность $K_1 \geq K_2 \geq \ldots \geq K_N$?

>18. [22] Что произойдет, если вновь применить алгоритм R к файлу, полученному в результате работы алгоритма R?

19. [HM22] Используйте аналогию со снегоочистителем, чтобы доказать, что первый отрезок, полученный при выборе с замещением, имеет длину примерно $(e-1)P$ записей.
20. [HM24] Какую примерно длину имеет первый отрезок, полученный при натуральном выборе, когда $P = P'$?

>21. [HM23] Определите приблизительную длину отрезков, полученных посредством натурального выбора при $P' < P$.

22. [HM40] Целью этого упражнения является определение средней длины отрезков, получаемых при натуральном выборе при $P' > P$. Пусть $\kappa = k + \theta$—действительное число ≥ 1, где $k = \lfloor \kappa \rfloor$, а $\theta = \kappa \bmod 1$, и рассмотрим функцию $F(\kappa) = F_k(\theta)$, где $F_k(\theta)$—полиномы, определяемые производящей функцией

$$\sum_{k \geq 0} F_k(\theta) z^k = e^{-\theta z}/(1 - z e^{1-z}).$$

[2] Восемьдесят семь лет тому назад наши предки основали на этом континенте новую нацию, посвятившую себя делу свободы и убежденную в том, что все люди созданы равными.—*Прим. перев.*

Таким образом, $F_0(\theta) = 1$, $F_1(\theta) = e - \theta$, $F_2(\theta) = e^2 - e - e\theta + \frac{1}{2}\theta^2$ и т. д.

Предположим, что в момент $t = 0$ снегоочиститель начинает моделировать процесс натурального выбора, и допустим, что за T единиц времени позади него упадут ровно P снежинок. В этот момент второй снегоочиститель начинает тот же путь, занимая в момент времени $t + T$ то же положение, что занимал первый снегоочиститель в момент t. В конце концов, к моменту κT позади первого снегоочистителя упадут ровно P' снежинок; он мгновенно очищает остаток дороги и исчезает.

Используя эту модель для интерпретации натурального выбора, покажите, что длина отрезка $e^\theta F(\kappa)P$ получается при

$$P'/P = k + 1 + e^\theta \left(\kappa F(\kappa) - \sum_{0 \le j \le \kappa} F(\kappa - j) \right).$$

23. [BM35] Предыдущее упражнение анализирует натуральный выбор в том случае, когда записи из резервуара всегда читаются в том же порядке, в котором они записывались: "первым включается—первым исключается". Оцените длину отрезков, которая получилась бы, если бы содержимое резервуара, оставшееся от предыдущего отрезка, читалось в совершенно *случайном* порядке, как если бы записи в резервуаре тщательно перемешивались между отрезками.
24. [BM39] Цель этого упражнения—анализ последствий, вызванных случайным изменением направления упорядочения отрезков в выборе с замещением.

a) Пусть $g_P(z_1, z_2, \ldots, z_k)$—та же производящая функция, что и в теореме K, но для каждого из k отрезков определено, является ли он возрастающим или убывающим. Например, мы можем считать, что все отрезки с нечетными номерами возрастающие, а с четными убывающие Покажите, что теорема K справедлива для каждой из 2^k производящих функций такого вида.

b) В силу (a) можно считать $P = 1$. Можно также предположить, что исходной является равномерно распределенная последовательность независимых случайных величин, заключенных между 0 и 1 Пусть

$$a(x, y) = \begin{cases} e^{1-x} - e^{y-x}, & \text{если } x \le y; \\ e^{1-x}, & \text{если } x > y. \end{cases}$$

Пусть $f(x)\,dx$—вероятность того, что определенный возрастающий отрезок начинается с x. Докажите, что $\left(\int_0^1 a(x, y) f(x)\,dx\right) dy$ есть вероятность того, что следующий отрезок начинается с y. [*Указание:* рассмотрите для каждого $n \ge 0$ вероятность того, что $x \le X_1 \le \ldots \le X_n > y$ при данных x и y.]

c) Рассмотрите отрезки, меняющие направление упорядочения с вероятностью p, другими словами, направление каждого отрезка, кроме первого, совпадает с направлением предыдущего отрезка с вероятностью $q = 1 - p$ и противоположно ему с вероятностью p. (Таким образом, если $p = 0$, то все отрезки имеют одинаковое направление; если $p = 1$, направление отрезков чередуется, а при $p = 1/2$ отрезки случайные и независимые) Пусть

$$f_1(x) = 1, \quad f_{n+1}(y) = p \int_0^1 a(x, y) f_n(1 - x)\,dx + q \int_0^1 a(x, y) f_n(x)\,dx.$$

Покажите, что вероятность того, что n-й отрезок начинается с x, есть $f_n(x)\,dx$, если $(n-1)$-й отрезок возрастающий, и $f_n(1 - x)\,dx$, если $(n - 1)$-й отрезок убывающий.

d) Найдите решение f для уравнения "установившегося режима"

$$f(y) = p \int_0^1 a(x, y) f(1 - x)\,dx + q \int_0^1 a(x, y) f(x)\,dx, \quad \int_0^1 f(x)\,dx = 1.$$

[*Указание:* покажите, что $f''(x)$ не зависит от x.]

e) Покажите, что последовательность $f''(x)$ части (c) весьма быстро сходится к функции $f(x)$ части (d).

f) Покажите, что средняя длина возрастающего отрезка, начинающегося с x, равна e^{1-x}.

g) Наконец, объедините все предыдущие результаты и докажите следующую теорему. *Если направления последовательных отрезков при выборе с замещением независимо изменяются на противоположные с вероятностью p, то средняя длина отрезка стремится к* $(6/(3 + p))P$. (Эта теорема при $p = 1$ впервые была доказана Кнутом [*CACM*, **6** (1963), 685–688]; при $p = 1/2$ ее доказал Э. Г. Конхейм в 1978 г.)

25. [BM40]Рассмотрите следующую процедуру.

N1. Прочитать запись, поместив ее в резервуар емкостью в одно слово. Затем прочитать следующую запись `R`, и пусть `K` будет ее ключом.

N2. Вывести содержимое резервуара, установить LASTKEY равным его ключу и опустошить резервуар.
N3. Если K < LASTKEY, то вывести R, установить LASTKEY ← K и перейти к N5.
N4. Если резервуар не пуст, вернуться к N2; в противном случае поместить R в резервуар.
N5. Прочитать новую запись R и установить K равным ее ключу. Перейти к N3. ∎

Эта процедура, в сущности, эквивалентна натуральному выбору с $P = 1$ и $P' = 1$ или $P' = 2$ (в зависимости от того, в какой момент мы опустошаем резервуар—как только он заполнится или когда нам надо будет записать. в заполненный резервуар новый элемент, переполняющий его), за исключением того, что эта процедура порождает *убывающие* отрезки и никогда не останавливается. Эти отклонения не приносят вреда, они удобны для нашей цели.

Действуя, как в упр. 24, обозначим через $f_n(x, y)\,dy\,dx$ вероятность того, что (x, y) суть значения (LASTKEY, K) соответственно сразу же после n-го выполнения шага N2. Докажите, что существует функция $g_n(x)$ от одной переменной, такая, что $f_n(x, y) = g_n(x)$, если $x < y$, и $f_n(x, y) = g_n(x) - e^{-y}(g_n(x) - g_n(y))$, если $x > y$. Функция $g_n(x)$ определяется соотношениями $g_1(x) = 1$,

$$g_{n+1}(x) = \int_0^x e^u g_n(u)\,du + \int_0^x dv\,(v+1)\int_v^1 du\,((e^v-1)g_n(u)+g_n(v)) + + x\int_x^1 dv\int_v^1 du\,((e^v-1)g_n(u)+g_n(v)).$$

Покажите далее, что ожидаемая длина n-го отрезка равна

$$\int_0^1 dx\int_0^x dy(g_n(x)(e^y-1)+g_n(y))\left(2-\frac{1}{2}y^2\right)+\int_0^1 dx\,(1-x)g_n(x)e^x.$$

[*Замечание.* Решение этого уравнения в установившемся режиме оказывается очень сложным; оно было численно найдено Дж. Мак-Кенной. Он показал, что длина отрезка стремится к предельному значению 2.61307209. Теорема K не применима к натуральному выбору, так что случай $P = 1$ нельзя распространить на другие P.]

26. [M33] Рассматривая алгоритм упр. 25 как определение натурального выбора для $P' = 1$, найдите среднюю длину *первого* отрезка для $P' = r$ при любом $r \geq 0$ по следующей схеме:

a) Покажите, что первый отрезок имеет длину n с вероятностью

$$(n+r)\left[\begin{matrix} n+r \\ n \end{matrix}\right] \Big/ (n+r+1)!.$$

b) Определим "числа Стирлинга второго порядка" $\left\{\begin{matrix} n \\ m \end{matrix}\right\}$ правилами

$$\left\{\begin{matrix} 0 \\ m \end{matrix}\right\} = \delta_{m0}, \quad \left\{\begin{matrix} n \\ m \end{matrix}\right\} = (n+m-1)\left(\left\{\begin{matrix} n-1 \\ m \end{matrix}\right\} + \left\{\begin{matrix} n-1 \\ m-1 \end{matrix}\right\}\right) \qquad \text{при } n > 0.$$

Докажите, что

$$\left[\begin{matrix} n+r \\ n \end{matrix}\right] = \sum_{0\leq k\leq r} \binom{n+r}{k+r}\left\{\begin{matrix} r \\ k \end{matrix}\right\}.$$

c) Докажите, что средняя длина первого отрезка будет, следовательно, $c_r e - r - 1$, где

$$c_r = \sum_{0\leq k\leq r} \left[\left[\begin{matrix} r \\ k \end{matrix}\right]\right] (r+k+1)/(r+k)!.$$

27. [25] В тексте рассматривается только случай сортировки записей фиксированного размера. Как разумным образом приспособить выбор с замещением к записям *переменной длины?*

Многофазное слияние Теперь, после того как мы выяснили, как можно образовать начальные отрезки, рассмотрим различные методы распределения отрезков по лентам и слияния их до тех пор, пока не получится единственный отрезок.

Предположим сначала, что в нашем распоряжении имеются три ленточных устройства: $T1$, $T2$ и $T3$; можно воспользоваться сбалансированным слиянием, описанным в начале § 5.4, для $P = 2$ и $T = 3$. Оно принимает следующий вид:

B1. Распределить начальные отрезки попеременно на ленты $T1$ и $T2$.

B2. Слить отрезки с лент $T1$ и $T2$ на $T3$; затем остановиться, если $T3$ содержит только один отрезок.

B3. Скопировать отрезки с $T3$ попеременно на $T1$ и $T2$, затем вернуться к шагу B2. ∎

Если начальное распределение дало 5 отрезков, то первый проход слияния приведет к $\lceil S/2\rceil$ отрезкам на $T3$, второй—к $\lceil S/4\rceil$ и т. д. Таким образом, если, скажем, $17 \le S \le 32$, то произойдет 1 проход распределения, 5 проходов слияния и 4 прохода копирования; в общем случае при $S > 1$ число проходов по всем данным будет равно $2\lceil \log_2 S\rceil$.

Проходы копирования в этой процедуре нежелательны, так как они не уменьшают числа отрезков. Можно обойтись половиной копирований, если использовать *двухфазную* процедуру:

A1. Распределить начальные отрезки попеременно на ленты $T1$ и $T2$.

A2. Слить отрезки с лент $T1$ и $T2$ на $T3$; остановиться, если $T3$ содержит только один отрезок.

A3. Скопировать *половину* отрезков с $T3$ на $T1$.

A4. Слить отрезки с лент $T1$ и $T3$ на $T2$; остановиться, если $T2$ содержит только один отрезок.

A5. Скопировать *половину* отрезков с $T2$ на $T1$. Вернуться к шагу A2. ∎

Число проходов по всем данным сократилось до $\frac{3}{2}\lceil \log_2 S\rceil + \frac{1}{2}$, так как в шагах A3 и A5 выполняется только "половина прохода", т. е. сэкономлено около 25% времени.

В действительности можно даже полностью устранить копирование, если начать с F_n отрезков на ленте T1 и с F_{n-1} отрезков на T2, где F_n и F_{n-1}—последовательные числа Фибоначчи. Рассмотрим, например, случай $n = 7$, $S = F_n + F_{n-1} = 13 + 8 = 21$:

	Содержимое T1	Содержимое T2	Содержимое T3	Примечания
Фаза 1.	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1	1, 1, 1, 1, 1, 1, 1, 1		Начальное распределение
Фаза 2.	1, 1, 1, 1, 1	–	2, 2, 2, 2,2, 2,2,2	Слияние 8 отрезков на T3
Фаза 3.	–	3, 3, 3, 3, 3	2,2,2	Слияние 5 отрезков на T3
Фаза 4.	5, 5, 5	3, 3	–	Слияние 3 отрезков на T1
Фаза 5.	5	–	8,8	Слияние 2 отрезков на T3
Фаза 6.	–	13	8	Слияние 1 отрезка на T2
Фаза 7.	21	–	–	Слияние 1 отрезка на T1

Здесь "2, 2, 2, 2, 2, 2, 2, 2", например, обозначает восемь отрезков относительной длины 2, если считать относительную длину каждого начального отрезка равной 1. Всюду в этой таблице числа Фибоначчи!

Полный проход по данным осуществляют только фазы 1 и 7; фаза 2 обрабатывает лишь 16/21 общего числа начальных отрезков, фаза 3—лишь 15/21 и т. д.; таким образом, суммарное число "проходов" равно $(21+16+15+15+16+13+21)/21 = 5\frac{4}{7}$, если предположить, что начальные отрезки имеют примерно равную длину. Для сравнения заметим, что рассмотренная выше двухфазная процедура затратила бы 8 проходов на сортировку этих же начальных отрезков. Мы увидим, что в общем случае эта схема Фибоначчи требует приблизительно $1.04 \log_2 S + 0.99$ проходов, что делает ее сравнимой с *четырехленточным* сбалансированным слиянием, хотя она использует только три ленты.

Эту идею можно обобщить на случай T лент при любом $T \ge 3$, используя $(T-1)$-путевое слияние. Мы увидим, например, что в случае четырех лент требуется только около $0.703 \log_2 S + 0.96$ проходов по данным. Обобщенная схема использует обобщенные числа Фибоначчи. Рассмотрим следующий пример с шестью лентами:

	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	Число обрабатываемых начальных отрезков
Фаза 1.	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	–	$31+30+28+24+16 = 129$
Фаза 2.	1^{15}	1^{14}	1^{12}	1^{8}	–	5^{16}	$16 \times 5 = 80$
Фаза 3.	1^{7}	1^{6}	1^{4}	–	9^{8}	5^{8}	$8 \times 9 = 72$
Фаза 4.	1^{3}	1^{2}	–	17^{4}	9^{4}	5^{4}	$4 \times 17 = 68$
Фаза 5.	1^{1}	–	33^{2}	17^{2}	9^{2}	5^{2}	$2 \times 33 = 66$
Фаза 6.	–	65^{1}	33^{1}	17^{1}	9^{1}	5^{1}	$1 \times 65 = 65$
Фаза 7.	129^{1}	–	–	–	–	–	$1 \times 129 = 129$

Здесь 1^{31} обозначает 31 отрезок относительной длины 1 и т.д.; везде используется пятипутевое слияние. Эта общая схема была разработана Р. Л. Гилстэдом [Proc. AFIPS Eastern Jt. Computer Conf., 18 (1960), 143–148], который назвал ее многофазным слиянием. Случай трех лент был ранее открыт Б. К. Бетцем [неопубликованная заметка, Minneapolis-Honeywell Regulator Co. (1956)].

Чтобы заставить многофазное слияние работать, как в предыдущем примере, необходимо после каждой фазы иметь "точное фибоначчиево распределение" отрезков по лентам. Читая приведенную

выше таблицу снизу вверх, можно заметить, что первые семь точных фибоначчиевых распределений при $T = 6$ суть $\{1,0,0,0,0\}$, $\{1,1,1,1,1\}$, $\{2,2,2,2,1\}$, $\{4,4,4,3,2\}$, $\{8,8,7,6,4\}$, $\{16,15,14,12,8\}$ и $\{31,30,28,24,16\}$. Теперь перед нами стоят следующие важные вопросы:

1) Какое правило скрыто за этими точными фибоначчиевыми распределениями?
2) Что делать, если S не соответствует точному фибоначчиевому распределению?
3) Как построить начальный проход распределения, чтобы он порождал нужное расположение отрезков на лентах?
4) Сколько "проходов" по данным потребует T-ленточное многофазное слияние (как функция от S—числа начальных отрезков)?

Мы обсудим эти четыре вопроса по очереди, при этом сначала дадим "простые ответы", а затем займемся более глубоким анализом.

Точные фибоначчиевы распределения можно получить, "прокручивая" рассмотренную схему в обратную сторону, циклически переставляя содержимое лент. Например, при $T = 6$ имеем следующее распределение отрезков:

Уровень	$T1$	$T2$	$T3$	$T4$	$T5$	Сумма	Лента с окончательным результатом
0	1	0	0	0	0	1	$T1$
1	1	1	1	1	1	5	$T6$
2	2	2	2	2	1	9	$T5$
3	4	4	4	3	2	17	$T4$
4	8	8	7	6	4	33	$T3$
5	16	15	14	12	8	65	$T2$
6	31	30	28	24	16	129	$T1$
7	61	59	55	47	31	253	$T6$
8	120	116	108	92	61	497	$T5$
. . . . .							
n	a_n	b_n	c_n	d_n	e_n	t_n	$T(k)$
n+1	$a_n + b_n$	$a_n + c_n$	$a_n + d_n$	$a_n + e_n$	a_n	$t_n + 4a_n$	$T(k-1)$
. . . . .							

(1)

(После начального распределения лента T6 всегда будет пустой.)

Из правила перехода от уровня n к уровню $n+1$ ясно, что условия

$$a_n \ge b_n \ge c_n \ge d_n \ge e_n \tag{2}$$

выполняются на любом уровне. В самом деле, легко видеть из (1), что

$$\begin{aligned} e_n &= a_{n-1}, \\ d_n &= a_{n-1} + e_{n-1} = a_{n-1} + a_{n-2}, \\ c_n &= a_{n-1} + d_{n-1} = a_{n-1} + a_{n-2} + a_{n-3}, \\ b_n &= a_{n-1} + c_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4}, \\ a_n &= a_{n-1} + b_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4} + a_{n-5}, \end{aligned} \tag{3}$$

где $a_0 = 1$ и где мы полагаем $a_n = 0$ при $n = -1, -2, -3, -4$. *Числа Фибоначчи p-го порядка* $F_n^{(p)}$ определяются правилами

$$\begin{aligned} F_n^{(p)} &= F_{n-1}^{(p)} + F_{n-2}^{(p)} + \cdots + F_{n-p}^{(p)} \qquad \text{при } n \ge p; \\ F_n^{(p)} &= 0 \qquad \text{при } 0 \le n \le p-2; \\ F_{p-1}^{(p)} &= 1. \end{aligned} \tag{4}$$

Другими словами, мы начинаем с $p-1$ нулей, затем пишем 1, а каждое следующее число является суммой p предыдущих чисел. При $p = 2$ это обычная последовательность Фибоначчи; для больших значений p эту последовательность впервые изучил, по-видимому, В. Шлегель в [*El Progreso Matematico*, **4** (1894), 173–174]. Шлегель вывел производящую функцию

$$\sum_{n\ge 0} F_n^{(p)} z^n = \frac{z^{p-1}}{1 - z - z^2 - \cdots - z^p} = \frac{z^{p-1} - z^p}{1 - 2z + z^{p+1}}. \tag{5}$$

Формула (3) показывает, что число отрезков на T1 в процессе шестиленточного многофазного слияния является числом Фибоначчи пятого порядка $a_n = F^{(5)}_{n+4}$.

В общем случае, если положить $P = T - 1$, распределения в многофазном слиянии для T лент будут аналогичным образом соответствовать числам Фибоначчи P-го порядка. В точном распределении n-го уровня на k-й ленте будет

$$F^{(P)}_{n+P-2} + F^{(P)}_{n+P-3} + \cdots + F^{(P)}_{n+k-2}$$

начальных отрезков для $1 \le k \le P$, а общее количество начальных отрезков на всех лентах будет, следовательно, равно

$$t_n = PF^{(P)}_{n+P-2} + (P-1)F^{(P)}_{n+P-3} + \cdots + F^{(P)}_{n-1}. \tag{6}$$

Это решает вопрос о "точном фибоначчиевом распределении". Но что мы должны делать, если S не равно в точности t_n ни при каком n? Как первоначально поместить отрезки на ленты?

Если S не является точным числом Фибоначчи (а чисел Фибоначчи не так уж много), то можно действовать, как в сбалансированном P-путевом слиянии, добавляя "фиктивные

Picture: Рис. 68. Сортировка многофазным слиянием.

отрезки"; поэтому можно считать, что S, в конце концов, будет точным. Есть несколько способов добавления фиктивных отрезков; мы еще не знаем, как это сделать "наилучшим" способом. В первую очередь рассмотрим метод распределения отрезков и приписывания фиктивных отрезков, который хотя и не самый оптимальный, но зато достаточно простой и, по-видимому, лучше всех других методов такой же степени сложности.

Алгоритм D. *(Сортировка многофазным слиянием с использованием "горизонтального" распределения.)* Этот алгоритм берет начальные отрезки и распределяет их один за другим по лентам, пока запас начальных отрезков не исчерпается. Затем он определяет, как надо сливать ленты, используя P-путевое слияние, в предположении, что имеются $T = P + 1 \ge 3$ лентопротяжных устройств. Ленту T можно использовать для хранения ввода, так как на нее не записывается ни одного начального отрезка. В памяти хранятся следующие таблицы:

$\texttt{A}[j]$, $1 \le j \le T$: Точное фибоначчиевое распределение, к которому мы стремимся.
$\texttt{D}[j]$, $1 \le j \le T$: Число фиктивных отрезков, которые считаются присутствующими в начале ленты на логическом устройстве с номером j.
$\texttt{TAPE}[j]$, $1 \le j \le T$: Номер физического лентопротяжного устройства, соответствующего логическому устройству с номером j.

(Оказалось, что удобно работать с "номерами логических лентопротяжных устройств", соответствие которых физическим устройствам меняется в процессе выполнения алгоритма.)

D1 [Начальная установка.] Установить $\texttt{A}[j] \leftarrow \texttt{D}[j] \leftarrow 1$ и $\texttt{TAPE}[j] \leftarrow j$ при $1 \le j < T$. Установить $\texttt{A}[T] \leftarrow \texttt{D}[T] \leftarrow 0$ и $\texttt{TAPE}[T] \leftarrow T$. Затем установить $l \leftarrow 1$, $j \leftarrow 1$.

D2 [Ввод на ленту j.] Записать один отрезок на ленту j и уменьшить $\texttt{D}[j]$ на 1. Затем, если ввод исчерпан, перемотать все ленты и перейти к шагу **D5**.

D3 [Продвижение j.] Если $\texttt{D}[j] < \texttt{D}[j+1]$, то увеличить j на 1 и вернуться к шагу **D2**. В противном случае, если $\texttt{D}[j] = 0$, перейти к шагу **D4**, а если $\texttt{D}[j] \ne 0$, установить $j \leftarrow 1$ и вернуться к шагу **D2**.

D4 [Подняться на один уровень.] Установить $l \leftarrow l+1$, $a \leftarrow \texttt{A}[1]$, затем для $j = 1, 2, \ldots, P$ (именно в этом порядке) установить $\texttt{D}[j] \leftarrow a + \texttt{A}[j+1] - \texttt{A}[j]$ и $\texttt{A}[j] \leftarrow a + \texttt{A}[j+1]$. (См. (1). Отметим, что $\texttt{A}[P+1]$ всегда 0. В этом месте будем иметь $\texttt{D}[1] > \texttt{D}[2] > \ldots > \texttt{D}[T]$.) Теперь установить $j \leftarrow 1$ и вернуться к шагу **D2**.

D5 [Слияние.] Если $l = 0$, то сортировка завершена, результат находится на $\texttt{TAPE}[1]$. В противном случае сливать отрезки с лент $\texttt{TAPE}[1], \ldots, \texttt{TAPE}[P]$ на $\texttt{TAPE}[T]$ до тех пор, пока $\texttt{TAPE}[P]$ не станет пустой и $\texttt{D}[P]$ не обратится в 0. Процесс слияния для каждого сливаемого отрезка должен протекать следующим образом. Если $\texttt{D}[j] > 0$ для всех j, $1 \le j \le P$, то увеличить $\texttt{D}[T]$ на 1 и уменьшить каждое $\texttt{D}[j]$ на 1 для $1 \le j \le P$; в противном случае сливать по одному отрезку с каждой ленты $\texttt{TAPE}[j]$, такой, что $\texttt{D}[j] = 0$, и уменьшить $\texttt{D}[j]$ на 1 для остальных j. (Таким образом, считается, что фиктивные отрезки находятся в *начале* ленты, а не в конце.)

D6 [Опуститься на один уровень.] Установить $l \leftarrow l - 1$. Перемотать ленты $\texttt{TAPE}[P]$ и $\texttt{TAPE}[T]$. (В действительности перемотка $\texttt{TAPE}[P]$ могла быть начата на шаге **D5** после ввода с нее последнего блока.) Затем установить $(\texttt{TAPE}[1], \texttt{TAPE}[2], \ldots, \texttt{TAPE}[T]) \leftarrow (\texttt{TAPE}[T], \texttt{TAPE}[1], \ldots, \texttt{TAPE}[T-1])$, $(\texttt{D}[1], \texttt{D}[2], \ldots, \texttt{D}[T]) \leftarrow (\texttt{D}[T], \texttt{D}[1], \ldots, \texttt{D}[T-1])$ и вернуться к шагу **D5**. ∎

Правило распределения, которое так лаконично сформулировано в шаге D3 этого алгоритма, стремится по возможности уравнять числа фиктивных отрезков на каждой ленте. Рисунок 69 иллюстрирует порядок распределения, когда мы переходим от уровня 4 (33 отрезка) к уровню 5 (65 отрезков) в сортировке с шестью лентами; если было бы, скажем, только 53 начальных

Picture: Рис. 69. Порядок, в котором отрезки с 34-го по 65-й распределяются на ленты при переходе с уровня 4 на уровень 5. (См. таблицу точных распределений на стр. 320.) Заштрихованные области соответствуют первым 33 отрезкам, которые были распределены к моменту достижения уровня 4.

отрезка, то все отрезки с номерами 54 и выше рассматривались бы как фиктивные. (На самом деле отрезки записываются в конец ленты, но удобнее считать, что они записываются в начало, так как предполагается, что фиктивные отрезки находятся в начале ленты.)

Мы уже рассмотрели первые три из поставленных выше вопросов, осталось выяснить число "проходов" по данным. Сравнивая наш шестиленточный пример с таблицей (1), мы видим, что суммарное количество обработанных начальных отрезков при $S = t_6$ есть $a_5t_1 + a_4t_2 + a_3t_3 + a_2t_4 + a_1t_5 + a_0t_6$, если исключить начальный проход распределения. В упр. 4 выводятся производящие функции

$$a(z) = \sum_{n\ge 0} a_n z^n = \frac{1}{1 - z - z^2 - z^3 - z^4 - z^5},$$
$$t(z) = \sum_{n\ge 1} t_n z^n = \frac{5z + 4z^2 + 3z^3 + 2z^4 + z^5}{1 - z - z^2 - z^3 - z^4 - z^5}. \qquad (7)$$

Отсюда следует, что в общем случае число обрабатываемых начальных отрезков при $S = t_n$ равно коэффициенту при z^n в произведении $a(z)\cdot t(z)$ плюс t_n (это дает начальный проход распределения). Теперь мы можем вычислить асимптотическое поведение многофазного слияния, как показано в упр. 5–7, и получаем результаты, приведенные в табл. 1.

В табл. 1 "отношение роста" есть предел $\lim_{n\to\infty} t_{n+1}t_n$, показывающий, во сколько приблизительно раз возрастает число

Таблица 1

Аппроксимация поведения сортировки многофазным слиянием

Ленты	Фазы	Проходы	Проходы/фазы, в процентах	Отношение роста
3	$2.078\ln S + 0.678$	$1.504\ln S + 0.992$	72	1.6180340
4	$1.641\ln S + 0.364$	$1.015\ln S + 0.965$	62	1.8392868
5	$1.524\ln S + 0.078$	$0.863\ln S + 0.921$	57	1.9275620
6	$1.479\ln S - 0.185$	$0.795\ln S + 0.864$	54	1.9659482
7	$1.460\ln S - 0.424$	$0.762\ln S + 0.797$	52	1.9835826
8	$1.451\ln S - 0.642$	$0.744\ln S + 0.723$	51	1.9919642
9	$1.447\ln S - 0.838$	$0.734\ln S + 0.646$	51	1.9960312
10	$1.445\ln S - 1.017$	$0.728\ln S + 0.568$	50	1.9980295
20	$1.443\ln S - 2.170$	$0.721\ln S - 0.030$	50	1.9999981

отрезков на каждом уровне. "Проходы" обозначают среднее количество обработок каждой записи, а именно $(1/S)$, умноженное на общее число начальных отрезков, обрабатываемых в течение фаз распределения и слияния. Установленные числа проходов и фаз справедливы в каждом случае с точностью до $O(S^{-\varepsilon})$ при некотором $\varepsilon > 0$ для точного распределения при $S \to \infty$.

На рис. 70 изображены в виде функций от S средние количества слияний каждой записи при использовании алгоритма D в случае неточных чисел. Заметим, что при использовании трех лент как раз после точных распределений появляются "пики" относительной неэффективности, но это явление в значительной степени исчезает при четырех или большем числе лент. Использование восьми или более лент дает сравнительно малое улучшение по сравнению с шестью или семью лентами.

***Более детальное рассмотрение.** В сбалансированном слиянии, требующем k проходов, каждая запись обрабатывается в ходе сортировки ровно k раз. Но многофазная процедура не является такой беспристрастной: некоторые записи могут обрабатываться

Picture: Рис. 70. Эффективность многофазного слияния, использующего алгоритм D.

много большее число раз, чем другие, и мы можем увеличить скорость, если условимся помещать фиктивные отрезки в часто обрабатываемые позиции.

По этой причине изучим более подробно многофазное распределение. Вместо того чтобы интересоваться только числом отрезков на каждой ленте, как в (1), припишем каждому отрезку его *число слияний*—сколько раз он обрабатывается в течение всего процесса сортировки. Вместо (1) получим следующую таблицу:

Уровень	$T1$	$T2$	$T3$	$T4$	$T5$
0	0	–	–	–	–
1	1	1	1	1	1
2	21	21	21	21	2
3	3221	3221	3221	322	32
4	43323221	43323221	4332322	433232	4332
5	5443433243323221	544343324332322	54434332433232	544343324332	54434332
. . . . .					
n	A_n	B_n	C_n	D_n	E_n
$n+1$	$(A_n+1)B_n$	$(A_n+1)C_n$	$(A_n+1)D_n$	$(A_n+1)E_n$	A_n+1
. . . . .					

(8)

Здесь A_n есть цепочка из a_n значений, представляющих числа слияний каждого отрезка на $T1$, если мы начинаем с распределения n-го уровня; B_n есть соответствующая цепочка для T2 и т. д. Обозначение "$(A_n+1)B_n$" читается: "A_n, все значения которой увеличены на 1, а за нею B_n".

Рисунок 71 (a), на котором "снизу вверх" изображены A_5, B_5, C_5, D_5, E_5, демонстрирует, каким образом числа слияний для каждого отрезка появляются на ленте; заметим, что отрезок в начале любой ленты будет обрабатываться 5 раз, в то время как отрезок в конце T1 будет обрабатываться лишь однажды.

Picture: Рис. 71. Анализ многофазного распределения пятого уровня на шести лентах: (a)—числа слияний; (b)—оптимальный порядок распределения.

Эта "дискриминация" при многофазном слиянии приводит к тому, что фиктивные отрезки лучше помещать в начало ленты, а не в конец. На рис. 71 (b) представлен оптимальный порядок распределения отрезков для случая пятиуровневого многофазного слияния; каждый новый отрезок помещается в позицию с наименьшим из оставшихся числом слияний. Заметим, что алгоритм D (рис. 69) несколько хуже, так как он заполняет некоторые позиции "4" до того, как заполнены все позиции "3". Рекуррентные соотношения (8) показывают, что все B_n, C_n, D_n, E_n являются начальными подцепочками A_n. В действительности, используя (8), можно вывести формулы

$$\begin{aligned} E_n &= (A_{n-1}) + 1, \\ D_n &= (A_{n-1}A_{n-2}) + 1, \\ C_n &= (A_{n-1}A_{n-2}A_{n-3}) + 1, \\ B_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}) + 1, \\ A_n &= (A_{n-1}A_{n-2}A_{n-3}A_{n-4}A_{n-5}) + 1, \end{aligned} \tag{9}$$

обобщающие соотношения (3), которые имеют дело только с длинами этих цепочек. Кроме того, из правил, определяющих цепочки A, следует, что структура в начале каждого уровня, в сущности, одна и та же; имеем

$$A_n = n - Q_n, \tag{10}$$

где Q_n есть цепочка из a_n значений, определяемая законом

$$\begin{aligned} Q_n = Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4) \qquad &\text{при } n \ge 1, \\ Q_0 = \text{'0'}; Q_n = (\text{пусто}) \qquad &\text{при } n < 0. \end{aligned} \tag{11}$$

Так как Q_n начинается с Q_{n-1}, то можно рассмотреть *бесконечную* цепочку Q_∞, первые a_n элементов которой совпадают с Q_n; эта цепочка, по существу, описывает все числа слияний в многофазном распределении. В случае шести лент имеем

$$Q_\infty = 01121223122323341223233423343441223233423343445233434453445454122 32\ldots. \tag{12}$$

В упр. 11 содержится интересная интерпретация этой цепочки.

При условии что A_n есть цепочка $m_1 m_2 \ldots m_{a_n}$, обозначим через $A_n(x) = x^{m_1} + x^{m_2} + \cdots + x^{m_{a_n}}$ соответствующую производящую функцию, описывающую, сколько раз появляется каждое число слияний;

аналогично введем $B_n(x)$, $C_n(x)$, $D_n(x)$, $E_n(x)$. Например, $A_4(x) = x^4 + x^3 + x^3 + x^2 + x^3 + x^2 + x^2 + x = x^4 + 3x^3 + 3x^2 + x$. В силу соотношений (9) имеем при $n \geq 1$

$$\begin{aligned}
E_n(x) &= x(A_{n-1}(x)),\\
D_n(x) &= x(A_{n-1}(x) + A_{n-2}(x)),\\
C_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x)),\\
B_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x)),\\
A_n(x) &= x(A_{n-1}(x) + A_{n-2}(x) + A_{n-3}(x) + A_{n-4}(x) + A_{n-5}(x)),
\end{aligned} \tag{13}$$

где $A_0(x) = 1$ и $A_n(x) = 0$ при $n = -1, -2, -3, -4$. Следовательно,

$$\sum_{n\geq 0} A_n(x)z^n = \frac{1}{1 - x(z + z^2 + z^3 + z^4 + z^5)} = \sum_{k\geq 0} x^k(z + z^2 + z^3 + z^4 + z^5)^k. \tag{14}$$

Рассматривая отрезки на всех лентах, положим

$$T_n(x) = A_n(x) + B_n(x) + C_n(x) + D_n(x) + E_n(x), \qquad n \geq 1; \tag{15}$$

из (13) немедленно получаем

$$T_n(x) = 5A_{n-1}(x) + 4A_{n-2}(x) + 3A_{n-3}(x) + 2A_{n-4}(x) + A_{n-5}(x),$$

а значит, и

$$\sum_{n\geq 1} T_n(x)z^n = \frac{x(5z + 4z^2 + 3z^3 + 2z^4 + z^5)}{1 - x(z + z^2 + z^3 + z^4 + z^5)}. \tag{16}$$

Соотношение (16) показывает, что легко вычислить коэффициенты $T_n(x)$:

	z	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}
x	5	4	3	2	1	0	0	0	0	0	0	0	0	0
x^2	0	5	9	12	14	15	10	6	3	1	0	0	0	0
x^3	0	0	5	14	26	40	55	60	57	48	35	20	10	4
x^4	0	0	0	5	19	45	85	140	195	238	260	255	220	170
x^5	0	0	0	0	5	24	69	154	294	484	703	918	1088	1168

(17)

Столбцы этой таблицы дают $T_n(x)$; например, $T_4(x) = 2x + 12x^2 + 14x^3 + 5x^4$. Каждый элемент этой таблицы (кроме элементов первой строки) является суммой пяти элементов, расположенных в предыдущей строке непосредственно левее него.

Число отрезков в точном распределении n-го уровня равно $T_n(1)$, а общее количество обрабатываемых отрезков в процессе их слияния равно производной $T_n'(1)$. Далее,

$$\sum_{n\geq 1} T_n'(x)z^n = \frac{5z + 4z^2 + 3z^3 + 2z^4 + z^5}{(1 - x(z + z^2 + z^3 + z^4 + z^5))^2}; \tag{18}$$

полагая $x = 1$ в (16) и (18), получаем, что число слияний для точного распределения n-го уровня есть коэффициент при z^n в $a(z)t(z)$ [ср. (7)]. Это согласуется с нашими предыдущими рассуждениями.

Функции $T_n(x)$ можно использовать для определения совершаемой работы, когда фиктивные отрезки добавляются оптимальным образом. Пусть $\sum_n(m)$ есть сумма наименьших m чисел слияний в распределении n-го уровня. Посмотрев на столбцы (17), мы без труда вычислим эти суммы $\sum_n(m)$:

$m =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$n = 1$	1	2	3	4	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n = 2$	1	2	3	4	6	8	10	12	14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n = 3$	1	2	3	5	7	9	11	13	15	17	19	21	24	27	30	33	36	∞	∞	∞	∞
$n = 4$	1	2	4	6	8	10	12	14	16	18	20	22	24	26	29	32	35	38	41	44	47
$n = 5$	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	32	35	38	41	44	47
$n = 6$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	33	36	38	42	45	48
$n = 7$	2	4	6	8	10	12	14	16	18	20	23	26	29	32	35	38	41	44	47	50	53

(19)

Например, если мы хотим отсортировать 17 отрезков, используя распределение 3-го уровня, то общее количество операций есть $\sum_3(17) = 36$, но если использовать распределение 4-го или 5-го уровня, то

общее количество операций в процессе слияния будет только $\sum_4(17) = \sum_5(17) = 35$. Выгоднее использовать уровень 4, хотя число 17 соответствует точному распределению 3-го уровня! В самом деле, по мере возрастания S оптимальный номер уровня оказывается значительно больше, чем используемый в алгоритме D.

Упражнение 14 показывает, что существует неубывающая последовательность чисел M_n, такая, что уровень n оптимален для $M_n \le S < M_{n+1}$, но не для $S \ge M_{n+1}$. В случае шести лент только что вычисленная таблица $\sum_n(m)$ дает

$$M_0 = 0, \quad M_1 = 2, \quad M_2 = 6, \quad M_3 = 10, \quad M_4 = 14.$$

Выше мы имели дело только со случаем шести лент, однако ясно, что те же идеи применимы к многофазной сортировке с T лентами для любого $T \ge 3$; просто в соответствующих местах надо заменить 5 на $P = T - 1$. В табл. 2 изображены последовательности M_n, полученные для различных значений T. Таблица 3 и рис. 72 дают представление об общем количестве обрабатываемых начальных отрезков после выполнения оптимального распределения фиктивных отрезков. (Формулы внизу табл. 3 следует принимать с осторожностью, так как это приближение по методу наименьших квадратов на области $1 \le S \le 5000$ ($1 \le S \le 10\,000$ при $T = 3$), что приводит к некоторому отклонению, поскольку данная область значений S не является одинаково подходящей для всех T. При $S \to \infty$ число обрабатываемых начальных отрезков после оптимального многофазного распределения асимптотически равно $S \log_p S$, но сходимость к этому асимптотическому пределу крайне медленная.)

При помощи табл. 4 можно сравнить метод распределения алгоритма D с результатами оптимального распределения, приведенными в табл. 3. Ясно, что алгоритм D не очень близок к оптимальному при больших S и T; однако непонятно, можно ли поступить в этих случаях существенно лучше алгоритма D, не прибегая к значительным усложнениям, особенно если мы не знаем S заранее. К счастью, заботиться о больших S приходится довольно редко (см. п. 5.4.6), так что алгоритм D на практике не так уж плох, на самом деле—даже весьма неплох.

Математически многофазная сортировка впервые была проанализирована У. К. Картером [Proc. IFIP Congress (1962), 62–66].

Picture: Рис. 72. Эффективность многофазного слияния с оптимальным начальным распределением (ср. с рис. 70).

Многие из приведенных результатов относительно оптимального размещения фиктивных отрезков принадлежат Б. Сэкману и Т. Синглеру [A vector model for merge sort analisis, неопубликованный доклад, представленный на симпозиум ACM по сортировке (ноябрь 1962), стр. 21]. Позднее Сэкман предложил горизонтальный метод распределения, используемый в алгоритме D; Дональд Шелл [*CACM*, **14** (1971), 713–719; **15** (1972), 28], независимо развив эту теорию, указал на соотношение (10) и подробно изучил несколько различных алгоритмов распределения. Дальнейшие полезные усовершенствования и упрощения были получены Дереком Э. Зэйвом [*JACM*, будет опубликовано].

Таблица 2

Число отрезков, при котором данный уровень оптимален

Уровень	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	2	2	M_1
2	3	4	5	6	7	8	9	10	M_2
3	4	6	8	10	12	14	16	18	M_3
4	6	10	14	14	17	20	23	26	M_4
5	9	18	23	29	20	24	28	32	M_5
6	14	32	35	43	53	27	32	37	M_6
7	22	55	76	61	73	88	35	41	M_7
8	35	96	109	194	98	115	136	44	M_8
9	56	173	244	216	283	148	171	199	M_9
10	90	280	359	269	386	168	213	243	M_{10}
11	145	535	456	779	481	640	240	295	M_{11}
12	234	820	1197	1034	555	792	1002	330	M_{12}
13	378	1635	1563	1249	1996	922	1228	1499	M_{13}
14	611	2401	4034	3910	2486	1017	1432	1818	M_{14}
15	988	4959	5379	4970	2901	4397	1598	2116	M_{15}
16	1598	7029	6456	5841	10578	5251	1713	2374	M_{16}
17	2574	14953	18561	19409	13097	5979	8683	2576	M_{17}
18	3955	20583	22876	23918	15336	6499	10069	2709	M_{18}
19	6528	44899	64189	27557	17029	30164	11259	15787	M_{19}

Таблица 3

Число начальных отрезков, обрабатываемых при оптимальном многофазном слиянии

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
10	36	24	19	17	15	14	13	12	
20	90	60	49	44	38	36	34	33	
50	294	194	158	135	128	121	113	104	
100	702	454	362	325	285	271	263	254	
1000	10371	6680	5430	4672	4347	3872	3739	3632	
5000	63578	41286	32905	28620	26426	23880	23114	22073	
$S\{$	(1.51	0.951	0.761	0.656	0.589	0.548	0.539	0.488) $\cdot S\ln S$	
$\{$	$+$(−.11	$+$.14	$+$.16	$+$.19	$+$.21	$+$.20	$+$.02	$+$.18) $\cdot S$	

Производящая функция (16) была впервые исследована У. Буржем [Proc. IFIP Congress (1971), I, 454–459].

А как обстоит дело с временем перемотки? До сих пор мы использовали "обрабатываемые начальные отрезки" как единственную меру эффективности для сравнения стратегий ленточного слияния. Но после каждой из фаз 2–6 в примерах в начале этого пункта ЭВМ должна ожидать перемотки двух лент; как предыдущая выводная лента, так и новая текущая выводная

Таблица 4

Число начальных отрезков, обрабатываемых при стандартном многофазном слиянии

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$
10	36	24	19	17	15	14	13	12
20	90	62	49	44	41	37	34	33
50	294	194	167	143	134	131	120	114
100	714	459	393	339	319	312	292	277
1000	10730	6920	5774	5370	4913	4716	4597	4552
5000	64740	43210	36497	32781	31442	29533	28817	28080

лента должны быть перемотаны в начало, прежде чем сможет выполняться следующая фаза. Это может вызвать существенную задержку, так как в общем случае предыдущая выводная лента содержит значительный процент сортируемых записей (см. столбец "проходы/фазы" в табл. 1). Досадно, когда ЭВМ простаивает во время операций перемотки, тогда как можно было бы, используя иную схему слияния, выполнить полезную работу с остальными лентами.

Эту задачу решает простая модификация многофазной процедуры, хотя она требует не менее пяти лент [см. диссертацию И. Сезари (Univ. of Paris (1968), 25–27), где эта идея приписывается Дж. Кэйрону]. Каждая фаза схемы Кэйрона сливает отрезки с $(T-3)$ лент на некоторую другую ленту, в то время как остающиеся две ленты перематываются.

Рассмотрим, например, случай шести лент и 49 начальных отрезков. В следующей таблице буквой R помечены ленты, перематывающиеся во время данной фазы; предполагается, что $T5$ содержит первоначальные отрезки:

Фаза	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	Время записи	Время перемотки
1	1^{11}	1^{17}	1^{13}	1^{8}	–	(R)	49	17
2	(R)	1^{9}	1^{5}	–	R	3^{8}	$8\times 3=24$	$49-17=32$
3	1^{6}	1^{4}	–	R	3^{5}	R	$5\times 3=15$	$\max(8,24)$
4	1^{2}	–	R	5^{4}	R	3^{4}	$4\times 5=20$	$\max(13,15)$
5	–	R	7^{2}	R	3^{3}	3^{2}	$2\times 7=14$	$\max(17,20)$
6	R	11^{2}	R	5^{2}	3^{1}	–	$2\times 11=22$	$\max(11,14)$
7	15^{1}	R	7^{1}	5^{1}	–	R	$1\times 15=15$	$\max(22,24)$
8	R	11^{1}	7^{0}	–	R	23^{1}	$1\times 23=23$	$\max(15,16)$
9	15^{1}	11^{1}	–	R	33^{0}	R	$0\times 33=0$	$\max(20,23)$
10	(15^{0})	–	R	49^{1}	(R)	(23^{0})	$1\times 49=49$	14

Здесь все перемотки, по существу, совмещены; за исключением фазы 9 ("фиктивная фаза", которая подготавливает окончательное слияние) и перемотки после начальной фазы распределения (когда перематываются все ленты). Если t есть время, необходимое для слияния такого количества записей, какое содержится в одном начальном отрезке, а r—время перемотки на один начальный отрезок, то этот процесс тратит около $182t+40r$ плюс время начального распределения и завершающей перемотки. Соответствующие выражения для стандартного многофазного метода, использующего алгоритм D, есть $140t+104r$, что несколько хуже, если $r=\frac{3}{4}t$, и несколько лучше, если $r=\frac{1}{2}t$.

Все сказанное о стандартном многофазном методе приложимо к многофазному методу Кэйрона; например, последовательность a_n теперь удовлетворяет рекуррентному соотношению

$$a_n = a_{n-2} + a_{n-3} + a_{n-4} \tag{20}$$

вместо (3). Читателю будет полезно проанализировать этот метод таким же образом, как мы анализировали стандартный многофазный, так как это улучшит понимание обоих методов (см., например, упр. 19 и 20).

В табл. 5 сведены факты о многофазном методе Кэйрона, аналогичные фактам об обычном многофазном методе, приведенным в табл. 1. Заметим, что. на самом деле при восьми и более. лентах метод Кэйрона становится *лучше* многофазного как по числу обрабатываемых отрезков, так и по времени перемотки, несмотря на то что он выполняет $(T-3)$-путевое слияние вместо $(T-1)$-путевого!

Это может казаться, парадоксальным, пока мы не поймем, что *высокий порядок слияния не обязательно означает эффективную сортировку.* В качестве крайнего рассмотрим случай, когда на ленту T1 помещается 1 отрезок и n отрезков—на T2, T3,

Таблица 5

Приблизительное поведение многофазного слияния Кэйрона

Ленты	Фазы	Проходы	Проходы/фазы, в процентах	Отношение роста
5	$3.566 \ln S + 0.158$	$1.463 \ln S + 1.016$	41	1.3247180
6	$2.616 \ln S - 0.166$	$0.951 \ln S + 1.014$	36	1.4655712
7	$2.337 \ln S - 0.472$	$0.781 \ln S + 1.001$	33	1.5341577
8	$2.216 \ln S - 0.762$	$0.699 \ln S + 0.980$	32	1.5701473
9	$2.156 \ln S - 1.034$	$0.654 \ln S + 0.954$	30	1.5900054
10	$2.124 \ln S - 1.290$	$0.626 \ln S + 0.922$	29	1.6013473
20	$2.078 \ln S - 3.093$	$0.575 \ln S + 0.524$	28	1.6179086

T4, T5; если мы будем поочередно выполнять слияния на T6 и T1, пока T2, T3, T4, T5 не станут пустыми, то время обработки будет равно $(2n^2 + 3n)$ длинам начальных отрезков, т. е., по существу, пропорционально S^2, а не $S \log S$, хотя все время производится пятипутевое слияние.

Расщепление лент. Эффективное совмещение времени перемотки является проблемой, возникающей во многих приложениях, а не только в сортировке; существует общий подход, который часто может быть использован. Рассмотрим итеративный процесс, который использует ленты следующим образом:

	T1	T2
Фаза 1	Вывод 1	–
	Перемотка	–
Фаза 2	Ввод 1	Вывод 2
	Перемотка	Перемотка
Фаза 3	Вывод 3	Ввод 2
	Перемотка	Перемотка
Фаза 4	Ввод 3	Вывод 4
	Перемотка	Перемотка

и т. д., где "вывод k" означает запись в k-й выводной файл, а "ввод k" означает его чтение. Можно устранить время перемотки, если использовать три ленты, как было предложено К. Вейсертом [*CACM,* **5** (1962), 102]:

	T1	T2	T3
Фаза 1	Вывод 1.1	–	–
	Вывод 1.2	–	–
	Перемотка	Вывод 1.3	–
Фаза 2	Ввод 1.1	Вывод 2.1	–
	Ввод 1.2	Перемотка	Вывод 2.2
	Перемотка	Ввод 1.3	Вывод 2.3
Фаза 3	Вывод 3.1	Ввод 2.1	Перемотка
	Вывод 3.2	Перемотка	Ввод 2.2
	Перемотка	Вывод 3.3	Ввод 2.3
Фаза 4	Ввод 3.1	Вывод 4.1	Перемотка
	Ввод 3.2	Перемотка	Вывод 4.2
	Перемотка	Ввод 3.3	Вывод 4.3

и т. д. Здесь "вывод $k.j$" означает запись j-й трети k-го выводного файла, а "ввод $k.j$" означает ее чтение. В конце концов будет исключено все время перемотки, если перемотка по крайней мере вдвое быстрее чтения/записи. Подобная процедура, в которой вывод в каждой фазе разделяется между лентами, называется "расщеплением лент".

Л. Мак-Аллестер [*CACM*, **7** (1964), 158–159] показал, что расщепление лент приводит к эффективному методу совмещения времени перемотки в многофазном слиянии. Его метод можно использовать с четырьмя или большим количеством лент, и он осуществляет $(T-2)$-путевое слияние.

Предположим снова, что у нас есть шесть лент и попытаемся построить схему слияния, которая работает следующим образом, расщепляя вывод на каждом уровне (буквы I, O и R обозначают соответственно ввод, вывод и перемотку):

Уровень	T1	T2	T3	T4	T5	T6	Число выводимых отрезков
7	I	I	I	I	R	O	u_7
	I	I	I	I	O	R	v_7
6	I	I	I	R	O	I	u_6
	I	I	I	O	R	I	v_6
5	I	I	R	O	I	I	u_5
	I	I	O	R	I	I	v_5
4	I	R	O	I	I	I	u_4
	I	O	R	I	I	I	v_4
3	R	O	I	I	I	I	u_3
	O	R	I	I	I	I	v_3
2	O	I	I	I	I	R	u_2
	R	I	I	I	I	O	v_2
1	I	I	I	I	R	O	u_1
	I	I	I	I	O	R	v_1
0	I	I	I	R	O	I	u_0
	I	I	I	O	R	I	

(21)

Чтобы закончить работу с одним отрезком на T4 и пустыми остальными лентами, мы должны иметь

$$\begin{aligned} v_0 &= 1,\\ u_0+v_1 &= 0,\\ u_1+v_2 &= u_0+v_0,\\ u_2+v_3 &= u_1+v_1+u_0+v_0,\\ u_3+v_4 &= u_2+v_2+u_1+v_1+u_0+v_0,\\ u_4+v_5 &= u_3+v_3+u_2+v_2+u_1+v_1+u_0+v_0,\\ u_5+v_6 &= u_4+v_4u_3+v_3+u_2+v_2+u_1+v_1, \end{aligned}$$

и т. д.; в общем случае требуется, чтобы

$$u_n+v_{n+1} = u_{n-1}+v_{n-1}+u_{n-2}+v_{n-2}+u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4} \tag{22}$$

при всех $n\ge 0$, если считать $u_j=v_j=0$ при всех $j<0$.

У этих уравнений нет единственного решения; в самом деле, если положить все u равными нулю, то получим обычное многофазное, слияние, причем одна лента будет лишней! Но если выбрать $u_n\approx v_{n+1}$, то время перемотки будет удовлетворительно совмещено.

Мак-Аллестер предложил взять $u_n=v_{n-1}+v_{n-2}+v_{n-3}+v_{n-4}$, $v_{n+1}=u_{n-1}+u_{n-2}+u_{n-3}+u_{n-4}$, так что последовательность

$$\langle x_0,x_1,x_2,x_3,x_4,x_5,\ldots\rangle = \langle v_0,u_0,v_1,u_1,v_2,u_2,\ldots\rangle$$

удовлетворяет однородному рекуррентному соотношению

$$x_n = x_{n-3}+x_{n-5}+x_{n-7}+x_{n-9}.$$

Оказалось, однако, что лучше положить

$$\begin{aligned} v_{n+1} &= u_{n-1}+v_{n-1}+u_{n-2}+v_{n-2},\\ u_n &= u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4}. \end{aligned}$$

Эта последовательность не только немного лучше по времени слияния, ее большое преимущество в том, что соответствующее время слияния можно проанализировать математически. [Вариант Мак-Аллестера для анализа крайне труден, потому что в одной фазе могут встречаться отрезки разной длины; мы увидим, что такого не может случиться при (23).]

Можно вывести число отрезков на каждой ленте на каждом уровне, двигаясь назад по схеме (21); мы получаем следующую схему сортировки:

Уровень	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	Время записи	Время перемотки
	1^{23}	1^{21}	1^{17}	1^{10}	–	1^{11}	82	23
7	1^{19}	1^{17}	1^{13}	1^{6}	R	$1^{11}4^{4}$	$4\times4=16$	$82-23$
	1^{13}	1^{11}	1^{7}	–	4^{6}	R	$6\times4=24$	25
6	1^{10}	1^{8}	1^{4}	R	4^{9}	$1^{8}4^{4}$	$3\times4=12$	10
	1^{6}	1^{4}	–	4^{4}	R	$1^{4}4^{4}$	$4\times4=16$	36
5	1^{5}	1^{3}	R	$4^{4}7^{1}$	4^{8}	$1^{3}4^{4}$	$1\times7=7$	17
	1^{2}	–	7^{3}	R	4^{5}	4^{4}	$3\times7=21$	23
4	1^{1}	R	$7^{3}13^{1}$	$4^{3}7^{1}$	4^{4}	4^{3}	$1\times13=13$	21
	–	13^{1}	R	$4^{2}7^{1}$	4^{3}	4^{2}	$1\times13=13$	34
3	R	$13^{1}19^{1}$	$7^{2}13^{1}$	$4^{1}7^{1}$	4^{2}	4^{1}	$1\times19=19$	23
	19^{1}	R	$7^{1}13^{1}$	7^{1}	4^{1}	–	$1\times19=19$	32
2	$19^{1}31^{0}$	$13^{1}19^{1}$	$7^{1}13^{1}$	7^{1}	4^{1}	R	$0\times31=0$	25
	R	19^{1}	13^{1}	7^{0}	–	31^{1}	$1\times31=31$	19
1	$19^{1}31^{0}$	19^{1}	13^{1}	7^{0}	R	$31^{1}52^{0}$	$0\times52=0$	$\max(36,31,23)$
	$19^{1}31^{0}$	19^{1}	13^{1}	–	52^{0}	R	$0\times52=0$	
0	$19^{1}31^{0}$	19^{1}	13^{1}	R	$52^{0}82^{0}$	$31^{1}52^{0}$	$0\times82=0$	
	(31^{0})	(19^{0})	–	82^{1}	(R)	(52^{0})	$1\times82=82$	0

Несовмещенная перемотка встречается только при перемотке вводной ленты T5 (82 единицы), в течение первой половины фазы второго уровня (25 единиц) и в течение окончательных фаз "фиктивного слияния" на уровнях 1 и 0 (36 единиц). Таким образом, время работы можно оценить величиной $273t+143r$; для алгоритма D соответствующая величина $268t+208r$ почти всегда хуже.

Нетрудно видеть (см. упр. 23), что длины отрезков, выводимых во время каждой фазы, суть последовательно

$$4,4,7,13,19,31,52,82,133,\ldots \tag{24}$$

при этом последовательность $\langle t_1,t_2,t_3,\ \ldots\rangle$ удовлетворяет закону

$$t_n=t_{n-2}+2t_{n-3}+t_{n-4}, \tag{25}$$

если считать $t_n=1$ при $n\le 0$. Можно также проанализировать оптимальное размещение фиктивных отрезков, рассмотрев строки чисел слияний, как мы делали для стандартного многофазного метода [ср. с (8)]:

Уровень	$T1$	$T2$	$T3$	$T4$	$T6$	Окончательный вывод на ленте
1	1	1	1	1	–	$T5$
2	1	1	1	–	1	$T4$
3	21	21	2	2	1	$T3$
4	2221	222	222	22	2	$T2$
5	23222	23222	2322	23	222	$T1$
6	333323222	33332322	333323	3333	2322	$T6$
.....						
n	A_n	B_n	C_n	D_n	E_n	$T(k)$
$n+1$	$(A_n''E_n+1)B_n$	$(A_n''E_n+1)C_n$	$(A_n''E_n+1)D_n$	$A_n''E_n+1$	A_n'	$T(k-1)$
.....						

(26)

где $A_n=A'nA_n''$ и A_n'' состоит из последних u_n чисел слияний A_n. Приведенное выше правило перехода с уровня n на уровень $n+1$ справедливо для *любой* схемы, удовлетворяющей (22). Если мы определяем u и v посредством (23), то строки A_n, ..., E_n можно выразить в следующем, довольно простом виде [ср. с (9)]:

$$\begin{aligned}
A_n &= (W_{n-1}W_{n-2}W_{n-3}W_{n-4})+1,\\
B_n &= (W_{n-1}W_{n-2}W_{n-3})+1,\\
C_n &= (W_{n-1}W_{n-2})+1,\\
D_n &= (W_{n-1})+1,\\
E_n &= (W_{n-2}W_{n-3})+1,
\end{aligned} \tag{27}$$

где

$$W_n = (W_{n-3}W_{n-4}W_{n-2}W_{n-3}) + 1 \qquad \text{при } n > 0;$$
$$W_0 = \text{'0'} \text{ и } W_n = (\text{пусто}) \qquad \text{при } n < 0. \tag{28}$$

Исходя из этих соотношений, легко подробно проанализировать случай шести лент.

В общем случае, если имеется $T \geq 5$ лент, то положим $P = T-2$ и определим последовательности $\langle u_n \rangle$, $\langle v_n \rangle$ по правилам

$$v_{n+1} = u_{n-1} + v_{n-1} + \cdots + u_{n-r} + v_{n-r};$$
$$u_n = u_{n-r-1} + v_{n-r-1} + \cdots + u_{n-P} + v_{n-P} \qquad \text{при } n \geq 0, \tag{29}$$

где $r = \lfloor P/2 \rfloor$, $v_0 = 1$ и $u_n = v_n = 0$ при $n < 0$. Если $w_n = u_n + v_n$, то имеем

$$w_n = w_{n-2} + \cdots + w_{n-r} + 2w_{n-r-1} + w_{n-r-2} + \cdots + w_{n-P}, \qquad n > 0,$$
$$w_0 = 1 \text{ и } w_n = 0 \qquad \text{при } n < 0. \tag{30}$$

При начальном распределении для уровня $n+1$ на ленту k помещается $w_n + w_{n-1} + \cdots + w_{n-P+k}$ отрезков при $1 \leq k \leq P$ и $w_{n-1} + \cdots + w_{n-r}$— на ленту T; лента $T-1$ используется для ввода. Затем u_n отрезков сливаются на ленту T, в то время как лента $T-1$ перематывается; v_n отрезков сливаются на $T-1$, пока T перематывается; u_{n-1} отрезков—на $T-1$, пока $T-2$ перематывается, и т. д.

Таблица 6

Приблизительное поведение многофазного слияния с расщеплением лент

Ленты	фазы	Проходы	Проходы/фазы в процентах	Отношение роста
4	$2.885 \ln S + 0.000$	$1.443 \ln S + 1.000$	50	1.4142136
5	$2.078 \ln S + 0.232$	$0.929 \ln S + 1.022$	45	1.6180340
6	$2.078 \ln S - 0.170$	$0.752 \ln S + 1.024$	34	1.6180340
7	$1.958 \ln S - 0.408$	$0.670 \ln S + 1.007$	34	1.6663019
8	$2.008 \ln S - 0.762$	$0.624 \ln S + 0.994$	31	1.6454116
9	$1.972 \ln S - 0.987$	$0.595 \ln S + 0.967$	30	1.6604077
10	$2.013 \ln S - 1.300$	$0.580 \ln S + 0.94l$	29	1.6433803
20	$2.069 \ln S - 3.164$	$0.566 \ln S + 0.536$	27	1.6214947

Таблица 6 показывает приблизительное поведение этой процедуры, когда S не слишком мало. Столбец "проходы/фазы" примерно указывает, какая часть всего файла перематывается во время каждой половины фазы и какая часть файла записывается за время каждой полной фазы. *Метод расщепления лент превосходит стандартный многофазный на шести или более лентах* и, вероятно; также на пяти лентах, по крайней мере для больших S.

Если $T = 4$, то указанная процедура стала бы, по существу, эквивалентной сбалансированному двухпутевому слиянию *без* совмещения времени перемотки, так как w_{2n+1} было бы равно 0 при всех n. Поэтому элементы табл. 6 при $T = 4$ были получены посредством небольшой модификации, состоящей в том, что полагалось

$$v_2 = 0, u_1 = 1, v_1 = 0, u_0 = 0, v_0 = 1 \text{ и } v_{n+1} = u_{n-1} + v_{n-1},$$
$$u_n = u_{n-2} + v_{n-2} \qquad \text{при } n \geq 2.$$

Это приводит к очень интересной схеме сортировки (см. упр. 25 и 26).

Упражнения

1. [16] На рис. 69 указан порядок, в котором алгоритм D распределяет по пяти лентам отрезки с 34-го по 65-й; в каком порядке распределяются отрезки с 1-го по 33-й?
>2. [21] Верно ли, что после двух фаз слияния в алгоритме D, т. е. когда мы во второй раз достигнем шага D6, все фиктивные отрезки исчезают?
>3. [22] Докажите, что по окончании шага D4 всегда выполнено условие $\mathtt{D}[1] \geq \mathtt{D}[2] \geq \ldots \geq \mathtt{D}[T]$. Объясните, важность этого условия для правильной работы механизма шагов D2 и D3.
4. [M20] Выведите производящие функции (7).
5. [BM26] (Э. П. Майлс мл., 1960.) Докажите, что при всех $p \geq 2$ многочлен $f_p(z) = z^p - z^{p-1} - \cdots - z - 1$ имеет p различных корней, из которых ровно один превосходит 1 по абсолютной величине. [*Указание:* рассмотрите многочлен $z^{p+1} - 2z^p + 1$.]
6. [BM24] Цель этого упражнения—рассмотреть способ составления табл. 1, 5 и 6. Предположим, что имеется схема слияния, свойства которой следующим образом характеризуются многочленами $p(z)$

и $q(z)$: (1) Число начальных отрезков в "точном распределении", требующем n фаз слияния, равно коэффициенту при z^n в $p(z)/q(z)$. (2) Число начальных отрезков, обрабатываемых в течение этих n фаз слияния, равно коэффициенту при z^n в $p(z)/q(z)^2$. (3) У многочлена $q(z^{-1})$ есть "главный корень" α, такой, что $q(\alpha^{-1}) = 0$, $q'(\alpha^{-1}) \neq 0$, $p(\alpha^{-1}) \neq 0$, и из $q(\beta^{-1}) = 0$ следует, что $\beta = \alpha$ или $|\beta| < |\alpha|$.

Докажите, что существует $\varepsilon > 0$, такое, что если S равно числу отрезков в точном распределении, требующем n фаз слияния, а во время этих фаз обрабатывается ρS отрезков, то $n = a\ln S + b + O(S^{\varepsilon})$, $\rho = c\ln S + d + O(S^{-\varepsilon})$, где

$$a = (\ln\alpha)^{-1}, \quad b = -a\ln\left(\frac{p(\alpha^{-1})}{-q'(\alpha^{-1})}\right) - 1, \quad c = a\frac{\alpha}{-q'(\alpha^{-1})},$$
$$d = \frac{(b+1)\alpha - p'(\alpha^{-1})/p(\alpha^{-1}) + q''(\alpha^{-1})/q'\alpha^{-1}}{-q'(\alpha^{-1})}.$$

7. [BM22] Пусть α_p—главный корень многочлена $f_p(z)$ из упр. 5. Каково асимптотическое поведение α_p при $p \to \infty$?
8. [M20] (Э. Нетто, 1901.) Пусть $N_m^{(p)}$ есть число способов выразить m в виде упорядоченной суммы целых чисел $\{1, 2, \ldots, p\}$. Например, если $p = 3$ и $m = 5$, то имеется 13 способов: $1+1+1+1+1 = 1+1+1+2 = 1+1+2+1 = 1+1+3 = 1+2+1+1 = 1+2+2 = 1+3+1 = 2+1+1+1 = 2+1+2 = 2+2+1 = 2+3 = 3+1+1 = 3+2$. Покажите, что $N_m^{(p)}$ являются обобщенными числами Фибоначчи.
9. [M20] Пусть $K_m^{(p)}$—число последовательностей из нулей и единиц, таких, что в них нет p последовательных единиц. Например, если $p = 3$ и $m = 5$, имеется 24 варианта: 00000, 00001, 00010, 00011, 00100, 00101, 00110, 01000, 01001, ..., 11011. Покажите, что $K_m^{(p)}$ являются обобщенными числами Фибоначчи.
10. [M27] *(Система счисления с обобщенными числами Фибоначчи.)* Докажите, что каждое неотрицательное целое n имеет единственное представление в виде суммы различных чисел Фибоначчи p-го порядка $F_j^{(p)}$ при $j \geq p$, удовлетворяющее условию, что не используются никакие p последовательные числа Фибоначчи.
11. [M24] Докажите, что n-й элемент цепочки Q_∞ в (12) равен количеству различных чисел Фибоначчи в представлении элемента $n-1$ числами Фибоначчи пятого порядка (см. упр. 10).
>12. [M20] Найдите зависимость между степенями матрицы

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

и точным фибоначчиевым распределением в (1).

>13. [22] Докажите следующее интересное свойство точных фибоначчиевых распределений: если окончательный вывод оказывается на ленте номер T, то число отрезков на всех других лентах *нечетное,* если окончательный вывод оказывается на некоторой ленте, отличной от T, то число отрезков будет *нечетным* на этой ленте и *четным* на остальных [см. (1)].
14. [M35] Пусть $T_n(x) = \sum_{k\geq 0} T_{nk}x^k$, где $T_n(x)$—многочлены, определенные в (16). (a) Покажите, что для каждого k существует число $n(k)$, такое, что $T_{1k} \leq T_{2k} \leq \ldots \leq T_{n(k)k} > T_{n(k)+1,k} \geq \ldots$. (b) При условии что $T_{n'k'} < T_{nk'}$ и $n' < n$, докажите, что $T_{n'k} \leq T_{nk}$ для всех $k \geq k'$. (c) Докажите, что существует неубывающая последовательность $\langle M_n\rangle$, такая, что $\sum_n(S) = \min_{j\geq 1}\sum_j(S)$ при $M_n \leq S < M_{n+1}$, но $\sum_n(S) > \min_{j\geq 1}\sum_j(S)$ при $S \geq M_{n+1}$. [См. (19).]
15. [M43] Верно ли, что $\sum_{n-1}(m) < \sum_n(m)$ влечет $\sum_n(m) \leq \sum_{n+1}(m) \leq \sum_{n+2}(m) \leq \ldots$? (Такой результат сильно упростил бы вычисление табл. 2.)
16. [M43] Определите асимптотическое поведение многофазного слияния с оптимальным распределением фиктивных отрезков.
17. [32] Верно ли, что отрезки для оптимального многофазного распределения можно разместить таким образом, что распределение $S+1$ начальных отрезков получается путем добавления одного отрезка (на соответствующую ленту) к распределению S начальных отрезков?
18. [30] Верно ли, что оптимальное многофазное распределение дает наилучшую возможную схему слияния в том смысле, что суммарное количество обрабатываемых начальных отрезков минимально, если требуется, чтобы начальные отрезки размещались не более, чем на $T-1$ лентах? (Временем перемотки пренебречь.)

19. [21] Составьте таблицу, аналогичную (1), для многофазного метода сортировки Кэйрона для шести лент.
20. [M24] Какая производящая функция для кэйроновской многофазной сортировки на шести лентах соответствует (7) и (16)? Какие соотношения, аналогичные (9) и (27), определяют строки чисел слияний?
21. [11] Что должно появиться на уровне 7 в (26)?
22. [M21] Каждый член последовательности (24) приблизительно равен сумме двух предыдущих. Наблюдается ли это явление для остальных членов последовательности? Сформулируйте и докажите теорему о $t_n - t_{n-1} - t_{n-2}$.

>23. [29] Какие изменения надо было бы сделать в (25), (27) и (28), если бы (23) заменилось на $v_{n+1} = u_{n-1} + v_{n-1} + u_{n-2}$, $u_n = v_{n-2} + u_{n-3} + v_{n-3} + u_{n-4} + v_{n-4}$?

23. [BM41] Вычислите асимптотическое поведение многофазной процедуры с расщеплением лент, если элемент v_{n+1} определен как сумма первыхΞq членов $u_{n-1}+v_{n-1}+\cdots+u_{n-P}+v_{n-P}$ при различных $P = T-2$ и $0 \le q \le 2P$. (В тексте рассматривается только случай $q = 2\lfloor P/2 \rfloor$; ср. с упр. 23.)
25. [19] Продемонстрируйте, как многофазное слияние с расщеплением лент, упомянутое в конце этого пункта, сортировало бы 32 начальных отрезка. (Дайте анализ фаза за фазой, как это сделано в тексте в примере с 82 отрезками и 6 лентами.)
26. [M21] Проанализируйте поведение многофазного слияния с расщеплением лент на четырех лентах при $S = 2^n$ и при $S = 2^n + 2^{n-1}$ (см. упр. 25).
27. [23] Если начальные отрезки распределены на лентах в соответствии с точным распределением, то многофазная стратегия превращается просто в "сливать до опустошения". Мы сливаем отрезки со всех непустых входных лент, пока одна из них не станет пустой, затем мы используем эту ленту как следующую выводную, а предыдущую выводную ленту используем как вводную.

Верно ли, что эта стратегия "сливать до опустошения" всегда выполняет сортировку независимо от того, как распределены начальные отрезки, при условии что мы распределяем их по крайней мере на две ленты (одна лента, конечно, будет оставлена пустой, чтобы она могла служить первой выводной лентой).

28. [M26] Предыдущее упражнение определяет весьма большое семейства схем слияния. Покажите, что многофазная схема *наилучшая* из них в следующем смысле: если имеется шесть лент и мы рассматриваем класс всех начальных распределений (a, b, c, d, e), таких, что стратегия "сливать до опустошения" требует n или меньше фаз для сортировки, то $a+b+c+d+e \le t_n$, где t_n—соответствующее число для многофазной сортировки (1).
29. [M47] Упр. 28 показывает, что многофазное распределение оптимально среди всех схем "сливать до опустошения" в смысле минимальности числа фаз. Но является ли оно оптимальным также в смысле минимальности числа проходов?

Пусть числа a и b взаимно простые, и предположим, что $a + b$ есть число Фибоначчи F_n. Верно ли следующее предположение, высказанное Р. М. Карпом: число начальных отрезков, обрабатываемых схемой "сливать до опустошения", начинающейся с распределения (a, b), больше или равно $((n-5)F_{n+1} + (2n+2)F_n)/5$? (Указанное значение достигается, когда $a = F_{n+1}$, $b = F_{n-2}$.)

30. [42] Составьте таблицу, аналогичную табл. 2, для многофазного слияния с расщеплением лент.

Каскадное слияние Другая основная схема, называемая "каскадным слиянием", на самом деле была открыта раньше многофазной [Б. К. Бетц и У. К. Картер, ACM Nat'l Conference, **14** (1959), Paper 14]. Ниже в таблице этот подход иллюстрируется для 6 лент и 190 начальных отрезков с использованием обозначений из п. 5.4.2:

	$T1$	$T2$	$T3$	$T4$	$T5$	$T6$	Количество обработанных начальных отрезков
Проход 1.	1^{55}	1^{50}	1^{41}	1^{29}	1^{15}	—	190
Проход 2.	—	$1^{5}{}_{*}$	2^{9}	3^{12}	4^{14}	5^{15}	190
Проход 3.	15^{5}	14^{4}	12^{3}	9^{2}	$5^{1}{}_{*}$	—	190
Проход 4.	—	$15^{1}{}_{*}$	29^{1}	41^{1}	50^{1}	55^{1}	190
Проход 5.	190^{1}	—	—	—	—	—	190

Каскадное слияние, подобно многофазному, начинается с "точного распределения" отрезков по лентам, хотя правила точного распределения отличны от правил п. 5.4.2. Каждая строка таблицы представляет полный проход по *всем* данным. Проход 2, например, получается посредством выполнения пятипутевого слияния с T1, T2, T3, T4, T5 на T6, пока T5 не станет пустой (при этом на T6 помещаются 15 отрезков относительной длины 5), затем четырехпутевого слияния с T1, T2, T3, T4 на T5, затем трехпутевого слияния на T4, двухпутевого слияния на T3 и, наконец, однопутевого слияния (операции копирования) с T1 на T2. Проход 3 получается таким же образом путем выполнения сначала пятипутевого слияния, пока одна лента не станет пустой, затем четырехпутевого и т. д. (Похоже, что этому пункту книги следовало бы присвоить номер 5.4.3.2.1, а не 5.4.3!)

Ясно, что операции копирования излишни, и их можно было бы опустить. Фактически, однако, в случае шести лент это копирование занимает только небольшой процент всего времени. Элементы, которые получаются простым копированием, отмечены в приведенной таблице звездочкой. Только 25 из 950 обрабатываемых отрезков принадлежат этому классу. Большая часть времени отводится пятипутевому и четырехпутевому слияниям.

На первый взгляд может показаться, что каскадная схема—довольно плохой вариант в сравнении с многофазной, так как стандартная многофазная схема использует все время $(T-1)$-путевое слияние, в то время как каскадная использует $(T-1)$-путевое, $(T-2)$-путевое, $(T-3)$-путевое и т. д., но в действительности она асимптотически *лучше,* чем многофазная, для шести и более лент! Как мы видели в п. 5.4.2, высокий порядок слияния не является гарантией эффективности. В табл. 1 показаны характеристики выполнения каскадного слияния по аналогии с подобной таблицей п. 5.4.2.

Нетрудно вывести "точные распределения" для каскадного слияния. Для шести лент имеем

Уровень	$T1$	$T2$	$T3$	$T4$	$T5$
0	1	0	0	0	0
1	1	1	1	1	1
2	5	4	3	2	1
3	15	14	12	9	6
4	55	50	41	29	15
5	190	175	146	105	55
.....					
n	a_n	b_n	c_n	d_n	e_n
$n+1$	$a_n+b_n+c_n+d_n+e_n$	$a_n+b_n+c_n+d_n$	$a_n+b_n+c_n$	a_n+b_n	a_n

(1)

Таблица 1

Характер поведения каскадного слияния

Ленты	Проходы (с копированием)	Проходы (без копирования)	Отношение роста
3	$2.078\ln S+0.672$	$1.504\ln S+0.992$	1.6180340
4	$1.235\ln S+0.754$	$1.102\ln S+0.820$	2.2469796
5	$0.946\ln S+0.796$	$0.897\ln S+0.800$	2.8793852
6	$0.796\ln S+0.821$	$0.773\ln S+0.808$	3.5133371
7	$0.703\ln S+0.839$	$0.691\ln S+0.822$	4.1481149
8	$0.639\ln S+0.852$	$0.632\ln S+0.834$	4.7833861
9	$0.592\ln S+0.861$	$0.587\ln S+0.845$	5.4189757
10	$0.555\ln S+0.869$	$0.552\ln S+0.854$	6.0547828
20	$0.397\ln S+0.905$	$0.397\ln S+0.901$	12.4174426

Отметим интересное свойство этих чисел—их относительные величины являются также и длинами диагоналей правильного $(2T-1)$-угольника. Например, пять диагоналей одиннадцатиугольника на рис. 73 имеют относительные длины, очень близкие к 190, 175, 146, 105 и 55! Мы докажем этот замечательный факт

Picture: Рис. 73. Геометрическая интерпретация каскадных чисел.

позднее в этом пункте, а также увидим, что относительные времена, затрачиваемые на $(T-1)$-путевое слияние, $(T-2)$-путевое слияние, ..., однопутевое слияние, приблизительно пропорциональны *квадратам* длин этих диагоналей.

***Начальное распределение отрезков.** Если число начальных отрезков в действительности не есть число Фибоначчи, мы можем, как обычно, вставить фиктивные отрезки. Поверхностный анализ ситуации показывает, что метод приписывания фиктивных отрезков несуществен, так как каскадное слияние всегда осуществляет полные проходы; если имеется 190 начальных отрезков, то каждая запись обрабатывается пять раз, как в приведенном выше примере, но если имеется 191 отрезок, то, очевидно, следует увеличить уровень, и теперь каждая запись будет обрабатываться шесть раз. К счастью, в действительности можно избежать такого резкого скачка. Дэвид Э. Фергюсон нашел способ так распределить начальные отрезки, что многие операции во время первой

Picture: Рис. 74. Эффективность каскадного слияния с распределением по алгоритму D.

фазы слияния сводятся к копированию содержимого ленты. Если обойти такие копирования (просто изменив "логические" номера ленточных устройств по отношению к "физическим" номерам, как в алгоритме 5.4.2D), то получим относительно плавный переход с уровня на уровень, как изображено на рис. 74.

Предположим, что (a, b, c, d, e), где $a \geq b \geq c \geq d \geq e$—точное распределение. Переопределив соответствие между логическими и физическими ленточными устройствами, мы можем представить, что реальное распределение—это (e, d, c, b, a), т. е. a отрезков на T5, b на T4 и т. д. Следующее точное распределение—это $(a+b+c+d+e, a+b+c+d, a+b+c, a+b, a)$; и если ввод исчерпывается прежде, чем мы достигаем этого следующего уровня, то будем считать, что ленты содержат соответственно $(D_1, D_2, D_3, D_4, D_5)$ фиктивных отрезков, где

$$\begin{gathered} D_1 \leq a+b+c+d, \quad D_1 \leq a+b+c, \quad D_3 \leq a+b, \\ D_4 \leq a, \quad D_5 = 0; \qquad D_1 \geq D_2 \geq D_3 \geq D_4 \geq D_5. \end{gathered} \tag{2}$$

Мы вольны представлять себе, что эти фиктивные отрезки появляются на лентах в любом удобном месте. Предполагается, что первый проход слияния даст a отрезков посредством пятипутевого слияния, затем b отрезков посредством четырехпутевого и т. д. Наша цель состоит в расположении фиктивных отрезков таким образом, чтобы заменить слияние копированием. Удобно выполнять первый проход слияния следующим образом:

1. Если $D_4 = a$, то вычесть a из всех D_1, D_2, D_3, D_4 и заявить, что T5—результат слияния. Если $D_4 < a$, то слить a отрезков с лент T1 по T5, используя минимально возможное число фиктивных отрезков на лентах T1–T5 так, чтобы новые значения D_1, D_2, D_3, D_4 удовлетворяли соотношениям

$$\begin{gathered} D_1 \leq b+c+d, \quad D_2 \leq b+c, \quad D_3 \leq b, \\ D_4 = 0; \qquad D_1 \geq D_2 \geq D_3 \geq D_4. \end{gathered} \tag{3}$$

Таким образом, если D_2 было первоначально $\leq b+c$, то мы не используем ни одного фиктивного отрезка с этой ленты на данном шаге; в то же время, если $b+c < D_2 \leq a+b+c$, мы используем ровно $D_2 - b - c$ фиктивных отрезков.

2. (Этот шаг аналогичен шагу 1, но с некоторым "сдвигом".) Если $D_3 = b$, то вычесть b из всех D_1, D_2, D_3 и заявить, что T4—результат слияния. Если $D_3 < b$, то слить b отрезков с лент T1–T4, уменьшая, если необходимо, число фиктивных отрезков, чтобы достичь

$$D_1 \leq b+c, \quad D_2 \leq b, \quad D_3 = 0; \qquad D_1 \geq D_2 \geq D_3.$$

3. И так далее.

Метод Фергюсона распределения отрезков по лентам можно проиллюстрировать, рассмотрев процесс перехода с уровня 3 на уровень 4 в (1). Допустим, что "логические" ленты (T1, ..., T5) содержали соответственно $(5, 9, 12, 14, 15)$ отрезков и что мы хотим довести это в конечном итоге до $(55, 50, 41, 29, 15)$. Эта процедура может быть кратко записана так:

Шаг	Добавить к T1	Добавить к T2	Добавить к T3	Добавить к T4	Добавить к T5	Сэкономленная величина
$(1,1)$	9	0	0	0	0	$15+14+12+5$
$(2,2)$	3	12	0	0	0	$15+14+9+5$
$(2,1)$	9	0	0	0	0	$15+14+5$
$(3,3)$	2	2	14	0	0	$15+12+5$
$(3,2)$	3	12	0	0	0	$15+9+5$
$(3,1)$	9	0	0	0	0	$15+5$
$(4,4)$	1	1	1	15	0	$14+5$
$(4,3)$	2	2	14	0	0	$12+5$
$(4,2)$	3	12	0	0	0	$9+5$
$(4,1)$	9	0	0	0	0	5

Сначала помещаем девять отрезков на T1, затем $(3, 12)$—на T1 и T2 и т. д. Если ввод исчерпается, скажем, во время шага $(3, 2)$, то "сэкономленная величина" составляет $15+9+5$. Это означает, что мы избегаем пятипутевого слияния 15 отрезков, двухпутевого слияния 9 отрезков и однопутевого слияния 5 отрезков посредством присвоения фиктивных отрезков. Другими словами, $15+9+5$ отрезков, присутствующих на уровне 3, не обрабатываются в течение первой фазы слияния.

Следующий алгоритм детально описывает этот процесс.

Алгоритм С. *(Сортировка каскадным слиянием со специальным распределением.)* Этот алгоритм распределяет начальные отрезки по лентам отрезок за отрезком, пока запас начальных отрезков не исчерпается. Затем он определяет, как следует сливать ленты, предполагая, что имеется $T \geq 3$ лентопротяжных устройств, при этом используется самое большее $(T-1)$-путевое слияние и ненужные однопутевые слияния устраняются. Лента T может быть использована для хранения ввода, так как на нее не попадает ни один начальный отрезок. Алгоритм работает со следующими массивами:

$\texttt{A}[j]$, $1 \leq j \leq T$: Последнее точное каскадное распределение, которое было достигнуто.
$\texttt{AA}[j]$, $1 \leq j \leq T$: Точное каскадное распределение, к которому мы стремимся.
$\texttt{D}[j]$, $1 \leq j \leq T$: Число фиктивных отрезков, предполагаемых присутствующими на логическом лентопротяжном устройстве с номером j.
$\texttt{M}[j]$, $1 \leq j \leq T$: Максимальное число фиктивных отрезков, которое желательно иметь на логическом лентопротяжном устройстве с номером j.
$\texttt{TAPE}[j]$, $1 \leq j \leq T$: Номер физического лентопротяжного устройства, соответствующего логическому лентопротяжному устройству с номером j.

C1 [Начальная установка.] Установить $\texttt{A}[k] \leftarrow \texttt{AA}[k] \leftarrow \texttt{D}[k] \leftarrow 0$ при $2 \leq k \leq T$; установить $\texttt{A}[1] \leftarrow 0$, $\texttt{AA}[1] \leftarrow 1$, $\texttt{D}[1] \leftarrow 1$; установить $\texttt{TAPE}[k] \leftarrow k$ при $1 \leq k \leq T$. Наконец, установить $i \leftarrow T-2$, $j \leftarrow 1$, $k \leftarrow 1$, $l \leftarrow 0$, $m \leftarrow 1$ и перейти к шагу **C5** (эти маневры являются одним из способов начать работу непосредственно во внутреннем цикле с соответствующей установкой управляющих переменных).

C2 [Начать новый уровень.] (Мы только что получили точное распределение. Но так как еще имеются исходные данные, то необходимо подготовиться к следующему уровню.) Увеличить l на 1. Установить $\texttt{A}[k] \leftarrow \texttt{AA}[k]$ при $1 \leq k \leq T$, затем установить $\texttt{AA}[T-k] \leftarrow \texttt{AA}[T-k+1] + \texttt{A}[k]$ при $k = 1, 2, \ldots, T-1$ (в таком порядке). Установить $(\texttt{TAPE}[1], \texttt{TAPE}[2], \ldots, \texttt{TAPE}[T-1]) \leftarrow (\texttt{TAPE}[T-1], \ldots, \texttt{TAPE}[2], \texttt{TAPE}[1])$ и установить $\texttt{D}[k] \leftarrow \texttt{AA}[k+1]$ при $1 \leq k < T$. Наконец, установить $i \leftarrow 1$.

C3 [Начать подуровень i.] Установить $j \leftarrow i$. (Переменные i и j представляют "шаг (i, j)" в таблице, иллюстрирующей метод Фергюсона.)

C4 [Начать шаг (i, j).] Установить $k \leftarrow j$ и $m \leftarrow \texttt{A}[T-j-1]$. Если $m = 0$ и $i = j$, то установить $i \leftarrow T-2$ и вернуться к **C3**; если $m = 0$ и $i \neq j$, вернуться к **C2**. (Переменная m представляет собой число отрезков, которое должно быть записано на ленту $\texttt{TAPE}[k]$; m бывает равно 0, только если $l = 1$.)

C5 [Ввести на ленту $\texttt{TAPE}[k]$.] Записать один отрезок на ленту номер $\texttt{TAPE}[k]$ и уменьшить $\texttt{D}[k]$ на 1. Затем, если ввод исчерпан, перемотать все ленты и перейти к шагу **C7**.

C6 [Продвижение.] Уменьшить m на 1. Если $m > 0$, вернуться к шагу **C5**. В противном случае уменьшить k на 1; если $k > 0$, установить $m \leftarrow \texttt{A}[T-j-1] - \texttt{A}[T-j]$ и вернуться к **C5**, если $m > 0$. В противном случае уменьшить j на 1; если $j > 0$, перейти к шагу **C4**. В противном случае увеличить i на 1; если $i < T < 1$, вернуться к шагу **C3**. В противном случае перейти к **C2**.

C7 [Подготовка к слиянию.] (К этому моменту начальное распределение завершено, и таблицы **A**, **AA**, **D** и **TAPE** описывают состояние всех лент в данный момент.) Установить $\texttt{M}[k] \leftarrow \texttt{AA}[k+1]$ при $1 \leq k < T$ и установить $\texttt{FIRST} \leftarrow 1$. (Переменная **FIRST** принимает ненулевое значение только во время первого прохода слияния.)

C8 [Каскад.] Если $l = 0$, остановиться; сортировка завершена, вывод находится на $\texttt{TAPE}[1]$. В противном случае при $p = T-1, T-2, \ldots, 1$ (в таком порядке) выполнять p-путевое слияние с лент $\texttt{TAPE}[1], \ldots, \texttt{TAPE}[p]$ на $\texttt{TAPE}[p+1]$ следующим образом:

Если $p = 1$, то моделировать однопутевое слияние обычной перемоткой $\texttt{TAPE}[2]$ и заменой $\texttt{TAPE}[1] \leftrightarrow \texttt{TAPE}[2]$. В противном случае, если $\texttt{FIRST} = 1$ и $\texttt{D}[p-1] = \texttt{M}[p-1]$, то моделировать p-путевое слияние, просто поменяв $\texttt{TAPE}[p] \leftrightarrow \texttt{TAPE}[p+1]$, перемотав $\texttt{TAPE}[p]$ и вычтя $M[p-1]$ из каждого $\texttt{D}[1], \ldots, \texttt{D}[p-1]$, $\texttt{M}[1], \ldots, \texttt{M}[p-1]$. В противном случае вычесть $\texttt{M}[p-1]$ из всех $\texttt{M}[1], \ldots, \texttt{M}[p-1]$. Затем слить по одному отрезку с каждой $\texttt{TAPE}[j]$, такой, что $1 \leq j \leq p$ и $\texttt{D}[j] \leq \texttt{M}[j]$; вычесть единицу из каждого $\texttt{D}[j]$, такого, что $1 \leq j \leq p$ и $\texttt{D}[j] > \texttt{M}[j]$, и поместить выводной отрезок на $\texttt{TAPE}[p+1]$. Продолжать работу, пока $\texttt{TAPE}[p]$ не станет пустой. Затем перемотать $\texttt{TAPE}[p]$ и $\texttt{TAPE}[p+1]$.

C9 [Опуститься на один уровень.] Уменьшить l на 1, установить $\texttt{FIRST} \leftarrow 0$, установить $(\texttt{TAPE}[1], \ldots, \texttt{TAPE}[T]) \leftarrow (\texttt{TAPE}[T], \ldots, \texttt{TAPE}[1])$. (К этому моменту все **D** и **M**—нули и таковыми останутся.) Вернуться к **C8**. ∎

Шаги C1–C6 этого алгоритма выполняют распределение, шаги C7–C9 выполняют слияние; эти две части совершенно независимы одна от другой, и можно было бы хранить $\texttt{M}[k]$ и $\texttt{AA}[k+1]$ в одних и тех же ячейках памяти.

Picture: Рис. 75. Каскадное слияние со специальным распределением.

***Анализ каскадного слияния.** Каскадное слияние поддается анализу с большим трудом, чем многофазное. Но этот анализ особенно интересен, поскольку содержит много замечательных формул.

Настоятельно рекомендуем читателям, интересующимся дискретной математикой, самостоятельно проанализировать каскадное распределение, прежде чем читать дальше, ведь числа имеют так много необычных свойств, открывать которые—одно удовольствие! Мы обсудим здесь лишь один из многих подходов, обращая особое внимание на методы получения результатов.

Для удобства рассмотрим случай шести лент. При этом будем стараться получить формулы, которые обобщаются на случай любого T. Соотношения (1) приводят к первой основной системе:

$$\begin{aligned}
a_n &= a_n && = \binom{0}{0}a_n,\\
b_n &= a_n - e_{n-1} = a_n - a_{n-2} && = \binom{1}{0}a_n - \binom{2}{2}a_{n-2},\\
c_n &= b_n - d_{n-1} = b_n - a_{n-2} - b_{n-2} && = \binom{2}{0}a_n - \binom{3}{2}a_{n-2} + \binom{4}{4}a_{n-4},\\
d_n &= c_n - c_{n-1} = c_n - a_{n-2} - b_{n-2} - c_{n-2} && = \binom{3}{0}a_n - \binom{4}{2}a_{n-2} + \binom{5}{4}a_{n-4} - \binom{6}{6}a_{n-6},\\
e_n &= d_n - b_{n-1} = d_n - a_{n-2} - b_{n-2} - c_{n-2} - d_{n-2} && = \binom{4}{0}a_n - \binom{5}{2}a_{n-2} + \binom{6}{4}a_{n-4} - \binom{7}{6}a_{n-6} + \binom{8}{8}a_{n-8}.
\end{aligned} \tag{4}$$

Обозначим $A(z) = \sum_{n\ge 0} a_n z^n$, ..., $E(z) = \sum_{n\ge 0} e_n z^n$ и определим многочлены

$$\begin{aligned}
q_m(z) &= \binom{m}{0} - \binom{m+1}{2}z^2 + \binom{m+2}{4}z^4 - \cdots =\\
&= \sum_{k\ge 0}\binom{m+k}{2k}(-1)^k z^{2k} = \sum_{0\le k\le m}\binom{2m-k}{k}(-1)^{m-k}z^{2m-2k}.
\end{aligned} \tag{5}$$

Результат (4) кратко можно истолковать так, что $B(z) - q_1(z)\times A(z)$, ..., $E(z) - q_4(z)A(z)$ сводятся к конечным суммам, соответствующим граничным условиям, а именно значениям a_{-1}, a_{-2}, a_{-3}, ..., которые появляются в (4) (при небольших n), но не в $A(z)$. Чтобы получить подходящие граничные условия, применим рекуррентное соотношение в обратную сторону для отрицательных уровней до уровня -8:

n	a_n	b_n	c_n	d_n	e_n
0	1	0	0	0	0
−1	0	0	0	0	1
−2	1	−1	0	0	0
−3	0	0	0	−1	2
−4	2	−3	1	0	0
−5	0	0	1	−4	5
−6	5	−9	5	−1	0
−7	0	−1	6	−14	14
−8	14	−28	20	−7	1

(Для семи лент таблица была бы аналогичной, однако строки с нечетными n были бы сдвинуты вправо на один элемент.) Тайна последовательности a_0, a_{-2}, a_{-4}, $\ldots = 1, 1, 2, 5, 14, \ldots$ мгновенно раскрывается специалистом по информатике, так как эта последовательность встречается в связи с очень многими рекуррентными алгоритмами (например, в упр. 2.2.1-4 и формуле 2.3.4.4-13)). Итак, мы предполагаем, что в случае T лент

$$\begin{aligned}
a_{-2n} &= \binom{2n}{n}\frac{1}{n+1} && \text{при } 0\le n\le T-2;\\
a_{-2n-1} &= 0 && \text{при } 0\le n\le T-3.
\end{aligned} \tag{6}$$

Чтобы проверить правильность этого предположения, достаточно показать, что (6) и (4) приводят к правильным результатам для уровней 0 и 1. Для уровня 1 это очевидно, а для уровня 0 нам надо проверить, что

$$\binom{m}{0}a_0 - \binom{m+1}{2}a_{-2} + \binom{m+2}{4}a_{-4} - \binom{m+3}{6}a_{-6} + \cdots = \sum_{k\ge 0}\binom{m+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1} = \delta_{m0} \tag{7}$$

для $0\le m\le T-2$. К счастью, эту сумму можно вычислить стандартными методами (это "задача 2", один из основных примеров в тексте п. 4.2.6).

Теперь можно вычислить коэффициенты $B(z)-q_1(z)A(z)$ и т. д. Рассмотрим, например, коэффициент при z^{2m} в $D(z)-q_3(z)A(z)$. Он равен

$$\sum_{k\ge 0}\binom{3+m+k}{2m+2k}(-1)^{m+k}a_{-2k}=\sum_{k\ge 0}\binom{3+m+k}{2m+2k}\binom{2k}{k}\frac{(-1)^{m+k}}{k+1}=$$
$$=(-1)^m\left(\binom{2+m}{2m-1}-\binom{3+m}{2m}\right)=$$
$$=(-1)^{m+1}\binom{2+m}{2m}$$

из результата "задачи 3" в п. 1.2.6. Таким образом, мы вывели формулы

$$\begin{aligned}A(z)&=q_0(z)A(z);\\ B(z)&=q_1(z)A(z)-q_0(z);\\ C(z)&=q_2(z)A(z)-q_1(z);\\ D(z)&=q_3(z)A(z)-q_2(z);\\ E(z)&=q_4(z)A(z)-q_3(z).\end{aligned}\tag{8}$$

Кроме того, имеем $e_{n+1}=a_n$; следовательно, $zA(z)=E(z)$ и

$$A(z)=q_3(z)/(q_4(z)-z).\tag{9}$$

Производящие функции были выражены при помощи q-многочленов, поэтому мы хотим лучше изучить q. В этом отношении полезно упр. 1.2.9-15, так как оно дает выражение в замкнутом виде, которое может быть записано как

$$q_m(z)=\frac{((\sqrt{4-z^2}+iz)/2)^{2m+1}+((\sqrt{4-z^2}-iz)/2)^{2m+1}}{\sqrt{4-z^2}}\tag{10}$$

Все упрощается, если теперь положить $z=2\sin\theta$:

$$q_m(2\sin\theta)=\frac{(\cos\theta+i\sin\theta)^{2m+1}+(\cos\theta-i\sin\theta)^{2m+1}}{2\cos\theta}=$$
$$=\frac{\cos(2m+1)\theta}{\cos\theta}.\tag{11}$$

(Это совпадение заставляет думать, что многочлены $q_m(z)$ хорошо известны в математике; и действительно, взглянув в соответствующие таблицы, видим, что $q_m(z)$, по существу, многочлен Чебышева второго рода, а именно $(-1)^mU_{2m}(z/2)$ в обычных обозначениях.)

Теперь можно определить корни знаменателя в (9): $q_4(2\sin\theta)=2\sin\theta$ сводится к

$$\cos 9\theta=2\sin\theta\cos\theta=\sin 2\theta.$$

Решения этого соотношения получаем, если только $\pm 9\theta=2\theta+\left(2n-\frac{1}{2}\right)\pi$; все такие θ дают корни знаменателя в (9) при условии, что $\cos\theta\ne 0$. (Если $\cos\theta=0$, то $q_m(\pm 2)=\pm(2m+1)$, никогда не равно ± 2.) Следовательно, получаем 8 различных корней:

$$q_4(z)-z=0\qquad\text{при } z=2\sin\frac{-5}{14}\pi,\quad 2\sin\frac{-1}{14}\pi,\quad 2\sin\frac{3}{14}\pi,$$
$$2\sin\frac{-7}{22}\pi,\quad 2\sin\frac{-3}{22}\pi,\quad 2\sin\frac{1}{22}\pi,\quad 2\sin\frac{5}{22}\pi,\quad 2\sin\frac{9}{22}\pi.$$

Так как $q_4(z)$—многочлен степени 8, мы учли все корни. Первые три из этих значений дают $q_3(z)=0$, так что $q_3(z)$ и $q_4(z)-z$ имеют общим делителем многочлен третьей степени. Остальные пять корней управляют асимптотическим поведением коэффициентов $A(z)$, если разложить (9) в элементарные дроби.

Перейдя к рассмотрению общего случая T лент, положим $\theta_k=(4k+1)\pi/(4T-2)$. Производящая функция $A(z)$ для T-ленточных каскадных чисел принимает вид

$$\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\frac{\cos^2\theta_k}{1-z/(2\sin\theta_k)}\tag{12}$$

(см. упр. 8); следовательно,

$$a_n = \frac{4}{2T-1} \sum_{-T/2<k<\lfloor T/2 \rfloor} \cos^2 \theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n . \tag{13}$$

Соотношения (8) приводят теперь к аналогичным формулам:

$$\begin{aligned} b_n &= \frac{4}{2T-1} \sum_{-T/2<k<\lfloor T/2 \rfloor} \cos \theta_k \cos 3\theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n ; \\ c_n &= \frac{4}{2T-1} \sum_{-T/2<k<\lfloor T/2 \rfloor} \cos \theta_k \cos 5\theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n ; \\ d_n &= \frac{4}{2T-1} \sum_{-T/2<k<\lfloor T/2 \rfloor} \cos \theta_k \cos 7\theta_k \left(\frac{1}{2 \sin \theta_k} \right)^n \end{aligned} \tag{14}$$

и т. д. В упр. 9 показано, что эти уравнения справедливы для всех $n \geq 0$, а не только для больших n. В каждой сумме член с $k = 0$ значительно превосходит все остальные, особенно если n достаточно велико; следовательно, "отношение роста" есть

$$\frac{1}{2 \sin \theta_0} = \frac{2}{\pi} T - \frac{1}{\pi} + \frac{\pi}{48T} + O(T^{-2}). \tag{15}$$

Каскадная сортировка впервые была исследована У. К. Картером [Proc. IFIP Congress (1962), 62–66], который получил численные результаты для небольших значений T, и Дэвидом Фергюсоном [*CACM*, **7** (1964), 297], который открыл первые два члена в асимптотическом поведении (15) отношения роста. Летом 1964 г. Р. У. Флойд получил явный вид $1/(2 \sin \theta_0)$ для отношения роста, так что точные формулы могли быть использованы для всех T. Глубокий анализ каскадных чисел был независимо выполнен Дж. Н. Рэйни [*Canadian J. Math.*, **18** (1966), 332–349], который наткнулся на них совершенно другим путем, не имея дела с сортировкой. Рэйни подметил аналогию с диагоналями (рис. 73) и вывел много других интересных свойств этих чисел. Флойд и Рэйни в своих доказательствах оперировали с матрицами (см. упр. 6).

Модификация каскадной сортировки. Если добавлена еще одна лента, то почти все время перемотки в течение каскадной сортировки можно совместить. Например, мы можем сливать T1–T5 на T7, затем T1–T4 на T6, затем T1–T3 на T5 (которая к этому моменту уже перемотана), затем T1–T2 на T4 и начать следующий проход, когда на T4 будет перемотано сравнительно немного данных. Эффективность этого процесса можно предсказать на основании изложенного выше анализа каскадного метода (дальнейшие подробности см. в п. 5.4.6).

Схема "компромиссного слияния", которая включает многофазную и каскадную схемы как частные случаи, была предложена Д. Э. Кнутом в [*CACM*, **6** (1963), 585–587]. Каждая фаза состоит из $(T-1)$-путевого, $(T-2)$-путевого, ..., P-путевого слияний, где P—любое фиксированное число между 1 и $T-1$. Если $P = T-1$, то это—многофазный метод; если $P = 1$, это—чистый каскадный метод; если $P = 2$, это—каскадный метод без фаз копирования. Анализ такой схемы был проделан Ч. Радке [*IBM Systems J.*, **5** (1966), 226–247] и У. Х. Буржем [Proc. IFIP Congress, **1** (1971), 454—459]. Бурж нашел производящую функцию $\sum T_n(x) z^n$ для каждого (P,T)-компромиссного слияния, обобщающую соотношение (5.4.2-16); он показал, что наилучшее значение P (с точки зрения наименьшего числа обрабатываемых начальных отрезков), как функции от S при $S \to \infty$ (если непосредственно использовать схему распределения и пренебречь временем перемотки), есть соответственно $(2, 3, 3, 4, 4, 4, 3, 3, 4)$ при $T = (3, 4, 5, 6, 7, 8, 9, 10, 11)$. Эти значения P с ростом T сильнее отклоняются в сторону каскадного, а не многофазного метода, и оказывается, что компромиссное слияние никогда не станет существенно лучше каскадного. С другой стороны, при оптимальном выборе уровней и распределении фиктивных отрезков, как описано в п. 5.4.2, чистый многофазный метод кажется наилучшим среди всех компромиссных слияний. К сожалению, оптимальное распределение сравнительно трудно реализовать.

Т. Л. Йонсен [*BIT*, **6** (1966), 129–143] исследовал сочетания сбалансированного и многофазного слияний; модификация сбалансированного слияния с совмещением перемоток была предложена М. Готцем [Digital Computer User's Handbook, ed. by. M. Klerer and G. A. Korn (New York: McGraw-Hill, 1967), 1.311–1.312]; можно представить себе и многие другие гибридные схемы.

Упражнения

1. [10] Используя табл. 1, сравните каскадное слияние с описанной в п. 5.4.2 версией многофазного слияния с "расщеплением лент". Какой метод лучше? (Временем перемотки пренебречь.)

>2. [22] Сравните каскадную сортировку с тремя лентами, использующую алгоритм C, и многофазную сортировку с тремя лентами, использующую алгоритм 5.4.2D Какие сходства и различия вы заметите?
3. [20] Составьте таблицу, показывающую, что происходит при сортировке на шести лентах 100 начальных отрезков при помощи алгоритма C.
4. [M20] (Дж. Н. Рэйни.) "Каскадное распределение n-го уровня" есть мультимножество, определенное следующим образом (в случае шести лент): $\{1,0,0,0,0\}$ есть каскадное распределение нулевого уровня; если $\{a,b,c,d,e\}$—каскадное распределение n-го уровня, то $\{a+b+c+d+e, a+b+c+d, a+b+c, a+b, a\}$ будет каскадным распределением $(n+1)$-го уровня (Так как мультимножество не упорядочено, то из единственного распределения n-го уровня можно образовать до 5 различных распределений $(n+1)$-го уровня.) (a) Докажите, что *любое* мультимножество $\{a,b,c,d,e\}$ из взаимно простых чисел является каскадным распределением n-го уровня при некотором n. (b) Докажите, что распределение, используемое в каскадной сортировке, *оптимально* в том смысле, что если $\{a,b,c,d,e\}$—любое распределение n-го уровня, причем $a \ge b \ge c \ge d \ge e$, то будем иметь $a \le a_n$, $b \le b_n$, $c \le c_n$, $d \le d_n$, $e \le e_n$, где $\{a_n, b_n, c_n, d_n, e_n\}$—распределение, определенное в (1).
>5. [20] Докажите, что каскадные числа, определенные в (1), удовлетворяют закону

$$a_k a_{n-k} + b_k b_{n-k} + c_k c_{n-k} + d_k d_{n-k} + e_k e_{n-k} = a_n \qquad \text{при } 0 \le k \le n.$$

[*Указание:* для лучшего понимания этого соотношения рассмотрите, сколько отрезков различной длины выводится в течение k-го прохода полной каскадной сортировки.]

6. [M20] Найдите 5×5-матрицу Q, такую, что первая строка Q^n содержит каскадные числа для шести лент $a_n b_n c_n d_n e_n$ при всех $n \ge 0$.
7. [M20] При условии что каскадное слияние применяется к точному распределению a_n начальных отрезков, найдите формулу для величины сэкономленной работы, когда исключается однопутевое слияние.
8. [BM23] Выведите формулу (12).
9. [BM26] Выведите формулу (14).
>10. [M28] Вместо системы (4) для изучения каскадных чисел воспользуйтесь в качестве исходных тождествами

$$\begin{aligned}
e_n &= a_{n-1} &&= \binom{1}{1} a_{n-1};\\
d_n &= 2a_{n-1} - e_{n-2} &&= \binom{2}{1} a_{n-1} - \binom{3}{3} a_{n-3};\\
c_n &= 3a_{n-1} - d_{n-2} - 2e_{n-2} &&= \binom{3}{1} a_{n-1} - \binom{4}{3} a_{n-3} + \binom{5}{5} a_{n-5}
\end{aligned}$$

и т. д. Полагая

$$r_m(z) = \binom{m}{1} z - \binom{m+1}{3} z^3 + \binom{m+2}{5} z^5 - \dots,$$

выразите $A(z)$, $B(z)$ и т. д. через эти r-многочлены.

11. [M38] Пусть

$$f_m(z) = \sum_{0 \le k \le m} \binom{\lfloor (m+k)/2 \rfloor}{k} (-1)^{\lceil k/2 \rceil} z^k.$$

Докажите, что производящая функция $A(z)$ для T-ленточных каскадных чисел равна $f_{T-3}(z)/f_{T-1}(z)$, причем числитель и знаменатель этого выражения не имеют общих делителей.

12. [M40] Докажите, что схема распределения Фергюсона оптимальна в том смысле, что при любом другом методе размещения фиктивных отрезков, удовлетворяющем (2), во время первого прохода будет обрабатываться не меньше начальных отрезков, *при условии* что во время этого прохода используется стратегия шагов C7–C9.
13. [40] В тексте предлагается большую часть времени перемотки совмещать путем добавления дополнительной ленты. Разработайте эту идею. (Например, схема, изложенная в тексте, включает ожидание перемотки ленты T4; не будет ли лучше исключить T4 из первой фазы слияния следующего прохода?)

Чтение ленты в обратном направлении Многие лентопротяжные устройства позволяют читать ленту в направлении, противоположном тому, в котором шла запись на нее. Схемы слияния, с которыми мы встречались до сих пор, всегда записывают информацию на ленту в прямом направлении, затем перематывают ленту, читают ее в прямом направлении и вновь перематывают. (Файлы на ленте, следовательно, ведут себя как очереди "первым включается—первым исключается".) Обратное чтение позволяет обойтись без обеих операций перемотки: мы записываем на ленту в прямом направлении и

читаем ее в обратном. (В этом случае файлы ведут себя как стеки, поскольку здесь действует правило "последним включается—первым исключается".)

Схемы сбалансированного, многофазного и каскадного слияний можно приспособить для обратного чтения. Основное отличие состоит в том, что *слияние изменяет порядок отрезков,* если мы читаем в прямом направлении и записываем в обратном. Если два отрезка находятся на ленте в возрастающем порядке, то их можно слить, читая в обратном направлении, но при этом порядок станет убывающим. Полученные таким путем убывающие отрезки станут возрастающими на следующем проходе; таким образом, алгоритм слияния должен уметь работать с отрезками обоих направлений. Программисту, впервые столкнувшемуся с обратным чтением, может показаться, что он стоит на голове!

В качестве примера обратного чтения рассмотрим процесс слияния 8 начальных отрезков с использованием *сбалансированного* слияния на 4 лентах. Можно следующим образом записать наши действия:

	$T1$	$T2$	$T3$	$T4$	
Проход 1.	$A_1A_1A_1A_1$	$A_1A_1A_1A_1$	—	—	Начальное распределение
Проход 2.	—	—	D_2D_2	D_2D_2	Слияние на $T3$ и $T4$
Проход 3.	A_4	A_4	—	—	Слияние на $T1$ и $T2$
Проход 4.	—	—	D_8	—	Окончательное слияние на $T3$

Здесь A_r обозначает отрезок, имеющий относительную длину r и расположенный в возрастающем порядке, если лента читается в прямом направлении, как в предыдущих наших примерах; D_r—аналогичное обозначение для убывающего отрезка длины r. Во время 2-го прохода возрастающие отрезки становятся убывающими: они оказываются убывающими при вводе, так как мы читаем в обратном направлении. Они вновь изменяют ориентацию на 3-м проходе.

Заметим, что описанный процесс завершается результатом на ленте $T3$ в *убывающем* порядке. Если это плохо (что зависит от того, должен ли результат читаться в обратном направлении, или же лента, содержащая его, должна быть снята и отложена для будущего использования), мы можем скопировать его на другую ленту, обратив направление. Более быстрым способом была бы перемотка $T1$ и $T2$ после 3-го прохода, при этом во время 4-го прохода получается A_8. Еще быстрее было бы начать с восьми *убывающих* отрезков на 1-м проходе, так как это поменяет местами все A и D. Однако для сбалансированного слияния 16 начальных отрезков потребовалось бы, чтобы начальные отрезки были возрастающими, а так как мы обычно не знаем заранее, сколько начальных отрезков будет образовано, то необходимо выбрать одно постоянное направление. Следовательно, идея перемотки после 3-го прохода, вероятно, наилучшая.

Каскадное слияние преобразуется таким же способом. Рассмотрим, например, сортировку 14 начальных отрезков на четырех лентах:

	$T1$	$T2$	$T3$	$T4$
Проход 1.	$A_1A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1$	$A_1A_1A_1$	—
Проход 2.	—	D_1	D_2D_2	$D_3D_3D_3$
Проход 3.	A_6	A_5	A_3	—
Проход 4.	—	—	—	D_{14}

Снова вместо D_{14} можно получить A_{14}, если перемотать $T1$, $T2$, $T3$ непосредственно перед последним проходом. Заметим, что это "чистое" каскадное слияние в том смысле, что все однопутевые слияния выполнены явным образом. Если бы мы запретили операции копирования, как в алгоритме 5.4.3D, то после 2-го прохода столкнулись бы с ситуацией

$$A_1 \qquad — \qquad D_2D_2 \qquad D_3D_3D_3$$

и было бы невозможно продолжать работу, используя трехпутевое слияние, так как мы не можем сливать отрезки противоположных направлений! Можно было бы избежать операции копирования $T1$ на $T2$, если перемотать $T1$ и начать читать ее в прямом направлении во время следующей фазы слияния (в то время как $T3$ и $T4$ читаются в обратном направлении). Но тогда пришлось бы вновь перемотать $T1$ после слияния, так что этот прием заменяет одно копирование на две перемотки.

Таким образом, метод распределения алгоритма 5.4.3C работает для обратного чтения не столь эффективно, как для прямого чтения; временные затраты резко возрастают каждый раз, когда число начальных отрезков проходит через "точное" каскадное распределение. Чтобы получить более гладкий проход между точными каскадными распределениями, можно использовать иной метод распределения (см. упр. 17).

Обратное чтение в многофазном слиянии. На первый взгляд (и даже на второй и третий!) схема многофазного слияния кажется совершенно неподходящей для обратного чтения. Предположим, например, что имеются 13 начальных отрезков и три ленты.

	$T1$	$T2$	$T3$
Фаза 1.	$A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1A_1A_1A_1$	—
Фаза 2.	—	$A_1A_1A_1$	$D_2D_2D_2D_2D_2$

Здесь мы встаем в тупик; можно было бы перемотать $T2$ или $T3$ и затем читать их в прямом направлении, в то время как остальные ленты—в обратном. Но это сильно запутало бы дело, и выгода от обратного чтения была бы относительно мала.

Остроумная идея, спасающая положение, состоит в том, чтобы *чередовать направления отрезков на каждой ленте.* Тогда слияние может происходить вполне согласованно:

Фаза 1.	$A_1 D_1 A_1 D_1 A_1$	$D_1 A_1 D_1 A_1 D_1 A_1 D_1 A_1$	–
Фаза 2.	–	$D_1 A_1 D_1$	$D_2 A_2 D_2 A_2 D_2$
Фаза 3.	$A_3 D_3 A_3$	–	$D_2 A_2$
Фаза 4.	A_3	$D_5 A_5$	–
Фаза 5.	–	D_5	D_8
Фаза 6.	A_{13}	–	–

Этот принцип был кратко упомянут Р. Л. Гилстэдом в его ранней статье о многофазном слиянии, более полно он описал его в *CACM,* **6** (1963), 220–223.

Этот ADA-метод четко работает для многофазного слияния с *любым* числом лент; можно показать, что A и D согласуются соответствующим образом на каждой фазе, при том только условии, что проход начального распределения порождает чередующиеся отрезки A и D на каждой ленте и что каждая лента кончается отрезком A (или каждая лента кончается отрезком D). Так как последний отрезок, записываемый в файл вывода во время одной фазы, имеет направление, противоположное направлению последнего использованного отрезка из файла ввода, то следующая фаза всегда находит свои отрезки с надлежащей ориентацией. Далее, мы видели в упр. 5.4.2-13, что большинство точных фибоначчиевых распределений требует *нечетного* числа отрезков на одной ленте (окончательной выводной ленте) и четного числа отрезков на всех остальных лентах. Если $T1$ предназначена для конечного вывода, то мы можем, следовательно, гарантировать, что все ленты будут кончаться отрезком A, если ленту $T1$ начнем с A, а все остальные ленты—с D. Можно использовать метод распределения, аналогичный алгоритму 5.4.2D, изменив его таким образом, чтобы распределение на каждом уровне имело в качестве выводной ленты $T1$. (Мы пропускаем уровни $1, T+1, 2T+1, \ldots$, так как это те уровни, на которых конечной выводной лентой является первоначально пустая лента.) Например, в случае шести лент можно использовать вместо (5.4.2-1) следующее распределение отрезков:

Уровень	$T1$	$T2$	$T3$	$T4$	$T5$	Сумма	Окончательный вывод на ленте
0	1	0	0	0	0	1	$T1$
2	1	2	2	2	2	9	$T1$
3	3	4	4	4	2	17	$T1$
4	7	8	8	6	4	33	$T1$
5	15	16	14	12	8	65	$T1$
6	31	30	28	24	16	129	$T1$
8	61	120	116	108	92	497	$T1$

(1)

Таким образом, на $T1$ всегда помещается нечетное число отрезков, тогда как на ленты с $T2$ по $T5$—четные числа (в убывающем порядке для упрощения присвоения фиктивных отрезков). Такое распределение имеет то преимущество, что конечная выводная лента известна заранее независимо от числа начальных отрезков, которые придется обрабатывать. Оказывается (см. упр. 3), что если используется эта схема, то результат всегда будет на $T1$ в возрастающем порядке.

Другой способ осуществить распределение для многофазной схемы с обратным чтением был предложен Д. Т. Гудвином и Дж. Л. Венном [*CACM,* **7** (1964), 315]. Мы можем распределять отрезки, почти как в алгоритме 5.4.2D, начиная с D-отрезка на каждой ленте. Когда ввод исчерпан, мы представляем себе фиктивный A-отрезок расположенным в начале единственной "нечетной" ленты, если только не достигнуто распределение со всеми нечетными числами. Остальные фиктивные отрезки мы представляем себе расположенными в конце лент или сгруппированными в пары в середине. Вопрос об оптимальном размещении фиктивных отрезков анализируется ниже в упр. 5.

Оптимальные схемы слияния. До сих пор мы обсуждали различные схемы слияния с лентами, не пытаясь найти метод, "наилучший из возможных". Определение оптимальной схемы кажется особенно сложным в случае прямого чтения, где взаимодействие времени перемотки и времени слияния с трудом поддается анализу. С другой стороны, если слияние осуществляется посредством обратного чтения и прямой записи, то, по существу, все перемотки устраняются и оказывается возможным дать довольно хорошую характеристику оптимальным схемам слияния. Ричард М. Карп предложил несколько интересных подходов к этой задаче, и мы завершаем этот пункт обсуждением развитой им теории.

Во-первых, нам необходим более удобный способ описания схем слияния вместо довольно таинственных таблиц "содержимого лент", которые использовались выше. Карп предложил два таких способа—*векторное представление схемы слияния* и *представление в виде дерева.* Оба представления полезны на практике, и мы опишем их по очереди.

Векторное представление схемы слияния состоит из последовательности "векторов слияния" $y^m \ldots y^1 y^0$, каждый из которых имеет T компонент; $y^{(i)}$ следующим образом изображает i-й с конца шаг слияния:

$$y_j^{(i)} = \begin{cases} +1, & \text{если лента с номером } j \text{ является вводной для данного слияния;} \\ 0, & \text{если лента с номером } j \text{ не используется в данном слиянии;} \\ -1, & \text{если на ленту с номером } j \text{ выводится результат данного слияния.} \end{cases} \tag{2}$$

Таким образом, ровно одна компонента $y^{(i)}$ равна -1, остальные компоненты равны 0 и 1. Итоговый вектор $y^{(0)}$ особый; это единичный вектор, имеющий 1 в позиции j, если окончательный отсортированный результат оказывается на устройстве j, и 0 в остальных местах. Из этого определения следует, что векторная сумма

$$v^{(i)} = y^{(i)} + y^{(i-1)} + \cdots + y^{(0)} \tag{3}$$

представляет собой распределение отрезков на лентах непосредственно перед i-м с конца шагом слияния, причем на ленте j находится $v_j^{(i)}$ отрезков. В частности, по $v^{(m)}$ можно судить, сколько отрезков помещает на каждую ленту начальный проход распределения.

Три схемы слияния, описанные ранее в этом пункте в табличной форме, имеют следующие векторные представления:

Сбалансированная ($T=4$, $S=8$)	Каскадная ($T=4$, $S=14$)	Многофазная ($T=3$, $S=13$)
$v^{(7)} = (\ 4,\ 4,\ 0,\ 0)$	$v^{(10)} = (\ 6,\ 5,\ 3,\ 0)$	$v^{(12)} = (\ 5,\ 8,\ 0)$
$y^{(7)} = (+1, +1, -1,\ 0)$	$y^{(10)} = (+1, +1, +1, -1)$	$y^{(12)} = (+1, +1, -1)$
$y^{(6)} = (+1, +1,\ 0, -1)$	$y^{(9)} = (+1, +1, +1, -1)$	$y^{(11)} = (+1, +1, -1)$
$y^{(5)} = (+1, +1, -1,\ 0)$	$y^{(8)} = (+1, +1, +1, -1)$	$y^{(10)} = (+1, +1, -1)$
$y^{(4)} = (+1, +1,\ 0, -1)$	$y^{(7)} = (+1, +1, -1,\ 0)$	$y^{(9)} = (+1, +1, -1)$
$y^{(3)} = (-1,\ 0, +1, +1)$	$y^{(6)} = (+1, +1, -1,\ 0)$	$y^{(8)} = (+1, +1, -1)$
$y^{(2)} = (\ 0, -1, +1, +1)$	$y^{(5)} = (+1, -1,\ 0,\ 0)$	$y^{(7)} = (-1, +1, +1)$
$y^{(1)} = (+1, +1, -1,\ 0)$	$y^{(4)} = (-1, +1, +1, +1)$	$y^{(6)} = (-1, +1, +1)$
$y^{(0)} = (\ 0,\ 0,\ 1,\ 0)$	$y^{(3)} = (\ 0, -1, +1, +1)$	$y^{(5)} = (-1, +1, +1)$
	$y^{(2)} = (\ 0,\ 0, -1, +1)$	$y^{(4)} = (+1, -1, +1)$
	$y^{(1)} = (+1, +1, +1, -1)$	$y^{(3)} = (+1, -1, +1)$
	$y^{(0)} = (\ 0,\ 0,\ 0,\ 1)$	$y^{(2)} = (+1, +1, -1)$
		$y^{(1)} = (-1, +1, +1)$
		$y^{(0)} = (\ 1,\ 0,\ 0)$

Может показаться неудобным нумеровать эти векторы с конца так, чтобы $y^{(m)}$ появлялся первым, а $y^{(0)}$—последним, но эта особая точка зрения оказывается предпочтительной при разработке теории. Для поиска оптимального метода неплохо начать с отсортированного вывода и представить себе, как его можно "разлить" на различные ленты, затем "разлить" их и т. д., рассматривая последовательные распределения $v^{(0)}$, $v^{(1)}$, $v^{(2)}$, ... в порядке, обратном тому, в котором они в действительности появляются в процессе сортировки. Фактически именно этот подход был уже использован нами при анализе многофазного и каскадного слияния.

Каждая схема слияния, очевидно, имеет векторное представление. И обратно, как легко видеть, последовательность векторов $y^{(m)} \ldots y^{(1)} y^{(0)}$ соответствует реальной схеме слияния тогда и только тогда, когда выполняются следующие три условия:

i) вектор $y^{(0)}$ является единичным;
ii) ровно одна компонента вектора $y^{(i)}$ равна -1, все остальные компоненты равны 0 или $+1$, $m \geq i \geq 1$;
iii) все компоненты вектора $y^{(i)} + \ldots + y^{(1)} + y^{(0)}$ неотрицательны, $m \geq i \geq 1$.

Представление схемы слияния в виде дерева дает другое изображение той же информации. Мы строим дерево с одним внешним "листовым" узлом для каждого начального отрезка и с одним внутренним узлом для каждого отрезка, полученного в результате слияния, таким образом, что потомками любого внутреннего узла являются отрезки, из которых он был сформирован. Каждый внутренний узел помечается номером шага, на котором был образован соответствующий отрезок, при этом шаги нумеруются в

обратном порядке, как в векторном представлении; кроме того, ребра непосредственно над каждым узлом помечаются именем ленты, на которой оказывается этот отрезок. Например, три приведенные выше схемы имеют представления в виде дерева, изображенные на рис. 76, если мы назовем ленты A, B, C, D, а не $T1$, $T2$, $T3$, $T4$.

Это удобное и наглядное представление многих существенных свойств схемы слияния; например, если отрезок на уровне 0 дерева (корень) должен быть возрастающим, то отрезки на уровне 1 должны быть убывающими, отрезки на уровне 2—возрастающими и т. д.; некоторый начальный отрезок является возрастающим тогда и только тогда, когда соответствующий внешний узел находится на уровне с четным номером. Далее, суммарное количество начальных отрезков, обрабатываемых при слиянии (не включая начальное распределение), в точности равно *длине внешнего пути* дерева, так как каждый начальный отрезок на уровне k обрабатывается ровно k раз.

Picture: Рис. 76. Представления трех схем слияния в виде дерева.

Любая схема слияния имеет представление в виде дерева, но не каждое дерево определяет схему слияния. Дерево, внутренние узлы которого помечены числами от 1 до m и ребра которого помечены именами лент, изображает правильную схему слияния с обратным чтением тогда и только тогда, когда

a) никакие два ребра, смежные с одним внутренним узлом, не имеют одинакового имени ленты;

b) если $i > j$ и если A есть имя ленты, то дерево не содержит конфигурации

Picture: p.362

c) если $i < j < k < l$ и если A—имя ленты, то дерево не содержит таких пар:

Picture: p.363.1

Условие (a) очевидно, так как вводные и выводная ленты слияния должны быть различны; условие (b) также очевидно. Условие "непересечения" (c) отражает характерное для операций обратного чтения ленты ограничение "последним включается — первым исключается". Отрезок, образованный на шаге k, должен быть удален прежде отрезка, сформированного ранее на той же ленте; следовательно, конфигурации (4) невозможны. Нетрудно проверить, что любое помеченное дерево, удовлетворяющее условиям (a), (b), (c), действительно соответствует некоторой схеме слияния с обратным чтением.

Если имеется T ленточных устройств, то из условия (a) следует, что степень каждого внутреннего узла равна $T-1$ или меньше. Приписывание подходящих меток всем таким деревьям не всегда возможно; например, если $T = 3$, то не существует схемы слияния с деревом вида

Picture: p.363.2

Такая форма дерева привела бы к оптимальной схеме слияния, если бы мы смогли соответствующим образом приписать номера шагов и номера лент, поскольку это единственное дерево с минимальной длиной внешнего пути среди деревьев, имеющих четыре внешних узла. Но в силу симметрии этой диаграммы имеется, по существу, только один способ пометить ее, соблюдая условия (a) и (b):

Picture: p.363.3

Однако при этом нарушается условие (c). Дерево, которое может быть помечено в соответствии с упомянутыми условиями с использованием T или меньше имен лент, называется T-lifo[3]деревом.

Другой способ характеризовать все помеченные деревья, которые могут возникнуть из схемы слияния, состоит в том, чтобы рассмотреть, как все подобные деревья могут быть "выращены". Начнем с некоторого имени ленты, скажем A, и с ростка дерева

Picture: p.364.1

Шаг i роста дерева состоит в выборе различных имен лент B, B_1, B_2, ..., B_k и замене *позже всего образованного* узла, соответствующего B,

Picture: p.364.2

[3] "Lifo"—аббревиатура для "last in—first out" (последним включается— первым исключается).—*Прим. перев.*

Это правило "последним образован—первым растет" в точности описывает способ построения представления в виде дерева непосредственно из векторного представления.

Определение строго оптимальной T-ленточной схемы слияния, т. е. дерева, имеющего минимальную длину пути среди всех T-lifo деревьев с данным числом внешних узлов, кажется весьма трудной задачей. Следующая неочевидная схема, например, оказывается наилучшим способом слить семь начальных отрезков, имея четыре ленты и читая в обратном направлении: Однопутевое слияние необходимо по существу для достижения оптимума! (См. упр. 8.) С другой стороны, не так трудно дать конструкции, *асимптотически* оптимальные для любого фиксированного T.

Пусть $K_T(n)$—минимальная длина внешнего пути, достижимая в T-lifo дереве с n внешними узлами. Используя теорию, развитую в п. 2.3.4.5, нетрудно доказать, что

$$K_T(n) \geq nq - \lfloor ((T-1)^q - n)/(T-2) \rfloor, \quad q = \lceil \log_{T-1} n \rceil, \tag{9}$$

так как это минимальная длина внешнего пути любого дерева с n внешними узлами и степенью любого узла $< T$. К настоящему моменту известны относительно немногие точные значения $K_T(n)$. Здесь приведены некоторые верхние оценки, которые, вероятно, точны:

$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$K_3(n) \leq$	0	2	5	9	12	16	21	25	30	34	39	45	50	56	61
$K_4(n) \leq$	0	2	3	6	8	11	14	17	20	24	27	31	33	37	40

(10)

Карп обнаружил, что любое дерево с внутренними узлами степени $< T$ является *почти* T-lifo деревом в том смысле, что оно может быть сделано T-lifo с помощью замены некоторых внешних узлов на однопутевые слияния. Фактически построение подходящей расстановки меток выполняется, довольно просто. Пусть A—конкретное имя ленты. Будем действовать следующим образом:

Шаг 1. Приписать имена лент ребрам диаграммы дерева любым способом, совместимым с условием (a), указанным выше, однако так, чтобы специальное имя A использовалось только в самом левом ребре ветви.

Шаг 2. Заменить каждый внешний узел вида

Picture: p.365

если только $B \neq A$.

Шаг 3. Занумеровать внутренние узлы в прямом порядке[4]. Результатом будет расстановка меток, удовлетворяющая условиям (a), (b), (c).

Например, если мы начнем с дерева

Picture: p.366.1

и трех лент, то эта процедура могла бы приписать метки таким образом:

Picture: p.366.2

Нетрудно проверить, что конструкция Карпа удовлетворяет дисциплине "последним образован—первым растет" в силу свойств прямого порядка (см. упр. 12).

Заметим, что результатом этого построения является схема слияния, в которой все начальные отрезки появляются на ленте A. Это предполагает следующую схему распределения и сортировки, которую можно назвать слиянием в *прямом порядке.*

P1. Распределять начальные отрезки на ленту A, пока ввод не будет исчерпан. Пусть S—число всех начальных отрезков.

P2. Выполнять приведенное выше построение, используя $(T-1)$-арное дерево с S внешними узлами с минимальной длиной пути, получая T-lifo дерево, длина внешнего пути которого превышает нижнюю границу (9) не более, чем на S.

P3. Сливать отрезки в соответствии с этой схемой. ∎

Результат в указанной схеме получается на какой угодно ленте. Но *эта схема имеет один серьезный изъян.* (Видит ли читатель, что именно здесь будет неправильно работать?) Дело в том, что схема требует, чтобы первоначально некоторые из отрезков на A были возрастающими, а некоторые—убывающими

[4] См. п. 2.3.1.—*Прим. перев.*

в зависимости от того, появляется ли соответствующий внешний узел на нечетном или четном уровне. Эту проблему можно разрешить, не зная S заранее, путем копирования отрезков, которые должны быть убывающими, на вспомогательную ленту (или ленты) непосредственно перед тем, как они требуются. Тогда суммарное количество операций, измеряемое в длинах начальных отрезков, окажется равным

$$S \log_{T-1} S + O(S). \tag{13}$$

Таким образом, слияние в прямом порядке определенно лучше многофазного или каскадного при $S \to \infty$; в действительности оно асимптотически *оптимально,* так как (9) показывает, что $S \log_{T-1} S + O(S)$— это наилучшее, что мы вообще можем надеяться получить с T лентами. С другой стороны, для сравнительно небольших значений S, обычно встречающихся на практике, слияние в прямом порядке весьма неэффективно; многофазный или каскадный методы проще и быстрее, если S относительно мало. Возможно, удастся изобрести простую схему распределения и слияния, которая сравнима с многофазной и каскадной для небольших S и асимптотически оптимальна при больших S.

Ниже в упражнениях второй части демонстрируется, как Карп аналогичным образом поставил вопрос для слияния с *прямым чтением.* Теория оказывается в этом случае значительно- более сложной, хотя были получены некоторые весьма интересные результаты.

Упражнения

(ПЕРВАЯ ЧАСТЬ)

1. [17] При слиянии с прямым чтением часто удобно отмечать конец каждого отрезка на ленте путем добавления искусственной "концевой" записи с ключом $+\infty$. Как следует видоизменить этот метод при обратном чтении?
2. [20] Будут ли столбцы таблицы, аналогичной (1), всегда неубывающими, или бывают случаи, когда нам придется "вычитать" отрезки с некоторой ленты при переходе от одного уровня к следующему?

>3. [20] Докажите, что метод многофазного распределения, описанный в связи с (1), дает после завершения сортировки на ленте T1, всегда отрезок A, если первоначально на T1 было $ADA\ldots$, а на T2–T5 было $DAD\ldots$.

4. [22] Как вы оцениваете идею выполнять многофазное слияние с обратным чтением, распределив все отрезки в *возрастающем* порядке и считая, что все позиции "D" первоначально заполнены фиктивными отрезками?

>5. [23] Какие формулы для строк чисел слияния вместо (8), (9), (10) и (11) из 5.4.2 будут справедливы для многофазного слияния с обратным чтением? Изобразите числа слияния для распределения пятого уровня на шести лентах, нарисовав диаграмму, аналогичную рис. 71(a).

6. [07] Каково векторное представление схемы слияния, представлением которой в виде дерева является (8)?
7. [16] Нарисуйте представление в виде дерева схемы слияния с обратным чтением, определенной следующей последовательностью векторов:

$$\begin{array}{llll}
v^{(33)} = (\ 20,\ \ 9,\ \ 5) & y^{(22)} = (+1,-1,+1) & y^{(10)} = (+1,+1,-1) \\
y^{(33)} = (+1,-1,+1) & y^{(21)} = (-1,+1,+1) & y^{(9)} = (+1,-1,+1) \\
y^{(32)} = (+1,+1,-1) & y^{(20)} = (+1,+1,-1) & y^{(8)} = (+1,+1,-1) \\
y^{(31)} = (+1,+1,-1) & y^{(19)} = (-1,+1,+1) & y^{(7)} = (+1,+1,-1) \\
y^{(30)} = (+1,+1,-1) & y^{(18)} = (+1,+1,-1) & y^{(6)} = (+1,+1,-1) \\
y^{(29)} = (+1,-1,+1) & y^{(17)} = (+1,+1,-1) & y^{(5)} = (-1,+1,+1) \\
y^{(28)} = (-1,+1,+1) & y^{(16)} = (+1,+1,-1) & y^{(4)} = (+1,-1,+1) \\
y^{(27)} = (+1,-1,+1) & y^{(15)} = (+1,+1,-1) & y^{(3)} = (-1,+1,+1) \\
y^{(26)} = (+1,-1,+1) & y^{(14)} = (+1,-1,+1) & y^{(2)} = (+1,-1,+1) \\
y^{(25)} = (+1,+1,-1) & y^{(13)} = (+1,-1,+1) & y^{(1)} = (-1,+1,+1) \\
y^{(24)} = (+1,-1,+1) & y^{(12)} = (-1,+1,+1) & y^{(0)} = (\ 1,\ \ 0,\ \ 0) \\
y^{(23)} = (+1,-1,+1) & y^{(11)} = (+1,+1,-1)
\end{array}$$

8. [23] Докажите, что (8)—оптимальный способ слияния с обратным чтением при $S = 7$ и $T = 4$ и что все методы, избегающие однопутевого слияния, хуже.
9. [M22] Докажите нижнюю оценку (9).
10. [41] При помощи ЭВМ составьте таблицу точных значений $K_T(n)$.

>11. [20] Верно ли утверждение, что для любой схемы слияния с обратным чтением, не использующей ничего, кроме $(T-1)$-путевого слияния, необходимо, чтобы отрезки на каждой ленте чередовались:

$ADAD\ldots$, т. е. она не будет работать, если два соседних отрезка окажутся одинаково упорядоченными?

12. [22] Докажите, что конструкция с прямым порядком Карпа всегда порождает помеченное дерево, удовлетворяющее условиям (a), (b), (c).
13. [16] Сделайте (12) более эффективным, удалив как можно больше однопутевых слияний, однако так, чтобы прямой порядок все еще давал правильную расстановку меток у внутренних узлов.
14. [40] Придумайте алгоритм, который выполняет слияние в прямом порядке без явного построения дерева в шагах P2 и P3, используя только $O(\log S)$ слов памяти для управления слиянием.
15. [M39] Конструкция Карпа с прямым порядком порождает деревья с однопутевым слиянием в некоторых терминальных узлах. Докажите, что если $T = 3$, то можно построить асимптотические оптимальные 3-lifo деревья, в которых используется только двухпутевое слияние

Другими словами, пусть $\hat{K}_T(n)$ будет минимальной длиной внешнего пути среди всех T-lifo деревьев с n внешними узлами, таких, что каждый внутренний узел имеет степень $T-1$. Докажите, что $\hat{K}_3(n) = n\log_2 n + O(n)$.

16. [M46] (Сохраняются обозначения упр. 15) Верно ли, что $\hat{K}_T(n) = n\log_{T-1}(n) + O(n)$ для *всех* $T \geq 3$, если $n \equiv 1 \pmod{T-1}$?

>17. [28] (Ричард Д. Пратт.) Чтобы получить возрастающий порядок в каскадном слиянии с обратным чтением, мы могли бы потребовать *четного* числа проходов слияния; это предполагает, что метод начального распределения несколько отличен от алгоритма 5.4.3C. (a) Измените (5.4.3-1) так, чтобы были представлены только точные распределения, которые требуют четного числа проходов слияния. (b) Сконструируйте схему начального распределения, осуществляющую интерполяцию между этими точными распределениями. [Иначе говоря, если число начальных отрезков попадает между точными распределениями, то желательно слить некоторые (но не все) отрезки дважды, чтобы достигнуть точного распределения.]

18. [46] Предположим, что имеется T ленточных устройств для некоторого $T \geq 3$ и что $T1$ содержит N записей, в то время как остальные ленты пусты. Возможно ли обратить порядок записей на $T1$ за число шагов, меньшее $O(N\log N)$, без обратного чтения? (Эта операция является, конечно, тривиальной, если допускается обратное чтение.) См. в упр. 5.2.5-14 класс таких алгоритмов, которые, однако, тратят порядка $N\log N$ шагов.

УПРАЖНЕНИЯ. (ВТОРАЯ ЧАСТЬ)

Следующие упражнения развивают теорию ленточного слияния с прямым чтением. В этом случае каждая лента действует как очередь, а не как стек. Схему слияния можно представить последовательностью векторов $y^{(n)}\ldots y^{(1)}y^{(0)}$ точно так же, как в тексте, но когда мы преобразуем векторное представление в представление в виде дерева, то мы заменяем правило "последним образован—первым растет" на "первым образован—первым растет". Таким образом, недопустимая конфигурация (4) должна быть заменена на

Picture: p.369

Дерево, которое может быть помечено так, чтобы изображать слияние с прямым чтением на T лентах, называется T-fifo[5] по аналогии с термином T-lifo в случае обратного чтения.

Если ленты можно прочитать в обратном направлении, они образуют очень хорошие стеки. Но, к сожалению, они не могут образовать очень хорошие универсальные очереди. Если мы в произвольном порядке записываем и читаем по правилу "первым включается—первым исключается", то приходится тратить много времени на перемотку от одной части ленты к другой. Хуже того—мы вскоре проскочим конец ленты! Мы сталкиваемся с той же проблемой, что и проблема выхода очереди за границу памяти [см. соотношения (2.2.2-4) и (2.2.2-5)], но решение в виде (2.2.2-6) и (2.2.2-7) не применимо к лентам, так как они не являются кольцевыми. Поэтому дерево будем называть *сильным* T-fifo деревом, если оно может быть помечено так, чтобы соответствующая схема слияния использовала ленты, подчиняясь особой дисциплине: "записать, перемотать, прочитать все, перемотать; записать, перемотать, прочитать все, перемотать; и т. д.".

19. [22] (Р. М. Карп.) Найдите бинарное дерево, которое не является 3-fifo.

>20. [22] Сформулируйте условие "сильного T-fifo" дерева в терминах достаточно простого правила относительно недопустимых конфигураций меток лент, аналогичного (4′).

21. [18] Нарисуйте представление в виде дерева схемы слияния с прямым чтением; определенной посредством векторов в упр. 7. Является ли это дерево сильным 3-fifo?
22. [28] (Р. М. Карп.) Докажите, что представления в виде дерева многофазного и каскадного слияния с точным распределением полностью одинаковы как в случае обратного чтения, так и в случае прямого

[5] "Fifo"—аббревиатура от "first in first out" (первым включается—первым исключается).—*Прим. перев.*

чтения, за исключением номеров, которыми помечены внутренние узлы. Найдите более широкий класс векторных, представлений схем слияния, для которых это верно.

23. [24] (Р. М. Карп.) Будем говорить, что отрезок $y^{(q)} \dots y^{(r)}$ схемы слияния является *стадией,* если никакая выводная лента не используется в дальнейшем как вводная, т. е. если не существует i, j, k, таких, что $q \geq i > k \geq r$ и $y_j^{(i)} = -1$, $y_j^{(k)} = +1$. Цель этого упражнения—доказать, что *каскадное слияние минимизирует число стадий* среди всех схем слияния с тем же числом лент и начальных отрезков.

Удобно ввести некоторые обозначения. Будем писать $v \to w$, если v и w—такие T-векторы, что существует схема слияния, которая в своей первой стадии переводит w в v (т. е. существует схема слияния $y^{(m)} \dots y^{(0)}$, такая, что $y^{(m)} \dots y^{(l+1)}$ является стадией, $w = y^{(m)} + \cdots + y^{(0)}$ и $v = y^{(l)} + \cdots + y^{(0)}$). Будем писать $v \preceq w$, если v и w—T-векторы, такие, что сумма наибольших k элементов вектора v не превышает суммы наибольших k элементов вектора w при $1 \leq k \leq T$. Так, например, $(2, 1, 2, 2, 2, 1) \preceq (1, 2, 3, 0, 3, 1)$, так как $2 \leq 3$, $2 + 2 \leq 3 + 3$, $\dots$, $2 + 2 + 2 + 2 + 1 + 1 \leq 3 + 3 + 2 + 1 + 1 + 0$. Наконец, если $v = (v_1, \dots, v_T)$, то пусть $C(v) = (s_T, s_{T-2}, s_{T-3}, \dots, s_1, 0)$, где s_k есть сумма наибольших k элементов вектора v.

(a) Докажите, что $v \to C(v)$. (b) Докажите, что $v \preceq w$ влечет $C(v) \preceq C(w)$. (c) Считая известным результат упр. 24, докажите, что каскадное слияние минимизирует число стадий.

24. [M35] Используя обозначения упр. 23, докажите, что $v \to w$ влечет $w \preceq C(v)$.

25. [M36] (Р. М. Карп.) Будем говорить, что сегмент $y^{(q)} \dots y^{(r)}$ схемы слияния является *фазой,* если ни одна из лент не используется и для ввода, и для вывода, т. е. если не существует i, j, k, таких, что $q \geq i$, $k \geq r$ и $y_j^{(i)} = +1$, $y_j^{(k)} = -1$. Цель этого упражнения—исследовать схему слияния, которая минимизирует число фаз. Мы будем писать $v \Rightarrow w$, если w может быть преобразовано в v за одну фазу (ср. с подобным обозначением, введенным в упр. 23), и пусть $D_k(v) = (s_k + t_{k+1}, s_k + t_{k+2}, \dots, s_k + t_T, 0, \dots, 0)$, где t_j обозначает j-й в порядке убывания элемент v и $s_k = t_1 + \cdots + t_k$. (a) Докажите, что $v \Rightarrow D_k(v)$ при $1 \leq k < T$. (b) Докажите, что из $v \preceq w$ следует $D_k(v) \preceq D_k(w)$ при $1 \leq k < T$. (c) Докажите, что из $v \Rightarrow w$ следует $w \preceq D_k(v)$ для некоторого k, $1 \leq k < T$. (d) Следовательно, схема слияния, сортирующая максимальное число начальных отрезков на T лентах за q фаз, может быть изображена последовательностью .целых чисел $k_1 k_2 \dots k_q$, такой, что начальное распределение есть $D_{k_q}(\dots D_{k_2}(D_{k_1}(u))\dots)$, где $u = (1, 0, \dots, 0)$. Эта стратегия минимума фаз, имеет сильное T-fifo представление, и она также входит в класс схем упр. 22. Когда $T = 3$, это *многофазная* схема, а при $T = 4$, 5, 6, 7 это вариация *сбалансированной* схемы.

26. [M46] (Р. М. Карп). Верно ли, что оптимальная последовательность $k_1 k_2 \dots k_q$, упомянутая в упр. 25, всегда равна $1\lceil T/2\rceil\lfloor T/2\rfloor\lceil T/2\rceil\lfloor T/2\rfloor \dots$ для всех $T \geq 4$ и всех достаточно больших q?

Осциллирующая сортировка

Еще один подход к сортировке слиянием был предложен Шелдоном Собелем в [*JACM,* **9** (1962), 372–375]. Вместо того чтобы начинать с прохода распределения, когда все начальные отрезки распределяются по лентам, он предложил алгоритм, который переключается то на распределение, то на слияние, так что большая часть сортировки происходит еще до того, как вся исходная информация будет полностью просмотрена. Предположим, например, что для слияния используется пять лент. По методу Собеля 16 начальных отрезков будут сортироваться следующим образом:

Операция	$T1$	$T2$	$T3$	$T4$	$T5$	Стоимость
Фаза 1. Распределение	A_1	A_1	A_1	A_1	—	4
Фаза 2. Слияние	—	—	—	—	D_4	4
Фаза 3. Распределение	—	A_1	A_1	A_1	D_4A_1	4
Фаза 4. Слияние	D_4	—	—	—	D_4	4
Фаза 5. Распределение	D_4A_1	—	A_1	A_1	D_4A_1	4
Фаза 6. Слияние	D_4	D_4	—	—	D_4	4
Фаза 7. Распределение	D_4A_1	D_4A_1	—	A_1	D_4A_1	4
Фаза 8. Слияние	D_4	D_4	D_4	—	D_4	4
Фаза 9. Слияние	—	—	—	A_{16}	—	16

Здесь, как и в п. 5.4.4, мы используем A_r и D_r для обозначения соответственно возрастающих и убывающих отрезков относительной длины r. Рассматриваемый метод начинает с записи по одному начальному отрезку на каждую из четырех лент и сливает их (читая в обратном направлении) на пятую ленту. Опять возобновляется распределение, на этот раз циклически сдвинутое на 1 вправо по отношению к лентам, и второе слияние дает еще один отрезок D_4. Когда этим способом сформированы четыре отрезка D_4, дополнительное слияние создает A_{16}. Процесс можно продолжать, создавая еще три A_{16}, сливая их в D_{64} и т. д. до тех пор, пока не исчерпаются исходные данные. Не нужно знать заранее длину исходных данных.

Если число начальных отрезков S есть 4^m, то нетрудно видеть, что этот метод обрабатывает каждую запись ровно $m + 1$ раз (один раз во время распределения и m раз во время слияния). Если S лежит

между 4^{m-1} и 4^m, то можно с помощью фиктивных отрезков увеличить S до 4^m; следовательно, общее время сортировки будет определяться $\lceil \log_4 S \rceil + 1$ проходами по всем данным. Это как раз то, что достигается при сбалансированной сортировке на *восьми* лентах; в общем случае осциллирующая сортировка с T рабочими лентами эквивалентна сбалансированному слиянию с $2(T-1)$ лентами, так как она делает $\lceil \log_{T-1} S \rceil + 1$ проходов по данным. Если S оказывается степенью $T-1$, то это самое лучшее, что можно получить при *любом* методе с T лентами, так как здесь достигается нижняя оценка из соотношения (5.4.4-9). С другой стороны, если S равно $(T-1)^{m-1}+1$, т. е. ровно на единицу больше степени $T-1$, то этот метод теряет почти целый проход.

В упр. 2 показано, как устранить часть этой лишней работы, используя специальную программу окончания. Еще одно усовершенствование было предложено в 1966 г. Денисом Л. Бэнчером, который назвал свою процедуру перекрестным слиянием. [См. H. Wedekind, Datenorganisation (Berlin W. de Gruyter, 1970). 164–166, и U. S. Patent 3540000 (10 ноября 1970).] Основная идея состоит в том, чтобы отложить слияние до тех пор, пока не будет накоплено больше сведений об S. Мы обсудим несколько измененную форму первоначальной схемы Бэнчера.

Эта улучшенная осциллирующая сортировка действует следующим образом:

	Операция	$T1$	$T2$	$T3$	$T4$	T5	Стоимость
Фаза 1.	Распределение	–	A_1	A_1	A_1	A_1	4
Фаза 2.	Распределение	–	A_1	A_1A_1	A_1A_1	A_1A_1	3
Фаза 3.	Слияние	D_4	–	A_1	A_1	A_1	4
Фаза 4.	Распределение	D_4A_1	–	A_1	A_1A_1	A_1A_1	3
Фаза 5.	Слияние	D_4	D_4	–	A_1	A_1	4
Фаза 6.	Распределение	D_4A_1	D_4A_1	–	A_1	A_1A_1	3
Фаза 7.	Слияние	D_4	D_4	D_4	–	A_1	4
Фаза 8.	Распределение	D_4A_1	D_4A_1	D_4A_1	–	A_1	3
Фаза 9.	Слияние	D_4	D_4	D_4	D_4	–	4

В этот момент мы не сливаем все D_4 в A_{16}, (если только не окажется, что исходные данные исчерпаны); лишь после того, как закончится

Фаза 15.	Слияние	D_4D_4	D_4D_4	D_4D_4	D_4	–	4

будет получен первый отрезок A_{16}:

Фаза 16.	Слияние	D_4	D_4	D_4-	A_{16}	16

Второй отрезок A_{16} появится после создания еще трех D_4:

Фаза 22.	Слияние	D_4D_4	D_4D_4	D_4	–	$A_{16}D_4$	4
Фаза 23.	Слияние	D_4	D_4	–	A_{16}	A_{16}	16

и т. д. (ср. с фазами 1–5). Преимущества схемы Бэнчера можно видеть, если имеется, например, только пять начальных отрезков: осциллирующая сортировка (ее модификация из упр. 2) выполняла бы четырехпутевое слияние (на фазе 2), за которым следовало бы двухпутевое слияние с общей стоимостью $4+4+4+1+5=14$, тогда как схема Бэнчера выполняла бы двухпутевое слияние (на фазе 3), за которым следовало бы четырехпутевое слияние с общей стоимостью $4+1+2+5=12$. (Оба метода также требуют небольших дополнительных затрат, именно однократной перемотки перед окончательным слиянием.)

Точное описание метода Бэнчера содержится ниже в алгоритме B. К сожалению, соответствующую процедуру, по-видимому, трудней понять, чем запрограммировать; легче объяснить этот метод ЭВМ, чем программисту! Частично это происходит по той причине, что рекурсивный метод выражен в итеративном виде и затем подвергнут некоторой оптимизации; читатель, возможно, обнаружит, что необходимо несколько раз проследить за работой алгоритма, чтобы действительно осознать, что же происходит.

Алгоритм B. *(Осциллирующая сортировка с перекрестным распределением.)* Этот алгоритм берет первоначальные отрезки и распределяет их но лентам, время от времени прерывая процесс распределения, чтобы слить содержимое некоторых лент.

Picture: Рис. 77. Осциллирующая сортировка с перекрестным распределением.

В алгоритме используется P-путевое слияние и предполагается, что есть $T = P + 1 \geq 3$ ленточных устройств (не считая устройства, которое может быть необходимо для хранения исходных данных). Ленточные устройства должны допускать чтение как в прямом, так и в обратном направлении; они обозначены числами 0, 1, ..., P. Используются следующие массивы:

$\mathtt{D}[j]$, $0 \leq j \leq P$—число фиктивных отрезков, наличие которых предполагается в конце ленты j.

$\mathtt{A}[l,j]$, $0 \le l \le L$, $0 \le j \le P$. Здесь L—достаточно большое число, такое, что будет введено не более P^{L+1} начальных отрезков. Если $\mathtt{A}[l,j] = k \ge 0$, то на ленте j имеется отрезок номинальной длины P^k, соответствующий "уровню l" работы алгоритма. Этот отрезок возрастающий, если k четно, и убывающий, если k нечетно. $\mathtt{A}[l,j] = -1$ означает, что на уровне l лента j не используется.

Инструкция "записать начальный отрезок на ленту j" является сокращенным обозначением следующих действий:

Установить $\mathtt{A}[l,j] \leftarrow 0$. Если исходные данные исчерпаны, то увеличить $\mathtt{D}[j]$ на 1; в противном случае записать отрезок на ленту j (в возрастающем порядке).

Инструкция "слить на ленту j" используется как краткое обозначение следующих действий:

Если $\mathtt{D}[i] > 0$ для всех $i \ne j$, то уменьшить $\mathtt{D}[i]$ на 1 при всех $i \ne j$ и увеличить $\mathtt{D}[j]$ на 1. В противном случае слить один отрезок на ленту j со всех лент $i \ne j$, таких, что $\mathtt{D}[i] = 0$, и уменьшить $\mathtt{D}[i]$ на 1 для всех остальных $i \ne j$.

B1 [Начальная установка.] Установить $\mathtt{D}[j] \leftarrow 0$ при $0 \le j \le P$. Затем записать начальный отрезок на ленту j при $1 \le j \le P$. Установить $\mathtt{A}[0,0] \leftarrow -1$, $l \leftarrow 0$, $q \leftarrow 0$.

B2 [Ввод завершен?] (В этот момент лента q пуста и всякая другая лента содержит самое большее один отрезок.) Если еще есть исходные данные, перейти к шагу **B3**. Однако если ввод исчерпан, то перемотать все ленты $j \ne q$, такие, что $\mathtt{A}[0,j]$ четно; затем слить на ленту q, читая все только что перемотанные ленты в прямом направлении, а остальные ленты—в обратном. Этим завершается сортировка; результат находится на ленте q в возрастающем порядке.

B3 [Начать новый уровень.] Установить $l \leftarrow l+1$, $r \leftarrow q$, $s \leftarrow 0$ и $q \leftarrow (q+1) \bmod T$. Записать начальный отрезок на ленту $(q+j) \bmod T$ при $1 \le j \le T-2$. (Таким образом, начальные отрезки записываются на все ленты, кроме лент q и r.) Установить $\mathtt{A}[l,q] \leftarrow -1$ и $\mathtt{A}[l,r] \leftarrow -1$.

B4 [Можно ли сливать?] Если $\mathtt{A}[l-1,q] \ne s$, вернуться к шагу **B3**.

B5 [Слияние.] (В этот момент $\mathtt{A}[l-1,q] = \mathtt{A}[l,j] = s$ при всех $j \ne q$, $j \ne r$.) Слить на ленту r. (См. выше определение этой операции.) Затем установить $s \leftarrow s+1$, $l \leftarrow l-1$, $\mathtt{A}[l,r] \leftarrow s$ и $\mathtt{A}[l,q] \leftarrow -1$. Установить $r \leftarrow (2q-r) \bmod T$. (В общем случае мы имеем $r = (q-1) \bmod T$, если s четно, и $r = (q+1) \bmod T$, если s нечетно.)

B6 [Закончен ли уровень?] Если $l = 0$, перейти к **B2**. В противном случае, если $\mathtt{A}[l,j] = s$ для всех $j \ne q$ и $j \ne r$, то перейти к **B4**. В противном случае вернуться к **B3**. ∎

Чтобы показать правильность этого алгоритма, мы можем использовать доказательство типа "рекурсивной индукции", так же как мы делали для алгоритма 2.3.1T. Предположим, что мы начинаем с шага B3 с $l = l_0$, $q = q_0$, $s_+ = \mathtt{A}[l_0, (q_0+1) \bmod T]$ и $s_- = \mathtt{A}[l_0, (q_0-1) \bmod T]$, и допустим, кроме того, что либо $s_+ = 0$, либо $s_- = 1$, либо $s_+ = 2$, либо $s_- = 3$, либо .... Можно проверить по индукции, что алгоритм в конце концов придет к шагу B5, не изменив с нулевой по l-ю строки $\mathtt{A}$ и со значениями переменных $l = l_0 + 1$, $q = q_0 \pm 1$, $r = q_0$ и $s = s_+$ или s_-, причем мы выбираем знак $+$, если $s_+ = 0$ или ($s_+ = 2$ и $s_- \ne 1$) или ($s_+ = 4$ и $s_- \ne 1, 3$) или ..., и мы выбираем знак $-$, если ($s_- = 1$ и $s_+ = 0$) или ($s_- = 3$ и $s_+ \ne 0, 2$) или .... Приведенный здесь набросок доказательства не очень элегантен, но и сам алгоритм сформулирован в виде, который больше годится для реализации, чем для проверки правильности.

Picture: Рис. 78 Эффективность осциллирующей сортировки, использующей метод алгоритма B и упр. 3.

На рис. 78 показана эффективность алгоритма B, выраженная средним числом слияний каждой записи в зависимости от S—числа начальных отрезков, причем предполагается, что начальные отрезки приблизительно равны по длине. (Соответствующие графики для многофазной и каскадной сортировки приведены на рис. 70 и 74.) При подготовке рис. 78 учтено небольшое усовершенствование, упомянутое в упр. 3.

Прямое чтение. Схема осциллирующей сортировки, по-видимому, требует возможности обратного чтения, поскольку приходится где-то накапливать длинные отрезки по мере того, как мы сливаем вновь введенные короткие отрезки. Тем не менее М. А. Готц [Proc. AFIPS Spring Jt. Comp. Conf.; **25** (1964), 599–607] нашел способ выполнить осциллирующую сортировку, используя только прямое чтение и простую перемотку. Его метод в корне отличается от остальных схем, которые мы видели в этой главе, поскольку

а) данные иногда записываются в начало ленты, причем предполагается, что данные, находящиеся в *середине* этой ленты, не разрушаются;

b) все начальные строки имеют фиксированную максимальную длину. Условие (a) нарушает свойство "первым включается—первым исключается", которое, как мы предположили, является характеристикой прямого чтения, однако оно может быть надежно реализовано, если между отрезками оставлять достаточное количество чистой ленты и если в нужные моменты пренебречь "ошибками четности". Условие (b) оказывается до некоторой степени противоречащим эффективному использованию выбора с замещением.

Осциллирующая сортировка Готца с прямым чтением имеет одно темное пятно—это один из первых алгоритмов, который был запатентован как алгоритм, а не как физическое устройство [U. S. Patent 3380029 (23 апреля 1968)]. Если положение не изменится, то это означает, что алгоритм нельзя использовать в программе без разрешения владельца патента. Метод Бэнчера (осциллирующая сортировка с обратным чтением) был запатентован IBM несколькими годами позже. [Таким образом, наступил конец эры, когда удовольствие от открытия нового алгоритма считалось достаточным вознаграждением! Так как программирование неотделимо от создания машины, а программы для ЭВМ теперь стоят денег, то патентование алгоритмов является неизбежным. Конечно, действия недальновидных людей, сохраняющих новые алгоритмы в строгом секрете, значительно хуже, чем широкая доступность алгоритмов, которые являются собственностью в течение лишь ограниченного времени.]

Центральная идея в методе Готца состоит в таком использовании лент, чтобы каждая лента начиналась с отрезка относительной длины 1, за которым следовал бы отрезок относительной длины P, затем P^2 и т. д. Например, если $T = 5$, то сортировка начинается следующим образом ("." указывает текущее положение головки чтения-записи на каждой ленте):

	Операция	$T1$	$T2$	$T3$	$T4$	$T5$	Стоимость	Примечания
Фаза 1.	Распределение	A_1	$.A_1$	$.A_1$	$.A_1$	$A_1.$	5	[T5 не перематывается]
Фаза 2.	Слияние	$A_1.$	$A_1.$	$A_1.$	$A_1.$	$A_1A_4.$	4	[Перемотка всех лет]
Фаза 3.	Распределение	A_1	$.A_1$	$.A_1$	$A_1.$	$.A_1A_4$	4	[T4 не перематывается]
Фаза 4.	Слияние	$A_1.$	$A_1.$	$A_1.$	$A_1A_4.$	$A_1.A_4$	4	[Перемотка всех лент]
Фвэа 5.	Распределение	A_1	$.A_1$	$A_1.$	$.A_1A_4$	$.A_1A_4$	4	[T3 не перематывается]
Фаза 6.	Слияние	$A_1.$	$A_1.$	$A_1A_4.$	$A_1.A_4$	$A_1.A_4$	4	[Перемотка всех лент]
Фаза 7.	Распределение	A_1	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	[T2 не перематывается]
Фаза 8.	Слияние	$A_1.$	$A_1A_4.$	$A_1.A_4$	$A_1.A_4$	$A_1.A_4$	4	[Перемотка всех лент]
фаза 9.	Распределение	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	[T1 не перематывается]
Фаза 10.	Слияние	$A_1A_4.$	$A_1.A_4$	$A_1.A_4$	$A_1.A_4$	$A_1.A_4$	4	[Нет перемотки]
Фаза 11.	Слияние	$A_1A_4A_{16}.$	$A_1A_4.$	$A_1A_4.$	$A_1A_4.$	$A_1A_4.$	16	[Перемотка всех лент]

И так далее. Во время фазы 1 лента T1 перематывается и одновременно на T2 записываются исходные данные, затем перематывается T2 и одновременно на T3 записываются исходные данные и т. д. В конце концов, когда исходные данные исчерпаны, начинают появляться фиктивные отрезки, и иногда необходимо вообразить, что они записаны явно на ленте полной длины. Например, если $S = 18$, то отрезки A_1 на T4 и T5 будут фиктивными во время фазы 9; нам придется продвинуться вперед по T4 и T5 при слиянии с T2 и T3 на T1 во время фазы 10, так как нам надо добраться до отрезков A_4 на T4 и T5 для подготовки к фазе 11. С другой стороны, фиктивный отрезок A_1 на T1 не обязательно должен существовать явно. Таким образом, "конец игры" несколько замысловат. Еще с одним примером применения этого метода мы встретимся в следующем пункте.

Упражнения

1. [22] В тексте имеется иллюстрация осциллирующей сортировки Собеля в ее первозданном виде для $T =$ 5 и $S = 16$. Дайте точное определение алгоритма, в котором эта процедура обобщается и сортируются $S = P^L$ начальных отрезков на $T = P + 1 \geq 3$ лентах. Постарайтесь найти алгоритм, который может быть очень просто описан.
2. [24] Если в изначальном методе Собеля $S = 6$, то мы могли бы заявить, что $S = 16$ и что имеется 10 фиктивных отрезков. Тогда фаза 3 в примере в тексте поместила бы фиктивные отрезки A_0 на T4 и T5; фаза 4 слила бы отрезки A_1 на T2 и T3 в D_2 на T1; фазы 5–8 не делали бы ничего; фаза 9 породила бы A_6 на T4. Лучше бы перемотать T2 и T3 сразу после фазы 3 и затем немедленно получать A_6 на T4 с помощью трехпутевого слияния.

   Покажите, как, основываясь на этой идее, улучшить окончание алгоритма из упр. 1, если S не является точной степенью P.

>3. [24] Составьте таблицу, показывающую поведение алгоритма B, если $T = 3$, предполагая, что имеется 9 начальных отрезков. Покажите, что эта процедура очевидно неэффективна в одном месте, и предложите изменения в алгоритме B, которые исправляют положение.
4. [21] На шаге B3 имеется установка как $\texttt{A}[l, q]$, так и $\texttt{A}[l, r]$ в -1. Покажите, что одна из этих операций всегда лишняя, так как соответствующий элемент массива A никогда не рассматривается.

5. [M25] Пусть S—число начальных отрезков в имеющихся исходных данных для алгоритма B. При каких значениях S не требуется *ни одной перемотки* на шаге B2?