ГЛАВА 5

СОРТИРОВКА

Нет дела, коего устройство было бы труднее,
ведение опаснее, а успех сомнительнее,
нежели замена старых порядков новыми.

— НИККОЛО МАКИАВЕЛЛИ, *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, Из *The Case of the Angry Mourner* (1951)

Компьютер "Treesort": при новом "компьютерном подходе"
к изучению природы вы получите возможность быстро распознавать
более 260 видов деревьев США, Аляски и Канады, включая пальмы,
растительность 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 порядке возрастания или убывания. 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 компьютера, во многом зависит скорость выполнения и простота алгоритмов, предназначенных для их обработки.

Хотя в словарях слово "сортировка" (sorting) определяется как процесс разделения объектов по виду или сорту, программисты традиционно используют его в гораздо более узком смысле, обозначая им такую перестановку предметов, при которой они располагаются в порядке возрастания или убывания. Такой 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 из имеющихся у нас лентопротяжных механизмов 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". 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" (sequencing), 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, но оно прочно вошло в программистский жаргон. Поэтому мы без дальнейших извинений будем использовать слово "сортировка" в узком смысле: "размещение по порядку". Вот некоторые из наиболее важных областей применения сортировки.

a) *Решение задачи группирования*, когда нужно собрать вместе все элементы с одинаковыми значениями некоторого признака. Допустим, имеется 10 000 элементов, расположенных в случайном порядке, причем значения многих из них равны и нужно переупорядочить массив так, чтобы элементы с равными значениями занимали соседние позиции в массиве. Это, по существу, тоже задача "сортировки", но в более широком смысле, и она легко может быть решена путем сортировки массива в указанном выше узком смысле, а именно — в результате расположения элементов в порядке неубывания $v_1 \le v_2 \le \cdots \le v_{10000}$. Эффект, который может быть достигнут после выполнения этой процедуры, и объясняет изменение первоначального смысла слова "сортировка".

b) *Поиск общих элементов в двух или более массивах.* Если два или более массивов рассортировать в одном и том же порядке, то можно отыскать в них все общие элементы за один последовательный просмотр всех массивов без возвратов. Именно этим принципом и воспользовался Перри Мейсон, чтобы раскрыть дело об убийстве (см. эпиграфы к этой главе). Оказывается, что, как правило, гораздо удобнее 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 массивам в качестве приемлемой замены прямой адресации.

c) *Поиск информации по значениям ключей.* Как мы узнаем из главы 6, сортировка используется и при поиске; с ее помощью можно сделать результаты обработки данных более удобными для восприятия человеком. В самом деле, подготовленный компьютером список, рассортированный в алфавитном порядке, зачастую выглядит весьма внушительно, даже если содержащиеся в нем числовые данные были рассчитаны неверно.

Хотя сортировка традиционно и большей частью использовалась для обработки коммерческих данных, в действительности она является инструментом, полезным в самых разных ситуациях, и поэтому о ней 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 мощных программных систем, продемонстрировавших богатые возможности сортировки, был компилятор LARC Scientific Compiler, разработанный Дж. Эрдвином (J. Erdwinn) и Д. Е. Фергюсоном (D. E. Ferguson) в фирме Computer Sciences Corporation в 1960 году. В этом оптимизирующем компиляторе для расширенного языка FORTRAN сортировка использовалась весьма интенсивно, так что различные алгоритмы компиляции работали с относящимися к ним фрагментами исходного текста программы, расположенными в удобной последова-

тельности. На первом проходе 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 фрагменты текста, в которых использовался данный идентификатор, оказывались собранными вместе. "Определяющие вхождения", специфицирующие идентификатор как имя функции, простую (параметр) или многомерную (массив) переменную, получали меньшие номера, поэтому они оказывались первыми в последовательности лексем, которые соответствовали этому идентификатору. Тем самым облегчалась проверка правильности употребления идентификаторов, распределение памяти с учетом деклараций `EQUIVALENCE` 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 упрощения оптимизации циклов, включения в листинг сообщений об ошибках и т. д. Короче говоря, компилятор был спроектирован так, что всю обработку файлов (а для их хранения использовались весьма распространенные в те времена устройства внешней памяти на магнитных барабанах) можно было выполнять последовательно. Поэтому-то данные и снабжались такими порядковыми номерами, которые позволяли упорядочивать эти данные различными удобными способами.

По оценкам производителей компьютеров в 60-х годах в среднем более четверти машинного времени тратилось на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Исходя из этих статистических данных 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 приведенных 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! Появление изощренных алгоритмов сортировки говорит 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 программирования вообще. Мы познакомимся со многими важными принципами манипулирования структурами данных и проследим за эволюцией различных методов сортировки, причем читателю часто будет предоставлена возможность самостоятельно "изобрести велосипед", как будто бы до него никто с подобными задачами не сталкивался. Обобщение этих частных методов позволит нам в значительной степени овладеть теми подходами, которые помогут

создавать качественные алгоритмы для решения других проблем, связанных с использованием компьютеров.

Методы сортировки служат великолепной иллюстрацией базовых концепций *анализа алгоритмов*, т. е. оценки качества алгоритмов, что, в свою очередь, позволяет разумно делать выбор среди, казалось бы, равноценных методов. Читатели, имеющие склонность к математике, найдут в этой главе немало поучительных 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 сформулировать 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 также *совершенным упорядочением*. Любое множество с отношением "$<$", удовлетворяющим обоим этим свойствам, поддается сортировке большинством методов, описанных в этой главе, хотя некоторые из методов годятся только для числовых и буквенных ключей с общепринятым отношением порядка.

Задача сортировки — найти такую перестановку записей $p(1)p(2)\ldots p(N)$ с индексами $\{1, 2, \ldots, N\}$, после которой ключи расположились бы в порядке неубывания:

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

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

$$p(i) < p(j) \qquad \text{для любых} \quad K_{p(i)} = K_{p(j)} \quad \text{и} \quad i < j. \tag{2}$$

В одних случаях придется 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:

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

Эти величины используются в качестве искусственных ключей, а также граничных признаков. 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 методы сортировки подразделяют на два класса: внутренние, когда все записи хранятся в быстрой оперативной памяти, и внешние, когда все записи в ней не помещаются. Методы внутренней сортировки обеспечивают большую гибкость при построении структур данных и доступа к ним, внешние же методы обеспечивают достижение нужного результата в "спартанских" условиях ограниченных ресурсов.

Достаточно хороший общий алгоритм затрачивает на сортировку N записей время пропорционально $N \log N$; при этом требуется около $\log N$ "проходов" по данным. Как мы увидим в разделе 5.3.1, это минимальное время, если записи расположены в произвольном порядке и сортировка выполняется попарным сравнением ключей. Так, если удвоить число записей, то и время при прочих равных условиях возрастет немногим более чем вдвое. (На самом деле, когда N неограниченно возрастает, время сортировки растет как $N(\log N)^2$, если все ключи различны, поскольку и размеры ключей увеличиваются, как минимум, пропорционально $\log N$ с ростом N; но практически N всегда остается ограниченным.)

С другой стороны, если известно, что ключи являются случайными величинами с некоторым непрерывным распределением, то, как мы увидим ниже, сортировка может быть выполнена в среднем за $O(N)$ шагов.

УПРАЖНЕНИЯ (часть 1)

1. [*M20*] Докажите, что из законов трихотомии и транзитивности вытекает *единственность* перестановки $p(1)\,p(2)\ldots p(N)$, если сортировка устойчива.

2. [*21*] Пусть каждая запись R_j 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):

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

Алиса рассортировала 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 она рассортировала каждую из n групп $R_{p(i_{j-1}+1)}, \ldots, R_{p(i_j)}$ по собственным малым ключам.

Билл взял тот же исходный файл и сначала рассортировал его по малым ключам, а затем получившийся файл рассортировал по большим ключам.

Взяв тот же исходный файл, Крис рассортировал его один раз в лексикографическом порядке, пользуясь 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. сортировка, при которой выполняются условия (1) и (2). (На самом деле существует, по крайней мере, три расположения записей, удовлетворяющих этим условиям!)

▸ **4.** [*21*] Составители словарей фактически не следуют жестко лексикографическому порядку, так как прописные и строчные буквы у них перемежаются. В частности, используется такой порядок:

$$\text{a} < \text{A|aa} < \text{AA} < \text{AAA} < \text{Aachen} < \text{aah} < \cdots < \text{zzz} < \text{ZZZ}.$$

Поясните, как реализовать подобный словарный порядок сортировки.

▸ **5.** [*M28*] Разработайте двоичный код для всех неотрицательных целых чисел, такой, что, если n закодировано в виде строки $\rho(n)$, то $m < n$ тогда и только тогда, когда $\rho(m)$ меньше в лексикографическом смысле, чем $\rho(n)$. Более того, $\rho(m)$ не должно быть префиксом $\rho(n)$ для любого $m \neq n$. По возможности длина $\rho(n)$ должна быть по крайней мере $\lg n + O(\log\log n)$ для всех больших n. (Такой способ кодирования очень удобен, если приходится сортировать текст, состоящий из чисел и слов, или если желательно отобразить довольно большие текстовые последовательности на двоичные коды.)

6. [*15*] Программисту на MIX Б. С. Даллу потребовалось выяснить, находится ли в ячейке A число, большее числа из ячейки B, меньшее или же равное ему. Он написал в программе "LDA A: SUB B", а потом проверил, какой результат получился в регистре A: положительный, отрицательный или нуль. Какую серьезную ошибку он допустил и как должен был поступить?

7. [*17*] Напишите подпрограмму для компьютера MIX для сравнения ключей с увеличенной точностью, исходя из следующих условий.

Вызов:	JMP COMPARE
Состояние при входе:	rI1 $= n$; CONTENTS(A + k) $= a_k$ и CONTENTS(B + k) $= b_k$ для $1 \le k \le n$; полагается, что $n \ge 1$.
Состояние при выходе:	CI = GREATER, если $(a_n, \ldots, a_1) > (b_n, \ldots, b_1)$; CI = EQUAL, если $(a_n, \ldots, a_1) = (b_n, \ldots, b_1)$; CI = LESS, 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 $a_k = b_k$ для $n \ge k > j$, но $a_j < b_j$.

▸ **8.** [*30*] В ячейках A и B содержатся соответственно числа a и b. Покажите, что можно написать программу для MIX, которая вычисляла бы $\min(a, b)$ и записывала результат в ячейку C, не пользуясь командами перехода. (*Предостережение.* Поскольку арифметическое переполнение невозможно обнаружить без операторов перехода, разумно так построить программу, чтобы переполнение не могло возникнуть ни при каких значениях a и b.)

9. [*M27*] Какова вероятность того, что после сортировки в порядке неубывания n независимых равномерно распределенных на отрезке [0, 1] случайных величин r-е от начала число окажется $\le x$?

УПРАЖНЕНИЯ (часть 2)

В каждом из этих упражнений поставлена задача, с которой может столкнуться программист. Предложите хорошее решение задачи, *предполагая, что вы имеете дело с "музейным" компьютером, у которого имеется сравнительно небольшая оперативная память, порядка нескольких тысяч слов* и около полудюжины накопителей на магнитной ленте (НМЛ) — этого количества вполне достаточно для выполнения сортировки. Алгоритм, который сможет эффективно работать в таких "спартанских" условиях, тем более будет прекрасно выполнять задачу на современных компьютерах.

10. [*15*] Имеется лента, на которой записан миллион слов данных. Как определить, сколько на этой ленте различных слов?

11. [*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 деклараций (карточек) о доходах от налогоплательщиков. Как бы вы стали отыскивать людей, которые сообщили не обо всех своих доходах?

12. [*M25*] (*Транспонирование матрицы.*) Имеется магнитная лента, содержащая миллион слов, которые представляют собой элементы матрицы 1000×1000, 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: $a_{1,1}\, a_{2,1} \ldots a_{1000,1}\, a_{1,2} \ldots a_{1000,2} \ldots a_{1000,1000}$. (Постарайтесь сделать не более десяти просмотров данных.)

13. [*M26*] В вашем распоряжении есть довольно большой файл из N слов. Как бы вы его "перетасовали" случайным образом?

14. [*20*] Вы работаете с двумя вычислительными системами, в которых по-разному упорядочены литеры (буквы и цифры). Как заставить первую систему сортировать файлы с буквенно-цифровой информацией, предназначенные для использования во второй системе?

15. [*18*] Имеется довольно большой список людей, родившихся в США, с указанием штата, в котором они родились. Как подсчитать тех, кто родился в каждом штате? (Предполагается, что ни один человек не указан в списке более одного раза.)

16. [*20*] Чтобы облегчить внесение изменений в большие программы, написанные на языке FORTRAN, желательно написать программу, которая формирует таблицу *перекрестных ссылок*. Входными данными для нее служит программа на FORTRAN, а в результате получается листинг исходной программы, снабженный указателем всех случаев употребления каждого идентификатора (т. е. имени) в программе. Как написать такую программу?

▸ **17.** [*33*] (*Сортировка библиографических карточек каталога.*) До появления компьютерных баз данных в каждой библиотеке существовал каталог библиографических карточек, с помощью которого читатели могли отыскать интересующие их издания. Но сортировка карточек в порядке, удобном для читателей, усложняется по мере увеличения библиотечного фонда. 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	Achtzehnhundertzwölf (числительное 1812 на немецком языке)
Bibliothèque d'histoire révolutionnaire	В французском тексте апостроф рассматривается как пробел
Bibliothèque des curiosités	Диатонические 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
Brown, John, of Boston	по 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

Текст карточки	*Замечания*
Brown, Dr. John, 1810–1882	Указание положения (Dr.) игнорируется
Brown-Williams, Reginald Makepeace	Дефис рассматривается как пробел
Brown America	Названия книг указываются 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 langen Tag	..., если существительное стоит в именительном падеже
Dix, Morgan, 1827–1908	Фамилии идут раньше других слов
1812 ouverture	Dix-huit cent douze (числительное 1812 на французском языке)
Le XIXe siècle français	Dix-neuvième (числительное XIX-й на французском языке)
The 1847 issue of U. S. stamps	Eighteen forty-seven (числительное 1847 на английском языке)
1812 overture	Eighteen twelve (числительное 1812 на английском языке)
I am a mathematician	(a book by Norbert Wiener) (книга Норберта Винера)
IBM journal of research and development	Аббревиатуры рассматриваются как ряд однобуквенных слов
ha-I ha-ehad	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 словом "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." заменяется словом "Mistress"
Mistress of mistresses	
Royal society of London	В названиях британских учреждений слово "royalty" (королевский) учитывается
St. Petersburger Zeitung	"St." заменяется словом "Saint", даже в немецком тексте
Saint-Saëns, Camille, 1835–1921	Дефис рассматривается как пробел
Ste-Marie, Gaston P	Sainte
Seminumerical algorithms	(a book by Donald Ervin Knuth)
Uncle Tom's cabin	(a book by Harriet Beecher Stowe)
U. S. bureau of the census	"U. S." заменяется словами "United States"

Текст карточки	*Замечания*
Vandermonde, Alexandre Théophile, 1735–1796	
Van Valkenburg, Mac Elwyn, 1921–	Пробел после префикса в фамилиях игнорируется
Von Neumann, John, 1903–1957	
The whole art of legerdemain	Артикль в начале текста игнорируется
Who's afraid of Virginia Woolf?	Апостроф в текстах на английском языке игнорируется
Wijngaarden, Adriaan van, 1916–	Фамилия никогда не начинается со строчной литеры

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

Предположим, вам поручено рассортировать большое количество таких карточек с помощью компьютера, а затем обслуживать огромную картотеку, причем у вас нет возможности изменить уже сложившийся порядок заполнения карточек. Как бы вы организовали информацию, чтобы упростить операции сортировки и включения новых карточек?

18. [*M25*] (Э. Т. Паркер (E. T. Parker).) Леонард Эйлер высказал предположение [*Nova Acta Acad. Sci. Petropolitanæ* **13** (1795), 45–63, §3; написана в 1778], что уравнение

$$u^6 + v^6 + w^6 + x^6 + y^6 = z^6$$

не имеет нетривиальных решений среди целых неотрицательных чисел u, v, w, x, y, z. Одновременно он предположил, что уравнение

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

не имеет решения в положительных целых числах при $n \geq 3$, но это предположение было опровергнуто уже в наше время, когда с помощью компьютера было найдено тождество $27^5 + 84^5 + 110^5 + 133^5 = 144^5$ [см. L. J. Lander, T. R. Parkin, and J. L. Selfridge, *Math. Comp.* **21** (1967), 446–459]. Множество аналогичных примеров было обнаружено и при $n = 4$ Н. Д. Элкисом (N. D. Elkies) [*Math. Comp.* **51** (1988), 825–835]. Подумайте, как можно было бы использовать сортировку для поиска примеров, опровергающих предположение Эйлера при $n = 6$.

▸ **19.** [*24*] Файл содержит большое количество 30-разрядных двоичных слов: $x_1, \ldots, x_N$. Придумайте хороший способ нахождения в нем всех *пар с взаимно дополняющими элементами* $\{x_i, x_j\}$. (Два слова называются взаимно дополняющими, если второе содержит нули во всех разрядах, в которых были единицы в первом слове, и наоборот; таким образом, они взаимно дополняющие тогда и только тогда, когда их сумма равна $(11\ldots1)_2$, если они рассматриваются как двоичные числа.)

▸ **20.** [*25*] Имеется файл, содержащий тысячу 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?

21. [*22*] Как бы вы поступили, если бы вам нужно было найти все пятибуквенные анаграммы, такие как **CARET**, **CARTE**, **CATER**, **CRATE**, **REACT**, **RECTA**, **TRACE**; **CRUEL**, **LUCRE**, **ULCER**; **DOWRY**, **ROWDY**, **WORDY**? [Или если бы вы, допустим, захотели узнать, существуют ли в английском языке наборы из десяти или более анаграмм, кроме замечательной серии

APERS, **ASPER**, **PARES**, **PARSE**, **PEARS**, **PRASE**, **PRESA**, **RAPES**, **REAPS**, **SPAER**, **SPARE**, **SPEAR**,

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

22. [*M28*] Пусть даны описания весьма большого числа ориентированных графов. Как можно сгруппировать *изоморфные* графы? (Два ориентированных 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.)

23. [*30*] В некоторой группе из 4 096 человек каждый имеет около 100 знакомых. Был составлен список всех пар людей, знакомых друг с другом. (Это отношение симметрично, т. е. если x знаком с y, то и y знаком с x. Поэтому в списке содержится примерно 200 000 пар.) Придумайте алгоритм, который по заданному k выдавал бы все *клики* из k человек. (*Клика* — это группа людей, в которой все знают друг друга.) Предполагается, что не бывает слишком больших клик (размером более 25).

▸ **24.** [*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 крайней западной точке цепи, не понял, что ему делать, и выбросил свой листок. Остальные 2 999 999 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 сведений для восстановления списка людей в исходном порядке. Программист нашел способ сделать это менее чем за 1 000 просмотров лент с данными, используя лишь последовательный доступ к файлам на лентах и небольшой объем оперативной памяти. Как ему это удалось?

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

25. [*M21*] (*Дискретные логарифмы.*) Пусть известно, что p — простое число (довольно большое), а a — первообразный корень по модулю p. Следовательно, для любого b в диапазоне $1 \le b < p$ существует единственное n, такое, что $a^n \bmod p = b$, $1 \le n < p$. (Это n называется индексом b по модулю p по отношению к a.) Как по заданному b найти n менее чем за $\Omega(n)$ шагов. [*Указание.* Пусть $m = \lceil \sqrt{p} \rceil$. Попытайтесь решить уравнение $a^{mn_1} \equiv ba^{-n_2}$ (по модулю p) при $0 \le n_1, n_2 < m$.]