МОСКОВСКИЕ ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ

Под редакцией Е. В. Андреевой, В. М. Гуровица
и В. А. Матюхина

Москва
Издательство МЦНМО
2006

УДК 519.671
ББК 22.18
М82

Издание осуществлено при поддержке Департамента образования г. Москвы и Московского института открытого образования

Московские олимпиады по информатике / Под ред. Е. В. Андреевой, В. М. Гуровица и В. А. Матюхина — М.: МЦНМО, 2006.— 256 с.: ил.

ISBN 5-94057-233-2

Книга предназначена для школьников, учителей информатики, студентов и просто любителей решать задачи по программированию. В ней приведены задачи Московских олимпиад по информатике (командных, заочных и личных туров) последних лет. Большинство задач приведены с подробными разборами и комментариями. Ко всем задачам прилагаются тесты для автоматической проверки их решений, которые можно найти на сайте `www.olympiads.ru/books`. Книга снабжена тематическим рубрикатором, в котором задачи упорядочены по темам и сложности. В качестве дополнительного материала читатель найдет в книге статьи о поиске в глубину и методе рекурсивного спуска, а также о том, зачем и как можно учить школьников программированию.

УДК 519.671, ББК 22.18

Авторы задач и текстов решений: Е. В. Андреева, В. Ю. Антонов, М. А. Бабенко, К. А. Батузов, Б. О. Василевский, В. М. Гуровиц, Ю. Е. Егоров, Р. А. Жуйков, Д. Н. Королёв, А. П. Лахно, Я. А. Леонов, А. А. Лунёв, В. А. Матюхин, П. И. Митричев, А. А. Петров, А. О. Тимофеев, М. О. Трухина, А. В. Фонарев, Е. А. Шавлюгин, С. В. Шедов, А. Ю. Юрьев.

ISBN 5-94057-233-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Олимпиады по информатике в последнее время пользуются все большей и большей популярностью. По сложившейся традиции на олимпиадах участникам предлагаются задачи по программированию, однако для успешного выступления школьник должен не только владеть языком программирования, но и уметь придумывать и реализовывать алгоритмы решения задач, оценивать время их работы, тестировать и отлаживать свои программы.

Московская олимпиада по информатике — это, на самом деле, не одна, а цикл олимпиад.

Традиционно в ноябре проводится **командная олимпиада** по информатике. В ней участвуют команды, состоящие из 3 человек. Команде выдается комплект задач и один компьютер. Как распределяются роли в команде — это проблема самой команды (или же участники делят между собой задачи, или же один из них придумывает алгоритмы, другой — пишет программы, а третий находит в них ошибки). Решения проверяются непосредственно во время олимпиады, т. е. команда, считая, что она решила задачу, посылает решение на проверку и уже через несколько минут получает ответ о правильности или неправильности решения. При подведении итогов в первую очередь учитывается, сколько задач удалось решить команде, а при равном числе задач более высокое место получает команда, которая потратила на решение задач меньше времени и сделала меньше неверных попыток. Команды-победительницы получают право участия во Всероссийской командной олимпиаде по программированию, которая проводится в конце ноября в г. Санкт-Петербурге.

Личная олимпиада состоит из двух этапов. Первый этап — заочный тур. Он проводится с октября по январь на сайте `www.olympiads.ru/moscow`. На сайте выкладываются условия задач, а также предоставляется возможность для сдачи решений на проверку. Решения так же, как и на командной олимпиаде, проверяются автоматической системой проверки на заранее подготовленных жюри олимпиады тестах, и потому результат проверки сообщается участнику буквально через несколько минут после отправки решения. Тестирующая система работает круглосуточно — в будни и выходные, днем и ночью. Таким образом, в течение трех месяцев, пока идет заочный тур, каждый может выбирать себе удобное время для решения задач.

Кроме того, на заочном туре жюри старается предложить достаточно большой набор из задач самых разных типов и самого разного уровня сложности, чтобы участие в туре было доступно и интересно как начинающим программистам, так и школьникам, уже достигшим определенных высот в программировании. Более того, продолжительность тура дает возможность школьникам использовать литературу, узнавать новые алгоритмы и применять их для решения задач, т. е. делать новые шаги в изучении информатики.

Заочный тур пользуется очень большой популярностью — в нем участвуют не только московские школьники, но и школьники (и даже студенты) практически всей России и ближнего зарубежья. Если в 2002/03 учебном году, когда заочный тур проводился впервые, в нем приняло участие чуть более 150 человек, то в 2005/06 — более 650.

Московские школьники, ставшие победителями заочного тура, приглашаются на очную личную олимпиаду (причем критерии приглашения для школьников разных классов — разные).

Личная олимпиада проводится в два тура, результаты которых суммируются (в 2005/06 учебном году проводится эксперимент, в рамках которого состоится отдельная олимпиада в один тур для школьников 7–9 классов).

На личной олимпиаде правила несколько иные. Здесь решения сдаются на проверку в конце тура. Тем самым, от участников требуется еще и умение тщательно тестировать свои решения.

Победители личной олимпиады приглашаются на сборы, по результатам которых формируется команда Москвы для участия во Всероссийской олимпиаде по информатике.

Официальный сайт Московской олимпиады по информатике:

`www.olympiads.ru/moscow`

Московская олимпиада по информатике проводится Департаментом образования г. Москвы, Московским центром непрерывного математического образования, Московским государственным университетом им. М. В. Ломоносова и Московским институтом открытого образования при поддержке компьютерного супермаркета «Никс».

В последние годы оргкомитет олимпиады возглавляли проф. А. Л. Семенов, член-корр. РАН В. П. Иванников, проф. В. Б. Алексеев, жюри олимпиады работало под руководством Е. В. Андреевой. Тестирующая система для проверки решений была разработана и поддерживалась А. В. Черновым. В подготовке и проведении олимпиады активное участие принимали В. Ю. Антонов, В. Д. Арнольд, М. А. Бабенко, Е. Я. Барский, К. А. Батузов, В. В. Болдырев, Б. О. Василевский, В. М. Гуровиц, Р. А. Жуйков, Д. П. Кириенко, Д. Н. Королев, А. П. Лахно, Я. А. Леонов, В. В. Малышко, А. В. Мамонтов, В. А. Матюхин, П. И. Митричев, В. Ю. Рубаев, А. А. Петров, А. О. Тимофеев, М. О. Трухина, Е. А. Шавлюгин, С. В. Шедов, А. А. Шестимеров, И. В. Ященко и многие другие.

В эту книгу вошли задачи Московских олимпиад последних лет. Ко всем задачам, кроме командного и заочного туров 2003/04, написаны разборы задач. Кроме того, со странички `www.olympiads.ru/books` можно скачать тесты и проверяющие программы ко всем задачам, а также условия задач в электронном виде. В конце книги также приводится список задач, вошедших в книгу, упорядоченный по тематике. Этот список может быть полезен учителям при подготовке занятий для подбора задач на определенную тему, а также школьникам, самостоятельно занимающимся изучением программирования.

Кроме того, в книгу включены три статьи. Первая из них посвящена поиску в глубину и его применениям (возможно, что читатели, знакомые с поиском в глубину, откроют для себя новые задачи, которые с его помощью можно решать). Вторая посвящена методу рекурсивного спуска и его применению для решения задачи о вычислении значения арифметического выражения, задаваемого пользователем. Третья статья в этом ряду несколько необычна — она адресована в первую очередь учителям: в ней председатель методической комиссии Московской олимпиады по информатике, учитель информатики Московской гимназии на Юго-Западе №1543 В. А. Матюхин делится своими мыслями о том, зачем вообще нужно изучать программирование в средней школе и как можно использовать на уроках систему автоматической проверки решений, аналогичную той, которая используется для оценки решений на олимпиадах по информатике.

УСЛОВИЯ ЗАДАЧ

Олимпиада I. Заочный тур 2002–2003 учебного года

Задача I-A. Таймер

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	8 мегабайт

Таймер — это часы, которые умеют подавать звуковой сигнал по прошествии некоторого периода времени. Напишите программу, которая определяет, когда должен быть подан звуковой сигнал.

Формат входных данных.

В первой строке входного файла записано текущее время в формате ЧЧ:ММ:СС (с ведущими нулями). При этом оно удовлетворяет ограничениям: ЧЧ — от 00 до 23, ММ и СС — от 00 до 59.

Во второй строке записан интервал времени, который должен быть измерен. Интервал записывается в формате Ч:М:С (где Ч, М и С — от 0 до 10^9, без ведущих нулей). Дополнительно, если Ч=0 (или Ч=0 и М=0), то они могут быть опущены. Например, `100:60` на самом деле означает 100 минут 60 секунд, что то же самое, что `101:0` или `1:41:0`. А `42` обозначает `42` секунды. `100:100:100` — 100 часов, 100 минут, 100 секунд, что то же самое, что `101:41:40`.

Формат выходных данных.

В выходной файл выведите в формате ЧЧ:ММ:СС время, когда прозвучит звуковой сигнал. При этом, если сигнал прозвучит не в текущие сутки, то дальше должна следовать запись `+<кол-во> days`. Например, если сигнал прозвучит на следующий день, то `+1 days`.

Примеры.

a.in	a.out
01:01:01 48:0:0	01:01:01+2 days

a.in	a.out
01:01:01 58:119	02:01:00
23:59:59 1	00:00:00+1 days

Задача I-B. Домой на электричках

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Одна из команд-участниц олимпиады решила вернуться домой на электричках. При этом ребята хотят попасть домой как можно раньше. К сожалению, не все электрички идут от города, где проводится олимпиада, до станции, на которой живут ребята. И, что еще более обидно, не все электрички, которые идут мимо их станции, останавливаются на ней (равно как вообще, электрички останавливаются далеко не на всех станциях, мимо которых они идут).

Все станции на линии пронумерованы числами от 1 до N. При этом станция номер 1 находится в городе, где проводится олимпиада, и в момент времени 0 ребята приходят на станцию. Станция, на которую нужно попасть ребятам, имеет номер E.

Напишите программу, которая по данному расписанию движения электричек вычисляет минимальное время, когда ребята могут оказаться дома.

Формат входных данных.

Во входном файле записаны сначала числа: N ($2 \leqslant N \leqslant 100$) и E ($2 \leqslant E \leqslant N$). Затем записано число M ($0 \leqslant M \leqslant 100$), обозначающее количество рейсов электричек. Далее идет описание M рейсов электричек. Описание каждого рейса электрички начинается с числа K_i ($2 \leqslant K_i \leqslant N$) — количества станций, на которых она останавливается, а далее следует K_i пар чисел, первое число каждой пары задает номер станции, второе — время, когда электричка останавливается на этой станции (время выражается целым числом из диапазона от 0 до 10^9). Станции внутри одного рейса упорядочены в порядке возрастания времени. В течение одного рейса электричка все время движется в одном направлении — либо от города, либо к городу.

Формат выходных данных.

В выходной файл выведите одно число — минимальное время, когда ребята смогут оказаться на своей станции. Если существующими рейсами электричек они добраться не смогут, выведите -1.

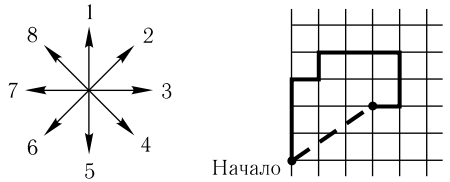
Пример.

b.in	b.out
5 3 4 2 1 5 2 10 2 2 10 4 15 4 5 0 4 17 3 20 2 35 3 1 2 3 40 4 45	20

Задача I-C. Клад

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	32 мегабайта

Найти закопанный пиратами клад просто: все, что для этого нужно, это карта. Как известно, пираты обычно рисуют карты от руки и описывают алгоритм нахождения клада так: «Встаньте около одинокой пальмы. Пройдите тридцать шагов в сторону леса, потом семнадцать шагов в сторону озера, …, наконец десять шагов в сторону большого булыжника. Клад находится под ним». Бóльшая часть таких указаний просто сводится к прохождению какого-то количества шагов в одном из восьми направлений: 1 — север, 2 — северо-восток, 3 — восток, 4 — юго-восток, 5 — юг, 6 — юго-запад, 7 — запад, 8 — северо-запад (см. рис.). Длина шага в любом направлении равна 1.



Путешествие по такому пути обычно является прекрасным способом посмотреть окрестности, однако в наше время постоянной спешки ни у кого нет времени на это. Поэтому кладоискатели хотят идти напрямую в точку,

где зарыт клад. Например, вместо того чтобы проходить три шага на север, один шаг на восток, один шаг на север, три шага на восток, два шага на юг и один шаг на запад, можно пройти напрямую примерно 3,6 шага (см. рис.).

Вам необходимо написать программу, которая по указаниям пиратов определяет точку, где зарыт клад.

Формат входных данных.

Первая строка входного файла содержит число N — число указаний ($1 \leqslant N \leqslant 40$). Последующие N строк содержат сами указания — номер направления (целое число от 1 до 8) и количество шагов (целое число от 1 до 1000). Числа разделены пробелами.

Формат выходных данных.

В выходной файл выведите координаты X и Y точки (два вещественных числа, разделенные пробелом), где зарыт клад, считая, что ось Ox направлена на восток, а ось Oy — на север. Изначально кладоискатель должен стоять в начале координат. Координаты необходимо вывести с погрешностью не более 10^{-3}.

Примеры.

c.in	c.out
6 1 3 3 1 1 1 3 3 5 2 7 1	3.000 2.000
1 8 10	-7.071 7.071

Задача I-D. Забавная игра

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Легендарный учитель математики Юрий Петрович придумал забавную игру с числами. А именно, взяв произвольное целое число, он переводит его в двоичную систему счисления, получая некоторую последовательность из нулей и единиц, начинающуюся с единицы. (Например, десятичное число

$19_{10} = 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$ в двоичной системе запишется как 10011_2.) Затем учитель начинает сдвигать цифры полученного двоичного числа по циклу (так, что последняя цифра становится первой, а все остальные сдвигаются на одну позицию вправо), выписывая образующиеся при этом последовательности из нулей и единиц в столбик. Он подметил, что независимо от выбора исходного числа получающиеся последовательности начинают с некоторого момента повторяться. И наконец, Юрий Петрович отыскивает максимальное из выписанных чисел и переводит его обратно в десятичную систему счисления, считая это число результатом проделанных манипуляций. Так, для числа 19 список последовательностей будет таким:

10011
11001
11100
01110
00111
10011
…

и результатом игры, следовательно, окажется число $1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 28$.

Поскольку придуманная игра с числами все больше занимает воображение учителя, отвлекая тем самым его от работы с ну очень одаренными школьниками, Вас просят написать программу, которая бы помогла Юрию Петровичу получать результат игры без утомительных вычислений вручную.

Формат входных данных.

Входной файл содержит одно целое число N ($0 \leqslant N \leqslant 32767$).

Формат выходных данных.

Ваша программа должна вывести в выходной файл одно целое число, равное результату игры.

Пример.

d.in	d.out
19	28

Задача I-E. Целые точки

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	32 мегабайта

Многоугольник (не обязательно выпуклый) на плоскости задан координатами своих вершин. Требуется подсчитать количество точек с целочисленными координатами, лежащих внутри него (но не на его границе).

Формат входных данных.
В первой строке содержится N ($3 \leqslant N \leqslant 1000$) — число вершин многоугольника. В последующих N строках идут координаты (X_i, Y_i) вершин многоугольника в порядке обхода по часовой стрелке. X_i и Y_i — целые числа, по модулю не превосходящие 1000000.

Формат выходных данных.
В выходной файл вывести одно число — искомое число точек.

Примеры.

e.in	e.out
4 -1 -1 -1 1 1 1 1 -1	1
3 0 0 0 2 2 0	0

Задача I-F. Степень

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	32 мегабайта

Для того чтобы проверить, как ее ученики умеют считать, Мария Ивановна каждый год задает им на дом одну и ту же задачу — для заданного натурального A найти минимальное натуральное N такое, что N в степени N (N, умноженное на себя N раз) делится на A. От года к году и от ученика к ученику меняется только число A.

Вы решили помочь будущим поколениям. Для этого вам необходимо написать программу, решающую эту задачу.

Формат входных данных.
Во входном файле содержится единственное число A ($1 \leqslant A \leqslant 10^9$ — на всякий случай; вдруг Мария Ивановна задаст большое число, чтобы «завалить» кого-нибудь...).

Формат выходных данных.
В выходной файл вывести единственное число N.

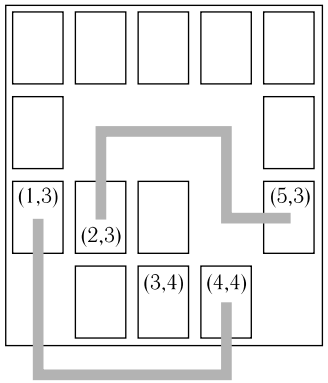
Примеры.

f.in	f.out
8	4
13	13

Задача I-G. Игра с фишками

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	32 мегабайта

Вы являетесь одним из разработчиков новой компьютерной игры. Игра происходит на прямоугольной доске, состоящей из $W \times H$ клеток. Каждая клетка может либо содержать, либо не содержать фишку (см. рис.).



Важной частью игры является проверка того, соединены ли две фишки путем, удовлетворяющим следующим свойствам:

1. Путь должен состоять из отрезков вертикальных и горизонтальных прямых.

2. Путь не должен пересекать других фишек.

При этом часть пути *может* оказаться вне доски. Так, на рисунке фишки с координатами (1, 3) и (4, 4) могут быть соединены; фишки с координатами (2, 3) и (5, 3) тоже могут быть соединены; а вот фишки с координатами (2, 3) и (3, 4) соединить нельзя — любой соединяющий их путь пересекает другие фишки.

Вам необходимо написать программу, проверяющую, можно ли соединить две фишки путем, обладающим вышеуказанными свойствами, и, в случае положительного ответа, определяющую минимальную длину та-

кого пути (считается, что путь имеет изломы, начало и конец только в центрах клеток (или «мнимых клеток», расположенных вне доски), а отрезок, соединяющий центры двух соседних клеток, имеет длину 1).

Формат входных данных.

Первая строка входного файла содержит два натуральных числа: W — ширина доски, H — высота доски ($1 \leqslant W$, $H \leqslant 75$). Следующие H строк содержат описание доски: каждая строка состоит ровно из W символов: символ «X» (заглавная английская буква «экс») обозначает фишку, символ «.» (точка) обозначает пустое место. Все остальные строки содержат описания запросов: каждый запрос состоит из четырех натуральных чисел, разделенных пробелами — X_1, Y_1, X_2, Y_2, причем $1 \leqslant X_1, X_2 \leqslant W$, $1 \leqslant Y_1, Y_2 \leqslant H$. Здесь (X_1, Y_1) и (X_2, Y_2) — координаты фишек, которые требуется соединить (левая верхняя клетка имеет координаты (1, 1)). Гарантируется, что эти координаты не будут совпадать (кроме последнего запроса; см. далее). Последняя строка содержит запрос, состоящий из четырех чисел 0; этот запрос обрабатывать не надо. Количество запросов не превосходит 20.

Формат выходных данных.

Для каждого запроса необходимо вывести одно целое число на отдельной строке — длину кратчайшего пути или 0, если такого пути не существует.

Примеры.

g.in	g.out
5 4 XXXXX X...X XXX.X .XXX. 2 3 5 3 1 3 4 4 2 3 3 4 0 0 0 0	5 6 0
4 4 XXXX .X.. ..X. X... 1 1 3 1 0 0 0 0	4

Задача I-H. Раскопки

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	32 мегабайта

Во время недавних раскопок на Марсе были обнаружены листы бумаги с таинственными символами на них. После долгих исследований ученые пришли к выводу, что надписи на них на самом деле могли быть обычными числовыми равенствами. Если бы этот вывод оказался верным, это доказало бы не только то, что на Марсе много лет назад были разумные существа, но и то, что они уже умели считать…

Ученые смогли понять, что в этом случае означают найденные символы, и перевели эти равенства на обычный математический язык — язык цифр, скобок, знаков арифметических действий и равенства. Кроме того, из других источников было получено веское доказательство того, что марсиане знали только три операции — сложение, умножение и вычитание (марсиане никогда не использовали «унарный минус»: вместо «-5» они писали «0-5»). Также ученые доказали, что марсиане не наделяли операции разным приоритетом, а просто вычисляли выражения (если в них не было скобок) слева направо: например, 3+3*5 у них равнялось 30, а не 18.

К сожалению, символы арифметических действий марсиане почему-то наносили специальными чернилами, которые, как оказалось, были не очень стойкими, и поэтому в найденных листках между числами вместо знаков действий были пробелы. Если вся вышеизложенная теория верна, то вместо этих пробелов можно поставить знаки сложения, вычитания и умножения так, чтобы равенства стали верными. Например, если был найден лист бумаги с надписью «18=7 (5 3) 2», то возможна такая расстановка знаков: «18=7+(5-3)*2» (помните про то, в каком порядке марсиане вычисляют выражения!). В то же время, если попался лист с надписью «5=3 3», то марсиане явно не имели в виду числового равенства, когда писали это…

Вы должны написать программу, находящую требуемую расстановку знаков или сообщающую, что таковой не существует.

Формат входных данных.

Первая строка входного файла состоит из натурального (целого положительного) числа, не превосходящего 2^{30}, знака равенства и последовательности натуральных чисел (не более десяти), произведение которых также не превосходит 2^{30}. Некоторые группы чисел (одно или более) могут быть окружены скобками. Длина входной строки не будет превосходить 80 символов, и других ограничений на количество и вложенность скобок

нет. Между двумя соседними числами, не разделенными скобками, всегда будет хотя бы один пробел, во всех остальных местах может быть любое (в том числе и 0) число пробелов (естественно, внутри числа пробелов нет).

Формат выходных данных.

В выходной файл необходимо вывести одну строку, содержащую полученное равенство (т. е. исходное равенство со вставленными знаками арифметических действий). В случае, если требуемая расстановка знаков невозможна, вывести строку, состоящую из единственного числа -1. Выходная строка не должна содержать пробелов.

Примеры.

h.in	h.out
18=7 (5 3) 2	18=7+(5-3)*2
5= 3 3	-1

Задача I-I. Деревни

Имя входного файла:	i.in
Имя выходного файла:	i.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	32 мегабайта

В тридесятом государстве есть N деревень. Некоторые пары деревень соединены дорогами. В целях экономии «лишних» дорог нет, т. е. из любой деревни в любую можно добраться по дорогам единственным образом.

Новейшие исследования показали, что тридесятое государство находится в сейсмически опасной зоне. Поэтому глава государства захотел узнать, какой ущерб может принести его державе землетрясение. А именно, он хочет узнать, какое *минимальное* число дорог должно быть разрушено, чтобы образовалась изолированная от остальных группа ровно из P деревень такая, что из любой деревни из этой группы до любой другой деревни из этой группы по-прежнему можно было добраться по неразрушенным дорогам (группа *изолирована* от остальных, если никакая неразрушенная дорога не соединяет деревню из этой группы с деревней не из этой группы).

Вы должны написать программу, помогающую ему в этом.

Формат входных данных.

Первая строка входного файла содержит два числа: N и P ($1 \leqslant P \leqslant N \leqslant 150$). Все остальные строки содержат описания дорог, по одному на строке: описание дороги состоит из двух номеров деревень (от 1 до N), которые эта дорога соединяет. Все числа во входном файле разделены пробелами и (или) переводами строки.

Формат выходных данных.

В выходной файл выведите единственное число — искомое количество дорог.

Примеры.

i.in	i.out
3 2 1 2 3 2	1
11 6 1 2 1 3 1 4 1 5 2 6 2 7 2 8 4 9 4 10 4 11	2

Комментарий. Во втором примере группа деревень (1, 2, 3, 6, 7, 8) окажется изолированной от остальных, если разрушить дороги 1–4 и 1–5.

Олимпиада II. Заочный тур 2003–2004 учебного года

Задачи этой олимпиады приводятся без решений.

Задача II-A. Подсчет баллов

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Решение каждой задачи заочного тура проверяется на наборе заранее заготовленных тестов. По результатам работы программы на каждом тесте участнику либо начисляются баллы за этот тест (когда программа выдала правильный ответ), либо не начисляются (когда во время работы программы произошли ошибки или выданный ответ не верен). Тесты могут иметь разную стоимость.

Дополнительные баллы начисляются участнику, если его программа прошла все тесты.

Участник может исправлять свое решение, и посылать его на проверку повторно (решение опять проверяется на том же наборе тестов). При этом за каждую попытку из количества набранных по задаче баллов вычитается штраф, который равен 0 при 1-й попытке, а при каждой следующей возрастает на 2 (т. е. 2 при второй, 4 при третьей, 6 при четвертой и т. д.).

Из баллов, полученных участником за каждую из попыток (с учетом начисленных штрафов), выбирается максимальный результат, который и засчитывается как *результат* данного участника по этой задаче. Это нужно, в частности, для того, чтобы последующие попытки не ухудшали уже полученный участником результат по задаче.

Например, если участник делает первую попытку и набирает 10 баллов, его результат по задаче равен 10 баллов. Пусть на второй попытке участник посылает решение, которое набирает 8 баллов. С учетом штрафа, за эту попытку участник имеет 6 баллов, однако его результат по задаче остается равным 10. Пусть с 3-й попытки решение набрало 20 баллов, тогда (с учетом штрафа) результат участника по задаче становится равен 16 баллам. Наконец, пусть с 4-й попытки решение проходит все тесты, тогда участник получает сумму баллов за все тесты, плюс призовые баллы за прохождение всех тестов, минус 6 баллов штрафа (если, конечно, эта величина не меньше 16 баллов, которые уже были у данного участника).

Напишите программу, которая определяет результат данного участника по этой задаче.

Формат входных данных.

Во входном файле записано сначала число N — количество тестов, на которых проверяются решения данной задачи ($1 \leqslant N \leqslant 100$). Далее идет N натуральных чисел, не превышающих 100, — баллы, которые начисляются за прохождение каждого из тестов. Далее идет целое число из диапазона от 0 до 100 — количество баллов, которое дополнительно начисляется за прохождение всех тестов.

Далее идет натуральное число M — количество попыток сдачи задачи ($1 \leqslant M \leqslant 100$). После чего идет M наборов по N чисел в каждом, задающих результаты проверки каждой из M попыток сдачи задачи на тестах. Цифра 0 обозначает, что соответствующий тест не пройден, 1 — пройден.

Формат выходных данных.

В выходной файл выведите M чисел, i-е число должно соответствовать результату участника после совершения им первых i попыток.

Примеры.

a.in	a.out
4 1 2 3 4 5 3 0 0 0 0 1 1 1 1 0 1 0 1	0 13 13
2 1 8 0 3 0 0 1 0 0 1	0 0 4

Задача II-B. Великая сеча

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Алеша Попович и Добрыня Никитич сражаются со стаей двух- и трехголовых драконов. Они по очереди взмахивают мечами, и одним махом могут отрубить любое (по своему желанию) число голов, но только у одного дракона. Отрубивший последнюю голову у последнего дракона получает в жены прекрасную принцессу.

Кто из богатырей (начинающий или второй) может получить в жены принцессу независимо от действий другого?

Формат входных данных.

Во входном файле записано два числа: N и M — количество двух- и трехголовых драконов соответственно (оба числа целые, из диапазона от 0 до 100).

Формат выходных данных.

В выходной файл выведите сначала число 1 или 2, определяющее, кто из богатырей имеет все шансы получить в жены принцессу (1 — тот, кто начинает, 2 — второй). В случае 1 выведите также все варианты его первого хода, которые к этому приводят: сначала выведите количество различных выигрышных ходов (при этом отрубание одинакового количества голов у разных двухголовых драконов считается одним и тем же ходом, так же и для трехголовых), а затем сами ходы. Каждый ход задается парой чисел: первое число определяет у скольких голового дракона нужно отрубать головы, а второе — сколько голов нужно отрубать.

Примеры.

b.in	b.out
2 2	2
1 2	1 2 2 2 3 2

Задача II-C. Водостоки

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Карту местности условно разбили на квадраты, и посчитали среднюю высоту над уровнем моря для каждого квадрата.

Когда идет дождь, вода равномерно выпадает на все квадраты. Если один из четырех соседних с данным квадратов имеет меньшую высоту над уровнем моря, то вода с текущего квадрата стекает туда (и, если есть возможность, то дальше), если же все соседние квадраты имеют бóльшую высоту, то вода *скапливается* в этом квадрате.

Разрешается в некоторых квадратах построить водостоки. Когда в каком-то квадрате строят водосток, то вся вода, которая раньше скапливалась в этом квадрате, будет утекать в водосток.

Если есть группа квадратов, имеющих одинаковую высоту и образующих связную область, то если рядом хотя бы с одним из этих квадратов есть квадрат, имеющий меньшую высоту, то вся вода утекает туда, если же такого квадрата нет, то вода стоит во всех этих квадратах. При этом достаточно построить водосток в любом из этих квадратов, и вся вода с них будет утекать в этот водосток.

Требуется определить, какое минимальное количество водостоков нужно построить, чтобы после дождя вся вода утекала в водостоки.

Формат входных данных.

Во входном файле записаны сначала числа: N и M, задающие размеры карты, — натуральные числа, не превышающие 100. Далее идет N строк по M чисел в каждой, задающих высоту квадратов карты над уровнем моря. Высота задается натуральным числом, не превышающим 10000. Считается, что квадраты, расположенные за пределами карты, имеют высоту 10001 (т. е. вода никогда не утекает в квадраты за пределами карты).

Формат выходных данных.

В выходной файл выведите минимальное количество водостоков, которое необходимо построить.

Пример.

c.in	c.out
4 5 1 2 3 1 10 1 4 3 10 10 1 5 5 5 5 6 6 6 6 6	2

Задача II-D. Коллекционирование этикеток

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Вася коллекционирует спичечные этикетки. Для этого у него есть N альбомов вместимостью K_1, K_2, …, K_N этикеток. Вася хочет, чтобы в случае утери одного любого альбома каждая этикетка осталась у него хотя бы в одном экземпляре. Для этого он покупает каждую этикетку в двух экземплярах, и наклеивает их в два разных альбома. Какое максимальное количество различных этикеток при этом может оказаться в его коллекции?

Формат входных данных.

Входной файл содержит сначала число N — количество альбомов, а затем N чисел K_1, K_2, …, K_N, задающих вместимости альбомов. N — натуральное число из диапазона от 2 до 1000. Вместимость каждого альбома задается натуральным числом, суммарная вместимость всех альбомов не превышает 100000 этикеток.

Формат выходных данных.

В выходной файл выведите сначала число E — максимальное количество различных этикеток, которое может собрать Вася с соблюдением выдвинутого условия. Затем выведите E пар чисел — каждая пара чисел задает номера двух альбомов, куда будет вклеена очередная этикетка.

Пример.

d.in	d.out
4 1 2 1 1	2 1 2 2 4

Задача II-E. Еловая аллея

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Мэр города Урюпинска решил посадить на главной аллее города, которая проходит с запада на восток, голубые ели. Причем сажать ели можно не во всех местах, а только на специально оставленных при асфальтировании аллеи клумбах.

Как оказалось, голубые ели бывают M различных сортов. Для ели каждого сорта известна максимальная длина ее тени в течение дня в западном и в восточном направлении (W_i и E_i соответственно). При этом известно, что ели растут гораздо лучше, если в течение дня они не оказываются в тени других елей.

Координатная ось направлена вдоль аллеи с запада на восток.

По заданным координатам клумб вычислите максимальное число елей, которое можно посадить, соблюдая условие о том, что никакая ель не должна попадать в тень от другой ели.

Формат входных данных.

Во входном файле записано сначала натуральное число M — количество сортов елей ($1 \leqslant M \leqslant 100$). Затем идет M пар чисел W_i, E_i, описывающих максимальную длину тени в западном и восточном направлении в течение дня для каждого сорта ели (числа W_i, E_i — целые, из диапазона от 0 до 30000). Далее идет натуральное число N — количество клумб, в которые можно сажать ели ($1 \leqslant N \leqslant 100$). Далее идет N чисел, задающих координаты клумб (координаты — целые числа, по модулю не превышающие 30000). Клумбы перечислены с запада на восток (в порядке возрастания их координат).

Примечание. Если на клумбе с координатой X мы посадили ель, максимальная тень которой в восточном направлении равна E, то все клумбы с координатами от $X+1$ до $X+E-1$ попадают в тень от этой ели, а клумба с координатами $X+E$ — уже нет. Аналогично для тени в западном направлении.

Формат выходных данных.

В выходной файл выведите сначала число A — максимальное количество елей, которые удастся посадить, а затем A пар чисел, описывающих ели. Первое число каждой пары задает номер клумбы, в которую сажают ель. Второе число определяет номер сорта этой ели.

Пример.

e.in	e.out
3 1000 3 1 200 128 256 3 1 2 4	2 1 1 3 2

Задача II-F. Шашки

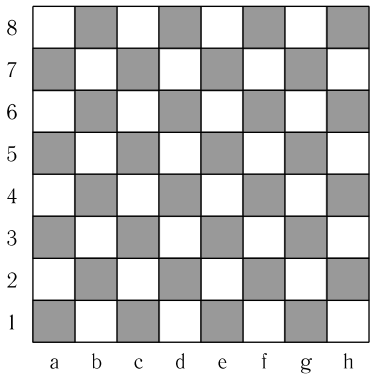
Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Петя и Вася играли в шашки по описанным ниже правилам. В какой-то момент забежавший в комнату кот перевернул доску, на которой играли Петя и Вася. К счастью, у них осталась запись сделанных ходов, используя которую можно восстановить расположение шашек к тому моменту, когда забежал кот.

Напишите программу, которая выведет положение шашек на доске после выполнения описанных ходов.

Правила игры.

Игра происходит на стандартной доске (8×8), которая располагается так, что у игрока, играющего белыми, левая нижняя клетка является черной и с нее начинается нумерация как строк, так и столбцов. Строки доски нумеруются числами от 1 до 8, столбцы — латинскими буквами от a до h.



В начале игры каждый из двух игроков имеет по 12 шашек своего цвета (белого или черного соответственно). Белые шашки располагаются на клетках a1, a3, b2, c1, c3, d2, e1, e3, f2, g1, g3, h2. Черные шашки располагаются на клетках a7, b6, b8, c7, d6, d8, e7, f6, f8, g7, h6, h8.

Игроки совершают ходы по очереди. Белые ходят первыми.

Шашки в процессе игры бывают двух видов: *обычная шашка* и *дамка*. В начале игры все шашки обычные. Белая шашка становится дамкой, если она оказывается в строке 8. Соответственно, черная шашка становится дамкой, если она оказывается в строке 1.

Шашка может совершать ходы двух типов:

1. *Простой ход* заключается в перемещении одной из шашек на одну клетку вперед по диагонали. Например, белая шашка с e3 может сходить на d4 или f4 (если соответствующая клетка свободна). А черная шашка с e3 может сходить на d2 или f2.

2. *Рубка* заключается в том, что шашка перепрыгивает через шашку (или дамку) противника, находящуюся в диагонально соседней с ней клетке при условии, что следующая клетка этой диагонали свободна. Шашка противника, которую срубили, убирается с доски. Если сразу после окончания рубки та же самая шашка может продолжить рубку, она ее продолжает этим же ходом. Рубка возможна в любом из 4-х диагональных направлений. Если в процессе рубки шашка оказывается в 1-й строке (для черных) или в 8-й (для белых), она становится дамкой.

Дамка может совершать следующие ходы:

1. *Простой ход* заключается в перемещении дамки по диагонали на любое число клеток (при этом все клетки, через которые происходит перемещение, должны быть свободны).

2. *Рубка* заключается в том, что шашка перепрыгивает через шашку (или дамку) противника, находящуюся на той же диагонали, что и рубящая дамка. Это можно делать при условии, что все клетки между рубящей дамкой и шашкой, которую рубят, а также клетка, следующая за шашкой, которую рубят, свободны. После рубки дамка может встать на любую клетку данной диагонали за клеткой, на которой стояла срубленная шашка (при условии, что все клетки на ее пути свободны). Если из своего нового положения дамка может совершить рубку, она должна ее совершить этим же ходом.

Формат входных данных.

В первой строке входного файла записано одно число N — количество ходов, которое было сделано с начала партии. Это число не превышает 1000.

В каждой из следующих N строк приводятся описания ходов (нечетные ходы совершались белыми, четные — черными). Описание каждого хода занимает ровно одну строку и записывается в следующем виде.

Простой ход записывается в виде `<начальная клетка>-<конечная клетка>`. Например, ход с c3 на d4 записывается как c3-d4.

Рубка записывается в виде `<начальная клетка>:<клетка после рубки>`. Если рубка продолжается, то ставится еще одно двоеточие и записывается клетка, где окажется шашка после следующей рубки и т. д. Например, e3:c5:e7 (шашка, изначально расположенная на e3, рубит шашку на d4 и оказывается на c5, после чего рубит шашку на d6 и оказывается на e7).

Рубка a1:h8 может быть осуществлена только дамкой (например, дамка с a1 рубит шашку, стоящую на c3, и заканчивает ход на h8). Рубка дамкой может быть и такой a1:d4:f6:h4 (дамка рубит шашку, стоящую на b2, затем продолжает рубку и рубит шашку на e5, и наконец, рубит шашку на g5). При этом после каждой рубки указывается клетка, на которой останавливается дамка перед следующей рубкой.

Строки с описанием ходов не содержат пробельных символов.

Формат выходных данных.

В выходной файл выведите изображение доски с расположенными на ней шашками. В первой строке выходного файла должна быть выведена 8-я строка доски, во второй — 7-я и т. д. В каждой строке должно быть ровно 8 символов, описывающих клетки столбцов с a по h.

Белая клетка должна быть изображена символом «.» (точка), пустая черная клетка — символом «-» (минус). Черная клетка, в которой стоит белая шашка — символом «w» (маленькая латинская буква w), а клетка с белой дамкой — символом «W» (большая латинская буква W). Аналогично клетка с черной шашкой должна быть изображена символом «b» (маленькая латинская буква b), а клетка с черной дамкой — символом «B» (большая латинская буква B).

Пример.

f.in	f.out
3 c3-d4 f6-e5 d4:f6	.b.b.b.b b.b.b.b. .b.b.w.b -.-.-.-. .-.-.-.- w.-.w.w. .w.w.w.w w.w.w.w.

Задача II-G. Мышка с колесиком

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Пользователь просматривает таблицу в Internet Explorer и пользуется для прокрутки изображения колесиком на мышке. При этом все изображение сдвигается вверх или вниз на T пикселей. Пользователю очень не нравится, когда курсор мыши оказывается на горизонтальных линиях, разделяющих строки таблицы. Поэтому он хочет выбрать такое положение для курсора мыши на экране, чтобы в процессе прокрутки до конца таблицы курсор как можно меньшее число раз пересекался с линиями таблицы.

При этом если в каком-то положении курсор оказывается на двух линиях таблицы, то это считается за два пересечения курсора с линиями таблицы. Если какую-то линию курсор мыши пересекает в двух положениях (т. е. например, высота курсора 10 пикселей, а при прокрутке таблица сдвигается на 7 пикселей, тогда курсор мыши может оказываться на одной линии в двух состояниях прокрутки), то это также считается за два пересечения.

Экран монитора имеет разрешение по вертикали U пикселей. Координаты введены так, что самые верхние точки экрана имеют координату 0, а нижние — координату $U-1$.

Курсор мыши имеет высоту H пикселей. Расположением курсора считается самая верхняя точка курсора. Таким образом, если мы говорим, что он расположен, например, в точке с координатами 0 на экране, то его изображение расположено в точках с координатами от 0 до $H-1$. Курсор мыши всегда целиком помещается на экране, т. е. допустимыми координатами для его расположения являются координаты от 0 до $U-H$.

Таблица, которую просматривает пользователь, имеет высоту L пикселей и состоит из $N-1$ строки, и следовательно, в ней N горизонтальных линий, которые имеют координаты X_1, X_2, …, X_N. При этом $0 = X_1 < X_2 < X_3 < \ldots < X_N = L-1$.

В начальный момент времени таблица расположена так, что линия, имеющая координату 0, в таблице отображается в 0-й строке пикселей монитора. Далее при прокрутке таблица каждый раз сдвигается на T пикселей (т. е. в 0-й строке монитора оказывается строка пикселей, имеющая в таблице координату T, координату $2T$ и т. д.). Так происходит до тех пор, пока на экране не окажется нижняя линия таблицы (которая имеет координату X_N). После этого дальнейшая прокрутка не происходит (если изначально $X_N < U$, то прокрутка вообще не происходит).

Формат входных данных.

Во входном файле задано сначала разрешение монитора по вертикали U, затем высота курсора мыши H, затем шаг прокрутки T. Далее задана высота таблицы L. Далее задано количество разделительных линий в таблице N и координаты X_1, X_2, …, X_N, где расположены эти линии относительно начала таблицы.

Ограничения.

- $10 \leqslant U \leqslant 512$,
- $1 \leqslant H \leqslant U$,
- $1 \leqslant T \leqslant U$,
- $2 \leqslant N \leqslant 200000$,
- $0 = X_1 < X_2 < \ldots < X_N = L-1 \leqslant 10^9$.

Формат выходных данных.

В выходной файл выведите сначала координату, в которой нужно расположить курсор мыши, а затем количество пересечений курсора мыши с линиями таблицы. В случае, если существует несколько начальных положений курсора мыши, выведите любое из них.

Примеры.

g.in	g.out
10 3 10 10 4 0 1 6 9	2 0

g.in	g.out
10 6 2 21 11 0 2 4 6 8 10 12 14 16 18 20	0 21

Задача II-H. «Левый лабиринт»

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

В спортзале размером $N \times M$ метров построили современный аттракцион под названием «Левый лабиринт». Для этого на полу спортзала с интервалом в 1 метр начертили линии, параллельные стенам спортзала. Таким образом, спортзал разбили на $N \times M$ клеток. Затем некоторые из этих клеток покрасили в черный цвет.

Аттракцион заключается в том, что участника ставят в некоторой клетке спортзала и просят как можно быстрее добежать до некоторой другой клетки. При этом накладываются следующие условия.

Участнику запрещено ходить по черным клеткам.

Придя в какую-то клетку, участник может пойти либо прямо, либо налево, либо направо (если в соответствующем направлении клетка не покрашена в черный цвет); ходить назад, а также ходить по диагонали запрещается.

За все время пути участнику разрешается повернуть направо (т. е. пойти из текущей клетки направо относительно того, откуда он пришел в данную клетку) не более K раз.

В начальной клетке участник может встать лицом в ту сторону, в какую ему захочется. С какой стороны участник прибежит в конечную клетку также не важно.

Известно, что на то, чтобы перебежать из клетки в соседнюю, участник тратит ровно 1 секунду. Требуется вычислить минимальное время, за которое участник сможет достичь конечной клетки.

Формат входных данных.

Во входном файле сначала записано число K — количество разрешенных поворотов направо (целое число из диапазона от 0 до 5), затем записаны числа N и M, задающие размеры спортзала, — натуральные числа, не превышающие 20. Далее записано N строк по M чисел в каждой. Число 0 обозначает непокрашенную клетку, число 1 — покрашенную, число 2 — клетку, откуда стартует участник, и число 3 — клетку, куда нужно добежать (клетки, помеченные 2 и 3, являются непокрашенными). В лабиринте всегда есть ровно одна начальная клетка и ровно одна клетка, в которую нужно попасть.

Формат выходных данных.

В выходной файл выведите минимальное время, за которое можно добраться в конечную клетку. Если попасть в конечную клетку с соблюдением всех условий нельзя, выведите -1.

Примеры.

h.in	h.out
1 5 5 2 0 0 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0 1 3 1 0 0 1	12
0 3 4 2 0 0 0 1 1 1 0 3 0 0 0	-1
0 1 3 2 1 3	-1

Задача II-I. Дремучий лес – 2

Имя входного файла:	i.in
Имя выходного файла:	i.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Будем говорить, что для наблюдателя лес является дремучим, если из своего текущего положения наблюдатель видит только деревья. Вам дана карта леса и координаты точки, в которой находится наблюдатель. Требуется определить, кажется ли лес дремучим данному наблюдателю.

На карте леса все деревья изображаются кругами. При этом в лесу бывают сросшиеся деревья (изображения таких деревьев на карте пересекаются), также одно дерево может находиться внутри другого. Точка, в которой стоит наблюдатель, не лежит внутри или на границе ни одного из деревьев.

Формат входных данных.

Во входном файле содержится сначала целое число N — количество деревьев ($1 \leqslant N \leqslant 50000$). Затем идут два числа, задающие координаты наблюдателя. Затем идет N троек чисел, задающих деревья. Первые два числа задают координаты центра, а третье — радиус. Все координаты задаются точно и выражаются вещественными числами не более чем с 2 знаками после десятичной точки, по модулю не превосходящими 100000.

Формат выходных данных.

В первой строке выходного файла должно содержаться сообщение `YES`, если лес является дремучим, и `NO` — если нет. Во втором случае вторая строка выходного файла должна содержать координаты точки, глядя в направлении которой наблюдатель не видит деревьев (т. е. луч, вдоль которого смотрит наблюдатель, не проходит внутри деревьев и не касается ни одного из деревьев). Координаты нужно вывести не менее чем с 3 знаками после десятичной точки. Модули координат не должны превышать 300000. Расстояние между выданной точкой и наблюдателем должно быть не меньше 1.

Примеры.

i.in	i.out
4 0 0 2 2 2 -2 2 2 -2 -2 2 2 -2 2	YES
2 10 10 0 0 1 0.5 0 2	NO 100.000 100.000

Задача II-J. Анаграммер

Имя входного файла:	j.in
Имя выходного файла:	j.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	8 мегабайт

Анаграммер — специальное устройство для получения из слова его анаграмм (т. е. слов, записанных теми же буквами, но в другом порядке). Это устройство умеет выполнять 2 операции:

1. Взять очередную букву исходного слова и поместить ее в стек.
2. Взять букву из стека и добавить ее в конец выходного слова.

Стек — это хранилище данных, которое работает по принципу «первый пришел — последний ушел». Стек можно представить себе в виде пирамидки. Когда мы добавляем букву в стек, это соответствует тому, что на стержень пирамидки сверху мы надеваем кольцо, на котором написана соответствующая буква. Когда берем букву из стека, то это соответствует тому, что мы снимаем со стержня верхнее кольцо и смотрим, какая буква на нем написана.

Например, слово TROT в слово TORT может быть преобразовано анаграммером двумя различными последовательностями операций: 11112222 или 12112212.

Напишите программу, которая по двум заданным словам вычисляет количество различных последовательностей операций анаграммера, которые преобразуют первое из этих слов во второе, а также находит сами эти последовательности.

Формат входных данных.

Первая строка входного файла содержит исходное слово, а вторая — слово, которое необходимо получить. Слова состоят только из заглавных латинских букв и имеют длину не более 50 символов. Оба слова имеют одинаковую длину. В этих строках не содержится пробелов.

Формат выходных данных.

В первой строке выходного файла должно содержаться количество последовательностей операций анаграммера, с помощью которых можно преобразовать первое слово во второе.

Если это количество не превышает 1000, то в последующих строках должны содержаться сами последовательности. Каждая последовательность должна быть выведена на отдельной строке и состоять из цифр 1 и 2 (указывающих порядок выполнения операций), выведенных без пробелов.

Примеры.

j.in	j.out
MADAM ADAMM	4 1111222122 1111222212 1121212122 1121212212
LONG GONG	0
AAAAAAAA AAAAAAAA	1430

Олимпиада III. Личная олимпиада 2003–2004 учебного года

Задача III-A. Наибольшее произведение

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	80 баллов

Дано N целых чисел. Требуется выбрать из них три таких числа, произведение которых максимально.

Формат входных данных.

Во входном файле записано сначала число N — количество чисел в последовательности ($3 \leqslant N \leqslant 10^6$). Далее записана сама последовательность: N целых чисел, по модулю не превышающих 30000.

Формат выходных данных.

В выходной файл выведите три искомых числа в любом порядке. Если существует несколько различных троек чисел, дающих максимальное произведение, то выведите любую из них.

Примеры.

a.in	a.out
9 3 5 1 7 9 0 9 -3 10	9 10 9
3 -5 -30000 -12	-5 -30000 -12

Система оценки.

Решения для ограничения $3 \leqslant N \leqslant 100$ оцениваются 30 баллами, для ограничения $3 \leqslant N \leqslant 5000$ — 60 баллами.

Задача III-B. Покупка билетов

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	100 баллов

За билетами на премьеру нового мюзикла выстроилась очередь из N человек, каждый из которых хочет купить 1 билет. На всю очередь работала только одна касса, поэтому продажа билетов шла очень медленно, приводя «постояльцев» очереди в отчаяние. Самые сообразительные быстро заметили, что, как правило, несколько билетов в одни руки кассир продает быстрее, чем когда эти же билеты продаются по одному. Поэтому они предложили нескольким подряд стоящим людям отдавать деньги первому из них, чтобы он купил билеты на всех.

Однако для борьбы со спекулянтами кассир продавала не более 3 билетов в одни руки, поэтому договориться таким образом между собой могли лишь 2 или 3 подряд стоящих человека.

Известно, что на продажу i-му человеку из очереди одного билета кассир тратит A_i секунд, на продажу двух билетов — B_i секунд, трех билетов — C_i секунд. Напишите программу, которая подсчитает минимальное время, за которое могли быть обслужены все покупатели.

Обратите внимание, что билеты на группу объединившихся людей всегда покупает первый из них. Также никто в целях ускорения не покупает лишних билетов (т. е. билетов, которые никому не нужны).

Формат входных данных.

Во входном файле записано сначала число N — количество покупателей в очереди ($1 \leqslant N \leqslant 5000$). Далее идет N троек натуральных чисел A_i, B_i, C_i. Каждое из этих чисел не превышает 3600. Люди в очереди нумеруются начиная от кассы.

Формат выходных данных.

В выходной файл выведите одно число — минимальное время в секундах, за которое могли быть обслужены все покупатели.

Примеры.

b.in	b.out
5 5 10 15 2 10 15 5 5 5 20 20 1 20 1 1	12
2 3 4 5 1 1 1	4

Задача III-C. Вырезанные фигуры

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	8 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	120 баллов

Эпидемия гриппа не обошла стороной семиклассника Алешу. Скучая дома, Алеша решил вырезать фигурки из листа клетчатой бумаги. Лист состоял из M строк и N столбцов клеток. Сначала Алеша нарисовал на листе границы фигур. Количество фигур было не меньше 2. Чтобы фигуры получались ровными, границы фигур Алеша рисовал строго по линиям имеющейся клетчатой разметки листа (при этом некоторые границы фигур могли пройти по границам листа). Форма фигур могла быть любой, но при этом все фигуры были связными (фигура называется *связной*, если из любой ее клетки можно добраться до любой другой, ходя только по клеткам фигуры и перемещаясь каждый раз в одну из 4 соседних по стороне клеток). Никакие две фигуры не имели общих точек, в том числе не касались углами клеток.

Затем Алеша вырезал нарисованные фигуры, делая разрезы только по их границам. При этом оставшаяся часть листа осталась связной (т. е. не распалась на несколько частей).

Лист с вырезами Алеша отсканировал. Сканер в своей памяти по результатам сканирования построил таблицу, состоящую из нулей и единиц, из M строк и N столбцов (строки нумеруются сверху вниз от 1 до M, столбцы — слева направо от 1 до N). Каждый элемент таблицы соответствовал клетке исходного листа. Единица обозначала, что соответствующая клетка листа осталась на месте, ноль — соответствующая клетка была вырезана.

На рис. 1 приведен пример клетчатого листа, а на рис. 2 — соответствующая ему таблица в памяти сканера.

После этого сканер представил полученную таблицу в специальном, описанном ниже формате и передал информацию на компьютер. Напишите программу, которая по полученной информации установит:

1. Сколько клеток было вырезано из листа?
2. Сколько фигур было вырезано?

Описание формата представления таблицы.

Последовательность подряд идущих по горизонтали или вертикали единиц будем называть полосой. Полосу можно задать 4 числами:

- направление (0 — горизонтальная, 1 — вертикальная);

- (i, j) — координаты начальной клетки полосы (начальной является самая левая клетка для горизонтальной полосы и самая верхняя — для вертикальной), i — номер строки клетки, j — номер столбца;
- d — длина полосы (количество подряд стоящих единиц).

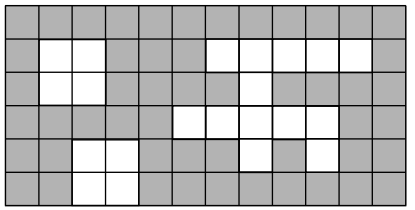


Рис. 1. Исходный клетчатый лист с вырезанными фигурами. Размер листа: $M = 6$, $N = 12$. Количество вырезанных фигур 3

1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	0	0	0	0	0	1
1	0	0	1	1	1	1	0	1	1	1	1
1	1	1	1	1	0	0	0	0	0	1	1
1	1	0	0	1	1	1	0	1	0	1	1
1	1	0	0	1	1	1	1	1	1	1	1

Рис. 2. Такая таблица строится в памяти сканера

Всю таблицу разобьем на полосы, состоящие из единиц, так, чтобы каждая единица принадлежала хотя бы одной полосе. При этом полосы могут пересекаться, а также накладываться. Таким образом, таблица представляется в виде описания всех полос, которое удовлетворяет трем дополнительным требованиям:

- в каждой клетке начинается не более одной полосы;
- полосы перечислены в порядке следования их начальных клеток (клетки перечисляются по строкам сверху вниз, в строке — слева направо);
- общее число полос не превышает 256000.

Заметим, что таблица может быть представлена в виде полос разными способами, но каждое представление позволяет однозначно восстановить таблицу.

Формат входных данных.

Во входном файле записано сначала число P (1 или 2) — номер пункта задачи, ответ на который требуется получить. Далее записаны размеры исходного листа — числа M и N ($1 \leqslant M \leqslant 4000$, $1 \leqslant N \leqslant 4000$). Затем записано число K ($0 \leqslant K \leqslant 256000$) — количество полос в описании полученной таблицы. Затем идет K четверок чисел, описывающих полосы (полосы перечисляются в порядке начальных клеток полос: по строкам сверху вниз, в строке — слева направо).

Формат выходных данных.

В выходной файл выведите искомое количество (если $P = 1$ — количество клеток, вырезанных из листа, если $P = 2$ — количество фигур, вырезанных из листа).

Примеры.

c.in	c.out
1 40 400 2 1 1 100 40 0 1 101 1	15959
2 40 400 2 1 1 100 40 1 1 101 1	2
1 6 12 17 0 1 1 10 1 1 4 4 0 1 5 2 0 1 7 6 1 2 1 5 1 2 5 4 0 2 6 1 1 2 12 5 0 3 4 4 0 3 10 3 0 4 2 2 1 4 11 3 1 5 2 1 0 5 6 2 1 5 9 2 0 6 1 2 0 6 5 7	22
2 6 12 12 0 1 1 12 1 1 12 6 1 2 1 5 0 2 4 3 0 3 4 4 0 3 10 3 1 3 11 4 0 4 1 5 1 5 2 2 0 5 5 3 1 5 9 2 0 6 5 8	3

Система оценки.

Полное решение каждого из пунктов 1 и 2 оценивается 60 баллами. Решение для ограничения $1 \leqslant M \leqslant 100$, $1 \leqslant N \leqslant 100$ в каждом из пунктов оценивается 20 баллами.

Задача III-D. Квадрат

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	60 баллов

Требуется в каждую клетку квадратной таблицы размером $N \times N$ поставить ноль или единицу так, чтобы в любом квадрате размером $K \times K$ было ровно S единиц.

Формат входных данных.

Во входном файле записаны три числа: N, K, S ($1 \leqslant N \leqslant 100$, $1 \leqslant K \leqslant N$, $0 \leqslant S \leqslant K^2$).

Формат выходных данных.

В выходной файл выведите заполненную таблицу. Числа в строке должны разделяться пробелом, каждая строка таблицы должна быть выведена на отдельной строке файла. Если решений несколько, выведите любое из них.

Примеры.

d.in	d.out
3 2 1	0 0 0 0 1 0 0 0 0
4 2 2	1 0 0 1 0 1 1 0 1 0 0 1 0 1 1 0

Примечание. Частичные решения для случая $K = 2$ будут оцениваться примерно половиной общего числа баллов.

Задача III-E. Поле чудес

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	60 баллов

Для игры в «Поле чудес» используется круглый барабан, разделенный на сектора, и стрелка. В каждом секторе записано некоторое число. В различных секторах может быть записано одно и то же число.

Однажды ведущий решил изменить правила игры. Он сам стал вращать барабан и называть игроку (который барабана не видел) все числа подряд в том порядке, в котором на них указывала стрелка в процессе вращения барабана. Получилось так, что барабан сделал целое число оборотов, т. е. последний сектор совпал с первым.

После этого ведущий задал участнику вопрос: какое наименьшее число секторов может быть на барабане? Напишите программу, отвечающую на этот вопрос.

Формат входных данных.

Во входном файле записано сначала число N — количество чисел, которое назвал ведущий ($2 \leqslant N \leqslant 30000$). Затем записано N чисел, на которые указывала стрелка в процессе вращения барабана. Первое число всегда совпадает с последним (в конце стрелка указывает на тот же сектор, что и в начале). Числа, записанные в секторах барабана, — натуральные, не превышающие 32000.

Формат выходных данных.

Выведите минимальное число секторов, которое может быть на барабане.

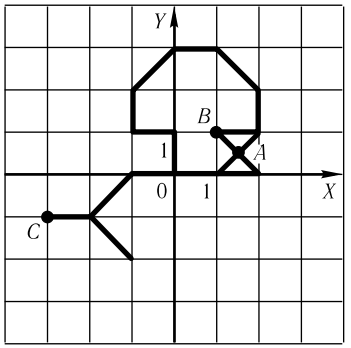
Примеры.

e.in	e.out
13 5 3 1 3 5 2 5 3 1 3 5 2 5	6
4 1 1 1 1	1
4 1 2 3 1	3

Задача III-F. Детская игра со спичками

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	120 баллов

На клетчатом поле введена система координат так, что центр координат находится в точке пересечения линий сетки и оси направлены вдоль линий сетки.



На этом поле выложили связную фигуру, состоящую из спичек. Использовались спички двух типов:

- спички длиной 1 выкладывались по сторонам клеток;
- спички длиной $\sqrt{2}$ выкладывались по диагоналям клеток.

Ребенок хочет сжечь фигуру. При этом он может поджечь ее в одной точке, имеющей целочисленные координаты (например, в точке A на рисунке поджигать фигуру нельзя, а в точках B и C — можно).

Известно, что огонь распространяется вдоль спички равномерно (но по каждой спичке — со своей скоростью). Спичка может гореть в нескольких местах (например, когда она загорается с двух концов; или когда в середине диагональной спички огонь перекидывается с одной спички на другую — огонь расползается по вновь подожженной спичке в обе стороны).

Напишите программу, которая определит, в какой точке нужно поджечь фигуру, чтобы она сгорела за минимальное время.

Формат входных данных.

Во входном файле записано сначала число N — количество спичек ($1 \leqslant N \leqslant 40$). Затем идет N пятерок чисел вида X_1, Y_1, X_2, Y_2, T, задающих координаты концов спички и время ее сгорания при условии, что она будет подожжена с одного конца (гарантируется, что каждая спичка имеет длину 1 или $\sqrt{2}$, все спички образуют связную фигуру и положение никаких двух спичек не совпадает). Все координаты — целые числа, по модулю не превышающие 200, время сгорания — натуральное число, не превышающее 10^7.

Формат выходных данных.

Выведите координаты целочисленной точки, в которой нужно поджечь фигуру, чтобы она сгорела за наименьшее время, а затем время, за которое в этом случае фигура сгорит. Время должно быть выведено с точностью не менее 2 знаков после десятичной точки. Если решений несколько, выведите любое из них.

Примеры.

f.in	f.out
1 0 0 1 1 1	0 0 1.00
5 0 0 0 1 1 1 0 0 1 10 0 0 1 0 1 0 0 1 1 1 2 2 1 1 1	0 0 3.25
3 1 1 1 2 10 1 2 2 2 10 1 1 2 2 50	2 2 35.00
16 0 0 0 1 1 -2 -1 -3 -1 1 -2 -1 -1 0 1 -2 -1 -1 -2 1 -1 0 0 0 1 0 3 1 3 1 1 3 2 2 1 2 2 2 1 1 2 1 1 0 1 2 0 1 1 1 2 0 1 0 1 2 1 1 1 1 0 0 1 0 1 0 1 -1 1 1 -1 1 -1 2 1 0 3 -1 2 1	0 0 4.50

Примечание. Частичные решения для случая, когда время сгорания каждой из спичек равно 1 (вне зависимости от ее длины), будут оцениваться приблизительно половиной общего количества баллов.

Задача III-G. Реклама

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	4 мегабайта
Максимальная оценка за задачу:	80 баллов

В супермаркете решили время от времени транслировать рекламу новых товаров. Для того чтобы составить оптимальное расписание трансляции рекламы, руководство супермаркета провело следующее исследование: в течение дня для каждого покупателя, посетившего супермаркет, было зафиксировано время, когда он пришел в супермаркет и когда он из него ушел.

Менеджер по рекламе предположил, что такое расписание прихода-ухода покупателей сохранится и в последующие дни. Он хочет так составить расписание трансляции рекламных роликов, чтобы каждый покупатель услышал не меньше двух рекламных объявлений. В то же время, он выдвинул условие, чтобы два рекламных объявления не транслировались одновременно и, поскольку продавцам все время приходится выслушивать эту рекламу, чтобы общее число рекламных объявлений за день было минимальным.

Напишите программу, которая составит такое расписание трансляции рекламных роликов. Рекламные объявления можно начинать транслировать только в целочисленные моменты времени. Считается, что каждое рекламное объявление заканчивается до наступления следующего целочисленного момента времени. Если рекламное объявление транслируется в тот момент времени, когда покупатель входит в супермаркет или уходит из него, покупатель это объявление услышать успевает.

Формат входных данных.

Во входном файле записано сначала число N — количество покупателей, посетивших супермаркет за день($1 \leqslant N \leqslant 3000$). Затем идет N пар натуральных чисел A_i, B_i, задающих, соответственно, время прихода и время ухода покупателей из супермаркета ($0 < A_i < B_i < 10^6$).

Формат выходных данных.

В выходной файл выведите сначала количество рекламных объявлений, которые будут протранслированы за день. Затем выведите в порядке возрастания моменты времени, в которые нужно транслировать рекламные объявления.

Если решений несколько, выведите любое из них.

Пример.

g.in	g.out
5 1 10 10 12 1 10 1 10 23 24	5 5 10 12 23 24

Олимпиада IV. Командная олимпиада 2003–2004 учебного года

Задачи этой олимпиады приводятся без решений.

Задача IV-A. Перегоны

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

На некоторой железнодорожной ветке расположено N станций, которые последовательно пронумерованы числами от 1 до N. Известны расстояния между некоторыми станциями. Требуется точно вычислить длины всех перегонов между соседними станциями или указать, что это сделать невозможно (т. е. приведенная информация является противоречивой или ее недостаточно).

Формат входных данных.

Во входном файле записаны сначала числа: N — количество станций ($2 \leqslant N \leqslant 100$) и E — количество пар станций, расстояния между которыми заданы ($0 \leqslant E \leqslant 10000$). Далее идет E троек чисел, первые два числа каждой тройки задают номера станций (это числа из диапазона от 1 до N), а третье — расстояние между этими станциями (все эти расстояния заданы точно и выражаются вещественными неотрицательными числами не более чем с 3 знаками после десятичной точки).

Формат выходных данных.

В случае, когда восстановить длины перегонов можно однозначно, в выходной файл выведите сначала число 1, а затем $N - 1$ вещественное число. Первое из этих чисел должно соответствовать расстоянию от 1-й станции до 2-й, второе — от 2-й до 3-й и т. д. Все числа должны быть выведены с точностью до 3 знаков после десятичной точки.

Если приведенная информация о расстояниях между станциями является противоречивой или не позволяет однозначно точно восстановить длины перегонов, выведите в выходной файл одно число 2.

Примеры.

a.in	a.out
3 2 1 2 1.250 3 1 3	1 1.250 1.750

a.in	a.out
3 2 1 2 1.250 3 1 3	1 1.250 1.750
4 4 1 2 1.250 1 3 1.255 2 4 0.010 1 1 0.000	1 1.250 0.005 0.005
5 6 1 4 3.000 3 1 2.000 2 4 2.000 1 2 1.000 4 2 2.000 3 5 1.000	1 1.000 1.000 1.000 0.000
3 1 1 1 1	2
3 3 1 2 1.250 1 3 1.300 2 3 1.000	2
3 2 1 2 1.000 1 3 0.005	2
4 2 1 2 1.250 1 4 1.251	2

Задача IV-B. Сломанный калькулятор

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

У калькулятора есть две ячейки памяти: содержимое первой из них всегда отображается на табло, вторая является буфером. В начальный момент времени на табло калькулятора отображается целое число X, а в буфере записано число 0. У калькулятора работают только две клавиши: «+»

и «=». При нажатии на «+» число, которое в данный момент отображено на табло, копируется в буфер. При нажатии на «=» число из буфера прибавляется к числу, отображенному на табло, и результат отображается на табло, число в буфере при этом не меняется.

Требуется за наименьшее количество нажатий клавиш на калькуляторе добиться того, чтобы на табло было отображено число Y.

Формат входных данных.

Входной файл содержит два целых числа: X и Y. Каждое из этих чисел по модулю не превышает 10^9.

Формат выходных данных.

В первую строку выходного файла выведите одно число — количество нажатий клавиш, которое потребуется для получения числа Y. Если из числа X получить число Y с помощью указанных операций невозможно, в выходной файл выведите одно число `-1`.

Примеры.

b.in	b.out	Пояснение (ответ дает следующая последовательность нажатий)
1 1	0	
-2 -6	3	+= =
1 8	6	+===+=
2 5	-1	

Задача IV-C. Операции с валютой

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Петя, изучая, как меняется курс рубля по отношению к доллару и евро, вывел закон, по которому происходят эти изменения (или думает, что вывел). По этому закону Петя рассчитал, каков будет курс рубля по отношению к доллару и евро в ближайшие N дней.

У Пети есть 100 рублей. В каждый из дней он может обменивать валюты друг на друга по текущему курсу без ограничения количества (при этом курс доллара по отношению к евро соответствует величине, которую можно получить, обменяв доллары на рубли, а потом эти рубли — на евро). Поскольку Петя будет оперировать не с наличной валютой, а со счетом в банке, то он может совершать операции обмена с любым (в том числе и нецелым) количеством единиц любой валюты.

Напишите программу, которая вычисляет, какое наибольшее количество рублей сможет получить Петя к исходу N-го дня.

Законы изменения курсов устроены так, что в течение указанного периода рублевый эквивалент той суммы, которая может оказаться у Пети, не превысит 10^8 рублей.

Формат входных данных.

Первая строка входного файла содержит одно число N ($1 \leqslant N \leqslant 5000$). В каждой из следующих N строк записано по 2 числа, вычисленных по Петиным законам для соответствующего дня, — сколько рублей будет стоить 1 доллар и сколько рублей будет стоить 1 евро. Все эти значения не меньше 0,01 и не больше 10000. Значения заданы точно и выражаются вещественными числами не более чем с двумя знаками после десятичной точки.

Формат выходных данных.

В выходной файл выведите искомую величину с точностью не менее двух знаков после десятичной точки.

Пример.

c.in	c.out
4 1 10 10 5.53 5.53 1.25 6 5	4000.00

Задача IV-D. Двухтуровая олимпиада

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Как известно, личная олимпиада по информатике проходит в два тура. На каждом из туров участники получают какие-то баллы, при этом итоговый балл определяется как сумма полученных баллов. Известны баллы, которые каждый участник получил на каждом из туров. Жюри хочет фальсифицировать итоги олимпиады так, чтобы победил «нужный» участник.

При этом жюри может делать следующие «подтасовки» (можно делать несколько «подтасовок» применительно как к одному и тому же, так и к разным турам):

- прибавить к результатам всех участников по одному из туров одно и то же положительное число;
- умножить результаты участников по одному из туров на некоторый коэффициент, больший 1.

При этом должна сохраниться правдоподобность результатов, которая заключается в том, что никто из участников не должен получить больше 100 баллов за каждый из туров.

Определите список участников, которые в результате таких фальсификаций могут оказаться победителями олимпиады (т. е. в сумме за два тура иметь не меньше баллов, чем каждый из остальных участников).

Формат входных данных.

Во входном файле записано сначала число участников N ($1 \leqslant N \leqslant 1000$), затем N пар чисел — результаты каждого участника за 1-й и за 2-й туры (результат участника за тур — это вещественное число от 0 до 100) не более чем с 3 знаками после десятичной точки.

Формат выходных данных.

В выходной файл выведите сначала количество участников, которые смогут стать победителями олимпиады, а затем в порядке возрастания их номера.

Пример.

d.in	d.out
4 45 90 70 80 0 0 75 75	2 2 4

Задача IV-E. Черно-белые палиндромы

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Дана полоса клетчатой бумаги длиной N клеток и шириной в 1 клетку, на которой некоторые клетки покрашены в черный цвет, а остальные — в белый. Такая полоса называется палиндромом, если последовательность черных и белых клеток при просмотре этой полосы слева направо оказывается такой же, как при просмотре справа налево.

Вам дана полоса длины N. Требуется разрезать ее на полоски, являющиеся палиндромами, так, чтобы количество получившихся полосок было строго меньше величины $\frac{2}{5}N + 3$.

Формат входных данных.

Первая строка входного файла содержит число N — длину исходной полосы (N — натуральное число, не превышающее 100000). Далее идет N чисел, описывающих раскраску полосы: 0 означает черную клетку, а 1 — белую.

Формат выходных данных.

В выходной файл выведите в порядке возрастания номера клеток исходной полосы, после которых нужно сделать разрезы.

Примеры.

e.in	e.out	Пояснение
6 0 1 0 1 1 0	3 5	Из исходной полосы мы получим 3 полосы-палиндрома, сделав разрезы после 3-й клетки (т. е. между 3-й и 4-й) и после 5-й (т. е. между 5-й и 6-й).
6 0 1 1 0 0 0	1 3	Данную полосу можно разрезать на 2 полосы-палиндрома, однако по условию не требуется искать решение с минимальным числом получившихся полосок — достаточно, чтобы число полосок удовлетворяло указанному в условии ограничению.
5 0 0 0 0 0		Исходная строка уже является палиндромом, поэтому можно ничего не разрезать.

Задача IV-F. Луч света в темном царстве

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Темное царство представляет собой лабиринт $N \times M$, некоторые клетки которого окружены зеркальными стенами, а остальные — пустые. Весь

лабиринт также окружен зеркальной стеной. В одной из пустых клеток лабиринта поставили светофор, который испускает лучи в 4 направлениях под 45 градусов относительно стен лабиринта. Требуется изобразить траекторию этих лучей.

Когда луч попадает в угол, образованный зеркальными стенами, дальше он идет так, как показано на рисунках (серым цветом показаны клетки, которые окружены зеркальными стенами). Аналогичным образом луч ведет себя, когда приходит на границу лабиринта.

Формат входных данных.

В первой строке входного файла записаны два натуральных числа N и M — число строк и столбцов в лабиринте (каждое из чисел не меньше 1 и не больше 100). В следующих N строках записано ровно по M символов в каждой — карта лабиринта. Символ «`*`» (звездочка) обозначает клетку, окруженную зеркальными стенами, «`.`» (точка) — пустую клетку, символ «`X`» (заглавная латинская буква X) — клетку, в которой расположен светофор (такая клетка ровно одна).

Формат выходных данных.

В выходной файл выведите N строк по M символов в каждой — изображение лабиринта с траекториями лучей. Здесь, как и раньше, «`*`» (звездочка) должна обозначать клетки, окруженные зеркальными стенами, «`.`» (точка) — пустые клетки, через которые лучи света не проходят, «`/`» (слеш) — клетки, через которые луч света проходит из левого нижнего угла в правый верхний (или обратно — из правого верхнего в левый нижний), «`\`» (обратный слеш) — клетки, через которые луч проходит из левого верхнего угла в правый нижний (или обратно), а символ «`X`» (заглавная латинская буква X) — клетки, через которые лучи проходят по обеим диагоналям.

Примеры.

f.in	f.out
5 6 ..*... ....* *X...* ....*.	./*./ \ /..X./ \./.X* *X./ /* / \X/*.

f.in	f.out
3 3 ... .X. ...	\./ .X. /.\

Задача IV-G. Распаковка строки

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Будем рассматривать только строки, состоящие из заглавных латинских букв. Например, рассмотрим строку AAAABCCCCCDDDD. Длина этой строки равна 14. Поскольку строка состоит только из латинских букв, повторяющиеся символы могут быть удалены и заменены числами, определяющими количество повторений. Таким образом, данная строка может быть представлена как 4AB5C4D. Длина такой строки 7. Описанный метод мы назовем *упаковкой* строки.

Напишите программу, которая берет упакованную строку и восстанавливает по ней исходную строку.

Формат входных данных.

Входной файл содержит одну упакованную строку. В строке могут встречаться только конструкции вида nA, где n — количество повторений символа (целое число от 2 до 99), а A — заглавная латинская буква, либо конструкции вида A, т. е. символ без числа, определяющего количество повторений. Максимальная длина строки не превышает 80.

Формат выходных данных.

В выходной файл выведите восстановленную строку. При этом строка должна быть разбита на строчки длиной ровно по 40 символов (за исключением последней, которая может содержать меньше 40 символов).

Примеры.

g.in	g.out
3A4B7D	AAABBBBDDDDDDD
22D7AC18FGD	DDDDDDDDDDDDDDDDDDDDDDAAAAAAACFFFFFFFFFF FFFFFFFFGD
95AB	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAB

g.in	g.out
40AB39A	AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Задача IV-H. Дремучий лес

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	8 мегабайт

Просека — это такая прямая линия, которая проходит через лес (т. е. деревья есть как с одной стороны от этой линии, так и с другой), и при этом она не проходит ни через одно из деревьев леса, а также не касается деревьев. Будем говорить, что лес является дремучим, если в нем нет ни одной просеки.

На плане леса все деревья изображаются кругами. Никакие два круга не пересекаются и не касаются друг друга. Требуется по этому плану определить, является ли лес дремучим.

Формат входных данных.

Во входном файле содержится сначала целое число N — количество деревьев ($1 \leqslant N \leqslant 200$). Затем идет N троек чисел, задающих деревья. Первые два числа задают координаты центра, а третье — радиус. Все данные задаются точно и выражаются вещественными числами не более чем с 2 знаками после десятичной точки, по модулю не превосходящими 1000.

Формат выходных данных.

В первой строке выходного файла должно содержаться сообщение `YES`, если лес является дремучим, и `NO` — если нет. Во втором случае вторая строка выходного файла должна содержать координаты двух точек, через которые проходит просека. Все координаты нужно выводить с восемью знаками после десятичной точки, модуль координаты не должен превышать 2000, и расстояние между выданными точками должно быть не меньше 100.

Примеры.

h.in	h.out
3 0 10 2 5 11 2 12.04 7 2	NO 2.50000000 0.00000000 2.50000000 100.00000000
3 0 0 1 2.05 0 1 1.02 -1.9 1	YES

Олимпиада V. Заочный тур 2004–2005 учебного года

Задача V-A. Автобусная экскурсия

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Оргкомитет Московской городской олимпиады решил организовать обзорную экскурсию по Москве для участников олимпиады. Для этого был заказан двухэтажный автобус (участников олимпиады достаточно много, и в обычный они не умещаются) высотой 437 сантиметров. На экскурсионном маршруте встречаются N мостов. Жюри и оргкомитет олимпиады очень обеспокоены тем, что высокий двухэтажный автобус может не проехать под одним из них. Им удалось выяснить точную высоту каждого из мостов. Автобус может проехать под мостом тогда и только тогда, когда высота моста превосходит высоту автобуса. Помогите организаторам узнать, можно ли провести эту экскурсию, а если нет, установить, под каким мостом автобус не сможет проехать.

Формат входных данных.

Во входном файле сначала содержится число N ($1 \leqslant N \leqslant 1000$). Далее идут N натуральных чисел, не превосходящих 10000, — высоты мостов в сантиметрах в том порядке, в котором они встречаются на пути автобуса.

Формат выходных данных.

В единственную строку выходного файла нужно вывести фразу «`No crash`», если экскурсию можно провести. Если же экскурсию провести нельзя, то нужно вывести сообщение «`Crash` k», где k — номер первого из мостов, под которым автобус не сможет проехать. Фразы выводить без кавычек, ровно с одним пробелом внутри.

Примеры.

a.in	a.out
1 763	No crash
3 763 245 113	Crash 2

a.in	a.out
1 437	Crash 1

Задача V-B. Сапер

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Мальчику Васе очень нравится известная игра «Сапер» («Minesweeper»).

В «Сапер» играет один человек. Игра идет на клетчатом поле (далее будем называть его картой) $N \times M$ (N строк, M столбцов). В K клетках поля стоят мины, в остальных клетках записано либо число от 1 до 8 — количество мин в соседних клетках, либо ничего не написано, если в соседних клетках мин нет. Клетки являются соседними, если они имеют хотя бы одну общую точку, в одной клетке не может стоять более одной мины. Изначально все клетки поля закрыты. Игрок за один ход может открыть какую-нибудь клетку. Если в открытой им клетке оказывается мина — он проигрывает, в противном случае игроку показывается число, которое стоит в этой клетке, и игра продолжается. Цель игры — открыть все клетки, в которых нет мин.

У Васи на компьютере есть эта игра, но ему кажется, что все карты, которые в ней есть, некрасивые и неинтересные. Поэтому он решил нарисовать свои. Однако фантазия у него богатая, а времени мало, и он хочет успеть нарисовать как можно больше карт. Поэтому он просто выбирает N, M и K и расставляет мины на поле, после чего все остальные клетки могут быть однозначно определены. Однако на определение остальных клеток он не хочет тратить свое драгоценное время. Помогите ему!

По заданным N, M, K и координатам мин восстановите полную карту.

Формат входных данных.

В первой строке входного файла содержатся числа: N, M и K ($1 \leqslant N \leqslant 200$, $1 \leqslant M \leqslant 200$, $0 \leqslant K \leqslant N \cdot M$). Далее идут K строк, в каждой из которых содержится по два числа, задающих координаты мин. Первое число в каждой строке задает номер строки клетки, где находится мина, второе число — номер столбца. Левая верхняя клетка поля имеет координаты (1, 1), правая нижняя — координаты (N, M).

Формат выходных данных.

Выходной файл должен содержать N строк по M символов — соответствующие строки карты. j-й символ i-й строки должен содержать символ «*» (звездочка), если в клетке (i, j) стоит мина, цифру от 1 до 8, если в этой клетке стоит соответствующее число, либо «.» (точка), если клетка (i, j) пустая.

Пример.

b.in	b.out
10 9 23 1 8 2 3 3 2 3 3 4 3 5 7 6 7 7 1 7 2 7 3 7 4 7 5 7 6 7 7 7 8 8 1 8 3 8 5 8 7 9 3 9 5 9 6 9 7	.111..1*1 13*2..111 1**3..... 13*2.111. .111.2*2. 233335*41 ********1 *6*7*8*41 13*4***2. .1122321.

Задача V-C. Робот К-79

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Петя написал программу движения робота К-79. Программа состоит из следующих команд:

- `S` — сделать шаг вперед;
- `L` — повернуться на 90° влево;
- `R` — повернуться на 90° вправо.

Напишите программу, которая по заданной программе для робота определит, сколько шагов он сделает, прежде чем впервые вернется на то место, на котором уже побывал до этого, либо установит, что этого не произойдет.

Формат входных данных.
Во входном файле записана одна строка из заглавных латинских букв `S`, `L`, `R`, описывающая программу для робота. Общее число команд в программе не превышает 200, при этом команд `S` не более 50.

Формат выходных данных.
В выходной файл выведите, сколько шагов будет сделано (т. е. выполнено команд `S`), прежде чем робот впервые окажется в том месте, через которое он уже проходил. Если такого не произойдет, выведите в выходной файл число `-1`.

Примеры.

c.in	c.out
SSLSLSLSSRSRS	5
LSSSS	-1

Задача V-D. Многочлен

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Васе задали несколько однотипных задач по математике: «найти значение многочлена». Он хочет написать программу, которая по заданному многочлену и значению x находила бы ответ. Напишите такую программу!

Формат входных данных.
В первой строке входного файла записан многочлен в виде суммы одночленов. Между одночленами находится знак «+» или «-». Перед первым одночленом может быть знак «-». Одночлен записывается как

`[<Коэффициент>*] x [^<Степень>]`

или

`<Коэффициент>`,

где `<Коэффициент>` — натуральное число, не превосходящее 100, `x` — символ переменной (всегда маленькая латинская буква x), `<Степень>` — натуральное число, не превосходящее 4. Параметры, взятые в квадратные скобки, могут быть опущены. Во второй строке записано одно целое число — значение `x`.

Формат выходных данных.
В выходной файл нужно записать одно число — значение данного многочлена при данном значении `x`.

Ограничения.
Все числа в исходном файле по модулю не превосходят 100. Количество одночленов не более 10 (могут быть одночлены одинаковой степени).

Примеры.

d.in	d.out
8*x+5 7	61
-2+x^1-3*x^2+x^2+100*x^3-2*x 0	-2

Задача V-E. Головоломка

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Петя разгадывает головоломку, которая устроена следующим образом. Дана квадратная таблица размера $N \times N$, в каждой клетке которой записана какая-нибудь латинская буква. Кроме того, дан список ключевых слов. Пете нужно, взяв очередное ключевое слово, найти его в таблице. Это значит найти в таблице все буквы этого слова, причем они должны быть расположены так, чтобы клетка, в которой находится каждая последующая буква слова, была соседней с клеткой, в которой записана предыдущая буква (клетки называются соседними, если они имеют общую сторону — т. е. соседствуют по вертикали или по горизонтали). Например, на рисунке ниже показано, как может быть расположено в таблице слово `olympiad`.

P	**O**	**L**	T	E
R	W	**Y**	**M**	S
O	**A**	**I**	**P**	T
B	**D**	A	N	R
L	E	M	E	S

Когда Петя находит слово, он вычеркивает его из таблицы. Использовать уже вычеркнутые буквы в других ключевых словах нельзя.

После того как найдены и вычеркнуты все ключевые слова, в таблице остаются еще несколько букв, из которых Петя должен составить слово, зашифрованное в головоломке.

Помогите Пете в решении этой головоломки, написав программу, которая по данной таблице и списку ключевых слов выпишет, из каких букв Петя должен сложить слово, т. е. какие буквы останутся в таблице после вычеркивания ключевых слов.

Формат входных данных.

В первой строке входного файла записаны два числа: N ($1 \leqslant N \leqslant 10$) и M ($0 \leqslant M \leqslant 200$). Следующие N строк по N заглавных латинских букв описывают ребус. Следующие далее M строк содержат слова. Слова состоят только из заглавных латинских букв, каждое слово не длиннее 200 символов. Гарантируется, что в таблице можно найти и вычеркнуть по описанным выше правилам все ключевые слова.

Формат выходных данных.

В единственную строку выходного файла выведите в любом порядке буквы, которые останутся в таблице.

Примеры.

e.in	e.out
5 3 POLTE RWYMS OAIPT BDANR LEMES OLYMPIAD PROBLEM TEST	ANSWER

e.in	e.out
3 2 ISQ ABC IQW I IS	ABCQQW

Задача V-F. Поиск прямоугольников

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

На поле $N \times M$ клеток (N строк и M столбцов) положили K прямоугольников один поверх другого в случайном порядке. Длины сторон прямоугольников выражаются целым числом клеток. Прямоугольники не выходят за границы поля. Границы прямоугольников совпадают с границами клеток поля.

0	2	2	2	2
0	2	2	2	2
1	1	2	2	2
1	1	0	0	0

Получившуюся ситуацию записали в таблицу чисел (каждой клетке поля соответствует клетка таблицы). Если клетка поля не закрыта прямоугольником, то в соответствующую клетку таблицы записали число 0. Если же клетка закрыта одним или несколькими прямоугольниками, то в соответствующую клетку таблицы записали число, соответствующее номеру самого верхнего прямоугольника, закрывающего эту клетку.

По содержимому таблицы требуется определить положение и размеры прямоугольников.

Гарантируется, что во входных данных содержится информация, которой достаточно для однозначного определения размеров прямоугольников.

Формат входных данных.

В первой строке входного файла записаны целые числа: N, M, K ($1 \leqslant N \leqslant 200$, $1 \leqslant M \leqslant 200$, $1 \leqslant K \leqslant 255$). Далее следует N строк по M чисел в каждой — содержимое таблицы. Все числа в таблице целые, находятся в диапазоне от 0 до K включительно.

Формат выходных данных.

В выходной файл необходимо выдать K строк. Каждая строка должна описывать соответствующий ее номеру прямоугольник четырьмя числами: R, C, H, W (R и C должны описывать координаты левого нижнего угла прямоугольника, а H и W — координаты правого верхнего угла). Числа должны разделяться пробелом.

Оси координат устроены следующим образом: начало координат находится в нижнем левом углу поля, а оси координат направлены вдоль сторон поля (ось Ox — вдоль нижней стороны, а ось Oy — вдоль левой стороны). Клетки поля имеют размер 1×1. Таким образом, координаты левого нижнего угла поля (0, 0), правого верхнего (M, N). Заметьте, что вы должны вывести координаты углов прямоугольников (как точек) в этой системе координат, а не координаты угловых клеток, покрытых прямоугольниками.

Пример.

f.in	f.out
4 5 2 0 2 2 2 2 0 2 2 2 2 1 1 2 2 2 1 1 0 0 0	0 0 2 2 1 1 5 4

Задача V-G. Разноцветные треугольники

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Выпуклый N-угольник разбит непересекающимися диагоналями на треугольники. (Многоугольник называется выпуклым, если любая его диагональ лежит внутри него.) Требуется покрасить каждую сторону и каждую проведенную диагональ в красный или синий цвет так, чтобы у каждого треугольника были стороны как красного, так и синего цвета.

Требуется привести любую из допустимых раскрасок.

Формат входных данных.

В первой строке записано одно число N ($4 \leqslant N \leqslant 100$) — количество вершин многоугольника.

Далее следуют $N - 3$ строки, в каждой из которых записана пара натуральных чисел — номера вершин, которые соединяет диагональ. Считается, что все вершины занумерованы последовательно натуральными числами от 1 до N.

Формат выходных данных.

В выходном файле должны быть $2N - 3$ строки. Каждая строка содержит 3 числа: номера вершин, которые соединяет данная сторона или диагональ, и цвет (1 — синий, 2 — красный), в который Вы красите данную сторону или диагональ.

Примеры.

g.in	g.out
4 1 3	1 2 1 2 3 1 3 4 1 4 1 1 1 3 2
6 1 3 3 5 5 1	1 2 1 2 3 1 3 4 1 3 5 2 4 5 1 5 6 2 5 1 1 6 1 2 1 3 2

Задача V-H. Побег с космической станции

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Представьте, что вы состоите на службе во внешней разведке Межгалактического Альянса Республиканских Сил (МАРС). Одному из агентов разведки крупно не повезло, и он был захвачен на засекреченной космической базе. К счастью, внешней разведке МАРС удалось заполучить план этой базы. И вот теперь вам поручено разработать план побега.

База представляет собой прямоугольник размером $N \times M$, со всех сторон окруженный стенами и состоящий из квадратных отсеков единичной площади. База снабжена K выходами, до одного из которых агенту необ-

ходимо добраться. В некоторых отсеках базы находятся стены. Ваш агент может перемещаться из отсека в любой из четырех соседних с ним, если в том отсеке, куда он хочет переместиться, нет стены.

Кроме того, база снабжена системой гипертуннелей, способных перемещать агента из одного отсека базы (вход в гипертуннель) в другой (выход из гипертуннеля). Когда агент находится в отсеке, где есть вход в гипертуннель, он может (но не обязан) им воспользоваться.

Начальное положение вашего агента известно. Вам необходимо найти кратчайший путь побега (т. е. путь, проходящий через минимальное количество отсеков).

Формат входных данных.

В первой строке входного файла записаны числа: N и M ($2 \leqslant N \leqslant 100$, $2 \leqslant M \leqslant 100$), задающие размеры базы. N — количество строк в плане базы, M — количество столбцов. Во второй строке записаны начальные координаты агента X_A, Y_A ($1 \leqslant X_A \leqslant N$, $1 \leqslant Y_A \leqslant M$). Первая координата задает номер строки, вторая — номер столбца. Строки нумеруются сверху вниз, столбцы — слева направо.

Далее следуют N строк по M чисел, задающих описание стен внутри базы: 1 соответствует стенке, 0 — ее отсутствию.

Далее в отдельной строке записано число H ($0 \leqslant H \leqslant 1000$) — количество гипертуннелей. В последующих H строках идут описания гипертуннелей. Каждый гипертуннель задается 4 числами: X_1, Y_1, X_2, Y_2 ($1 \leqslant X_1$, $X_2 \leqslant N$; $1 \leqslant Y_1$, $Y_2 \leqslant M$) — координатами входа и выхода гипертуннеля. Никакие два гипертуннеля не имеют общего входа.

После этого в отдельной строке следует число K ($1 \leqslant K \leqslant 10$) — количество выходов с базы. В последующих K строках идут описания выходов с базы. Каждый выход задается двумя координатами X и Y ($1 \leqslant X \leqslant N$; $1 \leqslant Y \leqslant M$).

Гарантируется, что начальные координаты агента не совпадают ни с одним из выходов и он не стоит в отсеке, занятом стеной. Никакие входы и выходы гипертуннелей, а также выходы с базы не находятся в отсеках, занятых стенами. Никакой вход в гипертуннель не совпадает с выходом с базы.

Формат выходных данных.

Если побег невозможен, выведите единственную строку с надписью «`Impossible`». В противном случае в первой строке выведите число L — количество отсеков в кратчайшем пути побега. В последующих L строках последовательно выведите координаты отсеков кратчайшего пути побега. Если решений несколько, то выведите любое из них.

Пример.

h.in	h.out
4 5 2 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 2 1 4 3 1 1 4 1 2 4	4 2 1 3 1 1 4 2 4

Задача V-I. Неподвижная точка карты

Имя входного файла:	i.in
Имя выходного файла:	i.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Первый учебный день шестиклассника Пети начался с урока географии. Учитель объяснял классу, что перед тем как изучать просторы нашей Родины, нужно научиться пользоваться географическими картами. Было также упомянуто и о том, что такое масштаб карты. В качестве домашней работы Пете и его одноклассникам задали нарисовать план (карту) своей комнаты, соблюдая масштабирование. Петю очень заинтересовало задание учителя, и поэтому как только он пришел из школы домой, он принялся рисовать план. Это занятие было очень увлекательным, но вскоре с работы пришла Петина мама, сказала, что здоровье превыше всего, и позвала его обедать. Во время обеда она по пути на кухню зашла в Петину комнату и решила, что ее надо проветрить. Для этого она открыла окно, перед которым стоял Петин стол.

После обеда Петя вернулся в комнату и обнаружил, что его творение сдуло ветром на пол. Сначала он обеспокоился тем, в порядке ли рисунок, но удостоверившись, что все нормально, не стал спешить и поднимать план с пола. Он вспомнил слова учителя географии, который в конце урока поведал им некое нетривиальное утверждение и предложил любопытным проверить его на досуге.

Утверждение гласило: «если взять две карты одной и той же области, сделанные с разным масштабом, и расположить меньшую поверх большей так, что меньшая карта окажется строго внутри большей, то можно найти такую точку (она называется «неподвижная точка»), что то, что изображено в этой точке на обеих картах будет соответствовать одной и той же точке местности». Петя заметил, что пол комнаты можно считать картой комнаты (масштаб 1:1). Он решил найти неподвижную точку для лежащего на полу нарисованного им плана и пола. Но Петя не сумел сделать это самостоятельно, поэтому он обратился к вам за помощью.

Формат входных данных.

Комната Пети и ее план имеют форму прямоугольника. Первая строка входного файла содержит два вещественных числа: ширину X и длину Y комнаты Пети ($1 \leqslant X \leqslant 1000$, $1 \leqslant Y \leqslant 1000$). Декартова система координат выбрана так, что углы комнаты расположены в точках с координатами (0, 0), (X, 0), (X, Y), (0, Y).

Вторая строка содержит восемь вещественных чисел, описывающих положение углов плана комнаты в той же самой системе координат. Сначала задаются координаты того угла плана, который соответствует углу комнаты с координатами (0, 0), затем (X, 0), (X, Y) и наконец, (0, Y). Гарантируется, что входные данные корректны, т. е. план является прямоугольником, линейные размеры плана находятся в полном соответствии с линейными размерами комнаты, план не выходит за границы комнаты.

Все числа во входном файле вещественные, заданы с точностью 5 знаков после десятичной точки. План выполнен в масштабе не менее 0,0001 и не более 1. Масштаб не может быть равен 1. Карта расположена лицевой стороной вверх.

Формат выходных данных.

В первую строку выходного файла выведите два вещественных числа — координаты неподвижной точки плана и пола. Ответ нужно выдать с 3 знаками после десятичной точки.

Пример.

i.in
10.00000 5.00000 3.00000 2.50000 1.00000 2.50000 1.00000 1.50000 3.00000 1.50000
i.out
2.500 2.083

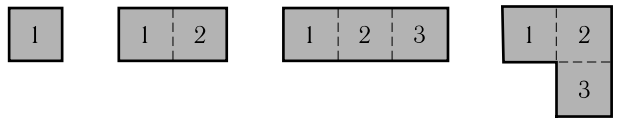
Задача V-J. Узор

Имя входного файла:	j.in
Имя выходного файла:	j.out
Максимальное время работы на одном тесте:	5 секунд
Максимальный объем используемой памяти:	16 мегабайт

В комнате решили сделать паркетный пол. Причем есть задумка выложить на полу некоторый узор.

Плитки паркета, которыми выкладывается пол комнаты, состоят из квадратиков 1×1, каждый из которых может быть либо белым, либо черным. В свою очередь, комната имеет размеры $N \times M$. На плане комнаты указано, какой квадрат комнаты какого цвета должен быть.

Существует несколько форм паркетных плиток:



Форма 1 Форма 2 Форма 3 Форма 4

Квадратики одной паркетной плитки могут быть окрашены по-разному. Может существовать несколько типов плиток одинаковой формы, но окрашенных по-разному. Плитки разных типов могут иметь разную стоимость. Количество плиток каждого типа не ограничено. Плитки разрешается как угодно поворачивать (на углы, кратные 90 градусам). Не разрешается разламывать плитки, а также класть их лицевой стороной вниз.

Изначально какая-то часть пола может уже быть выложена плиткой.

Требуется определить минимальную стоимость плитки, необходимой для того чтобы замостить оставшуюся часть комнаты.

Формат входных данных.

В первой строке входного файла записаны три числа: N, M (размеры комнаты) и K (количество доступных видов плитки). $1 \leqslant N \leqslant 8$, $1 \leqslant M \leqslant 8$, $1 \leqslant K \leqslant 10$. Далее идет описание желаемой раскраски пола. Описание представляет собой N строчек по M чисел в каждой, где 0 обозначает белый цвет, 1 — черный, 2 — то, что квадрат уже выложен плиткой. В последних K строчках находятся описания доступных типов плитки в следующем формате:

<форма> <стоимость> <окраска>

<форма> — это число от 1 до 4, описывающее форму плитки (см. рисунок выше).

<стоимость> — это натуральное число, не превосходящее 10000, задающее стоимость одной плитки такого типа.

<окраска> — это от одного до трех чисел 0 или 1. Количество чисел совпадает с количеством квадратиков, из которых состоит плитка. Числа задают цвета квадратиков плитки в том порядке, в каком квадратики пронумерованы на рисунке.

Формат выходных данных.

В выходной файл выведите единственное число — минимальную стоимость укладки или -1, если требуемым образом уложить плитку невозможно.

Пример.

j.in	j.out
4 3 3 2 2 2 2 0 0 2 1 2 2 2 2 2 10 0 0 1 5 1 4 6 0 0 1	15

Задача V-K. Склад

Имя входного файла:	k.in
Имя выходного файла:	k.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

Склад конторы MacroHard представляет собой прямоугольную комнату размером $N \times M$. На полу склада нарисована разметка, состоящая из линий, параллельных стенам склада, которые разбивают его на $N \times M$ квадратов 1×1.

Фирма выпускает высокотехнологичное оборудование, используемое в самых различных областях жизнедеятельности. Исторически сложилось так, что все изделия, выпускаемые этой компанией, имеют форму равнобедренного прямоугольного треугольника. При этом ассортимент изделий столь велик, что бывают изделия практически любых размеров.

Размещать изделия на складе разрешается только так, чтобы хотя бы одна сторона изделия была параллельна какой-то из стен склада, и вдобавок, все углы изделия находились в точках пересечения линий разметки

склада. Рисунки 1, 2, 3 иллюстрируют неправильное положение изделий, а 4, 5 — правильное.

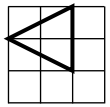


Рис. 1

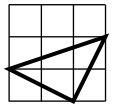


Рис. 2

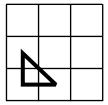


Рис. 3

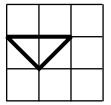


Рис. 4

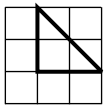


Рис. 5

Руководство фирмы узнало, что склад планирует посетить Комиссия по неэффективному использованию складских помещений. Чтобы избежать выплаты крупного штрафа, фирма решила в срочном порядке поместить на склад изделия так, чтобы на складе не осталось свободного места. При этом было решено, что продукция, которая уже находится на складе, перемещаться не будет.

Напишите программу, которая определит, какое минимальное количество изделий нужно добавить на склад, чтобы на нем не осталось свободного места.

Формат входных данных.

В первой строке входного файла записаны три целых числа: N, M (размеры склада) и K (количество изделий, которые уже находятся на складе). Следующие K строк содержат по 6 целых чисел — координаты углов соответствующего изделия. Система координат введена так, что оси координат параллельны стенам склада и при этом один из углов склада имеет координаты (0, 0), а противоположный — (N, M).

Ограничения.

$1 \leqslant N \leqslant 4$, $1 \leqslant M \leqslant 4$.

Формат выходных данных.

Первая строка выходного файла должна содержать одно число T — минимальное количество изделий, которые необходимо добавить, чтобы заполнить склад. Каждая из следующих T строк должна содержать по 6 чисел — координаты углов изделий.

Пример.

k.in	k.out
3 3 1 0 1 1 2 2 1	5 0 1 1 2 0 3 0 3 3 3 3 0 1 0 2 1 3 0 0 1 2 1 1 0 0 0 1 0 0 1

Задача V-L. Кубическая гостиница

Имя входного файла:	l.in
Имя выходного файла:	l.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	16 мегабайт

В связи с проведением межпланетного шашечного турнира было принято решение о строительстве орбитальной гостиницы. Она должна была представлять собой большой куб из $N \times N \times N$ блоков — маленьких кубиков $1 \times 1 \times 1$, и каждый блок должен был быть окрашен снаружи со всех сторон в какой-то один цвет. При этом некоторые блоки могли быть покрашены в один и тот же цвет.

Через год были сделаны фотографии гостиницы с каждой из 6 сторон: спереди, слева, сзади, справа, сверху, снизу. За год эксплуатации могло случиться так, что из-за непрочного крепления некоторые блоки, из которых была построена гостиница, оторвались и улетели в открытый космос. Комиссия по восстановлению гостиницы хочет по сделанным снимкам установить максимально возможное количество оставшихся блоков.

Итак, вам необходимо по видам гостиницы (куба $N \times N \times N$, из которого, возможно, выкинуты некоторые кубики $1 \times 1 \times 1$) с 6 сторон определить, из какого максимального количества блоков $1 \times 1 \times 1$ она может состоять. Может оказаться так, что гостиница состоит из двух или более не связанных между собой частей.

Формат входных данных.

В первой строке входного файла находится число N — размер гостиницы ($1 \leqslant N \leqslant 10$). В следующих N строках записаны виды гостиницы с 6 сторон (в следующем порядке: спереди, слева, сзади, справа, сверху, снизу). Каждый такой вид представляет собой таблицу $N \times N$, в которой различными заглавными латинскими буквами обозначены различные цвета, а символом «.» (точка) — то, что в этом месте можно будет смотреть прямо сквозь гостиницу. Два последовательных вида отделяются друг от друга ровно одним пробелом в каждой из N строк.

Нижняя граница вида сверху соответствует верхней границе вида спереди, а верхняя граница вида снизу — нижней границе вида спереди. Для видов спереди, сзади и с боков верх и низ вида соответствуют верху и низу гостиницы.

Входные данные корректны, т. е. во входном файле описано состояние, которое может получиться.

Формат выходных данных.

Выведите в выходной файл одно число — искомое максимальное количество оставшихся блоков в гостинице.

Примеры.

l.in	l.out
3 .R. YYR .Y. RYY .Y. .R. GRB YGR BYG RBY GYB GRB .R. YRR .Y. RRY .R. .Y.	11
2 ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ ZZ	8

Олимпиада VI. Личная олимпиада 2004–2005 учебного года

Задача VI-A. Праздники

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	80 баллов

Парламент некоторой страны принял новый закон о праздничных днях. Согласно этому закону первые K дней года, а также 23 февраля (день 2-го тура олимпиады по информатике) и 8 марта объявляются праздничными, а все остальные праздники отменяются. При этом все выходные (суббота и воскресенье), попавшие на праздничные дни, переносятся на следующие за этими праздниками рабочие дни.

В зависимости от того, на какой день недели приходится 1 января, количество нерабочих дней, которые идут подряд, может меняться.

Требуется определить, какое наибольшее количество нерабочих дней может идти подряд.

Формат входных данных.

Во входном файле записано единственное число K ($1 \leqslant K \leqslant 50$).

Формат выходных данных.

В выходной файл требуется записать единственное число — наибольшее количество нерабочих дней, идущих подряд.

Примеры.

a.in	a.out
2	4
10	16

Задача VI-B. Раскраска плиток

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	100 баллов

После того как к удивлению тетушки Полли ее забор был покрашен, она поручила Тому Сойеру обновить краску на плитках, которыми был вымощен их квадратный двор. Двор был покрыт $N \times N$ одинаковыми квадратными плитками, каждая из которых когда-то давно была покрашена в один из K цветов ($K < N$). Краска на плитках потускнела, и Тому Сойеру поручили их покрасить, на этот раз в один любой цвет (из тех же K цветов). Покрасить нужно все плитки, в том числе и те, которые уже были покрашены в этот цвет раньше.

Окунув кисть в ведро с краской один раз, можно перекрасить один горизонтальный или вертикальный ряд плиток. Чтобы разнообразить свою работу, Том придумал, что ряд плиток можно красить только цветом, которым на данный момент уже покрашены (старой или новой краской) по крайней мере две плитки выбранного ряда (вертикального или горизонтального). За один раз Том собирается красить допустимым цветом весь ряд целиком, независимо от того, были ли уже перекрашены какие-либо его плитки ранее. Помогите Тому определить, какое минимальное число раз ему придется обмакнуть кисть, чтобы перекрасить все плитки, следуя придуманным правилам, и в какой цвет окажутся окрашены все плитки.

Формат входных данных.

В первой строке входного файла записаны через пробел два числа: N — количество плиток в одном ряду ($1 < N \leqslant 200$) и K ($1 \leqslant K < N$). В каждой из следующих N строк записаны N натуральных чисел, обозначающих номера цветов красок, в которые когда-то были выкрашены соответствующие плитки данного горизонтального ряда. Номера цветов — натуральные числа в диапазоне от 1 до K.

Формат выходных данных.

В выходной файл выведите два числа: L — какое минимальное число раз придется окунать кисть в ведро с краской и номер краски C, в которую в результате окажутся перекрашены все плитки двора. Если таких красок может быть несколько, то выведите номер любой из них.

Если перекрасить все плитки, следуя придуманным Томом правилам, нельзя, выведите два раза число 0.

Примеры.

b.in	b.out
3 2 1 2 1 2 1 1 1 2 2	4 1

b.in	b.out
2 1 1 1 1 1	2 1

Задача VI-C. Маскарад

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	120 баллов

По случаю введения больших новогодних каникул устраивается великий праздничный бал-маскарад. До праздника остались считанные дни, поэтому срочно нужны костюмы для участников. Для пошивки костюмов требуется L метров ткани. Ткань продается в N магазинах, в которых предоставляются скидки оптовым покупателям. В магазинах можно купить только целое число метров ткани. Реклама магазина номер i гласит: «Мы с радостью продадим Вам метр ткани за P_i бурлей, однако, если Вы купите не менее R_i метров, то получите прекрасную скидку — каждый купленный метр обойдется Вам всего в Q_i бурлей». Чтобы воплотить в жизнь лозунг «экономика страны должна быть экономной», правительство решило потратить на закупку ткани для костюмов минимальное количество бурлей из государственной казны. При этом ткани можно купить больше чем нужно, если так окажется дешевле. Ответственный за покупку ткани позвонил в каждый магазин и узнал, что:

1) реклама каждого магазина содержит правдивую информацию о ценах и скидках;

2) магазин номер i готов продать ему не более F_i метров ткани.

Ответственный за покупку очень устал от проделанной работы и поэтому поставленную перед ним задачу «закупить ткань за минимальные деньги» переложил на своих помощников. Напишите программу, которая определит, сколько ткани нужно купить в каждом из магазинов так, чтобы суммарные затраты были минимальны.

Формат входных данных.

В первой строке входного файла записаны два целых числа: N и L ($1 \leqslant N \leqslant 100$, $0 \leqslant L \leqslant 100$). В каждой из последующих N строк находится описание магазина номер i — четыре целых числа: P_i, R_i, Q_i, F_i ($1 \leqslant Q_i \leqslant P_i \leqslant 1000$, $1 \leqslant R_i \leqslant 100$, $0 \leqslant F_i \leqslant 100$).

Формат выходных данных.

Первая строка выходного файла должна содержать единственное число — минимальное необходимое количество бурлей.

Во второй строке выведите N чисел, разделенных пробелами, где i-е число определяет количество метров ткани, которое нужно купить в i-м магазине. Если в i-м магазине ткань покупаться не будет, то на i-м месте должно стоять число 0. Если вариантов покупки несколько, выведите любой из них.

Если ткани в магазинах недостаточно для пошивки костюмов, выходной файл должен содержать единственное число -1.

Примеры.

c.in	c.out
2 14 7 9 6 10 7 8 6 10	88 10 4
1 20 1 1 1 1	-1

Задача VI-D. Билетики

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	80 баллов

В процессе установки турникетов в автобусах, разработчики столкнулись с проблемой проверки подлинности билета. Для ее решения был придуман следующий способ защиты от подделок.

Информация, записанная на билете, кодируется K числами (0 или 1). При этом непосредственно на билете записывается последовательность из N чисел ($N \geqslant K$) так, что числа, записанные на расстоянии K, совпадают. Таким образом, для проверки подлинности билета достаточно проверить, что все числа на расстоянии K совпадают. К сожалению, при считывании информации с билета иногда могут происходить ошибки — считается, что одно из чисел может исказиться (т. е. число 0 заменится на 1, или 1 — на 0). Такой билет все равно нужно считать подлинным. Во всех остальных случаях билет считается поддельным.

Напишите программу, которая по информации, считанной с билета, устанавливает его подлинность и указывает, при считывании какого из чисел могла произойти ошибка.

Формат входных данных.

В первой строке входного файла записаны числа N и K ($1 \leqslant N \leqslant 50000$, $1 \leqslant K \leqslant 1000$, $K \leqslant N$). Во второй строке записано N чисел, каждое из которых является 0 или 1 — информация, считанная с билета.

Формат выходных данных.

В первой строке выходного файла должно быть записано одно из двух сообщений — `OK` или `FAIL` (первое сообщение означает, что билет признан подлинным, второе — поддельным). В случае, если билет подлинный, во второй строке выведите `0`, если все числа были считаны правильно, или номер числа, в котором при считывании произошла ошибка. Если возможных ответов несколько, выведите любой из них (в частности, если для признания билета подлинным можно считать, что ошибок при считывании не было, а можно считать, что была ошибка в одном из чисел — правильным является любой из вариантов ответа).

Примеры.

d.in	d.out
6 2 1 0 1 0 1 0	OK 0
6 2 1 1 1 0 1 0	OK 2
6 2 1 1 1 0 0 0	FAIL

Задача VI-E. Сплоченная команда

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	80 баллов

Как показывает опыт, для создания успешной футбольной команды важны не только умения отдельных ее участников, но и сплоченность команды в целом. Характеристикой умения игрока является показатель его профессионализма (ПП). Команда является сплоченной, если ПП каждого из игроков не превосходит суммы ПП любых двух других (в частности, любая команда из одного или двух игроков является сплоченной). Перед тренерским составом молодежной сборной Москвы была поставлена задача сформировать сплоченную сборную с максимальной суммой ПП игроков (ограничений на количество игроков в команде нет).

Ваша задача состоит в том, чтобы помочь сделать правильный выбор из N человек, для каждого из которых известен его ПП.

Формат входных данных.

В первой строке входного файла записано целое число N ($0 \leqslant N \leqslant 30000$). В последующих N строках записано по одному целому числу P_i, представляющему собой ПП соответствующего игрока ($0 \leqslant P_i \leqslant 60000$).

Формат выходных данных.

В первой строке через пробел выведите число игроков, отобранных в команду, и их суммарный ПП. В последующих строках выведите номера игроков, вошедших в команду, в произвольном порядке — по одному числу в строке. Нумерация игроков должна соответствовать порядку перечисления игроков во входном файле. Если ответов несколько, выведите любой из них.

Примеры.

e.in	e.out
4 1 5 3 3	3 11 2 3 4
5 100 20 20 20	2 120 1 2

Задача VI-F. Сократи векторы

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	100 баллов

На плоскости задано N векторов — направленных отрезков, для каждого из которых известны координаты начала и конца (вектор, у которого начало и конец совпадают, называется *нуль-вектором*, можно считать, что нуль-вектор лежит на любой прямой, которая через него проходит). Введем следующие три операции над направленными отрезками на плоскости.

1. Направленные отрезки ненулевой длины, лежащие **на пересекающихся прямых**, можно заменить на их сумму, причем единственным образом. В этом случае отрезки переносятся вдоль своих прямых так, чтобы их начала совпадали с точкой пересечения прямых, и складываются по правилу сложения векторов (правилу параллелограмма, при этом началом результирующего вектора является точка пересечения прямых), см. рис. 1.

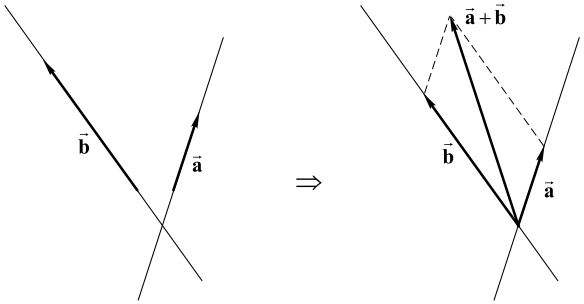


Рис. 1

2. Направленные отрезки, лежащие **на одной прямой**, также можно заменить на их сумму. Для этого конец одного отрезка (любого) нужно перенести в начало другого и сложить отрезки по правилу сложения векторов на прямой, см. рис. 2.

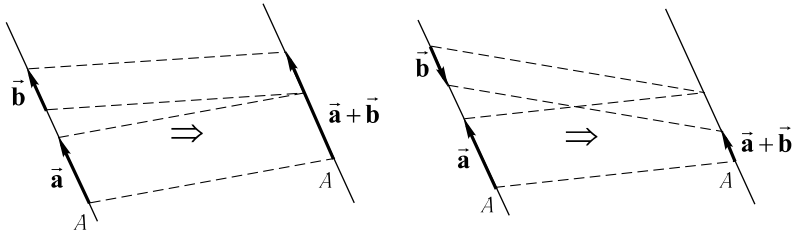


Рис. 2

Это правило применимо и в случае, когда один из векторов (или даже оба) являются нуль-векторами.

Заметим, что если складываемые векторы противоположно направлены и имеют одну и ту же длину, то результатом их сложения является нуль-вектор.

3. В любой точке плоскости можно *породить* два противоположно направленных отрезка равной (в том числе и нулевой) длины, см. рис. 3.

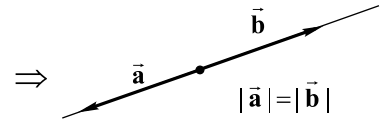


Рис. 3

Будем говорить, что две системы векторов *эквивалентны*, если от одной системы можно перейти к другой с помощью конечной последовательности перечисленных выше операций.

Требуется получить *любую* систему векторов, эквивалентную заданной, состоящую из как можно меньшего числа векторов.

Формат входных данных.

В первой строке входного файла записано число N — количество заданных векторов ($1 < N \leqslant 1000$). В каждой из следующих N строк через пробел записаны четыре числа, обозначающие координаты начала и конца каждого из векторов соответственно. Все координаты — целые числа, по модулю не превосходящие 1000.

Формат выходных данных.

В первой строке выходного файла следует записать число M — количество векторов в полученной системе ($1 \leqslant M \leqslant N$). В каждой из следующих M строк через пробел должны находиться четыре числа, обозначающие координаты начала и конца каждого из векторов соответственно. Все координаты — вещественные числа, записанные с 6 цифрами после точки.

Примеры.

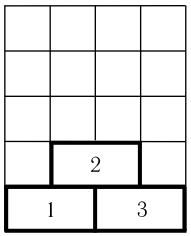
f.in	f.out
3 1 1 1 3 3 3 3 1 5 1 7 1	1 3.000000 3.000000 5.000000 3.000000
2 2 4 5 10 -2 -4 -5 -10	1 2.000000 4.000000 2.000000 4.000000

Задача VI-G. Strategy tetris

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта
Максимальная оценка за задачу:	100 баллов

Как и в обычном тетрисе, поле в игре Strategy Tetris представляет собой «стакан» шириной в W клеток ($1 \leqslant W \leqslant 10^9$) и бесконечной высоты. В этот стакан падают сверху N фигурок ($1 \leqslant N \leqslant 100000$). i-я фигурка представляет собой прямоугольник шириной в W_i клеток и высотой в одну клетку; самая левая клетка фигурки имеет абсциссу a_i ($1 \leqslant a_i \leqslant W - W_i + 1$). Фигурки падают по обычным правилам: если при падении фигурка хотя бы одной своей клеткой ложится на какую-либо уже упавшую фигурку, то ее движение прекращается.

В отличие от обычного тетриса игрок не имеет возможности вращать фигурки или смещать их по горизонтали в процессе падения — еще бы, это пришлось бы делать быстро и не было бы времени серьезно подумать над стратегией. Единственное, что он может, — это выбрать порядок, в котором эти N фигурок упадут в стакан (каждая по одному разу). Ваша задача — помочь ему выбрать такой порядок, при котором высота образовавшейся в результате падения конструкции была бы как можно меньше. (В отличие от обычного тетриса целиком заполненная фигурками горизонталь никуда не исчезает).



На рисунке ниже приведено заполнение стакана фигурками из примера входных данных (порядок заполнения соответствует выходному файлу, приведенному в примере).

Формат входных данных.

В первой строке входного файла записаны числа: N и W, а в последующих N строках — пары чисел a_i и W_i.

Формат выходных данных.

Выведите в выходной файл минимальную возможную высоту конструкции, а затем последовательность номеров фигурок, к этой высоте приводящую. Фигурки нумеруются натуральными числами от 1 до N в том порядке, в котором они указаны во входных данных. Если возможных вариантов несколько, выведите любой из них.

Пример.

g.in	g.out
3 4 1 2 2 2 3 2	2 1 3 2

Олимпиада VII. Командная олимпиада 2004–2005 учебного года

Задача VII-A. Москва-сортировочная

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Ежедневно диспетчеру железнодорожной станции «Москва-Сортировочная» приходится переставлять вагоны во многих поездах, чтобы они шли в заданном порядке. Для этого диспетчер может расцепить пришедший на станцию состав в произвольных местах и переставить образовавшиеся сцепки из одного или нескольких вагонов в произвольном порядке. Порядок вагонов в одной сцепке менять нельзя, также нельзя развернуть всю сцепку так, чтобы последний вагон в сцепке оказался первым в ней.

Диспетчер просит вашей помощи в определении того, какое минимальное число соединений между вагонами необходимо расцепить, чтобы переставить вагоны в составе в требуемом порядке.

Формат входных данных.

В первой строке входного файла содержится целое число N ($1 \leqslant N \leqslant 100$). Во второй строке содержится перестановка натуральных чисел от 1 до N (т. е. все натуральные числа от 1 до N в некотором порядке). Числа разделяются одним пробелом. Эта перестановка задает номера вагонов в приходящем на станцию составе. Требуется, чтобы в уходящем со станции составе вагоны шли в порядке их номеров.

Формат выходных данных.

Программа должна записать в выходной файл единственное целое число, равное минимальному количеству соединений между вагонами, которые нужно расцепить в данном составе, чтобы их можно было переставить по порядку.

Примеры.

a.in	a.out
4 3 1 2 4	2
5 5 4 3 2 1	4

a.in	a.out
2 1 2	0

Задача VII-B. Кафе

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Около Петиного университета недавно открылось новое кафе, в котором действует следующая система скидок: при каждой покупке более чем на 100 рублей покупатель получает купон, дающий право на один бесплатный обед (при покупке на сумму 100 рублей и меньше такой купон покупатель не получает).

Однажды Пете на глаза попался прейскурант на ближайшие N дней. Внимательно его изучив, он решил, что будет обедать в этом кафе все N дней, причем каждый день он будет покупать в кафе ровно один обед. Однако стипендия у Пети небольшая, и поэтому он хочет по максимуму использовать предоставляемую систему скидок так, чтобы его суммарные затраты были минимальны. Требуется найти минимально возможную суммарную стоимость обедов и номера дней, в которые Пете следует воспользоваться купонами.

Формат входных данных.

В первой строке входного файла записано целое число N ($0 \leqslant N \leqslant 100$). В каждой из последующих N строк записано одно целое число, обозначающее стоимость обеда в рублях на соответствующий день. Стоимость — неотрицательное целое число, не превосходящее 300.

Формат выходных данных.

В первой строке выдайте минимально возможную суммарную стоимость обедов. Во второй строке выдайте два числа: K_1 и K_2 — количество купонов, которые останутся неиспользованными у Пети после этих N дней и количество использованных им купонов соответственно.

В последующих K_2 строках выдайте в порядке возрастания номера дней, когда Пете следует воспользоваться купонами. Если существует несколько решений с минимальной суммарной стоимостью, то выдайте то из них, в котором значение K_1 максимально (на случай, если Петя когда-нибудь еще решит заглянуть в это кафе). Если таких решений несколько, выведите любое из них.

Пример.

b.in	b.out
5 35 40 101 59 63	235 0 1 5

Задача VII-C. EuroEnglish

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Европейская комиссия планирует принять решение о том, что официальным языком Евросоюза станет английский. Был также разработан план упрощения английской письменности, который планируется реализовать за четыре года.

Первоочередной задачей будет избавление от буквы `c`, которая в сочетаниях `ci` и `ce` будет заменяться на `s`, в сочетании `ck` — опускаться, а в остальных случаях заменяться на `k`. При этом все замены будут производиться в строгом порядке слева направо. Например, в слове `success` сначала первая из двух букв `c` заменится на `k`, а затем вторая — на `s`, т. е. получится `suksess`. А слово `cck` превратится в `kk`.

На второй год из английских слов изымут все удвоенные буквы: `ee` изменят на `i`, `oo` — на `u`, а в остальных комбинациях будут просто писать одну букву вместо двух одинаковых. Такие замены также будут делать строго в порядке слева направо. Так, слово `ooo` превратится в `uo`, а `oou` — просто в `u` (в нем сначала `oo` заменится на `u`, а затем `uu` — на `u`), слово `iee` превратится в `i` (в нем сначала `ee` заменится на `i`, а затем `ii` — на `i`).

На третий год на конце слова станут опускать букву `e`, если эта буква не является единственной буквой в слове.

Наконец, завершением реформы станет отмена артиклей (в английском языке три артикля: `a`, `an` и `the`). При этом удаляться эти артикли будут только тогда, когда они в исходном тексте были словами `a`, `an`, `the`. Например, текст `the table` после реформ первых трех лет превратится в `th tabl`, а после реформы четвертого года — просто в `tabl`. А слово `aaaaa` после реформы первых лет станет словом `a`, но поскольку изначально оно не было словом `a` (артиклем), то оно в итоге так и останется словом `a`.

Напишите программу, которая будет переводить классический английский текст на Евроинглиш.

Формат входных данных.

Во входном файле записана одна строка текста, состоящая не более чем из 200 символов: латинских строчных и заглавных букв, пробелов и знаков препинания (в тексте могут встречаться: точка, запятая, вопросительный и восклицательный знаки, двоеточие, тире, точка с запятой, открывающая и закрывающая скобки, апострофы, кавычки). Заглавные буквы могут встречаться только в начале слова. Нигде подряд не могут стоять два пробела. В начале и в конце строки не может стоять пробел. Слова отделяются друг от друга пробелами и (или) знаками препинания.

Формат выходных данных.

В выходной файл нужно выдать преобразованную строку с учетом следующих требований:

- начинаться с заглавной буквы должны те и только те слова, которые начинались с заглавной буквы в исходном тексте;
- не должно встречаться двух пробелов подряд;
- пробелы между словами и знаками препинания должны остаться там и только там, где они были в исходной строке, в начале и в конце строки пробелов быть не должно.

Примеры.

c.in	c.out
cacao and coffee	kakao and kofi
Cinderella! Where Is The Dress???	Sinderela! Wher Is Dres???
'A' is a letter	'' is leter
!!!Hello!!!A-the-"word"	!!!Helo!!!--"word"
Aaaa then the ckckck	A then k
"A"-the an	""-
A the an	
success	sukses

Задача VII-D. D++

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Мальчик Кирилл недавно придумал свой язык программирования. Он назвал его D++.

В D++ существует только один массив и 26 переменных. Переменные называются маленькими латинскими буквами от `a` до `z`. Есть 4 различных варианта операции присваивания (здесь и далее *переменная* — это маленькая латинская буква, задающая имя переменной, а *индекс* — натуральное число, задающее индекс элемента в массиве):

$$\textit{переменная}1 <- \textit{переменная}2,$$

т. е. значение переменной *переменная*1 становится равным значению переменной *переменная*2;

$$\textit{переменная} <- \textit{индекс},$$

т. е. значение переменной *переменная* становится равным значению элемента массива с индексом *индекс*;

$$\textit{индекс} <- \textit{переменная},$$

т. е. значение элемента массива с индексом *индекс* становится равным значению переменной *переменная*;

$$\textit{индекс}1 <- \textit{индекс}2,$$

т. е. значение элемента массива с индексом *индекс*1 становится равным значению элемента массива с индексом *индекс*2.

Других операций в языке D++ нет. В одной строке программы на D++ может быть записана только одна операция. В программе не может быть пустых строк, но могут быть незначащие пробелы.

Дана последовательность из N натуральных чисел. Все числа различны и не превосходят N. Будем считать, что эта последовательность записана в массиве языка D++ начиная с элемента под номером 1. Требуется написать программу на D++, которая упорядочит массив за наименьшее количество операций.

После выполнения программы последовательность также должна быть записана в массиве начиная с элемента под номером 1. Программа не должна использовать элементы массива с номерами меньше 1, а также с номерами больше N.

Формат входных данных.

В первой строке входного файла содержится единственное число N ($1 \leqslant N \leqslant 10000$). Во второй строке содержится N натуральных чисел — последовательность, записанная в массиве.

Формат выходных данных.

Выходной файл должен содержать программу на D++, которая сортирует данную последовательность, выполняя наименьшее число операций присваивания.

Пример.

d.in	d.out
2 2 1	x<-1 1<-2 2<-x

Задача VII-E. Скобки

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Назовем строку S правильной скобочной последовательностью, если она состоит только из символов «`{`», «`}`», «`[`», «`]`», «`(`», «`)`» и выполнено хотя бы одно из следующих трех условий:

1) S — пустая строка;
2) S можно представить в виде: $S = S_1 + S_2 + S_3 + \ldots + S_N$ ($N > 1$), где S_i — непустые правильные скобочные последовательности, а знак «+» обозначает конкатенацию (приписывание) строк;
3) S можно представить в виде $S = '\{' + C + '\}'$ или $S = '[' + C + ']'$ или $S = '(' + C + ')'$, где C является правильной скобочной последовательностью.

Дана строка, состоящая только из символов «`{`», «`}`», «`[`», «`]`», «`(`», «`)`». Требуется определить, какое минимальное количество символов надо вставить в эту строку для того, чтобы она стала правильной скобочной последовательностью.

Формат входных данных.

В первой строке входного файла записана строка, состоящая только из символов «`{`», «`}`», «`[`», «`]`», «`(`», «`)`». Длина строки не превосходит 100 символов.

Формат выходных данных.

Вывести в первую строку выходного файла единственное неотрицательное целое число — ответ на поставленную задачу.

Примеры.

e.in	e.out
{(})	2
([{}])	0

Задача VII-F. Двоякие числа

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Натуральное число называется *двояким*, если в его десятичной записи встречается не более двух различных цифр. Например, числа 3, 23, 33, 100, 12121 — двоякие, а числа 123 и 9980 — нет.

Для заданного натурального числа N требуется найти ближайшее к нему двоякое число (если таких чисел два — любое из них).

Формат входных данных.

Во входном файле записано одно натуральное число N, не превосходящее 30000.

Формат выходных данных.

В выходной файл требуется выдать единственное число — ближайшее двоякое к числу N.

Примеры.

f.in	f.out
123	122
2012	2020
11111	11111

Задача VII-G. OOO

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Одна серьезная организация (ОСО) решила построить очень охраняемый объект (ООО). Для этого она нашла на пустыре два одиноко стоящих прожектора. Каждый из прожекторов освещает какой-то угол, меньший 180 градусов. ОСО хочет, чтобы весь ООО был освещен обоими прожекторами.

Положения прожекторов и освещаемые ими углы заданы. Требуется найти максимальную площадь ООО, который удастся построить.

Возможна ситуация, что ООО может иметь сколь угодно большую площадь. Однако, если это не так, то гарантируется, что в этом случае максимально возможная площадь не будет превышать 10^{15}.

Формат входных данных.

Во входном файле последовательно заданы описания двух прожекторов. Каждый из прожекторов описывается следующим образом: сначала идут координаты прожектора, а затем — координаты двух точек: по одной точке на каждой из сторон угла, освещаемого им. Все числа целые, не превосходящие по модулю 10000.

Формат выходных данных.

Выведите максимально возможную площадь ООО с точностью не менее 3 знаков после десятичной точки. Если ООО построить не удастся, выведите `0`. Если ООО может иметь любую сколь угодно большую площадь, выведите число `-1`.

Примеры.

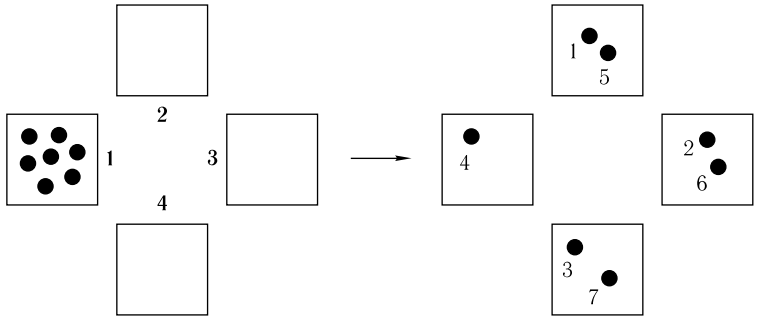
g.in	g.out
0 1 2 3 3 2 3 0 3 3 5 2	-1
0 1 2 3 3 2 3 0 3 3 -3 2	3.000
0 1 2 3 3 2 3 0 3 -2 -3 2	0

Задача VII-H. Калах

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Для игры в калах используют несколько расставленных по кругу коробочек, в которых лежат шарики. Ход осуществляется следующим образом.

Берутся все шарики из одной коробочки и раскладываются по одному в коробочки подряд начиная со следующей по часовой стрелке. Если шариков больше, чем коробочек, то процесс продолжается (шарики раскладываются по второму кругу, по третьему и т. д.), пока не будут разложены все шарики. В коробочку, из которой взяли шарики, их тоже кладут. Пример одного хода приведен на рисунке. Справа шарики пронумерованы в том порядке, в котором они раскладывались по коробочкам.



Петя, тренируясь перед соревнованиями, разложил шарики по коробочкам произвольным образом и стал делать произвольные ходы. После каждого хода он записывал номер коробочки, в которую попадал последний шарик. В некоторый момент он решил восстановить начальную конфигурацию по конечной и по тем записям, которые он делал. Напишите программу, которая поможет ему в этом.

Формат входных данных.

В первой строке входного файла записано два натуральных числа: $N \leqslant 100$ — количество коробочек и $M \leqslant 100$ — количество сделанных Петей ходов. Коробочки пронумерованы последовательно по часовой стрелке числами от 1 до N. В следующих N строках записано количество шариков в первой, второй, …, N-й коробочках в конечной конфигурации (по одному числу в каждой строке). В следующих M строках записаны номера коробочек, в которые был положен последний шарик на первом, втором, …, M-м ходу соответственно (по одному числу в каждой строке). Общее количество шариков не превосходит 10^9.

Формат выходных данных.

В выходной файл требуется вывести N чисел: первоначальное количество шариков в первой, второй, …, N-й коробочках.

Примеры.

Первый пример описывает ситуацию, изображенную на рисунке.

h.in	h.out
4 1 1 2 2 2 4	7 0 0 0
2 2 1 1 2 2	1 1

Задача VII-I. Сортировка масс

Имя входного файла:	i.in
Имя выходного файла:	i.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Как известно, Россия является одним из ведущих экспортеров нефти. Разные страны мира, от достаточно больших до сравнительно маленьких, нуждаются в этой нефти, как в воздухе. В ее состав в больших количествах входят ароматические углеводороды, которые обуславливают ее высокое качество. Доставка нефти в пункт назначения осуществляется с помощью нефтепровода. Считается, что количество нефти, отправленное в страну назначения, равно количеству полученной нефти. На самом деле это, конечно, не так. Как и многое другое, нефть воруют некоторые несознательные личности. Причем неофициально считается, что больше нефти воруют в нефтепроводах тех стран, куда нефти посылается больше (может быть, несознательные личности считают, что приносят таким образом меньше ущерба, кто знает…). Официальное руководство компании РусскаяНефть решило узнать, правдивый это слух или нет, чтобы усилить (а может, просто установить) охрану на тех нефтепроводах, где больше всего воруют нефть.

Для этого им нужно отсортировать нефтепроводы по количеству нефти, которая протекает в направлении какой-то страны за сутки. У компании РусскаяНефть, как и у любой уважающей себя компании, есть несколько штатных программистов, и руководство предложило им решить эту, в сущности, нетрудную задачу. Но программистов поставило в тупик то, что

данные о количестве нефти представлены в разных единицах измерения (начиная от граммов и заканчивая тоннами).

Поэтому они решили найти человека, который был бы в силах решить эту задачу за них, и обещают взять его на работу в эту перспективную и процветающую компанию. Решите задачу, и, кто знает, может повезет именно Вам?

Формат входных данных.

В первой строке входного файла находится целое число N — количество нефтепроводов ($1 \leqslant N \leqslant 1000$). В каждой из следующих N строк находится количество (точнее — масса) нефти, транспортируемой по соответствующему нефтепроводу за сутки, по одному значению в строке. Масса нефти задана целым числом от 1 до 10000 с указанием соответствующей единицы измерения. Число и единица измерения разделены ровно одним пробелом. Единица измерения задается одной из трех букв: `g` (граммы), `p` (пуды), `t` (тонны), причем перед этой буквой может стоять одна из приставок: `m` (милли-), `k` (кило-), `M` (мега-), `G` (гига-). Напомним, что эти приставки обозначают умножение единицы измерения на 10^{-3}, 10^3, 10^6 и 10^9 соответственно. 1 пуд = 16380 граммов, 1 тонна = 10^6 граммов.

Формат выходных данных.

В выходной файл выведите N строк, в которых должны быть записаны массы нефти в порядке неубывания. Каждая строка должна описывать массу нефти в одном из нефтепроводов.

Масса должна быть описана согласно формату входного файла. Единицы измерения массы в выходном файле не обязаны соответствовать единицам измерения массы во входном файле, главное, чтобы массы были равны исходным. Все числа в выходном файле, также как и во входном, должны быть натуральными и не превышать 10000.

Пример.

i.in	i.out
5 234 g 4576 mp 2 t 32 mg 2 Mg	32 mg 234 g 4576 mp 2 t 2 t

Олимпиада VIII. Заочный тур 2005–2006 учебного года

Задача VIII-A. Часы с боем

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Старинные часы бьют каждые полчаса. Причем в начале каждого часа они бьют столько раз, сколько сейчас часов (по 1 разу — в час ночи и в час дня, по 2 раза — в два часа ночи и в два часа дня и т. д., в полночь и в полдень они бьют, соответственно, по 12 раз). И еще 1 раз они бьют в середине каждого часа.

Дан промежуток времени (известно, что прошло строго меньше 24 часов). Напишите программу, определяющую, сколько ударов сделали часы за это время.

Формат входных данных.

В первой строке записан начальный момент времени, во второй строке — конечный. Моменты времени задаются двумя целыми числами, разделяющимися пробелом. Первое число задает часы (от 0 до 23), второе — минуты (от 1 до 59, при этом оно не равно 30).

Формат выходных данных.

В выходной файл выведите одно число — сколько ударов сделали часы за этот отрезок времени.

Примеры.

a.in	a.out
5 20 10 25	45
10 25 5 20	135
5 2 5 21	0

Задача VIII-B. Выборы жрецов

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

В стране Олимпиадии снова выборы.

Страна состоит из маленьких графств. Графства объединяются в конфедерации. Каждая конфедерация раз в год выбирает себе покровителя — одного из 200 жрецов. Этот ритуал называется Великими Перевыборами Жрецов и выглядит так: конфедерации одновременно подают заявления (одно от конфедерации) в Совет Жрецов о том, кого они хотели бы видеть своим покровителем (если заявление не подано, то считают, что конфедерация хочет оставить себе того же покровителя). После этого все заявки удовлетворяются. Если несколько конфедераций выбирают одного и того же Жреца, то они навсегда объединяются в одну. Таким образом, каждый Жрец всегда является покровителем не более чем одной конфедерации. Требуется написать программу, позволяющую Совету Жрецов выяснить номер Жреца-покровителя каждого графства после Великих Перевыборов. В Совете все графства занумерованы (начиная с 1). Все Жрецы занумерованы числами от 1 до 200 (некоторые из них сейчас могут не быть ни чьими покровителями).

Формат входных данных.

Во входном файле записано число N — количество графств в стране ($1 \leqslant N \leqslant 5000$), и далее для каждого графства записан номер Жреца-покровителя конфедерации, в которую оно входит (графства считаются по порядку их номеров). Затем указаны заявления от конфедераций. Сначала записано число M — количество поданных заявлений, а затем M пар чисел: первое число — номер текущего Жреца-покровителя, второе — номер желаемого Жреца-покровителя.

Все числа во входном файле разделяются пробелами и (или) символами перевода строки.

Формат выходных данных.

В выходной файл вывести для каждого графства одно число — номер его Жреца-покровителя после Великих Перевыборов. Сначала — для первого графства, затем — для второго и т. д.

Пример.

b.in	b.out
7 1 1 5 3 1 5 1 2 5 1 1 3	3 3 1 3 3 1 3

Задача VIII-C. Представление чисел

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Дано натуральное число N. Требуется представить его в виде суммы двух натуральных чисел A и B таких, что НОД (наибольший общий делитель) чисел A и B — максимален.

Формат входных данных.

Во входном файле записано натуральное число N ($2 \leqslant N \leqslant 10^9$).

Формат выходных данных.

В выходной файл выведите два искомых числа A и B. Если решений несколько, выведите любое из них.

Примеры.

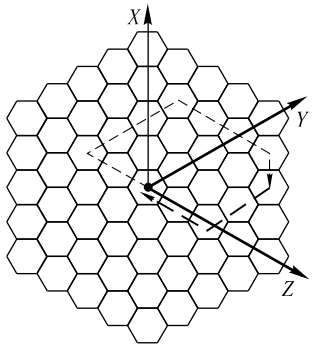
c.in	c.out
15	5 10
16	8 8

Задача VIII-D. Гексагон

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Поле для игры в новую игру «Гексагон» разбито на шестиугольники. Игрок, стартуя из некоторого начального шестиугольника, сделал несколько ходов. Каждый ход заключается в перемещении фишки в соседний

шестиугольник (имеющий с тем, в котором находилась фишка до начала хода, общую сторону) — тем самым, ход делается вдоль одного из направлений X, Y или Z (см. рис.). Игрок записал все свои ходы, причем, если фишка двигалась вдоль какого-либо направления несколько раз подряд, то в записи это обозначается указанием направления и количества ходов, которые были сделаны.



Напишите программу, которая найдет кратчайший (по количеству совершаемых ходов) путь в начальную клетку из той, где фишка оказалась после N ходов игрока.

Формат входных данных.

В первой строке входного файла записано число N — количество строк в записи перемещений фишки ($1 \leqslant N \leqslant 100$). Далее идет N строк с записью ходов: в каждой строке записана сначала большая буква `X`, `Y` или `Z`, задающая направление, затем пробел и число, задающее количество ходов в данном направлении (число может быть и отрицательным, если игрок перемещал фишку параллельно оси, но в направлении, противоположном направлению оси). Все числа по модулю не превышают 200.

Формат выходных данных.

В выходной файл выведите описание кратчайшего пути обратно в начальную клетку в том же формате, в каком описание задано во входном файле (за исключением ограничений). Все числа, определяющие количество ходов в каком-либо направлении, должны быть ненулевыми.

Пример.

d.in	d.out
4 Z -2 Y 3 Z 3 X -1	2 Y -2 Z -2

Задача VIII-E. Магия Копперфильда

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Всемирно известный маг Дэвид Копперфильд любит показывать следующий трюк. Квадрат из N столбцов и N строк, в каждой клетке которого находится какая-нибудь картинка, появляется на экране телевизора. Пусть все картинки пронумерованы следующим образом:

1	2	...	N
N+1	N+2	...	2*N
:	:	...	:
N*(N-1)+1	N*(N-1)+2	...	N*N

Дэвид просит каждого зрителя поставить палец на левую верхнюю картинку (т. е. в клетку номер 1), и Магия начинается: маг просит зрителей сдвинуть свой палец K_1 раз в произвольном направлении (сдвигать палец разрешается только на соседнюю картинку по горизонтали или по вертикали, оставлять палец на месте запрещено, при этом если, допустим, Дэвид попросил сдвинуть палец 3 раза, то можно, например, сдвинуть палец на одну клетку вправо, затем — на одну клетку вниз, затем — на одну вверх). Затем со словами: «Ваш палец не здесь» Дэвид убирает некоторые картинки, и — что удивительно — пальцы телезрителей действительно не указывают на те картинки, которые убирает Дэвид. Затем он просит сделать K_2 ходов и т. д. (если Дэвид уже убрал какую-то картинку, то ходить через эту клетку нельзя). В конце Дэвид убирает все картинки, кроме одной, и, улыбаясь, говорит: «Вы здесь» (аплодисменты).

Дэвиду приходится довольно часто повторять этот трюк, и, чтобы не ошибиться, он попросил написать программу, которая будет ему сообщать, сколько ходов должны делать телезрители и какие картинки нужно убирать. Напишите такую программу.

Формат входных данных.

Во входном файле записано число N — размер квадрата ($2 \leqslant N \leqslant 100$).

Формат выходных данных.

В выходной файл ваша программа должна печатать следующие строки чисел:

$$\begin{array}{llll} K_1 & X_{1,1} & X_{1,2} & \ldots \quad X_{1,m1} \\ K_2 & X_{2,1} & X_{2,2} & \ldots \quad X_{2,m2} \\ \ldots\ldots\ldots\ldots\ldots\ldots & & & \\ K_e & X_{e,1} & X_{e,2} & \ldots \quad X_{e,me}, \end{array}$$

где K_i — это число ходов, которые должны сделать телезрители, а $X_{i,1}$, … … , $X_{i,mi}$ — номера картинок, которые Дэвид должен убрать с экрана после этого. При этом все K_i должны удовлетворять условию $2N \leqslant K_i \leqslant 10000$ и все K_i должны быть различны. Каждая картинка (кроме той, которая

останется) должна убираться ровно один раз. После каждой просьбы зрителей сделать K_i ходов, Дэвид должен убирать хотя бы одну картинку. Каждое K_i должно печататься в начале новой строки. Ситуаций, когда телезритель остался на клетке, у которой нет соседних, а его просят куда-нибудь ходить, возникать не должно.

Пример.

e.in	e.out
3	8 4 6 13 9 10 7 1 7 8 11 3 5

Задача VIII-F. Кубическое уравнение

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Напишите программу, которая будет искать все целые x, удовлетворяющие уравнению:

$$Ax^3 + Bx^2 + Cx + D = 0,$$

где A, B, C, D — заданные целые числа.

Формат входных данных.
Во входном файле записаны четыре целых числа: A, B, C, D. Все числа по модулю не превышают $2 \cdot 10^9$.

Формат выходных данных.
В выходной файл выведите сначала количество решений этого уравнения в целых числах, а затем сами корни в порядке возрастания. Если уравнение имеет бесконечно много корней, выведите в выходной файл одно число -1 (минус один).

Примеры.

f.in	f.out
1 0 0 -27	1 3
0 1 2 3	0

Задача VIII-G. Скорая помощь

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Бригада скорой помощи выехала по вызову в один из отдаленных районов. К сожалению, когда диспетчер получил вызов, он успел записать только адрес дома и номер квартиры K_1, а затем связь прервалась. Однако он вспомнил, что по этому же адресу некоторое время назад скорая помощь выезжала в квартиру K_2, которая расположена в подъезде P_2 на этаже N_2. Известно, что в доме M этажей и количество квартир на каждой лестничной площадке одинаково. Напишите программу, которая вычисляет номер подъезда P_1 и номер этажа N_1 квартиры K_1.

Формат входных данных.
Во входном файле записаны пять положительных целых чисел: K_1, M, K_2, P_2, N_2. Все числа не превосходят 1000.

Формат выходных данных.
Выведите два числа P_1 и N_1. Если входные данные не позволяют однозначно определить P_1 или N_1, вместо соответствующего числа напечатайте 0. Если входные данные противоречивы, напечатайте два раза -1 (минус один).

Примеры.

g.in	g.out
89 20 41 1 11	2 3
11 1 1 1 1	0 1
3 2 2 2 1	-1 -1

Задача VIII-H. A-функция от строчки

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта

Дана строка S, состоящая из N символов. Определим функцию $A(i)$ от первых i символов этой сроки следующим образом: $A(i)$ = максимально

возможному k, для которого равны следующие строки:

$$S[1] + S[2] + S[3] + \ldots + S[k],$$
$$S[i] + S[i-1] + S[i-2] + \ldots + S[i-k+1],$$

где $S[i]$ — i-й символ строки S, а знак «+» означает, что символы записываются в строчку непосредственно друг за другом.

Напишите программу, которая вычислит значения функции A для заданной строчки при всех возможных значениях i от 1 до N.

Формат входных данных.

В первой строке входного файла записано одно число N ($1 \leqslant N \leqslant 200000$). Во второй строке записана строка длиной N символов, состоящая только из больших и (или) маленьких латинских букв.

Формат выходных данных.

В выходной файл выведите N чисел — значения функции $A(1)$, $A(2)$, …, $A(N)$.

Пример.

h.in	h.out
5 aabaa	1 2 0 1 5

Задача VIII-I. Олимпиада по алхимии

Имя входного файла:	i.in
Имя выходного файла:	i.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

В государстве алхимиков есть N населенных пунктов, пронумерованных числами от 1 до N, и M дорог. Населенные пункты бывают двух типов: деревни и города. Кроме того, в государстве есть одна столица (она может располагаться как в городе, так и в деревне). Каждая дорога соединяет два населенных пункта, и для проезда по ней требуется T_i минут. В столице было решено провести 1-ю государственную командную олимпиаду по алхимии. Для этого во все города из столицы были отправлены гонцы (по одному гонцу на город) с информацией про олимпиаду.

Напишите программу, которая посчитает, в каком порядке и через какое время каждый из гонцов доберется до своего города. Считается, что гонец во время пути не спит и нигде не задерживается.

Формат входных данных.

Во входном файле сначала записаны 3 числа: N, M, K — количество населенных пунктов, количество дорог и количество городов ($2 \leqslant N \leqslant 1000$, $1 \leqslant M \leqslant 10000$, $1 \leqslant K \leqslant N$). Далее записан номер столицы C ($1 \leqslant C \leqslant N$). Следующие K чисел задают номера городов. Далее следуют M троек чисел S_i, E_i, T_i, описывающих дороги: S_i и E_i — номера населенных пунктов, которые соединяет данная дорога, а T_i — время для проезда по ней ($1 \leqslant T_i \leqslant 100$).

Гарантируется, что до каждого города из столицы можно добраться по дорогам (возможно, через другие населенные пункты).

Формат выходных данных.

Выведите в выходной файл K пар чисел: для каждого города должен быть выведен его номер и минимальное время, спустя которое гонец может в нем оказаться (время измеряется в минутах с того момента, как гонцы выехали из столицы). Пары в выходном файле должны быть упорядочены по времени прибытия гонца.

Примеры.

i.in	i.out
5 4 5 1 1 2 3 4 5 1 2 1 2 3 10 3 4 100 4 5 100	1 0 2 1 3 11 4 111 5 211
5 5 3 1 2 4 5 2 1 1 2 3 10 3 4 100 4 5 100 1 5 1	5 1 2 1 4 101

Задача VIII-J. Лотерея

Имя входного файла:	j.in
Имя выходного файла:	j.out
Максимальное время работы на одном тесте:	4 секунды
Максимальный объем используемой памяти:	64 мегабайта

На одном из телеканалов каждую неделю проводится следующая лотерея. В течение недели участники делают свои ставки. Каждая ставка заключается в назывании какого-либо M-значного числа в системе счисления с основанием K (т. е. по сути, каждый участник называет M цифр, каждая из которых лежит в диапазоне от 0 до $K-1$). Ведущие нули в числах допускаются.

В некоторый момент прием ставок на текущий розыгрыш завершается, и после этого ведущий в телеэфире называет выигравшее число (это также M-значное число в K-ичной системе счисления). После этого те телезрители, у кого первая цифра их числа совпала с первой цифрой числа, названного ведущим, получают выигрыш в размере A_1 рублей. Те, у кого совпали первые две цифры числа, получают A_2 рублей (при этом, если у игрока совпала вторая цифра, но не совпала первая, он не получает ничего). Аналогично, угадавшие первые три цифры получают A_3 рублей и т. д. Угадавшие все число полностью получают A_M рублей. При этом, если игрок угадал t первых цифр, то он получает A_t рублей, но не получает призы за угадывание $t-1$, $t-2$ и т. д. цифр. Если игрок не угадал первую цифру, он не получает ничего.

Напишите программу, которая по известным ставкам, сделанным телезрителями, находит число, которое должен назвать телеведущий, чтобы фирма-организатор розыгрыша выплатила в качестве выигрышей минимальную сумму. Для вашего удобства ставки, сделанные игроками, уже упорядочены по неубыванию.

Формат входных данных.

В первой строке задаются числа N: (количество телезрителей, сделавших свои ставки, $1 \leqslant N \leqslant 100000$), M (длина чисел, $1 \leqslant M \leqslant 10$) и K (основание системы счисления, $2 \leqslant K \leqslant 10$). В следующей строке записаны M чисел A_1, A_2, …, A_M, задающих выигрыши в случае совпадения только первой, первых двух, …, всех цифр ($1 \leqslant A_1 \leqslant A_2 \leqslant \ldots \leqslant A_M \leqslant 100000$). В каждой из следующих N строк записано по одному M-значному K-ичному числу. Числа идут в порядке неубывания.

Формат выходных данных.

В первой строке выведите искомое число (если решений несколько — выведите любое из них), а во второй строке — сумму, которую при назывании телеведущим первого числа придется выплатить в качестве выигрыша.

Примеры.

j.in	j.out
10 3 2 1 3 100 000 000 001 010 100 100 100 100 110 111	011 6
1 1 10 100 0	7 0

Задача VIII-K. Коммерческий калькулятор

Имя входного файла:	k.in
Имя выходного файла:	k.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Фирма OISAC выпустила новую версию калькулятора. Этот калькулятор берет с пользователя деньги за совершаемые арифметические операции. Стоимость каждой операции в долларах равна 5% от числа, которое является результатом операции.

На этом калькуляторе требуется вычислить сумму N натуральных чисел (числа известны). Нетрудно заметить, что от того, в каком порядке мы будем складывать эти числа, иногда зависит, в какую сумму денег нам обойдется вычисление суммы чисел (тем самым, оказывается нарушен классический принцип «от перестановки мест слагаемых сумма не меняется»).

Например, пусть нам нужно сложить числа 10, 11, 12 и 13. Тогда, если мы сначала сложим 10 и 11 (это обойдется нам в \$1.05), потом результат — с 12 (\$1.65) и затем — с 13 (\$2.3), то всего мы заплатим \$5, если же сначала отдельно сложить 10 и 11 (\$1.05), потом — 12 и 13 (\$1.25)

и наконец, сложить между собой два полученных числа (\$2.3), то в итоге мы заплатим лишь \$4.6.

Напишите программу, которая будет определять, за какую минимальную сумму денег можно найти сумму данных N чисел.

Формат входных данных.

Во входном файле записано число N ($2 \leqslant N \leqslant 100000$). Далее идет N натуральных чисел, которые нужно сложить, каждое из них не превышает 10000.

Формат выходных данных.

В выходной файл выведите, сколько денег нам потребуется на нахождение суммы этих N чисел. Результат должен быть выведен с двумя знаками после десятичной точки.

Примеры.

k.in	k.out
4 10 11 12 13	4.60
2 1 1	0.10

Задача VIII-L. Пересечение кубов

Имя входного файла:	l.in
Имя выходного файла:	l.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

В пространстве с прямоугольной системой координат находятся два куба. Про них известно следующее:

- сторона каждого куба равна 2;
- центр (т. е. центр симметрии) каждого куба совпадает с началом данной системы координат;
- координаты вершин первого куба $A_1A_2A_3A_4A_5A_6A_7A_8$ следующие: $A_1(1, 1, 1)$, $A_2(1, -1, 1)$, $A_3(-1, -1, 1)$, $A_4(-1, 1, 1)$, $A_5(1, 1, -1)$, $A_6(1, -1, -1)$, $A_7(-1, -1, -1)$, $A_8(-1, 1, -1)$;
- вершины второго куба $B_1B_2B_3B_4B_5B_6B_7B_8$ пронумерованы так, что путем поворота кубы можно совместить, и при этом совместятся соответствующие их вершины (A_1 и B_1, A_2 и B_2, …, A_8 и B_8);
- координаты вершин второго куба даны во входном файле.

Требуется найти объем пересечения (т. е. общей части) этих кубов.

Формат входных данных.

Во входном файле записаны 8 троек действительных чисел — координаты вершин второго куба $B_1B_2B_3B_4B_5B_6B_7B_8$.

Формат выходных данных.

В выходной файл выведите одно число — искомый объем пересечения кубов. Ответ не должен отличаться от верного более чем на 0,00001.

Примеры.

l.in	l.out
1.0000000000 -1.0000000000 1.0000000000 1.0000000000 -1.0000000000 -1.0000000000 -1.0000000000 -1.0000000000 -1.0000000000 -1.0000000000 -1.0000000000 1.0000000000 1.0000000000 1.0000000000 1.0000000000 1.0000000000 1.0000000000 -1.0000000000 -1.0000000000 1.0000000000 -1.0000000000 -1.0000000000 1.0000000000 1.0000000000	8.00000
1.41421356237309504880168872420970 1 0 -1.41421356237309504880168872420971 -1.41421356237309504880168872420970 1 0 1.41421356237309504880168872420971 1.41421356237309504880168872420970 -1 0 -1.41421356237309504880168872420971 -1 -1.41421356237309504880168872420970 -1 0 1.41421356237309504880168872420971 -1	6.62742

Олимпиада IX. Командная олимпиада 2005–2006 учебного года

Задача IX-A. Результаты олимпиады

Имя входного файла:	a.in
Имя выходного файла:	a.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

По новым правилам Московской командной олимпиады итоги олимпиады подводятся следующим образом.

Команды решают задачи и могут во время тура посылать их на проверку. Задача проверяется на системе тестов.

Каждая задача оценивается из 2 баллов. 2 балла получает программа, которая проходит все тесты. Кроме того, 1 баллом оцениваются решения, которые проходят некоторое количество первых тестов в задаче (это количество определяется жюри, оно может быть различным в различных задачах, это количество, как и общее количество тестов по задаче, участникам не сообщается). Жюри может (однако не обязано) указать в условии дополнительные ограничения, которым удовлетворяют тесты, за прохождение которых ставится 1 балл.

Помимо баллов за решение задач, команды получают штрафное время. За каждую задачу штрафное время определяется следующим образом.

- Если за данную задачу у команды 0 баллов (независимо от того, делала команда попытки сдачи этой задачи или нет), штрафное время за эту задачу равно 0.
- Если у команды 1 балл за эту задачу, то штрафное время за задачу определяется как время в минутах от момента начала тура до момента, когда была сделана первая попытка, которая была оценена в 1 балл, плюс по 20 штрафных минут за каждую попытку по этой задаче, ей предшествовавшую. Заметьте, что все попытки, которые делаются после того, как команда получила 1 балл за задачу (но до того, как она получила 2 балла), не влияют на штрафное время и на баллы по задаче (даже если после этого будет сделана попытка, которая получит 0 баллов, 1 набранный балл у команды останется).
- Если у команды 2 балла за задачу, то штрафное время за эту задачу определяется как время в минутах от момента начала тура до момента, когда задача была сдана на 2 балла, плюс по 20 штрафных минут за каждую предшествовавшую неверную попытку по этой задаче (при этом неверными теперь считаются в том числе и попытки, получающие 1 балл). Заметьте, что все попытки, которые делаются после того, как команда решила задачу полностью, не влияют на штрафное время.

При этом, если произошла ошибка компиляции программы, то такая попытка не учитывается при подсчете неверных попыток (т. е. она вообще игнорируется).

Например, если на 1-й минуте команда послала программу, которая набрала 0 баллов, то штрафное время равно 0. Пусть на 2-й минуте команда послала решение, которое набрало 1 балл, тогда команда имеет за эту задачу 1 балл и штрафное время, равное 22 минутам (2 минуты от начала тура + 20 штрафных минут). Если на 3-й минуте команда сдала решение задачи на 2 балла, то результат по этой задаче у команды 2 балла, а штрафное время равно 43 минуты (3 минуты от начала тура $+ 2 \cdot 20$ минут за 2 предыдущие попытки).

И баллы и штрафное время, полученное командой по разным задачам, суммируются (получаются суммарные баллы и суммарное штрафное время). Команды ранжируются сначала по баллам, а при равном количестве баллов — по штрафному времени (чем меньше штрафное время, тем выше место команды).

Напишите программу, которая по протоколу попыток, сделанных командой, вычисляет количество набранных командой баллов и штрафное время.

Формат входных данных.

В первой строке записано число N — суммарное количество задач ($1 \leqslant N \leqslant 26$). Далее записано N чисел A_i — количество тестов в каждой задаче ($2 \leqslant A_i \leqslant 100$). Далее записано N чисел B_i — количество тестов, которое должно пройти в соответствующей задаче, чтобы команда получила за нее 1 балл ($0 < B_i < A_i$).

Далее идет M — количество сделанных командой попыток. Затем идет M строк, каждая из которых содержит 3 числа: первое задает время в минутах от начала тура до момента совершения данной попытки, второе — номер задачи, третье — количество пройденных тестов (или `-1`, если произошла ошибка компиляции). Все попытки упорядочены по времени, ни в какую минуту команда не делала более одной попытки.

По правилам олимпиады общее количество попыток не превышает 200, продолжительность тура — 3 месяцев (на самом деле, командная олимпиада длится обычно не более 5 часов, однако жюри олимпиады не исключает, что эти же правила когда-нибудь будут использоваться в заочном туре, который длится как раз 3 месяца).

Формат выходных данных.

В выходной файл выведите два целых числа — количество набранных командой баллов и суммарное штрафное время.

Примеры.

a.in	a.out
1 20 10 3 1 1 0 2 1 10 3 1 20	2 43
2 17 24 2 2 2 5 1 1 8 2 -1	0 0

Задача IX-B. Сокращение дроби

Имя входного файла:	b.in
Имя выходного файла:	b.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Дана дробь $\frac{a}{b}$. Требуется ее сократить, т. е. записать это же число в виде $\frac{c}{d}$, где c — целое число, d — натуральное число и d минимально возможное.

Формат входных данных.

Во входном файле записаны два целых числа: a и b ($-100 \leqslant a \leqslant 100$, $0 < b \leqslant 100$).

Формат выходных данных.

В выходной файл выведите два числа: c и d.

Примеры.

b.in	b.out
3 6	1 2
-2 5	-2 5

Оценка задачи.

1 балл получат программы, правильно решающие задачу для случая положительного числа a.

Задача IX-C. Современники

Имя входного файла:	c.in
Имя выходного файла:	c.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

Группа людей называется *современниками*, если был такой момент, когда они могли собраться все вместе и обсуждать какой-нибудь важный вопрос. Для этого в тот момент, когда они собрались, каждому из них должно было уже исполниться 18 лет, но еще не исполниться 80 лет.

Вам дан список дат жизни великих людей. Выведите всевозможные максимальные множества современников. Множество современников будем называть максимальным, если нет другого множества современников, которое включает в себя всех людей из первого множества.

Будем считать, что в день своего 18-летия человек уже может принимать участие в такого рода собраниях, а в день 80-летия, равно как и в день своей смерти, — нет.

Формат входных данных.

Во входном файле записано сначала число N — количество людей ($1 \leqslant N \leqslant 10000$). Далее в N строках входного файла записано по шесть чисел: первые три задают дату (день, месяц, год) рождения, следующие три — дату смерти (она всегда не ранее даты рождения). День (в зависимости от месяца, а в феврале — еще и года) от 1 до 28, 29, 30 или 31, месяц — от 1 до 12, год — от 1 до 2005.

Формат выходных данных.

В выходном файле должны быть записаны все максимальные множества современников. Каждое множество должно быть записано на отдельной строке и содержать номера людей (люди во входном файле нумеруются в порядке их задания начиная с 1). Номера людей должны разделяться пробелами.

Никакое множество не должно быть указано дважды.

Если нет ни одного непустого максимального множества, выведите в выходной файл одно число 0.

Гарантируется, что входные данные будут таковы, что размер правильного выходного файла не превысит 2 Мб.

Примеры.

c.in	c.out
3 2 5 1988 13 11 2005 1 1 1 1 1 30 1 1 1910 1 1 1990	2 3
3 2 5 1968 13 11 2005 1 1 1 1 1 30 1 1 1910 1 1 1990	2 1 3
3 2 5 1988 13 11 2005 1 1 1 1 1 10 2 1 1910 1 1 1928	0

Оценка задачи.

1 балл получат программы, правильно решающие задачу для случая $N \leqslant 100$.

Задача IX-D. Тройки чисел

Имя входного файла:	d.in
Имя выходного файла:	d.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Напишите программу, находящую количество троек целых чисел a, b, p таких, что p — простое число, числа удовлетворяют равенству:

$$\sqrt{a} - \sqrt{b} = \sqrt{p}$$

и каждое из чисел a, b и p лежит в промежутке от N до M (т. е. $N \leqslant a \leqslant M$, $N \leqslant b \leqslant M$, $N \leqslant p \leqslant M$).

Формат входных данных.

Во входном файле записаны целые числа N и M ($0 \leqslant N \leqslant M \leqslant 100000$).

Формат выходных данных.

В выходной файл выведите искомое количество троек чисел a, b, p.

Оценка задачи.

1 балл получат программы, правильно решающие задачу при ограничениях $0 \leqslant N \leqslant M \leqslant N + 5000$.

Примеры.

d.in	d.out
1 8	1
5 20	1
1 7	0

Задача IX-E. T2005

Имя входного файла:	e.in
Имя выходного файла:	e.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

Клавиатура сотового телефона выглядит так:

1 — пробел	2 — abc	3 — def
4 — ghi	5 — jkl	6 — mno
7 — pqrs	8 — tuv	9 — wxyz

Режим ввода T2005 устроен следующим образом. В телефоне есть словарь. Пользователь, чтобы ввести слово, последовательно нажимает клавиши, на которых написаны буквы этого слова. Например, чтобы ввести слово «begin», пользователь должен нажимать клавиши 23446. Но как только в словаре оказывается только одно слово с таким началом, это слово автоматически подставляется и, кроме того, после этого слова автоматически добавляется пробел. Например, пусть пользователь нажал клавиши 234, и оказалось, что слов, ввод которых начинается с нажатия именно этих клавиш, — ровно одно. Тогда автоматически подставится это слово и пробел после него, а все последующие нажатия клавиш уже будут относиться к вводу следующего слова.

Если для ввода какого-то слова нужно нажать последовательность клавиш, которая может являться началом какого-то другого слова, то после ввода слова нужно нажать клавишу 1, что соответствует вводу пробела. При вводе пробела считается, что вы ввели все слово целиком (а не только

какое-либо его начало). Если после ввода пробела оказалось, что в словаре такой последовательности клавиш удовлетворяет несколько слов, подставляется первое из них в алфавитном порядке. Если (опять же после ввода пробела) оказалось, что в словаре нет слова, которое может быть введено такой последовательностью клавиш, то все, что было введено после предыдущего пробела (введенного или автоматически подставленного, или, если в тексте ранее не встречалось ни одного пробела — от начала текста) удаляется. Если после ввода пробела (как нажатием «1», так и автоподстановкой) или в начале текста нажимается клавиша «1», то ее нажатие игнорируется.

Вам дан словарь и последовательность нажатий клавиш. Выведите текст, который был введен пользователем.

Примечание. В тексте используются только маленькие латинские буквы и символ пробел.

Формат входных данных.

Во входном файле записано сначала число N — число слов в словаре ($2 \leqslant N \leqslant 100000$). В следующих N строках записан словарь. Каждое слово записано в отдельной строке. Слова записаны в алфавитном порядке. Никакое слово в словаре не встречается дважды. Длина каждого слова не превосходит 10 символов.

Далее записано число M — количество нажатий клавиш ($1 \leqslant M \leqslant 20000$). Затем записано M чисел, разделяющихся пробелами, описывающих нажатые клавиши. Последовательность нажатий клавиш всегда заканчивается нажатием клавиши «1».

Формат выходных данных.

В выходной файл выведите одну строку — текст, который оказался введен пользователем. Пробел после последнего введенного слова также должен быть выведен в выходной файл.

Примеры.

e.in	e.out
5 po pod sasha shla shosse 12 7 4 5 7 2 7 6 1 7 4 6 1	shla sasha po shosse

e.in	e.out
5 aa b bb c cda 7 2 2 1 2 3 2 1	aa cda b
4 ap lap op vosem 11 1 6 2 4 1 1 8 5 9 1 1	op ap vosem lap
2 sem vosem 6 7 8 9 7 8 1	sem vosem
2 a z 2 5 1	Примечание: в этом примере выходной файл должен быть создан, но должен быть пустым, в частности, в него не нужно выводить пробел.

Оценка задачи.

1 балл получат программы, правильно решающие задачу при ограничениях $2 \leqslant N \leqslant 100$, $1 \leqslant M \leqslant 200$.

Задача IX-F. Роботы

Имя входного файла:	f.in
Имя выходного файла:	f.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

В подземелье есть N залов, соединенных туннелями. В некоторых залах находятся роботы, которые одновременно получили команду собраться в одном месте.

Роботы устроены так, что, получив команду, они все начали двигаться с такой скоростью, что туннель между двумя любыми залами преодолевают за 1 минуту. Роботы не могут останавливаться (в том числе и в залах), а также менять направление движения, находясь в туннелях (однако, попав в зал, робот может из него пойти по тому же туннелю, по которому он пришел в этот зал).

Напишите программу, вычисляющую, через какое минимальное время все роботы смогут собраться вместе (в зале или в туннеле).

Формат входных данных.

Во входном файле записаны сначала числа: N — количество залов ($1 \leqslant N \leqslant 400$) и K — количество туннелей ($1 \leqslant K \leqslant 20000$). Далее записано K пар чисел, каждая пара описывает номера залов, соединяемых туннелем (по туннелю можно перемещаться в обе стороны). Между двумя залами возможно несколько туннелей. Туннель может соединять зал с самим собой. Далее записано число M ($1 \leqslant M \leqslant 400$) — количество роботов. Затем идет M чисел, задающих номера залов, где вначале расположены роботы. В одном зале может быть несколько роботов.

Формат выходных данных.

В выходной файл выведите минимальное время в минутах, через которое роботы могут собраться вместе. Если роботы никогда не смогут собраться вместе, выведите одно число `-1` (минус один).

Примеры.

f.in	f.out
4 5 1 2 2 3 3 4 1 4 1 3 3 1 2 4	1
3 3 1 2 2 3 3 1 3 1 2 3	1.5

Оценка задачи.

1 балл получат программы, правильно решающие задачу в случае, когда встреча роботов произойдет в зале, при ограничениях $N \leqslant 100$, $K \leqslant 2000$, $M \leqslant 100$.

Задача IX-G. Монетки

Имя входного файла:	g.in
Имя выходного файла:	g.out
Максимальное время работы на одном тесте:	2 секунды
Максимальный объем используемой памяти:	64 мегабайта

В Волшебной стране используются монетки достоинством A_1, A_2, …, A_M. Волшебный человечек пришел в магазин и обнаружил, что у него есть ровно по две монетки каждого достоинства. Ему нужно заплатить сумму N. Напишите программу, определяющую, сможет ли он расплатиться без сдачи.

Формат входных данных.

Во входном файле записано сначала число N ($1 \leqslant N \leqslant 10^9$), затем — число M ($1 \leqslant M \leqslant 15$) и далее M попарно различных чисел A_1, A_2, …, A_M ($1 \leqslant A_i \leqslant 10^9$).

Формат выходных данных.

В выходной файл выведите сначала K — количество монет, которое придется отдать Волшебному человечку, если он сможет заплатить указанную сумму без сдачи. Далее выведите K чисел, задающих достоинства монет. Если решений несколько, выведите вариант, в котором Волшебный человечек отдаст наименьшее возможное количество монет. Если таких вариантов несколько, выведите любой из них.

Если без сдачи не обойтись, то выведите одно число `0`. Если же у Волшебного человечка не хватит денег, чтобы заплатить указанную сумму, выведите одно число `-1` (минус один).

Примеры.

g.in	g.out
5 2 1 2	3 2 2 1
7 2 1 2	-1
5 2 3 4	0

Оценка задачи.

1 балл получат программы, правильно решающие задачу с дополнительными ограничениями $A_i \leqslant 10^6$, $M \leqslant 10$.

Задача IX-H. Сверим часы

Имя входного файла:	h.in
Имя выходного файла:	h.out
Максимальное время работы на одном тесте:	3 секунды
Максимальный объем используемой памяти:	64 мегабайта

В компании MacroHard в последнее время резко участились опоздания сотрудников. Проанализировав ситуацию, руководство решило, что это вызвано большим разбросом в показаниях наручных часов сотрудников. После дополнительного совещания руководящего состава было постановлено, что все сотрудники должны перевести часы на одно и то же время (не важно какое).

Все сотрудники компании носят исключительно электронные часы одного образца. Время на них отображается в формате «HH:MM:SS», где HH — часы, MM — минуты, SS — секунды всегда отображаются в виде двух цифр ($00 \leqslant$ HH $\leqslant 23$, $00 \leqslant$ MM $\leqslant 59$, $00 \leqslant$ SS $\leqslant 59$). Перевод часов осуществляется с помощью двух кнопок. Первая кнопка меняет поле редактирования следующим образом: после первого нажатия часы переходят из режима отображения времени в режим редактирования поля HH, после второго — в режим редактирования поля MM, после третьего — в режим редактирования поля SS, а после четвертого возвращаются в режим отображения времени и т. д. по циклу. Каждое нажатие второй кнопки приводит к увеличению редактируемого поля на единицу (в режиме отображения времени ничего не происходит). При переполнении секунд поле SS обнуляется, а MM увеличивается на единицу, при переполнении минут поле MM обнуляется, а HH увеличивается на единицу, а при переполнении часов просто обнуляется поле HH.

И все бы хорошо, но в силу своей природной лени сотрудники хотят минимизировать суммарное число нажатий кнопок при переводе часов. При этом после перевода часов все часы должны оказаться в режиме отображения времени, вначале все часы также находятся в этом режиме.

Напишите программу, определяющую минимальное суммарное количество нажатий кнопок, достаточное для перевода часов всеми сотрудниками к одинаковым показаниям времени.

Формат входных данных.

Первая строка входного файла содержит натуральное число N — количество сотрудников компании ($1 \leqslant N \leqslant 200$). Последующие N строк содержат показания часов каждого из сотрудников в формате «HH:MM:SS».

Формат выходных данных.

В выходной файл выведите одно число — минимальное суммарное количество нажатий.

Пример.

h.in	h.out
2 08:01:01 07:59:00	7

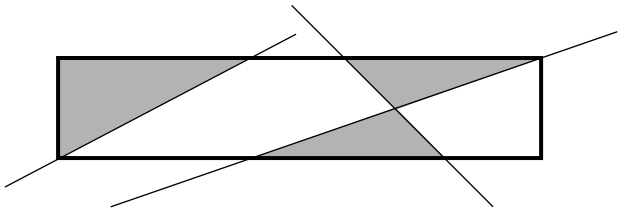
Система оценки.

1 балл получат программы, правильно решающие задачу при ограничении $1 \leqslant N \leqslant 2$.

Задача IX-I. Разрезанный прямоугольник

Имя входного файла:	i.in
Имя выходного файла:	i.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта

На плоскости нарисовали прямоугольник, после чего его разрезали прямыми. Напишите программу, которая вычислит, сколько из полученных кусков исходного прямоугольника имеют треугольную форму.



Рисунок, соответствующий 1-му примеру входных и выходных данных

Формат входных данных.

Во входном файле заданы сначала два положительных числа X, Y, задающие координаты правого верхнего угла прямоугольника. Прямоугольник

расположен в системе координат так, что левый нижний его угол имеет координаты (0, 0) и стороны параллельны осям координат.

Далее записано целое число N — количество разрезов ($1 \leqslant N \leqslant 200$). Далее записаны сами разрезы. Каждый разрез делался вдоль некоторой прямой. Каждая прямая, соответствующая разрезу, задается тремя числами A, B, C такими, что все точки (x, y) этой прямой (и только они) удовлетворяют уравнению: $Ax + By + C = 0$ (при этом всегда $A^2 + B^2 > 0$).

Все числа во входном файле (кроме N) вещественные, заданы с двумя знаками после десятичной точки и не превышают 10^4. Никакие две прямые не совпадают, никакая прямая не содержит сторон прямоугольника. Каждый разрез проходит через точки внутри исходного прямоугольника.

Формат выходных данных.

В выходной файл выведите одно целое число — количество частей исходного прямоугольника, имеющих треугольную форму.

Примеры.

i.in	i.out
5.00 1.00 3 1.00 -2.00 0.00 1.00 -3.00 -2.00 1.00 1.00 -4.00	3
4.00 2.00 2 1.00 -2.00 0.00 1.00 2.00 -4.00	4

Система оценки.

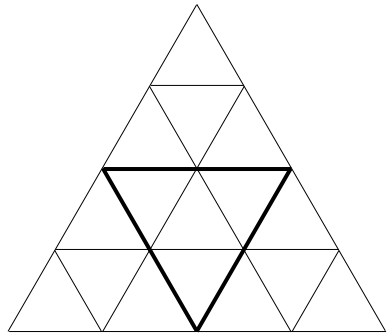
1 балл получат программы, правильно решающие задачу при ограничении $1 \leqslant N \leqslant 50$.

Задача IX-J. Количество треугольников

Имя входного файла:	j.in
Имя выходного файла:	j.out
Максимальное время работы на одном тесте:	1 секунда
Максимальный объем используемой памяти:	64 мегабайта

Рассмотрим фигуру, аналогичную показанной на рисунке (большой равносторонний треугольник, составленный из маленьких равносторонних

треугольников). На рисунке приведена фигура, состоящая из 4-х уровней треугольников.



Напишите программу, которая будет определять, сколько всего в ней треугольников (необходимо учитывать не только «маленькие» треугольники, а вообще все треугольники — в частности, треугольник, выделенный жирным, а также вся фигура являются интересующими нас треугольниками).

Формат входных данных.

Во входном файле записано одно число N — количество уровней в фигуре ($1 \leqslant N \leqslant 100000$).

Формат выходных данных.

Выведите в выходной файл количество треугольников в такой фигуре.

Примеры.

j.in	j.out
1	1
2	5
4	27

РЕШЕНИЯ ЗАДАЧ

Олимпиада I. Заочный тур 2002–2003 учебного года

Задача I-A. Таймер

Тема: задачи для начинающих.

Задачу можно решать, например, таким образом. Переведем заданное во входном файле время в секунды, считая началом отсчета начало суток. Например, время 12:30:00 соответствует $12 \cdot 3600 + 30 \cdot 60 = 45000$ секунд. Аналогично переведем требуемый интервал в секунды и прибавим его к начальному моменту времени — получим требуемое время, только заданное в секундах. Затем переведем это время в требуемый формат и выведем в выходной файл. При таком решении единственная возникающая проблема заключается в том, что полученный ответ может превысить максимально допустимое значение типа данных LongInt. Для хранения числа секунд можно было использовать тип данных Int64 языка FreePascal или тип данных long long int языка GCC.

Другой подход к решению состоит в том, что к начальному числу секунд прибавляется число секунд временно́го интервала, полученное число секунд переводится в минуты и секунды, минуты «запоминаются», а секунды записываются как секунды ответа. Далее прибавим число минут интервала и «запомненные» минуты к начальным минутам, «запомним» лишние часы и т. д. При таком решении типа данных LongInt достаточно.

Задача I-B. Домой на электричках

Тема: алгоритмы на графах — алгоритм Дейкстры.

Подсчитаем, когда мы можем попасть на каждую из станций. Для этого вначале запишем, что на станцию 1 мы можем попасть в начальный момент времени, а для остальных станций запишем значение «неизвестно» (например, выразив его числовой константой -1). Далее на очередном шаге найдем ту вершину A из «неизвестных», до которой мы можем добраться из одной из «известных» как можно раньше. Затем запишем в эту вершину то время, за которое до нее можно добраться, вместо пометки «неизвестно». Как только станция E перестанет быть помечена как «неизвестная», мы и узнаем самый быстрый способ добраться до нее. Если же на очередном шаге мы не нашли ни одной «неизвестной» станции, до которой можно

добраться из «известных», то прекращаем работу алгоритма — до нужной станции добраться невозможно. Ясно, что таким образом мы можем найти минимальное время поездки до любой станции, до которой можно доехать на заданных электричках.

Задача I-C. Клад

Тема: задачи для начинающих.

Для решения этой задачи достаточно было завести две переменные для хранения текущих координат кладоискателя и изменять их в соответствии с входными данными. Например, можно было для каждого указания разбирать 8 случаев — в зависимости от направления — и в каждом из них соответствующим образом передвигать кладоискателя. Однако при таком решении довольно легко ошибиться. Более рациональным было бы использование того факта, что 8 направлений пронумерованы подряд по часовой стрелке. Это приводит, например, вот к такому короткому решению:

```
Var I,N,Dir,Len:LongInt;
  X,Y:Real;
Begin
  Assign(Input,'c.in');
  Reset(Input);
  Read(N); X := 0; Y := 0;
  For I := 1 To N Do Begin
    Read(Dir,Len);
    X := X + Sin(Pi/4*(Dir-1))*Len;
    Y := Y + Cos(Pi/4*(Dir-1))*Len
  End;
  Assign(Output,'c.out');
  Rewrite(Output);
  WriteLn(X:0:3,' ',Y:0:3);
  Close(Output)
End.
```

Задача I-D. Забавная игра

Тема: задачи для начинающих.

Задачу можно решать непосредственно путем перевода исходного числа в двоичную систему счисления, получения всех его циклических сдвигов, перевода их в десятичную систему и выбора максимального из них:

```
Var I,J,Max,Num,NumDigits,Digit:Integer;
  Digits: Array[1..20] Of Integer;
```

```
Begin
  <ввод исходного числа в переменную Num>
  Max := Num;
  NumDigits := 0;
  {перевод в двоичную систему}
  While Num > 0 Do Begin
    Inc(NumDigits);
    Digits[NumDigits] := Num Mod 2;
    Num := Num Div 2
  End;
  For I:=1 To NumDigits - 1 Do Begin
    {сдвиг}
    Digit := Digits[1];
    For J := 2 To NumDigits Do
      Digits[J-1] := Digits[J];
    Digits[NumDigits] := Digit;
    {перевод в десятичную систему}
    Num := 0;
    For J := NumDigits DownTo 1 Do
      Num := Num*2 + Digits[J];
    If Num > Max Then
      Max := Num
  End;
  <вывод значения переменной Max в качестве ответа>
End;
```

Естественно, можно было решать эту задачу с использованием операции побитового сдвига `SHR`, не переводя число явно в двоичную систему.

Задача I-E. Целые точки

Тема: геометрия.

Эта задача проще всего решается с использованием формулы Пика: если вершины многоугольника расположены в точках с целочисленными координатами (в дальнейшем такие точки будут называться «целыми точками»), то выполнено следующее равенство: $S = K + M/2 - 1$, где S — площадь многоугольника, K — число целых точек, лежащих внутри многоугольника, M — число целых точек, лежащих на границе многоугольника. Зная координаты вершин многоугольника, мы можем подсчитать S и M, а значит, можем найти K, т. е. ответ к задаче, по формуле Пика. О том, как найти площадь S многоугольника, зная координаты его вершин, можно прочитать, например, в книге С. М. Окулова «Программирование в алгоритмах» (М.: БИНОМ, 2004). Что касается нахождения числа M, то эта задача сводится к следующей: зная координаты концов отрезка (целые числа), найти, сколько на нем целых точек. Можно заметить, что ответом

на последний вопрос будет НОД(Δ_x, Δ_y) + 1, где НОД(A, B) — наибольший общий делитель чисел A и B, а Δ_x и Δ_y — разности абсцисс и ординат концов отрезка соответственно. А тогда число M — просто сумма НОД(Δ_x, Δ_y) для всех сторон многоугольника.

Задача I-F. Степень

Тема: делимость.

Решение задачи основано на математических рассуждениях. Несложно понять, что искомое число `N` должно содержать в своем разложении на простые множители все простые множители заданного числа `A`. Более того, в большинстве случаев ответом является именно число `M` — произведение всех простых множителей числа `A`, взятых по одному разу. Такой ответ не является верным, только если полученное число `M` настолько мало, что меньше даже степени, в которой одно из простых чисел входит в разложение числа `A`. С другой стороны, как уже было сказано выше, правильный ответ всегда делится на `M`. Поэтому достаточно перебрать числа `M`, `2M`, `3M`, …, пока не найдем подходящее. Легко понять, что правильный ответ не превосходит `40M`, поэтому мы найдем его менее чем за 40 шагов. Единственная сложность, возникающая при таком решении, — возведение в большие степени и возникающие при этом большие числа. Для решения этой проблемы можно в программе вместо чисел как таковых использовать их разложения на простые множители. А именно, завести специальный тип «разложение на простые множители» (назовем его, скажем, `TDecomposition`):

```
Type
  TDecomposition = Record
    NumPrimes: LongInt; {Число различных простых множителей}
    Prime: Array[1..50] Of LongInt; {Множители}
    Degree: Array[1..50] Of LongInt;
    {Степени, с которыми они входят в разложение}
End;
```

Остается реализовать операции «разложение числа на простые множители», «умножение двух разложений», «возведение разложения в степень», «проверка делимости одного разложения на другое».

Задача I-G. Игра с фишками

Тема: алгоритмы на графах — поиск в ширину.

Задача решается стандартным волновым алгоритмом (его также называют «поиском в ширину»). А именно, вначале необходимо поставить в начальную клетку пометку «`0`», а остальные клетки оставить без пометок. Затем на шаге номер `n` мы ищем все клетки с пометкой «`n-1`»,

и если рядом с ними есть непомеченные пустые клетки, то помечаем эти пустые клетки пометкой «n». Как только окажется помеченной (скажем, меткой «m») клетка, находящаяся рядом с конечной клеткой, мы нашли длину кратчайшего пути от начальной клетки до конечной — она равна m+1. Если же мы никогда не пометим клетки, находящейся рядом с конечной клеткой, то пути от начальной до конечной не существует (наш алгоритм заканчивает работу, как только на очередном шаге не будет помечено ни одной клетки). Единственная проблема, возникающая при таком решении, состоит в том, что по условию путь может проходить вне доски. Для того чтобы решить эту проблему, достаточно добавить к доске по одному ряду пустых клеток с каждой из четырех сторон.

Задача I-H. Раскопки

Тема: перебор.

Решение задачи состоит из нескольких этапов. Вначале необходимо привести входные данные к удобному для работы виду. Например, можно удалить все лишние пробелы и поставить в те места, где необходимо вставить знак арифметического действия, какой-нибудь специальный символ, например, «^». Ниже приведен фрагмент программы, выполняющий такое преобразование:

```
While Pos(' ',Expr) > 0 Do Begin
  A := Pos(' ',Expr);
  If (A = 1) Or (A = Length(Expr)) Then
    Delete(Expr,A,1)
  Else If (Expr[A-1] In ['0'..'9',')']) And
      (Expr[A+1] In ['0'..'9','(']) Then
    Expr[A] := '^'
  Else Delete(Expr,A,1)
End;
A := 1;
While A < Length(Expr) Do Begin
  If ((Expr[A] = ')') And (Expr[A+1] In ['0'..'9','('])) Or
     ((Expr[A] In ['0'..'9',')']) And (Expr[A+1] = '(')) Then
    Insert('^',Expr,A+1)
  Else Inc(A)
End;
```

(здесь переменная Expr типа String — это обрабатываемое выражение, а переменная A имеет тип Integer). В первой части этого фрагмента удаляются все пробелы и расставляются те знаки «^», на месте которых стоят пробелы. Во второй части фрагмента вставляются знаки «^» между числом и скобкой даже в тех местах, где пробелов не было.

Затем необходимо написать функцию, вычисляющую значение выражения, в котором знаки арифметических действий уже расставлены. Эта задача называется «синтаксическим разбором», ее решение описано, например, в статье В. А. Матюхина «Подсчет значения арифметического выражения методом рекурсивного спуска» в конце этой книги.

И наконец, надо объединить уже написанные части функцией, перебирающей все возможные расстановки знаков и вызывающей для каждой расстановки функцию подсчета значения. Если значение хотя бы один раз совпадет с тем, что написано в левой части, значит, мы нашли решение задачи. Если же такого совпадения не случится, то требуемой расстановки знаков не существует. Пример функции, перебирающей все расстановки знаков, приведен ниже:

```
Function Rec(Start:LongInt):Boolean;
Begin
  Rec:=True;
  Repeat
    Inc(Start);
    If Start>Length(Expr) Then Begin
      If <значение выражения, полученного в переменной Expr, нужное>
        Then Exit;
      Rec:=False;
      Exit
    End
  Until Expr[Start]='^';
  Expr[Start]:='*';
  If Rec(Start) Then Exit;
  Expr[Start]:='+';
  If Rec(Start) Then Exit;
  Expr[Start]:='-';
  If Rec(Start) Then Exit;
  Expr[Start]:='^';
  Rec:=False
End;
```

Вызывается эта функция как Rec(0), она возвращает значение True в том случае, если искомая расстановка знаков существует, и False — в противном случае. Если расстановка знаков существует, в переменной Expr оказывается одна из возможных расстановок.

Задача I-I. Деревни

Тема: алгоритмы на графах, динамическое программирование.

Эта задача была, наверное, самой сложной задачей олимпиады. Во-первых, необходимо было понять, что деревни образуют *дерево* (связный

граф без циклов). Затем нужно «подвесить» это дерево за какую-нибудь из вершин, и таким образом, получить дерево с выделенным *корнем*, при этом у каждой вершины кроме корня будет известен ее *родитель*. Затем необходимо применить метод динамического программирования: для каждой вершины V и для каждого числа T от 1 до P подсчитаем, какое минимальное число дорог должно быть разрушено, чтобы в поддереве с корнем в V компонента связности (максимальная связная часть), содержащая вершину V, состояла из T вершин — назовем это число Best[V,T]. Будем подсчитывать значения Best, идя по вершинам от листьев к корню. Если для всех детей какой-то вершины V эти значения уже подсчитаны, мы можем подсчитать их и для этой вершины. Для этого рассмотрим как устроена какая-нибудь компонента связности из T вершин, содержащая вершину V. Она состоит из вершины V, а также из компонент, начинающихся в детях этой вершины и имеющих суммарный размер T-1. Тогда, перебрав все разбиения числа T-1 на неотрицательные целые слагаемые: $T-1=a_1+a_2+\ldots+a_m$ (их число m должно быть равно числу детей вершины V), мы берем в качестве очередного кандидата сумму $Best[V_1,a_1]+Best[V_2,a_2]+\ldots+Best[V_m,a_m]$ (где V_1, V_2, ..., V_m — дети вершины V), причем, если какое-то a_i равно 0, то мы считаем $Best[V_i,a_i]=1$. Далее из всех вышеупомянутых кандидатов мы выбираем наименьший и записываем в Best[V,T]. При таком решении довольно долгое время может занимать перебор всех разложений числа T-1 на несколько слагаемых, однако этого частично можно избежать — оставим данный момент читателю для размышлений. После того как значения Best подсчитаны, остается получить из них ответ к задаче. А именно, для того чтобы отрезать от остальных деревень группу из P вершин с корнем в вершине V, требуется разрушить Best[V,P] дорог, плюс еще одну дорогу в случае, если вершина V не является корнем дерева (а именно, дорогу, соединяющую поддерево с корнем в V с остальным деревом). Естественно, теперь необходимо выбрать из всех таких способов тот, который требует разрушения минимального количества дорог.

Автор разбора П. И. Митричев.

Олимпиада III. Личная олимпиада 2003–2004 учебного года

Задача III-A. Наибольшее произведение

Тема: задачи на разные темы.

Сначала попробуем решить задачу очевидным способом, который сразу приходит на ум. Переберем все тройки чисел и выберем из них такую, которая имеет максимальное произведение.

Приведем фрагмент реализации такого способа на языке Паскаль:

```
var
   n : longint; {число элементов в последовательности}
   a : array [1..MaxN] of integer;
       {массив из элементов последовательности}
   i, j, k : longint;
   t, max : longint;
   i1, i2, i3 : longint; {индексы найденной максимальной тройки}
begin
    ...
     max := -MaxLongInt-1;
     {перебираем все различные тройки элементов последовательности}
     for i := 1 to n do
         for j := 1 to n do
             for k := 1 to n do
                 if (i<>j) and (j<>k) and (i<>k) then
                  {все три индекса различны}
                  begin
                      t := longint(a[i]) * a[j] * a[k];
                      if t >= max then
                       begin
                            max := t;
                            i1 := i; i2 := j; i3 := k;
                       end;
                  end;
      writeln(a[i1], ' ', a[i2] ,' ', a[i3]);
end.
```

В этом решении существуют две проблемы. Во-первых, за требуемое время оно успевает сработать примерно при $N < 100$. Нетрудно подсчитать, что количество выполнений тела цикла (количество итераций цикла) равно N^3. Для подсчета времени работы программы удобно считать, что за 1 секунду выполняется порядка 1 миллиона итераций цикла (разумеет-

ся, если в теле цикла содержатся вызовы функций или другие циклы, то эта оценка неверна).

Кроме того, у решения есть еще один недостаток. Элементы последовательности по модулю могут достигать 30000 и вообще говоря, их произведение может выйти за пределы 32-битного целого типа longint. Можно схитрить и решить эту проблему, используя, например, тип extended для переменных t и max.

Однако есть более красивое решение, которое удовлетворяет всем ограничениям, поставленным в задаче, и при этом не прибегает ни к каким хитростям. Рассмотрим его.

В задаче нам необходимо найти такие три числа, произведение которых максимально; т. е., какие бы другие три числа мы не выбирали, их произведение будет всегда меньше или равно максимальному. Теперь давайте сообразим, какие именно числа в последовательности могут давать максимальное произведение. Первое, что приходит на ум — три максимальных числа последовательности. Для последовательностей с неотрицательными элементами это действительно так.

Рассмотрим пример:

3	1	5	0	**9**	4	**6**	2	**6**	6

Когда все числа в последовательности неотрицательны, то максимальное произведение дадут именно три максимальных элемента. В нашем примере это `9*6*6=324`.

Остается понять, как искать три максимальных элемента. Алгоритм нахождения максимального элемента массива широко известен и не требует пояснений. Но нам надо найти не только максимальный элемент, но и «второй», и «третий» максимумы. В нашем примере первый максимум = 9, второй (следующий по значению) = 6, третий = 6. Обратите внимание, что максимумы могут совпадать по значению.

К счастью, задача поиска трех максимумов решается несложно. Пусть в переменных `max1`, `max2`, `max3` хранятся первый, второй и третий максимум соответственно. Присвоим им начальные значения, равные −30000. В процессе работы программы они будут либо улучшены, либо оставлены без изменений (в случае, если последовательность состоит только из минимально возможных чисел, −30000).

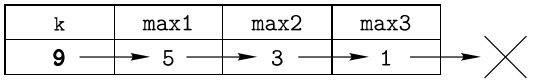
Воспользуемся идеей «проталкивания сверху вниз» очередного элемента в текущие три максимума. Снова обратимся к нашему примеру. Пусть мы уже просмотрели четыре элемента последовательности и правильно заполнили переменные `max1`, `max2` и `max3`.

max1	max2	max3
5	3	1

Мы считали из входного файла очередное число последовательности `k=9`. Сначала сравним его с переменной `max1`.

```
if k > max1 then
  begin
     max3 := max2; max2 := max1; max1 := k;
  end;
```

Поскольку $9 > 5$, то произведем «проталкивание» нового максимума — запишем `k` в `max1`, а в `max2` — старое значение `max1` (ведь оно было больше или по крайней мере равно `max2`!). В `max3` запишется старое значение `max2`, прежнее значение `max3` при этом теряется — хранить его нет больше смысла.



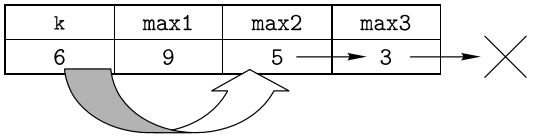
Таким образом, мы получим:

max1	max2	max3
9	5	3

Если окажется, что `k` $\leqslant$ `max1`, тогда следует его сравнить с `max2`.

```
if k > max2 then
  begin
    max3 := max2; max2 := k;
  end;
```

Например, для 7-го элемента нашего примера `k` = 6:



Мы не изменяем `max1`, вместо `max2` записывается `k`, а на место `max3` становится прежний `max2`.

В противном случае сравнивается `k` и `max3`.

```
if k > max3 then max3 := k;
```

Можно реализовывать не «проталкивание сверху вниз», а «проталкивание снизу вверх». Программная логика в этом случае претерпит лишь небольшие изменения:

```
if k>max3 then max3 := k;
if k>max2 then
  begin
     max3 := max2;
     max2 := k;
  end;
if k>max1 then
  begin
    max2 := max1;
    max1 := k;
  end;
```

Но у нас в последовательности могут быть еще и отрицательные числа! Произведение двух отрицательных чисел положительно, и поэтому необходимо найти два минимальных отрицательных числа. Произведем поиск первого и второго минимума `min1` и `min2` в последовательности аналогичным методом.

Теперь сравним два произведения: `min1*min2` и `max2*max3`. Нам надо выбрать максимальное из них. Очевидно, что максимальным произведением из трех элементов будет максимальный элемент последовательности `max1`, умноженный на максимум из `min1*min2` и `max2*max3`.

Вспомним, что в первом решении у нас были проблемы с переполнением при перемножении трех элементов последовательности. Чтобы избежать этих проблем здесь, мы не будем непосредственно производить перемножение трех чисел, а будем сравнивать между собой `min1*min2` и `max2*max3`.

Однако и это еще не все! Тем и замечательны олимпиадные задачи, что мы должны учитывать все возможные случаи, которые допускают ограничения на входные данные. А что будет, если все числа в последовательности отрицательные? Тогда, в случае `min1*min2>max2*max3` наш алгоритм найдет далеко не максимальное произведение. Ясно, что в этом случае ответом задачи будут просто три максимальных элемента массива, так как максимальным произведением (отрицательным!) будет `max1*max2*max3`.

Все остальные случаи полностью покрываются нашим алгоритмом. В том числе и различные случаи с нулем. В случае последовательности из отрицательных элементов и нескольких нулей максимальное произведение будет равно 0, но 0 в этом случае в нашем алгоритме обязательно попадет в `max1`, а остальные два элемента значения иметь не будут. Если же в последовательности есть хотя бы один положительный элемент, то

`max1` никогда не будет равен нулю, поэтому если нулю оказались равны `max2` и `max3`, то при наличии отрицательных элементов максимальное произведение будет формироваться из `min1`, `min2` и `max1`.

Приведем полный текст решения данной задачи на языке Паскаль. Обратите внимание на то, что мы не создаем массив и не храним последовательность элементов в памяти. Поскольку задача решается в один проход по массиву, то для нашего алгоритма этого и не требуется — в каждый момент времени нам требуется знать только один текущий элемент последовательности. Мы обрабатываем его, а затем «забываем».

```
program MOI2004_1;
var
   n : longint; {число элементов в последовательности}
   max1, max2, max3 : longint;
     {первый, второй и третий максимумы в последовательности}
   min1, min2 : longint;
     {первый и второй минимумы в последовательности}
   k : integer; {текущий элемент последовательности}
   i : longint;
begin
     assign(input, 'a.in');
     reset(input);
     assign(output, 'a.out');
     rewrite(output);
     max1 := -30000; max2 := max1; max3 := max1;
     min1 := 30000; min2 := min1;
     readln(n);
     {читаем по очереди все элементы последовательности}
     for i:=1 to n do
      begin
         read(k);
         if k > max1 then
           begin
             max3 := max2; max2 := max1; max1 := k;
           end
         else
         if k > max2 then
           begin
             max3 := max2; max2 := k;
           end
         else
         if k > max3 then
              max3 := k;

         if k < min1 then
```

```
          begin
            min2 := min1; min1 := k;
          end
        else
        if k < min2 then
             min2 := k;
    end;

    if (max1 > 0) and (min1*min2 > max2*max3) then
       writeln(max1, ' ', min1, ' ', min2)
    else
       writeln(max1, ' ', max2, ' ', max3);
    close(input);
    close(output);
end.
```

Автор разбора — С. В. Шедов.

Задача III-B. Покупка билетов

Тема: динамическое программирование.

Попробуем найти решение задачи, постепенно увеличивая размер очереди. Пусть у нас в очереди стоит всего один человек. Поскольку каждому человеку нужен один и только один билет (даже если 2 или 3 билета купить быстрее), то ответом задачи будет число A_1.

Теперь перейдем к очереди из 2 человек. Они могут купить билеты раздельно, тогда время покупки составит $A_1 + A_2$. Кроме того, они могут объединиться и купить билеты вместе, причем купить два билета по условию задачи может только первый человек. Из двух возможностей выбираем такую, которая займет меньше времени, и запоминаем это время в элементе D[2] специального массива. Таким образом, D[2] — минимальное время покупки билетов очередью из 2 первых человек.

Добавим в очередь третьего человека. Какие варианты есть у нас? Если третий человек покупает билет самостоятельно, то первые два, очевидно, не зависят от него. Тогда (внимание!) мы уже решили эту задачу! Мы только что нашли минимальное время покупки билетов для очереди из двух человек, оно хранится в D[2]. Общее время покупки в этом случае составит D[2] + A_3. Допустим, второй и третий человек решат купить билеты вместе. Ответом в этом случае будет $B_2 + A_1$ (два билета покупает второй человек, и первый человек покупает билет самостоятельно). Наконец, договориться смогут и все три человека. Тогда они купят билеты за C_1 — именно столько времени займет покупка трех билетов первым стоящим в очереди человеком. Запишем лучший вариант в D[3] — это будет минимальное время покупки билетов очередью из 3 первых человек.

Рассмотрим все вышесказанное на примере из условия.

$N = 5$

i	A_i	B_i	C_i
1	5	10	15
2	2	10	15
3	5	5	5
4	20	20	1
5	20	1	1

Заполним массив D начальными значениями, D[1] = A_1, остальные элементы пока неизвестны.

D[1]	D[2]	D[3]	D[4]	D[5]
5	???	???	???	???

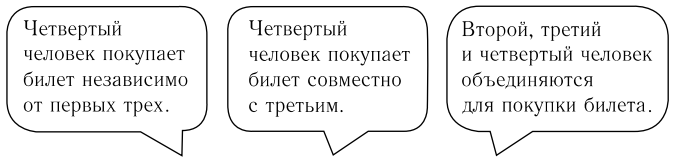
Переходим к выяснению значения D[2]. Имеем, что время покупки билетов раздельно составит $A_1 + A_2 = 5 + 2 = 7$, а при покупке вместе — 10. Итак, D[2] = min(7, 10) = 7.

D[1]	D[2]	D[3]	D[4]	D[5]
5	**7**	???	???	???

Добавляем в очередь третьего человека. Если он покупает билет отдельно, то общее время равно: D[2] + $A_3 = 7 + 5 = 12$. Если второй и третий договариваются между собой, то общее время будет: D[1] + $B_2 = 5 + 10 = 15$. Наконец, в случае совместной покупки билетов тремя любителями мюзиклов, им удастся это сделать за время $C_1 = 15$. Разумеется, выбираем самый быстрый вариант, и в данном случае D[3] = min(12, 15, 15) = 12.

D[1]	D[2]	D[3]	D[4]	D[5]
5	7	**12**	???	???

Дальше будем действовать совершенно аналогично. Добавляем к очереди четвертого человека. Выпишем формулу для D[4].



D[4] = min(D[3] + A_4, D[2] + B_3, D[1] + C_2).

Итак, D[4] = min(12 + 20, 7 + 5, 5 + 15) = min(32, 12, 20) = 12.

D[1]	D[2]	D[3]	D[4]	D[5]
5	7	12	**12**	???

Продолжая рассуждения аналогично, в общем виде получаем следующую формулу:

$$\mathrm{D}[i] = \min(\mathrm{D}[i-1] + A_i,\ \ \mathrm{D}[i-2] + B_{i-1},\ \ \mathrm{D}[i-3] + C_{i-2}).$$

Ответом к задаче будет служить элемент массива D[N]. В нашем случае это D[5]. Дорешаем наш пример: $\mathrm{D}[5] = \min(12 + 20,\ 12 + 20,\ 7 + 5) = \min(32,\ 32,\ 12) = 12$.

D[1]	D[2]	D[3]	D[4]	D[5]
5	7	12	12	**12 — ответ задачи**

Принцип, используемый при решении этой задачи, называется *динамическим программированием*. В данном случае мы решаем полную задачу, постепенно увеличивая ее размерность и используя решения более маленьких «подзадач».

Приведем полное решение задачи на языке Паскаль.

```
const
   MaxN = 5000;
var
   n : word; {число человек в очереди}
   a, b, c : array [0 .. MaxN] of integer;
   d : array [0 .. MaxN] of longint;
   i : integer;

function min(a,b : longint) : longint;
begin
     if a<b then min:=a else min:=b;
end;

begin
     assign(input, 'b.in');
     reset(input);
     assign(output, 'b.out');
     rewrite(output);

     readln(n);
     for i := 1 to n do
         read(a[i], b[i], c[i]);

     d[0] := 0;
     d[1] := a[1];
     d[2] := min(a[1]+a[2], b[1]);
     for i := 3 to n do
         d[i] := min(d[i-1]+a[i], min(d[i-2]+b[i-1], d[i-3]+c[i-2]));

     writeln(d[n]);
     close(input);
     close(output);
end.
```

Автор задачи — В. А. Матюхин, автор разбора — С. В. Шедов.

Задача III-C. Вырезанные фигуры

Тема: задачи на разные темы, алгоритмы на графах — поиск в ширину.

Вначале рассмотрим решение первого пункта задачи. Необходимо по внутреннему формату представления данных сканера перевести их в удобный нам формат — таблицу. Заметим, что задачи, подобные данной, сплошь и рядом встречаются в реальной жизни.

Наш сканер является полностью задокументированным устройством, кроме того, каждое представление позволяет однозначно восстановить таблицу. Саму таблицу мы хранить в памяти не будем — памяти просто не хватит. Можно было бы организовать побитовое хранение матрицы, но это сопряжено с техническими проблемами, которых мы постараемся избежать при помощи эффективного алгоритма. Будем получать таблицу последовательно по строкам, считая по ходу количество белых клеток. Еще раз подчеркнем, что саму таблицу мы в памяти хранить не будем.

Сначала решим упрощенную задачу. Допустим, что у нас не бывает вертикальных полос, а есть только горизонтальные. Рассмотрим решение на примере. Пусть дан фрагмент исходной таблицы:

i\j	1	2	3	4	5	6	7	8	9	10	11	12
1					■	■	■	■	■			
2		■	■				■	■	■	■	■	

Она кодируется следующими пятью горизонтальными полосами:

```
1 полоса: 1 5 4 - начальная клетка полосы (1,5), ее длина равна 4.
2 полоса: 1 7 3
3 полоса: 2 2 2
4 полоса: 2 7 5
5 полоса: 2 11 1
```

Расставим в таблице цифры, соответствующие полосам, которые описывают данную клетку.

i\j	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1	1,2	1,2	2			
2		3	3				4	4	4	4	4,5	

Как видим, некоторые клетки могут описываться сразу несколькими полосами. Считываем из файла самую первую полосу и храним ее в переменных `newi`, `newj` и `d` — начальная клетка полосы и ее длина. Затем идем по матрице до тех пор, пока текущая клетка не станет равна начальной клетке первой полосы. Очевидно, что все пройденные до этого момента клетки — белые. Не забываем считать их.

Начнем писать программу, постепенно уточняя ее:

```
for i := 1 to m do
  begin
   for j := 1 to n do
    begin
         if (i = newi) and (j = newj) then
         begin
      ... {дошли до полосы, надо что-то делать}
         end else
         inc(counter); {увеличиваем счетчик белых клеток}
     end;
  end;
```

Для нашего примера мы пройдем четыре белые клетки, пока не поравняемся с первой полосой (`i=1, j=5, d=4`). Теперь обрабатываем полосу, в которую мы пришли. В переменной `maxj` будем хранить столбец, на котором кончается первая полоса. Очевидно, что `maxj=j+d-1=5+4-1=8`.

i\j	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1	1	maxj	2			
2		3	3				4	4	4	4	4,5	

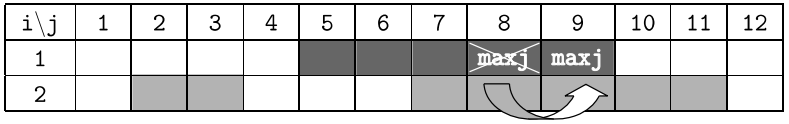
Таким образом, при дальнейшем проходе по первой строке, если номер столбца `j` будет меньше, чем `maxj`, то текущая клетка — черная, поскольку на ней лежит первая полоса.

На этом можно считать обработку первой полосы законченной. Поскольку эта полоса нам больше не понадобится, то считываем в переменные `newi, newj` и `d` следующую полосу. Так как по условию в каждой клетке начинается не более одной полосы, то обработку новой полосы мы проведем позже: когда придем в клетку, где она начинается. Обратите внимание, что в нашем алгоритме мы пользуемся тем фактом, что полосы в исходном файле задаются по строкам слева направо, иначе бы данный алгоритм был совершенно неприменим!

Вот как будет выглядеть текущая версия программы:

```
for i := 1 to m do
  begin
   for j := 1 to n do
    begin
         if (i = newi) and (j = newj) then
         begin
          maxj := j+d-1;
           {обновляем указатель на окончание черной полосы}
          readln(newi, newj, d); {читаем новую полосу}
         end;
         if (j > maxj) then inc(counter);
         {данная клетка белая, так как не покрывается
                               горизонтальной полосой}
     end;
  end;
```

Дальше мы пройдем две черные клетки (1, 5) и (1, 6), пока не поравняемся с началом второй полосы (`i=1, j=7, d=3`). Обработаем вторую полосу, т. е. определим, где она кончается: `maxj=j+d-1=7+3-1=9`. Указатель `maxj` сейчас указывает на столбец 8, а новая полоса заканчивается в 9 столбце, значит, указатель нужно передвинуть на 9, так как новая полоса простирается дальше прежней.



Такую проверку необходимо вставить в нашу программу:

```
         if maxj < j+d-1 then
            maxj := j+d-1;
```

Таким образом, мы подсчитываем белые клетки тогда, когда текущий столбец `j` не покрывается никакой горизонтальной полосой, т. е. `j>maxj`. Для первой строки мы найдем 7 белых клеток. А что необходимо сделать при переходе на новую строку? Разумеется — обнулить указатель `maxj`. Он был актуален только для предыдущей строки, а на новой строке будут уже новые полосы.

Вот как будет выглядеть уточненная версия нашей программы:

```
for i := 1 to m do
  begin
    for j := 1 to n do
```

```
      begin
        if (i = newi) and (j = newj) then
        begin
          if maxj < j+d-1 then
          maxj := j+d-1;
          readln(newi, newj, d);
        end;
        if (j > maxj) then inc(counter);
      end;
    maxj := 0;
    {получили очередную строку, переходим на следующую}
  end;
```

Теперь переходим к обработке второй строки. Как только мы поравняемся с началом очередной полосы, указатель `maxj` сместится на столбец 3, и пока j не превысит это значение, белые клетки считаться не будут. Затем мы пройдем три белые клетки и дойдем до четвертой полосы, и здесь опять указатель изменит свое значение. В отличие от предыдущей строки он не удлиняет текущую полосу, а указывает на конец новой полосы, но сути дела это не меняет.

i\j	1	2	3	4	5	6	7	8	9	10	11	12
1					1	1	1,2	1,2	2			
2		3	maxj				4	4	4	4	maxj	

Обратите внимание, что когда мы дойдем до пятой полосы, то указатель `maxj` обновлен не будет, так как пятая полоса заканчивается одновременно с четвертой.

Вернемся к исходной задаче и добавим вертикальные полосы. Поскольку мы, как и раньше, будем производить проход по строкам, то вертикальная полоса, которая началась в какой-то предыдущей строке, может влиять на текущую.

Рассмотрим пример:

i\j	1	2	3	4	5	6
1		1			2	2
2		1				
3		1,3		4		5
4		3		4		

Полосы 1, 3 и 4 — вертикальные, 2 и 5 — горизонтальные (для полосы в одну клетку на самом деле это не существенно). Когда мы будем обрабатывать вторую строку, то последней считанной полосой будет вторая, и мы уже не будем «помнить» информацию о первой полосе. Тем не менее, первая полоса дает на второй строке одну черную клетку и мы должны ее учесть.

Для этого заведем массив X из N элементов, где в $X[j]$ записываем, в какой строке заканчивается максимальная из просмотренных ранее вертикальных полос j-го столбца. Очевидно, что в процессе работы программы элементы этого массива могут изменяться. Например, после обработки первой полосы $X[2] = 3$, а после обработки третьей значение $X[2]$ будет изменено на 4. В четвертом столбце у нас только одна вертикальная полоса, значение $X[4] = 4$ изменено не будет.

Таким образом, если `i<=x[j]`, то текущая клетка покрывается какой-то вертикальной полосой, рассмотренной ранее. Изменим проверку того, является ли текущая клетка белой:

```
if (i > x[j]) and (j > maxj) then inc(counter);
```

Программа претерпит небольшие изменения. Приведем ее полный текст:

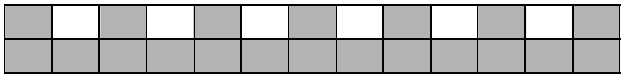
```
const
   MaxN = 4000;
var
   p : byte; {пункт задачи}
   m, n : word; {размер листа}
   k, ck : longint;
      {общее количество полос, количество считанных полос}
   newi, newj : word; {начальная клетка очередной полосы}
   dir : byte; {направление полосы}
   d : word; {длина полосы}
   x : array [1..MaxN] of integer;
      {массив для текущего состояния вертикальных полос}
   maxj : word;
      {максимальное окончание полосы в текущей горизонтали}
   i, j : word;
   counter : longint; {счетчик вырезанных клеток}

procedure init;
begin
   assign(input, 'c.in');
   reset(input);
   fillchar(x, sizeof(x), 0);
   readln(p, m, n, k);
   readln(dir, newi, newj, d);
   ck := 1;
```

```
   counter := 0;
   maxj := 0;
end;

procedure print;
var
   i, j : integer;
begin
    assign(output, 'c.out');
    rewrite(output);
    writeln(counter);
    close(output);
    close(input);
end;

procedure PutNewLine(dir, i, j, d : word; var maxj : word);
begin
      if (dir = 0) then {горизонтальное направление}
       begin
           if maxj < j+d-1 then
              maxj := j+d-1;
  {обновляем окончание горизонтальной полоски в текущей строке}
       end
      else {вертикальное направление}
       begin
           if x[j]<i+d-1 then
              x[j] := i+d-1;
  {обновляем окончание вертикальной полоски в данном столбце}
       end
end;

procedure solve; begin
   for i := 1 to m do
    begin
     for j := 1 to n do
      begin
           {если текущая клетка - начало новой полосы}
           if (i = newi) and (j=newj) then
           begin
              {обработаем эту полосу}
              PutNewLine(dir, i, j, d, maxj);
              {есть ли еще полосы для считывания}
              if ck<k then
               begin
                 readln(dir, newi, newj, d);
```

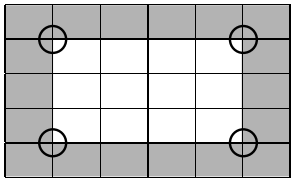


```
                 inc(ck)
               end
           end;
           if (i > x[j]) and (j > maxj) then inc(counter);
           {данная клетка белая, так как не покрывается ни
           вертикальной, ни горизонтальной полосой}
       end;
     maxj := 0; {получили очередную строку, переходим на следующую}
    end;
end;

begin
   init;
   solve;
   print;
end.
```

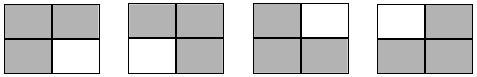
Перейдем к решению второго пункта задачи. Мы произвели перевод данных из внутреннего формата сканера в таблицу, но саму таблицу, как уже отмечалось выше, в памяти мы хранить не можем. Поэтому необходимо подсчитывать число фигур «на лету», имея в распоряжении одну или несколько строк. Можно применить один из известных алгоритмов послойного связывания, но наряду со сложностями в реализации он обладает недостатком — низкой производительностью на примерах, где области постоянно приходится связывать заново, например:



Существует алгоритм намного проще! Допустим, Алеша вырезал прямоугольник.



Назовем внешними углами прямоугольника шаблоны вида:

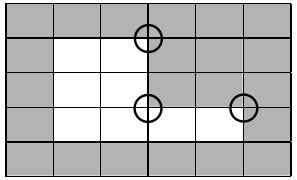


Поскольку фигуры не соприкасаются между собой, в том числе по диагонали, то существует всего 4 вида внешних углов. Кроме того, у каждого

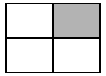
прямоугольника ровно четыре внешних угла. Если бы Алеша вырезал только прямоугольники, то достаточно было бы посчитать количество внешних углов, а затем разделить их количество на четыре.

Что меняется, когда Алеша вырезает не только прямоугольники?

Рассмотрим следующую фигуру:

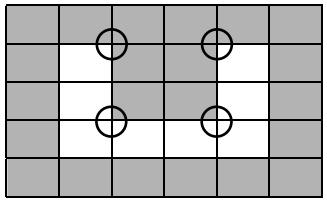


У фигуры появился один внутренний угол:



Очевидно, что внутренние углы — инвертированные шаблоны внешних углов. Количество внешних углов увеличилось на 1 — исчезает один старый внешний угол и появляются два новых.

Если же Алеша сделал внутренний вырез из прямоугольника:



Тогда у фигуры появляются два внутренних угла, но появляются и два новых внешних угла! Легко заметить, что, какие бы вырезы не делал Алеша, разность между количеством внешних углов и количеством внутренних углов остается постоянной. Это — инвариант. Поэтому число фигур:

$$F = \frac{\mathrm{Out} - \mathrm{In}}{4},$$

где Out — количество внешних углов всех фигур, In — количество внутренних углов всех фигур.

Реализовать этот способ в программе очень несложно. В каждый момент времени необходимо хранить только две строки таблицы и сравнивать с шаблоном квадраты размером 2×2.

Автор задачи и разбора — С. В. Шедов.

Задача III-D. Квадрат

Тема: задачи для начинающих.

Это задача на идею. Когда ее знаешь, то решение кажется очевидным. Однако придумать такое решение самому иногда (а точнее, даже очень часто) не так-то просто. Раскроем секрет задачи: достаточно сгенерировать любой квадрат $K \times K$, содержащий ровно S единиц, а затем просто заполнить такими квадратами квадрат $N \times N$.

Теперь проделаем все вышесказанное более аккуратно и подробно.

Шаг первый. Необходимо получить любой квадрат размером $K \times K$, в котором будет S единиц. Сделаем это каким-либо простым способом. Например, сначала заполним квадрат нулями. Затем будем проходить его по строкам и ставить единицы до тех пор, пока не поставим столько, сколько нам нужно. Если за полный проход по квадрату нам так и не удалось поставить S единиц, то это значит, что задача не имеет решения. Такой случай возможен только при $S > K^2$, а это противоречит условию.

Приведем функцию на языке Паскаль, которая генерирует необходимый квадрат. Функция `gen` получает размер квадрата `k`, необходимое число единиц в нем `s` и массив для записи результата. Функция возвращает `true`, если удалось сгенерировать квадрат, отвечающий нашим требованиям, и `false` в противном случае:

```
const
   MaxK=100;

type
   Square = array [1..MaxK, 1..MaxK] of byte;
   {квадрат KxK, повторением которого получаем ответ}

function gen(k, s : word; var a : Square) : boolean;
var
   i, j : integer;
   CurS : word; {сколько единиц уже удалось поставить в квадрате KxK}
begin
   CurS:=0;
   fillchar(a, sizeof(a), 0);   {вначале заполняем квадрат нулями}
   for i := 1 to k do
     for j := 1 to k do
        if CurS < s then
        {если число поставленных единиц меньше необходимого}
         begin
            a[i, j] := 1;  {то ставим очередную единицу}
            inc(CurS);
         end
     end;
```

```
    if CurS < s then gen := false else gen := true;
end;
```

Шаг второй. Распространим полученный квадрат размером $K \times K$ на искомый квадрат $N \times N$. Сначала рассмотрим, почему это действительно приводит к правильному результату.

Пусть $N = 7$, $K = 3$, $S = 4$. Приведенная выше функция **gen** получит следующий квадрат:

1	1	1
1	0	0
0	0	0

Назовем его образцом и заполним им квадрат размером 7×7:

i\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Обратите внимание, что в заполненном таким образом квадрате любой подквадрат размером 3×3 имеет сумму, равную четырем. Почему? Рассмотрим различные подквадраты 3×3 и проследим, что происходит с их суммой. Подквадрат с левым верхним углом (1, 1) является образцом, которым мы заполняли большой квадрат, поэтому его сумма, разумеется, равна четырем. Сдвинемся вправо и посмотрим на подквадрат с левым верхним углом в ячейке (1, 2).

i\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Первый и второй столбец нового квадрата принадлежат и старому. А что же приобретенный в результате сдвига новый столбец? Он тоже был в старом, поскольку 4-й столбец большого квадрата заполняется тем же самым квадратом-образцом! Выделенный квадрат можно свести к образцу, поменяв столбцы — третий на место первого, а первый и второй сдвинуть вправо. А поскольку от перемены мест столбцов сумма элементов квадрата не меняется, то и новый квадрат имеет требуемую сумму.

Теперь рассмотрим квадрат, сдвинутый на одну строку вниз. Сдвигаясь вниз, мы целиком «потеряли» первую строчку квадрата-образца, но и снова «приобрели» ее! Новый квадрат тоже легко формируется из квадрата-образца, но к перестановке столбцов необходимо добавить перестановку строк.

i\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Легко видеть, что как бы мы не сдвигали подквадрат, мы всегда теряем те же самые значения ячеек, что и приобретаем в результате сдвига! Это относится и к случаю, когда мы «упираемся» в границы квадрата $N \times N$.

i\j	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	0	0	1	0	0	1
3	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1
5	1	0	0	1	0	0	1
6	0	0	0	0	0	0	0
7	1	1	1	1	1	1	1

Реализовать заполнение квадрата $N \times N$ в программе предельно просто. Приведем ключевой фрагмент программного кода:

```
if gen(k, s, a) then      {генерируем образец}
   for i := 1 to n do     {заполняем образцом квадрат размером N*N}
    begin                 {результат выводим сразу в файл}
        for j := 1 to n do
```

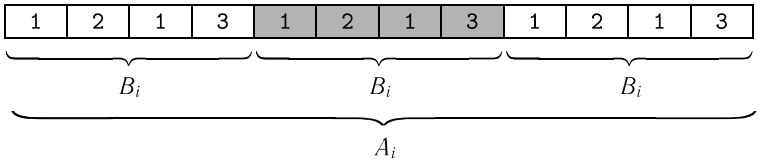
```
            write(a[(i-1) mod k+1,(j-1) mod k+1], ' ');
        writeln;
end;
```

Автор разбора — С. В. Шедов.

Задача III-E. Поле чудес

Тема: перебор, алгоритмы на строках.

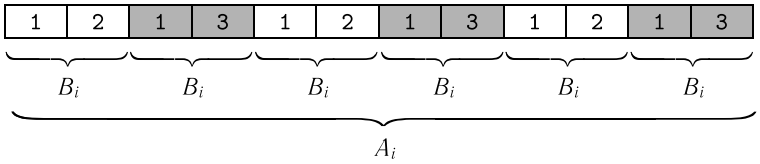
Отбросим последнее число, которое сказал ведущий, и сформулируем задачу по-другому: дана последовательность чисел A_i длиной N. Найти подпоследовательность B_i минимальной длины, повторением которой получена исходная последовательность.



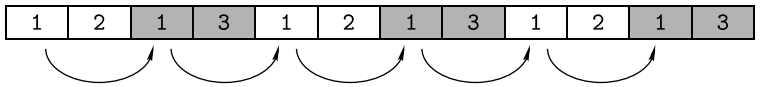
Заметим, что такая последовательность B_i всегда существует: по крайней мере, это сама исходная последовательность A_i (повторенная один раз).

Ограничения задачи дают возможность решать задачу перебором с некоторыми отсечениями. Пусть последовательность A_i длиной N была получена повторениями последовательности B_i длиной K. Поскольку последовательность B_i уложилась в последовательности A_i целое число раз, то N делится на K. Кроме того, K может изменяться в пределах от 1 до N. Будем проверять, является ли K делителем N, и если да, образована ли A_i из B_i. Сделать это можно несколькими способами.

Рассмотрим пример: $N = 12$, $K = 2$. Проверяем, образована ли последовательность A_i повторениями из последовательностей длиной 2. Для этого разобьем A_i на $N/K = 6$ частей и проверим, что получившиеся таким образом подпоследовательности B_i одинаковы.



Сначала «по цепочке» проверим, что все первые элементы последовательностей B_i равны.



В программе это будет выглядеть так:

```
j := 1;
while (j <= n-k) and (a[j] = a[j+k]) do inc(j, k);
```

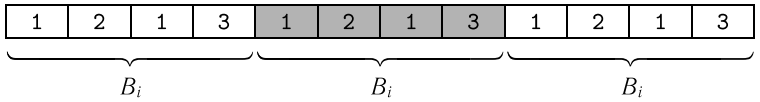
Затем проверим на равенство все вторые элементы. В нашем примере мы сразу получим неравенство ($2 \neq 3$), поэтому повторением последовательности длиной 2 исходная последовательность получена не была.

В общем случае необходимо проверить «по цепочке» на равенство все элементы последовательностей B_i от 1 до K.

Данный алгоритм реализуется следующим образом на языке Паскаль:

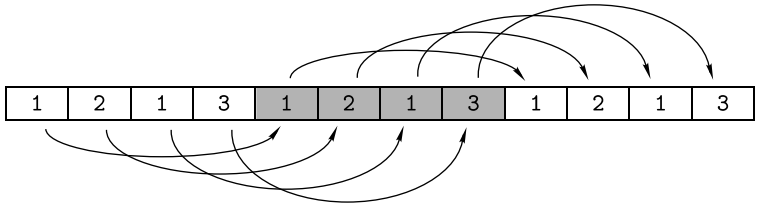
```
res := 0;
for k := 1 to n do
            {перебираем все возможные длины в порядке возрастания}
  if n mod k=0 then
   begin
     Ok := true;
     for i := 1 to k do
     begin
       j := i;
       while (j <= n-k) and (a[j] = a[j+k]) do inc(j, k);
       if j <= n-k then Ok := false;  { текущая проверка не прошла}
     end;
     if Ok then         { все проверки прошли успешно,
       begin                  значит последовательность B найдена }
         res := k;
         break;
       end;
   end;
```

Можно было поступить по-другому. Рассмотрим на примере той же последовательности: $N = 12$, $K = 4$. Разбиваем A_i на $N/K = 3$ части и опять проверяем, совпадают ли все получившееся после такого разбиения подпоследовательности B_i.



Для этого будем последовательно проверять, равен ли каждый элемент первой подпоследовательности соответствующему элементу второй,

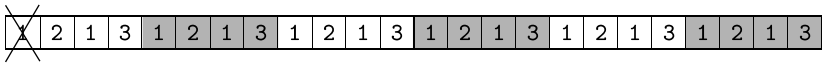
каждый элемент второй подпоследовательности — соответствующему элементу третьей и т. д. Здесь мы пользуемся свойством транзитивности равенства, т. е. тем, что из равенств `a=b` и `b=c` следует, что `a=c`.



Такой способ запрограммировать еще проще. Мы последовательно проходим по последовательности A_i (счетчик `j`) и каждый раз сравниваем `a[j]` и `a[j+k]`.

```
res := 0;
  for k := 1 to n do
                {перебираем все возможные длины в порядке возрастания}
    if n mod k=0 then
     begin
       j := 1;
       while (j <= n-k) and (a[j] = a[j+k]) do inc(j);
       if j > n-k then
         begin
   {все проверки прошли успешно, значит последовательность B найдена}
           res := k;
           break;
         end;
     end;
```

Заметим, что отсечение `n mod k=0` является очень эффективным. Например, число 30000 имеет всего 50 делителей, и мы проверяем 50 длин подпоследовательностей B_i, а не 30000. Однако при больших ограничениях, например $N > 10^6$, переборное решение может уже не успевать находить ответ за заданное время. Для таких случаев задача имеет более красивое решение. Запишем исходную последовательность A_i два раза подряд и выкинем из получившейся последовательности первое число.



В построенной таким образом последовательности будем искать первое вхождение последовательности A_i. Индекс найденного вхождения и будет минимальной длиной последовательности B_i (подумайте, почему это так!).

Заметим, что мы всегда найдем вхождение A_i, так как вторая половина построенной последовательности — это A_i.

Таким образом, задача свелась к нахождению подстроки в строке. Известны алгоритмы поиска подстроки с линейным относительно длины строки временем исполнения, например, алгоритм Кнута—Морриса—Пратта. Подробно об этом алгоритме можно прочитать, например, в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ» (М.: МЦНМО, 2000).

Автор разбора — С. В. Шедов.

Задача III-F. Детская игра со спичками

Тема: алгоритмы на графах — алгоритм Флойда.

Прежде всего заметим, что спички могут пересекаться только концами либо серединами. Других случаев пересечения быть не может. Таким образом, если разбить каждую спичку на две «половинки» вдвое меньшей длины, то полученные «полуспички» пересекаться будут только концами. Если все координаты исходных спичек умножить на два, то координаты всех «полуспичек» также будут выражаться целыми числами. Далее, если на два умножить также и время горения всех спичек, то время горения «полуспички» также будет выражаться целым числом секунд. Будем считать, что мы так уже поступили, и далее спичками именуются именно такие «полуспички».

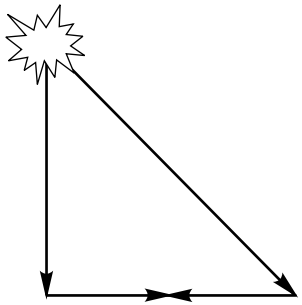
Мы перешли к аналогичной задаче с вдвое бóльшим количеством спичек с целыми координатами, пересекающихся только концами. Построим граф, ребрами которого будут спички, а вершинами — их концы. Задача сводится к нахождению такой вершины графа, при «поджигании» которой весь граф сгорает за минимальное время.

Будем решать задачу перебором по всем потенциальным «точкам поджигания». Так как в любом случае необходимо в выходной файл вывести кроме координат оптимальной точки общее время сгорания, задачу о нахождении времени сгорания графа при «поджигании» данной вершины тоже надо уметь решать.

Пусть нам дан граф, в котором на каждом ребре записано время сгорания соответствующей ему спички (эту величину будем называть весом или длиной ребра), и в нем зафиксирована некоторая вершина. Как найти время сгорания этого графа при «поджигании» этой вершины?

Ясно, что время сгорания графа равно расстоянию от зафиксированной вершины до наиболее удаленной от нее точки графа. Именно «наиболее удаленной точки», а не «наиболее удаленной вершины»! Простейший пример, где эти понятия различаются — треугольник (см. рис.).

Допустим, в приведенном выше примере время горения вертикальной и диагональной спичек — одна секунда, а время горения горизонтальной спички — четыре секунды. Тогда при поджигании этой фигуры в верхней точке через секунду огонь достигнет обоих концов основания. Еще две секунды потребуется пламени, чтобы сжечь основание. Таким образом, хотя самая удаленная вершина находится на расстоянии 1 от точки поджигания, суммарное время сгорания такой фигуры равно трем секундам.



Вычислим кратчайшие расстояния от данной вершины до всех остальных. Кратчайшее расстояние соответствует моменту времени, когда огонь достигнет данной вершины. Для нахождения расстояний можно использовать, например, алгоритм Флойда.

Алгоритм Флойда находит кратчайшие расстояния между всеми парами вершин в графе, делая число операций, пропорциональное кубу числа вершин графа. Программа, реализующая алгоритм Флойда, выглядит следующим образом:

```
for k:=1 to N do
  for i:=1 to N do
    for j:=1 to N do
      if a[i,j]<a[i,k]+a[k,j] then a[i,j]:=a[i,k]+a[k,j];
```

Остановимся на описании алгоритма Флойда более подробно. Пусть в матрице $A[i, j]$ записаны длины ребер графа (элемент $A[i, j]$ равен весу ребра, соединяющего вершины с номерами i и j, если же такого ребра нет, то в соответствующем элементе записано некоторое очень большое число, например 10^9). Построим новые матрицы $C_k[i, j]$ ($k = 0, \ldots, N$). Элемент матрицы $C_k[i, j]$ будет равен минимально возможной длине такого пути из i в j, что в качестве промежуточных вершин на этом пути используются вершины с номерами от 1 до k. Таким образом, рассматриваются пути, которые могут проходить через вершины с номерами от 1 до k (а могут и не проходить через какие-то из этих вершин), но заведомо не проходят через вершины с номерами от $k + 1$ до N. В матрицу записывается длина кратчайшего из таких путей. Если таких путей не существует, записывается то же большое число, которым обозначается отсутствие ребра.

Сформулируем следующие факты.

В матрице $C_0[i, j]$ записаны длины путей, которые не содержат ни одной промежуточной вершины. Таким образом, матрица $C_0[i, j]$ совпадает с исходной матрицей $A[i, j]$.

В матрице $C_N[i, j]$ записаны минимальные длины путей, которые в качестве промежуточных вершин используют все вершины графа — т. е. длины кратчайших путей, которые мы хотим получить.

Если у нас уже вычислена матрица $C_{k-1}[i, j]$, то элементы матрицы $C_k[i, j]$ можно вычислить по следующей формуле: $C_k[i, j] := \min(C_{k-1}[i, j], C_{k-1}[i, k] + C_{k-1}[k, j])$. В самом деле, рассмотрим кратчайший путь из вершины i в вершину j, который в качестве промежуточных вершин использует только вершины с номерами от 1 до k. Тогда возможны два случая:

1. Этот путь не проходит через вершину с номером k. Тогда его промежуточные вершины — это вершины с номерами от 1 до $k - 1$. Но тогда длина этого пути уже вычислена в элементе $C_{k-1}[i, j]$.

2. Этот путь проходит через вершину с номером k. Но тогда его можно разбить на две части: сначала мы из вершины i доходим оптимальным образом до вершины k, используя в качестве промежуточных вершины с номерами от 1 до $k - 1$ (длина такого оптимального пути вычислена в $C_{k-1}[i, k]$), а потом от вершины k идем в вершину j опять же оптимальным способом, и опять же используя в качестве промежуточных вершин только вершины с номерами от 1 до k ($C_{k-1}[k, j]$).

Выбирая из этих двух вариантов минимальный, получаем $C_k[i, j]$.

Последовательно вычисляя матрицы C_0, C_1, C_2 и т. д., мы и получим искомую матрицу C_N кратчайших расстояний между всеми парами вершин в графе.

Заметим теперь, что при вычислении очередной матрицы C_k нам нужны лишь элементы матрицы C_{k-1}, поэтому можно не хранить в памяти N таких матриц, а обойтись двумя — той, которую мы сейчас вычисляем, и той, которую мы вычислили на предыдущем шаге. На самом деле оказывается, что даже это излишне — все вычисления можно производить в одной матрице (подумайте, почему).

Вернемся к решению нашей задачи.

После нахождения кратчайших путей из «поджигаемой» вершины во все остальные, нам известно время, за которое огонь достигнет каждой из вершин. Теперь нужно проверить все ребра-спички на предмет того, сгорели ли они уже в процессе перемещения огня до вершин, а если нет, то нужно найти время, когда догорит данное ребро. Максимальное из этих времен даст ответ.

Пусть огонь достигает концов спички со временем сгорания L в моменты времени $T1$ и $T2$. Если $T1 = T2 + L$ или $T2 = T1 + L$, то огонь передавался по этой спичке и максимум из $T1$ и $T2$ и будет тем временем, к которому спичка сгорит полностью. Отметим, что разность между $T1$ и $T2$ не может быть больше L (подумайте, почему).

Пусть теперь разность между $T1$ и $T2$ не равна L. Это значит, что к разным концам спички огонь подошел разными путями, она будет гореть одновременно с обеих сторон и догорит где-то посередине. Напомним, что под спичкой мы понимаем половину спички, и пересекаться не в концах она уже ни с чем не может. Значит поджечь такая спичка никакую другую спичку не может — с обоих ее концов уже и так горит огонь! В простейшем случае, если спичка подожжена одновременно с обоих концов, она сгорает за время $L/2$. Трудность в том, что в общем случае время возгорания концов спички может не совпадать.

Можно вычесть одно и то же число из $T1$ и из $T2$, т. е. мы можем перейти к задаче, где один конец загорается в момент времени 0, а второй — в момент времени T. Общее время сгорания спички в таком случае будет равно $T + (L - T)/2$. Прийти к этой формуле проще всего исходя из следующих соображений. Пусть спичка имеет длину L сантиметров и горит со скоростью 1 сантиметр в секунду. Тогда первые T секунд она будет гореть с одного конца и сгорит на T сантиметров, а оставшееся время потребуется на то, чтобы сжечь $L - T$ сантиметров со скоростью 2 см/сек, т. е. время сгорания этого куска спички будет равно $(L - T)/2$. При $T = 0$ формула дает ответ $L/2$, а при $T = L$ — ответ L, что полностью согласуется с условием задачи.

Мы теперь умеем решать задачу о нахождении времени сгорания данной фигуры из спичек при ее поджигании в данной точке. Для этого нужно в соответствующем данной фигуре графе найти максимум из времен «догораний» каждого из ребер. И не забыть разделить его пополам — ведь при построении графа мы удвоили время сгорания каждой из спичек.

Теперь мы легко можем решить задачу перебором по вершинам. Проверив все потенциальные точки поджога и выбрав из них ту, при поджоге которой время сгорания минимально, мы найдем ответ. Необходимо учесть, что не каждая из вершин достроенного графа может быть точкой «поджигания». Так как мы раздвоили каждую спичку, в наш граф войдут также вершины, соответствующие серединам спичек. Из всех точек мы должны выбрать те, координаты которых нацело делятся на два.

Еще одно замечание — в данной задаче мы всюду работали с целыми числами. Но в двух местах это целое число делилось на два — при нахождении координат пересечения спичек и при вычислении времени сгорания спички, подожженной с обеих сторон. Из этого следует, что общее время сгорания можно представить в виде $X/4$, где X — целое число. Соответственно, дробная часть этого времени будет равна 0,0, 0,25, 0,5 или 0,75, и двух десятичных знаков для ее представления вполне достаточно.

Приведем полный текст программы:

```
Program Matches;

Const
  TaskID='f';
  InFile=TaskID+'.in';
  OutFile=TaskID+'.out';

Const
  MaxN=42; { Ограничение на N }
  MaxG=2*MaxN+1; { Ограничение на число вершин в графе }
  Infinity=MaxLongInt; { "Бесконечное" расстояние }

Var
  N:Integer;                       {             }
  Match:Array[1..MaxN]Of Record    { Входные     }
    X1,Y1,X2,Y2:Integer;           {      данные }
    Time:LongInt;                  {             }
  End;                             {             }

  NG:Integer;                                          {      }
  Vertex:Array[1..MaxG]Of Record                       {      }
    X,Y:Integer;                                       { Граф }
  End;                                                 {      }
  Edge,Distance:Array[1..MaxG,1..MaxG]Of LongInt;      {      }

  Res:Extended;      { Минимальное время сгорания }
  ResX,ResY:Integer; { Оптимальная точка поджога  }

Procedure Load;
Var
  I:Integer;
Begin
  Assign(Input,InFile);
  ReSet(Input);
  Read(N);
  For I:=1 To N Do
    With Match[I] Do
      Read(X1,Y1,X2,Y2,Time);
  Close(Input);
End;

Function GetVertex(VX,VY:Integer):Integer;
  { Функция, возвращающая номер вершины с заданными координатами.
    При отсутствии нужной вершины она создается }
```

```
Var
  I:Integer;
Begin
  For I:=1 To NG Do
    With Vertex[I] Do
      If (X=VX) And (Y=VY) Then Begin
        GetVertex:=I;
        Exit;
      End;

  Inc(NG); { Если нужная вершина не найдена }
  With Vertex[NG] Do Begin
    X:=VX;
    Y:=VY;
    For I:=1 To NG-1 Do Begin
      Edge[I,NG]:=Infinity;
      Edge[NG,I]:=Infinity;
    End;
    Edge[NG,NG]:=0;
  End;
  GetVertex:=NG;
End;

Procedure AddEdge(X1,Y1,X2,Y2:Integer; Time:Longint);
  { Функция, добавляющая ребро между двумя точками }
Var
  A,B:Integer;
Begin
  A:=GetVertex(X1,Y1);
  B:=GetVertex(X2,Y2);
  Edge[A,B]:=Time;
  Edge[B,A]:=Time;
End;

Procedure BuildGraph; { Процедура построения графа }
Var
  I:Integer;
Begin
  NG:=0;
  For I:=1 To N Do
    With Match[I] Do Begin
      AddEdge(X1*2,Y1*2,X1+X2,Y1+Y2,Time);
      AddEdge(X1+X2,Y1+Y2,X2*2,Y2*2,Time);
    End;
End;
```



```
Procedure FindShortestPaths;
Var
  K,I,J:Integer;
Begin
  Distance:=Edge;
  For K:=1 To NG Do
    For I:=1 To NG Do If Distance[I,K]<Infinity Then
      For J:=1 To NG Do If Distance[K,J]<Infinity Then
        If Distance[I,K]+Distance[K,J]<Distance[I,J] Then
          Distance[I,J]:=Distance[I,K]+Distance[K,J];
End;

Function BurnAt(At:Integer):Extended;
  { Функция, вычисляющая время сгорания при поджоге в точке At }
Var
  I,J:Integer;
  Cur,ThisEdge:Extended;
Begin
  Cur:=0;
  For I:=1 To NG Do If Distance[At,I]>Cur Then Cur:=Distance[At,I];
  For I:=1 To NG Do
    For J:=I+1 To NG Do If Edge[I,J]<Infinity Then Begin
      If (Distance[At,I]<Distance[At,J]+Edge[I,J]) And
         (Distance[At,J]<Distance[At,I]+Edge[I,J]) Then Begin
        If Distance[At,I]<Distance[At,J] Then
          ThisEdge:=Distance[At,J]+
                    (Edge[I,J]-(Distance[At,J]-Distance[At,I]))/2
        Else
          ThisEdge:=Distance[At,I]+
                    (Edge[I,J]-(Distance[At,I]-Distance[At,J]))/2;
        If ThisEdge>Cur Then Cur:=ThisEdge;
      End;
    End;
  BurnAt:=Cur;
End;

Procedure Solve;
Var
  I:Integer;
  Cur:Extended;
Begin
  Res:=Infinity;
  For I:=1 To NG Do
    With Vertex[I] Do
```

```
      If Not Odd(X) And Not Odd(Y) Then Begin
        Cur:=BurnAt(I);
        If Cur<Res Then Begin
          Res:=Cur;
          ResX:=X Div 2;
          ResY:=Y Div 2;
        End;
      End;
End;

Procedure Save;
Begin
  Assign(Output,OutFile);
  ReWrite(Output);
  WriteLn(ResX,' ',ResY);
  WriteLn(Res/2:0:2);
  Close(Output);
End;

Begin
  Load;
  BuildGraph;
  FindShortestPaths;
  Solve;
  Save;
End.
```

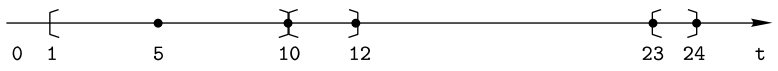
Автор задачи — В. А. Матюхин,
авторы разбора — Д. Н. Королев, П. И. Митричев.

Задача III-G. Реклама

Тема: сортировка.

При решении задачи всегда очень полезно представить какую-либо ее визуальную интерпретацию. Изобразим все время работы магазина временно́й осью, а время прихода и ухода покупателей — отрезками на этой оси. Теперь задачу можно переформулировать: поставить на оси минимальное количество точек с целочисленными координатами так, чтобы в каждом отрезке содержалось не менее двух точек.

Пример из условия в такой интерпретации будет выглядеть следующим образом:



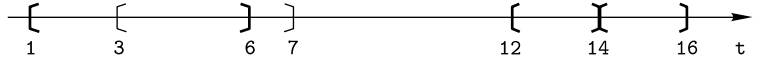
В программе будем использовать:

```
const
   MaxN=3000;
   Infinity=MaxLongInt; {"бесконечная" координата}

type
  TSegment = record
    left, right : longint;
  end;

var
  n : longint;                                {количество отрезков}
  segment : array [1..MaxN] Of TSegment;  {координаты отрезков}
  point : array [1..MaxN*2] Of longint;
                                  {точки, которые мы расставляем}
```

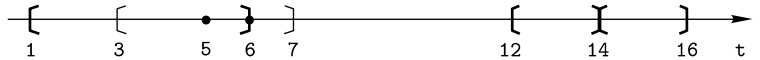
Отсортируем все отрезки по возрастанию правых границ, а при их равенстве — по убыванию левых.

Ограничение на количество отрезков ($N \leqslant 3000$) позволяет применить не только быструю сортировку, но и алгоритм сортировки с квадратичной временно́й сложностью, например метод «пузырька».

Перейдем теперь к основной части решения. Для этого рассмотрим следующий пример. Пусть дано четыре отсортированных отрезка: [1, 6], [3, 7], [12, 14], [14, 16].



В самом первом отрезке [1, 6] должны содержаться две точки. Их необходимо поставить как можно правее, т. е. в точках с целочисленными координатами 5 и 6. Почему? Поскольку это отрезок с наименьшей правой границей, то раньше него другие отрезки не могли закончиться. Но могли начаться другие отрезки! А чем правее мы располагаем точки, тем больше шанс, что они одновременно попадут и в другие отрезки — что нам выгодно. Еще раз обратим внимание на факт, что не существует более выгодной расстановки точек, чем данная: поскольку никакой другой отрезок раньше наших точек не заканчивается, то мы не упускаем ни одну потенциальную возможность улучшить расстановку точек.



В нашем примере поставленные две точки сразу попали и в отрезок [3, 7]. А вот если бы мы поставили точки, например, в координатах 1 и 2, то такая расстановка была бы неэффективной — мы покрыли бы ей только один отрезок, а не два.

Назовем точку 6 последней поставленной точкой, а точку 5 — предпоследней. В программе это запишется следующим образом:

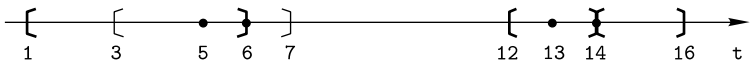
```
PrevLast := right-1;
Last := right;
```

где `right` — правая граница текущего отрезка.

Переходим к следующему отрезку [3, 7] — но внутри него уже стоят две точки, поскольку левая граница отрезка меньше, чем предпоследняя поставленная точка.

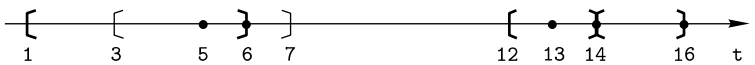
```
if left <= PrevLast then <ничего расставлять не нужно>;
```

Следующий отрезок [12, 14]. В нем не стоит еще ни одной точки, так как `left > last`. Из аналогичных соображений ставим в нем две точки как можно правее.



Последний отрезок [14, 16]. В нем уже содержится одна из поставленных точек, так как `Last = left`. Ставим еще одну точку на правую границу отрезка — координата равна 16. При этом предыдущей точкой станет точка 14.

```
PrevLast := Last;
Last := right;
```



Легко видеть, что такой алгоритм расстановки точек всегда дает оптимальный результат. Итак, для каждого отрезка мы смотрим, нужно ли поставить в нем одну или две точки, и если да, то ставим их как можно правее. Резюмируя все вышесказанное приведем ключевой фрагмент программного кода, реализующий данную логику:

```
procedure solve;
var
  Last, PrevLast : longint; {две последние поставленные точки}
  i : longint;
begin
  res := 0; {количество поставленных точек}
  Last := -Infinity;
  PrevLast := -Infinity;
  for i := 1 to N do
    with segment[i] do
      if last < left then   {необходимо поставить еще две точки}
        begin
          inc(res);
          point[res] := right-1;
          inc(res);
          point[res] := right;
          PrevLast := right-1;
          Last := right;
        end
       else
        if PrevLast < left then  {необходимо поставить еще одну точку}
          begin
            inc(res);
            point[res] := right;
            PrevLast := Last;
            Last := right;
          end;
end;
```

Авторы разбора — П. И. Митричев, С. В. Шедов.

Олимпиада V. Заочный тур 2004–2005 учебного года

Задача V-A. Автобусная экскурсия

Тема: задачи для начинающих.

В этой задаче нам нужно последовательно считывать из файла высоты мостов до тех пор, пока мы не встретим слишком низкий мост, либо пока не проверим все мосты. Поскольку высоту первого моста нам нужно считать в любом случае, воспользуемся циклом `repeat–until`:

```
read(N); {считываем количество мостов}
i:=0;    {номер текущего моста}

repeat
  read(h);
  inc(i);
until (h<=437) OR (i=n);
```

Мы можем выйти из цикла в двух случаях: либо встретив низкий мост, либо проверив все мосты. В зависимости от высоты последнего считанного моста, мы выводим ответ:

```
if h>437 then write ('No crash')
else write ('Crash ',i);
```

Автор разбора — В. М. Гуровиц.

Задача V-B. Сапер

Тема: задачи для начинающих.

Для того чтобы решить эту задачу, мы воспользуемся двумерным массивом `a`, представляющим игровое поле. Изначально мы заполним его нулями, а затем для каждой мины будем увеличивать на 1 числа во всех соседних с ней клетках. В саму же клетку, где стоит мина, мы будем записывать какое-нибудь большое число («бесконечность»). Вот как может выглядеть соответствующий фрагмент программы:

```
read(i,j); {читаем из файла координаты очередной мины}
inc(a[i-1][j-1]); inc(a[i-1][j]); inc(a[i-1][j+1]);
inc(a[i][j-1]); inc(a[i][j+1]);
inc(a[i+1][j-1]); inc(a[i+1][j]); inc(a[i+1][j+1]);
a[i][j]:=100;
```


Таким образом, когда мы обработаем все мины из входного файла, мы получим массив, в котором числа, не превосходящие 8, будут означать количество мин по соседству с данной клеткой, а числа от 100 и больше — клетки, в которых стоят мины. Останется лишь распечатать полученную схему:

```
for i:=1 to N do begin
  for j:=1 to M do
    if a[i][j]>8 then write('*') else
      if a[i][j]=0 then write('.') else
        write(a[i][j]);
  writeln;
end;
```

Осталось обсудить несколько тонкостей.

1. Мы никак отдельно не обрабатываем случай, когда мина стоит на границе поля и у нее меньше восьми соседних клеток. Чтобы упростить программу, мы воспользуемся стандартным трюком: увеличим поле со всех сторон на 1 ряд, т. е. будем использовать массив, индексы которого изменяются от 0 до $M + 1$ и от 0 до $N + 1$ соответственно. При этом в выходной файл мы выводим только нужную нам часть массива.

2. Код, который мы написали для увеличения чисел в клетках, не выглядит изящным. Можно заменить его, например, на такой:

```
for k:=-1 to 1 do
  for l:=-1 to 1 do
    inc(a[i+k][j+l]);
a[i][j]:=100;
```

Может показаться, что выбор из этих двух вариантов есть лишь вопрос вкуса, но на самом деле первый вариант таит в себе бóльшую опасность сделать опечатку, которую непросто будет найти.

Автор разбора — В. М. Гуровиц.

Задача V-C. Робот К-79

Тема: задачи для начинающих.

Решение этой задачи чем-то напоминает решение предыдущей. Представим себе, что робот ходит по большой шахматной доске, каждым ходом перемещаясь на одну клетку влево, вправо, вперед или назад. Мы будем хранить в двумерном массиве путь робота и после каждого очередного хода проверять, не попал ли робот в клетку, в которой он уже был раньше. Чтобы отслеживать передвижения робота, нам нужно в каждый момент

времени знать, в какую сторону он смотрит. Обозначим направления цифрами: 0 — направо, 1 — вперед, 2 — налево, 3 — назад. Тогда, если мы храним в переменной `dir` текущее направление движения, то поворот налево будет соответствовать прибавлению к `dir` единицы по модулю 4:

```
dir := (dir + 1) mod 4  {0->1, 1->2, 2->3, 3->0}
```

Поворот направо, аналогично, будет соответствовать вычитанию единицы по модулю 4.

Заведем два массива:

```
dx : array[0..3] of integer = (0,1,0,-1);
dy : array[0..3] of integer = (1,0,-1,0);
```

Их смысл в следующем: если `dir` — текущее направление движения робота, то за один ход его координата x изменяется на `dx[dir]`, а координата y на `dy[dir]`.

После этого несложно написать требуемую программу. Будем считать, что мы прочитали последовательность ходов робота в строковую переменную `S`.

```
i := 1; {номер анализируемого символа}
step := 0; {количество сделанных ходов S}
flag := false; {возвратились ли на пройденную ранее клетку}
x := 0; {текущее         }
y := 0; {положение робота}
was[0][0] := true; {робот начинает движение в клетке (0,0)}
repeat
 case s[i] of
    'L': dir := (dir + 1) mod 4;
    'R': dir := (dir - 1) mod 4;
    'S': begin
           inc(step);
           x := x + dx;
           y := y + dy;
           if was[x][y] then flag := true
              else was[x][y] := true;
    end;
    inc(i);
until (i>length(s)) OR flag;
if flag then write(step)
else write(-1);
```

Здесь, как и в задаче про мосты, после выхода из цикла мы должны выяснить, что послужило причиной выхода, и вывести либо количество сделанных ходов, либо -1.

В условии задачи сказано, что общее количество букв `S` во входном файле не превышает 50, поэтому нам достаточно завести массив

```
was : array [-50..50, -50..50] of boolean;
```

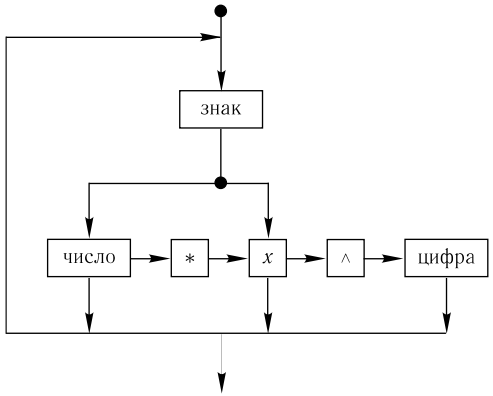
Для тех, у кого хватило терпения дочитать до этого места, откроем секрет: приведенное решение работать не будет! Дело в том, что операция `mod` в Турбо Паскале определена только для неотрицательных чисел, а в выражении `(dir - 1) mod 4` левый операнд может быть равен -1. Но это легко исправить. Достаточно прибавить к этому операнду 4: получится `(dir - 1 + 4) mod 4`. От этого значение выражения не изменится, но все числа станут положительными.

Автор разбора — В. М. Гуровиц.

Задача V-D. Многочлен

Тема: строки.

В этой задаче требуется проанализировать формулу и вычислить ее значение при заданном x. Составим схему, описывающую структуру данной формулы.



Может так случиться, что первое слагаемое в формуле не начинается со знака. Вместо того чтобы усложнять схему, мы исправим формулу:

```
if s[1]<>'-' then
  s := '+' + s;
```

Теперь можно разбирать формулу по схеме:

```
result := 0;
i := 1;       {номер текущей позиции в строке}
repeat
```

```
  coef:=0;    {коэффициент перед x}
  sign:=1;    {знак коэффициента}
  power:=0;   {степень x}
  {Знак}
  if s[i]='-' then sign:=-1;
  inc(i);
  if s[i]<>'x' then
    {коэффициент}
    repeat
      coef:=coef*10+(ord(s[i])-ord('0'));
      inc(i);
    until NOT((i<=length(s)) AND (s[i] in ['0'..'9']))
  else coef := 1;
  if (i<=length(s)) AND (s[i] = '*') then inc(i);  {пропускаем '*'}
  if (i<=length(s)) AND (s[i] = 'x') then begin
    inc(i);                                        {пропускаем 'x'}
    if(i<=length(s)) AND (s[i] = '^') then begin
      inc(i);                                      {пропускаем '^'}
      power := ord(s[i])-ord('0');
      inc(i);
    end
    else
      power := 1;
  end;
  result := result + sign*coef*xpow(power);
until i>length(s);
```

Написание функции `xpow(power)`, возводящей `x` в степень, мы оставляем читателю.

Жюри подготовило еще один сюрприз. Давайте подсчитаем, какое максимальное значение может принимать выражение во входном файле. В нем не более десяти слагаемых, каждое из которых не превышает $100 \cdot 100^4 = 10^{10}$. Таким образом, максимальное значение выражения равно 10^{11}. Такое число не поместится в 32-битный тип `longint`. Поэтому придется использовать тип `real`. При этом выводить требуется целое число, и функция округления не поможет, поскольку ее результат — тоже `longint`. В Турбо Паскале можно воспользоваться таким приемом:

```
write(result:0:0);
```

Второй ноль означает, что цифры после десятичной точки выводить не нужно.

Решение этой задачи в более общем случае описано в статье «Подсчет значения арифметического выражения методом рекурсивного спуска» в конце этой книги.

Автор разбора — В. М. Гуровиц.

Задача V-E. Головоломка

Тема: строки.

Заметим, что нам не важно, где именно находятся буквы, которые вычеркивает Петя. Поэтому не обязательно искать именно те буквы, которые составляют слова: мы можем вычеркивать такие же буквы из любой клетки таблицы. После этого наблюдения решение задачи уже становится делом техники. Приведем два способа решения задачи.

Первый способ. Воспользуемся стандартными функциями для работы со строками, чтобы сделать код лаконичным. Сначала мы запишем все символы из таблицы в одну строку (их не более 100, поэтому в Паскале можно воспользоваться стандартным типом `string`):

```
S:='';
for I:=1 to N do
begin
  ReadLn(V);
  S:=S+V;
end;
```

Теперь будем читать слова по одному и удалять каждую букву каждого слова из строки `S`:

```
for J:=1 to M do
begin
  ReadLn(V);
  for I:=1 to Length(V) do
    Delete(S, Pos(v[i], S), 1);
end;
```

Напомним, что функция `Pos` возвращает позицию первого вхождения символа в строку, а процедура `Delete` удаляет из данной строки символ (в общем случае — подстроку) в данной позиции. После выполнения этого фрагмента программы в строке `S` останутся ровно те символы, которые нам требуется вывести:

```
write(s);
```

Второй способ. Заведем массив:

```
a: array ['A'..'Z'] of integer;
```

Подсчитаем, сколько раз встречается каждая буква латинского алфавита в исходном квадрате:

```
for i:=1 to n do begin
  for j:=1 to n do begin
```

```
    read (c);
    inc(a[c])
  end;
  readln;
end;
```

Теперь будем считывать слова и «вычеркивать» встречающиеся в них буквы, уменьшая соответствующие счетчики:

```
for i:=1 to m do begin
  readln (s);
  for j:=1 to length(s) do
    dec(a[s[j]])
end;
```

Теперь напечатаем оставшиеся буквы:

```
for c:='A' to 'Z' do
  for i:=1 to a[c] do
    write (c);
```

Автор разбора — В. М. Гуровиц.

Задача V-F. Поиск прямоугольников

Тема: задачи на разные темы.

Для того чтобы найти, скажем, координату x левого нижнего угла прямоугольника из единиц, достаточно найти номер самого левого столбца, в котором встречается число 1. Это можно сделать, например, следующим образом. Пусть `a[i][j]` — число в клетке на пересечении i-й строки и j-го столбца таблицы, `r[k]` — координата x прямоугольника с номером k.

```
for j:=M downto 1 do
  for i:=1 to N do
    r[a[i][j]]:=j;
```

Разберем подробнее этот лаконичный код. Здесь мы двигаемся справа налево по столбцам и, встречая некоторое число `a[i][j]` в таблице, записываем в соответствующий элемент массива `r` номер столбца, в котором встречается соответствующее число. В конце концов, в каждом элементе `r[k]` будет записан номер самого последнего (т. е. самого левого) столбца, в котором встретилось число k.

Заметим, что мы обошлись без условных операторов. Но для этого нам пришлось хранить в памяти всю таблицу. Покажем теперь, как можно обрабатывать элементы таблицы по мере их считывания из файла. Для этого нам, увы, придется пожертвовать лаконичностью кода (зато он станет более очевидным).


```
for i:=1 to N do
  for j:=1 to M do begin
    read(t);
    if R[t]>j then R[t]:=j;
    if H[t]<j then H[t]:=j;
    if C[t]>i then C[t]:=i;
    if W[t]<i then W[t]:=i;
  end;
```

Автор разбора — В. М. Гуровиц.

Задача V-G. Разноцветные треугольники

Тема: игры и стратегии.

Алгоритм.

Будем перебирать все вершины от $(N-1)$-й до первой. Для i-й вершины будем перебирать отрезки $[i, i+1]$, $[i, i+2]$, …, $[i, i+N]$ и красить те из них, которые есть в нашей конфигурации, причем каждый следующий отрезок красить в цвет, противоположный цвету предыдущего покрашенного отрезка.

Доказательство правильности работы.

Пусть в нашей конфигурации присутствует треугольник с вершинами i, j, k $(i < j < k)$, внутри которого нет других отрезков. Тогда наш алгоритм покрасит отрезки $[i, j]$ и $[i, k]$ друг за другом, и потому — в разные цвета.

Автор задачи и разбора — В. М. Гуровиц.

Задача V-H. Побег с космической станции

Тема: алгоритмы на графах — поиск в ширину.

Построим граф, вершинами которого будут клетки, не занятые стенами. Соединим ребрами соседние клетки, а также клетки, соединенные гипертуннелями (поскольку гипертуннель работает только «в одну сторону», граф придется сделать ориентированным). Теперь осталось отождествить все вершины, соответствующие выходам (т. е. считать их одной вершиной). В построенном графе можно найти кратчайший путь, пользуясь, например, волновым алгоритмом (поиск в ширину).

Автор разбора — В. М. Гуровиц.

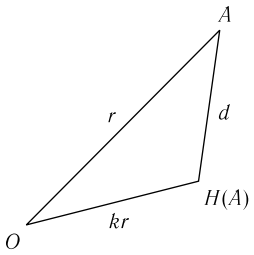
Задача V-I. Неподвижная точка карты

Тема: геометрия.

Известно, что для любой пары подобных фигур на плоскости существует *поворотная гомотетия* (композиция поворота вокруг некоторой точки и гомотетии с центром в этой точке), переводящая одну из фигур в другую. Центр поворотной гомотетии и будет искомой неподвижной точкой. Этим можно воспользоваться при решении задачи, причем различными способами.

Решение первое — «итеративное».

Пусть O — центр гомотетии H с коэффициентом k, переводящей больший прямоугольник в меньший. Рассмотрим произвольную точку A большего прямоугольника. $|OH(A)| = k|OA|$, $|OH(H(A))| = k|OH(A)| = k^2|OA|$, … …, $|OH(H(\dots H(A)\dots))| = k^n|OA|$. Заметим, что поскольку $k < 1$, то $k^n \to 0$, т. е. $H(H(\dots H(A)\dots)) \to O$. Следовательно, можно найти центр поворотной гомотетии (неподвижную точку), взяв произвольную точку на карте, найдя ее образ, затем образ ее образа и т. д. Как оценить, сколько шагов потребуется сделать, чтобы достигнуть нужной точности? Для ответа на этот вопрос обратимся к чертежу.



По неравенству треугольника $r < d + kr$, откуда, пользуясь тем, что $1 - k > 0$, получаем $r < d/(1 - k)$. Таким образом, если мы хотим найти центр гомотетии с некоторой точностью ε, нам достаточно дождаться, когда отношение $d/(1 - k)$ станет меньше ε.

Решение второе — «вычислительное».

Проще всего проводить вычисления в комплексных числах (хотя можно составить и решить систему линейных уравнений сразу в координатах).

Запишем в комплексных числах наше преобразование. Поворотная гомотетия с центром в начале координат — это умножение на некоторое комплексное число c такое, что $|c|$ — это коэффициент гомотетии, а $\operatorname{Arg} c$ — угол поворота. Поворотная гомотетия вокруг точки m — это композиция параллельного переноса на вектор $-m$, поворотной гомотетии вокруг начала координат и параллельного переноса на вектор m. Окончательно имеем формулу:

$$H(z) = c(z - m) + m.$$

Пусть при нашем преобразовании некоторая точка a переходит в a', а некоторая другая точка b — в b'. Подставим эти точки в формулу преобразования и получим систему из двух комплексных уравнений с двумя комплексными неизвестными c и m:

$$\begin{cases} c(a - m) + m = a', \\ c(b - m) + m = b'. \end{cases}$$

Вычитая из первого уравнения второе, получаем: $c(a - b) = a' - b'$, откуда $c = (a' - b')/(a - b)$ (что, впрочем, можно было понять и не составляя уравнение: при поворотной гомотетии вектор $a' - b'$ переходит в вектор $a - b$). Подставляя полученное значение c в одно из уравнений системы, находим m:

$$m = -\frac{b'a + a'b}{a - b + b' - a'}.$$

В нашем случае в качестве точек a и b можно взять две вершины меньшего прямоугольника, а в качестве точек a' и b' — соответствующие вершины большего прямоугольника. Формула упрощается за счет выбора $a = 0$, $b = x$.

Автор разбора — В. М. Гуровиц.

Задача V-J. Узор

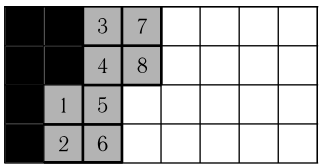
Тема: динамическое программирование.

Давайте внимательно посмотрим на задачу и на ограничения. Кажется, что при таких ограничениях можно воспользоваться перебором, но на практике такая программа будет слишком долго работать, поэтому воспользуемся *динамическим программированием по профилю*. Для данной задачи существует несколько подобных решений, мы же рассмотрим только одно из них.

Пусть клетка $A[i, j]$ расположена в i-й строке и в j-м столбце. Введем лексикографический порядок на множестве всех клеток: клетка $[i_1, j_1]$ меньше клетки $[i_2, j_2]$, если $j_1 < j_2$, или $j_1 = j_2$ и $i_1 < i_2$. В качестве профиля будем рассматривать любые наборы из $2N$ идущих подряд (в смысле введенного порядка) клеток. При решении данной задачи для всех профилей будет выполняться следующее свойство: все клетки, которые идут раньше профиля, уже замощены плитками (клетки, идущие раньше профиля — это клетки, лексикографически меньшие всех клеток профиля).

Профиль задается $2N$ битами (1 — замощенная клетка, 0 — свободная клетка), поэтому для данных $2N$ клеток существует ровно 2^{2N} различных профилей.

На рисунке изображен пример профиля. Серыми клетками обозначен профиль, а черными — клетки, идущие до него.



Заметим, что для данных в условии форм плиток не требуется профиль больше чем из

$2N$ клеток. Для каждого из профилей будет храниться минимальная стоимость плиток, которые были использованы для замощения клеток, идущих до него, и занятых клеток в нем, либо бесконечность, если такой профиль получить невозможно.

При переходе нужно из 2^{2N} профилей (назовем их *старыми*) получить 2^{2N} новых. Для этого можно перебрать все пары профилей и попробовать из каждого старого получить каждый новый всевозможными способами. Но мы сделаем по-другому: переберем все старые профили и попробуем получить из них новые путем добавления одной плитки так, чтобы последняя клетка нового профиля была занята (заметим, что таким образом будут рассмотрены все возможные расположения плиток). При добавлении новой плитки для всех форм, заданных в условии, кроме третьей, первая клетка старого профиля обязательно должна быть занята, для третьей — в зависимости от ориентации плитки. Так же нужно учитывать соответствие цветов, клетки, уже выложенные плиткой, и все возможные повороты плиток. Если на каком-то шаге встречается клетка, выложенная плиткой по условию, то для всех новых профилей она будет занята. Не стоит забывать, что на текущем шаге плитку можно не класть.

Для удобства предположим, что перед первым столбцом есть еще два столбца, которые изначально уже полностью выложены плиткой. Начнем с профилей, состоящих из клеток этих двух столбцов. Можно считать, что цена целиком заполненного профиля равна 0, а всех остальных профилей — бесконечности.

Ответом будет являться стоимость последнего профиля со всеми занятыми ячейками.

В рассмотренном алгоритме NM шагов (это равно количеству клеток в комнате), на каждом шаге выполняется $O(2^{2N} \cdot K)$ операций. Общее время работы алгоритма — $O(N \cdot M \cdot K \cdot 2^{2N})$, что удовлетворяет заданному ограничению по времени.

Автор разбора — В. Ю. Антонов.

Задача V-K. Склад

Тема: перебор.

Эту задачу можно решать перебором.

Можно подсчитать, что изделий, удовлетворяющих условию, на максимальном поле 4×4 всего 180. Все изделия, которые можно использовать при замощении заданного поля, нетрудно получаются перебором всех возможных треугольников на этом поле, которые, в свою очередь, можно найти перебором всех троек точек на поле. Для каждой тройки нужно проверить, что:

1) эти точки являются вершинами треугольника;
2) этот треугольник равнобедренный;
3) у него есть сторона, параллельная оси Ox или оси Oy;
4) треугольник не пересекается с изделиями-треугольниками, которые уже присутствуют на складе (пересечением считается пересечение ненулевой площади);
5) такой треугольник не был найден ранее и не был задан теми же точками в другом порядке.

Самое трудное из вышеперечисленного — проверить, пересекаются ли два треугольника. Какие случаи возможны?

1. Вершина одного из треугольников лежит *внутри* другого треугольника. Это можно проверить либо используя площади, либо используя вычисление знака векторного произведения (в принципе, это одно и то же: можно вычислять площадь как половину модуля векторного произведения).

2. Вершины одного треугольника (назовем его *маленьким*) лежат на сторонах и (или) совпадают с вершинами другого треугольника (назовем его *большим*). Нетрудно доказать, что в этом случае существует хотя бы одна сторона маленького треугольника, у которой середина лежит внутри большого треугольника (или треугольники совпадают — этот случай, как уже говорилось выше, мы отсечем на следующем, 5-м шаге проверки).

3. Остается случай, когда хотя бы одна вершина первого треугольника лежит вне второго, а две другие — либо на сторонах, либо тоже вне него. В этом случае треугольники имеют ненулевую площадь пересечения, только если сторона одного из треугольников пересекает сторону второго.

Дальнейшее решение состоит в переборе всех наборов из найденных треугольников. Конечно, простой перебор будет работать слишком долго, поэтому его необходимо оптимизировать. Приведем лишь некоторые возможные (впрочем, достаточные для решения задачи в указанных ограничениях) приемы оптимизации.

1. Отсортируем найденные треугольники по невозрастанию площадей и будем выбирать их при переборе именно в этом порядке.

2. Каждую пару треугольников заранее проверим на пересечение.

3. Как определить, что склад полностью заставлен изделиями? Будем хранить площадь, которую осталось замостить. При переборе эту величину пересчитывать легко, если заранее вычислить площадь каждого треугольника.

4. Пусть уже найдено какое-то решение (в нем `AnsCnt` изделий), и в следующем решении, которое мы строим, уже использовано `Cnt` изделий, и при этом осталось замостить площадь `S`, а текущий треугольник имеет площадь `SNow`. Все треугольники, которые еще не рассмотрены, имеют

площадь меньшую или равную `SNow`. Понятно, что для замощения оставшейся площади потребуется не меньше, чем `S/SNow` треугольников. Так же понятно, что если `AnsCnt <= Cnt + S/SNow`, то перебор в этом случае осуществлять не имеет смысла, так как найденное решение заведомо не хуже того решения, которое будет найдено.

Автор задачи — Е. А. Шавлюгин, автор разбора — В. Ю. Антонов.

Задача V-L. Кубическая гостиница

Тема: задачи на разные темы.

Воспользуемся *жадным алгоритмом*. Вооружимся для начала видом спереди и посмотрим спереди на только что построенную гостиницу. Если левый верхний блок не того цвета, что соответствующий блок на схеме, то удалим этот блок и посмотрим на блок за ним. Если и этот блок не того цвета, удалим и его и т. д., до тех пор, пока не найдем блок нужного цвета. Если на плане в этом месте дырка, то удалим сразу весь столбик блоков. Теперь перейдем к соседнему блоку и т. д. После того как полученная гостиница будет соответствовать виду спереди на плане, так же поступим с каждым из остальных видов. После этого опять вернемся к виду спереди и т. д., до тех пор, пока хотя бы один из видов будет отличаться от плана.

Автор разбора — В. М. Гуровиц.

Олимпиада VI. Личная олимпиада 2004–2005 учебного года

Задача VI-A. Праздники

Тема: задачи на разные темы.

Решим сначала более простую задачу: будем считать, что праздниками являются *только* первые K дней года. Попробуем вывести явную формулу, выражающую количество выходных дней W_K через K. Для маленьких K можно вычислить значения W_K вручную:

K	1	2	3	4	5
W_K	3	4	5	6	9

Теперь заметим, что если любое из значений K увеличить на 5, то количество выходных дней увеличится на 7. Поэтому $W_{K+5} = W_K + 7$. Теперь нетрудно написать и явную формулу:

$$W_K = 2 + (K \bmod 5) + 7(K \operatorname{div} 5),$$

где mod — взятие остатка, а div — получение целой части от деления (как в языке Паскаль).

Теперь перейдем к решению оригинальной задачи: вспомним про 23 февраля и 8 марта.

Итак, нам требуется найти такой год, на который придется наибольшее количество подряд идущих выходных дней. Год бывает високосным (29 дней в феврале) и не високосным (28 дней в феврале), причем как тот, так и другой может начинаться с любого из семи дней недели. Таким образом, нам придется перебрать 14 вариантов и выбрать из них наилучший.

Покажем, как подсчитать общее количество выходных дней для одного из этих вариантов. Будем идти по календарю и считать праздники, выпавшие на выходные (назовем используемую для этого переменную p). Как только мы встречаем праздник, выпавший на субботу или воскресенье, мы увеличиваем p на 1 (здесь под праздничными подразумеваются первые K дней года, а также 23 февраля и 8 марта; заметим, что при заданных в задаче ограничениях K первых праздничных дней года всегда заканчиваются до 23 февраля). Наоборот, как только мы встречаем рабочий день, мы уменьшаем p на 1. Когда p станет равна нулю и мы встретим рабочий день, процесс нужно остановить — это будет первый рабочий день в рассматриваемом году: все дни до него — выходные. После этого нужно дополнительно проверить, не являлись ли выходными 30 и 31 декабря предыдущего года, и при необходимости увеличить общее количество выходных в рассматриваемом варианте.

Покажем, как работает этот алгоритм, на конкретном примере. Пусть $k = 10$, и первое января — понедельник. Составим такую таблицу:

номер дня	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	11	12	13	14	15
день недели	пн	вт	ср	чт	пт	**сб**	**вс**	пн	вт	ср	чт	пт	**сб**	**вс**	пн
p	0	0	0	0	0	1	2	2	2	2	1	0	0	0	

Мы остановились на первом *рабочем* дне 15 января, встреченном *после* того, как значение p стало равно 0. Таким образом, мы получили 14 выходных. Добавим к ним еще два: 30 и 31 декабря предыдущего года (суббота и воскресенье) и получим ответ для данного случая: 16. Перебрав еще 6 случаев (еще 7, относящихся к високосному году, здесь можно не анализировать, так как наше рассмотрение всегда заканчивается январем) и выбрав наибольший ответ, мы увидим, что приведенное решение и является лучшим для $K = 10$. Заметим, что оно же могло быть получено и для года, который начинается с субботы:

номер дня	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	11	12	13	14	15	16	17
день недели	**сб**	**вс**	пн	вт	ср	чт	пт	**сб**	**вс**	пн	вт	ср	чт	пт	**сб**	**вс**	пн
p	1	2	2	2	2	2	2	3	4	4	3	2	1	0	0	0	

Таким образом, дни предыдущего года здесь учитывать не нужно.

Автор задачи и решения — В. М. Гуровиц.

Задача VI-B. Раскраска плиток

Тема: игры и стратегии.

Покажем сначала, что решение всегда существует. Так как $K < N$, то в каждом горизонтальном ряду плиток найдутся по крайней мере две плитки одного цвета. Выпишем для каждого ряда любой из таких цветов (в него мы заведомо можем перекрасить соответствующий ряд). Среди выписанных N значений цветов по крайней мере два окажутся одинаковыми. Окрасим в этот цвет соответствующие два горизонтальные ряда плиток. Теперь в каждом вертикальном ряду мы имеем две плитки одного и того же цвета. Следовательно, окунув еще N раз кисть в ведро с краской, Том сможет перекрасить все плитки по своим правилам, и искомое значение L не может превосходить $N + 2$.

Рассмотрим теперь, при каких условиях можно покрасить двор быстрее. Так как все плитки должны быть перекрашены, минимальное значение L равно N. В этом случае Том будет красить либо только горизонтальные ряды, либо только вертикальные, и перекрашивание одного ряда никак не повлияет на возможность красить в этот же цвет другие ряды. Следовательно, для того чтобы перекрасить двор за N разрешенных операций,

необходимо и достаточно иметь по две плитки одного и того же цвета в каждом из горизонтальных (или вертикальных) рядов.

За $N + 1$ операцию все плитки можно перекрасить в цвет C тогда и только тогда, когда в каждом горизонтальном (вертикальном) ряду есть не менее одной плитки цвета C, а еще как минимум в двух горизонтальных (вертикальных) рядах — не менее чем две плитки этого же цвета. Действительно, в этом случае сначала перекрасим те два горизонтальные (вертикальные) ряда, в которых имеются две плитки цвета C. Теперь мы можем перекрасить уже любой вертикальный (горизонтальный) ряд. Выберем для этого такой ряд, что после его окраски уже в каждом горизонтальном (вертикальном) ряду окажется по две плитки цвета C. Это всегда можно сделать, так как в остающихся неперекрашенными $N - 2$ рядах по одной плитке цвета C может располагаться максимум на $N - 2$ вертикалях (горизонталях), а любую из оставшихся двух вертикалей (горизонталей) мы можем перекрасить (см. рис.). Теперь у нас уже во всех горизонтальных рядах есть по две плитки цвета C, и мы можем окрасить в него оставшиеся $N - 2$ горизонтали (это быстрее, чем окраска $N - 1$ вертикали!). Общее количество операций L при этом равно: $2 + 1 + N - 2 = N + 1$.

5	4	1	3	2	1
4	5	1	3	2	3
1	1	1	2	1	3
4	4	3	1	2	2
2	2	2	4	3	2
4	4	1	2	3	3

5	4	1	3	2	1
3	3	3	3	3	3
1	1	1	2	1	3
4	4	3	1	2	2
2	2	2	4	3	2
3	3	3	3	3	3

5	3	1	3	2	1
3	3	3	3	3	3
1	3	1	2	1	3
4	3	3	1	2	2
2	3	2	4	3	2
3	3	3	3	3	3

Докажем, что это же условие является необходимым. Если существует горизонтальный ряд, в котором плиток цвета C нет, то нам заведомо потребуются две вспомогательные вертикали для его дальнейшего окрашивания. Кроме этого, мы будем красить все N горизонталей и общее количество операций окажется равным $N + 2$. Дополнительную вертикаль мы можем либо перекрасить сразу, если имеющиеся «вторые» плитки в двух горизонталях располагаются в одной вертикали, либо согласно процедуре, описанной выше. В любом случае требуется, чтобы как минимум в двух горизонталях изначально находилось по две плитки цвета C. Аналогичные рассуждения можно провести для вертикальных рядов.

Алгоритм решения задачи сформулирован. Осталось его эффективно реализовать. Опишем одну из возможных реализаций для горизонтальной закраски. В данном случае очень удобно использовать такой специфический тип данных языка программирования Паскаль как множество. Заведем массивы множеств `sh[1..N]` и `sh1[1..N]`. Каждый элемент первого

из них будет хранить множество цветов, которые встречаются в соответствующей горизонтали не менее двух раз. Аналогично, каждый элемент второго — множество цветов, которые встречаются в соответствующей горизонтали не менее одного раза. Заполнение этих массивов по двухмерному массиву цветов плиток двора `a[1..N,1..N]` производится так:

```
sh := []; sh1 := [];
for i := 1 to n do
for j := 1 to n do
 if a[i,j] in sh1[i]
  then  sh[i] :=  sh[i] + [a[i,j]]
  else sh1[i] := sh1[i] + [a[i,j]];
```

Теперь очень легко проверить возможность горизонтальной закраски за N операций: пересечение всех множеств `sh[i]` должно быть отлично от пустого. На Паскале эта проверка выглядит так:

```
sr := sh[1];{sr - вспомогательная переменная типа множество}
for i := 2 to n do sr := sr*sh[i];
if sr <> [] then {искомый цвет существует}
  for color := 1 to k do {ищем этот цвет}
    if color in sr then out(n,color);
```

Если такой цвет не найден, в том числе и по вертикальному направлению, то попытаемся перекрасить двор за $N+1$ операций:

```
sr := sh1[1];
for i := 2 to n do sr := sr*sh1[i];
if sr <> [] then
{в каждой горизонтали есть общий цвет}
  for color := 1 to k do
    if color in sr then
     begin
       cnt := 0;
       for i := 1 to n do
        {ищем ряды с 2 плитками}
         if color in sh[i] then inc(cnt);
       if cnt >= 2 then out(n+1,color)
     end;
```

Аналогичное рассмотрение нужно провести и по другому направлению. Если решение до сих пор не получено, то остается определить цвет, в который можно перекрасить двор за $N+2$ операций (здесь достаточно рассмотреть одно направление перекраски):

```
sr := [];
for i := 1 to n do
```



```
  if sh[i] * sr = [] then
    sr := sr + sh[i]
  else for color:=1 to k do
         if color in sh[i]*sr then
           out(n+2,color);
```

Таким образом, вычислительная сложность алгоритма решения задачи равна $O(N^2)$. Решение приемлемой сложности можно получить, заменив массивы множеств на двухмерные массивы размером `[1..N,1..K]`, имеющие тот же смысл, что и описанные выше множества.

Автор задачи и решения — Е. В. Андреева.

Задача VI-C. Маскарад

Тема: динамическое программирование.

Найдем способ определять минимальное количество денег, необходимое для того чтобы купить S метров ткани, используя магазины с 1-го по K-й (обозначим эту величину $A[S][K]$). Покажем, как вычислить $A[S][K]$, если мы уже знаем значения $A[s][k]$ для всех $s \leqslant S$, $k < K$.

Пусть, согласно оптимальному плану по закупке S метров ткани в магазинах с 1-го по K-й, в магазине номер K было куплено t метров ткани (t может принимать значения от 0 до $\min(S, F[K])$). Тогда $A[S][K] = \mathrm{Cost}(K, t) + A[S-t][K-1]$. Здесь $\mathrm{Cost}(K, t)$ — стоимость закупки t метров ткани в магазине K. Значит,

$$A[S][K] = \min_{t\in[0,\min(S,F[K])]} (\mathrm{Cost}(K, t) + A[S-t][K-1]). \qquad (*)$$

Тогда ровно L метров ткани можно купить за $A[L][N]$ бурлей. Заметим, что иногда бывает выгодно покупать более L метров ткани, так как когда начинают работать скидки, суммарная стоимость может получиться меньше, поэтому недостаточно вычислить значение $A[L][N]$. Ответом на поставленную задачу будет

$$\min_{S\in[L,\infty)} (A[S][N]). \qquad (**)$$

Поэтому в общем случае возникает задача нахождения минимума в бесконечной последовательности значений $A[S][N]$ для $S \geqslant L$. Однако в нашей задаче на S в $(**)$ можно наложить ограничение сверху. Дело в том, что стоимость ткани всегда положительна, поэтому если мы согласно оптимальному плану закупок уже купили в первых K магазинах не менее L метров ткани, то во всех остальных магазинах (с $(K+1)$-го по N-й) ткань закупаться уже не будет. Значит, $S \leqslant S_{\max} = L + \max\limits_{i\in[1,N]} (F[i]) - 1$.

Будем хранить значения $A[S][K]$ в двухмерном массиве $A[0..S_{\max}, 1..N]$. Для заполнения этого массива по формулам $(*)$ необходимы начальные

значения. Ими в нашем решении будет столбец $A[S][1]$ (стоимость покупки S метров ткани в первом магазине), значения элементов которого предлагается задать явным образом. В случае, если в первом магазине нет S метров ткани, элементу $A[S][1]$ нужно присвоить значение равное «бесконечности» (в программе бесконечность представляет собой какое-нибудь достаточно большое число, заведомо превосходящее ответ, для данной задачи подойдет число 10^6). Вычислим остальные элементы массива по формулам $(*)$ в порядке неуменьшения K, а при равных K — в порядке увеличения S.

Для вычисления плана закупок ткани вместе с заполнением массива A будем заполнять дополнительный массив $D[S][K]$, элементами которого будут значения t из формул $(*)$, на которых достигаются соответствующие минимальные значения. С помощью массива D несложно восстановить весь план закупок.

Общее время работы данного алгоритма $O(F_{\max}S_{\max}N)$.

Автор задачи — К. А. Батузов,
авторы разбора — К. А. Батузов, М. О. Трухина.

Задача VI-D. Билетики

Тема: задачи на разные темы.

Пусть дана последовательность чисел A_1, …, A_N, состоящая из 0 и 1. Пару чисел (A_i, A_j) таких, что $i < j$, $j - i = K$ и $A_i \neq A_j$, назовем *неправильной*. *Индексом неправильности последовательности* (ИНП) назовем количество неправильных пар (A_i, A_j) в ней. ИНП легко посчитать за один проход по массиву, в котором хранятся элементы последовательности: переберем все i от 1 до $N - K$ и сравним их с соответствующими $j = i + K$.

Если ИНП равен 0, то ответ в задаче `OK` и никаких чисел менять не нужно. Иначе для каждого из элементов последовательности найдем ИНП после изменения этого элемента. При изменении одного числа в последовательности нулей и единиц может измениться только правильность пар (A_{i-K}, A_j) и (A_i, A_{i+K}), если они, конечно, существуют. Дело в том, что $i - K$ может быть меньше 1 или $i + K$ — больше N. В этих случаях соответствующая пара отсутствует. Значит, в результате изменения одного числа ИНП может измениться на -2, -1, 0, $+1$ или $+2$. Если в результате изменения какого-то из чисел ИНП станет равным 0, то ответ: `OK` и номер измененного элемента, в противном случае ответ `FAIL`. Данный алгоритм работает за линейное относительно длины последовательности время.

Автор задачи — К. А. Батузов,
авторы разбора — К. А. Батузов, М. О. Трухина.

Задача VI-E. Сплоченная команда

Тема: сортировка.

Отсортируем игроков в порядке возрастания их ПП с помощью любого эффективного алгоритма сортировки с вычислительной сложностью $O(N \log N)$. При этом исходные номера игроков необходимо запомнить в отдельном массиве для последующего вывода ответа.

Критерием сплоченности команды является то, что ПП самого сильного игрока не превосходит суммы ПП двух самых слабых. Команду с максимальной суммой ПП всегда можно выбрать так, что игроки, в нее входящие, образуют непрерывную последовательность в отсортированном массиве.

Действительно, пусть существует игрок, не вошедший в команду, номер которого в отсортированном массиве больше номера самого слабого игрока команды, но меньше номера самого сильного. Тогда можно взять его в команду вместо самого слабого игрока. При этом команда, очевидно, останется сплоченной, а сумма ПП команды заведомо не уменьшится. Если по-прежнему существует игрок, не вошедший в команду, номер которого в отсортированном массиве больше номера самого слабого игрока команды, но меньше номера самого сильного, то можно повторно применить то же рассуждение. Эта процедура продолжается до тех пор, пока команда не составит непрерывную последовательность игроков в отсортированном массиве. Такой момент наступит, поскольку количество игроков между самым слабым и самым сильным игроками команды в отсортированном массиве на каждом шаге уменьшается, а количество игроков команды остается неизменным.

Таким образом, в отсортированном массиве необходимо найти непрерывную последовательность игроков, образующих сплоченную команду с максимальной суммой ПП. Кроме того, необходимо аккуратно разбирать случай $N = 0$, а также учитывать то, что любая команда из одного или двух игроков является сплоченной. При других значениях N искомую последовательность игроков можно найти одним из двух способов.

1. Храним два указателя: на самого слабого и самого сильного игрока команды — i и j. В переменной `sum` храним сумму ПП игроков с i-го по j-й. Изначально $i = 1$, $j = 2$.

Пока j не превосходит N, делаем следующую операцию. Если ПП j-го игрока не превосходит суммы ПП i-го и $(i + 1)$-го игроков (т. е. команда, состоящая из игроков с i-го по j-й, является сплоченной), то увеличиваем j на 1. Если при этом `sum > res`, где `res` — сумма ПП лучшей из уже найденных команд, то обновляем результат. Если же ПП j-го игрока указанную сумму превосходит, то увеличиваем i на 1. В зависимости от сдвига указателей нужно пересчитывать переменную `sum`.

Время работы алгоритма без учета сортировки $O(N)$, поскольку каждый указатель пробегает по массиву лишь однажды.

2. Для каждого i от 1 до $N-1$ двоичным поиском находим максимальное $j > i$ такое, что игроки с i-го по j-й образуют сплоченную команду. При этом, чтобы быстро находить сумму ПП игроков с i-го по j-й, необходимо предварительно за линейное время насчитать вспомогательный массив `s`, в котором `s[i]` — сумма ПП игроков с 1-го по i-й. Тогда сумма ПП игроков с i-го по j-й: `sum=s[j]-s[i-1]`. Выбирая максимум значений `sum` для всех i от 1 до $N-1$, получим искомый ответ. Время работы этого алгоритма $O(N \log N)$, что несколько хуже, чем в первом варианте.

Автор задачи и разбора — А. П. Лахно.

Задача VI-F. Сократи векторы

Тема: геометрия.

Будем исходить из физического смысла задачи. Если интерпретировать заданные векторы как силы, приложенные к твердому телу, то ответом должен быть минимальный набор сил, действующий на тело так же, как исходная система. При этом не должна измениться векторная сумма всех сил (это равнодействующая сил) и суммарный момент сил относительно произвольно выбранной точки (например, начала координат).

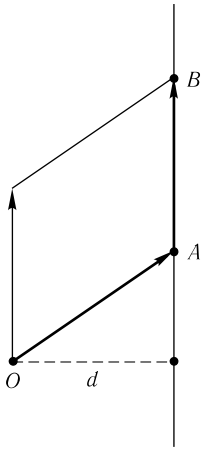


Рис. 1

Пусть на плоскости заданы точка и вектор $\vec{\mathbf{a}} = \overrightarrow{AB}$, d — расстояние от O до прямой AB. *Моментом вектора* $\vec{\mathbf{a}}$ относительно точки O называется величина $d \cdot |\vec{\mathbf{a}}|$, взятая со знаком «плюс», если угол между лучами OA и AB положительный (т. е. вектор $\overrightarrow{OA}$ поворотом против часовой стрелки на угол, меньший 180°, можно преобразовать в вектор, сонаправленный с $\overrightarrow{AB}$), «минус» в противном случае. Эта величина равна ориентированной площади $[\overrightarrow{OA}, \overrightarrow{AB}]$ параллелограмма, построенного на векторах $\overrightarrow{OA}$ и $\overrightarrow{AB}$ (рис. 1) и в геометрии называется косым произведением векторов. Если $\overrightarrow{OA} = (x_1, y_1)$, $\overrightarrow{AB} = (x_2, y_2)$, то $[\overrightarrow{OA}, \overrightarrow{AB}] = x_1 \cdot y_2 - y_1 \cdot x_2$. *Моментом системы* относительно точки O называется сумма моментов всех векторов относительно O. Далее словом момент мы обозначаем момент относительно точки O. Операции над векторами, описанные в условии задачи, назовем *элементарными*. Всякие два вектора, лежащие на различных параллельных прямых, направленные противоположно и равные по длине, будем называть *векторной парой* (или *парой*, если из контекста ясно, что речь идет о векторной паре).

Получаем следующий способ решения задачи. Фиксируем произвольную точку O и для заданных векторов считаем их сумму и суммарный момент M относительно точки O. Если сумма векторов отлична от нуля, ответом будет вектор $\vec{\mathbf{v}}$, равный этой сумме. Линия действия этого вектора подбирается таким образом, чтобы момент вектора $\vec{\mathbf{v}}$ относительно точки O был равен M. Один из способов найти начало A вектора $\vec{\mathbf{v}}$ исходя из двух условий:

$$(\overrightarrow{OA},\ \vec{\mathbf{v}}) = x \cdot v_x + y \cdot v_y = 0 \quad \text{и} \quad [\overrightarrow{OA},\ \vec{\mathbf{v}}] = x \cdot v_y - y \cdot v_x = M.$$

Здесь $\overrightarrow{OA} = (x,\ y)$ — искомый вектор, $\vec{\mathbf{v}} = (v_x,\ v_y)$ — вектор суммы исходной системы.

Если сумма векторов равна нулю, а $M \neq 0$, одним вектором в ответе, очевидно, обойтись нельзя. В этом случае достаточно отыскать пару противоположно направленных векторов одинаковой (например, единичной) длины, каждый из которых имеет момент $M/2$ относительно точки O. Наконец, если и сумма векторов, и их суммарный момент нулевые, ответом будет нуль-вектор, приложенный к любой точке плоскости. Дадим обоснование этого способа решения.

***Утверждение* 1.** *Всякий вектор $\overrightarrow{AB}$ можно заменить равным ему вектором $\overrightarrow{CD}$, если точки A, B и C лежат на одной прямой.*

Для этого нужно на прямой AB породить два вектора $\overrightarrow{CD}$ и $\overrightarrow{CD'} = -\overrightarrow{CD}$, сложить $\overrightarrow{CD'}$ и $\overrightarrow{AB}$ и суммарный нуль-вектор прибавить к $\overrightarrow{CD}$, получив вектор с началом C. Таким образом, любой вектор можно переносить вдоль линии его действия. Как следствие, нуль-вектор можно переносить в любую точку плоскости.

***Утверждение* 2.** *Элементарные операции сохраняют сумму векторов и момент системы относительно любой точки O.*

Это утверждение очевидно для суммы векторов. Для моментов это свойство линейности: $[\overrightarrow{OA}, \overrightarrow{AB}] + [\overrightarrow{OA}, \overrightarrow{AC}] = [\overrightarrow{OA}, \overrightarrow{AB} + \overrightarrow{AC}]$, которое вытекает из координатной записи косого произведения векторов.

***Утверждение* 3.** *Любая конечная система векторов эквивалентна одному вектору или векторной паре.*

Будем элементарными операциями сокращать количество векторов. Если имеются два непараллельных вектора или два вектора на одной прямой, их можно заменить на один. Если есть два коллинеарных вектора $\overrightarrow{AB}$ и $\overrightarrow{CD}$ разной длины, породим, например на прямой AC, две пары равных по длине векторов (рис. 2). Векторы $\overrightarrow{AA''}$ и $\overrightarrow{CC'}$ в сумме дают нуль-вектор $\overrightarrow{AA}$. Суммируя векторы $\overrightarrow{AA'}$ и $\overrightarrow{AB}$, $\overrightarrow{CC''}$ и $\overrightarrow{CD}$, получим два неколл-

линеарных вектора $\overrightarrow{AE}$ и $\overrightarrow{CF}$. Их неколлинеарность следует из сохранения суммы векторов (если бы $\overrightarrow{AE}$ и $\overrightarrow{CF}$ оказались коллинеарными, то неколлинеарными были бы $\overrightarrow{AE}+\overrightarrow{CF}$ и $\overrightarrow{AB}+\overrightarrow{CD}$, что противоречит утверждению 2. Затем складываем векторы $\overrightarrow{AA}$, $\overrightarrow{AE}$ и $\overrightarrow{CF}$, получая один результирующий вектор. Так же можно поступить и с сонаправленными векторами одной длины. Таким образом, количество векторов можно уменьшать до тех пор, пока не останется один вектор (возможно, это нуль-вектор), либо два противоположно направленных вектора одной длины с разными линиями действия, т. е. векторная пара.

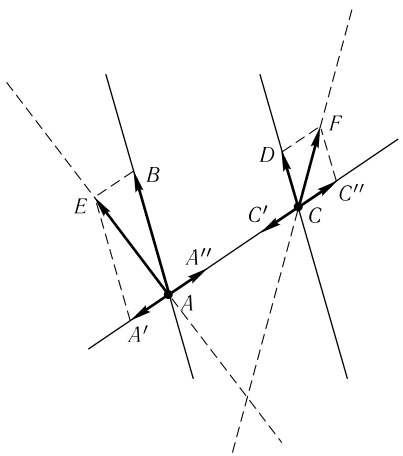


Рис. 2

Из утверждений 2 и 3 вытекает, что тип ответа (нуль-вектор, вектор или векторная пара) однозначно определяется суммарными вектором и моментом исходной системы. Остается показать, что *любой* ответ, для которого эти величины «правильные», может быть получен из исходной системы посредством элементарных преобразований. Рассмотрим разные типы ответа. Нуль-вектор, он и в Африке нуль-вектор (утверждение 1). Второй случай: ответом является один ненулевой вектор $\vec{\mathbf{v}}=\overrightarrow{AB}$. Если другой вектор $\vec{\mathbf{v}}_1=\overrightarrow{CD}$ с ним сонаправлен, имеет те же длину и момент, то линии действия векторов $\vec{\mathbf{v}}$ и $\vec{\mathbf{v}}_1$ совпадают. Значит, по утверждению 1 от $\vec{\mathbf{v}}$ можно перейти к $\vec{\mathbf{v}}_1$. Остается рассмотреть случай, когда ответом является векторная пара.

***Утверждение* 4.** *Две векторные пары, имеющие одинаковые моменты, эквивалентны.*

Пусть у нас есть две векторные пары $\vec{\mathbf{u}}$, $\vec{\mathbf{u}}'=-\vec{\mathbf{u}}$ и $\vec{\mathbf{v}}$, $\vec{\mathbf{v}}'=-\vec{\mathbf{v}}$ с одинаковым моментом M, причем эти пары не параллельны. Покажем, что от пары $\vec{\mathbf{u}}$, $\vec{\mathbf{u}}'$ можно перейти к паре $\vec{\mathbf{v}}$, $\vec{\mathbf{v}}'$. Будем считать (можем так считать по утверждению 1), что вектор $\vec{\mathbf{u}}$ имеет началом точку A пересечения прямых, на которых лежат $\vec{\mathbf{u}}$ и $\vec{\mathbf{v}}$, вектор $\vec{\mathbf{u}}'$ — точку A' пересечения прямых, на которых лежат $\vec{\mathbf{u}}'$ и $\vec{\mathbf{v}}'$ (рис. 3). Породим на прямой AA' такие два вектора $\overrightarrow{AB}$ и $\overrightarrow{A'B'}=-\overrightarrow{AB}$, чтобы вектор $\vec{\mathbf{v}}_1=\overrightarrow{AB}+\vec{\mathbf{u}}$ был параллелен вектору $\vec{\mathbf{v}}$. Это всегда возможно, поскольку $\vec{\mathbf{u}}$, $\vec{\mathbf{v}}$ и $\overrightarrow{AA'}$ попарно не параллельны. Векторы $\vec{\mathbf{v}}_1$ и $\vec{\mathbf{v}}_1'=\overrightarrow{A'B'}+\vec{\mathbf{u}}'$ образуют пару (они имеют сумму 0 и суммарный момент M), при этом векторы $\vec{\mathbf{v}}_1$ и $\vec{\mathbf{v}}$ лежат на одной прямой (как и векторы $\vec{\mathbf{v}}'$ и $\vec{\mathbf{v}}_1'$). Значит, $\vec{\mathbf{v}}_1=k\cdot\vec{\mathbf{v}}$, $\vec{\mathbf{v}}_1'=k\cdot\vec{\mathbf{v}}'$. Момент

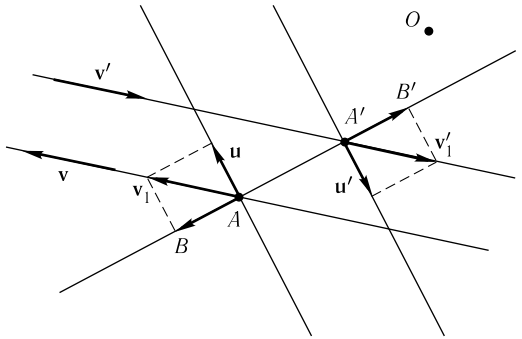


Рис. 3

этой пары равен $M=[\overrightarrow{OA},\ \vec{\mathbf{v}}_1]+[\overrightarrow{OA'},\ \vec{\mathbf{v}}_1']=k\cdot[\overrightarrow{OA},\ \vec{\mathbf{v}}]+k\cdot[\overrightarrow{OA'},\ \vec{\mathbf{v}}']=$ $=k\cdot M$. Отсюда $k=1$, $\vec{\mathbf{v}}_1=\vec{\mathbf{v}}$, $\vec{\mathbf{v}}_1'=\vec{\mathbf{v}}'$. Мы установили, что пары $\vec{\mathbf{u}}$, $\vec{\mathbf{u}}'$ и $\vec{\mathbf{v}}$, $\vec{\mathbf{v}}'$ эквивалентны, что и требовалось.

Если же все векторы $\vec{\mathbf{u}}$, $\vec{\mathbf{u}}'$, $\vec{\mathbf{v}}$ и $\vec{\mathbf{v}}'$ параллельны, то сначала переведем указанным способом пару $\vec{\mathbf{u}}$, $\vec{\mathbf{u}}'$ в любую пару, ей не параллельную, а затем из этой вспомогательной пары получим пару $\vec{\mathbf{v}}$, $\vec{\mathbf{v}}'$.

Итак, в приведенной задаче сразу можно было выписать ответ, исходя из простых физических соображений. Конечно, задачу можно решать и «в лоб», применяя цепочку элементарных преобразований. Порядок сокращения при этом не важен. Этот способ, по сути, описан при доказательстве утверждения 3. Участники олимпиады, получившие за решение задачи максимальное количество баллов, поступали именно таким образом. Но и при таком решении полезно понимать физическую подоплеку задачи, это поможет найти правильный путь к цели.

Авторы задачи — Е. В. Андреева, Ю. Е. Егоров,
автор разбора — Ю. Е. Егоров.

Задача VI-G. Strategy tetris

Тема: структуры данных.

Решение жюри, как и большинство предложенных участниками решений, использует структуру данных *куча* (подробнее про нее можно прочитать во многих книгах, например в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ» (М.: МЦНМО, 2000)). Позже было придумано решение, которое обходится без нее.

Рассмотрим решение, которое предполагалось изначально. Сопоставим каждой фигуре промежуток $[L_i,\ R_i]$, где L_i и R_i — абсциссы самой левой и самой правой клеток фигуры соответственно. Будем рассматривать фигуры в порядке возрастания L_i, определяя, в какую горизонталь их поста-

вить. Назовем горизонталь *допустимой* для фигуры F, если в нее может упасть F, с учетом уже упавших фигур. Горизонталь является *использованной*, если в ней есть хотя бы одна упавшая фигура, и *неиспользованной* в противном случае. Сопоставим каждой фигуре номер горизонтали, в которую она должна попасть. Заметим, что этот номер и номер строки в окончательном расположении могут отличаться (фигура может упасть ниже, что следует из дальнейшего описания). Воспользуемся *жадным алгоритмом*. Решение будет оптимальным, если каждую фигуру ставить *в любую допустимую использованную горизонталь*. Если такой нет, то поместим фигуру в неиспользованную горизонталь, которая имеет наименьший номер.

Почему же полученное решение будет оптимальным? Пусть это не так. Тогда найдем фигуру F с наименьшим параметром L_i, которая в нашем решении находится не там где в оптимальном. Пусть она стоит в строке x в нашем решении, а в оптимальном — в строке y. Тогда переложим F и то, что лежит после нее в строке x, в строку y, а то, что этому мешает — из строки y в строку x. Те фигуры, которые нам мешали, начинаются не левее F, так как по предположению F — первая фигура, расположение которой в двух решениях не совпадает. Заметим, что строка x в результате этого преобразования не может получиться пустой, поскольку это противоречит алгоритму (существовала допустимая использованная горизонталь, а фигуру F поставили в неиспользованную горизонталь). Если провести такое рассуждение последовательно для всех фигур, расположение которых в двух решениях не совпадает, то получится, что наше решение не отличается по высоте от оптимального, т. е. само является оптимальным.

Решение представляет собой сортировку интервалов $[L_i, R_i]$ и проход по отсортированному массиву. Но при этом проходе для каждой фигуры нужно искать ту строку, в которую ее можно поставить, что требует порядка N операций. Общее количество операций имеет порядок N^2, что не соответствует ограничению по времени. Поэтому для хранения последнего занятого элемента в горизонтали будем использовать структуру данных Heap (*куча*). Благодаря этой структуре данных можно определять, куда добавить рассматриваемую фигуру, за $O(1)$, а добавлять — за $O(\log N)$. Эта оптимизация позволяет написать алгоритм, работающий за $O(N \log N)$ операций.

Далее мы опишем другое решение, не использующее кучу. Рассмотрим начальные и конечные точки всех промежутков и отсортируем их все вместе по неубыванию координаты. При этом для каждой точки будем хранить кроме ее координаты информацию о том, какой фигуре она принадлежит и является ли она началом или концом промежутка. При одинаковой координате сначала будут идти начала промежутков, а потом — концы. Для каждой абсциссы можно определить, в каких горизонталях клетка с этой

абсциссой покрыта некоторой фигурой. Поэтому, если рассматриваемая точка — начало отрезка, можно определить, в какую горизонталь следует поставить фигуру. Будем хранить в списке все использованные горизонтали, в которых абсцисса рассматриваемой точки свободна. Сначала этот список будет пуст. Если встречается конец фигуры, то добавляем горизонталь, в которой она находится, в список (горизонталь «освободилась»). Если встречаем начало, то возможны две ситуации.

1. Список пуст (свободных горизонталей нет). Тогда добавляем новую горизонталь.

2. Список не пустой (есть свободные горизонтали). Тогда можно взять любую горизонталь из списка.

Благодаря тому, что мы рассматриваем начала промежутков *до* их концов, ситуации, когда для данной абсциссы горизонталь освободилась и сразу была занята началом с такой же абсциссой, возникнуть не может.

Остается вывести ответ. Для этого можно дополнительно хранить список фигур в каждой горизонтали.

Автор задачи — М. А. Бабенко, автор разбора — В. Ю. Антонов.

Олимпиада VII. Командная олимпиада 2004–2005 учебного года

Задача VII-A. Москва-сортировочная

Тема: задачи для начинающих.

Это одна из самых простых задач олимпиады. Заметим, что если после вагона с номером k прицеплен не вагон с номером $k+1$, то эти вагоны придется разъединить, чтобы восстановить правильный порядок. Поэтому для решения задачи достаточно подсчитать количество таких пар.

Автор задачи — В. М. Гуровиц, автор разбора — В. Ю. Антонов.

Задача VII-B. Кафе

Тема: динамическое программирование.

Воспользуемся методом динамического программирования.

Заведем двухмерный массив A, в ячейке $[i, j]$ которого будет храниться минимальная сумма, которую мог потратить Петя за первые i дней при условии, что после этого у него в запасе осталось j талонов на бесплатную еду. Положим $A[0, 0] = 0$; $A[0, i] = \infty$ при $i > 0$. Пусть $P[i]$ — цена обеда в i-й день. Для заполнения ячейки $A[i, j]$ можно воспользоваться следующей формулой:

Петя покупает i-й обед.

Петя использует талон на бесплатную еду.

$$A[i, j] = \begin{cases} \min(A[i-1, j] + p[i], & A[i-1, j+1]), \quad \text{при } P[i] \leqslant 100; \\ \min(A[i-1, j-1] + p[i], & A[i-1, j+1]), \quad \text{при } P[i] > 100. \end{cases}$$

Петя покупает i-й обед и при этом получает талон на бесплатную еду.

Ответом будет являться минимальное из чисел (назовем его Ans), хранящихся в N-м столбце. Рассмотрим все элементы N-го столбца, значения которых совпадают с Ans. Понятно, что K_1 — максимальный номер строки из всех номеров строк этих элементов. Как же посчитать параметр K_2, который также нужно вывести? Для этого дополнительно будем хранить в $B[i, j]$ общее количество купонов на бесплатные обеды, полученных Петей (считается по формулам, сходным с формулами для вычисления

$A[i, j]$). Можно не заводить второй массив, а считать K_2 непосредственно перед выводом ответа, используя метод, который будет описан ниже.

Остается вывести дни, в которые были потрачены купоны. Для ячейки массива $A[i, j]$ по числам в столбце $A[i-1]$ можно определить, был ли использован купон в i-й день. А зная K_1, можно вывести все дни, в которые были потрачены купоны, и подсчитать их количество K_2.

Автор задачи — А. П. Лахно, автор разбора — В. Ю. Антонов.

Задача VII-C. EuroEnglish

Тема: строки.

Эта задача не требует ничего, кроме обработки данной строки по заданным правилам. Тем не менее, она оказалась для участников олимпиады непростой. Мы не будем вдаваться в детали реализации, а лишь отметим несколько тонкостей, возникающих при написании программы.

1. Удобно проводить преобразования не в том порядке, в котором это предлагается делать в условии. Например ясно, что удобнее удалить артикли сразу, пока они еще не «испорчены» другими преобразованиями и не стали похожи на другие слова.

2. Не стоит забывать про то, что некоторые слова начинаются с заглавных букв! При этом нужно следить, чтобы «Oo» превращалось в «O», а не в «o» и т. п.

3. Чтобы отдельно не обрабатывать ситуацию с началом и с концом строки, удобно добавить в начало и в конец по несколько лишних символов, скажем, пробелов.

Автор задачи и разбора — В. М. Гуровиц.

Задача VII-D. D++

Тема: задачи на разные темы.

Заметим, что данная последовательность является *перестановкой*. Чтобы упорядочить элементы, воспользуемся перестановкой, *обратной* к данной. Сдвиги элементов будем делать по циклам, на которые распадается обратная перестановка. Для каждого элемента известно, в какой ячейке он должен находиться в упорядоченном массиве. Решение поставленной задачи представляет собой нахождение циклов в перестановке и циклический сдвиг элементов. Сдвиг внутри цикла осуществляется при помощи одной дополнительной переменной, т. е. если цикл состоит из чисел $P[i_1], P[i_2], \ldots$

..., $P[i_k]$, то требуемые операции будут выглядеть так:

$$x \leftarrow P[i_1]$$
$$P[i_1] \leftarrow P[i_2]$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$P[i_{k-1}] \leftarrow P[i_k]$$
$$P[i_k] \leftarrow x.$$

Не стоит забывать, что если элемент изначально стоит на своем месте (т. е. в перестановке есть цикл длины 1), то с этим элементом ничего делать не нужно.

Те, кто не знаком с алгеброй перестановок, могут представить себе граф, вершинами которого являются данные числа, а каждое ребро ведет от числа к тому числу, на место которого оно должно встать. Такой граф представляет собой объединение циклов, о которых и идет речь в разборе.

Автор задачи — К. А. Батузов, автор разбора — В. Ю. Антонов.

Задача VII-E. Скобки

Тема: динамическое программирование.

Пусть задана строка S, состоящая из символов «{», «}», «[», «]», «(», «)». Заведем двухмерный массив A, в элементе $[i, j]$ которого будем хранить минимальное количество скобок, которые нужно добавить к подстроке строки S с i-го по j-й символ ($S[i] \ldots S[j]$), чтобы получить правильную скобочную последовательность. Если $i > j$, то положим $A[i, j] = 0$. Будем находить решения, постепенно увеличивая разность между j и i. Для инициализации стоит заметить, что $A[i, i] = 1$ для любого i (подумайте почему). При увеличении разности между j и i вычисление $A[i, j]$ осуществляется по следующим правилам:

1) если $S[i]$ — закрывающая скобка, то в правильной скобочной последовательности до нее должна быть открывающая скобка, т. е. $A[i, j] = 1 + A[i + 1, j]$;
2) если $S[i]$ — открывающая скобка, то возможно несколько вариантов восстановления правильной скобочной последовательности:
   а) среди остальных скобок существует пара для этой, тогда: $A[i, j] = \min(A[i + 1, k - 1] + A[k + 1, j])$, где минимум берется по всем k таким, что $S[k]$ — закрывающая скобка, соответствующая открывающей скобке $S[i]$;
   б) соответствующую закрывающую скобку нужно добавить к строке, т. е. $A[i, j] = \min(A[i + 1, k] + A[k + 1, j])$ при $i \leqslant k \leqslant j$ (нужную закрывающую скобку вставляем после k-й скобки).

$A[i, j]$ равен минимуму из этих двух вариантов.

Ответом будет являться $A[1, \text{Length}(S)]$.

Авторы задачи — Р. А. Жуйков, М. О. Трухина, автор разбора — В. Ю. Антонов.

Задача VII-F. Двоякие числа

Тема: задачи для начинающих.

Эта задача быда одной из самых простых на олимпиаде. Ограничения на входные данные позволяют перебрать все числа, выделить среди них все двоякие и выбрать из этих чисел ближайшее к N. Достаточно перебрать числа от 1 до 30000, поскольку заданное число лежит в этом диапазоне, а 1 и 30000 сами являются двоякими числами.

Автор задачи — В. М. Гуровиц, автор разбора — В. Ю. Антонов.

Задача VII-G. OOO

Тема: геометрия.

Это самая сложная задача олимпиады, ее не решила ни одна команда. Задача сводится к аккуратному рассмотрению возможных взаимных расположений прожекторов. Для упрощения решения разделим все возможные варианты на группы:

1) прожекторы совпадают;
2) один из прожекторов находится на крайнем луче другого;
3) один из прожекторов находится внутри области, освещаемой другим прожектором;
4) ни один из прожекторов не находится в свете другого.

Детали реализации оставим читателям.

Авторы задачи — А. П. Лахно и А. О. Тимофеев, автор разбора — В. Ю. Антонов.

Задача VII-H. Калах

Тема: игры и стратегии.

Пусть нам известно, в какую коробочку попал последний шарик после некоторого хода. Будем доставать по одному шарику, начиная с этой коробочки и двигаясь против часовой стрелки до тех пор, пока очередная коробочка не окажется пустой. Несложно понять, что именно из этой

коробочки и был сделан ход и до этого хода в данной коробочке лежали как раз все те шарики, которые мы только что достали.

При реализации алгоритма возникает следующая трудность. Количество шариков, лежащих в коробочках, велико, поэтому если мы будем пытаться доставать по одному шарику, то просто не уложимся в отведенное нам время. Для ускорения алгоритма можно поступить следующим образом. Найдем коробочку с наименьшим числом шариков S. Ясно, что вынимая шарики по одному, мы совершим не менее S полных кругов, поскольку в каждой коробочке лежит хотя бы S шариков. Поэтому вместо того чтобы вынимать по одному шарику, мы можем вынуть сразу по S шариков из каждой коробочки, а затем закончить процесс, вынув еще по одному шарику из коробочек, предшествующих ближайшей из образовавшихся пустых коробочек.

Автор задачи и разбора — В. М. Гуровиц.

Задача VII-I. Сортировка масс

Тема: сортировка.

Переведем все массы в миллиграммы. Не стоит забывать, что не всякие типы позволяют хранить полученные данные. Так, в языке Паскаль для этого можно использовать тип Int64. Осталось отсортировать полученные массы и вывести их в соответствии с нужным форматом, что можно легко сделать, если хранить для каждой массы информацию о том, как она была задана во входном файле.

Авторы задачи — К. А. Батузов, Р. А. Жуйков, М. О. Трухина, автор разбора — В. Ю. Антонов.

Олимпиада VIII. Заочный тур 2005–2006 учебного года

Задача VIII-A. Часы с боем

Тема: задачи для начинающих.

Решение первое: моделирование. Будем «крутить» стрелки часов, каждую минуту оценивая ситуацию и считая удары. Этот алгоритм можно реализовать, например, так:

```
while (h1 <> h2) or (m1 <> m2) do begin
  inc(m1);            {прибавляем одну минуту}
    if m1 = 60 then begin
      inc(h1); h1 := h1 mod 24; m1 := 0;
      if h1 mod 12 = 0 then
        inc(res, 12)
      else
        inc(res, h1 mod 12);
    end
    else
      if m1 = 30 then
        inc(a);
 end;
```

(здесь `h1:m1` — начальное время, `h2:m2` — конечное время, `res` — количество ударов).

Решение второе: подсчет ударов. Пусть начальное время меньше конечного. Тогда можно подсчитать количество ударов следующим образом. Общее количество ударов складывается из:

- «часовых» ударов в моменты времени $(h1 + 1) : 00$, $(h1 + 2) : 00$, …, $h2 : 00$;
- «получасовых» ударов в моменты времени $(h1 + 1) : 30$, $(h1 + 2) : 30$, …, $h2 : 30$;
- «получасового» удара в момент времени $h1 : 30$, если $m1 < 30$;
- «получасового» удара в момент времени $h2 : 30$, если $m2 > 30$.

Теперь заметим, что за сутки часы бьют 180 раз. Поэтому в случае, когда конечное время меньше начального, можно поменять их местами, вычислить количество ударов по приведенной выше схеме и вычесть полученное число из 180.

Решение третье: явная формула. Обозначим через $U(t_1, t_2)$ количество ударов в промежуток времени $[t_1, t_2]$. Тогда

$$U(t_1, t_2) = \begin{cases} U(t_2, 0) - U(t_1, 0), & \text{если } t_2 \geqslant t_1; \\ 180 + U(t_2, 0) - U(t_1, 0), & \text{если } t_2 < t_1. \end{cases}$$

Для величины $U(t, 0)$ тоже можно вывести явную формулу. Например, такую:

```
U(h:m,0) = (h DIV 12) * 78 + {количество "часовых" ударов
                              за полные 12 часов            }
         + (h MOD 12) * ((h MOD 12) + 1) DIV 2
                             {количество "часовых" ударов
                              за остальное время            }
         + h                 {количество "получасовых" ударов}
         + ord (m > 30)      {еще один "получасовой" удар,
                              если m>30                     }
```

Автор задачи и разбора — В. М. Гуровиц.

Задача VIII-B. Выборы жрецов

Тема: задачи для начинающих.

После внимательного прочтения условия задачи становится понятно, что нужно считать из входного файла номера жрецов-покровителей и затем заменить некоторые из них на новые. Вот как это можно было реализовать.

1. Читаем номера жрецов-покровителей.

```
for i:=1 to N do
  read(a[i]);
```

2. Читаем пары (старый номер, новый номер).

```
for i:=1 to M do
  read(k,b[k]);
```

Теперь `b[k]` — либо новый номер, соответствующий старому номеру `k`, либо `0`, если номер не изменился.

3. Печатаем номера, при необходимости заменяя старый номер новым.

```
for i:=1 to N do
  if b[a[i]]>0 then write(b[a[i]],' ')
  else write(a[i],' ');
```

Автор разбора — В. М. Гуровиц.

Задача VIII-C. Представление чисел

Тема: делимость.

Пусть A и B — искомые числа, а d — их наибольший общий делитель. Поскольку $N = A + B$ и каждое из чисел A и B делится на d, то и N делится на d. Таким образом, в качестве кандидата на роль d можно рассматривать наибольший делитель числа N, меньший N.

Тогда $e = N : d$ — наименьший делитель числа N, больший 1.

Представим число N в виде суммы: $N = (N : e) + (N - N : e)$. Оба слагаемых делятся на d, поэтому можно положить: $A = N : e$, $B = N - N : e$. Таким образом, исходная задача свелась к нахождению наименьшего делителя e числа N, большего единицы.

Заметим, что либо $e \leqslant \sqrt{N}$, либо N — простое число и $e = N$. Действительно, если N составное и $e > \sqrt{N}$, то тогда $\frac{N}{e} \geqslant e > \sqrt{N}$, так как $\frac{N}{e}$ — тоже делитель числа N, отличный от 1, а e — наименьший делитель, отличный от 1. Но тогда $N = e \cdot \frac{N}{e} > \sqrt{N} \cdot \sqrt{N} = N$, что невозможно.

Следовательно, для нахождения числа e достаточно перебрать все числа от 2 до $\sqrt{N}$, и если среди этих чисел делитель числа N найден не будет, то положить $e = N$.

Заметим, что перебор всех чисел от 2 до N требует времени, выходящего за рамки временны́х ограничений задачи. Также слишком много времени требует поиск наибольшего делителя числа N, меньшего N.

Как часто бывает, текст решения в данном случае существенно короче, чем его описание:

```
i:=2;
while (N mod i > 0) AND (i<=sqrt(N)) do
  Inc(i);
if (N mod i > 0) then
  write(N-1,' ',1)
else
  write(N div i,' ',N - N div i);
```

Но оказывается, что это решение можно еще упростить!

Заметим, что если число N — простое, то НОД$(A, B) = 1$ для любых натуральных A и B, дающих в сумме N, поэтому можно вывести любые натуральные числа, сумма которых равна N. Пользуясь этим, можно еще сократить приведенную выше программу:

```
i:=2;
while (N mod i > 0) AND (i<=sqrt(N)) do
  Inc(i);
write(N div i,' ',N - N div i);
```

Совет: *при такой реализации* `sqrt(N)` *вычисляется при каждой итерации цикла* `while`. *Этого можно избежать, если заранее вычислить эту величину и использовать в условии цикла ее значение.*

Автор задачи — А. А. Петров, автор разбора — В. М. Гуровиц.

Задача VIII-D. Гексагон

Тема: задачи на разные темы.

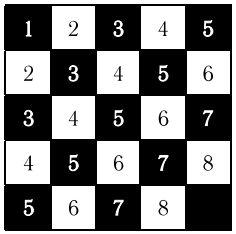
Пусть x — суммарная длина ходов, сделанных в направлении X, y — суммарная длина ходов в направлении Y, z — суммарная длина ходов в направлении Z. Заметим, что любую клетку поля можно задавать не тремя координатами, а двумя. В частности, клетка (X, Y, Z) совпадает с клеткой $(X - Z, Y + Z, 0)$, а также с клеткой $(X + Y, 0, Z - Y)$ и с клеткой $(0, Y + X, Z - X)$. Для того чтобы найти кратчайший путь из данной клетки, нужно записать ее координаты в таком виде, чтобы как можно больше координат оказались равными нулю.

Автор разбора — В. М. Гуровиц.

Задача VIII-E. Магия Копперфильда

Тема: игры и стратегии.

Сначала разберемся, в чем же состоит секрет фокуса. Раскрасим клетки квадрата в два цвета в шахматном порядке. Заметим, что после четного количества ходов мы всегда оказываемся в клетке того же цвета, что и начальная клетка, а после нечетного количества ходов мы всегда «меняем цвет». Таким образом, мы всегда знаем цвет клетки, на которую в данный момент указывает палец зрителя. Если мы будем каждый раз удалять клетки только соответствующего цвета, и при этом всегда оставлять связную фигуру, состоящую более чем из одной клетки, то цель будет достигнута.



Такой план можно реализовывать различными способами. Один из самых простых — удалять каждым ходом следующий диагональный ряд клеток. На рисунке числа обозначают номера ходов, во время которых удаляются соответствующие клетки.

Для вычисления номера клетки заметим, что каждая следующая клетка диагонали имеет номер на $N - 1$ больше, чем предыдущая клетка той же диагонали (если идти по диагонали налево и вниз).

Автор разбора — В. М. Гуровиц.

Задача VIII-F. Кубическое уравнение

Тема: делимость.

Решение этой задачи опирается на следующий несложный факт.

Если целое число x — решение уравнения n-й степени с ненулевым свободным членом, то x является делителем этого свободного члена.

Это несложно доказать: сумма (в нашем случае $Ax^3 + Bx^2 + Cx + D$) равна нулю, а все слагаемые, кроме последнего, делятся на x, следовательно, и последнее слагаемое, т. е. свободный член, делится на x.

Таким образом, если D отлично от нуля, то нам достаточно перебрать все делители числа D (включая отрицательные) и, подставив каждый из них в уравнение, выбрать те, которые являются корнями уравнения. Если же $D = 0$, то можно «сократить» на x, не забыв при этом корень $x = 0$. Если в полученном уравнении свободный член опять окажется равным нулю, нужно еще раз сократить на x и т. д.

Случай, когда $A = B = C = D = 0$ — единственный случай, когда уравнение имеет бесконечно много решений, и потому требуется вывести в выходной файл `-1`. Во всех остальных случаях уравнение либо имеет не более трех корней, либо вовсе не имеет корней. Поэтому отсортировать полученные корни по возрастанию не составит труда.

Заметим, что приведенное решение не зависит от того, является ли данное уравнение кубическим, т. е. равно ли A нулю.

Автор задачи — Е. В. Андреева, автор разбора — В. М. Гуровиц.

Задача VIII-G. Скорая помощь

Тема: делимость, перебор.

Сначала решим более простую задачу. По номеру квартиры, количеству квартир на одном этаже и количеству этажей в доме определим подъезд и этаж, где находится квартира. Пусть K — заданный номер квартиры, A — количество квартир на этаже, а M — количество этажей в доме. Тогда искомый номер подъезда будет равен $(K - 1) \operatorname{div} (M * A) + 1$. Для поиска номера этажа занумеруем квартиры в каждом подъезде, начиная с 1. Новый номер квартиры будет равен $K' = (K - 1) \bmod (M * A) + 1$, а искомый номер этажа $(K' - 1) \operatorname{div} A + 1$.

Для решения основной задачи будем перебирать возможное количество квартир на этаже (от 1 до 1000), проверять совпадение номера этажа $N2$ и подъезда $P2$ для заданной во входном файле квартиры $K2$ и, если полученные и имеющиеся данные совпадают, воспользуемся вышеописанным решением для квартиры, заданной номером $K1$.

При выводе ответа стоит учесть, что если правильных ответов несколько, то у них могут совпадать только этажи, только подъезды или ничего не совпадать.

Автор разбора — В. Ю. Антонов.

Задача VIII-H. A-функция от строчки

Тема: алгоритмы на строках.

Сведем эту задачу к другой, похожей задаче и воспользуемся методом динамического программирования.

Определим функцию $z(i)$ следующим образом: $z(i)$ = максимально возможному k, такому, что равны следующие строки:

$$S[1] + S[2] + S[3] + \ldots + S[k],$$
$$S[i] + S[i+1] + S[i+2] + \ldots + S[i+k-1],$$

где $S[i]$ — i-й символ строки S, а знак «+» означает, что символы записываются в строчку непосредственно друг за другом; $z(1)$ положим равным нулю.

Будем решать задачу нахождения $z(i)$ для всевозможных значений i от 1 до N.

Предположим, что новая задача решена для первых $i-1$ элементов строки. Среди уже посчитанных значений функции z найдем такое, z-функция для которого максимально «выступает», т. е. найдем такое $l < i$, что $l + z(l) - 1$ максимально. Возможны два случая.

1. $l + z(l) - 1 < i$. В этом случае $z(i)$ будем вычислять, используя определение.
2. $l + z(l) - 1 \geqslant i$. Если $i + z(i-l+1) < l + z(l)$, то $z(i)$ будет равно $z(i-l+1)$. Если же $i + z(i-l+1) \geqslant l + z(l)$, то значения функции z будем считать используя определение, учитывая, что $l + z(l) - i$ символов уже совпадает.

Покажем, что данный алгоритм можно реализовать так, чтобы количество выполняемых операций было пропорционально длине заданной строки S.

Заметим, что l не обязательно пересчитывать на каждом шаге. Предположим, что для $(i-1)$-го элемента строки уже вычислена функция z и значение l. Тогда для i-го элемента строки l может либо не измениться, либо поменять значение на $i-1$.

Сумма l и $z[l]$ может только увеличиваться с увеличением l, при этом она может увеличиваться не более чем N раз. Поэтому при подсчете z при помощи определения (первый случай и часть второго случая) потребуется не более N операций.

Для решения исходной задачи модифицируем заданную строку S, поставив после нее любой символ, которого в ней нет. После этого символа запишем перевернутую строку S ($S[N] + S[N-1] + \ldots + S[2] + S[1]$). Для полученной строки посчитаем z-функцию, которая была описана выше. Ответом будут являться значения функции для $2N+1$, $2N$, …, $N+2$ элементов.

Автор задачи — Б. О. Василевский, автор разбора — В. Ю. Антонов.

Задача VIII-I. Олимпиада по алхимии

Тема: алгоритмы на графах — алгоритм Дейкстры.

Решение этой задачи использует один из стандартных алгоритмов на графах: алгоритм Дейкстры. Не будем приводить здесь его описание, так как его можно найти во многих книгах (к примеру, в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ» (М.: МЦНМО, 2000)).

Для каждого населенного пункта при помощи алгоритма Дейкстры найдем наименьшее время, за которое гонец может до него добраться. После этого остается выбрать населенные пункты, которые являются городами, и отсортировать их по неубыванию вычисленного времени.

Автор задачи — Я. А. Леонов, автор разбора — В. Ю. Антонов.

Задача VIII-J. Лотерея

Тема: структуры данных, динамическое программирование.

Введем несколько определений.

Глубиной вершины v называется минимальное количество ребер, которое требуется пройти, чтобы попасть из корня дерева в v.

Глубина дерева — максимальная из глубин вершин, принадлежащих дереву.

Сыном вершины v называется вершина, которая соединена ребром с v и имеет глубину бóльшую, чем глубина v.

Представим множество M-значных чисел в K-ичной системе счисления в виде дерева следующим образом. Глубина дерева будет равна M, и из каждой вершины будет выходить по K ребер, соответствующих различным цифрам. Каждому M-значному числу в K-ичной системе будет соответствовать путь из корня дерева в одну из висячих вершин глубины M. *Путь для числа J* выбирается следующим образом: если мы находимся на глубине i, то идем по ребру, на котором написана $(i+1)$-я цифра числа J.

Путем от корня до данной вершины v будем называть число J такое, что при прохождении пути, соответствующего J, мы окажемся в вершине v.

В каждой вершине будем хранить количество чисел, первые цифры которых совпадают с путем от корня до данной вершины. Тогда исходная задача сводится к поиску пути в дереве, для которого сумма чисел, записанных на пройденных в пути вершинах, умноженных на соответствующие коэффициенты, будет минимальна. Коэффициент для корня равен 0, для вершин, находящихся на глубине 1, равен A_1, а для вершин, находящихся на глубине $i > 1$, равен $A_i - A_{i-1}$.

Переформулированную таким образом задачу можно решить при помощи метода динамического программирования. Будем решать задачу «снизу вверх», т. е. будем искать пути с минимальными суммами в поддеревьях, корни которых находятся на глубине i, а параметр i будем перебирать от M до 0. Предположим, что ответ для вершин, находящихся на глубине $i+1$, уже посчитан. Тогда ответом для вершины на глубине i будет являться сумма числа, записанного в вершине, умноженного на соответствующий коэффициент, и минимума из ответов для сыновей данной вершины. Инициализацией для алгоритма является нахождение ответа для вершин, находящихся на глубине M. Это делается нахождением произведения соответствующего коэффициента на количество билетов с номером, который совпадает с путем от корня дерева до данной вершины.

Заметим, что для данной задачи не требуется полностью строить дерево, в нем может быть не более чем NM элементов. А весь алгоритм потребует порядка NMK операций.

При помощи вышеописанного решения рассчитывается минимальная сумма выигрыша. Для вывода номера билета, который является выигрышным, для каждой вершины нужно запоминать, какой из ее сыновей был выбран.

Автор задачи — А. А. Лунев, автор разбора — В. Ю. Антонов.

Задача VIII-К. Коммерческий калькулятор

Тема: структуры данных.

Для решения данной задачи воспользуемся *жадным алгоритмом*. Среди данных n чисел выберем два наименьших, сложим их и запишем полученную сумму вместо этих чисел. Теперь из полученного набора из $n-1$ чисел опять выберем два наименьших и повторим операцию. Так будем действовать до тех пор, пока не вычислим сумму всех n чисел. Например,

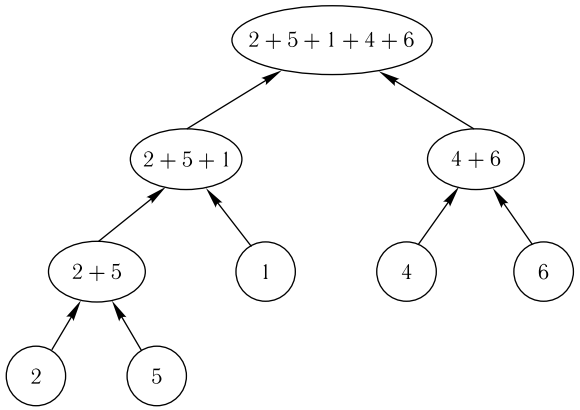
для набора (1, 2, 4, 5, 6) алгоритм будет работать следующим образом:

$$\underline{1}, \underline{2}, 4, 5, 6 \to \underline{3} = 1 + 2, \underline{4}, 5, 6 \to$$
$$\to 7 = 3 + 4, \underline{5}, \underline{6} \to \underline{7}, \underline{11} = 5 + 6 \to \mathbf{18} = 7 + 11.$$

Сначала покажем, как эффективно реализовать данный алгоритм, а затем докажем, что предложенный *жадный* способ вычисления суммы является самым дешевым.

Если для хранения набора чисел использовать отсортированный массив, то нам потребуется порядка N^2 операций для вставки в массив N новых значений, что не удовлетворяет ограничениям по времени. Поэтому для реализации предложенного алгоритма воспользуемся структурой данных *куча*. Не будем здесь приводить описание этой структуры, т. к. его можно найти во многих книгах (к примеру, в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ» (М.: МЦНМО, 2000)). В этом случае количество операций будет пропорционально $N \log N$, что удовлетворяет всем ограничениям задачи.

Теперь докажем, что указанный способ вычисления суммы — самый дешевый. Нам достаточно доказать, что существует самый дешевый способ вычисления, при котором на первом шаге складываются два самых маленьких числа. Тогда утверждение будет доказано по индукции. Пусть это не так. Рассмотрим самый выгодный порядок операций. Построим дерево, соответствующее данному порядку вычислений. Как оно устроено, понятно из примера на рисунке.



Заметим, что то, сколько раз мы выплачиваем 5 % каждого из данных n чисел, определяется глубиной этих чисел в дереве (поскольку именно столько раз эти числа участвуют в сложениях).

Пусть в таком дереве две вершины с минимальными значениями m и n не являются братьями и находятся на глубине h_m и h_n соответственно. Рассмотрим какие-либо две вершины-брата l и r, находящиеся на самом нижнем уровне на глубине H (в данном примере — вершины 2 и 5). Поменяем их с вершинами m и n (если одна из вершин l и r совпадает с одной из вершин m и n, то ее менять не нужно) — в нашем примере нужно поменять вершины 1 и 5. Посмотрим, как изменится выплачиваемая сумма. Она увеличится на $((H-h_m)\cdot m+(H-h_n)\cdot n)\cdot 0,05$ и уменьшится на $((H-h_m)\cdot l+(H-h_n)\cdot r)\cdot 0,05$. Поскольку $m\leqslant l$, и $n\leqslant r$, то $(H-h_m)\cdot m+(H-h_n)\cdot n<((H-h_m)\cdot l+(H-h_n)\cdot r)$, в итоге наша сумма может только уменьшиться, что противоречит тому, что мы рассматриваем наилучший порядок действий. Мы пришли к противоречию, предположив, что сложение двух наименьших чисел не приводит к наилучшему алгоритму. Следовательно, наш алгоритм является оптимальным, что и требовалось доказать.

Отметим, что аналогичный жадный алгоритм используется для построения кодов Хаффмена, применяющихся для сжатия текстов. Об этом также можно прочитать в книге Т. Кормена, Ч. Лейзерсона, Р. Ривеста «Алгоритмы: построение и анализ» (М.: МЦНМО, 2000).

Авторы разбора — В. Ю. Антонов и В. М. Гуровиц.

Задача VIII-L. Пересечение кубов

Тема: геометрия.

Задачу можно было решить двумя принципиально разными способами. Первый способ (предложенный Б. О. Василевским) — вывод точных формул для вычисления объема. Второй способ (предложенный А. О. Юрьевым) основывается на приближенных вычислениях.

Первый способ.

Рассматриваемое решение не сильно отличается от решения более общей задачи: надо найти фигуру (многогранник) T_3 пересечения двух выпуклых многогранников T_1 и T_2, причем известна их общая внутренняя точка $O(0,\,0,\,0)$.

Определение. Плоскость назовем *собственной* для многогранника T, если она содержит какую-либо его грань (многоугольник).

Идея решения состоит в том, чтобы отсекать от T_1 части, не принадлежащие T_2. В силу выпуклости это можно реализовать так: для каждой собственной для T_2 плоскости P выбрасываем все точки, лежащие по другую сторону от точки O относительно P, и сохраняем полученную фигуру.

В конце концов получим многогранник T_3. Таким образом, про многогранник T_2 нам нужно знать только список его собственных плоскостей.

Сначала договоримся, как мы будем хранить многогранник. Для данного решения удобно представить его как набор граней (многоугольников). Для каждой такой грани будем хранить плоскость, в которой она находится (соответствующая собственная плоскость для данного многогранника), и упорядоченный список вершин (две соседние вершины соединены ребром). Обсудим это подробнее для случая кубов.

Предложение 1. При отсечении от первого куба всего, что не принадлежит второму, в любой момент у любой грани существует точка L, которая «сохранится» до конца процесса, т. е. которая принадлежит T_3.

В самом деле, общая вписанная сфера T_1 и T_2 (с центром O радиуса 1) принадлежит T_3, более того, все грани T_1 и T_2 касаются ее, следовательно, все «промежуточные» грани будут тоже касаться ее. Тогда точка касания данной грани и сферы и будет искомой точкой L.

Эта точка обладает еще одним полезным свойством.

Предложение 2. Если опустить из O перпендикуляр на плоскость грани, его основание будет совпадать с L.

Это следует из свойства касательной плоскости к сфере.

Предложение 3. Пусть плоскость задается в пространстве уравнением $Ax+By+Cz+D=0$. Тогда вектор $(A,\,B,\,C)$ ортогонален этой плоскости. (Это утверждение является полным аналогом теоремы планиметрии о том, что если прямая задается уравнением $Ax+By+C=0$, то вектор $(A,\,B)$ перпендикулярен этой прямой.)

Теперь мы обладаем достаточными знаниями для того, чтобы понять, как эффективно хранить плоскость для каждой грани. Пусть она задается уравнением $Ax+By+Cz+D=0$ и точка L (описанная выше) имеет координаты $(a,\,b,\,c)$. Тогда векторы $(a,\,b,\,c)$ и $(A,\,B,\,C)$ коллинеарны. Пусть $d\cdot(a,\,b,\,c)=(A,\,B,\,C)$, $d\neq 0$. Тогда уравнение плоскости примет вид: $ax+by+cz+D/d=0$. Если взять $x=a$, $y=b$ и $z=c$, то равенство будет верным, так как L лежит в этой плоскости. Получаем: $a^2+b^2+c^2=-D/d$. Заметим, что L лежит также на сфере радиуса 1, т. е. $a^2+b^2+c^2=1$. Таким образом, уравнение плоскости можно записать через координаты точки L: $ax+by+cz=1$. Поэтому вместо уравнения плоскости будем хранить координаты точки L (они понадобятся также при подсчете объема).

Теперь научимся упорядочивать список вершин данной грани R. Перед этим сделаем так, чтобы в списке не было двух совпадающих вершин.

Предложение 4. Чтобы упорядочить вершины выпуклого многоугольника R в порядке обхода, достаточно отсортировать их по полярному углу в данной плоскости относительно точки L.

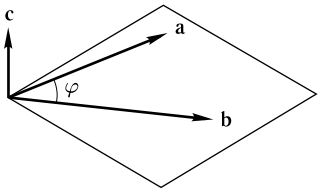
Это следует из того, что многоугольник R выпуклый и L лежит внутри R.

Определение. Ориентированным объемом трех векторов $\vec{\mathbf{a}}(a_1, a_2, a_3)$, $\vec{\mathbf{b}}(b_1, b_2, b_3)$, $\vec{\mathbf{c}}(c_1, c_2, c_3)$ назовем величину

$$V(\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{c}}) = a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2 - a_3b_2c_1 - a_1b_3c_2 - a_2b_1c_3.$$

Модуль этой величины равен объему параллелепипеда, с ребрами, образованными векторами $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$, $\vec{\mathbf{c}}$. Иногда эта величина будет положительна, иногда — отрицательна. Когда какой знак будет получаться, обсудим подробнее.

Пусть все три вектора отложены от начала координат. Представим себе, что мы сидим на конце вектора $\vec{\mathbf{c}}$ и смотрим на плоскость, содержащую $\vec{\mathbf{a}}$ и $\vec{\mathbf{b}}$. Рассмотрим такой минимальный угол $\varphi > 0°$, при повороте на который по часовой стрелке вокруг начала координат в плоскости векторов $\vec{\mathbf{a}}$ и $\vec{\mathbf{b}}$ вектор $\vec{\mathbf{a}}$ станет сонаправлен вектору $\vec{\mathbf{b}}$. Для всех $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$, $\vec{\mathbf{c}}$ таких, что $\varphi < 180°$, знак $V(\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{c}})$ будет одинаков. Аналогично, для всех $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$, $\vec{\mathbf{c}}$ таких, что $\varphi > 180°$, знак $V(\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{c}})$ также будет одинаков, причем он будет отличаться от знака ориентированного объема в предыдущем случае (это обобщение ориентированной площади для двух векторов на плоскости).



Вернемся к упорядочиванию вершин. Рассмотрим вектор $\vec{\mathbf{c}} = \overrightarrow{OL}$ с координатами (l_1, l_2, l_3) — это координаты точки L. Пусть надо сравнить две вершины грани $A(a_1, a_2, a_3)$ и $B(b_1, b_2, b_3)$. Тогда вектор $\vec{\mathbf{a}} = \overrightarrow{OA} - \overrightarrow{OL}$, $\vec{\mathbf{b}} = \overrightarrow{OB} - \overrightarrow{OL}$, где $\overrightarrow{OA}$ и $\overrightarrow{OB}$ — радиус-векторы точек A и B. Тогда векторы $\vec{\mathbf{a}}$ и $\vec{\mathbf{b}}$ лежат в плоскости, параллельной $l_1x + l_2y + l_3z = 1$ и проходящей через O. Условимся теперь, что если $V(\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{c}}) < 0$, то вершина A идет в нашем упорядочивании раньше вершины B.

Таким образом мы можем отсортировать список вершин каждой грани.

Теперь перейдем непосредственно к описанию алгоритма решения задачи.

Из первого куба надо «извлечь» собственные для него плоскости, для этого достаточно вычислить координаты соответствующих точек L.

Инициализация второго куба T_2 должна проходить примерно так: считываем вершины, для каждой грани составляем список вершин, вычисляем координаты точки L для каждой грани — это координаты центра грани (а так как это квадрат, то их несложно найти). Потом упорядочиваем список.

Теперь каждой гранью куба T_1 (пусть P — плоскость, содержащая грань) надо отсечь от текущего многогранника T (который сначала равен T_2) все точки (назовем их *ненужными*), лежащие по другую сторону от точки O относительно плоскости P. На самом деле это будет выглядеть так: будут создаваться новые вершины на гранях, пересекающих P (точнее — точки пересечения ребер с P), удаляться ненужные вершины граней, создаваться новые грани из точек пересечения P и граней T.

Обсудим это подробнее. Подумаем, каким может быть взаимное расположение T и плоскости P.

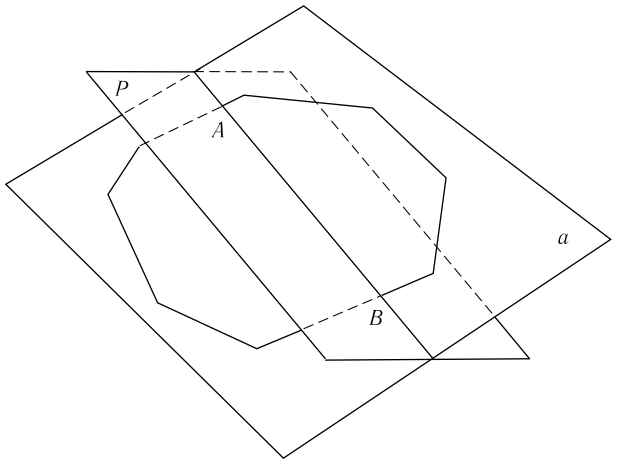
Если P совпадает с какой-либо плоскостью, содержащей какую-либо грань T, то P не может повлиять на T (не существует точек T, лежащих по другую сторону от O относительно P).

В противном случае P пересекает некоторые грани T.

(1) Нам нужен список всех точек, лежащих в P и при этом на сторонах граней T. Понятно, что именно из них получится новая грань — многоугольник, являющийся сечением T плоскостью P.

(2) Во-вторых, надо модифицировать грани, пересекающие P, так, чтобы на них не осталось ненужных точек.

Придумаем, как выполнить (2), потом в результате выполним и (1).



У нас будет глобальный список найденных точек (1). Время от времени будем его пополнять.

Рассмотрим произвольную грань T, пересекающую P. Пусть эта грань лежит в плоскости a. Плоскость P пересекается плоскостью a по прямой. Эта прямая имеет с гранью (многоугольником) общие точки по условию. Их количество либо 1, либо 2, либо бесконечно много.

Если 1 — то P касается грани и ненужных точек на грани нет. Просто запомним в список эту точку.

Если 2 — то нужно удалить из списка вершин грани ненужные вершины, добавить в него эти две точки пересечения и сделать обновление: удалить совпадающие вершины из списка и отсортировать (понятно, что сортировка необходима для удобства поиска пересечений — достаточно одного прохода по списку). Найденные две точки пересечения надо занести в список точек (1).

Последний случай (бесконечно много общих точек у грани и прямой) возможен тогда и только тогда, когда прямой полностью принадлежит какая-то сторона S грани. С точки зрения реализации этот случай не отличается от предыдущего. В список заносятся вершины стороны S. Никакой внутренней вершины в списке не должно оказаться, так как мы ищем пересечение отрезка и плоскости в том случае, когда оба конца не лежат в этой плоскости, иначе это просто бессмысленно.

Предложение 5. Рассмотрим плоскость P: $ax + by + cz = 1$ и две точки $A(x_1, y_1, z_1)$ и $B(x_2, y_2, z_2)$. Отрезок AB пересекает P по внутренней (отличной от A и B) точке тогда и только тогда, когда величины

$$v_1 = a \cdot x_1 + b \cdot y_1 + c \cdot z_1 - 1,$$
$$v_2 = a \cdot x_2 + b \cdot y_2 + c \cdot z_2 - 1$$

имеют разные знаки.

Упражнение. Доказать, что координаты точки пересечения (точки C) плоскости $ax + by + cz = 1$ и отрезка, соединяющего две точки $A(x_1, y_1, z_1)$ и $B(x_2, y_2, z_2)$ (в случае, если существует точка пересечения и она является внутренней для AB) такие:

$$C\left(\frac{x_1 \cdot v_2 - x_2 \cdot v_1}{v_2 - v_1}, \frac{y_1 \cdot v_2 - y_2 \cdot v_1}{v_2 - v_1}, \frac{z_1 \cdot v_2 - z_2 \cdot v_1}{v_2 - v_1}\right),$$

где

$$v_1 = a \cdot x_1 + b \cdot y_1 + c \cdot z_1 - 1,$$
$$v_2 = a \cdot x_2 + b \cdot y_2 + c \cdot z_2 - 1.$$

Вернемся к (1). У нас имеется список всех точек, принадлежащих одновременно P и какому-нибудь ребру многогранника T. Выпуклая оболочка этих точек, очевидно, образует новую грань (сечение). Ни одна из этих точек не лежит внутри оболочки, так как все эти точки лежат на границе многогранника (не внутри). Удалим совпадающие точки в списке, отсортируем (у нас имеются координаты точки L для этой плоскости — (a, b, c) — коэффициенты уравнения плоскости). Получилась полноценная грань нового многогранника. Добавим ее в список граней.

Такие действия нужно проделать для каждой собственной плоскости.

Осталось сказать о способах вычисления объема имеющегося у нас многогранника. Напомним, что $O(0, 0, 0)$ лежит строго внутри него. Поэтому искомый объем равен сумме объемов пирамид с вершиной O и каждой гранью в качестве основания.

Напомним, что объем пирамиды равен $H \cdot S/3$, где H — расстояние от вершины до плоскости основания, S — площадь основания. В данном случае $H = 1$ — одно из свойств точки L для данной плоскости. Остается научиться находить площадь грани.

Понятно, как на плоскости находить площадь многоугольника $A_1 \dots A_n$, если известны координаты какой-нибудь точки M внутри него. Можно просто просуммировать площади таких треугольников: OA_iA_{i+1}, $1 \leqslant i \leqslant n$, считая, что $A_{n+1} = A_1$. Но как найти площадь треугольника в пространстве?

Уточним, что нам известно про такой треугольник. Одна из его вершин — точка L, две другие (A и B) — это соседние вершины некоторой грани. Можно посчитать площадь, например, используя формулу Герона, или же посчитать синус угла между векторами $\overrightarrow{LA}$ и $\overrightarrow{LB}$ с помощью скалярного произведения и найти площадь как половину произведениия длин двух сторон на синус угла между ними.

Можно эту площадь найти другим способом. Вспомним, что абсолютная величина $V(\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{c}})$ равна объему параллелепипеда, построенного на векторах $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$, $\vec{\mathbf{c}}$. В данном случае полезно посмотреть $V(\overrightarrow{OL}, \overrightarrow{LA}, \overrightarrow{LB}) = V_0$. $|\overrightarrow{OL}| = 1$ и $\overrightarrow{OL}$ перпендикулярен плоскости LAB. Следовательно (по формуле объема параллелепипеда), V_0 численно равно площади параллелограмма $LABC$, где векторы $\overrightarrow{LA}$ и $\overrightarrow{BC}$ равны. А это — удвоенная площадь LAB, что нам и было нужно.

Замечание. Описание реализации гораздо длиннее самой реализации.

Второй способ.

Рассмотрим тело, получившееся при пересечении двух кубов. Будем вычислять его объем приближенно, используя *метод трапеций*. Так как пересечение двух кубов не содержит криволинейных поверхностей, то результат приближенных вычислений будет очень близок к верному значению даже при небольшой точности вычислений.

Разобьем отрезок $[-1; 1]$ оси Oz на N равных частей (от величины N будет зависеть точность вычислений). Пусть длина каждой части равна $\Delta z = 2/N$. Будем строить сечения тела плоскостями, проходящими параллельно плоскости xOy через середины построенных отрезков. По теореме Лагранжа на каждом отрезке $[z_i, z_{i+1}]$ найдется точка z_0 такая, что объем V_i части фигуры, ограниченной плоскостями, проходящими через точки z_i и z_{i+1}, удовлетворяет равенству: $V_i = \mathrm{Cut}(z_0)\Delta z$, где $\mathrm{Cut}(z)$ — площадь сечения тела плоскостью, проходящей через точку z.

Таким образом, чем меньше Δz (соответственно больше N), тем лучше точка $(z_i + z_{i+1})/2$ приближает z_0 и тем точнее вычисляется объем.

Остается вопрос: как вычислять Cut(z)? Очевидно, что мы можем сначала строить сечение каждого из кубов отдельно, а затем находить площадь пересечения получившихся многоугольников. Более того, заметим, что сечение куба A не зависит от координаты z: оно является квадратом с вершинами в точках (1; 1), (1; −1), (−1; −1), (−1; 1). Сечение куба B можно найти, если последовательно пересечь плоскость со всеми его ребрами, а затем построить из получившихся точек выпуклый многоугольник.

Пересечение многоугольников ищется по стандартной схеме, его вершинами будут:

- все вершины первого многоугольника, принадлежащие второму;
- все вершины второго многоугольника, принадлежащие первому;
- точки попарного пересечения всех сторон обоих многоугольников.

Автор задачи — Б. О. Василевский,
авторы разбора — Б. О. Василевский и А. Ю. Юрьев.

Олимпиада IX. Командная олимпиада 2005–2006 учебного года

Задача IX-A. Результаты олимпиады

Тема: задачи для начинающих.

Рассмотрим сначала случай, когда задача всего одна. Пусть по этой задаче уже набрано points очков и в момент времени t сделана очередная попытка, причем сданное решение проходит L тестов. Обозначим через A общее количество тестов по данной задаче, через B — количество тестов на 1 балл. Теперь рассмотрим несколько возможных случаев, объединив их в таблицу.

<table>
<tr><th>points/L</th><th>$L=-1$</th><th>$0 \leqslant L < B$</th><th>$B \leqslant L < A$</th><th>$L = A$</th></tr>
<tr><td>0</td><td>—</td><td rowspan="2">Inc(attempts);</td><td>points:=1;
penalty:=t+attempts*20;
Inc(attempts);</td><td rowspan="2">points:=2;
penalty:=attempts*20+t;</td></tr>
<tr><td>1</td><td>—</td><td>Inc(attempts);</td></tr>
<tr><td>2</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>

Здесь penalty — штрафное время, attempts — количество попыток.

Для того чтобы решить исходную задачу, нужно заменить переменные A, B, penalty, attempts, points на соответствующие массивы.

Авторы задачи — В. А. Матюхин и жюри Московской олимпиады,
автор разбора — В. М. Гуровиц.

Задача IX-B. Сокращение дроби

Тема: делимость.

Как известно, чтобы сократить дробь, нужно разделить ее числитель и знаменатель на их наибольший общий делитель (НОД). НОД(a, b) — это натуральное число, не превосходящее a и b, т. е. в ограничениях задачи — не превосходящее 100. Его можно найти, перебирая все натуральные числа от 100 до 1 до тех пор, пока не встретится число, на которое делится и a, и b. Заметим, что этот алгоритм будет работать, даже если число a отрицательное или равно нулю.

Автор разбора — В. М. Гуровиц.

Задача IX-C. Современники

Тема: сортировка.

Разобьем решение задачи на несколько шагов.

Подготовительный этап.

1. Про каждого человека выясним, дожил ли он до 18 лет.
2. Для всех людей, доживших до 18 лет, определим две даты: день, когда они впервые могли участвовать в обсуждении (день 18-летия), и день, когда они могли участвовать в обсуждении в последний раз (день перед днем 80-летия или перед днем смерти). Первую дату будем называть *начальной*, а вторую — *конечной*.
3. Составим общий список всех таких дат, отметив для каждой даты ее тип (*начальная* или *конечная*) и человека, к которому она относится. Отметим, что мы записываем в один массив и начальные, и конечные даты.
4. Отсортируем этот массив по возрастанию дат. В случае равных дат поставим начальные даты перед конечными.

Основной этап.

Будем идти по отсортированному списку слева направо. В каждый момент мы рассматриваем некоторое множество людей, которые одновременно могут принимать участие в обсуждении. Поясним это на примере. Пусть после сортировки получился такой список:

Дата	1.1.1	1.1.1	2.1.1	3.1.1	4.1.1	5.1.1
Человек №	1	2	1	3	2	3
Тип даты	начало	начало	конец	начало	конец	конец
Позиция	(1)	(2)	(3)	(4)	(5)	(6)

Пройдем по таблице слева направо. В первой позиции у нас появляется один человек, который может участвовать в обсуждении (№ 1). Во второй позиции к нему добавляется еще один человек, и множество людей, которые могут что-либо обсуждать, принимает вид $\{1, 2\}$. В третьей позиции множество уменьшается, и в нем остается только человек № 2. В четвертой позиции множество принимает вид $\{2, 3\}$, в пятой — $\{3\}$, в шестой позиции множество становится пустым.

Утверждение. Рассмотрим все позиции, соответствующие начальным датам, такие, что сразу после них идет позиция, соответствующая конечной дате (в нашем примере это позиции 2 и 4). Рассмотрим множества людей, соответствующие этим позициям (в нашем примере — $\{1, 2\}$ и $\{2, 3\}$). Эти и только эти множества будут максимальными. Кроме того, все эти множества различны.

Доказательство. 1. Докажем, что все полученные множества максимальны. Пусть в некоторой позиции, удовлетворяющей условиям утверждения, мы получили множество, которое не является максимальным. Поскольку в этой позиции написана начальная дата для некоторого человека, то ни в одно из ранее полученных множеств этот человек входить не мог. Поэтому наше множество не содержится (и, кстати, не совпадает) ни с одним из множеств, которые были получены до него. С другой стороны, в следующей позиции записана конечная дата для некоторого человека, следовательно, ни в одном из следующих множеств этот человек не встретится, из чего следует, что ни одно из следующих множеств не содержит наше множество (и не совпадает с ним).

2. В первом пункте мы уже доказали, что выбранное нами множество не совпадает ни с какими другими множествами, следовательно, все выбранные нами множествами различны.

3. Наконец докажем, что других максимальных множеств нет. Пусть это не так, и существует пропущенное нами максимальное множество M. Рассмотрим позицию, в которой оно было получено. Ясно, что в этой позиции записана начальная дата, иначе предыдущее множество содержало бы наше. Также ясно, что в следующей позиции записана конечная дата, иначе наше множество содержалось бы в следующем. Но тогда данная позиция удовлетворяет условиям утверждения. Полученное противоречие доказывает, что других максимальных множеств нет.

Теперь обсудим структуры данных, которые могут быть использованы для реализации предложенного алгоритма. В каждый момент времени нам нужно хранить номера людей, которые входят в текущее множество современников. Нам необходимо уметь быстро добавлять в это множество элементы и удалять из него элементы. Для этого можно было бы воспользоваться стандартным типом `set` в Паскале или реализовать множество самостоятельно на базе массива. Но нам нужно еще уметь выводить все элементы множества. Для предложенных структур данных эта операция занимает время, пропорциональное максимальному размеру множества или длине всего массива. Этого не достаточно, чтобы решить задачу в предложенных ограничениях.

Покажем, как можно реализовать множество, чтобы сложность вывода всех элементов была пропорциональна количеству элементов. Поскольку в задаче имеется ограничение на размер выходного файла, то ограничено и суммарное количество элементов в максимальных множествах. Итак, сейчас мы реализуем множество на базе массива. Объявим массивы `prev` и `next`, количество элементов в каждом из которых будет равно общему количеству людей. Если в данный момент человек с номером k присутствует в нашем множестве, то в элементе `next[k]` будем хранить номер человека, который был включен в множество после k-го (или 0, если

после k-го человека в множество еще никто добавлен не был), а в элементе `prev[k]` — номер человека, включенного в множество непосредственно перед k-м. Кроме того, будем хранить в отдельной переменной `beg` номер первого человека в текущем множестве. Таким образом, мы объединяем элементы множества в *двусвязный список*, что дает нам возможность при выводе переходить непосредственно от предыдущего человека к следующему.

Автор задачи — Е. В. Андреева, автор разбора — В. М. Гуровиц.

Задача IX-D. Тройки чисел

Тема: делимость.

Все решения данного уравнения имеют вид: $a = pn^2$, $b = p(n-1)^2$ (это будет доказано ниже).

Таким образом, для каждого p из отрезка $[N, M]$ требуется найти все такие n, что $N \leqslant p(n-1)^2 < pn^2 \leqslant M$, т. е. $N/p \leqslant (n-1)^2 < n^2 \leqslant M/p$, что равносильно неравенству $\sqrt{\frac{N}{p}} \leqslant n - 1 < n \leqslant \sqrt{\frac{M}{p}}$. Таким образом, количество решений на 1 меньше, чем количество целых чисел на отрезке $\left[\sqrt{\frac{N}{p}}, \sqrt{\frac{M}{p}}\right]$. Итак, решение задачи сводится к проверке на простоту каждого числа от N до M и нахождению количества решений для каждого из найденных простых чисел указанным выше способом.

Докажем теперь, что все решения имеют указанный вид. Для этого достаточно в данном уравнении перенести $\sqrt{b}$ в правую часть и возвести обе части в квадрат. Получим: $a = b + 2\sqrt{bp} + p$. Поскольку a, b и p — целые числа, то и $\sqrt{bp}$ — целое число, т. е. b равно произведению p и квадрата некоторого целого числа, что и требовалось доказать.

Автор задачи — А. О. Тимофеев, автор разбора — В. М. Гуровиц.

Задача IX-E. T2005

Тема: сортировка, строки.

Будем хранить для каждого слова последовательность клавиш, которые нужно нажать, чтобы ввести полностью данное слово. Пересортируем словарь в порядке возрастания числовых последовательностей, причем воспользуемся *устойчивой* сортировкой, не меняющей порядок слов, которым соответствуют равные числовые последовательности.

Напишем функцию, которая по набору нажатых клавиш определяет, сколько слов в словаре имеют такое начало, и если такое слово ровно одно, возвращающую это слово. Кроме того, нам понадобится аналогичная функция, находящая слово, которое полностью определяется заданным набором клавиш.

Теперь перейдем к описанию основного алгоритма. Для хранения введенных символов будем использовать переменную `v` строкового типа следующим образом. Будем по одной считывать нажимаемые клавиши и дописывать соответствующие цифры в конец строки `v`. После считывания каждого символа будем проверять, сколько слов в словаре имеют соответствующее начало (это можно делать, используя двоичный поиск). Возможны несколько случаев:

1) если такое слово ровно одно, то печатаем его в выходной файл, после него печатаем пробел и обнуляем переменную `v`;
2) если таких слов нет, то опять обнуляем переменную `v` и пропускаем все клавиши во входном файле до следующей единицы включительно;
3) если очередная нажатая клавиша — 1, то проверяем, сколько слов в словаре соответствуют этому набору нажатых клавиш. Если такое слово одно, то действуем как в пункте 1), если таких слов несколько — выбираем первое из них, если таких слов нет — действуем как в пункте 2).

Автор задачи — Е. В. Андреева, автор разбора — В. М. Гуровиц.

Задача IX-F. Роботы

Тема: алгоритмы на графах — поиск в ширину.

Переведем задачу на язык графов. Вершинами нашего графа будут залы, а ребрами — туннели. Граф взвешенный, вес каждого ребра — 1. Тогда роботам требуется за кратчайшее время встретиться в вершине или на ребре. Сразу отметим два факта.

1. Если робот может прийти в вершину за время t, он может повторно попасть туда же за время $t + 2$.

2. Если роботы встречаются в туннеле (на ребре), то это происходит в середине, причем часть роботов идет по ребру в одну сторону, а часть — в другую (иначе они бы уже встретились в одной вершине).

Теперь становится понятно, как найти минимальное время, за которое роботы могут собраться в данной вершине v: смотрим, за какое наименьшее четное (нечетное) время роботы придут в v (это наибольшая из длин кратчайших четных (нечетных) путей каждого робота до v). На ум приходит мысль разбить каждое ребро на две равные части, т. е. добавить вершины, которые будут соответствовать серединкам. Но мы так делать

не будем, так как это решение требует некоторой аккуратности. Если мы знаем время T_i, за которое данный робот добирается до вершины i, то до центра ребра uv робот доберется за $\min(T_u, T_v) + 1/2$. Появляется следующая идея: для каждого робота найдем кратчайшие четный и нечетный пути до всех вершин графа. После чего для каждой вершины найдем время сбора в ней. То же самое сделаем с каждым ребром (сбор в центре) и возьмем общий минимум. Если собраться нигде не удалось (ясно, что это бывает только в случае, когда найдутся два робота в разных компонентах связности графа), ответом будет -1.

Начнем планировать решение. Самый интересный момент — поиск кратчайших путей (четных и нечетных). Для этого слегка модифицируем обход в ширину, вычисляя сразу 2 величины — четное и нечетное расстояния. Граф будем хранить в виде списков смежных вершин. При считывании входных данных можно сразу удалить кратные ребра, но ни в коем случае нельзя удалять петли (если в вершине есть петля, это позволяет роботу вернуться туда же не за 2 хода, а за 1). Все расстояния в задаче полуцелые (т. е. становятся целыми при умножении на 2), а значит, если считать, что вес ребра — 2, то все вычисления можно проводить с помощью целых типов данных (это дает хороший выигрыш во времени, а ограничение на время работы программы в задаче довольно строгое). В конце же просто разделим ответ на 2.

Теперь можно привести полный текст решения задачи на языке Паскаль.

```
program robots;

type integer = longint;

var m : array [1..400, 1..400, 0..1] of integer;
    l : array [1..400, 0..400] of integer;
    n, k, nr, i, j, t : integer;
    r : array [1..400] of integer;
    a, b : array [1..20000] of integer;
```

m[i][j][k] — матрица кратчайших путей: i — вершина, из которой мы ищем путь, j — пункт назначения, k — четность пути (0 или 1); l[i][j] — список смежных вершин: i — вершина, для которой хранится список, l[i] — сам список (l[i][0] — его длина); a[i] и b[i] — концы ребер.

```
function min(i, j : integer) : integer;
begin
   if i < j then min := i else min := j;
end;
```



```
function max(i, j : integer) : integer;
begin
   if i < j then max := j else max := i;
end;

var o : array [1..1000000] of integer;
    u : array [1..400] of boolean;
```

o — массив, с помощью которого реализована очередь в обходе графа; u показывает, находится ли данная вершина в очереди.

```
procedure bfs(v : integer);
var p1, p2, cv : integer;
begin
   fillchar(u, sizeof(u), 0);
   p1 := 1;
   p2 := 1;
   o[p1] := v;
   u[v] := true;
   while p1 <= p2 do begin
      cv := o[p1]; inc(p1);
      u[cv] := false;
      for i := 1 to l[cv][0] do begin
         if m[v][l[cv][i]][0] > m[v][cv][1]+2 then begin
            m[v][l[cv][i]][0] := m[v][cv][1]+2;
            if not u[l[cv][i]] then begin
               u[l[cv][i]] := true;
               inc(p2);
               o[p2] := l[cv][i];
            end;
         end;
         if m[v][l[cv][i]][1] > m[v][cv][0]+2 then begin
            m[v][l[cv][i]][1] := m[v][cv][0]+2;
            if not u[l[cv][i]] then begin
               u[l[cv][i]] := true;
               inc(p2);
               o[p2] := l[cv][i];
            end;
         end;
      end;
   end;
end;
```

Собственно, модифицированный обход графа в ширину использует очередь, реализованную при помощи массива o. Если путь данной четности до вершины не существует, значение соответствующей ячейки будет равно

бесконечности (число, большее 1000000). За одну итерацию цикла из очереди берется следующая вершина, обновляются расстояния для смежных вершин. Те, для которых расстояния обновились и которые не находятся в очереди, туда заносятся.

```
var an, lc, ln, rc, rn, c : integer;
    s : array [1..400, 1..400] of boolean;
```

s — матрица смежности. Используется для удобства построения списка смежных вершин.

```
begin
   assign(input, 'f.in');
   reset(input);
   assign(output, 'f.out');
   rewrite(output);
```

Считываем входные данные.

```
   read(n);
   fillchar(m, sizeof(m), 57);

   read(k);
   for i := 1 to n do m[i][i][0] := 0;
   fillchar(s, sizeof(s), 0);
   for i := 1 to k do begin
      read(a[i], b[i]);
      s[a[i]][b[i]] := true;
      s[b[i]][a[i]] := true;
   end;
```

Сразу построим список смежных вершин для обхода в ширину.

```
   for i := 1 to n do begin
      l[i][0] := 0;
      for j := 1 to n do if s[i][j] then begin
          inc(l[i][0]); l[i][l[i][0]] := j;
      end;
   end;
```

Узнаем, где стоят роботы.

```
   read(nr);
   for i := 1 to nr do read(r[i]);
```

Запускаем обход.

```
   for i := 1 to nr do bfs(r[i]);
```


Обошли граф всеми роботами. Теперь перебираем ребра и вершины для подсчета ответа.

```
   an := 1000000;
   for i := 1 to k do begin
      c:= 0;
      for j := 1 to nr do
         c := max(c, min(min(m[r[j]][a[i]][0], m[r[j]][a[i]][1]),
                         min(m[r[j]][b[i]][0], m[r[j]][b[i]][1]))+1);
      an := min(an, c);
   end;
   for i := 1 to n do begin
      lc := 0; ln := 0;
      for j := 1 to nr do begin
         lc := max(lc, m[r[j]][i][0]);
         ln := max(ln, m[r[j]][i][1]);
      end;
      an := min(an, min(lc, ln));
   end;
```

Выводим ответ.

```
   if an < 100000 then writeln((an/2):0:1) else writeln(-1);

   close(input);
   close(output);
end.
```

Автор задачи — Е. В. Андреева,
автор разбора — А. В. Фонарев, участник олимпиады:
его команда была единственной, решившей на олимпиаде данную задачу.

Задача IX-G. Монетки

Тема: перебор.

Рассмотрим некоторый допустимый набор монет. Пусть $c[i]$ ($i = 1, \ldots, M$) — количество монет достоинством $A[i]$ в этом наборе. Тогда $c[i] = 0$, 1 или 2. Таким образом, каждый набор монет можно задать последовательностью из M чисел — нулей, единиц и двоек и, обратно, каждая такая последовательность задает некоторый набор монет. Количество всевозможных наборов равно, тем самым, $3^M \leqslant 3^{15} = 14\,348\,907$. Таким образом, за отведенное время можно перебрать все наборы и выбрать среди них подходящий.

Опишем процедуру перебора всех последовательностей длины M из цифр 0, 1, 2. Для решения подобных задач требуется некоторым образом упорядочить перебираемые объекты и описать алгоритм перехода от

предыдущего объекта к следующему. Наши последовательности можно считать M-значными числами и упорядочить их по возрастанию: $000\ldots00$, $000\ldots01$, $000\ldots02$, $0000\ldots10$, …, $222\ldots21$, $222\ldots22$. Как по данному числу $c[1]c[2]c[3]\ldots c[M]$ получить следующее? Если последняя цифра этого числа не двойка, то достаточно просто увеличить ее на 1. Если она равна двойке, то заменим ее нулем и посмотрим на предпоследнюю цифру. Если она отлична от 2, то увеличим ее на единицу, в противном случае и ее заменим нулем и перейдем к следующей цифре, и т. д. Например, пусть у нас уже есть число 1022011222. Просматриваем его цифры справа налево, заменяя двойки нулями, пока не встретим первую не двойку: 1022011**000**. Эту цифру увеличиваем на единицу: 1022012000. Получили следующее число. Поскольку мы знаем, сколько всего чисел, мы можем при переборе использовать цикл `for`.

Для каждого набора требуется вычислить сумму $c[1]\cdot A[1]+c[2]\times\times A[2]+\ldots+c[M]\cdot A[M]$ и из всех таких сумм выбрать наименьшую, а если таких сумм несколько, то выбрать из них ту, для которой количество монет $c[1]+c[2]+\ldots+c[M]$ наименьшее.

Автор разбора — В. М. Гуровиц.

Задача IX-H. Сверим часы

Тема: перебор.

В сутках всего 86400 секунд, т. е. в задаче возможны 86400 вариантов ответа. За отведенное время можно перебрать все и найти среди них тот, для которого общее количество нажатий наименьшее.

Для реализации этого алгоритма напишем функцию, которая определяет, сколько раз нужно нажать кнопки на часах, чтобы превратить показания времени $h1:m1:s1$ в показания времени $h2:m2:s2$. Если $h1=h2$, $m1=m2$ и $s1=s2$, то нажимать на кнопки нам не потребуется (функция возвращает ноль). Во всех остальных случаях нам придется 4 раза нажать на первую кнопку (перейти в режим редактирования часов, затем — минут, затем — секунд и вернуться в режим отображения времени). Подсчитаем, сколько раз придется нажимать на вторую кнопку. Для того чтобы установить нужное время в разряде секунд, понадобится $s1-s$ нажатий, если $s1\geqslant s$, или $s1-s+60$ нажатий, если $s1<s$. В последнем случае автоматически увеличивается количество минут `Inc(m)`. Аналогично разбираем разряд минут ($m1-m$ нажатий, если $m1\geqslant m$, и $m1-m+60$ в противном случае, при этом количество часов увеличивается на 1: `Inc(h)`) и разряд часов ($h1-h$ нажатий, если $h1\geqslant h$, и $h1-h+60$ в противном случае).

Теперь осталось считать все входные данные из файла и для каждого возможного времени на дисплее часов посчитать количество операций, необходимых для того чтобы установить это время на всех часах. Среди всех полученных значений выберем наименьшее.

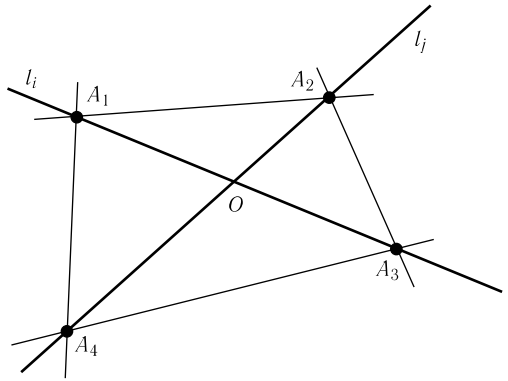
Автор задачи — А. П. Лахно, автор разбора — В. М. Гуровиц.

Задача IX-I. Разрезанный прямоугольник

Тема: геометрия.

К множеству прямых, пересекающих прямоугольник, добавим те, которые содержат его стороны.

После этого для каждой прямой l_i из этого множества заведем массив, в который запишем все точки ее пересечения с остальными прямыми. Отсортируем точки в этом массиве по абсциссе, а при равных абсциссах — по ординате; одинаковые точки удалим. Теперь для каждой точки пересечения этой прямой за время $O(\log N)$ можно найти две другие точки, являющиеся для нее соседними в отсортированном массиве.



Переберем все пары пересекающихся прямых l_i и l_j. Пусть две такие прямые пересекаются в точке O; A_1 и A_3 — точки, соседние с O на прямой l_i, A_2 и A_4 — на прямой l_j. Если некоторый треугольник образован прямыми l_i, l_j и некоторой прямой l_k, то он совпадает с одним из следующих треугольников: OA_1A_2, OA_2A_3, OA_3A_4, OA_4A_1.

Значит, достаточно проверить, существует ли для каждой из пар точек (A_1, A_2), (A_2, A_3), (A_3, A_4), (A_4, A_1) такая прямая l_k, что данные точки являются соседними в соответствующем ей массиве (если эти точки есть в массиве, но соседними не являются, то найдется прямая l_m, проходящая через точку O и разрезающая треугольник, образованный прямыми l_i, l_j и l_k, на две части).

Таким образом, мы найдем все треугольники, которые данные прямые вырезают на плоскости. Их количество будет иметь порядок N^2 (каждая

пара прямых порождает не более 4 треугольников). Выберем из них только те, которые лежат внутри нашего прямоугольника.

Подсчитаем время работы алгоритма:

$$O(N^2)[\text{перебор пар прямых } (l_i, l_j)]\times$$
$$\times (O(\log N)[\text{поиск } A_m] + O(N)[\text{поиск } l_k] + O(\log N)[\text{проверка того, что соответствующие две точки — соседние на } l_k]) = O(N^3).$$

Автор задачи — К. А. Батузов,
авторы разбора — М. О. Трухина, К. А. Батузов.

Задача IX-J. Количество треугольников

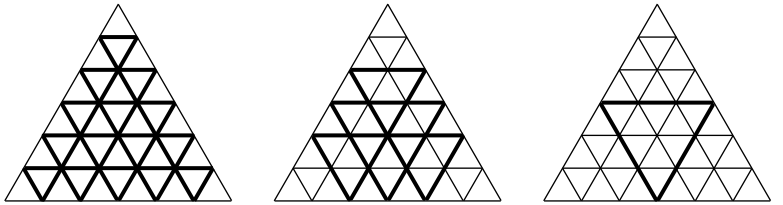
Тема: динамическое программирование.

Заметим, что все возможные треугольники — равносторонние, и одна из их сторон — горизонтальная.

Такие треугольники могут быть двух типов: горизонтальная сторона снизу (треугольник ориентирован как вся фигура) или горизонтальная сторона сверху (треугольник ориентирован как выделенный треугольник на рисунке из условия).

Подсчитаем сначала количество треугольников первого типа. Заметим, что каждый такой треугольник однозначно задается своим основанием, а значит, парой его концов — узлов сетки, лежащих на одной горизонтали. Если в некоторой горизонтали k узлов, то пару точек можно выбрать $k(k-1)/2$ способами, т. е. существуют $k(k-1)/2$ треугольников с таким основанием. Если фигура состоит из n уровней, то в верхней горизонтали имеются 2 узла, в следующей — 3, …, в последней — $(n+1)$. Таким образом, количество треугольников первого типа равно $\frac{1}{2}(1\cdot 2 + 2\cdot 3 + \ldots + n(n+1))$. Для предложенных в задаче ограничений на входные данные можно вычислить эту сумму используя цикл. Но можно вычислить эту сумму в явном виде, не используя компьютер. Как это сделать, мы покажем ниже.

Чуть сложнее подсчитать количество треугольников второго типа. Покажем один из возможных способов это сделать. Для удобства обозначим $S_n = 1 + 2 + 3 + \ldots + n = n(n+1)/2$. Подсчитаем отдельно количество треугольников со стороной 1, со стороной 2, со стороной 3 и т. д. Из рисунков на следующей странице видно, что эти значения равны S_{n-1}, S_{n-3}, S_{n-5}, … соответственно (слагаемые каждой из сумм S_i — это количество треугольников соответствующего размера в одном горизонтальном ряду). Таким образом, общее количество треугольников второго типа равно: $S_{n-1} + S_{n-3} + S_{n-5} + \ldots = (n(n-1) + (n-2)(n-3) + (n-4)(n-5) + \ldots)/2$ (последнее слагаемое в этой сумме равно S_2 или S_1, в зависимости от четности n). Эту сумму также можно сосчитать используя цикл, а можно упростить, записав ответ в явном виде.



Теперь покажем, как можно вычислить требуемые суммы. Начнем с первой: $1\cdot 2 + 2\cdot 3 + \ldots + n(n+1) = (1\cdot 1 + 1) + (2\cdot 2 + 2) + \ldots + (n\cdot n + n) = (1^2 + 2^2 + \ldots + n^2) + (1 + 2 + \ldots + n) = \frac{n(n+1)(2n+1)}{6} + \frac{n(n+1)}{2} = \frac{n(n+1)(2n+4)}{6} = \frac{n(n+1)(n+2)}{3}$. Использованную формулу суммы квадратов можно доказать, например, по индукции (впрочем, как и всю полученную формулу).

Теперь вычислим вторую сумму. Для этого введем следующие обозначения: $E_{2n} = 1\cdot 2 + 3\cdot 4 + 5\cdot 6 + \ldots + (2n-1)\cdot 2n$, $O_{2n+1} = 2\cdot 3 + 4\times 5 + 6\cdot 7 + \ldots + 2n\cdot(2n+1)$. Тогда

$$O_{2n+1} = 2\cdot 3 + 4\cdot 5 + 6\cdot 7 + \ldots + 2n\cdot(2n+1) = (1\cdot 2 + 2\cdot 2) + (3\cdot 4 + 2\cdot 4) + (5\cdot 6 + 2\cdot 6) + \ldots + ((2n-1)\cdot 2n + 2\cdot 2n) = (1\cdot 2 + 3\cdot 4 + 5\cdot 6 + \ldots + (2n-1)\cdot 2n) + (2\cdot 2 + 2\cdot 4 + 2\cdot 6 + \ldots + 2\cdot 2n) = E_{2n} + 4\cdot(1 + 2 + 3 + \ldots + n) = E_{2n} + 2n(n+1).$$

Теперь воспользуемся посчитанной ранее суммой:

$$\frac{2n(2n+1)(2n+2)}{3} = 1\cdot 2 + 2\cdot 3 + \ldots + 2n(2n+1) = E_{2n} + O_{2n+1} = E_{2n} + (E_{2n} + 2n(n+1)) = 2E_{2n} + 2n(n+1).$$

Отсюда находим:

$$E_{2n} = \frac{n(2n+1)(2n+2)}{3} - n(n+1),\ O_{2n+1} = \frac{n(2n+1)(2n+2)}{3} + n(n+1).$$

Общее количество треугольников второго типа на поле из n уровней равно $O_n/2$, если n нечетно, и $E_n/2$, если n четно.

А общее количество треугольников обоих типов равно $k(k+1)(4k+1)/2$, если $n = 2k$, и $(k+1)(4k^2+7k+2)/2$, если $n = 2k+1$.

Автор задачи — А. А. Петров, автор разбора — В. М. Гуровиц.

СТАТЬИ

Поиск в глубину и его применение

А. П. Лахно

Поиск в глубину (или обход в глубину) является одним из основных и наиболее часто употребляемых алгоритмов анализа графов.

Согласно этому алгоритму обход вершин графа осуществляется по следующему правилу. Начиная с некоторой вершины, мы идем по ребрам графа, пока не упремся в *тупик*. Вершина называется *тупиком*, если в ней нет исходящих ребер, ведущих в непосещенные вершины. После попадания в тупик мы возвращаемся назад вдоль пройденного пути, пока не обнаружим вершину, у которой есть исходящие ребра, ведущие в непосещенные вершины, и из нее идем по одному из таких ребер. Процесс кончается, когда мы возвращаемся в начальную вершину, а все соседние вершины уже оказались посещенными. Если после этого остаются непосещенные вершины, то повторяем поиск из одной из них в соответствии с вышеописанным алгоритмом. Так делаем до тех пор, пока не обнаружим все вершины графа.

Наиболее подходящим способом для реализации данного алгоритма является рекурсия. В этом случае все возвраты вдоль пройденного пути будут осуществляться автоматически, в результате работы механизма реализации рекурсии в языке программирования (заметим, что не все языки позволяют записывать в явном виде рекурсивные алгоритмы).

Исходный граф $G = (V, E)$, где V — множество вершин графа, а E — множество его ребер, может быть как ориентированным, так и неориентированным.

Переходя в вершину u из вершины v по ребру (v, u), мы запоминаем предшественника u при обходе в глубину: $p[u] = v$. Для вершин, у которых предшественников нет, положим $p[u] = -1$. Таким образом, получается *дерево поиска в глубину*. Если поиск повторяется из нескольких вершин, то получается несколько деревьев, образующих *лес поиска в глубину*. Лес поиска в глубину состоит из всех вершин исходного графа и ребер, по которым эти вершины впервые достигнуты.

Для наглядности будем считать, что в процессе работы алгоритма вершины графа могут быть белыми, серыми и черными. Изначально все вершины помечены белым цветом: `color`$[v] =$ `white`. Впервые обнаружив вершину v, мы красим ее серым: цветом `color`$[v] =$ `grey`. По окончании обработки всех исходящих ребер красим вершину v в черный цвет: `color`$[v] =$ `black`.

Таким образом, белый цвет соответствует тому, что вершина еще не обнаружена, серый — тому, что вершина уже обнаружена, но обработаны еще не все исходящие из нее ребра, черный — тому, что вершина уже обнаружена и все исходящие из нее ребра обработаны.

Помимо того, для каждой вершины в процессе поиска в глубину полезно запоминать еще два параметра: в $d[v]$ будем записывать «время» первого попадания в вершину, а в $f[v]$ — «время» окончания обработки всех исходящих из v ребер. При этом $d[v]$ и $f[v]$ представляют собой целые числа из диапазона от 1 до $2|V|$, где $|V|$ — число вершин графа.

Вершина v будет белой до момента $d[v]$, серой между $d[v]$ и $f[v]$, черной после $f[v]$.

Цвета вершин и пометки времени представляют собой удобный инструмент для анализа свойств графа и, как будет показано в дальнейшем, широко используются в различных алгоритмах на графах, в основе которых лежит поиск в глубину.

Ниже приводится схема возможной реализации поиска в глубину *Depth-first search* (*Dfs*):

```
1  Procedure Dfs(v);
2  begin
3    color[v] := grey;
4    time := time + 1; d[v] := time;
//цикл по всем ребрам, исходящим из v
5    for {u: (v, u) ∈ E} do
6      if color[u] = white then begin
7        p[u] := v;
8        Dfs(u)
9      end;
10   color[v] := black;
11   time := time + 1; f[v] := time
12 end;
// основная программа
13 for {v ∈ V} do begin
// инициализация значений
14   color[v] := white;
15   p[v] := -1;
16   d[v] := 0; f[v] := 0
17 end;
18 time := 0;
//цикл по всем вершинам
19 for {v ∈ V} do
20   if color[v] = white then Dfs(v);
```

Рассмотрим пошаговое исполнение алгоритма на примере конкретного ориентированного графа (жирным помечаются ребра, вошедшие в лес поиска в глубину):

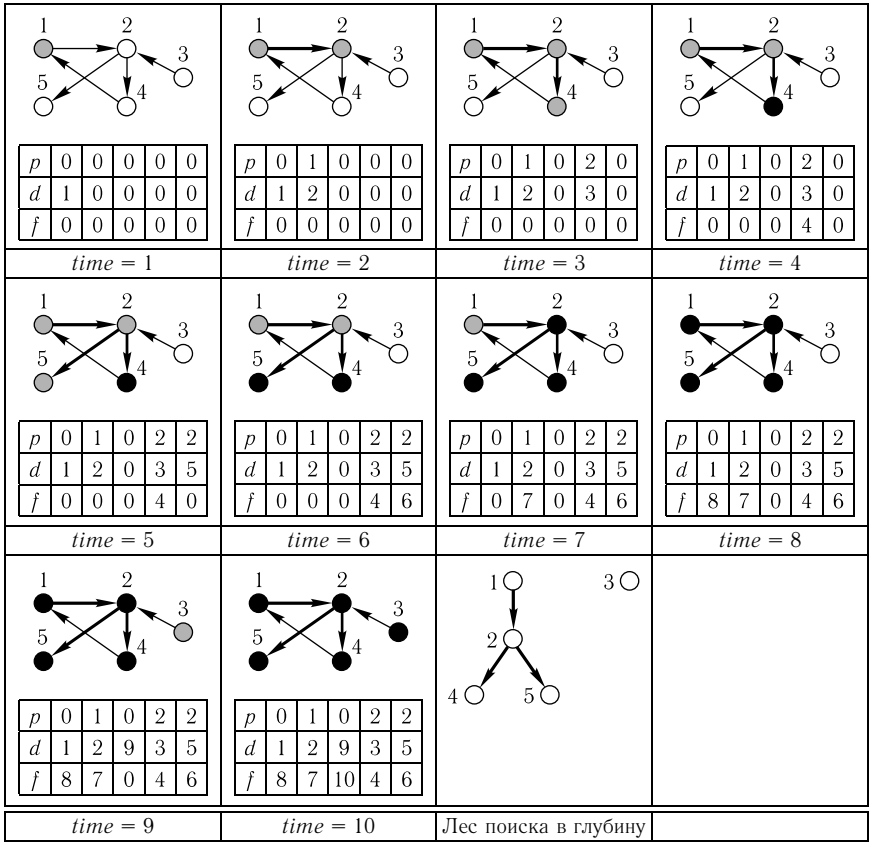


Рис. 1

Подсчитаем общее число операций при выполнении поиска в глубину на графе $G = (V, E)$. Изначальная инициализация данных (строки 13–18 алгоритма) и внешний цикл по вершинам (строки 19–20) требуют $O(V)$ времени.

Для каждой вершины процедура *Dfs* вызывается ровно один раз, причем время работы каждого вызова определяется временем, необходимым на просмотр всех исходящих из вершины ребер (5–9). Это время зависит от способа хранения графа.

Если хранить граф в виде матрицы смежности, то цикл в процедуре *Dfs* (строки 5–9) занимает $O(V)$ времени. Тогда суммарное время работы всех

вызовов *Dfs* равно $O(V^2)$. Таким образом, время работы поиска в глубину при использовании матрицы смежности равно $O(V^2)$.

Если же хранить граф списками смежности или списком ребер, то цикл (5–9) занимает O (число исходящих из вершины ребер) времени. Суммарное время работы всех вызовов *Dfs* равно $O(E)$, так как каждое ребро просматривается лишь однажды. Таким образом, время работы поиска в глубину при использовании списков смежности или списка ребер равно $O(V + E)$.

В графах, где число ребер E не велико (порядка числа вершин V), второй способ хранения, несмотря на несколько более сложную реализацию, дает ощутимый выигрыш во времени.

Свойства пометок времени

Очень красивое и важное свойство пометок времени состоит в том, что время обнаружения и время окончания обработки образуют правильную скобочную структуру. Действительно, будем обозначать обнаружение вершины открывающей скобкой с индексом номера вершины, а окончание обработки — закрывающей скобкой с индексом номера вершины. Тогда последовательность событий, выстроенная в порядке возрастания времени, будет правильно построенным скобочным выражением.

Так, например, для графа с рис. 1 мы получим следующее скобочное выражение:

time	1	2	3	4	5	6	7	8	9	10
	$({}^1$	$({}^2$	$({}^4$	${}^4)$	$({}^5$	${}^5)$	${}^2)$	${}^1)$	$({}^3$	${}^3)$

Поясним, из каких соображений следует это свойство. Для белой вершины v обозначим через $W(v)$ множество всех белых вершин, доступных из вершины v по путям, в которых все промежуточные вершины также являются белыми.

Вызов процедуры *Dfs* для белой вершины v делает серой эту вершину, затем полностью обрабатывает все вершины из $W(v)$, оставляя серые и черные вершины без изменений, после чего делает вершину v черной.

Таким образом, для любой вершины u из $W(v)$ верно: $d[v] < d[u] < f[u] < f[v]$, что соответствует выражению $({}^u\ldots({}^v\ldots{}^v)\ldots{}^u)$.

Для строгого доказательства утверждения о правильной скобочной структуре пометок времени можно рассуждать по индукции. При этом, доказывая требуемое свойство рекурсивной процедуры *Dfs*, предполагаем, что для всех внутренних рекурсивных вызовов это свойство уже выполнено.

При поиске в глубину как в ориентированном, так и в неориентированном графе для любых двух вершин u и v выполняется ровно одно из трех утверждений:

1) отрезки $[d[u], f[u]]$ и $[d[v], f[v]]$ не пересекаются;

2) отрезок $[d[u], f[u]]$ целиком содержится внутри отрезка $[d[v], f[v]]$, и u — потомок v в дереве поиска в глубину;
3) отрезок $[d[v], f[v]]$ целиком содержится внутри отрезка $[d[u], f[u]]$, и v — потомок u в дереве поиска в глубину.

Эти утверждения позволяют быстро определять взаимное расположение вершин в лесу поиска в глубину.

Классификация ребер

Ребра ориентированного графа делятся на несколько категорий в зависимости от их роли при поиске в глубину.

1. *Ребра деревьев* — это ребра, входящие в лес поиска в глубину. На рис. 2 это ребра (1, 2), (2, 3), (2, 5), (3, 4).
2. *Обратные ребра* — это ребра, соединяющие вершину с ее предком в дереве поиска в глубину (ребра-циклы, возможные в ориентированных графах, считаются обратными ребрами). На рис. 2 это ребра (5, 1), (6, 6).
3. *Прямые ребра* — это ребра, соединяющие вершину с ее потомком, но не входящие в лес поиска в глубину: (2, 4) на рис. 2.
4. *Перекрестные ребра* — все остальные ребра графа. Они могут соединять две вершины из одного дерева поиска в глубину, если ни одна их этих вершин не является предком другой, или же вершины из разных деревьев: (5, 4), (6, 1) на рис. 2.

Тип ребра (v, u) можно определить по цвету вершины u в момент, когда ребро исследуется в первый раз: белый цвет означает ребро дерева ((v, u) войдет в лес поиска в глубину), серый (u является предком v) — обратное ребро, черный (ни одна из них не является предком другой) — прямое или перекрестное ребро.

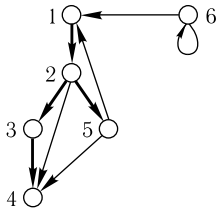


Рис. 2

Эта классификация оказывается полезной в различных задачах, решаемых с использованием поиска в глубину. Так например, ориентированный граф не имеет циклов тогда и только тогда, когда поиск в глубину не находит в нем обратных ребер.

В неориентированном графе одно и то же ребро можно рассматривать с разных концов, и в зависимости от этого оно может попасть в разные категории. Будем относить ребро к той категории, которая стоит раньше в классификации для ориентированных графов.

Например, для графа с рис. 3 ребро (1, 4) можно считать как прямым, так и обратным. В соответствии с принятым утверждением, мы отнесем его к категории обратных.

Оказывается, что при принятых соглашениях прямых и перекрестных ребер в неориентированных графах не будет. Действительно, пусть (v, u) — произвольное ребро неориентированного графа. Без ограничения общности можно считать, что при поиске в глубину $d[v] < d[u]$. Тогда вершина v должна быть обнаружена и обработана раньше, чем закончится обработка вершины u, так как v содержится и в списке вершин, смежных с u. Если ребро (v, u) в первый раз обрабатывается в направлении от v к u, то (v, u) становится ребром дерева. Если же оно впервые обрабатывается в направлении от u к v, то оно становится обратным ребром (в этот момент вершина v серая, так как она обнаружена, но ее обработка еще не завершена).

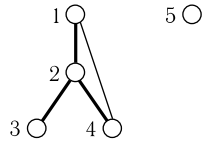


Рис. 3

Таким образом, для графа на рис. 3 ребра (1, 2), (2, 3) и (2, 4) будут ребрами дерева, а (1, 4) — обратным ребром.

Перейдем к рассмотрению стандартных задач, решаемых с помощью поиска в глубину. Напомним, что в предыдущем разделе мы уже фактически показали, как с помощью поиска в глубину проверить ацикличность ориентированного графа или, наоборот, убедиться в наличии циклов.

Компоненты связности

Неориентированный граф G называется *связным*, если любые две его вершины достижимы друг из друга по ребрам графа. Несвязный граф распадается на несколько связных частей, никакие две из которых не соединены ребрами. Эти части и называются *компонентами связности* графа.

Так например, граф на рис. 4 распадается на три компоненты связности: (1, 6), (2, 7, 5, 4), (3).

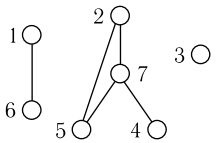


Рис. 4

Возникает задача разбиения неориентированного графа $G = (V, E)$ на компоненты связности.

Эта задача решается с помощью поиска в глубину следующим образом. Запускаем поиск в глубину из первой вершины. Все вершины, обнаруженные в ходе этого алгоритма, принадлежат одной компоненте связности. Если остались необнаруженные вершины, то запускаем поиск в глубину из любой из них. Вновь обнаруженные вершины принадлежат другой компоненте связности. Повторяем этот процесс до тех пор, пока не останется необнаруженных вершин, каждый раз относя вновь обнаруженные вершины к очередной компоненте связности.

Ниже приводится схема возможной реализации алгоритма разбиения графа на компоненты связности:

```
1  Procedure Dfs(v);
2  begin
```

```
3    comp[v] := num;
4    for {u:(v, u) ∈ E} do
5      if comp[u] = 0 then Dfs(u)
6  end;
//основная программа
7  for {v ∈ V} do comp[v] := 0;
8  num := 0;
9  for {v ∈ V} do
10  if comp[v] = 0 then begin
//найдена очередная компонента связности
11    inc(num);
12    Dfs(v)
13  end;
```

По окончании работы программы переменная `num` будет содержать количество компонент связности графа G, а в массиве `comp` будут храниться номера компонент, к которым принадлежат соответствующие вершины.

Граф связен в том и только том случае, когда все его вершины обнаружены после первого же запуска поиска в глубину (`num = 1`, т. е. граф состоит из одной компоненты связности).

Топологическая сортировка

Пусть имеется ориентированный граф G без циклов. Задача о топологической сортировке этого графа состоит в том, чтобы указать такой порядок вершин, при котором ребра графа ведут только из вершин с меньшим номером к вершинам с бóльшим номером. Если в графе есть циклы, то такого порядка не существует. Можно переформулировать задачу о топологической сортировке следующим образом: расположить вершины графа на горизонтальной прямой так, чтобы все ребра графа шли слева направо. В жизни это соответствует, например, следующим проблемам: в каком порядке следует располагать темы в школьном курсе математики, если известно для каждой темы, знания каких других тем необходимы для ее изучения; в каком порядке следует надевать на себя комплект одежды, начиная с нижнего белья и заканчивая верхней одеждой. Очевидно, что зачастую задача топологической сортировки имеет не единственное решение.

На рис. 5 представлен граф и один из вариантов его топологической сортировки: 1, 5, 2, 6, 3, 4. Заметим, что, например, вершину с номером 6 можно поставить в любое место топологической сортировки этого графа. Вершины 1 и 5 также могут располагаться в другом порядке (сначала 5, а затем 1). В остальном порядок топологической сортировки вершин данного графа фиксирован.

Алгоритм нахождения топологической сортировки также основан на поиске в глубину. Применим поиск в глубину к нашему графу G. Завершая

обработку каждой вершины (делая ее черной), заносим ее в начало списка. По окончании обработки всех вершин полученный список будет содержать

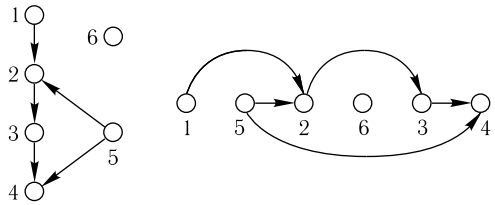


Рис. 5

топологическую сортировку данного графа. Обозначим количество вершин в графе n. Так как в результирующий список должны попасть все n вершин нашего графа, то реализовать его можно на обычном массиве, записывая в него элементы списка начиная с n-го места в массиве и заканчивая первым.

Покажем, что полученный список действительно удовлетворяет основному свойству топологической сортировки: все ребра ведут от вершин с меньшим номером к вершинам с бóльшим номером в нашем списке. Предположим противное: пусть существует ребро (v, u) такое, что вершина u встречается в нашем списке раньше вершины v. Но тогда обработка вершины v была завершена раньше, чем обработка вершины u. Это означает, что в момент окончания обработки вершины v вершина u могла быть либо белой, либо серой. В первом случае мы должны были бы пройти по ребру (v, u), но не сделали этого. Во втором случае поиск в глубину нашел бы обратное ребро, т. е. граф G содержал бы циклы. Оба случая приводят к противоречию, а значит, наше предположение неверно и указанный список действительно является топологической сортировкой.

Ниже приводится схема возможной реализации алгоритма построения топологической сортировки. Причем, если в графе есть циклы, что означает невозможность его топологической сортировки, это также будет обнаружено в процессе работы алгоритма.

```
1  Procedure Dfs(v);
2  begin
3    color[v] := grey;
4    for {u:(v, u) ∈ E} do begin
5      if color[u] = white then Dfs(v)
6      else if color[u] = grey then
7           {граф имеет циклы, конец}
8    end;
9    inc(num); list[n-num+1] := v;
10   color[v] := black
```

```
11 end;
//основная программа
12 for v := 1 to n do begin
13   color[v] := white; list[v] := 0
14 end;
15 num := 0;
16 for v := 1 to n do
17   if color[v] = white then Dfs(v);
//печатаем вариант топологической сортировки
18 for v := 1 to n печать(list[v]);
```

По окончании работы алгоритма массив `list` будет содержать топологическую сортировку данного графа. Условие в строке 6 осуществляет проверку на наличие циклов (ориентированный граф не имеет циклов тогда и только тогда, когда поиск в глубину не находит в нем обратных ребер). При нахождении обратного ребра (строка 7) следует выдать соответствующее сообщение и завершить исполнение алгоритма. Последнее для рекурсивного алгоритма не совсем тривиально.

Мосты

Мостом неориентированного графа G называется ребро, при удалении которого увеличивается количество компонент связности графа. Соответственно, для связного графа мостом называется ребро, при удалении которого граф перестает быть связным.

При удалении всех мостов граф распадается на компоненты связности, которые называются *компонентами реберной двусвязности*.

Между любыми двумя вершинами одной компоненты реберной двусвязности существуют, по крайней мере, два пути, не пересекающиеся по ребрам. Верно и обратное утверждение: любые две вершины, между которыми существуют два пути, не пересекающиеся по ребрам, принадлежат одной компоненте реберной двусвязности.

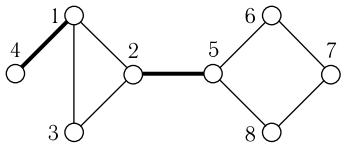


Рис. 6

Между двумя компонентами реберной двусвязности не может быть более одного ребра — в противном случае они образовывали бы одну компоненту реберной двусвязности. Если две различные компоненты реберной двусвязности соединены ребром, то это ребро — мост.

Так, например, для графа на рис. 6 мостами будут ребра: (1, 4) и (2, 5), а компонентами реберной двусвязности, соответственно, наборы вершин: (1, 2, 3), (4), (5, 6, 7, 8).

Рассмотрим задачу нахождения мостов для заданного неориентированного графа G.

Эта задача решается с помощью двух поисков в глубину.

При первом поиске, проходя по ребру (v, u) в направлении от вершины v к вершине u, «ориентируем» его против направления движения, т. е. запрещаем прохождение по ребру (v, u) в направлении от v к u. При обнаружении новой вершины заносим ее в конец списка `list`.

Применим описанный алгоритм к графу на рис. 6. Начиная из вершины 1, проходим по ребрам (1, 2) и (2, 3), ориентируя их против направления движения. Из вершины 3 не выходят ребра, ведущие в непосещенные вершины. Возвращаемся в вершину 2, проходим по ребрам (2, 5), (5, 6), (6, 7) и (7, 8). Возвращаемся в вершину 1, проходим по ребру (1, 4). Возвращаемся в вершину 1 и завершаем поиск в глубину.

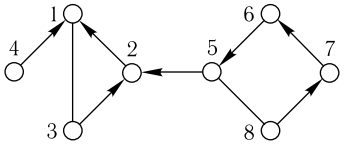


Рис. 7

На рис. 7 представлено как будет выглядеть граф с рис. 6 по окончании первого поиска в глубину. При этом по «ориентированным» ребрам можно ходить только в направлении, указанном стрелками, а по «неориентированным» ребрам — в обоих направлениях. Список `list` выглядит следующим образом: 1, 2, 3, 5, 6, 7, 8, 4.

Второй поиск в глубину осуществляется с учетом «ориентированных» ребер. Внешний цикл по вершинам должен идти в том порядке, в каком они записаны в списке `list`. Получающиеся деревья поиска красим каждое в свой цвет. Эти деревья являются компонентами реберной двусвязности нашего графа. Для графа на рис. 7 получим следующие три дерева поиска: (1, 3, 2), (5, 8, 7, 6) и (4).

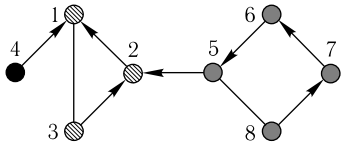


Рис. 8

Теперь остается только выбрать ребра исходного графа G, соединяющие вершины разного цвета (рис. 8): (1, 4) и (2, 5) — именно они и будут мостами.

Поясним, почему деревья поиска в глубину, получающиеся во время второго поиска, являются компонентами реберной двусвязности нашего графа.

Пусть очередное дерево поиска получилось в результате запуска из некоторой вершины x компоненты реберной двусвязности A. Докажем, что полученное дерево совпадает с A, в предположении, что все деревья поиска, полученные ранее, действительно образуют компоненты реберной двусвязности.

Во-первых, покажем, что дерево поиска в глубину не может содержать вершин из какой-то другой компоненты реберной двусвязности B (рис. 9).

Если между компонентами A и B нет ребер, то утверждение очевидно. Пусть между A и B есть некоторое ребро (v, u) (более одного ребра меж-

ду A и B быть не может, поскольку в противном случае они образовывали бы одну компоненту реберной двусвязности).

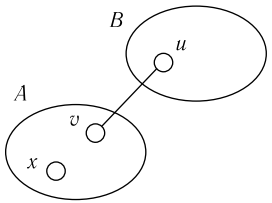


Рис. 9

Если вершина u уже обработана, то пройти по ребру (v, u), а значит, и попасть в компоненту B нельзя.

Если же вершина u еще не обработана, то она стоит в списке `list` заведомо позже вершины v. Это значит, что при первом поиске в глубину мы попали в v, когда u была еще белой. Не пройдя по ребру (v, u), из вершины v в вершину u попасть невозможно (иначе A и B образовывали бы одну компоненту реберной двусвязности). Поэтому при первом поиске в глубину «сориентируем» ребро (v, u) в направлении от u к v, а значит, при втором поиске в глубину попасть из A в B будет уже невозможно.

Покажем теперь, почему дерево поиска, начатого из некоторой вершины x компоненты реберной двусвязности A, будет содержать все вершины из A (рис. 10). Будем рассуждать от противного. Предположим, что некоторая вершина y, лежащая в этой же компоненте, не войдет в дерево поиска. Поскольку по предположению все деревья поиска, полученные ранее, действительно образуют компоненты реберной двусвязности, то y может не попасть в дерево второго поиска в глубину только в том случае, когда она недостижима из x в графе с учетом ориентации ребер. Обозначим через X множество вершин компоненты A, достижимых из вершины x при втором поиске в глубину, а через Y — множество вершин компоненты A, из которых при втором поиске в глубину достижима вершина y. Поскольку X и Y лежат внутри одной компоненты реберной двусвязности, то между ними есть, по крайней мере, два ребра. А так как вершина y недостижима из вершины x, то все ребра между X и Y ориентированы от Y к X. Все ориентированные ребра принадлежат деревьям первого поиска в глубину, причем ребра между X и Y принадлежат одному дереву (X и Y — подмножества компоненты реберной двусвязности A, а компонента реберной двусвязности всегда связна). Но тогда получается, что при построении одного дерева поиска в глубину мы из X в Y попадали минимум два раза, а из Y в X ни одного, чего быть не может — противоречие. Следовательно, все вершины A достижимы из x, а значит, войдут в дерево поиска.

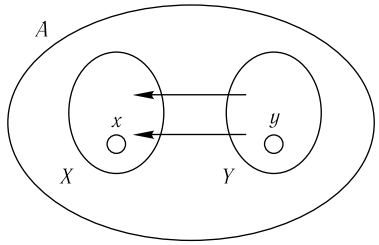


Рис. 10

Таким образом, деревья поиска в глубину, получающиеся при втором поиске в глубину, являются компонентами реберной двусвязности исходного графа G.

Ниже приводится схема возможной реализации алгоритма поиска мостов в неориентированном графе $G = (V, E)$:

```
//первый поиск в глубину
1   Procedure Dfs1(v);
2   begin
3     color[x] := 1;
4     inc(num); list[num] := v;
5     for {u:(v, u)   E} do
6       if color[u] = 0 then begin
7         Dfs1(u);
8       {запретить прохождение по ребру (v, u) в направлении от v к u}
9       end;
10   end;
//второй поиск в глубину
11  Procedure Dfs2(v);
12  begin
13    color[x] := num;
14    for {u:(v, u)   E} do
15      if (color[u] = 0) and {можно идти от v к u} then Dfs2(u);
16  end;
//основная программа
//вызов первого поиска
17  for v := 1 to n do begin
18    color[v] := 0; list[v] := 0;
19  end;
20  num:=0;
21  for v:=1 to n do
22    if color[v] = 0 then Dfs1(v);
//вызов второго поиска
23  for v := 1 to n do color[v] := 0;
24  num:=0;
25  for v:=1 to n do
26    if color[list[v]] = 0 then begin
27      inc(num);
28      Dfs2(list[v]);
29    end;
//печатаем ребра, являющиеся мостами
30  for {v, u:(v, u)   E} do
31    if (color[v]<>color[u]) then печать((v, u))
```

Для того чтобы запрещать прохождение по ребру в заданном направлении (строка 8), а также проверять возможность прохождения (строка 15),

полезно вместе с каждым ребром хранить два флажка, отвечающие за возможность прохождения по ребру в каждом из направлений.

По окончании работы программы переменная `num` будет содержать количество компонент реберной двусвязности заданного графа, а в массиве `color` будут храниться номера компонент реберной двусвязности, к которым принадлежат соответствующие вершины.

Точки сочленения

Точкой сочленения неориентированного графа G называется вершина, при удалении которой вместе со всеми смежными ребрами увеличивается количество компонент связности графа. Соответственно, для связного графа точкой сочленения называется вершина, при удалении которой граф перестает быть связным.

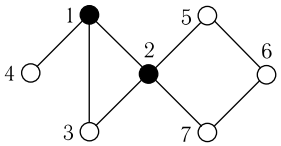


Рис. 11

Так, например, для графа на рис. 11 точками сочленения будут вершины 1 и 2.

Рассмотрим задачу нахождения точек сочленения для заданного неориентированного графа G.

Эта задача решается с помощью поиска в глубину следующим образом.

Применив к графу, изображенному на рис. 11, алгоритм поиска в глубину, получим некоторый лес поиска в глубину (рис. 12).

Корень дерева (начальная вершина) поиска в глубину является точкой сочленения тогда и только тогда, когда у него более одного сына в дереве поиска в глубину. Сыновьями вершины v являются вершины, впервые обнаруженные из v при поиске в глубину. Для графа, изображенного на рис. 12, вершина 1 имеет двух сыновей: 4 и 2, и поэтому она является точкой сочленения.

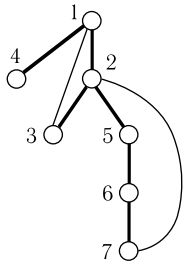


Рис. 12

Действительно, если корень дерева имеет только одного сына или же вообще не имеет сыновей, то он, очевидно, не является точкой сочленения. Если же у корня более одного сына, то к нему «подвешены» несколько поддеревьев, между которыми нет ребер (при поиске в глубину в неориентированном графе перекрестных ребер быть не может). Таким образом, при удалении корня количество компонент связности увеличится, а значит, он является точкой сочленения.

Так для графа, изображенного на рис. 12, к вершине 1 «подвешены» два поддерева. Первое состоит из одной вершины 4, второе — из вершин 2, 3, 5, 6, 7. При удалении вершины 1 граф распадается на две компоненты связности: (4) и (2, 3, 5, 6, 7).

Для того чтобы понять, является ли промежуточная вершина v дерева поиска в глубину точкой сочленения, надо узнать, существует ли поддерево с корнем в одном из сыновей u вершины v, которое «отсоединится» от дерева поиска при удалении самой вершины v. Поддерево может «отсоединиться» от дерева поиска в том и только том случае, когда из вершин этого поддерева нет обратных ребер, ведущих в вершины, обнаруженные до вершины v.

Пусть $up[u]$ — минимальное время обнаружения среди всех вершин тех, которые могут быть достигнуты из поддерева с корнем в вершине u при прохождении ровно по одному обратному ребру.

Промежуточная вершина дерева поиска в глубину v является точкой сочленения тогда и только тогда, когда у нее существует сын u такой, что $up[u] \geqslant d[v]$, где $d[v]$ — время обнаружения вершины v. В соответствии с этим признаком, вершина 2 графа (рис. 12) является точкой сочленения, поскольку у вершины 2 есть сын 5 такой, что $up[5] = d[2]$ (в поддереве с корнем в вершине 5 есть только одно обратное ребро (7, 2)).

Действительно, если существует такой сын u вершины v, что $up[u] \geqslant d[v]$, то при удалении вершины v поддерево с корнем u отделится от дерева, образовав новую компоненту связности, а значит, v — точка сочленения. Если же такого сына не существует, то при удалении v количество компонент связности не увеличится, так как все поддеревья с корнями в сыновьях вершины v соединены с «верхней» частью исходного дерева обратными ребрами.

Таким образом, для нахождения точек сочленения в процессе поиска в глубину необходимо отслеживать количество сыновей $ch[v]$ у корней деревьев поиска в глубину, а также вычислять величину $up[v]$ для всех вершин графа.

Величина $up[v]$ вычисляется как минимум из величин $up[u]$ для всех вершин u, являющихся сыновьями v в дереве поиска в глубину, и времен обнаружения $d[w]$ всех вершин w, достижимых непосредственно из v по обратному ребру. Изначально $up[v]$ присваивается значение $n + 1$, заведомо большее времени обнаружения любой вершины.

Ниже приводится схема возможной реализации алгоритма поиска точек сочленения в неориентированном графе $G = (V, E)$:

```
1   Procedure Dfs(v);
2   begin
3     color[v] := 1; inc(time);
4     d[v] := time; up[v] := n + 1;
5     for {u:(v, u)   E} do
6       if color[u] = 0 then begin
7         inc(ch[v]); Dfs(u);
          //u является сыном v в дереве поиска
```

```
8         up[v] := min(up[v], up[u]);
          //проверка внутренних вершин
9         if up[u] >= d[v] then
10          r[v] := true
11      end else
          //u достижима из v по обратному ребру
12        up[v] := min(up[v], d[u]);
13  end;
//основная программа
14  for v := 1 to n do begin
      //инициализация данных
15    color[v] := 0; d[v] := 0;
16    up[v] := 0; ch[v] := 0;
17    r[v] := false;
18  end;
19  time:=0;
20  for v := 1 to n do
21    if color[v] = 0 then begin
22      Dfs(v);
        //проверка количества детей у корней деревьев
23      if ch[v] > 1 then r[v] := true
24                   else r[v] := false;
25    end;
```

По окончании работы программы массив r содержит информацию о том, какие вершины являются точками сочленения: $r[v] =$ `true` тогда и только тогда, когда v — точка сочленения данного графа.

Поиск мостов и точек сочленения представляет интерес, например, для анализа надежности компьютерных сетей.

Задача о поиске мостов соответствует вопросу нахождения соединительных линий, при поломке одной из которых сеть перестает быть связной, а задача о поиске точек сочленения позволяет решить вопрос нахождения компьютеров, при поломке одного из которых сеть перестает быть связной.

Компоненты сильной связности

Компонентой сильной связности ориентированного графа называется максимальное множество вершин, в котором существуют пути из любой вершины в любую другую. Так например, граф на рис. 13 состоит из четырех компонент сильной связности: (1, 2, 4), (3, 5), (6) и (7).

Каждая вершина ориентированного графа G принадлежит какой-либо компоненте сильной связности, но некоторые ребра могут не принадлежать никакой компоненте сильной связности. Такие ребра соединяют вершины из разных компонент сильной связности.

Связи между компонентами сильной связности можно представить путем создания *конденсации* графа G. Конденсацией графа G называется граф, построенный следующим образом. Наборы вершин, образующие одну компоненту сильной связности C исходного графа, сливаются в одну вершину конденсации C^*. Из вершины C^* в вершину D^* конденсации есть ребро тогда и только тогда, когда в графе G есть ребро, ведущее из некоторой вершины x компоненты сильной связности C в некоторую вершину y компоненты сильной связности D. Конденсация графа, изображенного на рис. 13, представлена на рис. 14. Вершины 1, 2, 4 исходного графа, образующие одну компоненту сильной связности, сливаются в одну вершину, вершины 3, 5 — во вторую, вершина 6 — в третью, а вершина 7 — в четвертую. Ребра (4, 3), (2, 3) и (6, 7) отражены в конденсации, поскольку соединяют вершины разных компонент сильной связности.

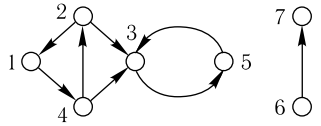


Рис. 13

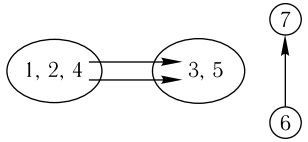


Рис. 14

В конденсации не может быть циклов, поскольку если бы существовал цикл, то все компоненты сильной связности, входящие в этот цикл, образовывали бы одну компоненту сильной связности.

Рассмотрим задачу нахождения компонент сильной связности ориентированного графа $G = (V, E)$.

Эта задача решается с помощью двух поисков в глубину по следующему алгоритму.

1. Сначала выполняется поиск в глубину на исходном графе G. По окончании обработки очередной вершины заносим ее в начало списка `list`.
2. Строим транспонированный граф $G^T = (V, E^T)$, полученный из исходного графа G заменой направления всех ребер на противоположное.
3. Выполняется поиск в глубину на графе G^T. При этом во внешнем цикле вершины перебираются в том порядке, в каком они записаны в список `list`, т. е. в порядке убывания времени выхода. Каждое дерево поиска в глубину красится в свой цвет.

Деревья поиска в глубину, получающиеся на 3-м шаге, и будут компонентами сильной связности исходного графа G. Докажем это утверждение.

Сначала покажем, что если две вершины v и u принадлежат одной компоненте сильной связности, то они войдут в одно дерево второго поиска

в глубину. Поскольку вершины v и u принадлежат одной компоненте сильной связности, то в исходном графе G существует путь как из v в u, так и из u в v. Поскольку граф G^T получен из графа G транспонированием всех его ребер, то в нем также существует путь как из v в u, так и из u в v, а следовательно, v и u попадут в одно дерево второго поиска в глубину.

Теперь покажем, что если две вершины v и u принадлежат одному и тому же дереву второго поиска в глубину, то они лежат в одной компоненте сильной связности графа G. Пусть x — корень дерева поиска, которому принадлежат вершины v и u. Поскольку v является потомком x, то в графе G^T есть путь от вершины x к вершине v, а в графе G, соответственно, есть путь от вершины v к вершине x.

При поиске в глубину на графе G^T вершина v обнаружена позже, чем вершина x, т. е. x имеет меньший номер в списке `list`, а значит, обработка вершины v заканчивается раньше, чем обработка вершины x при первом поиске в глубину.

Предположим, что при первом поиске в глубину в графе G вершина v обнаружена раньше вершины x. Но тогда из-за наличия пути от вершины v к вершине x мы обнаружим и завершим обработку вершины x до окончания обработки вершины v, что противоречит их взаимному расположению в списке `list`.

Значит, предположение неверно и обнаружение x происходит при первом поиске в глубину раньше, чем обнаружение v. Таким образом, $d[x] < d[v] < f[v] < f[x]$, т. е. v является потомком x в дереве первого поиска в глубину, а значит, в графе G существует путь из x в v. А так как существует и путь из v в x, то x и v принадлежат одной компоненте сильной связности.

Аналогично доказывается, что x и u принадлежат одной компоненте сильной связности. Поэтому v и u тоже принадлежат одной компоненте сильной связности, так как существует путь из вершины v в вершину u через вершину x и путь из вершины u в вершину v через вершину x.

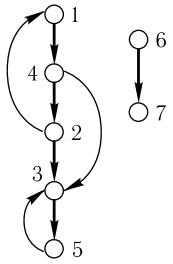


Рис. 15

Таким образом показано, что деревья поиска в глубину, получающиеся на 3-м шаге, являются компонентами сильной связности исходного графа G.

Рассмотрим работу приведенного алгоритма на примере графа, изображенного на рис. 13. Сначала выполняется первый поиск в глубину (рис. 15) на исходном графе G: начиная из вершины 1 проходим по ребрам (1, 4), (4, 2), (2, 3), (2, 5). Дальше идти некуда, возвращаемся в вершину 1, последовательно занося вершины, из которых уходим, в начало списка `list`. Начиная из вершины 6 идем по ребру (6, 7), после чего возвращаемся в вершину 6 и завершаем первый поиск

в глубину. После его окончания список `list` выглядит следующим образом: 6, 7, 1, 4, 2, 3, 5.

Строим по графу G (рис. 13) транспонированный граф G^T (рис. 16).

Запускаем второй поиск в глубину, перебирая во внешнем цикле вершины в соответствии со списком `list`. При этом вершина 6 составляет первое дерево поиска, вершина 7 — второе, вершины 1, 2, 4 — третье, а вершины 3 и 5 — четвертое. Красим вершины в соответствии с номером их дерева поиска (рис. 16). Разбиение вершин по цветам соответствует разбиению исходного графа G на компоненты сильной связности.

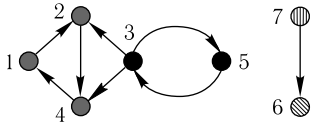


Рис. 16

Ниже приводится схема возможной реализации алгоритма нахождения компонент сильной связности для заданного ориентированного графа G:

```
//первый поиск в глубину
1   Procedure Dfs1(v);
2   begin
3     color[v] := 1;
4     for {u:(v, u) ∈ E} do
5         if color[u] = 0 then Dfs1(u);
6     dec(num); list[num] := v;
7   end;
//второй поиск в глубину
8   Procedure Dfs2(v);
9   begin
10    color[v] := num;
11    for {u:(v, u) ∈ E^T} do
12        if color[u] = 0 then Dfs2(u);
13  end;
//основная программа
//вызов первого поиска в глубину
14  for v := 1 to n do begin
15    color[v] := 0; list[v] := 0;
16  end;
17  num:=n;
18  for v := 1 to n do
19    if color[v] = 0 then Dfs1(v);
//построение транспонированного графа G^T=(V, E^T)
20  E^T:= ∅;
21  for {u:(v, u) ∈ E} do E^T := E^T ∪ (u, v);
//вызов второго поиска в глубину
22  for v := 1 to n do begin
23    color[v] := 0; list[v] := 0;
24  end;
```

```
25  num:=0;
26  for v := 1 to n do
27    if color[list[v]] = 0 then begin
28      inc(num);
29      Dfs2(list[v]);
30    end;
```

По окончании работы программы переменная `num` будет содержать количество компонент сильной связности графа G, а в массиве `color` будут храниться номера компонент сильной связности, к которым принадлежат соответствующие вершины.

Построение транспонированного графа G^T занимает $O(V+E)$ времени при хранении графа списками смежности или списком ребер и $O(V^2)$ — при хранении графа матрицей смежности.

В заключение отметим, что все предложенные в статье алгоритмы, основанные на поиске в глубину, имеют оценку сложности $O(V+E)$ при хранении графа списками смежности или списком ребер и $O(V^2)$, соответственно, — при хранении графа матрицей смежности. Оценка времени работы в большинстве из предложенных алгоритмов определяется именно временем работы поиска в глубину.

Литература

1. *Кормен Т.*, *Лейзерсон Ч.*, *Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 2000.
2. *Ахо А.В.*, *Хопкрофт Д.*, *Ульман Дж.Д.* Структуры данных и алгоритмы. М.: Издательский дом «Вильямс», 2003.
3. *Макконнелл Дж.* Анализ алгоритмов. Вводный курс. М.: Техносфера, 2002.
4. *Окулов С.М.* Программирование в алгоритмах. М.: БИНОМ. Лаборатория знаний, 2004.
5. *Оре О.* Теория графов. М.: Наука, 1968.

Подсчет значения арифметического выражения методом рекурсивного спуска

В. А. Матюхин

На практике часто встречается следующая задача: пользователь вводит арифметическое выражение, например: `(1+2*3)*(4+5)+6*(7+8)+9`, нужно вычислить его значение. Если вы когда-нибудь пробовали писать программу, решающую такую задачу, то, наверное, понимаете, что это не так просто, как кажется с первого взгляда. Если никогда не пробовали, то для того чтобы в полной мере оценить изящность метода, который будет описан в этой статье, попробуйте отложить книгу, сесть за компьютер и попытаться написать такую программу.

Мы рассмотрим красивое решение описанной задачи, использующее *метод рекурсивного спуска*. Фрагменты программы будут приводиться на языке Паскаль, однако думаем, что перевод их на язык Си (равно как и на любой другой) не должен составить большого труда даже для тех, кто с языком Паскаль не знаком.

Сначала рассмотрим более простую задачу: предположим, что все числа у нас — натуральные, а из операций встречаются только сложение и умножение.

Разбиение на лексемы

Для начала давайте напишем процедуру `nextlexem`, которая будет из выражения выделять поочередно все лексемы. *Лексема* — это минимальная единица текста, имеющая самостоятельный смысл. В нашем случае лексемами будут являться знаки арифметических операций, скобки и числа (при этом — обратите внимание! — лексемой является не каждая цифра числа, а число целиком). Эта же процедура будет игнорировать все незначимые символы — пробелы, табуляции и т. д. Еще одна лексема, которая нам понадобится, — конец строки — будет соответствовать тому, что мы хотим выделить из строки следующую лексему, а строка кончилась.

Итак, опишем следующий тип данных:

```
type TLexem = (_Num, _Plus, _Mul, _Open,
               _Close, _End);
```

Можно сделать практически то же самое по-другому:

```
type TLexem = byte;
const _Num = 0;          {Число}
      _Plus = 1;         {Знак сложения}
      _Mul = 2;          {Знак умножения}
```

```
    _Open = 3;          {Открывающая скобка}
    _Close = 4;         {Закрывающая скобка}
    _End = 5;           {Конец выражения}
```

Имена `_Num, _Plus` и т. д. мы умышленно начинаем с подчеркивания, чтобы случайно не перепутать их с именем какой-либо переменной, функции и т. д.

Введем две переменных:

```
var curlex:TLexem;          {текущая лексема}
    vl:Longint;             {значение}
```

В переменной `curlex` будем хранить текущую лексему, выделенную из выражения, каждый вызов процедуры `nextlexem` будет устанавливать значение этой переменной. И если в случае, когда текущей лексемой является `_Plus, _Mul` и т. д., эта информация является исчерпывающей, то когда текущая лексема — число (`_Num`), необходимо еще знать значение этого числа. Для этого будет использоваться переменная `vl` (ее значение также будет устанавливаться процедурой `nextlexem`).

Написание процедуры `nextlexem` никаких алгоритмических трудностей не представляет, хотя и не является совсем тривиальным. Скорее написание этой процедуры — довольно рутинная работа, требующая достаточной аккуратности. Мы оставляем ее читателям в качестве самостоятельного упражнения.

В дальнейшем будем считать, что эта процедура уже написана и ее вызов устанавливает значение глобальной переменной `curlex`, а в случае, если `curlex=_Num`, то и переменной `vl` (в остальных случаях в переменной `vl` может быть записано любой значение). Первый вызов процедуры `nextlexem` должен выделять первую лексему. Второй вызов — вторую и т. д.

Определение арифметического выражения

Теперь давайте попробуем определить, что же собственно мы хотим вычислить, т. е. что же такое арифметическое выражение. Для этого давайте ненадолго вернемся в детство и вспомним, как же происходит вычисление выражений.

Сначала выполняются все умножения и операции внутри скобок и над этими операциями записываются их результаты. Можно представить, что мы стираем знаки тех операций, которые мы выполнили, и те числа, которые в них участвовали, и записываем вместо этого результаты этих операций. Тогда на последнем шаге нам придется вычислять что-то типа `63+90+9` (здесь и дальше все вычисления мы будем показывать на примере, который уже был приведен в начале статьи: `(1+2*3)*(4+5)+6*(7+8)+9`).

Итак, можно сказать, что *арифметическое выражение* (употребляя слово «выражение», далее будем иметь в виду, конечно, арифметическое выражение) — это последовательность слагаемых (состоящая из одного или более слагаемого), разделенных знаком «+». Формально это записывается так:

```
<Выражение> ::= <слагаемое> {+ <слагаемое>}
```

Здесь фигурные скобки означают, что то, что в них записано, может быть повторено 0 или более раз (повторено 0 раз, т. е. ни разу — например, выражение `6*7` состоит из одного слагаемого).

Давайте теперь заглянем глубже и аналогично попробуем сформулировать, что же такое слагаемое. *Слагаемое* — это последовательность множителей (состоящая из одного или более множителей), разделенных знаком «*».

```
<Слагаемое> ::= <множитель> {* <множитель>}
```

Наконец, множитель — это число или выражение, заключенное в скобки:

```
<Множитель> ::= <число> | (<выражение>)
```

Здесь вертикальная черта | обозначает «или».

Таким образом, понятие выражения мы определили рекурсивно через само это понятие. Это определение кажется очень странным, поэтому мы проиллюстрируем его примером.

Итак, рассмотрим выражение `(1+2*3)*(4+5)+6*(7+8)+9` (см. рис. 1). Оно состоит из трех слагаемых: `(1+2*3)*(4+5)`, `6*(7+8)` и `9`. Первое слагаемое, в свою очередь, состоит из двух множителей: `(1+2*3)` и `(4+5)`. Первый из этих множителей — это есть выражение, взятое в скобки. В свою очередь, это выражение состоит из двух слагаемых: `1` и `2*3`. Первое из них состоит из одного множителя, который является числом `1`. Второе: `2*3` состоит из двух множителей, каждый из которых является числом. Но если бы, скажем, вместо числа `3` было записано, например `(1+2)`, то этот множитель являлся бы выражением, взятым в скобки, и в свою очередь, состоял из двух слагаемых и т. д.

Изображенная на рис. 1 структура называется *деревом разбора выражения* (`мн.` обозначает множитель, `сл.` — слагаемое, `выраж.` — выражение).

Попробуйте взять произвольное выражение и самостоятельно «разложить» его на составные части в соответствии с нашим определением.

Давайте подытожим, что же у нас получилось.

```
<Выражение> ::= <слагаемое> {+ <слагаемое>}
<Слагаемое> ::= <множитель> {* <множитель>}
<Множитель> ::= <число> | (<выражение>)
```

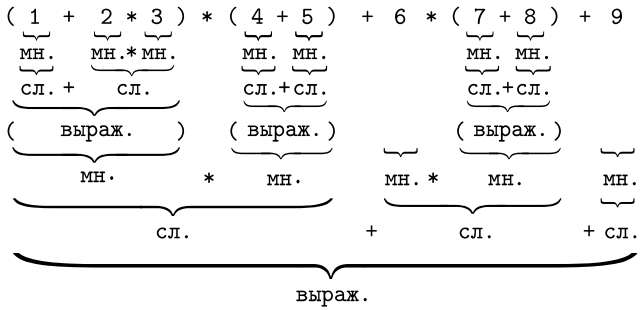


Рис. 1

Говорят, что мы определили *грамматику*, задающую арифметические выражения. А сам способ записи грамматики, который мы использовали, называется *формулами Бэкуса—Науэра*.

Написание программы

Теперь перейдем к написанию программы. Писать программу будем строго следуя нашим определениям.

Напишем три функции.

```
function expr:longint;
```

Будем считать, что при вызове этой функции в `curlex` записана первая лексема арифметического выражения. Тогда эта функция будет вычислять значение этого выражения и возвращать его в качестве своего результата. После выполнения этой функции в `curlex` будет записана первая лексема, следующая за этим выражением.

Например, если дано выражение

```
(1+2*3)*(4+5)+6*(7+8)+9
```

и при вызове функции в `curlex` записана первая открывающая скобка, то в качестве результата функции будет возвращено значение всего выражения (т. е. число `162`). А в `curlex` будет записано `_End`. Если же при вызове текущая лексема — единица (т. е. `curlex=_Num, vl=1`), то результатом будет значение выражения `1+2*3`, т. е. число `7`, а в `curlex` будет стоять закрывающая скобка, идущая после этого выражения.

Также напишем функцию

```
function item:longint;
```

которая аналогично функции `expr` будет вычислять значение слагаемого, первая лексема которого является при вызове текущей, и функцию

```
function mult:longint;
```

которая будет вычислять значение множителя.

Начнем с функции `expr`, считая, что остальные функции уже написаны.

```
function expr:longint;
var a:longint;  {эта переменная будет использоваться
                 для хранения промежуточных результатов}
begin
```

Смотрим на определение. Выражение всегда начинается со слагаемого. Заметим, что первая лексема выражения всегда является первой лексемой слагаемого. А вычислить значение слагаемого мы можем с помощью функции `item`:

```
a:=item;
```

Заметим, что после этого вызова в переменной `a` записано значение этого слагаемого, а `curlex` указывает на первую лексему за этим слагаемым. Теперь возможны две ситуации: если текущая лексема — знак «+», то за ним идет еще одно слагаемое, дальше снова может идти «+» и т. д. Второй вариант — не «+», но тогда это уже к выражению отношения не имеет, т. е. выражение закончилось. Итак:

```
while curlex=_Plus do begin
  {Раз текущая лексема - плюс, то мы должны, запомнив это,
                                перейти к следующей лексеме}
  nextlexem;
  {теперь curlex - первая лексема слагаемого, которое надо
                        прибавить к тому, что записано в a}
  a:=a+item;
  end;
expr:=a;
end;
```

Итого, нашу функцию мы написали буквально в 5 строк.

Теперь напишем функцию `item`. Заметьте, что определение слагаемого полностью аналогично определению выражения. Значит, и функция будет полностью аналогична функции `expr`:

```
function item:longint;
var a:longint;
begin
a:=mult;
while curlex=_Mul do begin
  nextlexem;
  a:=a*mult;
  end;
item:=a;
end;
```

Осталось совсем чуть-чуть: написать функцию `mult`. Сделаем это тоже в соответствии с определением.

```
function mult:longint;
begin
case curlex of
_Num: begin
  {текущая лексема - число, значит, множитель равен числу,
                                        хранящемуся в переменной vl}
  mult:=vl;
  {но дальше надо не забыть сдвинуться на следующую лексему -
   ведь curlex должно указывать на следующую после множителя лексему}
  nextlexem;
  end;
_Open: begin
  {текущая лексема - открывающая скобка}
  nextlexem; {сдвигаемся на следующую лексему}
  {теперь мы стоим на первой лексеме выражения, значение которого
                                    и будет значением нашего множителя}
  mult:=expr;
  {теперь мы стоим на первой лексеме после выражения.
  Это должна быть закрывающая скобка, которую нужно "проглотить" -
  ведь мы должны закончить на следующей за нашим множителем лексеме,
                               а скобка - последняя лексема множителя}
  if curlex=_Close then nextlexem
  else error; {если вдруг оказалось, что текущая лексема была
  не закрывающая скобка, значит, наше выражение содержит ошибки,
  например: (1+2, т.е. скобка не закрывается, тогда мы должны выдать
         сообщение об ошибке, пусть у нас это делает процедура error}
  end
else error;
  {если первая лексема вычисляемого множителя - не число
  и не открывающая скобка, то это опять же является признаком того,
                               что в выражении ошибка, например: 1**2}
end;
```

Осталось написать основную программу.

Помимо описанных ранее типа `TLexem` и переменных `curlex` и `vl`, нам потребуется еще одна переменная:

```
var v:longint;
begin
{здесь должна идти инициализация переменных,
             ввод выражения пользователем - оставляем это вам}
nextlexem; {счиываем первую лексему. Теперь в curlex записана
            первая лексема нашего выражения,
                   и его можно вычислить, вызвав функцию expr}
v:=expr;
{теперь, по идее, мы должны оказаться в конце выражения,
             если это не так, значит, в выражении есть ошибки}
if curlex<>_End then error;
writeln(v);  {печатаем результат}
end. {все!}
```


Это действительно все. Похоже на фокус.

Осталась, пожалуй, лишь одна техническая проблема. Функция `expr` использует функцию `item`, `item` использует `mult`, а `mult` использует `expr`. Какая из них должна быть описана раньше? Оказывается, что Паскаль дает возможность решить эту проблему. Можно, например, описать функции в таком порядке: сначала `mult`, потом — `item`, и наконец `expr`. А для того чтобы уже в функции `mult` компилятор знал, что функция `expr` присутствует в программе, но будет описана позднее, нужно перед описанием `mult` написать прототип (заголовок) функции `expr` и ключевое слово `forward`, которое как раз и говорит компилятору, что функция будет описана позднее. Таким образом,

```
function expr:longint; forward;
```

Пример

Давайте посмотрим, как это все будет работать.

Итак, пусть нам дали наше любимое выражение:

`(1+2*3)*(4+5)+6*(7+8)+9.`

Первый вызов `nextlexem` из основной программы установит `curlex==_Open`. Дальше мы вызываем `expr`, `expr` устроена так, что она сразу вызывает `item`, а `item` вызывает `mult`. `mult` видит открывающую скобку, «проглатывает» ее, в `curlex` оказывается `_Num` и `vl=1`. Дальше мы вызываем `expr`, `expr` устроена так, что она сразу вызывает `item`, а `item` вызывает `mult`. `mult` видит, что `curlex=_Num`, тогда она возвращает значение `vl=1` и вызывает `nextlexem`. Теперь `curlex=_Plus`. После этого `mult` завершается и мы попадаем в `item`, из которого произошел вызов. Заметьте, что мы как раз стоим на первой лексеме после считанного множителя. `_Plus` — это не `_Mul`, поэтому `item` считает, что слагаемое закончилось (тем самым, оно состоит из одного множителя, и это действительно так) и мы возвращаемся в `expr`.

`expr` видит `_Plus` и понимает, что в выражении есть еще одно слагаемое. Запоминая в `a` значение `1`, которое является значением первого слагаемого, она вызывает `nextlexem`, после чего `curlex=_Num, vl=2`. Вызываем `item`, `item` вызывает `mult`. `mult` возвращает `vl=2` и делает `nextlexem`,

после чего `curlex=_Mul`. Тем самым, в `item` мы попадаем в цикл `while`, делаем `nextlexem` (в результате `curlex=_Num, vl=3`) и снова вызываем `mult`. `mult`, делая `nextlexem` (в результате `curlex=_Close`), возвращает нам 3. `item` умножает 2 на 3 и, поскольку `curlex<>_Mul`, цикл завершается. Значением `item` является `6`, и, возвращаясь в `expr`, мы к имевшемуся там числу `1` прибавляем `6`. После чего `curlex<>_Plus`, и `expr` также завершается со значением `1+6=7`.

Мы вернулись в `mult`, и в `curlex` должна быть закрывающая скобка, что и имеет место быть. Мы делаем `nextlexem` и возвращаемся со значением `7` в `item`. Дальше попробуйте смоделировать работу программы самостоятельно (кажется, что это легче проделать самому, чем про это читать).

Заметьте, что весь подсчет происходит строго в соответствии с нашим определением.

Обобщение

Что же делать, если в нашем выражении присутствуют вычитание и деление, возведение в степень, унарные минусы, дробные числа, символы переменных и т. д.? Это потребует изменения определения того, что же является выражением, а соответственно, и вычисляющих его значение функций. Покажем, например, как изменится определение выражения, а соответственно, и фукнция `expr`, если добавить еще и вычитание.

```
<Выражение> ::= <слагаемое> {(+|-) <слагаемое>}
```

Еще раз напомним, что вертикальная черта обозначает «или».

```
function expr:longint;
var a,b:longint;
    c:TLexem;
begin
a:=item;
while (curlex=_Plus) or (curlex=_Minus) do begin
  c:=curlex;
  nextlexem;
  b:=item;
  if c=_Plus then a:=a+b else a:=a-b;
  end;
expr:=a;
end;
```

Деление, естественно, добавляется в определение слагаемого (там же, где умножение). Полезно не забыть, выполняя деление, проверить, что делитель не равен 0. При появлении вещественных чисел нужно не забыть

изменить тип результата, возвращаемого функциями, и типы переменных для промежуточных значений, а также добавить разбор дробных чисел в процедуру `nextlexem`.

Добавление переменных и вызовов функций отразится на вычислении множителя (в функции `mult` в операторе `case` добавятся новые ветви). Например, если в выражении возможны переменные, то можно завести специальный тип лексем — переменная, а в переменную `vl` в этом случае записывать числовой номер этой переменной (храня отдельный массив соответствий между именами переменных и их номерами). Немного иначе добавляется операция возведения в степень — она потребует добавления нового правила в грамматику.

Метод рекурсивного спуска можно использовать и при вычислении логических выражений, и даже для разбора программы на языке Паскаль (хотя грамматика языка в этом случае будет значительно сложнее). Однако стоит заметить, что бывают такие языки, для которых метод рекурсивного спуска не применим при разборе программ. Соответствующие примеры можно найти в литературе.

В заключение хотелось бы поблагодарить А. В. Чернова и В. М. Гуровица за ряд ценных замечаний по тексту этой статьи.

Литература

1. *Ахо А.*, *Сети Р.*, *Ульман Д.* Компиляторы. Принципы, технологии, инструменты. М.: Издательский дом «Вильямс», 2001.

Преподавание программирования с использованием системы автоматической проверки решений

В. А. Матюхин

В этой статье мы немного поговорим о том, зачем и как можно учить школьников программированию. Эта статья адресована в первую очередь учителям информатики.

Компьютер — действующее лицо педагогического процесса

Программирование — уникальный предмет. Дело в том, что при изучении почти любого другого предмета учебный процесс строится на взаимодействии ученика и учителя. При изучении же программирования в этот процесс включается еще одно действующее лицо — компьютер.

Что фактически требуется от школьника на первых порах изучения программирования? Взять задачу, которую он отлично понимает как нужно решать без компьютера (например, задачу поиска минимума среди набора чисел), после чего попытаться осознать, что же он (школьник) делает, решая данную задачу, и сформулировать это словами, а затем перевести этот алгоритм на язык машины и получить программу.

И вот здесь в действие вступает компьютер. Во-первых, если в программе допущены синтаксические ошибки, то компилятор сразу на это укажет. Во-вторых, когда все синтаксические ошибки исправлены, часто программа все равно не работает. И опять же ребенок может увидеть это сам, запустив программу на каком-нибудь примере.

Очень важно, что ребенок имеет возможность увидеть, а зачастую и исправить ошибки без всякого вмешательства учителя. Во-первых, в этом случае значительно снижается время получения «обратной связи» школьником (учитель просто физически не имеет возможности с такой скоростью просматривать работы школьников и указывать на ошибки — как правило, на других предметах школьники сначала в течение какого-то времени выполняют работу, потом сдают и на следующем уроке получают проверенную работу). Во-вторых, когда на ошибку указывает не товарищ, не учитель, а бездушное устройство, у школьника исчезает страх ошибку совершить. Пользуясь тем, что можно быстро что-нибудь исправить, снова запустить программу и очень быстро узнать, что и этот вариант неверен (или, наоборот, верен), школьник довольно активно пробует разные варианты. Тем самым, он учится самостоятельно преодолевать проблемы, искать источники ошибок и т. д.

Роль отладчика

Еще один очень полезный инструмент при изучении программирования — отладчик. С его помощью школьник может выполнять программу по шагам и отслеживать значения переменных. Фактически отладчик позволяет школьнику сравнить, что должно происходить при выполнении программы по мнению школьника и что происходит на самом деле.

Мне очень нравится сравнение роли отладчика с зеркалом. Когда человек учится танцевать, он делает это перед зеркалом, чтобы видеть, что у него получается. Когда человек учится программировать, фактически он учится излагать свои мысли на языке программирования — абсолютно строгом, формальном языке. Программа — это изложение мыслей школьника о том, что нужно делать для решения задачи. Отладчик позволяет увидеть, где то, что написано в программе, не совпало с тем, что он хотел написать.

Пользоваться отладчиком не всегда просто. Но научить школьников им пользоваться очень важно. Часто школьнику значительно проще обратиться к учителю с вопросом «а где у меня ошибка?», и бывает, что учителю достаточно одного взгляда на программу, чтобы эту ошибку увидеть и указать на нее школьнику. Однако обучающий эффект будет значительно больше, если школьник найдет эту ошибку сам.

Система автоматической проверки

Итак, школьник добился того, что программа работает на каком-то примере. Вроде бы задача решена. Это далеко не всегда так. В задаче могут быть хитрые случаи, специально спрятанные «грабли», или программа написана так, что работает лишь в случаях специального типа, но именно на таком примере и проверил свое решение школьник.

Теперь, по идее, к школьнику должен подойти учитель и проверить его программу. Запустить ее несколько раз, предложив ей разные входные данные. Однако, во-первых, учитель физически не может запустить программу больше чем на 2-3 примерах, и может так получиться, что и эти тесты программа пройдет. Во-вторых, входные данные в задаче могут быть достаточно большими, и учителю потребуется довольно много времени на то, чтобы их набить (а ведь надо набить без ошибок!). В-третьих, ответ на этих примерах может быть неоднозначным, и потребуется еще время на то, чтобы понять, правильный ответ вывела программа школьника или нет. На самом деле, эту проверку можно переложить с учителя на компьютер.

Компьютер сам может давать программе школьника наборы входных данных («подсовывая» ей разные входные файлы или эмулируя ввод с клавиатуры). Сам запускать программу школьника и анализировать выданный ею ответ.

Это и есть автоматическая проверка решения. При этом система автоматической проверки — не универсальная программа, которая может проверять решение любой задачи. Это программа, которая умеет «подсовывать» тесты (т. е. наборы входных данных), запускать решение, анализировать результат и, быть может, как-то вести протокол, и поддерживать таблицу результатов. Чтобы система могла проверять некоторую задачу, нужно, чтобы были подготовлены тесты специально для этой задачи.

Причем, в отличие от учителя, компьютер может проверить решение очень качественно — за минуту можно прогнать до сотни разных тестов (в том числе и с достаточно большими по объему входными данными). Если все они прошли, то программа признается правильной. Если какой-то тест не прошел — то, очевидно, программа содержит ошибку.

Дальнейшая тактика может быть разной. Школьнику может выдаваться этот тест (т. е. набор входных данных), чтобы, пользуясь отладчиком, школьник мог найти и исправить ошибку. Так правильно поступать на первых порах изучения программирования. А можно не выдавать тест, а лишь сообщить о том, что в программе имеется ошибка (часто принято сообщать номер теста, чтобы школьник мог сравнить, к чему привели его исправления по сравнению с предыдущей попыткой сдать задачу — программа стала работать хуже и проходить меньше тестов, или наоборот, ему удалось исправить ошибку, из-за которой программа не работала в прошлый раз, и теперь нужно искать еще какие-то ошибки). В такой ситуации школьник учится анализировать задачу, выделять из нее ключевые моменты, понимать, что в его программе может приводить к ошибке. Это очень непростое, но очень важное умение.

Требования к программам

Для того чтобы программы могли проверяться автоматически, они должны удовлетворять некоторым формальным требованиям. В частности, входные данные даются в строго определенном формате, выходные данные программа также должна выдавать в строго определенном формате. Например, если программа работает абсолютно правильно, но выдает ответ — два числа — не в требуемом, а в обратном порядке, то такая программа не будет принята.

Однако, как показывает практика, школьники через некоторое время привыкают к такого рода формальным требованиям, и это почти перестает вызывать у них затруднения. Более того, как мне кажется, научить школьников строго следовать некоторой формальной спецификации — это тоже одна из целей обучения, и такой навык пригодится им практически в любой сфере деятельности.

Иногда, правда, при сдаче решений системе автоматической проверки возникают очень специфические проблемы. Например, когда ребенок запускает программу из среды разработки, все переменные автоматически обнуляются и программа работает верно, а при запуске под тестирующей системой в какой-нибудь ячейке памяти оказывается не ноль, и программа не работает из-за того, что не все переменные в ней инициализируются. Но это уже скорее проблема не системы проверки, а программирования в целом. И умение не забывать про такие детали, а тем более умение понимать, из-за чего может возникать проблема, умение находить и исправлять ошибку даже в том случае, когда воспроизвести ее не удается, — это очень важное качество профессионального программиста. И в общем, никуда от таких деталей не деться — ведь программирование складывается, в частности, и из этого.

Авторитет системы

Как уже говорилось, чтобы можно было проверять решения некоторой задачи, должны быть подготовлены тесты именно для этой задачи. Кроме того, когда ответ в задаче неоднозначен, часто требуется написание программы, которая по входным данным и выданному проверяемой программой результату устанавливает его правильность. Это непростая работа, которая требует специальных умений. В последнее время на сайтах (преимущественно посвященных олимпиадам по программированию, так как технология автоматической проверки решений изначально применялась именно на олимпиадах по программированию и лишь потом была перенесена в учебный процесс) накоплена довольно большая база задач с тестами, которые можно использовать. Это значительно облегчает подготовку к занятиям. Однако часто требуется адаптация тестов к их использованию под конкретной тестирующей системой.

Важно, чтобы тесты не содержали ошибок. Первая реакция школьника на сообщение системы об ошибочности решения — «у меня все правильно, это у вас неправильные тесты». И здесь то, что на первых порах школьник получает тот тест, на котором его программа не работает, очень важно. Он начинает смотреть этот тест и действительно находит ошибку в своей программе. Так формируется «авторитет» системы.

Если же часто будет оказываться, что решение школьника абсолютно правильное, и он был вынужден (из-за ошибок в тестах и в проверяющей программе) искать в нем несуществующую ошибку и потратил на это очень много времени, то это, наоборот, работает на разрушение авторитета. Если же у системы не будет авторитета, убедить школьника искать ошибку и добиваться прохождения всех тестов будет очень-очень сложно.

Еще один важный момент — это то, что система тестов для задачи должна быть полной. Если в решении есть ошибка, то она обязательно должна тестами находиться. Например, если задача формулируется: «дано до 100 чисел, найти минимальное из них», то обязательно должен быть

тест именно на 100 чисел (потому что если школьник, например, несколько некорректно работает с массивом и обращается к (N+1)-му его элементу, то эта ошибка проявится лишь на таком тесте). Полнота системы тестов также работает на укрепление «авторитета» системы — получить от системы сообщение «задача решена» становится престижно.

И, сдавая свое решение на проверку, школьник в этой ситуации уверен, что его решение будет полноценно проверено (в противовес ситуации, когда школьник долго-долго возился с программой, подошел учитель, запустил ее на одном тесте и сказал: «Молодец, Вася»).

Еще один аспект автоматической проверки — полная беспристрастность. Все решения проверяются абсолютно одинаково. У системы не бывает «любимчиков», которым прощается то, что не прощается другим, не бывает нелюбимых учеников.

Роль учителя

Может показаться, что при использовании системы автоматической проверки учитель вообще не нужен. Это не так.

Благодаря использованию системы автоматической проверки, во время урока у учителя освобождается время, и теперь он может больше времени посвятить тем ученикам, которым нужна его помощь.

Учитель нужен, чтобы помогать ученикам искать нетривиальные ошибки в программах, когда им не удается найти их самостоятельно.

Учитель нужен, чтобы давать подсказки, наводящие предложения ученикам, когда у них что-то не получается.

Учитель нужен, чтобы с учениками, решившими задачу, обсуждать, какие еще бывают способы ее решения, как решить задачу красивее.

Наконец, полезно время от времени после сдачи программы системе автоматической проверки заставлять школьников сдавать программу учителю. Во-первых, при автоматической проверке не отслеживается, чтобы код программы был написан красиво. Конечно, понятие красоты очень субъективное. Однако нужно требовать от школьников выделения логических блоков, циклов и т. д. Как уже говорилось, программа — это отражение мыслей школьника. Наводя «красоту» в программе, на самом деле мы наводим порядок в его мыслях. А пока в программе каша — в голове будет то же самое.

Кроме того, изучая код программы, можно увидеть и показать школьнику, что и как можно делать красивее и проще.

Чему учатся школьники, изучая программирование

Прежде всего, школьники учатся очень точно и очень формально излагать свои мысли.

Во-вторых (и об этом тоже уже говорилось), они учатся работать самостоятельно, экспериментировать, искать пути решения возникшей проблемы.

Школьники учатся анализировать задачу, выделять, какие бывают ключевые моменты, где можно допустить ошибку.

Школьники учатся тестировать свои программы, а значит, учатся относиться критически к продуктам своего труда. Рассмотрим ситуацию, когда школьник получил от системы ответ, что в программе есть ошибки, но не получил теста, на котором эта ошибка проявляется. Тогда ему прежде всего нужно подобрать тест, на котором его программа работает неправильно (а дальше с помощью отладчика найти, что же именно в ней неверно). Возникает ситуация, когда школьник радуется, если ему удается найти тест, на котором его программа не работает. Согласитесь, что в педагогическом процессе такая ситуация бывает не часто.

Написание программ позволяет школьнику лучше понять как устроен и работает компьютер, какие задачи с его помощью можно решать.

Решая уже более сложные задачи, участвуя в олимпиадах, школьник знакомится с теорией алгоритмов. Учится применять стандартные алгоритмы и адаптировать их для нестандартных задач. Учится строить математическую модель по словесному описанию условия задачи. Учится оценивать сложность алгоритма и скорость работы программы (часто до того, как начинает эту программу писать).

Участвуя в командных олимпиадах, школьники учатся работать в команде, распределять обязанности, совместно работать на общий результат.

Мне кажется, что большинство этих навыков будут полезны, даже если в дальнейшем жизнь школьника никак не будет связана с программированием.

Благодарности

Эта статья написана по опыту преподавания в Московской гимназии на Юго-Западе № 1543, а также по опыту проведения Летних компьютерных школ (`www.olympiads.ru/sis`). Многие идеи, которые я использую в своей практике, принадлежат моему учителю информатики, декану Вятского государственного гуманитарного университета Окулову Станиславу Михайловичу, заведующей кафедрой информатики СУНЦ МГУ Андреевой Елене Владимировне, руководителю Мытищинской школы программистов Шедову Сергею Валерьевичу. Родоначальником систем автоматической проверки в России был Суханов Антон Александрович — многие заложенные им идеи до сих пор используются в архитектуре проверяющих систем. Мне хотелось бы выразить этим людям огромную признательность. А также поблагодарить Чернова Александра Владимировича, Станкевича Андрея Сергеевича, Корнеева Георгия Александровича, Умнова Владими-

ра Игоревича и Каменских Дениса Вячеславовича, чьими тестирующими системами мне неоднократно приходилось пользоваться.

Ссылки

Тестирующие системы:

`http://www.ejudge.ru`
`http://olympiads.ru/school/system/download/t-run/index.shtml`
`http://olympiads.ru/school/system/download/olympiads/index.shtml`

Сайты, где можно найти задачи с тестами:

`http://www.olympiads.ru`
`http://neerc.ifmo.ru/school`
`http://www.informatics.ru`

Задачи для школьных уроков информатики можно найти здесь:

`http://olympiads.ru/problems`

ТЕМАТИЧЕСКИЙ РУБРИКАТОР ЗАДАЧ

*Здесь приведен полный список задач книги, упорядоченный по темам. Внутри тем мы попытались упорядочить задачи по возрастанию сложности, однако этот критерий безусловно является субъективным. Некоторые задачи попали сразу в несколько разделов. Звездочкой * отмечены задачи повышенной сложности.*

Задачи для начинающих

V-A. Автобусная экскурсия
I-C. Клад
IV-G. Распаковка строки
VII-A. Москва-сортировочная
I-A. Таймер
VIII-B. Выборы жрецов
V-B. Сапер
VII-F. Двоякие числа
VIII-A. Часы с боем
II-A. Подсчет баллов
IX-A. Результаты олимпиады
I-D. Забавная игра
V-C. Робот К-79
III-D. Квадрат
II-F. Шашки
IV-F. Луч света в темном царстве

Сортировка

VII-I. Сортировка масс
II-D. Коллекционирование этикеток
VI-E. Сплоченная команда
III-G. Реклама
IX-E. T2005
IX-C. Современники

Структуры данных

VI-G. Strategy tetris
VIII-J. Лотерея
VIII-K. Коммерческий калькулятор

Строки

V-D. Многочлен
V-E. Головоломка
VII-C. EuroEnglish

IX-E. T2005
IV-F. Луч света в темном царстве

Алгоритмы на строках

III-E. Поле чудес
VIII-H. A-функция от строчки

Перебор

IX-H. Сверим часы
VIII-G. Скорая помощь
IX-G. Монетки
I-H. Раскопки
III-E. Поле чудес
V-K. Склад*

Динамическое программирование

IV-C. Операции с валютой
III-B. Покупка билетов
VII-B. Кафе
IX-J. Количество треугольников
VI-C. Маскарад
VII-E. Скобки
II-E. Еловая аллея
VIII-J. Лотерея
II-J. Анаграммер
V-J. Узор*
I-I. Деревни*

Алгоритмы на графах

Поиск в ширину

II-C. Водостоки
V-H. Побег с космической станции
I-G. Игра с фишками
III-C. Вырезанные фигуры
IX-F. Роботы

Алгоритм Дейкстры

VIII-I. Олимпиада по алхимии
I-B. Домой на электричках
II-H. «Левый лабиринт»

Алгоритм Флойда

III-F. Детская игра со спичками
I-I. Деревни*

Игры и стратегии

II-B. Великая сеча
VIII-E. Магия Копперфильда
II-D. Коллекционирование этикеток
V-G. Разноцветные треугольники
VII-H. Калах
VI-B. Раскраска плиток

Делимость

IX-B. Сокращение дроби
VIII-C. Представление чисел
I-F. Степень
IV-B. Сломанный калькулятор
VIII-F. Кубическое уравнение
VIII-G. Скорая помощь
IX-D. Тройки чисел

Задачи на разные темы

V-F. Поиск прямоугольников
III-A. Наибольшее произведение
VI-A. Праздники
IV-E. Черно-белые палиндромы
VI-D. Билетики
VIII-D. Гексагон
VII-D. D++
III-C. Вырезанные фигуры
IV-A. Перегоны
IV-D. Двухтуровая олимпиада
II-G. Мышка с колесиком
V-L. Кубическая гостиница

Геометрия

I-E. Целые точки
V-I. Неподвижная точка карты
IX-I. Разрезанный прямоугольник
II-I. Дремучий лес - 2
IV-H. Дремучий лес
VI-F. Сократи векторы
VII-G. OOO*
VIII-L. Пересечение кубов*

МОСКОВСКИЕ ОЛИМПИАДЫ ПО ИНФОРМАТИКЕ

Под ред. Е. В. Андреевой, В. М. Гуровица и В. А. Матюхина

Корректор С. А. Ботова

Подписано в печать 10.04.2006 г. Формат $60 \times 90\ {}^{1}/_{16}$. Бумага офсетная.
Печать офсетная. Печ. л. 16. Тираж 3000 экз. Заказ №

Издательство Московского центра непрерывного математического образования
119002, Москва, Большой Власьевский пер., 11. Тел. 241–74–83.

Отпечатано с готовых диапозитивов в ФГУП «Полиграфические ресурсы».