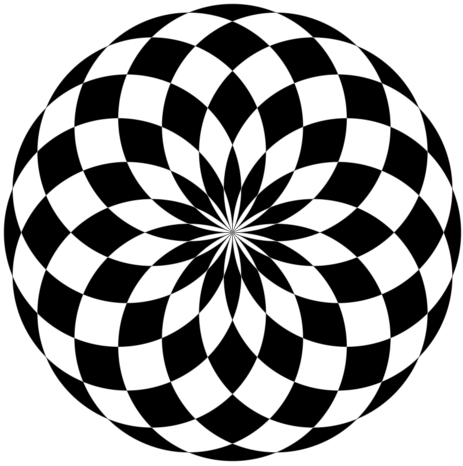
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ (ПОВОЛЖСКИЙ) УНИВЕРСИТЕТ
РЕСПУБЛИКАНСКИЙ ОЛИМПИАДНЫЙ ЦЕНТР РТ

Олимпиады по информатике школьников Татарстана

2015-2016 и 2016-2017



Казань – 2017

УДК 372.800.4
ББК 74.263.2

Печатается по решению учебно-методической комиссии
Института математики и механики КФУ
им. Н.И. Лобачевского

Киндер М.И.

Олимпиады по информатике школьников Татарстана. 2015-2016 и 2016-2017: Учебно-методическое пособие / М.И. Киндер. — Казань: Казанский федеральный университет, 2017. — 113 с.

Брошюра предназначена для школьников, учителей, преподавателей и тренеров по олимпиадной информатике. В ней представлены задачи, предлагавшиеся в 2015-2016 и 2016-2017 учебных годах на муниципальном и региональном этапах Всероссийской олимпиады школьников Татарстана по информатике. Для каждой задачи приведены идея решения, система оценки и описание конкретной реализации на одном из языков, принятом на олимпиадах по спортивному программированию.

2015-2016 учебный год

Муниципальный этап 27-й Всероссийской олимпиады по информатике среди школьников Республики Татарстан состоялся 7 декабря 2015 г. В составлении задач муниципальной олимпиады принимали участие преподаватели Казанского Федерального Университета:

М. И. Киндер, А. А. Попов, Р. Р. Гайфуллин.

Региональный этап олимпиады проходил с 30 января по 1 февраля 2016 г. Кроме школьников 9-11 классов — традиционных участников регионального этапа — на олимпиаду были приглашены также ученики 7-8 классов, показавшие хорошие результаты на муниципальном этапе Всероссийской олимпиады. Специально для школьников 7-8 классов жюри составило несколько задач, сложность которых, по мнению жюри, соответствовала этой возрастной категории участников олимпиады.

Материалы муниципального и регионального этапа Всероссийской олимпиады по информатике среди школьников Республики Татарстан (результаты участников, архив жюри с тестами, генераторами тестов и решениями жюри) доступны в интернете по адресу:

http://kpfu.ru/math/olimpiady-dlya-shkolnikov-i-studentov/olimpiady-shkolnikov-po-informatike

Ниже представлены условия и подробные решения задач всех перечисленных олимпиад. Для каждой задачи приведена идея решения, система оценки и описание конкретной реализации на языке Pascal или C++. Для большинства олимпиадных заданий указаны фамилии авторов идеи и фамилии тех, кто готовил эти решения.

Муниципальный этап, 2015-2016

Задача 1. Две доминошки (7-11 классы)

Имя входного файла:	`domino.in`
Имя выходного файла:	`domino.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Доминошка представляет собой прямоугольную плитку размером 1×2, разделенную на две половинки. На каждой из них нарисовано от 0 до 6 точек. По правилам игры две доминошки можно поставить рядом, если у них есть половинки с одинаковым числом точек. Например, доминошки $\boxed{1}\boxed{2}$ и $\boxed{1}\boxed{4}$, а также $\boxed{0}\boxed{1}$ и $\boxed{5}\boxed{1}$ можно поставить рядом, а доминошки $\boxed{1}\boxed{2}$ и $\boxed{3}\boxed{4}$ — нельзя. (Числа означают количества точек на половинках доминошек.)

Вам необходимо для заданной пары доминошек определить, можно ли их поставить рядом друг с другом.

Формат входных данных

В первой строке записаны через пробел два целых числа a и b — количество точек на половинках первой доминошки ($0 \leqslant a, b \leqslant \leqslant 6$). Во второй строке записаны через пробел два целых числа c и d — количество точек на половинках второй доминошки ($0 \leqslant c, d \leqslant 6$). Гарантируется, что доминошки отличаются количеством точек хотя бы на одной из половинок.

Формат выходных данных

Выведите число -1, если доминошки нельзя поставить рядом друг с другом. В противном случае запишите через пробел исходные числа a, b, c, d в порядке расположения доминошек по правилам игры. Если решений несколько, выведите любое из них.

Система оценивания

Задача оценивается в 100 баллов. Баллы начисляются, только если пройдены *все* тесты.

Примеры

domino.in	domino.out
0 1 1 4	0 1 1 4
0 1 5 1	0 1 1 5
1 2 3 4	-1

Задача 2. Красивое число (7-8 классы)

Имя входного файла:	`beauty.in`
Имя выходного файла:	`beauty.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Целое положительное число m мы назовём *красивым*, если в его записи между любыми двумя чётными цифрами есть хотя бы одна нечётная цифра, а между любыми двумя нечётными цифрами — хотя бы одна чётная цифра.

Вам нужно определить, будет ли данное m красивым числом.

Формат входных данных

В первой строке одно целое число n — количество цифр в записи числа m ($1 \leqslant n \leqslant 10^6$). Во второй строке записаны без пробелов n цифр от 0 до 9. Первый символ в этой строке — ненулевая цифра.

Формат выходных данных

Выведите «`Yes`», если данное число красивое, и «`No`» — в противном случае.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	20	$1 \leqslant n \leqslant 100$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	80	$1 \leqslant n \leqslant 10^6$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Примеры

beauty.in	beauty.out
3 123	yes
5 18190	no

Задача 3. Очередь (7-11 классы)

Имя входного файла:	`queue.in`
Имя выходного файла:	`queue.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

У кассы стадиона стоит длинная очередь из n человек. Как обычно, на время обеденного перерыва кассу закрыли, и недовольная очередь футбольных фанатов разошлась по своим делам. Когда обед подходил к концу, все снова собрались у кассы. Ну и как же их теперь расставить в прежнем порядке? К счастью, все футбольные фанаты носили футболки с различными номерами на спине и каждый из них помнил номер на футболке стоявшего

перед ним. Разумеется, кроме первого, стоявшего у кассы.

Вам необходимо восстановить порядок стоявших в очереди фанатов.

Формат входных данных

В первой строке записано одно целое число n — количество фанатов в очереди ($2 \leqslant n \leqslant 2 \cdot 10^6$). Следующие $n-1$ строк содержат по два разделённых пробелом целых числа a и b — номера на футболках стоявших рядом друг с другом фанатов, где a — номер на футболке фаната, стоявшего за фанатом в футболке с номером b ($1 \leqslant a, b \leqslant n$).

Формат выходных данных

В единственной строке запишите через пробел n целых чисел — номера на футболках фанатов в обратном порядке очереди, начиная с последнего и заканчивая первым, стоявшим у кассы.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	50	$2 \leqslant n \leqslant 40\,000$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	50	$2 \leqslant n \leqslant 2 \cdot 10^6$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Примеры

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

Задача 4. Две дроби (7-11 классы)

Имя входного файла:	fractions.in
Имя выходного файла:	fractions.out
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Однажды мудрая Сова подарила ослику Иа-Иа на его день рождения скромный, но очень полезный вычислительный прибор, который умеет выполнять три операции:

(**A**) прибавить единицу к данному числу;
(**B**) вычесть единицу из данного числа;
(**C**) заменить ненулевое число на обратное к нему.

К сожалению, на клавиатуре прибора отсутствуют многие клавиши, поэтому некоторые числа приходится получать из других чисел с помощью указанных трёх операций. Представьте, что ослику Иа-Иа из дроби a/b нужно получить дробь c/d.

Как ему это сделать, используя только операции **A**, **B** и **C**?

Формат входных данных

В первой строке записаны через пробел два числа a и b — числитель и знаменатель несократимой дроби a/b, во второй строке также записаны через пробел два числа c и d — числитель и знаменатель несократимой дроби c/d. Все числа a, b, c, d целые, $1 \leqslant a, b, c, d \leqslant 10^6$. Дроби a/b и c/d различны.

Формат выходных данных

В первой строке запишите одно число n — количество необходимых операций, которое не должно превышать 2 000 001. Во второй строке укажите последовательность из n символов A, B и C латинского алфавита — операций, с помощью которых из числа a/b можно получить c/d. Если решений несколько, выведите любое из них. Если из дроби a/b получить дробь c/d невозможно, выведите -1.

Система оценивания

Задача оценивается в 100 баллов. Баллы начисляются за каждый пройденный тест.

Примеры

fractions.in	fractions.out
3 2 3 1	3 BCA
3 2 2 3	1 C

Задача 5. Муравей на кубе (9-11 классы)

Имя входного файла:	cube.in
Имя выходного файла:	cube.out
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Муравей Яша гуляет по поверхности куба с ребром длины a. По сигналу тревоги он бежит к входу в муравейник, который находится в одной из вершин куба. Из всех возможных путей муравей всегда выбирает самый короткий, и поэтому ему нужно заранее узнать длину кратчайшего пути до муравейника.

Формат входных данных

В первой строке записано единственное целое число a — длина ребра куба ($1 \leqslant a \leqslant 10^9$). Вторая строка содержит три разделенных пробелами целых числа x, y и z — координаты муравья

$(0 \leqslant x, y, z \leqslant a)$. Гарантируется, что числа x, y и z определяют точку на поверхности куба с ребром a. Начало системы координат совпадает с входом в муравейник, оси координат направлены вдоль рёбер куба.

Формат выходных данных

Выведите длину кратчайшего пути муравья. Ответ считается правильным, если абсолютная или относительная погрешность не превышает 10^{-4}.

Система оценивания

Номер подзадачи	Баллы	Ограничения a, x, y, z	Комментарии
1	20	$1 \leqslant a \leqslant 10^4$, $xyz = 0$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	60	$1 \leqslant a \leqslant 10^4$, $xyz > 0$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	20	$1 \leqslant a \leqslant 10^9$, $xyz > 0$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

cube.in	cube.out
10 0 0 10	10.0000
10 3 4 0	5.0000

Задача 6. Факторизации (9-11 классы)

Имя входного файла:	`factor.in`
Имя выходного файла:	`factor.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Ванька Жуков, студент первого курса, выучившийся три месяца назад всяким штукам из университетского курса теории чисел, долго не ложился спать. Дождавшись, когда всё в доме стихло, он достал из рюкзака изрядно потёртый планшет и занялся своим любым делом — *факторизацией*. Этим словом он называл разложение произвольного натурального числа n числа в произведение целых положительных чисел, бóльших 1. Два разложения числа n, которые отличались лишь порядком сомножителей, Ванька Жуков считал одинаковыми. Например, число 12 в его подсчётах имело 4 различные факторизации: 12, $6 \cdot 2$, $4 \cdot 3$ и $3 \cdot 2 \cdot 2$. Среди факторизаций Ванька любил выделять такие, у которых наибольший множитель не превосходит заданного числа m.

Ванька перевёл глаза на тёмное окно, в котором мелькало отражение планшета, и, кажется, представил решение задачи в общем случае. Для проверки его вычислений вам необходимо подсчитать количество факторизаций данного числа n с наибольшей частью, не превосходящей m.

Формат входных данных

В единственной строке записаны два целых числа n и m ($1 \leqslant n \leqslant 10^8$, $1 \leqslant m \leqslant n$).

Формат выходных данных

Выведите одно целое число — искомое количество факторизаций.

Примеры

factor.in	factor.out
1 1	0
12 6	3
12 12	4

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	20	$1 \leqslant n \leqslant 100$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	80	$1 \leqslant n \leqslant 10^8$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Региональный этап, 2015-2016

Задача 1. Призы

Имя входного файла:	`prizes.in`
Имя выходного файла:	`prizes.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Пётр участвует в конкурсе, в котором разыгрывается n призов. Призы пронумерованы от 1 до n.

По итогам конкурса участник может набрать от 2 до n баллов. Если участник наберет k баллов, то он получит один из призов с номером от 1 до k. Перед тем, как участник выберет свой приз, ведущий конкурса удаляет один из призов из списка. Затем участник может выбрать любой приз из оставшихся $k - 1$.

Список призов стал известен Петру. Он определил для каждого приза его ценность, для i-го приза она задается целым числом a_i.

Требуется написать программу, которая по заданным ценностям призов определяет для каждого k от 2 до n, приз с какой максимальной ценностью гарантированно достанется Петру, если он наберет в конкурсе k баллов.

Формат входных данных

Первая строка входного файла содержит число n ($2 \leqslant n \leqslant 100\,000$). Вторая строка этого файла содержит n целых чисел: $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant 10^9$).

Формат выходных данных

Выходной файл должен содержать одну строку, содержащую $n - 1$ целых чисел: для каждого k от 2 до n должна быть выведена ценность приза, который достанется Петру, если он наберет k баллов.

Пример

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

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты этой подзадачи успешно пройдены.

Подзадача 1 (24 баллов)

$1 \leqslant n \leqslant 100$.

Подзадача 2 (24 баллов)

$1 \leqslant n \leqslant 5000$.

Подзадача 3 (52 баллов)

$1 \leqslant n \leqslant 100\,000$.

Задача 2. Космическое поселение

Имя входного файла:	`space.in`
Имя выходного файла:	`space.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Для освоения Марса требуется построить исследовательскую базу. База должна состоять из n одинаковых модулей. Каждый модуль представляет собой жилой отсек, который в основании имеет форму прямоугольника размером $a \times b$ метров.

Для повышения надежности модулей инженеры могут добавить вокруг каждого модуля дополнительный защитный слой. Толщина этого слоя должна составлять целое число метров, и все модули должны иметь одинаковую толщину защитного слоя. Модуль с защитным слоем, толщина которой равна d метрам, будет иметь в основании форму прямоугольника размером $(a + 2d) \times (b + 2d)$ метров.

Все модули должны быть расположены на заранее подготовленном прямоугольном поле размером $w \times h$ метров. При этом они

должны быть организованы в виде регулярной сетки: их стороны должны быть параллельны сторонам поля, и модули должны быть ориентированы одинаково.

Требуется написать программу, которая по заданным количеству и размеру модулей, а также размеру поля для их размещения, определяет максимальную толщину дополнительного защитного слоя, который можно добавить к каждому модулю.

Формат входных данных

Входной файл содержит пять разделенных пробелами целых чисел: n, a, b, w и h ($1 \leqslant n, a, b, w, h \leqslant 10^{18}$). Гарантируется, что без дополнительного защитного слоя все модули можно разместить в поселении описанным образом.

Формат выходных данных

Выходной файл должен содержать одно целое число: максимальную возможную толщину дополнительного защитного слоя. Если дополнительный защитный слой установить не удастся, требуется вывести число 0.

Примеры

space.in	space.out
11 2 3 21 25	2
1 5 5 6 6	0

Пояснение к примеру

В первом примере можно установить дополнительный защитный слой толщиной 2 метра и разместить модули на поле, как показано на рисунке.

Во втором примере жилой отсек имеет в основании размер 5×5 метров, а поле — размер 6×6 метров. Добавить дополнительный защитный слой к модулю нельзя.

Система оценивания

Подзадача 1 (26 баллов)

$1 \leqslant n \leqslant 1000$, $1 \leqslant a, b, w, h \leqslant 1000$.

Баллы начисляются, если пройдены все тесты этой подзадачи.

Подзадача 2 (23 баллов)

$1 \leqslant n \leqslant 1000$, $1 \leqslant a, b, w, h \leqslant 10^9$.

Баллы начисляются, если пройдены все тесты этой подзадачи.

Подзадача 3 (24 баллов)

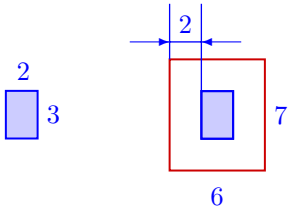
$1 \leqslant n \leqslant 10^9$, $1 \leqslant a, b, w, h \leqslant 10^{18}$.

В этой подзадаче 8 тестов, каждый тест оценивается в 3 балла. Баллы за каждый тест начисляются независимо.

Подзадача 4 (27 баллов)

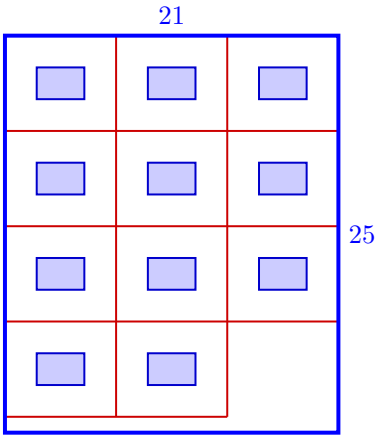
$1 \leqslant n \leqslant 10^{18}$, $1 \leqslant a, b, w, h \leqslant 10^{18}$.

В этой подзадаче 9 тестов, каждый тест оценивается в 3 балла. Баллы за каждый тест начисляются независимо.



Жилой модуль

Жилой модуль с с дополнительной защитой толщиной 2



Размещение модулей на поле

Задача 3. Странные строки

Имя входного файла:	`strange.in`
Имя выходного файла:	`strange.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Рассмотрим строку s, состоящую из строчных букв латинского алфавита. Примером такой строки является, например, строка «`abba`».

Подстрокой строки s называется строка, составленная из одного или нескольких подряд идущих символов строки s. Обозначим через $W(s)$ множество, состоящее из всех возможных подстрок строки s. При этом каждая подстрока входит в это множество не более одного раза, даже если она встречается в строке s несколько раз. Например,

$$W(\text{«abba»}) = \{\text{«a»}, \text{«b»}, \text{«ab»}, \text{«ba»}, \text{«bb»}, \text{«abb»}, \text{«bba»}, \text{«abba»}\}.$$

Подпоследовательностью строки s называется строка, которую можно получить из s удалением произвольного числа символов. Обозначим через $Y(s)$ множество, состоящее из всех возможных подпоследовательностей строки s. Аналогично $W(s)$, каждая подпоследовательность строки s включается в $Y(s)$ ровно один раз, даже если она может быть получена несколькими способами удаления символов из строки s. Поскольку любая подстрока строки s является также ее подпоследовательностью, то множество $Y(s)$ включает в себя $W(s)$, но может содержать также и другие строки. Например,

$$Y(\text{«abba»}) = W(\text{«abba»}) \cup \{\text{«aa»}, \text{«aba»}\}.$$

Знак $\cup$ обозначает объединение множеств.

Будем называть строку s *странной*, если для нее выполняется $W(s) = Y(s)$. Например, строка «`abba`» не является странной, а строка «`abb`» является, так как для нее $W(\text{«abb»}) = Y(\text{«abb»}) = \{\text{«a»}, \text{«b»}, \text{«ab»}, \text{«bb»}, \text{«abb}\}$».

Будем называть *странностью* строки число ее различных странных подстрок. При вычислении странности подстрока считается один раз, даже если она встречается в строке s в качестве

подстроки несколько раз. Так, для строки «abba» ее странность равна 7, любая ее подстрока, кроме всей строки, является странной.

Требуется написать программу, которая по заданной строке s определяет ее странность.

Формат входных данных

Входной файл содержит строку s, состоящую из строчных букв латинского алфавита. Строка имеет длину от 1 до 200 000.

Формат выходных данных

Выходной файл должен содержать одно целое число — странность заданной во входном файле строки.

Пример

strange.in	strange.out
abba	7

Система оценивания

В этой задаче четыре подзадачи. Баллы за каждую подзадачу начисляются только в случае, если все тесты для данной подзадачи успешно пройдены.

Подзадача 1 (29 баллов)

Строка s состоит только из букв «a» и «b». Длина строки s не превышает 50.

Подзадача 2 (12 баллов)

Длина строки s не превышает 50.

Подзадача 3 (25 баллов)

Длина строки s не превышает 1000.

Подзадача 4 (34 баллов)

Длина строки s не превышает 200 000.

Задача 4. Поездка на каникулах

Имя входного файла:	`trains.in`
Имя выходного файла:	`trains.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Железная дорога Флатландии представляет собой прямую, вдоль которой расположены n станций. Будем называть участок железной дороги от некоторой станции до следующей перегоном.

Поезд следует от станции 1 до станции n, делая остановку на каждой станции. В поезде k мест, пронумерованных от 1 до k. На поезд продаются билеты, каждый билет характеризуется тремя числами: s, t и a. Такой билет позволяет проехать от станции s до станции t на месте a.

Иван планирует в один из дней летних каникул проехать на поезде от одной станции до другой. Он выяснил, что на поезд в этот день уже продано m билетов, и возможно уже нет мест, свободных на всех перегонах между интересующими его станциями. Билет от одной станции до другой на определенное место можно купить, только если это место свободно на всех перегонах между этими станциями.

Иван сообразил, что иногда все равно можно проехать от одной станции до другой, купив несколько билетов и пересаживаясь с одного места на другое на некоторых промежуточных станциях. Разумеется, пересаживаться с места на место неудобно, поэтому Иван хочет купить минимальное количество билетов, чтобы на каждом перегоне у него было свое место.

Иван еще не решил, от какой станции и до какой он поедет. Он записал q вариантов поездки, и для каждого из них хочет узнать, какое минимальное число билетов ему придется купить, если он выберет этот вариант.

Требуется написать программу, которая по заданному описанию уже проданных билетов и вариантов поездки Ивана определяет для каждого варианта, какое минимальное количество билетов необходимо купить, чтобы совершить такую поездку.

Формат входных данных

Первая строка входного файла содержит числа n, m и k

($2 \leqslant n \leqslant 200\,000, 0 \leqslant m \leqslant 200\,000, 1 \leqslant k \leqslant 200\,000$) — количество станций, количество уже проданных билетов и количество мест в поезде. Последующие m строк содержат информацию о проданных билетах. Каждая строка содержит три числа: s_i, t_i и a_i — номер станции, от которой куплен билет, номер станции, до которой куплен билет, и номер места, на которое куплен билет ($1 \leqslant s_i < t_i \leqslant n, 1 \leqslant a_i \leqslant k$). Гарантируется, что все билеты куплены таким образом, что ни на каком перегоне ни на какое место нет более одного билета.

Далее идет строка, которая содержит только одно число q ($1 \leqslant q \leqslant 200\,000$). Последующие q строк содержат описания вариантов поездки. Каждая строка содержат два числа: f_j, d_j — номер станции, от которой Иван хочет поехать в этом варианте, и номер станции, до которой он хочет поехать ($1 \leqslant f_j < d_j \leqslant n$).

Формат выходных данных

Выходной файл должен содержать q чисел: для каждого варианта поездки требуется вывести минимальное количество билетов, которое необходимо купить Ивану, чтобы совершить соответствующую поездку. Если поездку совершить невозможно, то для этого варианта требуется вывести -1.

Пример

trains.in	trains.out
5 4 3 1 4 1 2 5 3 2 3 2 4 5 2 3 1 5 3 5 4 5	-1 2 1

Пояснение к примеру

На перегоне от 2-й до 3-й станции все места заняты, поэтому проехать от 1-й до 5-й станции невозможно. От 3-й до 5-й станции

можно проехать, используя два билета: от 3-й до 4-й станции на место 2 и от 4-й до 5-й на место 1. От 4-й до 5-й станции можно проехать, используя один билет на место 1.

Система оценивания

В этой задаче три подзадачи. Баллы за каждую подзадачу начисляются только в случае, если все тесты этой подзадачи успешно пройдены.

Тест из примера не подходит под ограничения для подзадач 1 и 2, но решение принимается на проверку только в том случае, если оно выводит правильный ответ на тесте из примера. Решение должно выводить правильный ответ на тест даже, если оно рассчитано на решение только каких-либо из подзадач 1 и 2.

Подзадача 1 (33 балла)

$n \leqslant 100$, $m \leqslant 100$, $k \leqslant 100$, $q = 1$.

Подзадача 2 (30 баллов)

$n \leqslant 200\,000$, $m \leqslant 200\,000$, $k \leqslant 200\,000$, $q = 1$.

Подзадача 3 (37 баллов)

$n \leqslant 200\,000$, $m \leqslant 200\,000$, $k \leqslant 200\,000$, $q \leqslant 200\,000$.

Задача 5. Три сына

Имя входного файла:	`division.in`
Имя выходного файла:	`division.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Во владениях короля Флатландии находится прямая дорога длиной n километров, по одну сторону от которой расположен огромный лесной массив. Король Флатландии проникся идеями защиты природы и решил превратить свой лесной массив в заповедник. Но сыновья стали сопротивляться: ведь им хотелось получить эти земли в наследство.

У короля три сына: младший, средний и старший. Король решил, что в заповедник не войдут участки лесного массива, кото-

рые он оставит сыновьям в наследство. При составлении завещания король хочет, чтобы для участков выполнялись следующие условия:

- каждый участок должен иметь форму квадрата, длина стороны которого выражается целым положительным числом. Одна из сторон каждого квадрата должна лежать на дороге. Пусть участки имеют размеры $a \times a$, $b \times b$ и $c \times c$;
- стороны квадратов должны полностью покрывать дорогу: величина $a + b + c$ должна быть равна n;
- участок младшего сына должен быть строго меньше участка среднего сына, а участок среднего сына должен, в свою очередь, быть строго меньше участка старшего сына, то есть должно выполняться неравенство $a < b < c$;
- суммарная площадь участков $a^2 + b^2 + c^2$ должна быть минимальна.

Требуется написать программу, которая по заданной длине дороги определяет размеры участков, которые следует выделить сыновьям короля.

Формат входных данных

Входной файл содержит одно целое число n ($6 \leqslant n \leqslant 10^9$).

Формат выходных данных

Выходной файл должен содержать три целых положительных числа, разделенных пробелами: a, b и c — длины сторон участков, которые следует выделить младшему, среднему и старшему сыну, соответственно. Если оптимальных решений несколько, разрешается вывести любое.

Пример

division.in	division.out
6	1 2 3

Пояснение к примеру

Размеры участков, которые следует выделить сыновьям, отмечены на рисунке.

Система оценивания

В этой задаче четыре подзадачи. Баллы за подзадачу начисляются только в случае, если все тесты для данной подзадачи пройдены.

Подзадача 1 (25 баллов)

$n \leqslant 50$.

Подзадача 2 (25 баллов)

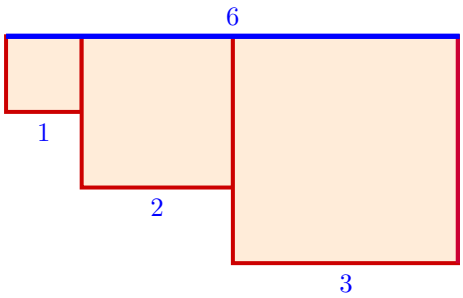
$n \leqslant 2000$.

Подзадача 3 (25 баллов)

$n \leqslant 40\,000$.

Подзадача 4 (25 баллов)

$n \leqslant 10^9$.



Задача 6. Гипершашки

Имя входного файла:	`game.in`
Имя выходного файла:	`game.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Андрей работает судьей на чемпионате по гипершашкам. В каждой игре в гипершашки участвуют три игрока. По ходу игры каждый из игроков набирает некоторое положительное целое число баллов. Если после окончания игры первый игрок набрал

a баллов, второй — b, а третий — c, то говорят, что игра закончилась со счётом $a : b : c$.

Андрей знает, что правила игры гипершашек устроены таким образом, что в результате игры баллы любых двух игроков различаются не более, чем в k раз.

После матча Андрей показывает его результат, размещая три карточки с очками игроков на специальном табло. Для этого у него есть набор из n карточек, на которых написаны числа x_1, x_2, …, x_n. Чтобы выяснить, насколько он готов к чемпионату, Андрей хочет понять, сколько различных вариантов счета он сможет показать на табло, используя имеющиеся карточки.

Требуется написать программу, которая по числу k и значениям чисел на карточках Андрея, определяет количество различных вариантов счёта, которые Андрей может показать на табло.

Формат входных данных

Первая строка входного файла содержит два целых числа n и k ($3 \leqslant n \leqslant 100\,000, 1 \leqslant k \leqslant 10^9$). Вторая строка входного файла содержит n целых чисел x_1, x_2, …, x_n ($1 \leqslant x_i \leqslant 10^9$).

Формат выходных данных

Выходной файл должен содержать одно целое число — искомое количество различных вариантов счёта.

Пример

game.in	game.out
5 2 1 1 2 2 3	9

Пояснение к примеру

В приведенном примере Андрей сможет показать 9 следующих вариантов счёта: 1 : 1 : 2, 1 : 2 : 1, 2 : 1 : 1, 1 : 2 : 2, 2 : 1 : 2, 2 : 2 : 1, 2 : 2 : 3, 2 : 3 : 2, 3 : 2 : 2. Для других троек чисел, которые можно составить с использованием имеющихся карточек, баллы каких-то двух игроков будут различаться более, чем в $k = 2$ раза.

Система оценивания

В этой задаче четыре подзадачи. Баллы за подзадачу начисляются только в случае, если все тесты для данной подзадачи пройдены.

Тест из примера не подходит под ограничения для подзадач 1 и 3, но решение принимается на проверку только в том случае, если оно выводит правильный ответ на тесте из примера. Решение должно выводить правильный ответ на тест, даже если оно рассчитано на решение только каких-либо из подзадач 1 и 3.

Подзадача 1 (15 баллов)

$3 \leqslant n \leqslant 100\,000, k = 1, 1 \leqslant x_i \leqslant 100\,000$.

Подзадача 2 (23 балла)

$3 \leqslant n \leqslant 100, 1 \leqslant k \leqslant 100, 1 \leqslant x_i \leqslant 100$.

Подзадача 3 (30 баллов)

$3 \leqslant n \leqslant 100\,000, 1 \leqslant k \leqslant 10^9, 1 \leqslant x_i \leqslant 10^9$, все x_i различны.

Подзадача 4 (32 балла)

$3 \leqslant n \leqslant 100\,000, 1 \leqslant k \leqslant 10^9, 1 \leqslant x_i \leqslant 10^9$.

Задача 7. Интересные числа

Имя входного файла:	`numbers.in`
Имя выходного файла:	`numbers.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Софья считает число *интересным*, если его цифры идут в неубывающем порядке. Например, числа 123, 1111 или 888999 — интересные.

Софья заинтересовалась, сколько существует интересных положительных чисел, лежащих в диапазоне от L до R включительно. Это число может оказаться довольно большим для больших L и R, поэтому Софья хочет найти остаток от деления этого числа на $10^9 + 7$.

Требуется написать программу, которая по заданным L и R определяет количество интересных чисел, лежащих в диапазоне

от L до R включительно, и выводит остаток от деления этого числа на $10^9 + 7$.

Формат входных данных

Входной файл содержит две строки. Первая строка содержит число L, вторая строка содержит число R ($1 \leqslant L \leqslant R \leqslant 10^{100}$).

Формат выходных данных

Выходной файл должен одно целое число — остаток от деления количества интересных чисел, лежащих в диапазоне от L до R включительно, на $10^9 + 7$.

Пример

numbers.in	numbers.out
1 100	54

Система оценивания

Подзадача 1 (21 балл)

$L = 1$, $1 \leqslant R \leqslant 1000$.

Баллы за подзадачу начисляются только в случае, если все тесты подзадачи пройдены.

Подзадача 2 (до 22 баллов)

$1 \leqslant L \leqslant R \leqslant 10^{18}$.

В этой подзадаче 11 тестов, каждый тест оценивается в 2 балла. Баллы за каждый тест начисляются независимо.

Подзадача 3 (до 24 баллов)

$L = 1$, $R = 10^k$ для некоторого целого k, $2 \leqslant k \leqslant 100$.

В этой подзадаче 8 тестов, каждый тест оценивается в 3 балла. Баллы за каждый тест начисляются независимо.

Подзадача 4 (до 33 баллов)

$1 \leqslant L \leqslant R \leqslant 10^{100}$.

В этой подзадаче 11 тестов, каждый тест оценивается в 3 балла. Баллы за каждый тест начисляются независимо.

Задача 8. Гармоничная последовательность

Имя входного файла:	`sequence.in`
Имя выходного файла:	`sequence.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Цикл лекций в университете Флатландии посвящен изучению последовательностей.

Профессор называет последовательность целых чисел a_1, a_2, $\ldots$, a_n *гармоничной*, если каждое число, кроме a_1 и a_n, равно сумме соседних: $a_2 = a_1 + a_3$, $a_3 = a_2 + a_4$, $\ldots$, $a_{n-1} = a_{n-2} + a_n$. Например, последовательность $[\,1, 2, 1, -1\,]$ является гармоничной, поскольку $2 = 1 + 1$ и $1 = 2 + (-1)$.

Рассмотрим две последовательности равной длины: $A = [\,a_1, a_2, \ldots, a_n\,]$ и $B = [\,b_1, b_2, \ldots, b_n\,]$. Расстоянием между этими последовательностями будем называть величину

$$d(A, B) = |\,a_1 - b_1\,| + |\,a_2 - b_2\,| + \ldots + |\,a_n - b_n\,|.$$

Например, $d([\,1, 2, 1, -1\,], [\,1, 2, 0, 0\,]) = |\,1 - 1\,| + |\,2 - 2\,| + |\,1 - 0\,| + |\,-1 - 0\,| = 0 + 0 + 1 + 1 = 2$. В конце лекции профессор написал на доске последовательность из n целых чисел $B = [\,b_1, b_2, \ldots, b_n\,]$ и попросил студентов в качестве домашнего задания найти гармоничную последовательность $A = [\,a_1, a_2, \ldots, a_n\,]$ такую, что $d(A, B)$ минимально. Чтобы облегчить себе проверку, профессор просит написать в качестве ответа только искомое минимальное расстояние $d(A, B)$.

Требуется написать программу, которая по заданной последовательности B определяет, на каком минимальном расстоянии от последовательности B найдется гармоничная последовательность A.

Формат входных данных

Первая строка входного файла содержит целое число n — количество элементов в последовательности ($3 \leqslant n \leqslant 300\,000$). Вторая строка содержит n целых чисел $b_1, b_2, \ldots, b_n$ ($-10^9 \leqslant b_i \leqslant 10^9$).

Формат выходных данных

Выходной файл должен содержать одно целое число — мини-

мальное возможное расстояние от последовательности во входном файле до гармоничной последовательности.

Пример

sequence.in	sequence.out
4 1 2 0 0	2

Пояснение к примеру

В приведенном примере оптимальной является, например, гармоничная последовательность $[1, 2, 1, -1]$.

Система оценивания

В этой задаче пять подзадач. Баллы за подзадачу начисляются только в случае, если все тесты для данной подзадачи пройдены.

Тест из примера не подходит под ограничения для подзадачи 1, но решение принимается на проверку только в том случае, если оно выводит правильный ответ на тесте из примера. Решение должно выводить правильный ответ на тест, даже если оно рассчитано на решение только подзадачи 1.

Подзадача 1 (14 баллов)

$n = 3$, $-10 \leqslant b_i \leqslant 10$.

Подзадача 2 (14 баллов)

$3 \leqslant n \leqslant 500$, $-100 \leqslant b_i \leqslant 100$.

Подзадача 3 (16 баллов)

$3 \leqslant n \leqslant 100\,000$, $-100 \leqslant b_i \leqslant 100$.

Подзадача 4 (16 баллов)

$3 \leqslant n \leqslant 1000$, $-10^9 \leqslant b_i \leqslant 10^9$.

Подзадача 5 (40 баллов)

$3 \leqslant n \leqslant 300\,000$, $-10^9 \leqslant b_i \leqslant 10^9$.

Задача 9. История одного города (7-8 классы)

Имя входного файла:	`levels.in`
Имя выходного файла:	`levels.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

... тайна построения градоначальнического организма наукой достаточно еще не обследована...
(Салтыков-Щедрин М. «История города Глупова»)

Известный градоначальник города Глупова Василиск Семёнович Бородавкин отличался «неслыханной административной въедчивостью, которая с особенной энергией проявлялась в вопросах, касавшихся выеденного яйца». В свободное время Бородавкин придумывал и проводил в жизнь исключительно полезные прожекты, направленные на благо любимого города. Объехав и лично осмотрев весь город, градоначальник произвёл подсчёт числа домов a_k, у которых было более k этажей, — для каждого значения $k = 0, 1, 2, \ldots$. (Так, если бы в Глупове было всего 4 дома с числом этажей 5, 3, 3, 2 соответственно, то $a_0 = a_1 = 4, a_2 = 3, a_3 = 1$ и $a_4 = 1$.) Полученная последовательность из m положительных чисел $a_0, a_1, \ldots, a_{m-1}$ была отослана им в столицу для получения дальнейших инструкций по управлению городом Глуповым.

Вам необходимо составить программу, которая восстанавливает количество этажей в каждом доме города по известным значениям $a_0, a_1, \ldots, a_{m-1}$.

Формат входных данных

В первой строке записано одно число m — количество положительных чисел в последовательности (a_k) $(1 \leqslant m \leqslant 10^6)$. Вторая строка содержит невозрастающую последовательность из m разделенных пробелом целых чисел $a_0, a_1, \ldots, a_{m-1}$ $(1 \leqslant a_i \leqslant 10^6)$.

Формат выходных данных

Выходные данные должны содержать две строки.

В первой строке запишите одно число n — количество домов в городе Глупове, во второй строке n целых положительных чисел в порядке невозрастания — количество этажей в домах города.

Пример

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

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты этой подзадачи успешно пройдены.

Подзадача 1 (50 баллов)

$1 \leqslant m \leqslant 40\,000$.

Подзадача 2 (50 баллов)

$1 \leqslant m \leqslant 10^6$.

Задача 10. Кофе (7-8 классы)

Имя входного файла:	`coffee.in`
Имя выходного файла:	`coffee.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

— Чем хороший кофе отличается от лучшего?
— Рекламой.
(У. Сароян «Тигр Тома Трейси».)

Знаете ли вы, что благодаря смешиванию различных сортов кофе можно создать свой собственный и неповторимый вкус? Чаще всего это кофейные смеси из одного или двух сортов кофе с разных континентов и разного способа обработки. Даже хранят кофе различных сортов по-особому.

Итак, у вас есть $m \cdot n$ пачек кофе m различных сортов, которые вы собираетесь упаковать в ящики по n пачек в каждом. Для сохранения аромата и вкусовых качеств рекомендуется хранить в одном ящике *не более двух* сортов кофе.

Требуется составить программу упаковки кофе по ящикам так, чтобы в каждом ящике было не более двух сортов кофе.

Формат входных данных

В первой строке записаны два целых числа m и n ($1 \leqslant m \leqslant 200\,000$, $1 \leqslant n \leqslant 10\,000$). Во второй строке записаны разделенные пробелом m целых положительных чисел $k_1, k_2, \ldots, k_m$ — количества пачек кофе каждого сорта. Суммарное число пачек всех сортов $m \cdot n$.

Формат выходных данных

Выведите последовательность из m строк, каждая из которых содержит описание упаковки соответствующего ящика. В i-ой строке запишите сначала целое число s — количество сортов кофе в i-ом ящике ($1 \leqslant s \leqslant 2$), затем s пар целых положительных чисел s_i и n_i, где s_i — номер сорта кофе, n_i — количество пачек кофе сорта s_i, которыми заполнен i-ый ящик ($1 \leqslant s_i \leqslant m$). Количество пачек кофе в каждом ящике n. Если возможных упаковок несколько, выведите любую из них.

Пример

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

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты этой подзадачи успешно пройдены.

Подзадача 1 (50 баллов)

$1 \leqslant m \leqslant 40\,000$.

Подзадача 2 (50 баллов)

$1 \leqslant m \leqslant 200\,000$.

2016-2017 учебный год

Муниципальный этап 28-й Всероссийской олимпиады по информатике среди школьников Республики Татарстан состоялся 5 декабря 2016 г. В составлении задач муниципальной олимпиады принимали участие преподаватели Казанского Федерального Университета:

М. И. Киндер, А. А. Попов, Р. Р. Гайфуллин.

Региональный этап олимпиады проходил с 4 февраля по 6 февраля 2017 г. Кроме школьников 9-11 классов — традиционных участников регионального этапа — на олимпиаду были приглашены также ученики 7-8 классов, показавшие хорошие результаты на муниципальном этапе Всероссийской олимпиады. Специально для школьников 7-8 классов жюри составило несколько задач, сложность которых, по мнению жюри, соответствовала этой возрастной категории участников олимпиады.

Материалы муниципального и регионального этапа Всероссийской олимпиады по информатике среди школьников Республики Татарстан (результаты участников, архив жюри с тестами, генераторами тестов и решениями жюри) доступны в интернете по адресу:

http://kpfu.ru/math/olimpiady-dlya-shkolnikov-i-studentov/olimpiady-shkolnikov-po-informatike

Ниже представлены условия и подробные решения задач всех перечисленных олимпиад. Для каждой задачи приведена идея решения, система оценки и описание конкретной реализации на языке Pascal или `C++`. Для большинства олимпиадных заданий указаны фамилии авторов идеи и фамилии тех, кто готовил эти решения.

Муниципальный этап, 2016-2017

Задача 1. Билеты (7-11 классы)

Имя входного файла:	`tickets.in`
Имя выходного файла:	`tickets.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Группа студентов и школьников собирается в музей. Для школьников до 10 лет (включительно) вход в музей бесплатный, а для остальных школьников и студентов до 18 лет включительно стоимость билета составляет половину от полной стоимости билета. Студенты старше 18 лет покупают билет за полную стоимость — за 100 рублей.

Вам необходимо подсчитать, сколько рублей стоят билеты на всю группу.

Формат входных данных

В первой строке записано целое n — количество студентов и школьников в группе ($1 \leqslant n \leqslant 10^6$). Во второй строке записаны n целых чисел, каждое из которых не меньше 7 и не больше 25, — возрасты студентов и школьников.

Формат выходных данных

Выведите одно целое число — суммарную стоимость билетов на всю группу.

Система оценивания

Задача оценивается в 100 баллов. Баллы начисляются за каждый пройденный тест.

Примеры

tickets.in	tickets.out
2 9 10	0
3 10 15 20	150

Задача 2. Палиндром (7-8 классы)

Имя входного файла:	palindrome.in
Имя выходного файла:	palindrome.out
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Наэкал кабан на баклажан.

Sum summus mus.
(*Я — сильнейшая мышь.*)

Палиндромом будем называть число, запись которого в десятичной системе счисления одинаково читается слева направо и справа налево. Например, числа 121 и 1331 являются палиндромами, а число 330 — нет. (Напомним, что запись числа не может начинаться с нуля). Палиндромы встречаются не так часто, но иногда их можно сконструировать, переставляя цифры какого-нибудь числа. В частности, переставляя цифры в числе 330, можно получить палиндром 303.

Вам необходимо выяснить, можно ли из из всех цифр данного числа составить палиндром.

Формат входных данных

В первой строке одно целое n — количество цифр в данном числе ($1 \leqslant n \leqslant 10^6$). Во второй строке записано число из n десятичных цифр, первая цифра которого отлична от нуля.

Формат выходных данных

Выведите `yes`, если из числа можно образовать n-значный палиндром. Иначе — выведите `no`.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	30	$1 \leqslant n \leqslant 9$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$1 \leqslant n \leqslant 255$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$1 \leqslant n \leqslant 10^6$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

palindrome.in	palindrome.out
3 330	yes
1 7	yes
3 120	no

Задача 3. Сумма факториалов (7-11 классы)

Имя входного файла:	`factorial.in`
Имя выходного файла:	`factorial.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Как известно, факториалом целого положительного числа n называется произведение всех натуральных чисел от 1 до n включительно: $n! = 1 \cdot 2 \cdot \ldots \cdot n$. Любое целое положительное число $n > 1$ можно представить в виде суммы факториалов несколькими способами. Например, $3 = 1! + 1! + 1! = 2! + 1!$.

Ваша задача — для данного натурального числа n найти разложение в сумму факториалов с *наименьшим* числом слагаемых. (Например, для числа $n = 10$ такая сумма состоит из трёх факториалов: $3! + 2! + 2!$.)

Формат входных данных

Входные данные содержат одно целое число n ($1 \leqslant n \leqslant 10^{18}$).

Формат выходных данных

Выведите одно целое число — наименьшее количество слагаемых-факториалов в разложении данного числа n.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	50	$1 \leqslant n \leqslant 10^9$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	50	$1 \leqslant n \leqslant 10^{18}$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Примеры

factorial.in	factorial.out
8	2
10	3

Задача 4. НОК (7-11 классы)

Имя входного файла:	`lcm.in`
Имя выходного файла:	`lcm.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Двоечник Петька был ленивым и часто прогуливал уроки. Теплым майским днём решил Петька вместо урока математики сходить на речку, через лес. Но не смог он добраться до речки — встретилась ему на пути Баба-Яга. Решила она проучить Петьку и не отпускать домой, пока не решит задачу по математике. По той самой теме, которую он прогулял. А задача была такая. Назвала Баба-Яга два натуральных числа m и k. И нужно было Петьке найти количество упорядоченных наборов из k натуральных чисел, у которых наименьшее общее кратное равно данному числу m. Например, для $m = 10$ и $k = 2$ существует 9 наборов из двух целых положительных чисел, у которых наименьшее общее кратное равно 10:

(1; 10), (10; 1), (2; 10), (10; 2), (5; 10), (10; 5), (10; 10), (2; 5), (5; 2).

И теперь вам нужно написать программу, которая позволит дать ответ на задачу и поможет Петьке вернуться домой.

Формат входных данных

Входные данные содержат два целых числа m и k — наименьшее общее кратное и количество чисел в наборах ($1 \leqslant m \leqslant 10^9$, $2 \leqslant k \leqslant 10^{18}$).

Формат выходных данных

Выведите искомое количество наборов по модулю $(10^9 + 9)$.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		m, k	
1	30	$1 \leqslant m \leqslant 100$, $k = 2, 3$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$1 \leqslant m \leqslant 10^9$, $2 \leqslant k \leqslant 10^5$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$1 \leqslant m \leqslant 10^9$, $2 \leqslant k \leqslant 10^{18}$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

lcm.in	lcm.out
10 2	9
10 3	49

Задача 5. Наименьший палиндром (9-11 классы)

Имя входного файла:	`palindrome.in`
Имя выходного файла:	`palindrome.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Нажал кабан на баклажан.

Sum summus mus.
(Я — сильнейшая мышь.)

Палиндромом будем называть число, запись которого в десятичной системе счисления одинаково читается слева направо и справа налево. Например, числа 131 и 2112 являются палиндромами, а число 2211 — нет. (Напомним, что запись числа не может начинаться с нуля). Палиндромы встречаются не так часто, но иногда их можно сконструировать, переставляя цифры какого-нибудь числа. В частности, переставив цифры в числе 2211, можно получить два палиндрома 1221 и 2112.

Вам необходимо из всех цифр данного числа составить *наименьшее* число-палиндром.

Формат входных данных

В первой строке одно целое n — количество цифр в данном числе ($1 \leqslant n \leqslant 10^6$). Во второй строке записано число из n десятичных цифр, первая цифра которого отлична от нуля.

Формат выходных данных

Выведите -1, если из числа нельзя образовать n-значный палиндром. Иначе выведите наименьшее число-палиндром, которое можно составить из всех цифр данного числа.

Система оценивания

Задача оценивается в 100 баллов. Баллы начисляются за каждый пройденный тест.

Примеры

palindrome.in	palindrome.out
4 2211	1221
1 7	7
3 120	-1

Задача 6. Заводы в городе (9-11 классы)

Имя входного файла:	`distance.in`
Имя выходного файла:	`distance.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

В 2014 году открылась первая в мире велодорожка из солнечных батарей.

Город Солнечный состоит из k районов, в каждом из которых n домов. Планировка города такая, что все n домов в каждом районе расположены в вершинах выпуклого n-угольника $A_1A_2\dots A_n$. Городу требуется электроэнергия, поэтому в каждом из k районов планируется построить свой завод по производству солнечных батарей. Партия местных экологов требует установить каждое предприятие внутри или на границе района так, чтобы сумма расстояний до всех домов района была бы *наибольшей*. В случае необходимости можно построить завод на месте одного из домов района. (Разумеется, переселив для этого всех жильцов этого дома.)

Ваша задача — определить требуемое расположение всех k заводов города.

Формат входных данных

В первых строке записано два целых числа: k — количество районов ($1 \leqslant k \leqslant 100$) и n — количество домов в районах города ($3 \leqslant n \leqslant 500$). Число домов n одно и то же для всех районов. В каждой из следующих k строк записаны через пробел $2n$ целых чисел — координаты вершин A_1, A_2, …, A_n выпуклого многоугольника $A_1A_2\dots A_n$, заданные в порядке его обхода по часовой стрелке. Координаты всех вершин — целые числа, не превосходящие по модулю 10^4. Взаимное расположение районов не имеет значения.

Формат выходных данных

Выходные данные должны содержать k строк. В i-ой строке сначала запишите два числа — координаты искомой точки, для

которой сумма расстояний до всех домов i-го района наибольшая, затем ещё одно число — значение этой суммы. Ответ считается правильным, если абсолютная или относительная погрешность не превышает 10^{-5}. Если искомых точек несколько, выведите координаты любой из них.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		m, k	
1	25	$1 \leqslant k \leqslant 100$, $n = 3$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	25	$1 \leqslant k \leqslant 100$, $3 \leqslant n \leqslant 20$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	25	$1 \leqslant k \leqslant 100$, $3 \leqslant n \leqslant 300$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.
4	25	$1 \leqslant m \leqslant 10^9$, $3 \leqslant n \leqslant 500$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Пример

distance.in	distance.out
1 3 0 0 0 3 4 0	4.00 0.00 9.000000

Региональный этап, 2016-2017

Задача 1. Кампус

Имя входного файла:	`building.in`
Имя выходного файла:	`building.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Новое здание кампуса Университета Байтбурга имеет n этажей, пронумерованных снизу вверх от 1 до n. Комнаты студентов расположены в нескольких подъездах.

В каждом подъезде на этажах, номер которых кратен числу k, расположено по x комнат, а на остальных этажах расположено по y комнат.

Комнаты внутри каждого подъезда пронумерованы последовательными натуральными числами. Номера комнат на первом этаже имеют наименьшие значения в этом подъезде, затем следуют номера комнат на втором этаже, и так далее. Комнаты в первом подъезде пронумерованы, начиная с 1, в каждом следующем подъезде нумерация комнат начинается с числа, следующего после максимального номера комнаты в предыдущем подъезде.

На рисунке 1 показаны номера комнат в здании с $n = 7$ этажами, 3 подъездами, и параметрами $k = 3, x = 2, y = 3$.

	Подъезд 1	Подъезд 2	Подъезд 3
7 этаж	17, 18, 19	36, 37, 38	55, 56, 57
6 этаж	15, 16	34, 35	53, 54
5 этаж	12, 13, 14	31, 32, 33	50, 51, 52
4 этаж	9, 10, 11	28, 29, 30	47, 48, 49
3 этаж	7, 8	26, 27	45, 46
2 этаж	4, 5, 6	23, 24, 25	42, 43, 44
1 этаж	1, 2, 3	20, 21, 22	39, 40, 41

Рис. 1. Пример нумерации комнат в здании

Для организации расселения студентов администрация кампуса должна по номеру комнаты оперативно определять этаж, на котором она находится.

Требуется написать программу, которая по заданным числам n, k, x и y, а также по номерам комнат, определяет для каждой комнаты, на каком этаже она находится.

Формат входных данных

Первая строка входного файла содержит натуральные числа n, k, x и y ($1 \leqslant n \leqslant 10^9$, $1 \leqslant k \leqslant n$, $1 \leqslant x, y \leqslant 10^9$). Соседние числа разделены ровно одним пробелом.

Вторая строка входного файла содержит натуральное число q — количество номеров комнат, для которых требуется определить этаж ($1 \leqslant q \leqslant 1000$).

Третья строка содержит q целых чисел $a_1, a_2, \ldots, a_q$ — номера комнат ($1 \leqslant a_i \leqslant 10^{18}$). Можно считать, что в здании так много подъездов, что все комнаты с заданными номерами существуют.

Формат выходных данных

Требуется вывести q чисел в одну строку. Для каждого номера комнаты во входном файле требуется вывести номер этажа, на котором она находится.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		n, x, y	q, a_i	
1	31	$1 \leqslant n \leqslant 10$, $1 \leqslant x, y \leqslant 10$	$q = 1$, $1 \leqslant a_i \leqslant 100$	
2	19	$1 \leqslant n \leqslant 10^7$, $1 \leqslant x, y \leqslant 10^9$	$q = 1$, $1 \leqslant a_i \leqslant 10^7$	**1**
3	16	$1 \leqslant n \leqslant 10^9$, $1 \leqslant x, y \leqslant 10^9$ $x = y$	$1 \leqslant q \leqslant 1000$, $1 \leqslant a_i \leqslant 10^{18}$	
4	34	$1 \leqslant n \leqslant 10^9$, $1 \leqslant x, y \leqslant 10^9$	$1 \leqslant q \leqslant 1000$, $1 \leqslant a_i \leqslant 10^{18}$	**1, 2, 3**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

building.in	building.out
7 3 2 3 4 1 19 20 50	1 7 1 5

Задача 2. Калькулятор

Имя входного файла:	`calc.in`
Имя выходного файла:	`calc.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

В качестве домашнего задания по информатике ученикам предложено разработать специальный калькулятор, который устроен следующим образом.

Сначала пользователь вводит целое положительное число n, которое выводится на экран. Затем пользователь может нажимать на три кнопки: **A**, **B** и **C**.

При нажатии на кнопку **A** число, которое выведено на экран, делится на 2. Если число на экране нечётное, то остаток отбрасывается. Например, результат этой операции для числа 80 равен 40, а для числа 239 равен 119.

При нажатии на кнопку **B** к числу, которое выведено на экран, прибавляется 1 и результат делится на 2. Остаток от деления отбрасывается. Например, результат операции для числа 80 равен 40, а для числа 239 равен 120.

При нажатии на кнопку **C** происходит следующее. Если число, которое выведено на экран, положительное, то из него вычитается 1 и результат делится на 2, остаток отбрасывается. Если же перед нажатием на кнопку **C** на экран было выведено число 0, то оно остается неизменным. Например, результат операции для числа 80 равен 39, а для числа 239 равен 119.

Пользователь ввёл число n и собирается нажать на кнопки

операций в некотором порядке. В частности, он планирует нажать на кнопку **A** суммарно a раз, на кнопку **B** — b раз и на кнопку **C** — c раз. Его заинтересовал вопрос, какое минимальное число может получиться в результате выполнения описанных операций.

Требуется написать программу, которая по введённому числу n и числам a, b и c, показывающим количество произведённых на калькуляторе операций разного типа, определяет минимальное число, которое может получиться в результате работы калькулятора.

Формат входных данных

Входной файл содержит разделённые пробелом четыре целых числа n, a, b и c ($1 \leqslant n \leqslant 10^{18}, 0 \leqslant a, b, c \leqslant 60$).

Формат выходных данных

Требуется вывести одно число — минимальное число, которое может получиться у пользователя в результате работы калькулятора.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		n	a, b, c	
1	26	$1 \leqslant n \leqslant 10^9$	$0 \leqslant a+b+c \leqslant 7$	
2	23	$1 \leqslant n \leqslant 10^{18}$	$c = 0$	
3	24	$1 \leqslant n \leqslant 10^{18}$	$b = 0$	
4	27	$1 \leqslant n \leqslant 10^{18}$		**1, 2, 3**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

calc.in	calc.out
72 2 1 1	4

Пояснение к примеру

В примере пользователю необходимо оптимально действовать следующим образом: нажать на кнопку **B** и получить число 36, затем нажать на кнопку **A** и получить число 18, затем нажать на кнопку **C** и получить число 8, затем второй раз нажать на кнопку **A** и получить число 4.

Задача 3. Размещение данных

Имя входного файла:	`data.in`
Имя выходного файла:	`data.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Телекоммуникационная сеть крупной IT-компании содержит n серверов, пронумерованных от 1 до n. Некоторые пары серверов соединены двусторонними каналами связи, всего в сети m каналов. Известно, что по каналам связи можно передавать данные с любого сервера на любой другой сервер, возможно с использованием одного или нескольких промежуточных серверов.

Множество серверов A называется *отказоустойчивым*, если при недоступности любого канала связи выполнено следующее условие. Для любого не входящего в это множество сервера X существует способ передать данные по остальным каналам на сервер X хотя бы от одного сервера из множества A.

На рисунке 2 показан пример сети и отказоустойчивого множества из серверов с номерами 1 и 4. Данные на сервер 2 можно передать следующим образом. При недоступности канала между серверами 1 и 2 — с сервера 4, при недоступности канала между серверами 2 и 3 — с сервера 1. На серверы 3 и 5 при недоступности любого канала связи можно по другим каналам передать данные с сервера 4.

В рамках проекта группе разработчиков компании необходимо разместить свои данные в сети. Для повышения доступности данных и устойчивости к авариям разработчики хотят продублировать свои данные, разместив их одновременно на нескольких серверах, образующих отказоустойчивое множество. Чтобы

минимизировать издержки, необходимо выбрать минимальное по количеству серверов отказоустойчивое множество. Кроме того, чтобы узнать, насколько гибко устроена сеть, необходимо подсчитать количество способов выбора такого множества, и поскольку это количество способов может быть большим, необходимо найти остаток от деления этого количества способов на число $10^9 + 7$.

Требуется написать программу, которая по заданному описанию сети определяет следующие числа: k — минимальное количество серверов в отказоустойчивом множестве серверов, c — остаток от деления количества способов выбора отказоустойчивого множества из k серверов на число $10^9 + 7$.

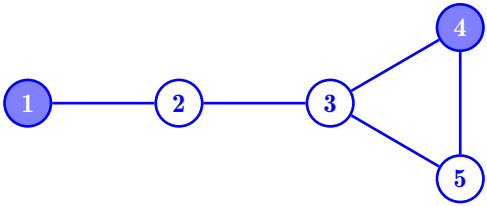


Рис. 2. Пример сети и отказоустойчивого множества серверов

Формат входных данных

Первая строка входного файла содержит целые числа n и m — количество серверов и количество каналов связи соответственно ($2 \leqslant n \leqslant 200\,000, 1 \leqslant m \leqslant 200\,000$).

Следующие m строк содержат по два целых числа и описывают каналы связи между серверами. Каждый канал связи задается двумя целыми числами — номерами серверов, которые он соединяет.

Гарантируется, что любые два сервера соединены напрямую не более чем одним каналом связи, никакой канал не соединяет сервер сам с собой, и для любой пары серверов существует способ передачи данных с одного из них на другой, возможно с использованием одного или нескольких промежуточных серверов.

Формат выходных данных

Выведите два целых числа, разделенных пробелом: k — минимальное число серверов в отказоустойчивом множестве серверов,

c — количество способов выбора отказоустойчивого множества из k серверов, вычисленное по модулю $10^9 + 7$.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		n	m	
1	25	$2 \leqslant n \leqslant 10$	$1 \leqslant m \leqslant 45$	
2	27	$2 \leqslant n \leqslant 200\,000$	$m = n - 1$	
3	28	$2 \leqslant n \leqslant 1000$	$1 \leqslant m \leqslant 5000$	**1**
4	21	$2 \leqslant n \leqslant 200\,000$	$1 \leqslant m \leqslant 200\,000$	**1, 2, 3**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

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

Пояснение к примеру

В приведенном примере отказоустойчивыми являются следующие множества из двух серверов: $\{1, 3\}$, $\{1, 4\}$, $\{1, 5\}$.

Задача 4. Полезные ископаемые

Имя входного файла:	`mining.in`
Имя выходного файла:	`mining.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Ведётся проект по освоению планеты соседней звёздной системы. Для добычи полезных ископаемых планируется направить на планету несколько партий роботов.

Участок поверхности планеты, на котором планируется добывать полезные ископаемые, представляет собой клетчатый прямоугольник размером $w \times h$, клетки участка имеют координаты от $(1, 1)$ до (w, h). В некоторых клетках участка находятся базы специалистов, в которые могут быть доставлены партии роботов. Всего на участке размещено s баз, и i-я база находится в клетке с координатами (x_i, y_i).

Каждая партия роботов характеризуется тремя параметрами: j-я партия доставляется на базу b_j, содержит n_j роботов и каждый робот партии обладает мобильностью m_j.

Когда партия роботов доставляется на соответствующую базу, каждый робот этой партии перемещается по поверхности планеты от базы до некоторой клетки. Если мобильность робота равна m, он может не более m раз переместиться на одну из восьми соседних клеток, как показано на рисунке 3.

Требуется написать программу, которая по заданному описанию сети определяет следующие числа: k — минимальное количество серверов в отказоустойчивом множестве серверов, c — остаток от деления количества способов выбора отказоустойчивого множества из k серверов на число $10^9 + 7$.

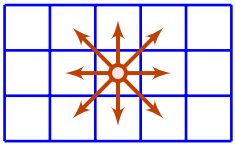


Рис. 3. Возможные перемещения робота в восьми направлениях

После того как роботы из всех доставленных партий размещаются на участке, они активируются и начинают добычу полезных ископаемых. В процессе перемещения в одной клетке может одновременно находиться произвольное количество роботов. Однако после активации в каждой клетке должно находиться не более q роботов.

Руководством проекта получена информация о t партиях роботов, которые могут быть последовательно отправлены на пла-

нету. После доставки всех партий роботов, учитывая их ограниченную мобильность, возможна ситуация, что не удастся разместить роботов на участке так, чтобы в каждой клетке оказалось не больше q роботов. Поэтому руководство должно выбрать k первых партий роботов, где $0 \leqslant k \leqslant t$, которые будут полностью доставлены на соответствующие базы. После этого, если $k < t$, следует дополнительно принять z из n_{k+1} роботов следующей, $(k+1)$-й партии, $0 \leqslant z < n_{k+1}$.

Все полученные таким образом роботы должны с учетом ограничения на мобильность разместиться на участке таким образом, чтобы в каждой клетке было не более q роботов. После этого они будут активированы и начнут добычу полезных ископаемых. Разумеется, руководство проекта старается максимизировать количество роботов, которые будут доставлены на планету, поэтому с учетом описанных ограничений требуется максимизировать k, а затем максимизировать z.

Требуется написать программу, которая по размерам участка, числу q, описанию расположения баз, а также количеству запланированных партий роботов и их описанию определяет максимальное число k — количество партий роботов, максимальное число z — дополнительное количество роботов из $(k+1)$-й партии, чтобы, доставив роботов на планету, их можно было разместить на участке таким образом, чтобы в каждой клетке оказалось не более q роботов.

Формат входных данных

Первая строка входного файла содержит числа w, h, s и q $(1 \leqslant w, h \leqslant 10^5, 1 \leqslant s \leqslant 4, 1 \leqslant q \leqslant 100)$. Последующие s строк содержат по два целых числа x_i, y_i и описывают базы специалистов $(1 \leqslant x_i \leqslant w,\ 1 \leqslant y_i \leqslant h)$.

Следующая строка содержит число t — количество партий роботов $(1 \leqslant t \leqslant 100)$. Последующие t строк описывают партии роботов и содержат по три целых числа: b_j, n_j и m_j $(1 \leqslant b_j \leqslant s,$ $1 \leqslant n_j \leqslant w \cdot h \cdot q,\ 0 \leqslant m_j < \max(w, h))$.

Формат выходных данных

Требуется вывести два числа k и z $(0 \leqslant k \leqslant t)$. Если $k = t$, то z должно быть равно 0, иначе должно выполняться условие $0 \leqslant z < n_{k+1}$.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		w, h	s, q	
1	18	$1 \leqslant w, h \leqslant 20$	$s = 1$, $q = 1$	
2	12	$1 \leqslant w, h \leqslant 20$	$1 \leqslant s \leqslant 2$, $q = 1$	**1**
3	9	$1 \leqslant w, h \leqslant 20$	$1 \leqslant s \leqslant 3$, $q = 1$	**1, 2**
4	10	$1 \leqslant w, h \leqslant 20$	$1 \leqslant s \leqslant 3$, $1 \leqslant q \leqslant 100$	**1, 2, 3**
5	15	$1 \leqslant w, h \leqslant 10^5$	$s = 1$, $1 \leqslant q \leqslant 100$	**1**
6	36	$1 \leqslant w, h \leqslant 10^5$	$1 \leqslant s \leqslant 4$, $1 \leqslant q \leqslant 100$	**1, 2, 3, 4, 5**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

mining.in	mining.out
4 3 2 1 1 1 3 2 3 1 4 1 2 9 1 1 12 2	1 7

Пояснение к примеру

В приведённом примере описания входных данных следует полностью принять первую партию роботов и дополнительно при-

нять 7 роботов из второй партии. На рисунке 4 показано, как можно разместить этих роботов на участке, чтобы в каждой клетке было не более одного робота. Базы специалистов показаны кружками. Клетки, в которых окажутся роботы с базы 1, показаны диагональной штриховкой, а клетки, в которых окажутся роботы с базы 2, показаны серым цветом.

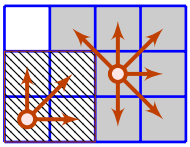


Рис. 4. Возможное размещение роботов на участке в примере

Задача 5. Автоматизированное управление доставкой

Имя входного файла:	`delivery.in`
Имя выходного файла:	`delivery.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Группа программистов регионального сортировочного центра работает над автоматизацией управления доставкой почты.

Посылки принимаются в клиентских почтовых пунктах. Почтовый пункт принимает посылки, вес каждой из которых составляет целое число килограммов. Минимальный вес посылки равен 1 *кг*, а максимальный вес — k *кг*. Принятые посылки помещаются в специальный пакет.

Если после приёма очередной посылки суммарный вес посылок в пакете больше или равен x *кг*, то пакет доставляется в муниципальный почтовый центр, где пакет с посылками перемещается в специальный контейнер.

Если после доставки очередного пакета суммарный вес посылок в контейнере больше или равен y *кг*, то контейнер перевозится в региональный сортировочный центр, откуда посылки уже доставляются получателям.

Суммарный вес посылок в контейнере при его перевозке может различаться в зависимости от массы принятых посылок. Необходимо выяснить, каким может быть минимальный суммарный вес посылок в контейнере при перевозке его из муниципального почтового центра в региональный сортировочный центр.

Требуется написать программу, которая по заданным значениям k — максимального веса посылки, x — необходимого веса пакета для его отправки в муниципальный почтовый центр и y — необходимого веса контейнера для его отправки в региональный сортировочный центр определяет минимальный вес контейнера при его перевозке.

Формат входных данных

Входной файл содержит три целых положительных числа, по одному на строке. Первая строка содержит число k ($1 \leqslant k \leqslant 10^9$). Вторая и третья строки содержат числа x и y соответственно ($1 \leqslant x, y \leqslant 10^9$).

Формат выходных данных

Требуется вывести одно целое число — минимальный возможный вес контейнера при перевозке.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		k	x, y	
1	21	$k = 1$	$1 \leqslant x, y \leqslant 100$	
2	18	$k = 2$	$1 \leqslant x, y \leqslant 100$	
3	21	$1 \leqslant k \leqslant 100$	$1 \leqslant x, y \leqslant 100$	**1, 2**
4	17	$1 \leqslant k \leqslant 40\,000$	$1 \leqslant x, y \leqslant 40\,000$	**1, 2, 3**
5	23	$1 \leqslant k \leqslant 10^9$	$1 \leqslant x, y \leqslant 10^9$	**1, 2, 3, 4**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

delivery.in	delivery.out
2 7 20	21

Пояснение к примеру

В приведённом примере принимаются посылки весом 1 *кг* и 2 *кг*. При накоплении посылок с суммарным весом хотя бы в 7 *кг* пакет доставляется из клиентского почтового пункта в муниципальный почтовый центр. При накоплении посылок с суммарным весом хотя бы в 20 *кг* контейнер перевозится из муниципального почтового центра в региональный сортировочный центр.

Минимальный возможный вес контейнера в данном примере составляет 21 *кг* и достигается, например, следующим образом: в муниципальный почтовый центр последовательно доставляется 3 пакета по 7 *кг* каждый. Пакет весом 7 *кг* может получиться, например, после приёма семи посылок по 1 *кг*.

Задача 6. Большой линейный коллайдер

Имя входного файла:	`linear.in`
Имя выходного файла:	`linear.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Группа учёных работает в международной научной лаборатории, которая занимается исследованием поведения элементарных частиц в экспериментальной установке «Большой линейный коллайдер» (БЛК). Установка БЛК представляет собой прямую, на которой расположено несколько частиц и которые могут перемещаться вдоль этой прямой.

В очередном эксперименте размещены n частиц, каждая из которых представляет собой либо отрицательно заряженную частицу — электрон e^-, либо положительно заряженную частицу — позитрон e^+. В эксперименте i-я частица исходно размещается в

точке с координатой x_i. После начала эксперимента в результате работы БЛК частицы перемещаются вдоль прямой в разные стороны: e^- частицы двигаются в направлении уменьшения координаты, а e^+ частицы — в направлении увеличения координаты. Абсолютные величины скоростей всех частиц одинаковы и равны 1.

Если в процессе перемещения частицы e^- и e^+ оказываются в одной точке, то они взаимодействуют и обе исчезают, при этом они не влияют на дальнейшее поведение остальных частиц.

Учёные выбрали m различных моментов времени t_1, t_2, …, t_m и решили выяснить, какое количество частиц находится в БЛК непосредственно после каждого из этих моментов времени. Отсчёт времени начинается с момента 0, когда частицы приходят в движение. Частицы, исчезнувшие в результате взаимодействия в момент времени t_i, не должны учитываться при подсчёте количества частиц для этого момента времени.

Требуется написать программу, которая по описанию исходного расположения и типов частиц для каждого заданного момента времени определяет количество частиц, находящихся в БЛК непосредственно после этого момента.

Формат входных данных

Первая строка входного файла содержит число n — количество частиц ($1 \leqslant n \leqslant 200\,000$). Последующие n строк описывают частицы следующим образом: каждая строка содержит два целых числа x_i и ν_i — координату i-й частицы и ее тип соответственно ($-10^9 \leqslant x_1 < x_2 < \ldots < x_n \leqslant 10^9$, число ν_i равно -1 или 1). Частица e^- имеет тип $\nu_i = -1$, а частица e^+ — тип $\nu_i = 1$.

Следующая строка содержит целое число m — количество моментов времени, которые выбрали учёные ($1 \leqslant m \leqslant 200\,000$). Последняя строка содержит m целых чисел: $t_1, t_2, \ldots, t_m$ ($0 \leqslant t_1 < t_2 < \ldots < t_m \leqslant 10^9$).

Формат выходных данных

Для каждого момента времени во входном файле требуется вывести одно число — количество частиц в БЛК непосредственно после этого момента.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		n, x_i	m, t_i	
1	35	$1 \leqslant n \leqslant 100$, $-100 \leqslant x_i \leqslant 100$	$m = 1$, $0 \leqslant t_1 \leqslant 100$	
2	12	$1 \leqslant n \leqslant 100$, $-10^9 \leqslant x_i \leqslant 10^9$	$m = 1$, $0 \leqslant t_1 \leqslant 10^9$	**1**
3	12	$1 \leqslant n \leqslant 200\,000$, $-10^9 \leqslant x_i \leqslant 10^9$	$m = 1$, $0 \leqslant t_1 \leqslant 10^9$	**1, 2**
4	41	$1 \leqslant n \leqslant 200\,000$, $-10^9 \leqslant x_i \leqslant 10^9$	$1 \leqslant m \leqslant 200\,000$, $0 \leqslant t_i \leqslant 10^9$	**1, 2, 3**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

linear.in	linear.out
4 -1 1 0 -1 1 1 5 -1 4 0 1 2 3	4 2 0 0

Пояснение к примеру

В приведённом примере в начальный момент в БЛК находятся 4 частицы: частица e^+ в точке -1, частица e^- в точке 0, частица e^+ в точке 1 и частица e^- в точке 5.

В момент времени 0.5 первые две частицы e^+ и e^- сталкиваются в точке с координатой -0.5 и исчезают. В момент времени 1 оставшиеся две частицы находятся в точках с координатами 2 и 4 соответственно. В момент времени 2 они сталкиваются в точке 3 и исчезают. Больше в БЛК частиц нет.

Задача 7. Силовые поля

Имя входного файла:	`power.in`
Имя выходного файла:	`power.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

В физико-биологической лаборатории исследуют воздействие излучения на растения при облучении через *силовые* поля.

Экспериментальная установка содержит квадратную платформу размером $10^9 \times 10^9$, заполненную плодородной почвой. Над платформой установлен источник излучения. Между источником излучения и платформой можно включать n силовых полей.

Генератор силовых полей установлен над точкой $(0, 0)$. При этом i-е силовое поле представляет собой прямоугольник со сторонами, параллельными границам платформы, и координатами двух противоположных углов $(0, 0)$ и (x_i, y_i).

В эксперименте планируется изучать воздействие излучения на растения при облучении k силовых полей. Из заданных n полей необходимо выбрать k полей для эксперимента. Ученые хотят выбрать силовые поля таким образом, чтобы площадь участка платформы, над которой находятся все k выбранных силовых полей, была максимальна.

Требуется написать программу, которая по заданным целым числам n, k и описанию n силовых полей определяет, какие k силовых полей необходимо выбрать для эксперимента, чтобы площадь участка, покрытого этими k силовыми полями, была максимальна, и выводит площадь этого участка.

Формат входных данных

Первая строка входного файла содержит целые числа n и k $(1 \leqslant k \leqslant n \leqslant 200\,000)$ — общее количество силовых полей и количество силовых полей, которые нужно выбрать для эксперимента.

Следующие n строк содержат по два целых числа x_i, y_i $(1 \leqslant x_i,\ y_i \leqslant 10^9)$ — координаты дальнего от начала координат угла прямоугольного участка i-го силового поля.

Формат выходных данных

Требуется вывести одно целое число — максимальную площадь искомого участка.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		n	k	
1	18	$1 \leqslant n \leqslant 20$	$1 \leqslant k \leqslant n$	
2	25	$1 \leqslant n \leqslant 300$	$1 \leqslant k \leqslant n$	**1**
3	20	$1 \leqslant n \leqslant 3000$	$1 \leqslant k \leqslant n$	**1, 2**
4	17	$1 \leqslant n \leqslant 200\,000$	$k = 2$	
5	20	$1 \leqslant n \leqslant 200\,000$	$1 \leqslant k \leqslant n$	**1, 2, 3, 4**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

power.in	power.out
5 3 3 5 2 2 2 5 4 4 5 3	9

Пояснение к примеру

На рис. 5 показаны пять силовых полей, заданных во входном файле. Оптимальный способ выбрать из них три поля для эксперимента показан на рис. 6.

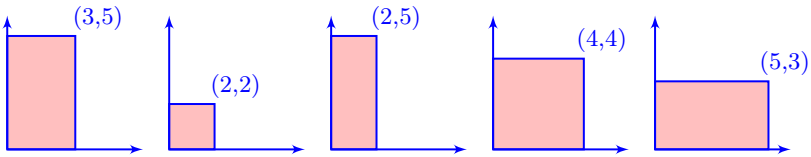


Рис. 5. Силовые поля в примере описания входных данных

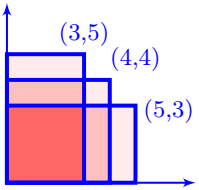


Рис. 6. Оптимальный выбор трёх из пяти силовых полей

Задача 8. Повышение квалификации

Имя входного файла:	`qual.in`
Имя выходного файла:	`qual.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Взаимодействие сотрудников в некоторой компании организовано в виде иерархической структуры. Каждый из n сотрудников имеет свой уникальный номер — целое число от 1 до n, директору компании присвоен номер 1. У любого сотрудника, кроме директора, есть ровно один непосредственный начальник. Начальник сотрудника i имеет номер p_i, причем $p_i < i$.

Сотрудник x является подчинённым уровня 1 сотрудника y, если $p_x = y$. Для $k > 1$ сотрудник x является подчинённым уровня k сотрудника y, если сотрудник p_x является подчинённым уровня $k - 1$ сотрудника y.

Директор компании собирается направить некоторых сотрудников на курсы повышения квалификации. Для этого он решил выбрать два числа L и R и направить на курсы всех сотрудников с номерами i, такими что $L \leqslant i \leqslant R$.

Перед тем, как выбрать числа L и R, директор собрал m пожеланий от сотрудников компании; j-е пожелание задается двумя числами u_j и k_j и означает, что сотрудник u_j просит отправить на курсы одного из своих подчинённых уровня k_j. Для экономии средств директор хочет выбрать такие L и R, чтобы количество сотрудников, направленных на повышение квалификации, было минимальным возможным, но при этом все пожелания были выполнены.

Требуется написать программу, которая по заданным в компании отношениям начальник-подчиненный и пожеланиям сотрудников определяет такие числа L и R, что если отправить на курсы повышения квалификации всех сотрудников с номерами от L до R включительно, то все пожелания будут выполнены, а количество сотрудников, направленных на повышение квалификации, будет минимальным возможным. Если оптимальных пар чисел L, R будет несколько, требуется найти ту из них, в которой значение L минимально.

Формат входных данных

Первая строка входного файла содержит число n — количество сотрудников компании ($2 \leqslant n \leqslant 200\,000$). Вторая строка содержит $(n-1)$ чисел: p_2, p_3, …, p_n ($1 \leqslant p_i < i$) — номера непосредственных начальников сотрудников.

Третья строка содержит одно число m — количество пожеланий от сотрудников.

Следующие m строк задают пожелания сотрудников и содержат по два целых числа u_j, k_j ($1 \leqslant u_j < n$, $1 \leqslant k_j < n$, причём гарантируется, что у сотрудника u_j есть хотя бы один подчинённый уровня k_j).

Формат выходных данных

Необходимо вывести два искомых числа L и R. Если оптимальных пар L, R несколько, требуется вывести ту, в которой значение L минимально.

Система оценивания

Номер подзадачи	Баллы	Ограничения		Необходимые подзадачи
		n, m	p_i	
1	19	$2 \leqslant n \leqslant 50$ $1 \leqslant m \leqslant 50$		
2	25	$2 \leqslant n \leqslant 3000$ $1 \leqslant m \leqslant 3000$		**1**
3	21	$2 \leqslant n \leqslant 200\,000$ $1 \leqslant m \leqslant 200000$	$p_i = i - 1$, для всех i	
4	35	$2 \leqslant n \leqslant 200\,000$ $1 \leqslant m \leqslant 200000$		**1, 2, 3**

Баллы за каждую подзадачу начисляются, если все тесты этой подзадачи и необходимых подзадач успешно пройдены.

Пример

qual.in	qual.out
7 1 1 2 2 3 3 3 1 1 3 1 1 2	3 6

Пояснение к примеру

На повышение квалификации будут направлены сотрудники с номерами 3, 4, 5 и 6. Сотрудник с номером 3 является подчинённым уровня 1 сотрудника с номером 1, сотрудник с номером 4 — подчиненным уровня 2 сотрудника с номером 1, а сотрудник с номером 6 — подчинённым уровня 1 сотрудника с номером 3.

Задача 9. Гипертроичность (7-8 классы)

Имя входного файла:	`ternary.in`
Имя выходного файла:	`ternary.out`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Сверхсекретный научно-исследовательский институт занимается разработкой специального троичного гиперкомпьютера. Если обычный компьютер оперирует числами, представленными в двоичной системе счисления, то для гиперкомпьютера изобретена особая *гипертроичная* система. В инструкции к гиперкомпьютеру написано следующее:

Гипертроичным представлением числа n называется представление n в виде суммы степеней тройки, среди которых каждая степень встречается не более *трёх* раз.

Выяснилось, что у числа может быть несколько гипертроичных представлений. К примеру, у числа 9 их три: 9, $3+3+3$ и $3+3+1+1+1$.

Вычислите количество возможных гипертроичных представлений заданного числа n.

Формат входных данных

Входной файл содержит целое число n ($1 \leqslant n \leqslant 2 \cdot 10^{18}$).

Формат выходных данных

Выведите единственное число — количество возможных гипертроичных представлений n.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Необходимые подзадачи
		n	
1	60	$1 \leqslant n \leqslant 10^{16}$	
2	40	$1 \leqslant n \leqslant 2 \cdot 10^{18}$	**1**

Баллы за подзадачу 1 начисляются, если пройдены все тесты

этой подзадачи. Баллы за подзадачу 2 начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Примеры

ternary.in	ternary.out
3	2
9	3

Задача 10. Будильники (7-8 классы)

Имя входного файла:	`clocks.in`
Имя выходного файла:	`clocks.out`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

На столе старого часовщика лежат n остановившихся будильников, пронумерованных натуральными числами от 1 до n. Будильники измеряют время в часах, причём в одном часе миллион минут, а каждая минута длится миллион секунд. Для отладки механизмов часовщик должен синхронизировать время на всех будильниках. Для этого он передвигает стрелки *вперед* на некоторое время (возможно, нулевое). Величину такого передвижения назовем временем перевода.

Ваша задача — подсчитать *наименьшее* суммарное время перевода, необходимое для того, чтобы все будильники показывали одинаковое время.

Формат входных данных

В первой строке записано единственное целое число n — количество будильников ($2 \leqslant n \leqslant 10^5$). В каждой i-ой из n следующих строк указано время h, m, s, которое показывают i-ые часы. Целые числа h, m и s указывают количество часов, минут и секунд соответственно ($0 \leqslant h < 12$, $0 \leqslant m < 10^6$, $0 \leqslant s < 10^6$).

Формат выходных данных

В единственной строке запишите через пробел три целых числа h, m и s — наименьшее суммарное время перевода, где h, m и

s — целые числа, указывающие количество часов, минут и секунд соответственно ($0 \leqslant m < 10^6$, $0 \leqslant s < 10^6$).

Система оценивания

Номер подзадачи	Баллы	Ограничения	Необходимые подзадачи
		n	
1	50	$2 \leqslant n \leqslant 10^3$	
2	50	$2 \leqslant n \leqslant 10^5$	**1**

Баллы за подзадачу 1 начисляются, если пройдены все тесты этой подзадачи. Баллы за подзадачу 2 начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Примеры

clocks.in	clocks.out
2 10 0 0 3 0 0	5 0 0
3 11 999999 999999 0 0 0 11 999999 999999	0 0 2

Решения задач

Задача 1. Две доминошки

Автор задачи : *фольклор.*
Разработчик : *Киндер М.И.*
Разбор задачи: *Киндер М.И.*

Две доминошки $(a\,;\,b)$ и $(c\,;\,d)$ можно поставить рядом только, когда одно из чисел в паре $(a\,;\,b)$ совпадает с одним из чисел в паре $(c\,;\,d)$, то есть когда $a = c$ или $a = d$ или $b = c$ или $b = d$. В каждом из этих случаев легко восстанавливается порядок расположения доминошек. Если не выполняется ни одно из этих условий, выводим -1.

---

if $a = c$ **then** write$(b,' ',a,' ',a,' ',d)$ **else**
if $a = d$ **then** write$(b,' ',a,' ',a,' ',c)$ **else**
if $b = c$ **then** write$(a,' ',b,' ',b,' ',d)$ **else**
if $b = d$ **then** write$(a,' ',b,' ',b,' ',c)$ **else**
write$('-1')$;

Задача 2. Красивое число

Автор задачи : *Киндер М.И.*
Разработчик : *Киндер М.И.*
Разбор задачи: *Киндер М.И.*

Если число n удовлетворяет условию, то чётные и нечётные цифры в его записи обязательно чередуются. Поэтому если в записи n первая слева цифра нечётная (чётная), то все цифры на нечётных местах тоже должны быть нечётными (чётными), а все цифры на чётных местах — чётными (нечётными). Другими словами, каждые две *соседние* цифры в записи числа n имеют разные остатки при делении на 2. Проверку этого утверждения можно сделать за $O(n)$ операций.

Приведем фрагмент кода на языке Pascal:

```
flag := 1;
for k := 1 to n - 1 do
    if m[k] mod 2 = m[k + 1] mod 2 then flag := 0;
if flag = 0 then write('no');
if flag = 1 then write('yes');
```

Задача 3. Очередь

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Приведем решение подзадачи 1. Отметим очевидный факт: за фанатом, стоящим в конце очереди, никто не стоит. Поэтому для каждого i-ого фаната (первый элемент в i-ой строке) пытаемся найти фаната (среди вторых элементов), который стоит за ним. Как мы знаем, для фаната, стоящего в конце, такого нет. Действуя таким образом, за $O(n^2)$ операций мы найдем номер первого с конца очереди фаната. По этому номеру восстанавливаем всю очередь.

Приведем решение подзадачи 2. Очередь фанатов образует перестановку чисел от 1 до n, поэтому можно подсчитать количество вхождений каждого числа и определить два номера — начало и конец очереди, — то есть числа, которые входят в заданный набор по одному разу. Одновременно в специальном массиве запоминаем номера на футболках фанатов, стоящих перед каждым из них. Для фаната, стоящего в конце очереди, этот номер (по умолчанию) равен 0. Определив номер последнего фаната, восстанавливаем всю очередь, используя специальный массив стоящих впереди фанатов. Сложность алгоритма — $O(n)$.

Задача 4. Две дроби

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Приведем одно из возможных решений. Пусть a/b — исходная, а c/d — требуемая дробь. Сначала покажем, как из любой дроби можно получить число 1. Если $a > b$, применяем операцию **B** вычитания единицы. Каждая такая операция, очевидно, уменьшает числитель дроби на величину знаменателя. Используя эту операцию несколько раз (а именно, $a \,\texttt{div}\, b$), придём к дроби a'/b, у которой числитель a' будет уже меньше знаменателя b. Применяя операцию **C**, заменим эту дробь на обратную b/a', у которой числитель b опять будет больше знаменателя a'. Вновь повторяем операцию **B** по уменьшению числителя до тех пор, пока числитель не будет меньше знаменателя, и так далее. Поскольку числитель и знаменатель дроби не может уменьшаться безгранично, через несколько шагов придем к дроби с числителем 1. Используя операцию **C**, получим дробь со знаменателем 1, то есть целое число. После нескольких операций вычитания приходим к 1. (Приведённая процедура, по сути, совпадает с алгоритмом Евклида вычисления наибольшего общего делителя чисел a и b.)

С помощью описанной процедуры любую несократимую дробь a/b можно привести к 1. И наоборот, из 1 можно получить несократимую дробь c/d. Для этого последовательность операций от c/d к 1 нужно записать в обратном порядке, причём каждую операцию **B** в этой последовательности заменить обратной к ней операцией **A**. Обратная для операции **C** — сама операция **C**, поэтому заменять её не нужно.

Теперь процесс преобразования дроби a/b в дробь c/d можно описать схемой $a/b \to 1 \to c/d$.

Сложность алгоритма — $O(a + b + c + d)$.

Задача 5. Муравей на кубе

Автор задачи : *фольклор.*
Разработчик : *Попов А.А.*
Разбор задачи: *Попов А.А.*

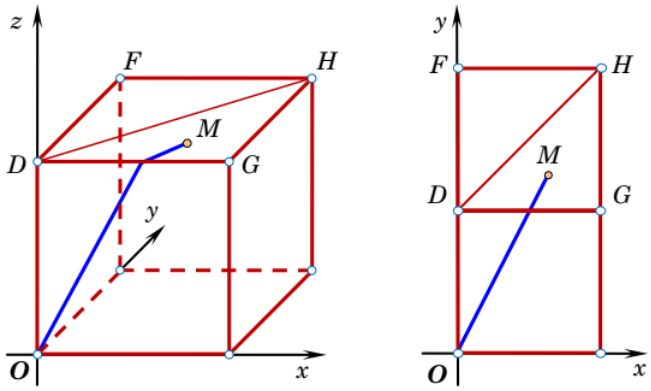
Если муравей находится на одной из граней, содержащей начало координат, то длину кратчайшего пути до точки O можно найти, используя теорему Пифагора. Например, если муравей находится в плоскости XOZ, то длина кратчайшего пути равна OM, где $OM^2 = x^2 + z^2$.

Если муравей находится на одной из граней, не содержащей начало координат, то, повернув эту грань вокруг одного из её ребер, можно добиться, чтобы весь путь муравья лежал в одной плоскости. Отметим, что длина пути при таких поворотах не меняется. Повороты можно сделать вокруг двух рёбер грани, на которой находится муравей. Выбор ребра рассмотрим на примере, в котором муравей находится на плоскости $DGHF$.

Если муравей находится внутри треугольника DGH (то есть $x > y$), то кратчайший путь достигается при повороте грани вокруг ребра DG, при этом $OM^2 = x^2 + (a + y)^2$.

Если муравей находится внутри треугольника DFH (то есть $x < y$), то при повороте грани вокруг ребра получаем $OM^2 = y^2 + (a + x)^2$.

Если $x = y$, то справедлива любая из этих формул.



Задача 6. Факторизации

Автор задачи : *Киндер М.И.*
Разработчик : *Киндер М.И.*
Разбор задачи: *Киндер М.И.*

Решение задачи основано на использовании идеи динамического программирования.

Пусть $F(n, m)$ — количество неупорядоченных факторизаций числа n, у которых наибольший множитель не превосходит m. Расположим все делители числа n в невозрастающем порядке и рассмотрим произвольное разложение числа $n = d_1 \cdot d_2 \cdot \ldots \cdot d_k$ на множители с наибольшей частью $d_1 \leqslant m$. Заметим, что все остальные множители в произведении $d_2 \cdot d_3 \cdot \ldots \cdot d_k = n \,\mathtt{div}\, d_1$ не превосходят d_1, поэтому количество таких факторизаций с наибольшей частью $d_1 \leqslant m$ равно $F(n \,\mathtt{div}\, d_1, d_1)$. В качестве d_1 можно выбрать любой делитель d числа n, не превосходящий m. Суммируя по всем таким d, получим рекуррентное соотношение:

$$F(n, m) = \sum_{\substack{d \mid n \\ d \leqslant m}}^{n} F(n \,\mathtt{div}\, d, d).$$

(Сумма вычисляется во всем делителям d числа n.) Например,

$$F(10, 10) = F(1, 10) + F(2, 5) + F(5, 2) + F(10, 1).$$

Техническая реализация этого алгоритма сравнительно несложная. Сначала находим список делителей $d[1], d[2], \ldots, d[j]$ данного числа n:

```
j := 0;
for i := 1 to m do
    if n mod i = 0 then begin inc(j); d[j] := i; end;
```

Требуемое количество факторизаций будет равно значению рекурсивной функции $Fact(n, j)$:

```
Function Fact (n : integer; m : integer) : integer;
begin
    if m = 1 then begin Result := 0; exit; end;
    if n = 1 then begin Result := 1; exit; end;
    s := 0;
    for i := 1 to m do
        if n mod d[i] = 0 then inc(s, Fact(n div d[i], i));
    Result := s;
end;
```

Региональный этап, 2015-2016

Задача 1. Призы

Автор задачи : *жюри.*
Разработчик : *жюри.*
Разбор задачи: *жюри.*

Основные темы задачи «Призы»[1] — линейный перебор вариантов, поиск первого и второго максимума в массиве. Возможные частичные решения основаны на алгоритмах сортировки и умении использовать встроенные в стандартную библиотеку алгоритмы сортировки.

Решение задачи основано на классическом алгоритме поиска второго максимума в массиве. Будем поддерживать две переменные — текущий максимум в массиве и второй по величине элемент. Тогда при добавлении в рассмотрение очередного приза обновим при необходимости первый и второй максимум.

Приведем основной фрагмент кода решения на языке C++.

```
for (int i = 0; i < n; ++i) {
    int num;
    scanf("%d", &num);
    if (num > m1) {
        m2 = m1;
        m1 = num;
    } else if (num > m2) m2 = num;
    if (i > 0) {cout << m2 << " ";}
};
```

Альтернативное решение может базироваться на использовании встроенной в язык программирования структуры данных для поддержания упорядоченного множества. Приведем пример решения на языке C++ с использованием структуры данных

[1] Все задачи регионального этапа составлены жюри и Центральной предметно-методической комиссией Всероссийской олимпиады школьников по информатике, поэтому здесь и далее в качестве авторов и разработчиков задач сокращенно указывается *жюри.*

`std::multiset`. В множестве поддерживаются два максимальных элемента, каждый раз добавляется один элемент и удаляется минимальный элемент.

```
in >> n;
multiset<int> ma;
int x, y;
in >> x >> y;
ma.insert(x);
ma.insert(y);
for (int i = 1; i < n; i++) {
    out << *ma.begin() << " ";
    in >> x;
    ma.insert(x);
    ma.erase(ma.begin());
};
```

Сложность такого решения $O(n)$. Частичные решения для первой подзадачи могут использовать квадратичную сортировку для поддержания массива отсортированным после добавления каждого элемента. Сложность такого решения $O(n^3)$.

Частичные решения для второй подзадачи могут использовать сортировку за $O(n \log n)$ для поддержания массива отсортированным после добавления каждого элемента. В частности, возможно использование встроенной в стандартную библиотеку функции сортировки. Время работы такого решения — $O(n^2 \log n)$.

Альтернативным подходом может быть использование сортировки «вставками». Будем поддерживать массив отсортированным и каждый раз при добавлении очередного элемента будем «утапливать» его до необходимого места. Время работы такого решения — $O(n^2)$.

Задача 2. Космическое поселение

Автор задачи : *жюри.*
Разработчик : *жюри.*
Разбор задачи: *жюри.*

Основная тема задачи — двоичный поиск. Возможные частичные решения основаны на линейном переборе вариантов и идее перебора делителей числа до квадратного корня.

Для полного решения этой задачи воспользуемся двоичным поиском. Заметим, что если можно установить защиту толщиной d, то можно установить и меньшую защиту.

Для проверки возможности установки защиты данной толщины переберем два варианта ориентации модулей. Пусть, например, модуль ориентирован так, что сторона длиной a ориентирована вдоль стороны поля длиной w. Тогда после установки защиты толщиной d вдоль этой стороны можно установить w `div` $(a + 2d)$ модулей (при делении округлять следует вниз). Аналогично, вдоль другой стороны можно установить h `div` $(b+2d)$ модулей. В целом же на поле можно установить $(w$ `div` $(a+2d)) \times (h$ `div` $(b+2d))$ модулей. Если это число больше или равно n, то защиту толщиной d или больше установить можно, иначе — нельзя.

Приведем основной фрагмент кода решения на языке `C++`.

```
long long L = 0;
long long R = min(w, h) + 1;
while (L + 1 < R) {
    long long M = (L + R)/2;
    long long ad = a + 2 * M;
    long long bd = b + 2 * M;
    long long ac = w / ad;
    long long bc = h / bd;
    if (ac * bc >= n) {L = M;} else {R = M; }
};
return L;
```

Отметим следующий вопрос, связанный с возможным переполнением при вычислении результатов 64-битного типа. В случае, если M больше ответа на задачу, переполнения не про-

исходит, так как перемножаются числа, произведение которых меньше n. Если же M оказывается меньше ответа, то, благодаря структуре двоичного поиска, M меньше ответа не более чем вдвое, значит вычисляемое значение не превышает $4n$ и также помещается в 64-битный тип данных.

Для решения подзадачи 1 достаточно перебрать все возможные значения толщины защиты линейным проходом.

Для решения подзадачи 2 требуется заметить, что при размещении модулей вдоль каждой стороны поля размещается не более n модулей, поэтому можно осуществить перебор, сколько модулей будет расположено вдоль стороны длиной w и проверить для такого количества модулей вдоль этой стороны, какая толщина защиты возможна.

Для решения подзадачи 3 можно развить эту идею, заметив, что вдоль одной из сторон размещается не более $\texttt{sqrt}(n)$ модулей, а значит, можно ускорить перебор.

Задача 3. Странные строки

Автор задачи : *жюри.*
Разработчик : *жюри.*
Разбор задачи: *жюри.*

Основные темы задачи — структуры данных, работа со строками, сканирующая прямая. Возможные частичные решения основаны на использовании моделирования, структуры данных «множество».

Сначала докажем следующее вспомогательное утверждение: строка z является *странной* в том и только в том случае, когда выполняются следующие свойства:

1) строка z содержит не более двух различных символов;

2) все одинаковые символы в строке z идут подряд.

Докажем сначала второе утверждение. Действительно, пусть, например, в строке содержатся два одинаковых символа a, между которыми находится еще один символ, отличный от a. Пусть a встречается в строке k раз. Тогда строка из k символов a является подпоследовательностью z, но не ее подстрокой.

Пусть теперь в z встречаются хотя бы три различных символа. По доказанному, одинаковые символы должны идти подряд. Но тогда, если первый символ строки a, а последний — b, то строка ab является подпоследовательностью z, но не ее подстрокой. Наоборот, если строка z состоит из k символов a, после которых идет l символов b, то каждое из множеств $W(z)$ и $Y(z)$ состоит только из строк, в которых сначала идет не более k символов a, а затем не более l символов b.

Таким образом, для определения странности строки требуется найти количество ее подстрок, которые имеют именно такую структуру.

Будем решать задачу независимо для каждой упорядоченной пары (a, b) символов.

Пусть у нас есть несколько блоков подряд идущих символов a, после которых идет блок подряд идущих символов b. Соединим все пары (k_i, l_i) длин таких блоков в массив. Теперь нам надо найти количество наборов (k, l), для которых найдется пара (k_i, l_i), такая что $1 \leqslant k \leqslant k_i$ и $1 \leqslant l \leqslant l_i$.

Для подсчёта количества таких наборов отсортируем все пары по k_i, а затем по l_i. Для каждой пары (k_i, l_i) заменим l_i на наибольшее l_j с условием $k_j \geqslant k_i$. Теперь оставим по одной паре с каждым k_i и просуммируем величины $(k_i - k_{i-1}) \times l_i$ для всех i. Получившееся значение суммы является искомым.

Отметим, что с помощью описанного алгоритма мы фактически находим площадь объединения прямоугольников с осями, параллельными осям координат, общей вершиной (0, 0) и положительными координатами противоположного угла.

В завершение описания полного решения отметим, что построение списков пар (k_i, l_i) можно осуществить *одним проходом* по массиву.

При решении подзадачи 1 достаточно сформулировать утверждение о том, в каком случае строка является странной. После этого все подстроки могут быть проверены на странность независимо. При этом для устранения дубликатов можно, например, сложить все такие подстроки в структуру данных «множество».

Для решения подзадачи 2 не требуется новых идей по сравнению с подзадачей 1, однако при некоторых вариантах реализации повышается техническая сложность. Именно по этой причине

стоимость подзадачи 2 невысока.

Решение подзадачи 3 требует эффективного метода устранения повторяющихся подстрок. Например, все наборы (k, l), для которых существует пара (k_i, l_i), такая что $1 \leqslant k \leqslant k_i$ и $1 \leqslant l \leqslant l_i$, можно объединить в структуру данных «множество», при этом множество должно быть отдельным для каждой пары символов (a, b).

Задача 4. Поездка на каникулах

Автор задачи : *жюри.*
Разработчик : *жюри.*
Разбор задачи: *жюри.*

Это самая сложная задача первого тура, она относится к темам: жадные алгоритмы, структуры данных для минимума, двоичные подъемы. Возможные частичные решения могут использовать перебор вариантов и моделирование процесса.

Полное решение этой задачи требует нескольких идей.

Пусть для каждой станции i определена величина $go1[i]$ — максимальный номер станции, до которой можно доехать от станции i, используя ровно 1 билет. Заметим, что нам не важно, на каком месте мы приедем на станцию пересадки, поэтому выгодно от станции i всегда ехать либо до конечной станции маршрута, либо до станции $go1[i]$.

Решим сначала задачу для *одного* варианта поездки, используя построенный массив $go1$. Начнем со станции f. Если $go1[f] \geqslant \geqslant t$, очевидно, достаточно одного билета. Иначе переместимся на станцию $go1[f]$, найдем ответ для неё и прибавим 1. Следует также учесть, что если $go1[i] = i$, то от станции i нельзя поехать дальше, и если в процессе перемещения мы встречаем такую станцию, то ответ для рассматриваемого вариант «`-1`». Время работы предложенного алгоритма на запрос — $O(n)$.

Разберёмся теперь, как построить массив $go1$.

Разумеется, возможно, «наивное» построение, которое для каждой станции и каждого места сначала определяет, до какой станции можно доехать с использованием этого билета, а затем

выбирает максимум. В сочетании с описанным методом по массиву $go1$ мы можем найти число билетов для одного варианта за время $O(n)$ и решить подзадачу 1. Для решения подзадачи 2 требуется более совершенный способ построения массива $go1$. Будем говорить, что место a свободно на протяжении отрезка $[c, d]$, если никакой из проданных билетов не занимает это место на перегонах этого отрезка станций. Отсортировав введённые билеты по месту, а затем по начальной станции, легко построить все максимальные свободные отрезки станций.

Отсортируем свободные отрезки по начальной станции.

Рассмотрим станции по очереди. Будем поддерживать множество «открытых» свободных отрезков, которые начинаются не позже рассматриваемой станции, в структуре данных, которая позволяет осуществлять поиск максимума по ключу. (Для этой цели подойдут двоичная куча, дерево отрезков или `std::set`.) В качестве ключа будем использовать конец отрезка.

Рассматривая очередную станцию i, добавим все свободные отрезки, которые в ней начинаются, в множество открытых отрезков. Теперь $go1[i]$ равно либо максимальному концу открытого отрезка, либо значению i, если множество открытых отрезков пусто или все они заканчиваются до i. Этот метод позволяет решить подзадачу 2.

Для решения подзадачи 3 требуется более эффективный алгоритм ответа на запрос. Воспользуемся методом «двоичных подъемов». Подсчитаем величину $go[j][i]$, равную максимальному номеру станции, до которой можно доехать от станции, используя не более 2^j билетов.

Фрагмент кода, вычисляющего $go[j][i]$ по значению $go1[i]$ приведен ниже.

```
for (int i = 1; i <= sz; i++) {
    for (int j = 0; j < n; j++) {
        int u = go[i - 1][j];
        go[i][j] = go[i - 1][u];
    }
};
```

В качестве sz в этом фрагменте следует выбрать минималь-

ную величину, такую что $2^{sz} \geqslant n$.

Теперь для получения ответа на запрос будем делать все уменьшающиеся «прыжки» по степеням двойки. Приведём фрагмент кода, отвечающего на запрос с использованием массива *go*.

```
// перемещаемся от fr до to
if (go[sz][fr] < to) { cout << to << endl; }
else {
    int steps = 0;
    int curK = sz;
    while (go1[fr] < to) {
        while (go[curK][fr] >= to) { curK--; }
        steps += 1 << curK;
        fr = go[curK][fr];
    }
    cout << steps + 1 << endl;
}
```

Задача 5. Три сына

Автор задачи :	*жюри.*
Разработчик :	*жюри.*
Разбор задачи:	*жюри.*

Основные темы задачи — линейный перебор вариантов, вывод формулы. Возможны также частичные решения, основанные на полном переборе вариантов.

Прежде всего, отметим, что для получения минимума суммы квадратов a, b и c необходимо выбрать эти числа, близкими к $n\,/\,3$. Формализуем это утверждение.

Докажем сначала вспомогательный факт для двух чисел.

Пусть a и b — различные положительные целые числа с фиксированной суммой $a + b$, и $a < b$. Тогда если сумма квадратов $a^2 + b^2$ минимальна, то $b - a \leqslant 2$.

Действительно, если это не так, и $b - a > 2$, то $a + 1 \neq b - 1$ и $(a+1)^2 + (b-1)^2 = a^2 + b^2 + 2(a-b) + 2 < a^2 + b^2 - 4 + 2 < a^2 + b^2$.

Следовательно, взяв вместо a и b числа $a+1$ и $b-1$, мы получим два различных числа с такой же суммой, но меньшей суммой квадратов.

Пусть теперь a, b и c — требуемые в задаче числа. Применим предыдущее утверждение для a и b при фиксированном c, а также для b и c при фиксированном a. В результате получим, что все числа не более чем на 4 отстоят от значения $n/3$.

На этом доказанном факте, основано следующее решение. Переберём все тройки целых положительных чисел, где каждое число, не более чем на 4 отличается от $n/3$, и выберем среди них тройку с *минимальной* суммой квадратов.

Приведем пример программы на C++, которая решает поставленную задачу.

```
int n3 = n / 3;
int delta = 4;
int minv = max(n3 - delta, 1);
int maxv = min(n3 + delta, n);
long long best = (long long) n * n;
int ba, bb, bc = 0;
for (long long a = minv; a <= maxv; a++)
    for (long long b = a + 1; b <= maxv; b++)
        for (long long c = b + 1; c <= maxv; c++)
            if (a + b + c == n && a * a + b * b + c * c < best) {
                best = a * a + b * b + c * c;
                ba = a;
                bb = b;
                bc = c;
            };
cout << ba << " " << bb << " " << bc << endl;
```

Отметим, что вместо числа 4 можно использовать и любое большее значение. Главное, чтобы оно не было слишком большим, чтобы решение по-прежнему «проходило» по времени.

Частичные решения для подзадач 1, 2 и 3 позволяют получить частичные баллы в случае неполного развития этой идеи.

Для решения подзадачи 1 можно перебрать все возможные значения a, b и c и выбрать из них оптимальное. Время работы

такого варианта решения — $O(n^3)$.

Для решения подзадачи 2 следует заметить, что можно перебрать лишь два значения, например, a и b, а значение c вычислить из равенства $a + b + c = n$. Время работы — $O(n^2)$.

Наконец, для решения подзадачи 3 можно, например, перебрать большее из трёх значений c, и заметить, что $a + b = n - c = m$, а для минимизации $a^2 + b^2$ следует взять $a = (m - 1) / 2$, $b = (m + 1) / 2$ при нечётном m, и $a = m / 2 - 1$, $b = m / 2 + 1$ — при чётном m. Время работы — $O(n)$.

Задача 6. Гипершашки

Автор задачи : *жюри.*
Разработчик : *жюри.*
Разбор задачи: *жюри.*

Основные темы задачи — сортировка, метод двух указателей, комбинаторика. Возможны частичные решения, основанные на полном переборе вариантов.

Рассмотрим три случая:

1) *все три игрока набрали равное число баллов;*
2) *все три игрока набрали попарно различное число баллов;*
3) *два игрока из трёх набрали равное число баллов, а количество баллов у третьего отлично от числа баллов двух других.*

Научимся находить количество вариантов счёта в каждом из трёх случаев. (Ответом в задаче будет сумма значений, полученных для этих трёх случаев.) Отметим, что в подзадаче 1 выполнено условие $k = 1$, и, следовательно, имеет место только первый случай, а в подзадаче 3 все числа различны, и, поэтому ситуация относится ко второму случаю. Этим можно воспользоваться для написания частичных решений подзадач.

Поместим входные данные в массив и отсортируем его. Одним проходом по получившемуся массиву заменим числа на пары (x_i, c_i), где c_i — параметр, который учитывает, сколько раз на карточках у Андрея повторяется значение x_i, причём во всех парах значения x_i различны. Заметим, что массив пар получился

отсортированным по x_i.

Приведем пример программы на C++, которая решает поставленную задачу.

```
cin >> n >> k;
vector<long long> x(n);
for (int i = 0; i < n; i++)
    cin >> x[i];

sort(x.begin(), x.end())

vector<pair <long long, int> > p;
int pr = -1;
for (int i = 0; i < n; i++)
    if (i == n - 1 || x[i] != x[i + 1]) {
        p.push_back({x[i], i - pr});
        pr = i;
    }
int m = p.size();
```

В случае, когда все три игрока набрали равное число баллов, Андрей может показать счёт, если у него есть не менее трёх карточек с этим числом баллов. Поэтому количество таких вариантов равно количеству чисел, которые встречаются хотя бы 3 раза. Ниже приведен код на C++, который считает число таких вариантов.

```
long long ans1 = 0;
for (int i = 0; i < m; i++)
    if (p[i].second >= 3) ans1++;
```

Количество вариантов в остальных двух случаях можно найти методом двух указателей.

Для определения количества вариантов счёта, в котором баллы всех игроков различны, сделаем следующее. Для каждого варианта максимального из баллов трёх игроков найдем минимальное значение среди баллов другого игрока, которое можно показать и которое не нарушает условия, что баллы различаются не

более чем в k раз. Пусть для максимального значения x_i соответствующее минимальное значение равно x_j. Тогда два других значения баллов можно выбрать среди значений $x_j, x_{j+1}, \ldots, x_{i-1}$, то есть существует $\frac{1}{2}(i-j)(i-j-1)$ способов это сделать. Поскольку три различных числа можно упорядочить 6 способами, это число необходимо умножить на 6.

Просуммируем эти значения по всем i. Заметим, что при увеличении i значение j также может только увеличиваться, поэтому проход занимает $O(n)$ операций.

Приведем код на C++, который подсчитывает это число вариантов.

```
long long ans2 = 0;
int j = 0;
for (int i = 0; i < m; i++) {
    while (j < i && p[j].first * k < p[i].first) j++;
    ans2 += 6LL * (i - j) * (i - j - 1) / 2;
}
```

Наконец, аналогичный метод применим для подсчета количества вариантов счёта, когда два игрока из трёх набрали равное число баллов, а количество баллов у третьего отлично от числа баллов двух других.

```
long long ans3 = 0;
int j = 0;
for (int i = 1; i < m; i++) {
    while (j < i && p[j].first * k < p[i].first) j++;
    if (p[i].second > 1)   ans3 += 3 * (i - j);
}
j = m - 1;
for (int i = m - 2; i >= 0; i--) {
    while (j > i && p[i].first * k < p[j].first) j--;
    if (p[i].second > 1)   ans3 += 3 * (j - i);
}
```

Для решения подзадачи 1 достаточно рассмотреть вариант, когда у всех игроков одинаковое число баллов. Отметим, что до-

полнительное ограничение $1 \leqslant x_i \leqslant 100\,000$ позволяет обойтись без сортировки пар или сложных структур данных и запоминать количество карточек с каждым числом в массиве.

Для решения подзадачи 2 можно перебрать все тройки карточек, для каждого варианта счёта в трёхмерном массиве отметить, можно ли получить этот счёт, и затем вывести требуемое количество вариантов.

Для решения подзадачи 3 нужно реализовать описанное выше решение, при этом не требуется разбирать случай, когда два игрока набрали одинаковое число баллов.

Задача 7. Интересные числа

Автор задачи : *жюри.*
Разработчик : *жюри.*
Разбор задачи: *жюри.*

Основные темы задачи — динамическое программирование. Возможны частичные решения, основанные на использовании перебора, а также различные неэффективные решения с использованием динамического программирования.

Пусть $d[R]$ — количество интересных чисел на отрезке $[1; R]$, тогда количество интересных чисел на отрезке $[L; R]$ равно $d[R] - d[L-1]$. Из этого рассуждения следует, что достаточно научиться считать количество интересных чисел, не превышающих $\leqslant R$.

Для решения подзадачи 1 переберём все числа от L до R и для каждого из них проверим, является ли оно интересным. Это решение проходит все тесты первой подзадачи с ограничениями $L = 1, R \leqslant 1\,000$.

Для решения подзадачи 2 переберём только интересные числа, сохраняя в переборе уже составленную часть числа и последнюю цифру. На каждом шаге будем дописывать к «текущему» интересному числу только те цифры, которые сохраняют свойство интересности числа. Это решение проходит все тесты второй подзадачи с ограничениями $1 \leqslant L \leqslant R \leqslant 10^{18}$.

Разберём третью подзадачу с ограничениями $L = 1, R = 10^k$, где $2 \leqslant k \leqslant 100$, в которой нужно подсчитать количество инте-

ресных чисел, состоящих *не более* чем из k цифр. Эта задача равносильна подсчёту количества в точности k-значных интересных чисел, в записи которых допускаются ведущие нули. (k-значное число из одних нулей лишнее, поэтому из ответа потом вычтем 1.)

Пусть $d[k][j]$ — количество k-значных интересных чисел, у которых последняя цифра равна j. У каждого такого числа первые слева $k-1$ цифр образуют интересное $(k-1)$-значное число, у которого последняя $(k-1)$-я цифра не больше j. Поэтому при $k > 1$ и для всех j:

$$d[k][j] = \sum_{i=0}^{j} d[k-1][i], \qquad d[1][j] = 1.$$

Все вычисления проводятся по модулю $(10^9 + 7)$, поэтому «длинной» арифметики не потребуется.

Рассмотрим «комбинаторное» решение подзадачи 3. Отметим, что k-значное интересное число с возможными ведущими нулями — это последовательность длины k из 9 групп одинаковых цифр вида

$$\underbrace{00\ldots0}_{k_0}\ \underbrace{11\ldots1}_{k_1}\ \ldots\ \underbrace{99\ldots9}_{k_9} \qquad (k_i \geqslant 0).$$

Добавив между группами символ $*$, получим набор из $k+9$ символов. Обратно, в любом наборе из $k+9$ символов выберем 9 позиций и в каждую из них поставим символ $*$. Все позиции до первого символа $*$ заполним *нулями*, позиции от первого до второго символа $*$ заполним *единицами*, и так далее. Полученная последовательность без символов $*$ — это k-значное интересное число с ведущими нулями. Значит, количество k-значных интересных чисел равно числу способов выбрать 9 из $k+9$ позиций. Это количество равно числу сочетаний

$$C_{k+9}^{9} = \frac{(k+9)(k+8)\ldots(k+1)}{1\cdot 2\cdot \ldots \cdot 9}.$$

Итак, искомое количество интересных чисел в подзадаче 3 равно $C_{k+9}^{9} - 1$.

Обратимся теперь к полному решению задачи.

Пусть теперь $R = \overline{a_1 a_2 \dots a_n}$ не является степенью 10. Рассмотрим префикс интересного числа $R_0 = \overline{b_1 b_2 \dots b_n}$, не превосходящего R. Если в префиксе R_0 есть хотя бы одна цифра $b_i < a_i$, то после этой цифры продолжение интересного числа может быть любым. Если же в префиксе R_0 таких цифр нет, то все цифры b_i должны совпадать с цифрами a_i префикса R и следующая цифра не должна превышать следующей цифры R.

Пусть l — длина общего префикса интересного числа R_0 и R. Проведём вычисления для каждого значения l.

Пусть $d[k][j]$ — количество k-значных интересных чисел, у которых последняя цифра равна j, и при этом первые l цифр совпадают с соответствующими цифрами R. Тогда формула пересчёта не меняется:

$$d[k][j] = \sum_{i=0}^{j} d[k-1][i].$$

Изменяются лишь начальные значения: вместо $d[1][j] = 1$ будет $d[l+1][j] = 1$ для всех j с условием $a_l \leqslant j < a_{l+1}$. (Первые l цифр числа R идут в неубывающем порядке.)

Задача 8. Гармоничная последовательность

Автор задачи :	*жюри.*
Разработчик :	*жюри.*
Разбор задачи:	*жюри.*

Основные темы задачи — методы оптимизации, линейный перебор вариантов, двоичный поиск. Возможные частичные решения основаны на использовании линейного перебора вариантов.

Обозначим первые два числа гармонической последовательности через x и y. Тогда сама последовательность будет иметь вид

$$x,\ y,\ y-x,\ -x,\ -y,\ x-y,\ x,\ y, \dots .$$

Заметим, что последовательность имеет цикл длины шесть. Значит, если начальная последовательность имеет вид b_1, b_2, b_3, …, то необходимо подобрать целые числа x и y, которые минимизи-

руют функцию $d = |x - b_1| + |y - b_2| + |y - x - b_3| + |-x - b_4| + |-y - b_5| + |x - y - b_6| + \ldots$, которую можно записать в виде

$$d = |\,x - b_1\,| + |\,x - (-b_4)\,| + |\,y - b_2\,| + |\,y - (-b_5)\,| + \\ + |\,(y - x) - b_3\,| + |\,(y - x) - (-b_6)\,| + \ldots\,.$$

Таким образом, необходимо минимизировать сумму модулей величин, которые можно разбить на три группы. Слагаемые первой группы имеют вид $|\,x - a\,|$, второй — $|\,y - b\,|$, а третьей — $|\,(y - x) - c\,|$.

Заметим, что существует пара (x, y), которая минимизирует функцию d, такая что как минимум в двух из трёх групп значение хотя бы одного из модулей равно нулю. (Это свойство несложно доказать от противного.) Выделяя соответствующие модули, равные нулю, мы можем однозначно восстановить числа x и y. Эти рассуждения дают решение с алгоритмической сложностью $O(n^3)$: зафиксируем две из трёх групп, переберём, какие именно модули равны нулю, затем восстановим последовательность и обновим ответ.

Предположим, что все числа в исходной последовательности по модулю не превосходят A. Заметим, что существует правильный ответ, в котором одно из чисел x и y не превосходит по модулю A, а другое $2A$. Действительно, поскольку одно из слагаемых вида $|\,x - a_i\,|$ или $|\,y - b_i\,|$ обязательно равно нулю, отсюда однозначно определяется значение одной из переменных. Значение другой переменной при этом можно будет найти из уравнения вида $|\,(y - x) - c_j\,| = 0$.

Таким образом, для решения подзадачи 1 можно просто перебрать значения трёх элементов последовательности в пределах от -20 до 20.

Для решения подзадачи 2 можно перебрать значения x и y в пределах от -200 до 200, восстановить последовательность и выбрать лучший вариант.

Для решения подзадачи 3 можно перебрать значения x, получить выражение вида $|\,y - y_1\,| + |\,y - y_2\,| + \ldots$, которое необходимо минимизировать. Далее выбрать модуль, который будет равен нулю, и, используя префиксные и суффиксные суммы, посчитать значение выражения.

Для решения подзадачи 4 с помощью перебора можно определить, какие два модуля равны нулю, а затем с помощью префиксных и суффиксных сумм посчитать ответ.

Для решения подзадачи 5 необходимо заметить следующий факт. Если для некоторого x найдено минимальное значение y, которое является оптимальным, то для большего x оптимальное значение y не может быть меньше найденного.

Таким образом, полное решение задачи выглядит следующим образом. Разобьем все модули на три группы. Отсортируем модули в порядке увеличения значения константы в них. Переберём две группы, в которых будут равные нулю модули. (Без ограничения общности можно считать, что это группы вида $|x - x_i|$ и $|y - y_i|$). Выберём, какой из модулей первой группы равен нулю. Будем поддерживать указатель на модуль из второй группы, для которого при обращении его значения в ноль получается оптимальный ответ. При изменении выбранного модуля первой группы, попытаемся использовать модуль с большей константой во второй группе. (Пока следующий модуль даёт ответ лучше, чем текущий, будем увеличивать указатель).

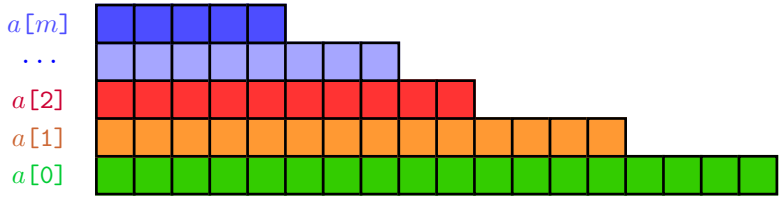
Для того, чтобы посчитать ответ для конкретной пары, необходимо определить, какие из модулей третьей группы «раскрываются» с плюсом, а какие — с минусом. Для этого можно воспользоваться двоичным поиском, а далее с помощью префиксных и суффиксных сумм посчитать значение выражения. Таким образом, общая сложность решения равна $O(n \log n)$.

В заключение заметим, что функция, которую необходимо минимизировать, является выпуклой вниз и по x, и по y, поэтому для решения задачи можно также воспользоваться различными численными методами, например, градиентным спуском или двумерным троичным поиском. В зависимости от эффективности вычисления значения целевой функции такие решения могут набрать от 44 до 100 баллов.

Задача 9. История одного города

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Изобразим числа последовательности $a_0, a_1, \ldots, a_{m-1}$ в виде таблицы. (Они называются *диаграммами Юнга.*) Тогда элементы исходной последовательности соответствуют столбцам этой таблицы. Осталось посчитать число «этажей» в каждом столбце. Ясно, что общее количество домов в городе равно $a[0]$. Обозначим через $b[j]$ — количество этажей в j-ом доме.



Для решения подзадачи 1 можно непосредственно посчитать количество «этажей» в каждом доме. Пример кода на языке Pascal:

```
for i := 0 to m - 1 do
    for j := 1 to a[i] do inc(b[j]);
writeln(a[0]);
for i := 1 to a[0] do
    write(b[i],' ');
```

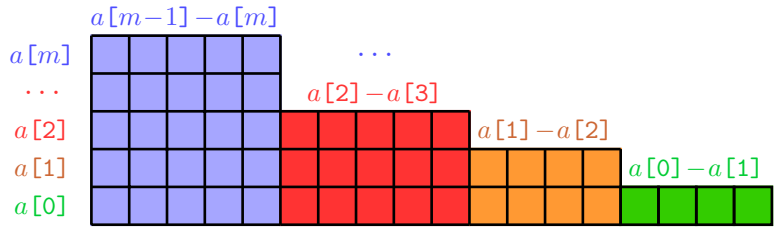
Это решение проходит тесты при всех $m \leqslant 40\,000$. Сложность алгоритма — $O(m^2)$.

Для полного решения задачи достаточно заметить, что количество одноэтажных домов равно $a[0] - a[1]$, двухэтажных домов $a[1] - a[2]$, и так далее, наконец, количество m-этажных домов равно $a[m-1] - a[m]$. Поэтому для каждого $i = 0, 1, \ldots, m-1$, выводим количество этажей $i+1$ в количестве $a[i] - a[i+1]$ раз.

Пример кода на языке Pascal:

```
writeln(a[0]);
for i := m − 1 downto 0 do
    for j := a[i + 1] + 1 to a[i] do
        write(i + 1,' ');
```

Это решение проходит тесты при всех $m \leqslant 10^6$. Сложность алгоритма — $O(m)$.



Задача 10. Кофе

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Основные темы задачи — методы оптимизации, структура данных «стек». Возможные частичные решения основаны на использовании перебора вариантов.

Прежде всего, заметим, что всегда существуют два сорта кофе `A` и `B` такие, что кофе сорта `A` — не более n пачек, а кофе сорта `B` — не менее n пачек. В самом деле, если это не так и, например, кофе каждого сорта более n пачек, то общее количество пачек кофе превышает значение $m \cdot n$. Аналогичным рассуждением получаем противоречие и в случае, когда кофе каждого сорта менее n пачек.

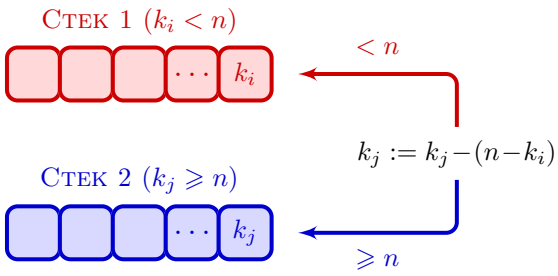
Теперь упакуем в один ящик всё кофе сорта `A` и добавим, если необходимо, кофе сорта `B`. После этого осталось $(m-1) \cdot n$ пачек

кофе $(m-1)$ сортов и мы можем повторить процедуру для задачи с *меньшим* числом сортов кофе.

Для решения подзадачи 1 в последовательности $k_1, k_2, \ldots, k_m$ найдём два числа k_i и k_j, для которых $k_i < n$ и $k_j \geqslant n$. Заполним ящик двумя сортами кофе i и j, после этого заменяем значение k_i на 0, а значение k_j — на $k_j - (n - k_i)$. Продолжаем этот процесс до тех пор, пока остаётся хотя бы одно число $k_j > 0$. Это решение проходит тесты для всех $m \leqslant 40\,000$, его сложность — $O(m^2)$.

Для решения подзадачи 2 оптизимируем поиск чисел k_i и k_j, для которых $k_i < n$ и $k_j \geqslant n$. Организуем два стека: в первом будем хранить числа k_i последовательности, которые меньше n, во втором — все числа k_i, которые не меньше n. На каждом шаге из первого и второго стека забираем последние («верхние») числа k_i и k_j. Из этих чисел формируем ящик с двумя сортами кофе i и j. После этого заменяем значение k_j на $k_j - (n - k_i)$. Если теперь $k_j < n$, помещаем это число в первый стек; иначе — во второй стек. Продолжаем этот процесс, пока первый стек непустой. Теперь из второго стека извлекаем числа, равные n, пока не опустеет второй стек.

Сложность этого решения — $O(m)$.



Муниципальный этап, 2016-2017

Задача 1. Билеты

Автор задачи : *фольклор.*
Разработчик : *Киндер М.И.*
Разбор задачи: *Киндер М.И.*

Основные темы этой простой задачи — работа с циклами и условными операторами.

Начальное значение искомой суммы s считаем равным 0. В цикле просматриваем значения всех чисел, обозначающих возраст участников группы. Если возраст меньше 11, то значение суммы s не изменяется, и мы переходим к просмотру следующего числа. Если возраст меньше 19, значение s увеличиваем на 50, в противном случае увеличиваем s на 100.

Приведем фрагмент кода на языке Pascal:

```
for i := 1 to n do begin
    read(d);
    if d < 11 then continue;
    if d < 19 then inc(s, 50) else inc(s, 100);
end;
write(s);
```

Задача 2. Палиндром

Автор задачи : *фольклор.*
Разработчик : *Киндер М.И.*
Разбор задачи: *Киндер М.И.*

Основные темы задачи — работа с массивами цифр.

Сначала отметим, что в десятичной записи любого палиндрома равноудаленные от концов цифры совпадают. Из n цифр заданного числа можно будет составить палиндром только в том случае, когда каждая цифра от 0 до 9 входит в этот набор *чёт-*

ное количество раз (возможно, ни разу), или же ровно одна цифра входит нечётное число раз, а все остальные — чётное число. Например, последнему условию удовлетворяют числа 1112002 и 2200550, из которых можно составить палиндромы 2011102 и 2050502. В записи этих палиндромов цифра, входящая нечётное число раз, находится ровно посередине. Исключение составляют числа, которые содержат чётное число нулей и ровно одну ненулевую цифру. К ним относится, в частности, число 10000, из которого составить палиндром невозможно.

Реализация алгоритма требует аккуратной проверки всех этих условий, а также умения работать с отдельными цифрами-символами исходного набора. Сложность этого алгоритма — $O(n)$ операций.

Задача 3. Сумма факториалов

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Основные темы задачи — применение жадного алгоритма и перевод чисел в специальную систему счисления.

Первое решение. Воспользуемся жадным алгоритмом. Из данного числа x вычтем *наибольший* возможный факториал $n!$ и к полученной разности применим те же рассуждения: из полученной разности снова вычтем наибольший факториал, и так далее. Количество вычитаемых факториалов и будет искомым ответом в задаче.

Реализация этого алгоритма несложная. Корректность жадного алгоритма следует из неравенства.

$$1 \cdot 1! + 2 \cdot 2! + \ldots + (n-1) \cdot (n-1)! < n!\,.$$

Действительно, если для заданного числа x выполняется неравенство $n! \leqslant x < (n+1)!$, то представление x в виде суммы наименьшего количества факториалов обязательно содержит слагаемое $n!$. В противном случае, как следует из приведённого неравенства, сумма «остальных» факториалов будет меньше x. (Ес-

ли факториал $k!$ входит с коэффициентом $c_k > k$, то слагаемое $c_k \cdot k! = (k+1)! + (c_k - k - 1)\, k!$ можно заменить суммой меньшего числа слагаемых.)

Второе решение основано на применении так называемой *факториальной системы счисления*, её основанием служит последовательность натуральных чисел: $1!$, $2!$, $\ldots$, $k!$. В факториальной системе натуральное число x представляется единственным образом в виде

$$x = c_1 \cdot 1! + c_2 \cdot 2! + \ldots + c_n \cdot n!\,, \quad \text{где } 0 \leqslant c_k \leqslant k.$$

Отсюда понятно, что наименьшее количество факториалов, при сложении которых получается данное число x, равно сумме «цифр» c_k в записи числа x в факториальной системе счисления, то есть равно $s = c_1 + c_2 + \ldots + c_n$.

Осталось научиться переводить число в факториальную систему. Оказывается, для этого не обязательно вычислять факториалы натуральных чисел. Действительно, каждое из чисел $k!$ делится на все натуральные числа от 1 до k, поэтому слагаемые вида $c_k \cdot k!$ делятся на 2 при любом $k \geqslant 2$. Значит, число c_1 равно остатку от деления x на 2. Разделив нацело x на 2, то есть отбрасывая слагаемое $c_1 \cdot 1!$, применим те же рассуждения к числу x `div` 2. Теперь число c_2 равно остатку от деления x `div` 2 на 3. Этот процесс продолжаем до тех пор, пока частное от деления на очередное число k не станет равным нулю.

Ниже приведен фрагмент кода на языке Pascal, реализующего этот алгоритм.

```
s := 0;    k := 2;
while (n > 0) do begin
    s := s + x mod k;
    x := x div k;
    inc(k);
end;
write(s);
```

Задача 4. НОК

Автор задачи : *Киндер М.И.*
Разработчик : *Киндер М.И.*
Разбор задачи: *Киндер М.И.*

Основные темы задачи — комбинаторика, факторизация натуральных чисел и бинарный алгоритм возведения в степень. Возможны также частные решения, основанные на переборе.

Для решения ПОДЗАДАЧИ 1 с небольшими значениями k и m ($2 \leqslant k \leqslant 3$, $1 \leqslant m \leqslant 100$) все требуемые наборы можно найти несложным перебором. Такое решение проходит тесты на 30 баллов.

Для решения ПОДЗАДАЧИ 2 воспользуемся следующими комбинаторными рассуждениями. Пусть m — наименьшее общее кратное всех чисел нашего набора, и пусть $m = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_r^{\alpha_r}$ — разложение на простые множители. Условию задачи удовлетворяют все наборы чисел $\alpha_i = p_1^{\alpha[i,1]} p_2^{\alpha[i,2]} \dots p_r^{\alpha[i,r]}$, $1 \leqslant i \leqslant k$, у которых неотрицательные показатели $\alpha[i,s]$ степеней простых чисел p_s не превосходят α_s и для которых выполняются условия

$$\max(\alpha[1,s], \alpha[2,s], \dots, \alpha[k,s]) = \alpha_s\,,$$

для всех $1 \leqslant s \leqslant r$. Подсчитаем число наборов по каждому показателю α отдельно. Количество наборов, которые удовлетворяют условиям $0 \leqslant \alpha_i \leqslant \alpha$ для всех i от 1 до k, равно

$$(\alpha+1)^k - \alpha^k\,.$$

В самом деле, число α_i может принимать $(\alpha+1)$ значений — это все неотрицательные числа от 0 до α, и по правилу произведения количество таких наборов равно $(\alpha+1)^k$. Осталось исключить из этого множества наборы, для которых $\max(\alpha_1, \alpha_2, \dots, \alpha_k) < \alpha$. Для таких наборов $0 \leqslant \alpha_i \leqslant \alpha-1$ для всех i от 1 до k, поэтому их количество равно α^k. Значит, количество требуемых наборов равно $(\alpha+1)^k - \alpha^k$. Теперь по правилу произведения число наборов по всем показателям α_s равно:

$$L(m,k) = \prod_{i=1}^{r} [(\alpha_i+1)^k - \alpha_i^k]\,.$$

Например, для $m = 10 = 2^1 \cdot 5^1$ и $k = 2$ получаем

$$L(10, 2) = [(1 + 1)^2 - 1^2] \cdot [(1 + 1)^2 - 1^2] = 9$$

упорядоченных наборов из двух чисел, у которых НОК равен 10. Алгоритмическая сложность такого решения — $O(Fact(m) + k)$. Такое решение проходит тесты ещё на 35 баллов.

Для получения полного решения ($1 \leqslant k \leqslant 10^{18}$) можно воспользоваться бинарным алгоритмом возведения в степень при вычислении величин $(\alpha + 1)^k$ и α^k. Алгоритмическая сложность такого решения — $O(Fact(m) + \log k)$.

Задача 5. Наименьший палиндром

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Основные темы задачи — работа с массивами цифр.

Сначала отметим, что в десятичной записи любого палиндрома равноудаленные от концов цифры совпадают. Из n цифр заданного числа можно будет составить палиндром только в том случае, когда каждая цифра от 0 до 9 входит в этот набор *чётное* количество раз (возможно, ни разу), или же ровно одна цифра входит нечётное число раз, а все остальные — чётное число. Например, последнему условию удовлетворяют числа 1112002 и 2200550, из которых можно составить палиндромы 2011102 и 2050502. В записи этих палиндромов цифра, входящая нечётное число раз, находится ровно посередине. Исключение составляют числа, которые содержат чётное число нулей и ровно одну ненулевую цифру. К ним относится, в частности, число 10000, из которого составить палиндром невозможно.

Теперь составим наименьший палиндром. Подсчитаем количество вхождений $d[m]$ каждой цифры m от 0 до 9. Наименьшую ненулевую цифру поставим на первое место палиндрома. Сформируем «первую» половину палиндрома. Сначала выводим на печать каждую из цифр m в точности $d[m]$ `div` 2 раз. «Середина» наименьшего палиндрома заполняется цифрой m, которая входит

в запись исходного числа нечётное число раз. «Вторая» половина палиндрома получается симметрично «первой».

Реализация алгоритма требует аккуратной проверки всех этих условий, а также умения работать с отдельными цифрами-символами исходного набора. Сложность алгоритма — $O(n)$ операций.

Задача 6. Заводы в городе

Автор задачи :	*фольклор*
Разработчик :	*Киндер М.И.*
Разбор задачи:	*Киндер М.И.*

Основные темы задачи — вычисление функции суммы расстояний и нахождение максимума функции. Возможны также частные решения, основанные на неполном переборе или тернарном поиске искомой точки вдоль границы области.

Обозначим через $sum(t)$ сумму расстояний от точки t до всех точек многоугольника, а через $F(X, Y)$ — функцию двух точек X и Y, которая с помощью тернарного поиска на отрезке XY находит наибольшее значение суммы расстояний $sum(t)$.

Решение подзадачи 1. Пусть X — искомая точка в треугольнике ABC. Рассмотрим одну из сторон треугольника, например, AB. Начиная от точки A, будем двигаться по направлению к точке B вдоль отрезка AB. Для каждой промежуточной точки X отрезка AB будем вычислять значение функции $F(X, C) = \max sum(t)$ на отрезке XC. Промежуточные точки X тоже будем выбирать с помощью тернарного поиска по отрезку AB. Среди найденных значений функции $F(X, C)$, конечно, будем запоминать только наибольшее.

Таким образом, фактически будет найдено наибольшее значение $sum(t)$ в треугольнике ABC и на его границе. Сложность алгоритма — $O(k \cdot it^2)$, где k — количество запросов (число районов), it — количество итераций для вычисления наибольшего значения функции $F(X, Y)$ вдоль выбранного отрезка.

Решение подзадачи 2. Идею решения подзадачи 1 можно применить в более общей ситуации, когда количество вершин

многоугольника не превышает 20. Выберем вершину A_1 многоугольника и для каждого треугольника $A_1A_iA_{i+1}$ $(2 \leqslant i \leqslant n-1)$ найдем наибольшее значение функции $sum(t)$, как это указано в решении подзадачи 1. Сложность алгоритма — $O(k \cdot n^2 \cdot it^2)$, где n — количество вершин многоугольника.

Решение подзадачи 3. Будем искать наибольшее значение функции $sum(t)$ только вдоль границы многоугольника. Другими словами, на каждой стороне многоугольника A_iA_{i+1} $(1 \leqslant i \leqslant n, A_{n+1} = A_1)$ найдём максимум функции $sum(t)$, а затем среди них выберем самое наибольшее значение. (Корректность этого алгоритма следует из решения подзадачи 4.) Сложность алгоритма — $O(k \cdot n^2 \cdot it)$.

Решение подзадачи 4. Наибольшее значение функции $sum(t)$ достигается в одной из вершин многоугольника. Доказательство этого факта можно провести с помощью несложных геометрических рассуждений, и мы его здесь опускаем. Таким образом, для полного решения задачи достаточно перебрать все вершины многоугольника и для каждой из них вычислить значение функции $sum(t)$, а потом выбрать среди них наибольшее. Сложность алгоритма — $O(k \cdot n^2)$.

Региональный этап, 2016-2017

Задача 1. Кампус

Автор задачи : *Станкевич А.С.*
Разработчик : *Станкевич А.С.*
Разбор задачи: *Станкевич А.С.*

Основные темы задачи — модульная арифметика, использование формулы. Возможные частичные решения основаны на линейном поиске и более простых формулах для частных случаев.

Для решения задачи сначала вычислим суммарное количество комнат в одном подъезде. Число этажей, номер которых кратен k, равно целой части от деления n на k. Значит, общее число комнат в подъезде равно $p = (n \texttt{ div } k) \cdot x + (n - (n \texttt{ div } k)) \cdot y$. (Запись $a \texttt{ div } b$ обозначает целую часть от деления a на b.) Заменив в каждом запросе числа a_i на числа $(a_i - 1) \texttt{ mod } p$, будем считать, что все комнаты находятся в первом подъезде и нумерация комнат начинается с 0.

Если бы все этажи имели по y комнат, то этаж, на котором находится комната, был бы равен $a_i \texttt{ div } y$. Это решение подходит для третьей подзадачи.

Для получения номера этажа в общем случае необходимо действовать следующим образом. Разобьем все этажи на блоки по k подряд идущих (последний блок может содержать менее k этажей). Каждый такой блок содержит $b = x + (k - 1) \cdot y$ комнат. Значит, ниже искомой комнаты находится $a_i \texttt{ div } b$ полных блоков, а в блоке, в котором она находится, ниже нее находится $\texttt{min}((a_i \texttt{ mod } b) \texttt{ div } y, k - 1)$ этажей. Суммируя эти значения, получаем номер этажа, на котором находится комната.

Приведем фрагмент кода на языке C++, вычисляющий номер этажа для одной комнаты.

```
long long p = (n / k) * x + (n - n / k) * y;
a = (a - 1) % p;
long long b = x + (k - 1) * y;
long long res = a / b * k + min((a % b) / y, k - 1) + 1;
```

Задача 2. Калькулятор

Автор задачи :	*Станкевич А.С.*
Разработчик :	*Станкевич А.С.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — динамическое программирование, жадные алгоритмы. Возможные частичные решения основаны на линейном переборе вариантов, разборе отдельных случаев.

Первое решение этой задачи основано на идее динамического программирования. Заметим, что все три описанные в условии операции являются *убывающими* — чем больше значение n, тем меньше значение результата.

Пусть $dp[i][j][k]$ — минимальное число, которое можно получить из n после нажатия i раз на кнопку **A**, j раз на кнопку **B** и k раз на кнопку **C**. Тогда $dp[0][0][0] = n$, а ответом в задаче будет значение $dp[a][b][c]$.

Значение $dp[i][j][k]$ при всех i, j, k будем использовать для возможного уменьшения чисел $dp[i+1[j][k]$, $dp[i][j+1][k]$ и $dp[i][j][k+1]$, возникающих при выполнении операций **A**, **B** и **C** соответственно.

Приведем фрагмент кода на языке C++, вычисляющий элементы массива dp.

```
dp[0][0][0] = n;
for (int i = 0; i <= a; i++) {
  for (int j = 0; j <= b; j++) {
    for (int k = 0; k <= c; k++) {
      if (i < a)
        dp[i+1][j][k] = min(dp[i+1][j][k], dp[i][j][k]/2);
      if (j < b)
        dp[i][j+1][k] = min(dp[i][j+1][k], (dp[i][j][k]+1)/2);
      if (k < c)
        dp[i][j][k+1] = min(dp[i][j][k+1], (dp[i][j][k]-1)/2);
    }
  }
}
cout << d[a][b][c];
```

Второй подход заключается в том, чтобы применить жадный алгоритм. Проанализируем действия пользователя с конца.

Сначала определим максимально возможное значение t, из которого можно получить заданное число X после нажатия какой-нибудь кнопки. Если X было получено в результате операции **A**, то это максимальное значение t равно $2X+1$. Если число X было получено в результате операции **B**, то максимальное значение t равно $2X$. Наконец, если X появилось после нажатия кнопки **C**, то $t = 2X + 2$.

Теперь уже несложно проверить, что максимальное значение t, из которого можно получить X, достигается после нескольких нажатий кнопок, причём оптимально в конце нажимать кнопку **C**, перед ней кнопку **A**, а в начале — кнопку **B**. Разворачивая действия пользователя обратно, получаем, что всегда оптимально нажимать сначала кнопку **B**, затем — кнопку **A**, в конце — кнопку **C**. Применив такую последовательность операций к исходному числу, получим минимальный возможный результат.

При решении подзадач 2 и 3 можно использовать описанные идеи не полностью: например, возможно учесть только нажатие кнопки **C** после нажатия кнопки **A**, или учесть только нажатие кнопки **B** до нажатия кнопки **A**.

Для решения подзадачи 1 достаточно воспользоваться полным перебором вариантов.

Задача 3. Размещение данных

Автор задачи :	*Станкевич А.С.*
Разработчик :	*Станкевич А.С.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — алгоритмы на графах, обход в глубину. Возможные частичные решения основаны на использовании перебора, линейного поиска, менее эффективных алгоритмах на графах.

Математическая формулировка задачи состоит в следующем.

Задан неориентированный граф из n вершин и m рёбер. Требуется пометить такое *минимальное* число k вершин, что при уда-

лении любого ребра существует путь от каждой вершины графа до одной из помеченных. Кроме того, требуется также определить число способов пометить таким образом k вершин.

В подзадаче 1 рассматриваются графы с небольшим числом вершин и рёбер: $1 \leqslant n \leqslant 10, 1 \leqslant m \leqslant 45$. Для ее решения сделаем полный перебор всех множеств и для каждого из них проверим, подходит ли оно.

В подзадаче 2 входные данные связаны условием $m = n-1$. Связный граф, у которого число рёбер на единицу меньше числа вершин, является *деревом*. Оптимальный способ пометить вершины только один — необходимо пометить все листья дерева, то есть вершины степени 1.

Действительно, не пометить лист нельзя, поскольку после удаления ребра, соединяющего этот лист с деревом, из него нельзя будет достичь никакой другой вершины. С другой стороны, после удаления любого ребра в обеих получившихся компонентах связности остается хотя бы один из листьев исходного дерева, поэтому всех отмеченных листьев достаточно для выполнения условия задачи.

Теперь разберём идею полного решения задачи. Напомним, что *мостом* называется ребро, удаление которого увеличивает число компонент связности. Поскольку при удалении любого ребра, не являющегося мостом, граф остается связным, каждая помеченная вершина будет по-прежнему достижима из любой другой вершины графа, и значит, помеченные вершины не могут входить в мосты графа.

Удалим все мосты и сожмём в одну вершину каждую полученную компоненту связности, после этого восстановим мосты. Получившийся граф будет *деревом*, а для него помеченными вершинами, как мы уже знаем, должны быть все *листья*. Поскольку листья дерева соответствуют компонентам связности, полученным после удаления мостов, для решения достаточно выбрать любую вершину в каждой *такой* компоненте и пометить её.

Таким образом, количество помеченных вершин k равно числу листьев в специальном дереве, или числу компонент связности, которые получаются после удаления всех мостов графа, причём учитывать нужно только те компоненты, из которых выходил ровно один мост. Число способов выбрать требуемые k вершин

равно произведению размеров этих компонент связности.

Для решения подзадачи 3 достаточно найти мосты перебором всех рёбер графа и соответствующей проверкой на связность компонент после удаления.

Для решения подзадачи 4 нужно воспользоваться эффективным алгоритмом поиска мостов в графе за время $O(n + m)$. (См., например, Кормен Т., Лейзерсон Ч. и др. *Алгоритмы. Построение и анализ*, 2005, гл. 22.)

Задача 4. Полезные ископаемые

Автор задачи :	*Наумов С.С.*
Разработчик :	*Наумов С.С.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — двоичный поиск, комбинаторика и паросочетания в графах. Возможные частичные решения основаны на переборе вариантов, жадных алгоритмах, моделировании, алгоритмах поиска максимального паросочетания, алгоритмах поиска максимального потока.

Для полного решения задачи нужно воспользоваться *леммой Холла*. Рассмотрим двудольный граф с долями X и Y. Пусть $A \subset X$ — множеств вершин из доли X, и пусть $N(A) \subset Y$ — множество соседей вершин из множества A. Лемма Холла утверждает: в графе существует насыщающее долю X паросочетание тогда и только тогда, когда для любого A соответствующее множество $N(A)$ содержит вершин не меньше, чем A. (Доказательство леммы можно найти, например, в кн. Липский В. *Комбинаторика для программистов*. 1988. с. 164.)

Применим двоичный поиск по числу принятых на планету партий, а также по числу роботов последней, неполной, партии. Пусть зафиксированы все партии, которые будут приняты на планету, а также количество роботов из следующей партии. Необходимо научиться проверять, можно ли таким образом распределить роботов по клеткам, чтобы в каждой клетке оказалось не более q роботов.

Рассмотрим сначала задачу при $q = 1$. Построим двудольный граф, у которого вершины одной доли — это роботы, а верши-

ны другой доли — клетки. Соединим робота и клетку ребром, если он может добраться до этой клетки. Заметим, что распределение роботов по клеткам соответствует насыщающему первую долю паросочетанию в получившемся графе. Используя алгоритмы поиска максимального паросочетания, можно проверить, существует ли в графе такое паросочетание. Этот подход позволяет решить подзадачи 1–3.

При $q > 1$ во второй доле каждой клетке соответствует не одна, а q вершин. Задача, по-прежнему, сводится к проверке существования насыщающего первую долю паросочетания, но количество вершин слишком велико, и поэтому алгоритмы поиска максимального паросочетания не помогут решить даже подзадачу 4.

Будем проверять наличие паросочетания, используя лемму Холла. Рассмотрим подмножество роботов. Заметим, что имеет смысл рассматривать только целые группы, а также если рассматривается группа с мобильностью m, то осмысленно включить в подмножество и все группы на той же базе с мобильностью, не превышающей m, поскольку это увеличивает размер A, но не меняет размер $N(A)$. Таким образом, если условие леммы Холла выполняется для такого выбора подмножеств, то оно тем более выполняется и для остальных подмножеств.

Итак, отсортируем на каждой базе принятые партии роботов по неубыванию мобильности. Переберём для каждой базы несколько первых партий роботов и для них построим множество достижимых клеток. Если количество клеток в этом множестве, умноженное на q, меньше общего числа роботов в выбранных партиях, то мы нашли контрпример к лемме Холла, и распределить роботов по полю нельзя. Если для всех множеств контрпример не найден, то искомое распределение возможно.

Осталось понять, как вычислить суммарное количество клеток, достижимых роботами. Для каждой базы это множество представляет собой квадрат, достижимой самой мобильной группой с этой базы. Для подсчёта воспользуемся формулой включения-исключения: переберём все подмножества баз, для каждого подмножества вычислим пересечение всех искомых квадратов и учтем его со знаком «+», если выбрано нечётное число баз, либо со знаком «−», если выбрано чётное число баз.

Оценим время работы алгоритма. Пусть на i-ю базу отправлено g_i групп роботов. Тогда необходимо перебрать $g_1 \cdot g_2 \cdot \ldots \cdot g_s$ вариантов выбора групп роботов, а для каждого варианта ещё 2^s вариантов в формуле включения-исключения. Произведение при фиксированной сумме максимально для равных сомножителей, поэтому итоговая оценка времени работы — $O((2t/s)^s)$.

Для решения подзадач с $s = 1$ можно использовать жадный алгоритм, отдавая каждому роботу наиболее удаленную из свободных клеток, до которых он может добраться. Этот алгоритм можно также обобщить и для решения подзадач с $s = 2$.

Задача 5. Автоматизированное управление доставкой

Автор задачи :	*Станкевич А.С., Лопатин А.С.*
Разработчик :	*Станкевич А.С., Лопатин А.С.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — линейный перебор вариантов, вывод формулы. Возможные частичные решения основаны на полном переборе вариантов, динамическом программировании.

Заметим сначала, что пакет, отправляемый в муниципальный почтовый центр, может иметь любой вес от a до $(a + k - 1)$ *кг*. Вес $(a + i)$ *кг* может получиться, например, после приёма $(a - 1)$ пакета по 1 *кг* и одного пакета в $(i + 1)$ *кг*.

Проверим теперь, можно ли добиться, чтобы вес контейнера при перевозке его в региональный сортировочный центр был равен b *кг*. Вычислим максимальное количество пакетов, которое могло быть перевезено в муниципальный почтовый центр перед отправкой контейнера; это количество равно $c = b$ `div` a. Минимальный суммарный вес всех этих пакетов равен $c \cdot a$, а максимальный вес — $c \cdot (a + k - 1)$, все промежуточные значения достижимы. Значит, если $c \cdot a \leqslant b \leqslant c \cdot (a + k - 1)$, ответ равен b.

В противном случае доставка c пакетов, даже если они все имеют максимальный возможный вес $a + k - 1$, не приводит к отправке контейнера. Значит, для его отправки необходим $c + 1$ пакет. Минимальный вес $(c+1)$ пакета равен $(c+1) \cdot a$. Значит, в этом случае ответ равен $(c+1) \cdot a$.

```
cnt = y / x;
lo = cnt * x;
hi = cnt * (x + k - 1);
if (lo <= y && y <= hi)
    ans = y;
else
    ans = x * (cnt + 1);
```

Задача 6. Большой линейный коллайдер

Автор задачи :	*Лопатин А.С., Станкевич А.С.*
Разработчик :	*Лопатин А.С., Станкевич А.С.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — линейный проход, структуры данных, комбинаторика. Возможное частичное решение основано на полном переборе вариантов, моделировании.

Построим по описанию частиц *скобочную* последовательность: частице e^+ сопоставим открывающую скобку, а частице e^- — закрывающую скобку. Частицы взаимно уничтожаются тогда и только тогда, когда они соответствуют парным скобкам.

Найдем для каждой частицы момент, когда она будет уничтожена. Для этого пройдем по построенной скобочной последовательности слева направо, и будем поддерживать стек открытых, но пока не закрытых скобок. Если встречаем открывающуюся скобку, поместим ее в стек. Если встречаем закрывающуюся скобку, то посмотрим, пуст ли стек. Если стек пуст, то у этой скобки нет парной, и соответствующая частица не будет уничтожена. Если же стек не пуст, то скобка на вершине стека является парной к рассматриваемой. Они уничтожатся, когда окажутся посередине отрезка между ними; если их начальные координаты равны x_i и x_j, то это произойдет в точке $\frac{1}{2}(x_i + x_j)$ через время $\frac{1}{2}|x_j - x_i|$.

Теперь отсортируем совместно все те моменты времени, когда происходят аннигиляции частиц, а также моменты запросов о количестве частиц в коллайдере. При этом для каждого запроса учитываем предшествующие им аннигиляции в соответствии с условием задачи.

Исходно в коллайдере $c = n$ частиц. Если происходит уничтожение пары частиц, значение c уменьшается на 2. Если встречаем запрос о количестве частиц, выводим текущее значение c.

Чтобы не проверять на совпадение действительные значения, моменты аннигиляции и времена в запросах нужно умножить на 2 и перейти к целым числам.

Задача 7. Силовые поля

Автор задачи :	*Корнеев Г.А.*
Разработчик :	*Корнеев Г.А.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — структуры данных, сканирующая прямая. Возможные частичные решения основаны на использовании перебора, сортировки.

Пусть требуется выбрать k полей, и пусть минимальная ширина выбранного поля равна x_i. Рассмотрим все поля, у которых ширина равна хотя бы x_i. Среди них оптимально выбрать k полей с максимальными значениями y_j. На базе этого наблюдения получим следующее решение задачи.

Отсортируем все прямоугольники (силовые поля) по убыванию x_i. Рассмотрим первые k прямоугольников, поместим их в структуру данных `T`, позволяющую добавлять элементы, получать и удалять минимальный по значению высоты элемент y_j.

Вычислим площадь пересечения прямоугольников как произведение текущего значения ширины x_i и минимального значения y_m. Затем перейдем к следующему прямоугольнику. Рассмотрим его высоту y_j. Если оно меньше минимального значения в `T`, то нет смысла брать этот прямоугольник, его использование будет заведомо хуже уже рассмотренных вариантов. Иначе удалим из `T` прямоугольник с минимальным y_m и обновим площадь пересечения прямоугольников, вычислив произведение x_i и нового минимального значения y'_m. Рассмотрев таким образом все прямоугольники, получим оптимальный ответ.

В качестве структуры `T` можно использовать `std::set` из `C++`, приоритетную очередь или дерево отрезков.

Оценка времени работы решения — $O(n \log n)$.

Задача 8. Повышение квалификации

Автор задачи : *Пересадин И.В.*
Разработчик : *Пересадин И.В.*
Разбор задачи: *Станкевич А.С.*

Основные темы задачи — работа с деревьями, обход в глубину, сортировка событий. Возможные частичные решения основаны на использовании линейного перебора вариантов.

Переформулируем задачу математически. Задано подвешенное дерево, вершины которого пронумерованы таким образом, что родитель любой вершины имеет номер меньше, чем эта вершина. Есть несколько требований вида (p_i, k_i), где p_i — вершина, а k_i — целое число. Требуется выбрать такой отрезок минимальной длины $[L, R]$, чтобы для каждой вершины p_i существовала вершина с номером в выбранном отрезке, для которой p_i является предком k_i-го уровня.

Рассмотрим сначала недостаточно эффективное решение, которое тем не менее приблизит нас к пониманию полного решения задачи. Для каждого требования (p_i, k_i) составим список L_i номеров вершин, которые являются потомками уровня k_i вершины p_i. Необходимо выбрать несколько вершин так, чтобы из каждого множества L_i была выбрана хотя бы одна вершина, а разность между максимальным и минимальным номерами выбранных вершин была как можно меньше.

Зафиксируем выбранную вершину с минимальным номером, обозначим этот номер через x. Заметим, что для каждого списка L_i оптимально выбрать вершину с минимальным номером y_i, таким что $y_i \geqslant x$. Перебрав все варианты выбора вершины x, например, из списка L_1, найдя для каждого из вариантов максимум из вершин y_i, и выбрав минимум из всех вариантов выбора x, мы получим ответ на задачу.

Если построить списки L_i в явном виде, перебрать все варианты вершины x и искать каждый раз вершины y_i просмотром всех элементов каждого из списков, описанное решение будет иметь время работы $O(n^3)$. Такое решение подходит для ПОДЗАДАЧИ 1.

Заметим, что построение списков L_i требует $O(n^2)$ времени; кроме того, их суммарный размер равен $O(n^2)$. Отсортируем вер-

шины каждого из списков по номеру и заметим, что при увеличении x значения номеров y_i не уменьшаются. Таким образом, в каждом списке содержится указатель на y_i, и можно, перемещая указатели только вперед по спискам, обработать все варианты выбора x за суммарное время $O(n^2)$. Время работы такого решения $O(n^2 \log n)$ из-за сортировки списков, а в случае применения корзинной или цифровой сортировки время улучшается до $O(n^2)$. Оба варианта подходят для решения ПОДЗАДАЧИ 2.

Наконец, перейдём к описанию полного решения. Обойдём дерево в глубину, присвоив каждой вершине вспомогательные значения в порядке входа в эти вершины. Объединим вершины на фиксированной глубине h в список $V[h]$ в порядке входа. Заметим, что для каждого i список L_i состоит из вершин, идущих подряд в списке $V[h]$, то есть соответствует отрезку в этом списке. Заметим, что на каждом уровне такие отрезки либо вложены один в другой, либо не пересекаются.

Если в одном из таких отрезков A содержится другой отрезок B, то ограничения на отрезок A можно убрать: если в ответе будет вершина из отрезка B, то она будет также принадлежать отрезку A. Таким образом, на каждой глубине оставим набор непересекающихся отрезков, в каждом из которых должна быть выбрана хотя бы одна вершина.

Ключевая идея оптимизации решения до времени работы $O(n \log n)$ заключается в следующем. Рассмотрим m_i — максимальный номер вершины в множестве L_i. Обозначим через m_{min} минимальное значение среди m_i. Заметим, что нельзя выбирать значение x больше, чем m_{min}, поскольку в этом случае будут списки L_i, из которых мы не сможем выбрать ни одной вершины. И наоборот, если у нас выбрано значение $x \leqslant m_{min}$, то из каждого списка можно выбрать вершину с номером не меньше x так, чтобы все ограничения были удовлетворены.

Поскольку на каждой глубине отрезки, соответствующие оставленным спискам L_i, не пересекаются, суммарное количество вершин в них не больше, чем n. Отсортируем вершины во всех списках L_i.

Рассмотрим значения y_i, выбранные для некоторого значения x. Пусть y_{max} равен максимуму среди них. Для фиксированного x нужно рассмотреть в качестве кандидата на ответ отрезок с x

до y_{max} — это минимальный отрезок, для которого выполнены все ограничения. Для того, чтобы пересчитать y_i при увеличении x на единицу, достаточно посмотреть, была ли выбрана вершина, для которой выполнялось равенство $y_i = x$. Если такого значения y_i не было, то ни один из номеров y_i пересчитывать не придется. Иначе, нужно заменить y_i на следующую вершину в списке L_i; поскольку вершины в нём отсортированы по номерам, это будет новая минимальная вершина с номером не меньше x на этой глубине, удовлетворяющая ограничению i.

Таким образом, перебирая x от 1 до m_{min}, и пересчитывая множество y_i, мы рассмотрим все отрезки, которые выполняют все ограничения, и у которых нельзя сдвинуть правую границу влево. Ответом будет самый короткий из всех таких отрезков.

Время работы решения — $O(n \log n)$, удаление лишних списков и сортировка списков L_i требует $O(n \log n)$ времени, а остальная часть решения выполняется за $O(n)$.

Задача 9. Гипертроичность

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И., Балакирев М.*
Разбор задачи:	*Киндер М.И.*

Основные темы задачи — динамическое программирование, рекурсия. Возможные частичные решения основаны на использовании линейного перебора вариантов.

Пусть $T(n)$ — количество гипертроичных представлений числа n. Рассмотрим нескольких первых значений этой функции и убедимся, что

$$T(3m+1) = T(m), \quad T(3m+2) = T(m).$$

Другими словами, если $n \not\vdots 3$, то $T(n) = T(n \texttt{ div } 3)$. Действительно, в этом случае гипертроичная запись числа n заканчивается одной или двумя единицами. Отбросив их, получим число, кратное 3. Разделив его на 3, получим гипертроичную запись числа n `div` 3. Например, из равенства $20 = 9{+}3{+}3{+}3{+}1{+}1$ получаем гипертроичную запись числа $6 = 3 + 1 + 1 + 1$.

Если же число n кратно 3 и $n = 3m+3$, то его гипертроичная запись заканчивается тройкой или суммой трёх единиц.

Если запись n оканчивается тройкой, разделим все слагаемые на 3 и получим гипертроичную запись числа n `div` $3 = m+1$. Если же запись числа n заканчивается суммой трёх единиц, то, отбросив одну единицу, мы получим гипертроичное представление числа $3m+2$, причем $T(3m+2) = T(m)$.

Итак, для числа $n = 3m+3$ получаем

$$T(3m+3) = T(m+1) + T(m).$$

Для вычислении функции $T(n)$ теперь можно воспользоваться рекурсивной процедурой. Это даёт решение ПОДЗАДАЧИ 1.

Для решения ПОДЗАДАЧИ 2 и для ускорения вычисления $T(n)$ проведем нехитрую оптимизацию. Используя идею динамического программирования, будем запоминать вычисленные значения $T(n)$ для всех $n \leqslant 10^6$, а для вычислений при $n > 10^6$ будем использовать ранее сохранённые значения функции T.

Задача 10. Будильники

Автор задачи :	*Киндер М.И.*
Разработчик :	*Киндер М.И., Балакирев М.*
Разбор задачи:	*Киндер М.И.*

Основные темы задачи — префикс-суммы, сортировка. Возможные частичные решения основаны на использовании неэффективного вычисления сумм.

Отметим на одном циферблате положения часовых стрелок всех будильников. Граница циферблата разобьется на n дуг, как это показано на рис. 7. Занумеруем дуги по кругу. Пусть часовая стрелка проходит эти дуги за время $x_1, x_2, \ldots, x_n$ соответственно (некоторые x_i, возможно, нулевые). Если установить на всех часах время, соответствующее положению внутри дуги, то каждая часовая стрелка пройдет через *начало* этой дуги, и значит, суммарное время перевода будет больше, чем если установить все часы на начало дуги.

Для решения ПОДЗАДАЧИ 1 достаточно вычислить суммарное время перевода S_i для каждого i-го будильника. Всего бу-

дильников n, и значит, для вычисления всех S_i требуется порядка $O(n^2)$ операций. Осталось найти наименьшее в массиве (S_i).

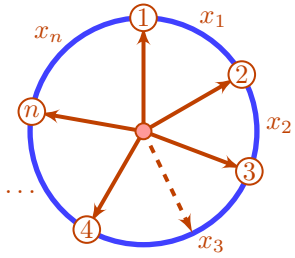


Рис. 7. Пример расположения часовых стрелок

Для полного решения задачи оптимизируем подсчёт сумм S_i времени перевода. Пусть S_i — суммарное время, необходимое для установки всех часов на начало i-ой дуги. Например, время перевода на начало первой дуги равно x_n для будильника № n и $x_2 + x_3 + \ldots + x_n$ для будильника № 2. Тогда можно выразить все величины

$$\begin{aligned} S_1 &= x_2 + 2x_3 + \ldots + (n-2)x_{n-1} + (n-1)x_n\,; \\ S_2 &= x_3 + 2x_4 + \ldots + (n-2)x_n \quad + (n-1)x_1\,; \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ S_n &= x_1 + 2x_2 + \ldots + (n-2)x_{n-2} + (n-1)x_{n-1}\,. \end{aligned}$$

Отсортируем исходный массив показаний времени и вычислим элементы массива (x_i). Заметим, что разность соседних чисел массива (S_i) теперь равна

$$S_1 - S_2 = (x_2 + x_3 + \ldots + x_n) - (n-1)x_1, \text{ т.е.}$$

$$S_2 = S_1 + nx_1 - S, \text{ где } S = \sum_{i=1}^{n} x_i\,.$$

Сумма S равна 0, только если все будильники показывают одинаковое время, или же равна 12 часам, если это не так. Теперь для вычисления всех сумм S_i требуется порядка $O(n)$ операций. Осталось найти наименьшее число в массиве (S_i).

Время работы этого решения — $O(n \log n)$.

Оглавление