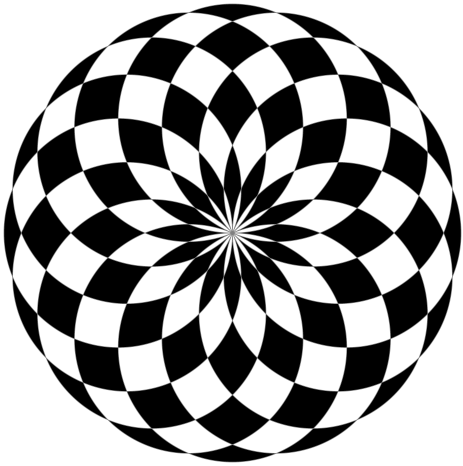
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ (ПОВОЛЖСКИЙ) УНИВЕРСИТЕТ
РЕСПУБЛИКАНСКИЙ ОЛИМПИАДНЫЙ ЦЕНТР РТ

Олимпиады по информатике школьников Татарстана

2017-2018 и 2018-2019



Казань – 2022

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. 2017-2018 и 2018-2019: Учебно-методическое пособие / М.И. Киндер. — Казань: Казанский федеральный университет, 2022. — 121 с.

Брошюра предназначена для школьников, учителей, преподавателей и тренеров по олимпиадной информатике. В ней представлены задачи, предлагавшиеся в 2017-2018 и 2018-2019 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.

2017-2018 учебный год

Муниципальный этап 30-й Всероссийской олимпиады по информатике среди школьников Республики Татарстан состоялся 5 декабря 2017 г. Олимпиада представляет собой практический тур, на котором школьникам 7-11 классов предлагается решить 4-5 задач. Как правило, первые две задачи в списке заданий муниципального этапа опираются, в основном, на логику алгоритмического мышления и не требуют «профессиональных» олимпиадных навыков. Следующие, более сложные задачи охватывают достаточно большой спектр различных стандартных и нестандартных алгоритмов. Автор большинства задач муниципальной олимпиады 2017 г. — преподаватель Казанского Федерального Университета *М. И. Киндер.*

Региональный этап олимпиады проходил с 27 по 29 января 2018 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, 2017-2018

Задача 1. Конфеты (7-11 классы)

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

Карлсон — самый лучший в мире упаковщик конфет!

Если количество конфет в некоторых коробках оказывается разным, Карлсон перекладывает часть конфет так, чтобы во всех коробках было конфет поровну. Если это сделать невозможно, он съедает одну конфету и так поступает до тех пор, пока не наступит момент, когда оставшиеся конфеты можно будет распределить поровну между коробками. Например, если в трёх коробках находится 3, 4 и 5 конфет соответственно, то Карлсон перекладывает одну конфету из последней коробки в первую, при этом во всех коробках будет одинаковое количество конфет — по 4. Однако, если в коробках 3, 4 и 7 конфет, то Карлсон сначала съедает 2 конфеты в последней коробке, а затем перекладывает из неё одну конфету в первую коробку.

Ваша задача — определить *наименьшее* число конфет, которые придётся съесть Карлсону, а также *наименьшее* число конфет, которые ему нужно переложить, чтобы во всех коробках было одинаковое количество конфет.

Формат входных данных

Первая строка содержит одно целое число n — количество коробок с конфетами ($2 \leqslant n \leqslant 1000$). Во второй строке записаны через пробел n целых чисел — количества конфет в каждой из коробок. В каждой коробке целое количество конфет — не менее одной и не более 1000.

Формат выходных данных

В ответе запишите через пробел два целых числа: наименьшее количество конфет, которые придётся Карлсону съесть (возможно, ни одной), и наименьшее количество конфет, которые ему нужно переложить.

Система оценивания

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

Примеры

candy.in	candy.out
3 3 4 5	0 1
3 7 3 4	2 1

Задача 2. Похожие числа (7-8 классы)

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

Вам даны два числа a и b, состоящие из одинакового количества цифр. *Степенью похожести* будем называть количество совпадающих цифр в десятичной записи a и b. Например, в записи чисел $a = 2233$ и $b = 4232$ есть три совпадающие цифры — две двойки и одна тройка, поэтому их степень похожести равна трём. А вот в записи чисел $a = 100$ и $b = 234$ нет одинаковых цифр, поэтому степень их похожести равна 0.

Вам необходимо составить программу, которая определяет степень похожести двух заданных чисел.

Формат входных данных

В первой строке одно целое n — количество цифр в записи чисел a и b ($1 \leqslant n \leqslant 10^6$). Во второй строке — число a, записанное в виде n десятичных цифр, разделённых пробелом. В третьей строке — число b, записанное в виде n десятичных цифр, разделённых пробелом. Первая (слева) цифра у чисел a и b отлична от нуля.

Формат выходных данных

Выведите степень похожести чисел a и b.

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	$1 \leqslant n \leqslant 10^4$	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

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

Задача 3. Штабеля (7-11 классы)

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

В соответствии с технологической документацией каждый штабель формируется из k коробок путем установки их друг на друга, каждая коробка весит целое число килограммов. Для обес-

печения прочности штабеля вес каждой коробки должен быть не меньше суммарного веса всех находящихся над ней коробок. Такой штабель считается *прочным*, и он может быть допущен к эксплуатации. Например, из трёх коробок массой 1, 3 и 5 килограммов можно сформировать прочный штабель: коробку массой 1 кг нужно поставить на верх штабеля, массой 3 кг — второй сверху, и, наконец, коробку массой 5 кг — в низ штабеля.

Вам необходимо составить программу, которая для каждого набора из k коробок определяет, можно ли из них сформировать прочный штабель.

Формат входных данных

В первой строке два целых числа: n — количество штабелей и k — количество коробок в каждом штабеле ($1 \leqslant n \leqslant 20\,000$; $2 \leqslant k \leqslant 50$). В каждой из следующих n строк записаны разделенные пробелом k целых чисел — массы коробок в каждом из n штабелей. Каждая коробка весит целое число килограммов, не менее 1 кг и не более 10^{18} кг.

Формат выходных данных

Выходные данные содержат n строк. В i-й строке должно быть записано `yes`, если из k коробок i-го набора можно сформировать прочный штабель, и `no` — в противном случае.

Примеры

stack.in	stack.out
1 3 4 1 5	yes
2 3 5 1 3 5 3 3	yes no

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, k	
1	30	$1 \leqslant n \leqslant 10$, $2 \leqslant k \leqslant 50$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$1 \leqslant n \leqslant 5\,000$, $2 \leqslant k \leqslant 50$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$1 \leqslant n \leqslant 20\,000$, $2 \leqslant k \leqslant 50$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Задача 4. Цепочка вычитаний (7-11 классы)

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

На доске записаны $n+1$ степеней двоек: 1, 2^1, 2^2, 2^3, $\ldots$, 2^n. Разрешается стереть любые два числа и вместо них записать их разность — *неотрицательное* число. После n таких операций на доске останется только одно число.

Вам необходимо найти такую последовательность операций вычитания, с помощью которой из исходных чисел получается данное число k.

Формат входных данных

Входные данные содержат два целых числа n и k ($1 \leqslant n \leqslant 60$; $1 \leqslant k \leqslant 10^{18}$). Первое число n означает, что на доске первоначально записаны $n+1$ чисел: 1, 2^1, 2^2, 2^3, $\ldots$, 2^n. Второе задаёт число, которое необходимо получить.

Формат выходных данных

Если число k получить можно, выведите n строк. В i-й строке запишите два числа, которые стираются на i-м шаге. (Вместо них появится разность стёртых чисел. Это новое число можно использовать на одном из следующих шагов.) Если решений несколько, выведите любое из них. Выведите `-1`, если число k получить невозможно.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, k	
1	30	$1 \leqslant n \leqslant 10$, $1 \leqslant k \leqslant 10$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$1 \leqslant n \leqslant 30$, $1 \leqslant k \leqslant 10^9$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$1 \leqslant n \leqslant 60$, $1 \leqslant k \leqslant 10^{18}$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

subtract.in	subtract.out
3 5	1 2 4 1 8 3
3 4	-1

Пояснение к примеру

В первом примере $n = 3$, и число $k = 5$ получается из последовательности 1, 2^1, 2^2, 2^3 с помощью цепочки из $n = 3$ вычитаний: $5 = 8 - (4 - (2 - 1))$, то есть сначала составляем разность чисел 2

и 1 (остаётся $2 - 1 = 1$), затем — разность 4 и полученного числа 1 (остаётся $4 - 1 = 3$), наконец, стираем два оставшихся числа 8 и 3, заменив их разностью $8 - 3 = 5$.

Во втором примере ответ `-1`, так как число $k = 4$ нельзя получить из того же набора чисел 1, 2^1, 2^2, 2^3 с помощью $n = 3$ вычитаний.

Задача 5. Скобочки (9-11 классы)

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

Строка, состоящая из символов «(» и «)», называется *скобочной последовательностью*. Скобочная последовательность считается *правильной*, если она может быть получена следующим образом:

- пустая строка является правильной скобочной последовательностью;
- если S и T — правильные скобочные последовательности, то строка (ST) также является правильной скобочной последовательностью.

Например, скобочные последовательности ((())) и (()()) из трёх пар скобок правильные, а последовательности)()() и ())() — неправильные. Конечно, неправильной будет также любая строка с несовпадающим числом открывающихся и закрывающихся скобок.

Вам необходимо написать программу, которая для заданного значения n определяет количество правильных скобочных последовательностей с n открывающимися и n закрывающимися скобками.

Формат входных данных

В строке записано одно целое число n ($1 \leqslant n \leqslant 2\,500$).

Формат выходных данных

Выведите одно число — количество правильных скобочных

последовательностей с n открывающимися и n закрывающимися скобками, вычисленное по модулю $(10^9 + 9)$.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	30	$1 \leqslant n \leqslant 10$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$1 \leqslant n \leqslant 100$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$1 \leqslant n \leqslant 2\,500$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

brackets.in	brackets.out
3	2
4	4

Пояснение к примеру

Во втором примере для $n = 4$ пар открывающихся и закрывающихся скобок имеются четыре правильные скобочные последовательности ((((()))), (((()))), ((())()) и (()(())).

Задача 6. Эх, дороги... (9-11 классы)

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

В стране Далёкой есть n городов и m соединяющих их дорог. Города пронумерованы числами от 1 до n, дороги — числами от 1 до m. Любые два города i и j соединены не более чем одной двусторонней дорогой, по которой можно попасть из i в j и из j в i. Кроме того, каждая дорога соединяет лишь разные города, то есть нет дорог из i в i.

В настоящее время страна переживает экономический кризис, и бюджета хватает на обслуживание только части дорог. Для уменьшения расходов правительством страны решено оставить некоторые из них, а остальные закрыть. Для каждого города i был составлен список городов, до которых можно добраться из него, двигаясь по дорогам Далёкой страны. Разумеется, этот список достижимых для него городов должен остаться прежним и после того, как часть дорог будет закрыта.

Ваша задача — составить список из *наименьшего* числа дорог, которые нужно оставить в стране (при этом остальные дороги будут закрыты).

Формат входных данных

В первой строке два целых числа n и m — количество городов и дорог соответственно ($2 \leqslant n \leqslant 3\,000; 1 \leqslant m \leqslant \frac{1}{2}n(n-1)$ и $m \leqslant 1\,500\,000$). В следующих m строках — описание дорог страны. Каждая дорога задается двумя числами — номерами городов, которые она соединяет.

Формат выходных данных

В первой строке запишите одно число k — наименьшее количество дорог, которые нужно оставить. В следующих k строках — описание оставленных в стране дорог. Каждая такая дорога задается двумя числами — номерами городов, которые она соединяет. Если решений несколько, выведите любое из них.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	30	$2 \leqslant n \leqslant 10$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$2 \leqslant n \leqslant 1\,000$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$2 \leqslant n \leqslant 3\,000$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Пример

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

Пояснение к примеру

В примере $n = 6$ городов соединены $m = 4$ дорогами, причём из города 4 не выходит ни одной дороги. В ответе список из трёх дорог, которые следует оставить в стране.

Региональный этап, 2017-2018

Задача 1. Улучшение успеваемости

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В лицее на уроках информатики ответы учеников оцениваются целым числом баллов от 2 до 5. Итоговая оценка по информатике выставляется как среднее арифметическое оценок на всех уроках, округленное до ближайшего целого числа. Если среднее значение находится ровно посередине между двумя целыми числами, то оценка округляется вверх.

Примеры округления оценок приведены в таблице.

Оценки на уроках	**Среднее арифметическое**	**Итоговая оценка**
2, 3, 5	$\frac{2+3+5}{3} = 3\frac{1}{3}$	3
3, 3, 4, 4	$\frac{3+3+4+4}{4} = 3\frac{1}{2}$	4
5, 5, 5, 3, 5	$\frac{5+5+5+3+5}{5} = 4\frac{3}{5}$	5

Все ученики лицея стремятся получить итоговую оценку по информатике не ниже 4 баллов. К сожалению, один из учеников получил на уроках a двоек, b троек и c четверок. Теперь он планирует получить несколько пятерок, причем хочет, чтобы итоговая оценка была не меньше 4 баллов.Ему надо понять, какое минимальное количество пятерок ему необходимо получить, чтобы добиться своей цели.

Требуется написать программу, которая по заданным целым неотрицательным числам a, b и c определяет минимальное количество пятерок, которое необходимо получить ученику, чтобы его итоговая оценка по информатике была не меньше 4 баллов.

Формат входных данных

Входные данные содержат три строки. Первая строка содержит целое неотрицательное число a, вторая строка содержит целое неотрицательное число b, третья строка содержит целое неотрицательное число c ($0 \leqslant a, b, c \leqslant 10^{15}$, $a + b + c \geqslant 1$).

Формат выходных данных

Выходные данные должны содержать одно число: минимальное число пятерок, которое необходимо получить ученику, чтобы итоговая оценка была не меньше 4 баллов.

Примечание

Следует обратить внимание, что входные данные в этой и других задачах не помещаются в стандартный 32-битный тип данных. Необходимо использовать 64-битный тип данных (`long long` в `C++`, `int64` в Паскале, `long` в Java).

Пример

стандартный ввод	стандартный вывод
2 0 0	2

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. По каждой подзадаче предоставляется полная информация о прохождении тестов.

Подзадача 1 (13 баллов)

$1 \leqslant a \leqslant 100$, $b = 0$, $c = 0$ (ученик получал только двойки).

Подзадача 2 (14 баллов)

$a = 0$, $1 \leqslant b \leqslant 100$, $c = 0$ (ученик получал только тройки).

Подзадача 3 (15 баллов)

$0 \leqslant a, b, c \leqslant 100$. Необходимые подзадачи — 1 и 2.

Подзадача 4 (28 баллов)

$0 \leqslant a, b, c \leqslant 10^6$. Необходимые подзадачи — 1, 2 и 3.

Подзадача 5 (30 баллов)

$0 \leqslant a, b, c \leqslant 10^{15}$. Необходимые подзадачи — 1, 2, 3 и 4.

Задача 2. Квадраты и кубы

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В лаборатории теории чисел одного университета изучают связь между распределением квадратов и кубов натуральных чисел.

Пусть задано целое неотрицательное число k. Рассмотрим множество натуральных чисел от a до b, включительно. Будем называть *k-плотностью* этого множества количество пар натуральных чисел x и y, таких, что $a \leqslant x^2 \leqslant b$, $a \leqslant y^3 \leqslant b$, причем $|\,x^2 - y^3\,| \leqslant k$.

Например, 2-плотность множества натуральных чисел от 1 до 30 равна 3, так как подходят следующие пары:

- $x = 1$, $y = 1$, $|\,x^2 - y^3\,| = |\,1 - 1\,| = 0$;
- $x = 3$, $y = 2$, $|\,x^2 - y^3\,| = |\,9 - 8\,| = 1$;
- $x = 5$, $y = 3$, $|\,x^2 - y^3\,| = |\,25 - 27\,| = 2$.

Требуется написать программу, которая по заданным натуральным числам a и b, а также целому неотрицательному числу k определяет k-плотность множества натуральных чисел от a до b, включительно.

Формат входных данных

Входные данные содержат три строки. Первая строка содержит натуральное число a, вторая строка содержит натуральное число b, третья строка содержит целое неотрицательное число k ($1 \leqslant a \leqslant b \leqslant 10^{18}$, $0 \leqslant k \leqslant 10^{18}$).

Формат выходных данных

Выходные данные должны содержать одно целое число: искомую k-плотность множества натуральных чисел от a до b, включительно.

Примеры

стандартный ввод	стандартный вывод
1 30 2	3

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. По каждой подзадаче предоставляется полная информация о прохождении тестов.

Подзадача 1 (10 баллов)

$1 \leqslant a \leqslant b \leqslant 1\,000$, $k = 0$.

Подзадача 2 (10 баллов)

$1 \leqslant a \leqslant b \leqslant 10^{18}$, $k = 0$. Необходимые подзадачи — 1.

Подзадача 3 (15 баллов)

$1 \leqslant a \leqslant b \leqslant 1\,000$, $0 \leqslant k \leqslant 10$. Подзадачи — 1.

Подзадача 4 (15 баллов)

$1 \leqslant a \leqslant b \leqslant 10^{6}$, $0 \leqslant k \leqslant 10$. Подзадачи — 1, 3.

Подзадача 5 (15 баллов)

$1 \leqslant a \leqslant b \leqslant 10^{9}$, $0 \leqslant k \leqslant 10$. Подзадачи — 1, 3, 4.

Подзадача 6 (15 баллов)

$1 \leqslant a \leqslant b \leqslant 10^{9}$, $0 \leqslant k \leqslant 10^{9}$. Подзадачи — 1, 3, 4, 5.

Подзадача 7 (20 баллов)

$1 \leqslant a \leqslant b \leqslant 10^{18}$, $0 \leqslant k \leqslant 10^{18}$. Подзадачи — 1–6.

Задача 3. Лифт

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В современном многоэтажном офисе крупной компании установлен новый лифт. В компании работает n сотрудников. Для проверки эффективности системы управления лифтом требуется провести моделирование его работы в конце рабочего дня, когда все сотрудники должны покинуть здание и спуститься на первый этаж.

В здании m этажей, пронумерованных от 1 до m снизу-вверх. Известно, что i-й сотрудник подходит к лифту в секунду t_i на этаже a_i, чтобы спуститься на первый этаж.

На каждом этаже могут находиться люди, ожидающие лифт. Когда очередной сотрудник подходит к лифту, он вызывает лифт, если на этом этаже лифт еще не вызван, либо присоединяется к ожидающим лифт. Таким образом, помимо вызвавшего лифт, вместе с ним лифт могут ожидать и другие сотрудники.

В каждый момент времени не более одного вызова является *активным*.

Изначально лифт свободен и находится на первом этаже. Когда поступает первый вызов, этот вызов становится активным и лифт отправляется на соответствующий этаж. Если несколько вызовов поступает одновременно, активным становится вызов от сотрудника с меньшим номером.

Лифт перемещается между этажами со скоростью один этаж в секунду. Когда лифт оказывается на этаже, откуда был сделан активный вызов, в него заходят все, кто уже ожидает лифт на этом этаже, и лифт отправляется вниз на первый этаж со скоростью один этаж в секунду.

При движении вниз лифт останавливается на тех этажах, в которых был сделан вызов на момент проезда лифта мимо этого этажа. Все ожидающие лифт сотрудники заходят в него и вызов на этом этаже сбрасывается. Когда лифт завершает движение на первом этаже, все люди выходят из лифта, а лифт ожидает следующего вызова.

Если в момент, когда лифт освободился, есть хотя бы один необслуженный вызов, активируется вызов, который поступил раньше других. Если несколько вызовов поступило одновременно, активируется вызов от сотрудника с меньшим номером. Лифт продолжает обслуживание описанным образом, пока все n сотрудников не окажутся на первом этаже.

Будем считать, что люди входят и выходят из лифта мгновенно. Каждую секунду сначала люди подходят и вызывают лифт, а затем выполняются соответствующие действия (лифт перемещается на соседний этаж, в него входят или из него выходят люди, принимается решение, на какой вызов лифт должен отреагировать).

Требуется написать программу, которая по описанию вызовов лифта для каждого сотрудника определяет, в какой момент этот сотрудник окажется на первом этаже.

Формат входных данных

Первая строка входных данных содержит целые числа n и m — количество людей, вызывающих лифт, и количество этажей в здании ($1 \leqslant n \leqslant 10^5$, $2 \leqslant m \leqslant 10^9$).

Следующие n строк описывают сотрудников, i-я из этих строк содержит два целых числа t_i и a_i — секунду, в которую i-й сотрудник подходит к лифту, и номер этажа, на котором это происходит ($1 \leqslant t_1 \leqslant t_2 \leqslant \ldots \leqslant t_n \leqslant 10^9$, $2 \leqslant a_i \leqslant m$).

Формат выходных данных

Выходные данные должны содержать n целых чисел, для каждого сотрудника требуется вывести секунду, в которую он выйдет из лифта на первом этаже.

Пример

стандартный ввод	стандартный вывод
5 4 2 3 2 4 5 2 5 3 9 3	6 12 6 12 12

Пояснение к примерам

Пример работы лифта по шагам показан в таблице.

Время в секундах	1 этаж	2 этаж	3 этаж	4 этаж
0	[]			
1	[]			
2	$[]\uparrow^3$		☺_1	☺_2
3		$[]\uparrow^3$	☺_1	☺_2
4			$[] \leftarrow \text{☺}_1$	☺_2
5		$[\text{☺}_1] \leftarrow \text{☺}_3$	☺_4	☺_2
6	$[\text{☺}_1 \rightarrow, \text{☺}_3 \rightarrow]\uparrow^4$		☺_4	☺_2
7		$[]\uparrow^4$	☺_4	☺_2
8			$[]\uparrow^4 \quad \text{☺}_4$	☺_2
9			$\text{☺}_4, \text{☺}_5$	$[] \leftarrow \text{☺}_2$
10			$[\text{☺}_2] \leftarrow \text{☺}_4, \leftarrow \text{☺}_5$	
11		$[\text{☺}_2, \text{☺}_4, \text{☺}_5]$		
12	$[\text{☺}_2 \rightarrow, \text{☺}_4 \rightarrow, \text{☺}_5 \rightarrow]$			

Использованные обозначения

Обозначение	Пояснение
[]	обозначение лифта
$[]\uparrow^j$	лифт двигается на активный вызов, сделанный на j-м этаже
☺_i	i-й сотрудник ждет лифта
$\leftarrow \text{☺}_i$	i-й сотрудник заходит в лифт
$[\text{☺}_i]$	i-й сотрудник находится в лифте
$[\text{☺}_i \rightarrow]$	i-й сотрудник выходит из лифта

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. По каждой подзадаче предоставляется полная информация о прохождении тестов.

Задача 4. Мониторинг труб

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

Газораспределительная система одного региона устроена следующим образом. Она содержит n узлов, пронумерованных от 1 до n, некоторые узлы соединены односторонними трубами. Узел с номером 1 соответствует центральному газохранилищу.

Система узлов описывается числами от p_2, p_3, ... , p_n. Для всех i от 2 до n узел с номером p_i соединен односторонней трубой с узлом i, газ по этой трубе передается от узла p_i к узлу i. Известно, что возможно доставить газ по трубам от центрального газохранилища до любого узла системы (возможно, с использованием промежуточных узлов). В системе используются трубы различных типов, тип трубы обозначается буквой английского алфавита от «`a`» до «`z`». Труба, соединяющая узел p_i с узлом i, имеет тип c_i.

Для проверки качества труб используется специальный робот. Он помещается в систему труб в одном из узлов и перемещается по трубам, каждый раз проверяя трубу, по которой он перемещается. Робот может перемещаться по трубам только в том же направлении, в котором по трубе передается газ. Совершив одно или несколько перемещений по трубам между узлами, робот извлекается из системы труб.

Каждый запуск робота должен соответствовать одной из m заданных *спецификаций*, пронумерованных от 1 до m. Спецификация с номером t представляет собой строку s_t, состоящую из строчных букв английского алфавита. Запуск соответствует спецификации s_t, если количество перемещений робота по трубам во время запуска совпадает с длиной s_t, и для всех j от 1 до длины s_t на j-м шаге робот перемещается по трубе, тип которой совпадает с $s_t[j]$ — символом на позиции j в спецификации.

Если запуск робота соответствует спецификации с номером t, то стоимость этого запуска составляет w_t. Оператору системы необходимо проверить все трубы, для этого можно запускать робот несколько раз. Каждый раз выбирается спецификация и

маршрут робота по трубам, соответствующие выбранной спецификации. Необходимо проверить все трубы так, чтобы суммарная стоимость запусков робота для проверки качества труб была минимальна. Одну и ту же трубу можно проверять несколько раз.

Требуется написать программу, которая по описанию системы труб и списку спецификаций определяет минимальную суммарную стоимость запусков робота, в результате которых все трубы будут проверены, а также список необходимых для этого запусков (по требованию).

Формат входных данных

В первой строке входных данных находятся три целых числа n, m и t — количество узлов системы труб, количество спецификаций запусков робота и параметр, указывающий, требуется ли вывести список запусков робота или только их минимальную суммарную стоимость ($1 \leqslant n \leqslant 500$; $1 \leqslant m \leqslant 10^5$, t равно 0 или 1).

В последующих $(n-1)$ строках содержится информация о трубах, $(i-1)$-я из этих строк содержит разделенные пробелом значения p_i и c_i, где p_i — целое число, задающее номер узла, из которого ведет труба в i-й узел, а c_i — строчная буква английского алфавита, задающая тип этой трубы ($1 \leqslant p_i \leqslant i-1$).

В последующих m строках содержится информация о спецификациях, i-я из этих строк содержит разделенные пробелом целое число w_i — стоимость запуска робота в соответствии с этой спецификацией, и состоящую из строчных букв английского алфавита строку s_i — саму спецификацию ($1 \leqslant w_i \leqslant 10^9$). Суммарная длина строк s_i не превышает 10^6.

Формат выходных данных

Первая строка выходных данных должна содержать одно число — минимальную суммарную стоимость запусков робота, в результате которых все трубы будут проверены. Если проверить все трубы невозможно, требуется вывести `-1`.

Если $t = 0$, то больше ничего выводить не требуется.

Если $t = 1$ и проверить трубы возможно, то далее следует вывести список описаний запусков робота. В этом случае вторая строка выходных данных должна содержать число k — количество запусков робота, которое необходимо выполнить для проверки труб. В следующих k строках необходимо вывести по три

целых числа a_i, b_i и c_i — номер узла, в котором начинается запуск, номер узла, в котором заканчивается запуск, и номер спецификации, которой соответствует запуск.

Если оптимальных способов проверки несколько, требуется вывести любой из них.

Пример

стандартный ввод	стандартный вывод
3 3 0 1 a 2 b 3 a 4 b 2 a	6
7 3 1 1 a 2 a 3 b 3 b 1 b 6 b 3 aab 5 b 2 ab	15 4 1 4 1 2 5 3 1 6 2 6 7 2

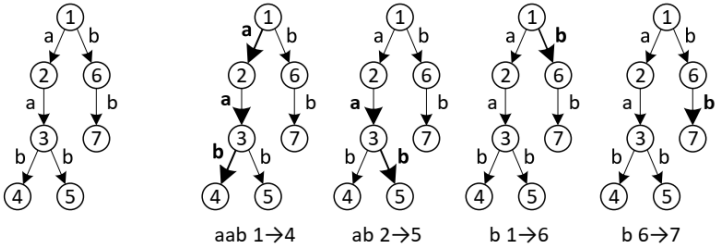
Пояснение к примеру

Система труб, заданная во втором примере входных данных, и оптимальный способ проверки всех труб для этого случая приведены на рисунке ниже.

Необходимо обратить внимание на следующие моменты:

- трубу можно проверять несколько раз, так в приведенном примере дважды проверена труба из узла 2 в узел 3;
- одну и ту же спецификацию разрешается использовать несколько раз, в приведенном примере вторая спецификация используется дважды, для проверки труб из узла 1 в узел 6 и из узла 6 в узел 7;

- робот может перемещаться по трубам только в том же направлении, по которому по трубе передается газ, спецификацию «ab» нельзя использовать для проверки труб по маршруту $2 \to 1 \to 6$, так как робот не может переместиться из узла 2 в узел 1.



Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты этой подзадачи успешно пройдены. Информация о проверке тестов в подзадачах 1 и 2 — полная, в подзадачах 3-6 сообщаются баллы.

Подзадача 1 (9 баллов)

$1 \leqslant n \leqslant 500$, $1 \leqslant m \leqslant 10^5$, $t = 0$. Длина каждой строки s_i равна 1.

Подзадача 2 (10 баллов)

$1 \leqslant n \leqslant 500$, $1 \leqslant m \leqslant 10^5$, $t = 0$. Для всех i выполнено $p_i = i - 1$.

Подзадача 3 (22 балла)

$1 \leqslant n \leqslant 15$, $1 \leqslant m \leqslant 10^5$, $t = 0$.

Подзадача 4 (20 баллов)

$1 \leqslant n \leqslant 500$, $1 \leqslant m \leqslant 500$, $t = 0$.

Подзадача 5 (19 баллов)

$1 \leqslant n \leqslant 500$, $1 \leqslant m \leqslant 10^5$, $t = 0$. Необходимые подзадачи 1-4.

Подзадача 6 (20 баллов)

$1 \leqslant n \leqslant 500$, $1 \leqslant m \leqslant 10^5$, $t = 1$. Необходимые подзадачи 1-5.

Задача 5. Удаление чисел

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В ряд выписаны натуральные числа от 1 до n и задано натуральное число k.

Выполняется один или несколько шагов по удалению чисел в этом ряду. На очередном шаге оставшиеся числа просматриваются в возрастающем порядке, и каждое k-е число удаляется. Если после очередного шага осталось меньше k чисел, то процесс удаления чисел завершается.

Необходимо определить, на каком шаге будет удалено число n, или выяснить, что оно не будет удалено до завершения процесса.

Например, пусть $n = 13$, $k = 2$.

- На первом шаге будут удалены числа 2, 4, 6, 8, 10 и 12, останутся числа 1, 3, 5, 7, 9, 11 и 13;
- На втором шаге будут удалены числа 3, 7 и 11, останутся числа 1, 5, 9 и 13;
- На третьем шаге будут удалены числа 5 и 13, останутся числа 1 и 9;
- На четвертом шаге будет удалено число 9, останется число 1. Поскольку осталось одно число, процесс завершается.

Таким образом, число 13 будет удалено на третьем шаге.

Требуется написать программу, которая по заданным числам n и k определяет, на каком шаге будет удалено число n.

Формат входных данных

Первая строка содержит целое число n ($3 \leqslant n \leqslant 10^{18}$).

Вторая строка содержит целое число k ($2 \leqslant k \leqslant 100$, $k < n$).

Формат выходных данных

Требуется вывести одно целое число — номер шага, на котором будет удалено число n, или число 0, если число n не будет удалено.

Пример

стандартный ввод	стандартный вывод
13 2	3

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. Информация о проверке тестов в подзадачах — полная.

Подзадача 1 (16 баллов)

$3 \leqslant n \leqslant 1\,000$, $k = 2$.

Подзадача 2 (10 баллов)

$3 \leqslant n \leqslant 10^{18}$, $k = 2$. Необходимые подзадачи 1.

Подзадача 3 (14 баллов)

$3 \leqslant n \leqslant 1\,000$, $2 \leqslant k \leqslant 100$, $k < n$. Необходимые подзадачи 1.

Подзадача 4 (20 баллов)

$3 \leqslant n \leqslant 10^{6}$, $2 \leqslant k \leqslant 100$, $k < n$. Необходимые подзадачи 1, 3.

Подзадача 5 (40 баллов)

$3 \leqslant n \leqslant 10^{18}$, $2 \leqslant k \leqslant 100$, $k < n$. Необходимые подзадачи 1–4.

Задача 6. Старая книга

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

Группа юных археологов работает на раскопках здания древней библиотеки. Летом они обнаружили остатки старой книги и, изучив их, сделали следующие выводы.

Книга содержит несколько страниц, каждая страница содержит либо текст, либо иллюстрацию. Первые k страниц книги точно содержат иллюстрации. Все страницы книги пронумерованы, но номер страницы написан только на страницах, содержащих текст. Сумма номеров страниц с текстом равна s.

К сожалению, ни общее количество страниц в книге, ни количество иллюстраций установить не удалось. Тем не менее, юных археологов заинтересовал вопрос, какое минимальное количество иллюстраций могло быть в книге.

Например, если $k = 1$, а $s = 8$, то страницы книги могли иметь следующее содержание (буквой «Т» обозначена страница, содержащая текст, а буквой «И» — страница, содержащая иллюстрацию):

- ИТИИИТ, то есть пронумерованы страницы 2 и 6, 4 иллюстрации;
- ИИТИТ, то есть пронумерованы страницы 3 и 5, 3 иллюстрации;
- ИИИИИИИТ, то есть пронумерована страница 8, 7 иллюстраций.

Минимальное количество иллюстраций равно 3.

Требуется написать программу, которая по заданным целым числам k и s определяет минимальное количество иллюстраций, которое могло быть в книге.

Формат входных данных

Первая строка содержит целое число k ($0 \leqslant k \leqslant 10^9$).

Во второй строке записано целое число s ($k + 1 \leqslant s \leqslant 10^{12}$).

Формат выходных данных

Требуется вывести одно целое число — минимальное количество иллюстраций в книге.

Пример

стандартный ввод	стандартный вывод
1 8	3

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. Информация о проверке тестов в подзадачах — полная.

Подзадача 1 (15 баллов)

$k = 0$; $1 \leqslant s \leqslant 200$.

Подзадача 2 (20 баллов)

$k = 0$; $1 \leqslant s \leqslant 10^{12}$. Необходимые подзадачи 1.

Подзадача 3 (30 баллов)

$0 \leqslant k \leqslant 199$; $k + 1 \leqslant s \leqslant 200$. Необходимые подзадачи 1.

Подзадача 4 (35 баллов)

$0 \leqslant k \leqslant 10^9$; $k + 1 \leqslant s \leqslant 10^{12}$. Необходимые подзадачи 1–3.

Задача 7. Красота фейерверка

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В лаборатории теоретической пиротехники изучают новые технологии организации фейерверков. Фейерверк представляется как корневое дерево, а поскольку в мощном фейерверке его элементы также взрываются, порождая новые фейерверки, то ученые вводят операцию возведения корневого дерева в степень.

Корневое дерево содержит одну или несколько вершин. Одна из вершин выделена и называется *корнем дерева*, для каждой из остальных вершин ровно одна другая вершина является её *родителем*. При этом от любой вершины можно добраться до корня, последовательно переходя от вершины к ее родителю. Вершина, которая не является родителем никакой другой вершины, называется *листом*. Если вершина x является родителем вершины y, то вершина y является *ребёнком* вершины x. Будем говорить, что вершина и ее родитель соединены *ребром*.

На рис. 1 показан пример корневого дерева с корнем в вершине 1. Родителем вершин 2 и 3 является вершина 1, родителем вершины 4 является вершина 2. Вершины 2 и 3 — дети вершины 1, а вершина 4 — ребёнок вершины 2. Листьями являются вершины 3 и 4.

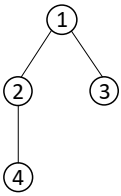


Рис. 1. Пример корневого дерева с корнем 1 и листьями 3, 4.

Фейерверк задается своим *базовым деревом* T и *мощностью* m. Фейерверк представляется деревом, которое получается в результате возведения дерева T в степень m. Операция возведения дерева в степень устроена следующим образом. Если $m = 1$, то результат T^1 — само дерево T. Для $m > 1$ рассмотрим дерево T^{m-1}. Выполним следующую операцию: для каждого листа x дерева T^{m-1} создадим копию дерева T и назначим лист x родителем корня соответствующей копии. Получившееся дерево будет деревом T^m.

На рис. 2 показано дерево, представленное на рис. 1, в степенях 1, 2 и 3.

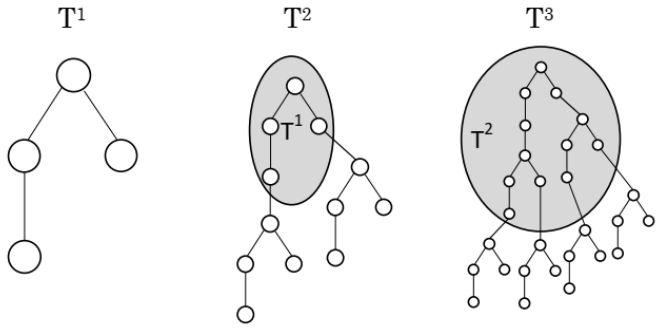


Рис. 2. Пример возведения дерева в степени 1, 2 и 3.

Путем в дереве называется последовательность вершин, в которой две соседние вершины соединены ребром. Все вершины в пути должны быть различны.

Для того чтобы оценить красоту фейерверка, необходимо определить, какое максимальное количество вершин может содержать путь в дереве, которым представляется фейерверк. На рис. 3 приведен путь в дереве T^2, содержащий максимальное количество вершин. Таким образом, красота фейерверка с базовым деревом T и мощностью 2 равна 10.

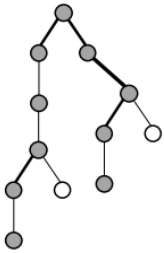


Рис. 3. Путь в дереве T^2, содержащий максимальное количество вершин.

Требуется написать программу, которая по описанию дерева T и натуральному числу m определяет красоту фейерверка с базовым деревом T и мощностью m.

Формат входных данных

Первая строка содержит два натуральных числа n и m — количество вершин в базовом дереве фейерверка T и его мощность ($3 \leqslant n \leqslant 200\,000$, $1 \leqslant m \leqslant 200\,000$).

Вторая строка описывает дерево T и содержит $(n-1)$ целых чисел: p_2, p_3, …, p_n — номера родителей вершин 2, 3, …, n, соответственно ($1 \leqslant p_i \leqslant i-1$).

Формат выходных данных

Требуется вывести одно целое число — красоту фейерверка, представляемого деревом T^m.

Пример

стандартный ввод	стандартный вывод
4 2 1 1 2	10

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. По каждой подзадаче предоставляется полная информация о прохождении тестов.

Подзадача 1 (19 баллов)

$3 \leqslant n \leqslant 5\,000$, $m = 1$.

Подзадача 2 (10 баллов)

$3 \leqslant n \leqslant 200\,000$, $m = 1$. Подзадачи — 1.

Подзадача 3 (20 баллов)

$3 \leqslant n \leqslant 5\,000$, $1 \leqslant m \leqslant 5\,000$. Подзадачи — 1.

Подзадача 4 (19 баллов)

$3 \leqslant n \leqslant 5\,000$, $1 \leqslant m \leqslant 200\,000$. Подзадачи — 1, 3.

Подзадача 5 (32 баллов)

$3 \leqslant n \leqslant 200\,000$, $1 \leqslant m \leqslant 200\,000$. Подзадачи — 1–4.

Задача 8. Обработка больших данных

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`512 мегабайт`

В научной лаборатории разрабатывается новая архитектура суперкомпьютера, позволяющая эффективно обрабатывать большие объемы данных.

Суперкомпьютер имеет 2^k ячеек памяти, пронумерованных от 0 до $2^k - 1$. *Отрезком* $[L, R]$ называется последовательность подряд идущих ячеек памяти с номерами от L до R, включительно.

Некоторые отрезки памяти являются *корректными*. Отрезок памяти $[L, R]$ является корректным, если его длина $(R - L + 1)$ равна 2^i для некоторого i, а число L делится на 2^i. Например, если $k = 3$, то ячейки памяти пронумерованы от 0 до 7, а корректными являются отрезки [0, 7], [0, 3], [4, 7], [0, 1], [2, 3], [4, 5], [6, 7], [0, 0], [1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [6, 6] и [7, 7].

Ключевой операцией в новой архитектуре является операция `STORE`, которая за одно действие присваивает одно и то же значение всем ячейкам памяти некоторого *корректного* отрезка.

Исходно все ячейки памяти содержат значение 0. В лаборатории планируют запустить на суперкомпьютере программу обработки данных, но перед её запуском необходимо инициализировать память определенным образом. А именно: первые c_1 ячеек памяти необходимо заполнить значениями v_1, следующие c_2 ячеек — значениями v_2, и так далее, последние c_n ячеек памяти необходимо заполнить значениями v_n, где $1 \leqslant v_i \leqslant m$.

Ученым надо выяснить, какое минимальное количество операций `STORE` необходимо выполнить, чтобы проинициализировать память требуемым образом.

Например, если $k = 3$, $n = 3$, $m = 2$, $c_1 = 1$, $v_1 = 1$, $c_2 = 2$, $v_2 = 2$, $c_3 = 5$, $v_3 = 1$, то итоговое содержимое памяти должно быть следующим: $[1, 2, 2, 1, 1, 1, 1, 1]$. В этом случае для инициализации памяти достаточно трёх операций `STORE`:

- `STORE`([0, 7], 1) — после этой операции все ячейки памяти содержат значение 1;
- `STORE`([1, 1], 2) — после этой операции содержимое памяти будет $[1, 2, 1, 1, 1, 1, 1, 1]$;
- `STORE`([2, 2], 2) — после этой операции содержимое памяти будет $[1, 2, 2, 1, 1, 1, 1, 1]$, как и требуется.

Заметим, что операцию `STORE`([1, 2], 2) выполнить нельзя, потому что [1, 2] не является корректным отрезком памяти.

Требуется написать программу, которая по заданному содержимому памяти определяет минимальное количество операций `STORE`, необходимых для инициализации памяти требуемым образом.

Формат входных данных

Первая строка содержит три целых числа k, n, m ($0 \leqslant k \leqslant 30$,

$1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 10^9$).

Следующие n строк содержат по два целых числа, i-я из этих строк содержит числа c_i и v_i ($1 \leqslant c_i \leqslant 2^k$, $1 \leqslant v_i \leqslant m$, $c_1 + c_2 + + \ldots + c_n = 2^k$).

Формат выходных данных

Требуется вывести одно целое число — минимальное количество операций `STORE`, которое необходимо выполнить для инициализации памяти заданным образом.

Пример

стандартный ввод	стандартный вывод
3 3 2 1 1 2 2 5 1	3

Система оценивания

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. По каждой подзадаче предоставляется информация о набранных баллах.

Подзадача 1 (15 баллов)

$0 \leqslant k \leqslant 3$, $1 \leqslant n \leqslant 8$, $1 \leqslant m \leqslant 8$.

Подзадача 2 (15 баллов)

$0 \leqslant k \leqslant 19$, $1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 10$. Подзадачи – 1.

Подзадача 3 (15 баллов)

$0 \leqslant k \leqslant 30$, $1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 10$. Подзадачи – 1, 2.

Подзадача 4 (10 баллов)

$0 \leqslant k \leqslant 30$, $1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 50$. Подзадачи – 1, 2, 3.

Подзадача 5 (15 баллов)

$0 \leqslant k \leqslant 19$, $1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 10^9$. Подзадачи – 1, 2.

Подзадача 6 (30 баллов)

$0 \leqslant k \leqslant 30$, $1 \leqslant n \leqslant 10^5$, $1 \leqslant m \leqslant 10^9$. Подзадачи – 1–5.

Задача 9. Поменяй жизнь! (7-8 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Коту Леопольду посвящается ...

Жители города Иннополиса — обычные люди, они дружат или враждуют между собой.

Отношение дружбы и вражды в этом городе обладает свойством взаимности, то есть если A — друг (или враг) B, то B — друг (или враг) A. Среди жителей города встречаются не только друзья, но и враждующие между собой. За один день какой-нибудь житель может начать новую жизнь: перессориться со всеми своими друзьями и подружиться со всеми своими врагами. Если же у него нет ни одного друга (врага), то, начав новую жизнь, он может подружиться (поссориться) со всеми остальными жителями города.

Требуется написать программу для подсчета наименьшего количества дней, за которое все жители города могут подружиться, а также для определения списка жителей, которым для этого нужно начать новую жизнь.

Формат входных данных

Первая строка содержит одно целое n — число жителей города ($3 \leqslant n \leqslant 1000$). В каждой i-й из последующих n строк записано сначала количество d_i друзей i-го жителя ($0 \leqslant d_i < n$), а затем d_i различных целых чисел, не превосходящих n, — номера его друзей. Остальные жители враждуют с ним.

Формат выходных данных

Выведите `-1`, если всех жителей города подружить невозможно. Иначе в первой строке запишите одно число — наименьшее количество дней, за которое все жители могут подружиться. Во второй строке — последовательность разделенных пробелом целых чисел — номеров жителей, которым для этого нужно начать

новую жизнь. Если возможных решений несколько, выведите любое из них.

Система оценивания

Данная задача содержит две подзадачи. Для оценки каждой подзадачи используется своя группа тестов. Баллы за подзадачу начисляются только в том случае, если все тесты из этой группы успешно пройдены. Информация по каждой подзадаче — результат проверки на каждом тесте.

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	40	$3 \leqslant n \leqslant 10$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	60	$3 \leqslant n \leqslant 1\,000$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.

Примеры

стандартный ввод	стандартный вывод
3 0 0 0	-1
3 1 2 1 1 0	1 3

Задача 10. Карандаши (7-8 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Фабрика по производству карандашей отличается сложным технологическим процессом с тщательным планированием и контролем качества.

Согласно требованиям каждый набор должен быть упакован карандашами одного цвета или одного размера, или одного цвета и одного размера. Другими словами, в наборе не может быть карандашей разного цвета и разной длины. И если такие два карандаша оказались в одном наборе, считается, что они образуют *бракованную пару*. Например, пара `1 2` и `2 3` является бракованной, а пары `2 3` и `1 3` или `2 3` и `2 3` не являются.

Все готовые карандаши пронумерованы и выложены в ряд. Несколько *подряд лежащих* карандашей в этом ряду упаковывают в карандашный набор. Для каждого такого набора нужно выяснить, есть ли в нём бракованная пара карандашей.

Требуется написать программу, которая для различных наборов подряд лежащих карандашей определит номера карандашей, образующих бракованную пару, или сообщит, что такой пары нет.

Формат входных данных

В первой строке записано одно число n — количество всех карандашей ($2 \leqslant n \leqslant 100\,000$). В следующих n строках записаны по два целых числа a_i и b_i, задающих цвет и размер i-го карандаша соответственно ($1 \leqslant a_i, b_i \leqslant 10^9$, $1 \leqslant i \leqslant n$).

В $(n+2)$-й строке записано одно целое число k — количество карандашных наборов, в каждом из которых нужно определить номера двух бракованных карандашей ($1 \leqslant k \leqslant 100\,000$). В следующих k строках записаны пары целых чисел n_1 и n_2 — номера первого и последнего карандаша набора соответственно, в котором необходимо найти бракованную пару карандашей ($1 \leqslant n_1 < n_2 \leqslant n$).

Формат выходных данных

Выходные данные должны содержать k строк, каждая из которых содержит два разделённых пробелом числа — номера ка-

рандашей, образующих бракованную пару в соответствующем карандашном наборе. Если решений несколько, можно вывести любое из них. Если в наборе бракованная пара отсутствует, требуется вывести в соответствующей строке `0 0`.

Система оценивания

Данная задача содержит четыре подзадачи. Для оценки каждой подзадачи используется своя группа тестов. Баллы за подзадачу начисляются только в том случае, если все тесты из этой группы успешно пройдены. По запросу сообщается результат проверки на каждом тесте.

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, k	
1	20	$2 \leqslant n \leqslant 100$, $1 \leqslant k \leqslant 100$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	30	$2 \leqslant n \leqslant 1\,000$, $1 \leqslant k \leqslant 1\,000$	Баллы начисляются, если пройдены все тесты этой подзадачи.
3	20	$2 \leqslant n \leqslant 5\,000$, $1 \leqslant k \leqslant 5\,000$	Баллы начисляются, если пройдены все тесты этой подзадачи.
4	20	$2 \leqslant n \leqslant 100\,000$, $1 \leqslant k \leqslant 100\,000$	Баллы начисляются, если пройдены все тесты этой подзадачи.

Пример

стандартный ввод	стандартный вывод
4 2 2 1 2 1 3 2 3 2 2 3 2 4	0 0 2 4

2018-2019 учебный год

Муниципальный этап 31-й Всероссийской олимпиады по информатике среди школьников Республики Татарстан состоялся 5 декабря 2018 г. На муниципальной олимпиаде школьникам 7-11 классов предлагается решить 5 задач. Как правило, первые две задачи в списке заданий опираются, в основном, на логику алгоритмического мышления и не требуют «профессиональных» олимпиадных навыков. Следующие три задачи — более сложные, они охватывают достаточно большой спектр различных стандартных и нестандартных алгоритмов. Автор большинства задач муниципальной олимпиады 2018 г. — преподаватель Казанского Федерального Университета *М. И. Киндер.*

Региональный этап олимпиады проходил с 26 по 28 января 2019 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. Для школьников 7-8 классов жюри составило несколько задач, сложность которых, по мнению жюри, соответствовала возрастной категории участников.

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, 2018-2019

Задача 1. Карточная игра (7-11 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Том и Джерри играют в карточную игру, правила которой очень просты. На столе лежат n карт лицевой стороной вверх, на каждой карте записано по одному числу. За один ход разрешается убрать со стола любые две карты с *равными* числами. Игрок, который не может сделать ход из-за того, что на столе не осталось ни одной пары карт с равными числами, считается проигравшим. Первым ходит Том.

Вам необходимо определить, кто из них выиграет — Том или Джерри.

Формат входных данных

В первой строке записано одно целое число n – количество карт($1 \leqslant n \leqslant 10^5$). В следующей строке записаны через пробел n целых чисел, каждое от 1 до 10^5 включительно.

Формат выходных данных

Выведите `1`, если выиграет Том; выведите `2`, если выиграет Джерри.

Примеры

стандартный ввод	стандартный вывод
3 2 4 2	1
5 1 3 3 1 1	2

Пояснение к примеру

В первом примере есть только одна пара равных чисел (2, 2), и игра заканчивается сразу после первого хода Тома. Во втором

примере Том убирает одну из пару равных чисел (3, 3) или (1, 1), а затем Джерри убирает вторую пару и выигрывает, так как не остаётся ни одной пары равных чисел.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	30	$1 \leqslant n \leqslant 1\,000$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	35	$1 \leqslant n \leqslant 50\,000$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	35	$1 \leqslant n \leqslant 10^5$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Задача 2. Супердвоичная система счисления (7-11 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`256 мегабайт`

Все динозавры делятся на 10 *групп — те, кто знают двоичную систему счисления, и те, кто не знают.*

Из «Большой энциклопедии динозавров».

Недавно палеонтологи обнаружили останки динозавра Linhenykus monodactylus, у которого на каждой передней конечности было только по одному пальцу. Распространение десятичной системы счисления связывают с количеством пальцев рук у челове-

ка. Значит, динозавры пользовались двоичной системой счисления. Точнее, супердвоичной системой, в которой для записи чисел использовались только «цифры» -1, 0 или 1. *Супердвоичной записью* числа n динозавры называли представление n в виде $2^k a_k + \ldots + 2^2 a_2 + 2a_1 + a_0$, где каждое из чисел a_i равно -1, 0 или 1 и $a_i \cdot a_{i+1} = 0$ для всех $0 \leqslant i \leqslant k-1$. Например, число 3 в этой системе записывалось в виде $\mathbf{1\,0\,{-}1}$, так как $3 = 2^2 \cdot \mathbf{1} + 2 \cdot \mathbf{0} + (\mathbf{-1})$.

Ваша задача — научиться записывать числа в супердвоичной системе динозавров.

Формат входных данных

В единственной строке записано целое число n ($1 \leqslant n \leqslant 10^{18}$).

Формат выходных данных

Единственная строка содержит последовательность из разделенных пробелом целых чисел $a_k, \ldots, a_1, a_0$, образующих запись числа n в супердвоичной системе счисления. Число a_k является первой (слева) цифрой в записи числа n, а a_0 — его последней цифрой. Если таких представлений несколько, выведите любое из них.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	30	$1 \leqslant n \leqslant 100$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	30	$1 \leqslant n \leqslant 10^5$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	40	$1 \leqslant n \leqslant 10^{18}$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

стандартный ввод	стандартный вывод
1	1
3	1 0 -1

Задача 3. Популярный рейтинг (7-11 классы)

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

На конференцию по проблемам в области информационных технологий приехали n известных программистов и учёных со всего мира. Авторитет конференции зависит от рейтинга участников; рейтинг каждого учёного — это целое положительное число r, равное количеству его научных публикаций. Число r считается *популярным*, если более половины участников конференции имеют рейтинг r.

Вам необходимо составить программу, которая из данных n рейтингов учёных определяет популярный.

Формат входных данных

В первой строке записано одно число n — количество участников конференции ($2 \leqslant n \leqslant 10^6$). Во второй строке записаны n целых положительных чисел из промежутка $[1; 10^9]$ — рейтинги участников. Гарантируется, что среди них есть популярный рейтинг.

Формат выходных данных

Выведите одно целое число — популярный рейтинг участников конференции.

Примеры

стандартный ввод	стандартный вывод
2 1 1	1
5 5 8 5 8 8	8

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, r_i	
1	20	$2 \leqslant n \leqslant 10^3$, $1 \leqslant r_i \leqslant 10^5$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	20	$2 \leqslant n \leqslant 10^5$, $1 \leqslant r_i \leqslant 10^7$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	30	$2 \leqslant n \leqslant 10^5$, $1 \leqslant r_i \leqslant 10^9$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.
4	30	$2 \leqslant n \leqslant 10^6$, $1 \leqslant r_i \leqslant 10^9$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Задача 4. Шестерёночки (7-11 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Наш мир состоит из множества не пригнанных друг к другу шестерёнок. И дело здесь не в механизмах, а в Часовщике. Не хватает Часовщика.

А. Сент-Экзюпери «Военный лётчик».

Даны n шестерёнок, некоторые из них соединены между собой. Две сцепленные шестерёнки могут вращаться только в раз-

ных направлениях. Вам необходимо выяснить, может ли вращаться *вся* система шестерёнок, и если может, указать *наименьшее* количество шестерёнок, которые нужно заставить вращаться.

Формат входных данных

В первой строке записаны два целых числа: n — количество шестерёнок и m — количество сцеплений между ними ($2 \leqslant n \leqslant 10^3$, $1 \leqslant m \leqslant 10^5$). В каждой из следующих m строк записаны два различных числа i и j, которые определяют номера сцепленных шестерёнок. Все шестерёнки пронумерованы целыми числами от 1 до n.

Формат выходных данных

В первой строке запишите одно число k — наименьшее количество шестерёнок, которые нужно заставить вращаться. В следующей строке k целых чисел — номера этих шестерёнок. Если решений несколько, выведите любое из них. Если запустить все шестерёнки невозможно, выведите `-1`.

Примеры

стандартный ввод	стандартный вывод
6 3 4 5 2 1 3 2	3 1 4 6
4 3 1 2 2 4 4 1	-1

Пояснение к примеру

В первом примере имеется $n = 6$ шестерёнок, между ними $m = 3$ соединения. Все они будут вращаться, если запустить три шестерёнки с номерами 1, 4 и 6.

Во втором примере *все* шестерёнки вращаться не смогут, поэтому в ответе `-1`.

Система оценивания

Номер подзадачи	Баллы	Ограничения n, m	Комментарии
1	20	$2 \leqslant n \leqslant 4$, $1 \leqslant m \leqslant 6$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	40	$2 \leqslant n \leqslant 100$, $1 \leqslant m \leqslant 10^3$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	40	$2 \leqslant n \leqslant 10^3$, $1 \leqslant m \leqslant 10^5$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Задача 5. Сортировочная (9-11 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`256 мегабайт`

Даны n целых чисел d_1, d_2, …, d_n ($2 \leqslant n \leqslant 20$). Вам разрешается менять местами любые два числа в этом наборе, если одно из них делится на другое.

Требуется отсортировать исходный набор по возрастанию, переставляя на каждом шаге только два числа.

Формат входных данных

В первой строке записано одно целое n — количество чисел в исходном наборе ($2 \leqslant n \leqslant 20$). Во второй строке записаны n различных целых положительных чисел из промежутка $[1; 10^9]$. Гарантируется, что количество требуемых перестановок не превышает 10^6.

Формат выходных данных

В первой строке запишите целое число k — количество перестановок, которое вам понадобилось для сортировки исходного набора чисел по возрастанию. В каждой из следующих k строк запишите по два целых числа из исходного набора, которые меняются местами в соответствии с правилами сортировки (одно из них обязательно должно делиться на другое). Если решений несколько, выведите любое из них. Если требуемую сортировку сделать невозможно, запишите `-1`.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	20	$2 \leqslant n \leqslant 4$	Баллы начисляются, если пройдены все тесты этой подзадачи.
2	20	$2 \leqslant n \leqslant 20$, $\min d_i = 1$	Баллы начисляются, если пройдены все тесты этой и предыдущей подзадачи.
3	60	$2 \leqslant n \leqslant 20$	Баллы начисляются, если пройдены все тесты этой и предыдущих подзадач.

Примеры

стандартный ввод	стандартный вывод
2 12 6	1 6 12
2 7 12	0
3 5 4 2	-1

Региональный этап, 2018-2019

Общая информация о проверке

Во всех задачах баллы за подзадачу начисляются только, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены. Информация о тестировании приведена в таблице.

Тип информации	Пояснение
Полная	Для каждого теста подзадачи сообщается результат работы программы участника на этом тесте.
Первая ошибка	Для подзадачи сообщается одно из двух: • если все тесты пройдены, то сообщаются баллы за подзадачу; • если хотя бы один тест не пройден, сообщается номер первого не прошедшего теста и результат работы программы участника на этом тесте.
Только баллы	Для подзадачи сообщаются баллы за эту подзадачу.

Задача 1. Два измерения

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

Ученые планируют провести важный эксперимент с использованием исследовательского модуля на планете X-2019. В процессе эксперимента будет проведено два измерения: основное и контрольное. Каждое измерение занимает ровно один час и должно начинаться спустя целое число часов после начала работы исследовательского модуля.

Данные эксперимента планируется немедленно передать на орбитальную станцию. Канал связи с орбитальной станцией будет установлен с l-го по r-й час от начала работы исследовательского модуля, включительно. Кроме того, согласно плану эксперимента между измерениями планета должна совершить целое число оборотов вокруг своей оси. Планета X-2019 осуществляет оборот вокруг своей оси за a часов.

Таким образом, если измерения осуществляются на i-м и j-м часу, то должны выполняться неравенства $l \leqslant i < j \leqslant r$, а величина $(j - i)$ должна быть кратна a. Теперь учёным необходимо понять, сколько существует различных способов провести измерения.

Требуется написать программу, которая по заданным границам времени измерений l и r и периоду обращения планеты вокруг своей оси a определяет количество возможных способов провести измерения: количество пар целых чисел i и j, таких что $l \leqslant i < j \leqslant r$, и величина $(j - i)$ кратна a.

Формат входных данных

Входные данные содержат три целых числа, по одному на строке: l, r и a ($1 \leqslant l < r \leqslant 10^9$, $1 \leqslant a \leqslant 10^9$).

Формат выходных данных

Выведите одно целое число — количество способов провести измерения.

Пример

стандартный ввод	стандартный вывод
1 5 2	4
4 9 6	0

Пояснение к примерам

В первом примере можно провести измерения в следующие пары часов: (1, 3), (1, 5), (2, 4), (3, 5).

Во втором примере продолжительности работы канала связи недостаточно, чтобы провести два измерения.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		l, r, a	
1	30	$1 \leqslant l < r \leqslant 100$, $1 \leqslant a \leqslant 100$	Информация о проверке — полная.
2	30	$1 \leqslant l < r \leqslant 10^5$, $1 \leqslant a \leqslant 10^5$	Информация о проверке — полная. Требуемая подзадача – 1.
3	40	$1 \leqslant l < r \leqslant 10^9$, $1 \leqslant a \leqslant 10^9$	Информация о проверке – полная. Требуемые подзадачи – 1, 2.

Задача 2. Полные квадраты

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

С целью поиска закономерностей иногда полезно сгенерировать длинную последовательность по определенным правилам. Известно, например, что последовательность 0, $0+1$, $0+1+3$, $0+1+3+5$, …, $0+1+3+\ldots+(2n-1)$, …, составленная из сумм нескольких первых нечетных натуральных чисел, состоит из квадратов целых чисел: 0, 1, 4, 9, …, n^2, … .

Обобщим эту последовательность следующим образом: будем использовать вместо начального значения не ноль, а число k. Получим последовательность: k, $k+1$, $k+1+3$, $k+1+3+5$, …, $k+1+3+\ldots+(2n-1)$, … . В отличие от случая $k=0$ в этой последовательности могут встречаться не только полные квадраты. Необходимо найти минимальное целое неотрицательное число, квадрат которого встречается в этой последовательности.

Требуется написать программу, которая по заданному целому числу k определяет, квадрат какого минимального неотрицатель-

ного целого числа встречается в описанной последовательности, либо выясняет, что в ней вообще не встречается полных квадратов.

Формат входных данных

В единственной строке содержится целое число k — начальное число в последовательности ($-10^{12} \leqslant k \leqslant 10^{12}$).

Обратите внимание, что для считывания и хранения такого большого числа необходимо использовать 64-битный тип данных.

Формат выходных данных

Выведите минимальное неотрицательное целое число, квадрат которого встречается в описанной последовательности. Если в последовательности не встречается квадратов целых чисел, выведите `«none»`.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		k	
1	7	$0 \leqslant k \leqslant 10^3$	Информация о проверке — полная.
2	10	$0 \leqslant k \leqslant 10^5$	Первая ошибка. Требуемая подзадача — 1.
3	27	$0 \leqslant k \leqslant 10^{12}$	Первая ошибка. Требуемые подзадачи — 1, 2.
4	7	$-10^3 \leqslant k \leqslant 10^3$	Информация о проверке — полная. Требуемая подзадача – 1.
5	10	$-10^5 \leqslant k \leqslant 10^5$	Первая ошибка. Требуемые подзадачи — 1, 2, 4.
6	39	$-10^{12} \leqslant k \leqslant 10^{12}$	Первая ошибка. Требуемые подзадачи — 1–5.

Пример

стандартный ввод	стандартный вывод
0	0
-5	2
2	none

Пояснение к примеру

В первом примере каждое число последовательности является полным квадратом. Минимальный из них — это 0, $0^2 = 0$.

Во втором примере последовательность начинается так: -5, -4, -1, 4, 11, 20, ... Минимальное неотрицательное число, квадрат которого встречается в последовательности — это 2, $2^2 = 4$.

В третьем примере последовательность начинается так: 2, 3, 6, 11, 18, ... В ней нет квадратов целых чисел.

Задача 3. Автоматизация склада

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

Компания занимается автоматизацией склада. На складе хранятся n видов товаров, пронумерованных от 1 до n, каждый вид товара хранится в своём помещении. Товар вида i хранится в помещении с номером i.

Специальный робот обслуживает запросы по получению товаров со склада. Для доступа в помещения склада робот использует специальные электронные карты. Карты у робота хранятся в специальном отсеке, из которого он может вынуть верхнюю карту. Вынутую карту робот может вернуть в отсек на любое место: на верхнюю позицию, между любыми двумя картами или на самую нижнюю позицию.

Чтобы открыть помещение, робот действует следующим образом. Он вынимает карты из отсека для их хранения и возвращает их обратно в отсек, пока на верхней позиции не окажется карта

от помещения, которое ему необходимо открыть. После этого, вынув эту карту, робот использует её, чтобы открыть помещение, и затем также возвращает в отсек для хранения карт. Если суммарно роботу потребовалось вынуть из отсека x карт, включая ту, которой он в итоге открыл помещение, будем говорить, что для открытия помещения робот совершил x действий.

В начале рабочего дня роботу поступил заказ на выдачу m товаров: a_1, a_2, …, a_m. Робот должен выдать товары именно в этом порядке. Для этого он последовательно выполняет следующие действия: открывает помещение, в котором лежит очередной товар, берет товар, закрывает помещение и выдаёт товар клиенту. После этого робот переходит к выдаче следующего товара.

Исходно электронные карты лежат в отсеке в следующем порядке, от верхней к нижней: b_1, b_2, …, b_n. Для каждого помещения в отсеке лежит ровно одна карта.

Время выдачи товаров со склада зависит от того, сколько раз суммарно роботу придётся вынимать верхнюю карту из отсека для их хранения, чтобы найти карту от очередного помещения. Необходимо таким образом выбрать места, куда робот должен возвращать вынутые карты, чтобы минимизировать суммарное количество действий робота для открытия помещений.

Требуется написать программу, которая по заданным целым числам n и m, последовательности выдаваемых товаров a_1, a_2, …, a_m и начальному положению карт в отсеке для хранения b_1, b_2, …, b_n определяет, какое минимальное количество действий придется совершить роботу, чтобы открыть все помещения в необходимом порядке. Для каждой вынутой карты необходимо также указать позицию, на которую её необходимо вернуть, чтобы добиться оптимального количества действий.

Формат входных данных

Первая строка входных данных содержит два целых числа n и m ($1 \leqslant n$, $m \leqslant 3 \cdot 10^5$) — количество видов товаров и количество товаров, которые необходимо выдать со склада.

Вторая строка содержит m целых чисел a_1, a_2, …, a_m ($1 \leqslant \leqslant a_i \leqslant n$) — типы товаров, которые необходимо выдать со склада, перечисленные в том порядке, в котором это необходимо сделать.

Третья строка содержит n различных целых чисел b_1, b_2, …,

b_n $(1 \leqslant b_i \leqslant n)$ — порядок, в котором карты исходно находятся в отсеке для их хранения, перечисленные от верхней к нижней.

Формат выходных данных

Первая строка должна содержать число k — минимальное количество действий, которое потребуется совершить роботу, чтобы выдать товары в заданном порядке.

Далее выведите k чисел. Для каждого действия робота выведите одно число — позицию, на которую ему следует вернуть вынутую карту в отсек для хранения. Если карта возвращается на самую верхнюю позицию, следует вывести 1; если после одной карты, — выведите 2, и так далее, для последней позиции следует вывести n.

Если существует несколько способов минимизировать суммарное число действий, выведите любой из них.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, m	
1	5	$1 \leqslant n \leqslant 5 \cdot 10^4$, $m = n$, $a_i = b_i$ $(1 \leqslant i \leqslant n)$	Информация о проверке — полная.
2	10	$1 \leqslant n \leqslant 5 \cdot 10^4$, $m = n$, $a_i = b_{n-i+1}$	Информация о проверке — полная.
3	31	$1 \leqslant n \leqslant 2000$, $1 \leqslant m \leqslant 2000$	Первая ошибка.
4	14	$1 \leqslant n \leqslant 5 \cdot 10^4$, $1 \leqslant m \leqslant 5 \cdot 10^4$, a_i — различные	Первая ошибка. Требуемые подзадачи — 1, 2.
5	14	$1 \leqslant n \leqslant 5 \cdot 10^4$, $1 \leqslant m \leqslant 10^5$	Первая ошибка. Требуемые подзадачи — 1, 2, 3, 4.
6	26	$1 \leqslant n \leqslant 3 \cdot 10^5$, $1 \leqslant m \leqslant 3 \cdot 10^5$	Первая ошибка. Требуемые подзадачи — 1, 2, 3, 4, 5.

Пример

стандартный ввод	стандартный вывод
1 1 1 1	1 1
4 5 4 1 2 4 4 4 3 2 1	7 4 4 2 4 4 1 4
2 2 1 2 2 1	3 2 2 2

Пояснение к примеру

Во втором примере карты в отсеке робота перемещаются следующим образом:

Действие	Перед действием	Извлечённая карта	Открытое помещение	Позиция, куда помещается карта	После действия
1	**4**, 3, 2, 1	4	4	4	3, 2, 1, **4**
2	**3**, 2, 1, 4	3	–	4	2, 1, 4, **3**
3	**2**, 1, 4, 3	2	–	2	1, **2**, 4, 3
4	**1**, 2, 4, 3	1	1	4	2, 4, **3**, 1
5	**2**, 4, 3, 1	2	2	4	4, 3, **1**, 2
6	**4**, 3, 1, 2	4	4	1	**4**, 3, 1, 2
7	**4**, 3, 1, 2	4	4	4	3, 1, 2, **4**

Задача 4. Машинное обучение

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`2 секунды`
Ограничение по памяти:	`512 мегабайт`

В лаборатории искусственного интеллекта разработали новый метод машинного обучения. В процессе обучения программы используется n итераций. Каждая итерация заключается в том, что обучаемая программа запускается на некотором обучающем наборе.

Были подготовлены обучающие наборы сложности от 0 до k. План обучения задаётся массивом целых чисел $[a_1, a_2, \ldots, a_n]$, где a_i задаёт сложность набора, используемого на i-й итерации обучения. Для всех i от 1 до n должно выполняться неравенство $0 \leqslant a_i \leqslant k$.

Выяснилось, что эффективность плана обучения зависит от битовых представлений сложностей обучающих наборов. Для эффективности плана необходимо, чтобы для любых двух значений i и j, где $1 \leqslant i < j \leqslant n$, выполнялось $(a_i \texttt{ and } a_j) = a_i$. Напомним, что побитовое «и» (`and`) двух целых неотрицательных чисел устроено следующим образом: запишем оба числа в двоичной системе счисления, i-й двоичный разряд результата равен 1, если у обоих аргументов он равен 1. Например, $(14 \texttt{ and } 7) = (1110_2 \texttt{ and } 0111_2) = 110_2 = 6$. Эта операция реализована во всех современных языках программирования, в языках `C++`, `Java` и `Python` она записывается как «`&`», в Паскале как «`and`».

Однако постоянное использование наборов одной сложности не даёт прогресса в обучении. Чтобы этого избежать, для плана обучения должны быть выполнены m требований следующего вида. Каждое требование задаётся двумя числами l_i и r_i и означает, что $a_{l_i} \neq a_{r_i}$.

Сотрудники лаборатории хотят найти количество эффективных планов, которые удовлетворяют всем требованиям. Так как это число может быть очень большим, нужно найти его остаток от деления на $10^9 + 7$.

Требуется написать программу, которая по заданным целым числам n и k, а также m требованиям вида l_i, r_i определяет количество эффективных планов, которые удовлетворяют всем требованиям, и выводит остаток от деления этого количества на число $10^9 + 7$.

Формат входных данных

Первая строка содержит три целых числа n, m и k — количество итераций обучения, количество требований и максимальную сложность обучающего набора ($1 \leqslant n \leqslant 3 \cdot 10^5$, $0 \leqslant m \leqslant 3 \cdot 10^5$, $0 \leqslant k \leqslant 10^{18}$).

Следующие m строк описывают требования, i-я строка содержит два целых числа l_i, r_i, которые означают, что $a_{l_i} \neq a_{r_i}$ ($1 \leqslant l_i < r_i \leqslant n$). Гарантируется, что все требования различны.

Формат выходных данных

Выведите одно целое число — остаток от деления количества эффективных планов, удовлетворяющих всем требованиям, на число $10^9 + 7$.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, m, k	
1	8	$1 \leqslant n \leqslant 500$, $m = 0$, $0 \leqslant k \leqslant 500$	Информация о проверке — полная.
2	20	$1 \leqslant n \leqslant 3 \cdot 10^5$, $m = 0$, $0 \leqslant k \leqslant 10^7$	Первая ошибка. Требуемые подзадачи — 1.
3	10	$1 \leqslant n \leqslant 3 \cdot 10^5$, $m = 0$, $0 \leqslant k \leqslant 10^{18}$	Первая ошибка. Требуемые подзадачи — 1, 2.
4	8	$1 \leqslant n \leqslant 50$, $0 \leqslant m \leqslant 50$, $0 \leqslant k \leqslant 50$	Первая ошибка.
5	16	$1 \leqslant n \leqslant 2000$, $0 \leqslant m \leqslant 2000$, $0 \leqslant k \leqslant 10^7$	Первая ошибка. Требуемые подзадачи — 1, 4.
6	6	$1 \leqslant n \leqslant 2000$, $0 \leqslant m \leqslant 2000$, $0 \leqslant k \leqslant 10^{18}$	Первая ошибка. Требуемые подзадачи — 1, 4, 5.
7	10	$1 \leqslant n \leqslant 3 \cdot 10^5$, $0 \leqslant m \leqslant 200$, $0 \leqslant k \leqslant 10^7$	Первая ошибка. Требуемые подзадачи — 1, 2, 4.
8	6	$1 \leqslant n \leqslant 3 \cdot 10^5$, $0 \leqslant m \leqslant 200$, $0 \leqslant k \leqslant 10^{18}$	Первая ошибка. Требуемые подзадачи — 1, 2, 3, 4, 7.
9	16	$1 \leqslant n \leqslant 3 \cdot 10^5$, $0 \leqslant m \leqslant 3 \cdot 10^5$, $0 \leqslant k \leqslant 10^{18}$	Первая ошибка. Требуемые подзадачи — 1–8.

Примеры

стандартный ввод	стандартный вывод
2 0 3	9
3 1 2 1 2	2

Пояснение к примерам

Все возможные планы для первого теста: $[0;0]$, $[0;1]$, $[0;2]$, $[0;3]$, $[1;1]$, $[1;3]$, $[2;2]$, $[2;3]$, $[3;3]$. Для второго теста: $[0,1,1]$, $[0,2,2]$.

Задача 5. Неисправный марсоход

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

Марсоход, осуществляющий международную миссию на Марсе, неисправен. Для восстановления его работоспособности необходимо повысить мощность его батареи.

Мощность батареи марсохода задаётся целым положительным числом. Текущая мощность батареи равна a, для восстановления работоспособности марсохода необходимо повысить её мощность до значения b. Для изменения мощности батареи на марсоход с Земли можно передавать специальные сигналы двух типов: `X` и `Y`. Сигнал типа `X` увеличивает текущую мощность батареи на 1, а сигнал типа `Y` увеличивает текущую мощность батареи на 2.

Организаторы миссии хотели бы изменить мощность батареи до необходимой, передав минимальное количество сигналов. К сожалению, из-за особенности устройства марсохода, если мощность батареи оказывается кратна целому числу c, он окончательно выходит из строя и перестаёт реагировать на сигналы.

Требуется написать программу, которая по заданным начальной мощности батареи a, необходимой мощности батареи b и це-

лому числу c определяет минимальное количество сигналов, которое необходимо передать на марсоход, чтобы восстановить его работоспособность.

Формат входных данных

Входные данные содержит три целых числа: a, b и c, по одному на строке ($1 \leqslant a < b \leqslant 10^9$, $2 \leqslant c \leqslant 10^9$, a не кратно c, b не кратно c).

Формат выходных данных

Требуется вывести одно целое число — минимальное количество сигналов, которые необходимо передать на марсоход.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		a, b, c	
1	25	$1 \leqslant a < b \leqslant 15$, $2 \leqslant c \leqslant 15$	Информация о проверке — полная.
2	25	$1 \leqslant a < b \leqslant 10^5$, $2 \leqslant c \leqslant 10^5$	Первая ошибка. Требуемые подзадачи – 1.
3	25	$1 \leqslant a < b \leqslant 10^9$, $c = 2$	Информация о проверке – полная.
4	25	$1 \leqslant a < b \leqslant 10^9$, $2 \leqslant c \leqslant 10^9$	Информация о проверке – полная. Требуемые подзадачи – 1, 2, 3.

Пример

стандартный ввод	стандартный вывод
2 7 3	3
4 10 3	4

Пояснение к примеру

В первом примере можно действовать следующим образом: отправить на марсоход сигналы `Y`, `X`, `Y`. Мощность батареи меняется следующим образом: $2 \to 4 \to 5 \to 7$.

Во втором примере можно действовать следующим образом: отправить на марсоход сигналы `X`, `Y`, `X`, `Y`. Мощность батареи меняется следующим образом: $4 \to 5 \to 7 \to 8 \to 10$.

Задача 6. Интервальные тренировки

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В академии физической культуры разработали новый метод интервальных тренировок спортсменов. В соответствии с этим методом спортсмен должен тренироваться каждый день, однако рост нагрузки должен постоянно сменяться её снижением и наоборот.

План тренировки представляет собой набор целых положительных чисел $a_1, a_2, \ldots, a_m$, где a_i описывает нагрузку спортсмена в i-й день. Любые два соседних дня должны иметь различную нагрузку: $a_i \neq a_{i+1}$. Чтобы рост нагрузки и её снижение чередовались, для i от 1 до $m-2$ должно выполняться следующее условие: если $a_i < a_{i+1}$, то $a_{i+1} > a_{i+2}$; если же $a_i > a_{i+1}$, то $a_{i+1} < a_{i+2}$.

Суммарная нагрузка в процессе выполнения плана должна составлять n, то есть $a_1 + a_2 + \ldots + a_m = n$. Ограничения на количество дней в плане нет — число m может быть любым, но нагрузка в первый день тренировок зафиксирована: $a_1 = k$.

Прежде чем приступить к тестированию нового метода, руководство академии хочет выяснить, сколько различных планов тренировок удовлетворяет описанным ограничениям.

Требуется написать программу, которая по заданным n и k определяет, сколько различных планов тренировок удовлетворяют описанным ограничениям, и выводит остаток от деления количества таких планов на число $10^9 + 7$.

Формат входных данных

В первой строке записаны целые числа n, k ($1 \leqslant n \leqslant 5000$, $1 \leqslant k \leqslant n$).

Формат выходных данных

Выведите одно число — остаток от деления количества планов тренировок на $10^9 + 7$.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n	
1	23	$1 \leqslant n \leqslant 10$	Информация о проверке — полная.
2	20	$1 \leqslant n \leqslant 30$	Первая ошибка. Требуемые подзадачи — 1.
3	23	$1 \leqslant n \leqslant 500$	Первая ошибка. Требуемые подзадачи — 1, 2.
4	34	$1 \leqslant n \leqslant 5000$	Только баллы. Требуемые подзадачи — 1, 2, 3.

Пример

стандартный ввод	стандартный вывод
6 2	4
3 3	1

Пояснение к примеру

В первом примере подходят следующие планы: [2, 1, 2, 1], [2, 1, 3], [2, 3, 1], [2, 4].

Во втором примере единственный подходящий план [3].

Задача 7. Экспедиция

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

Планируется отправить экспедицию к соседней звёздной системе. Были отобраны n кандидатов, пронумерованных от 1 до n, среди которых необходимо выбрать участников экспедиции.

Организаторы хотят отправить в экспедицию как можно больше кандидатов. Среди кандидатов был проведён опрос, в процессе которого каждый мог указать не более, чем одного из остальных кандидатов, с которым он не готов отправиться в экспедицию. Результатом опроса для i-го кандидата является целое число a_i, которое равно номеру кандидата, с которым i-й кандидат не готов отправиться в экспедицию, либо `-1`, если i-й кандидат готов отправиться в экспедицию в любом составе.

Теперь организаторы должны выбрать, кто из кандидатов отправится в экспедицию. Решено было выбрать участников экспедиции так, что если туда входит некоторый кандидат i, и $a_i \neq -1$, то туда не входит кандидат a_i. Организаторы хотят выбрать максимальное количество участников экспедиции.

Требуется написать программу, которая по заданным результатам опроса кандидатов определяет максимальное количество кандидатов, которых можно отправить в экспедицию.

Формат входных данных

В первой строке записано целое число n — количество кандидатов ($1 \leqslant n \leqslant 300\,000$).

В следующих n строках даны результаты опроса, i-я из этих строк содержит результат опроса i-го кандидата, целое число a_i ($a_i = -1$ или $1 \leqslant a_i \leqslant n$, $a_i \neq i$).

Формат выходных данных

Выведите одно целое число — максимальное количество кандидатов, которых можно отправить в экспедицию.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, a_i	
1	19	$1 \leqslant n \leqslant 20$	Информация о проверке — полная.
2	10	$a_1 = -1$, $a_i = i - 1$, $i > 1$	Первая ошибка.
3	15	$a_i < i$ для всех i	Первая ошибка. Требуемые подзадачи — 2.
4	13	$1 \leqslant n \leqslant 2\,000$	Первая ошибка. Требуемые подзадачи — 1.
5	43	$1 \leqslant n \leqslant 300\,000$	Первая ошибка. Требуемые подзадачи — 1, 2, 3, 4.

Пример

стандартный ввод	стандартный вывод
4 2 4 2 1	2
3 2 -1 2	2

Задача 8. Разбиение на пары

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`3 секунды`
Ограничение по памяти:	`512 мегабайт`

Космические археологи обнаружили на планете в соседней звездной системе n древних артефактов, которые они пронумеровали от 1 до n. Каждый артефакт имеет k различных параметров, каждый параметр характеризуется целым числом. Артефакт i имеет параметры $a_{i,1}$, $a_{i,2}$, …, $a_{i,k}$. Оказалось, что первые параметры у всех артефактов различны: для всех $i \neq j$ выполнено $a_{i,1} \neq a_{j,1}$, при этом другие параметры у артефактов могут совпадать.

Учёные также обнаружили текст, в соответствии с которым для активации артефактов их необходимо особым образом разбить на пары и совместить. Разбиение артефактов на пары является *корректным*, если для каждого t от 1 до k можно выбрать такое число b_t, что оно лежит на отрезке между значениями t-го параметра артефактов каждой пары. То есть если артефакты i и j образуют пару, должно выполняться условие $a_{i,t} \leqslant b_t \leqslant a_{j,t}$ или условие $a_{i,t} \geqslant b_t \geqslant a_{j,t}$.

Теперь ученые хотят выяснить, верно ли расшифрован текст. Для этого необходимо проверить, существует ли корректное разбиение артефактов на пары. Каждый артефакт должен войти ровно в одну пару в разбиении.

Требуется написать программу, которая по описанию параметров артефактов определяет, можно ли разбить их на пары так, чтобы для каждого параметра существовало значение, лежащее между значениями этого параметра артефактов каждой пары, и в случае положительного ответа выводит такое разбиение.

Формат входных данных

В первой строке заданы целые числа n и k — количество артефактов и количество параметров ($2 \leqslant n \leqslant 2 \cdot 10^5$, n — чётно, $1 \leqslant k \leqslant 7$).

В следующих n строках задано по k целых чисел $a_{i,1}$, $a_{i,2}$, …, $a_{i,k}$ — параметры артефактов ($-10^9 \leqslant a_{i,j} \leqslant 10^9$, все значения $a_{i,1}$ различны).

Формат выходных данных

Выведите «NO», если требуемого разбиения на пары не существует.

В противном случае выведите «YES» в первой строке. Далее выведите $n/2$ строк, в каждой строке выведите по два числа — номера артефактов, из которых следует составить пару. Каждый артефакт должен быть выведен ровно один раз.

Если существует несколько корректных разбиений артефактов на пары, выведите любое из них.

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, k	
1	10	$2 \leqslant n \leqslant 10$	Информация о проверке — полная.
2	7	$2 \leqslant n \leqslant 2 \cdot 10^5$, $k = 1$	Первая ошибка.
3	15	$2 \leqslant n \leqslant 2 \cdot 10^5$, $a_{i,t}$ различны для всех t	Первая ошибка. Требуемые подзадачи — 2.
4	15	$2 \leqslant n \leqslant 2 \cdot 10^5$, $1 \leqslant k \leqslant 2$	Первая ошибка. Требуемые подзадачи — 2.
5	26	$2 \leqslant n \leqslant 400$, $1 \leqslant k \leqslant 7$	Первая ошибка. Требуемые подзадачи — 1.
6	27	$2 \leqslant n \leqslant 2 \cdot 10^5$, $1 \leqslant k \leqslant 7$	Первая ошибка. Требуемые подзадачи — 1, 2, 3, 4, 5.

Пример

стандартный ввод	стандартный вывод
6 2 8 6 1 5 6 3 3 1 4 7 7 2	YES 1 4 2 6 3 5
4 3 1 -1 -1 2 1 1 3 -1 1 4 1 -1	NO

Пояснение к примерам

В первом примере артефакты можно разбить на пары следующим образом: $(8,6)-(3,1)$, $(1,5)-(7,2)$, $(6,3)-(4,7)$. При этом разбиении подходят, например, значения $b_1 = 4$, $b_2 = 4$.

Задача 9. Успешные пары (7-8 классы)

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Раиса Михайловна ведет кружок программирования. Сегодня на занятие пришли Аня, Боря, Вова и Гоша. Для подготовки к командному соревнованию необходимо разбить учеников на пары. У каждого школьника разный уровень подготовки. Раиса Михайловна оценивает уровень Ани в a баллов, уровень Бориса в b баллов, Вовы — c баллов, а Гоши — d баллов.

Для успешного выступления на соревновании суммарный уровень школьников в паре должен быть не меньше, чем L. Так считает Раиса Михайловна, и теперь ей нужно определить, ка-

кое наибольшее количество «успешных» пар можно образовать из школьников.

Формат входных данных

В первой строке даны 5 целых чисел — a, b, c, d и L. Все числа от 1 до 10.

Формат выходных данных

Выведите одно число — наибольшее число «успешных» пар: 0, 1 или 2.

Система оценивания

Баллы начисляются за каждый пройденный тест.

Примеры

стандартный ввод	стандартный вывод
4 4 1 2 5	2
2 3 1 1 6	0

Пояснение к примерам

В первом примере ответ можно получить, если разбить детей на пары, например, следующим образом: Аня с Вовой, а Боря с Гошей.

Задача 10. Наибольшее число (7-8 классы)

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	`1 секунда`
Ограничение по памяти:	`512 мегабайт`

В школе, где учится робот RT-2019, изучается тема «Конструирование чисел». На занятии роботам предлагается n карточек, на каждой из которых записано одно целое положительное число. Необходимо расположить этих карточки в ряд так, чтобы получившееся число было наибольшим из возможных. Не всякий робот может решить эту задачу, но человек легко справится с ней. Не правда ли?

Итак, по заданным n числам на карточках вам нужно найти наибольшее число, которое можно составить из них.

Формат входных данных

Первая строка содержит одно целое n — количество карточек ($1 \leqslant n \leqslant 10^4$). В следующей строке через пробел n целых положительных чисел a_i, записанных на карточках. Все числа не превосходят 10^6.

Формат выходных данных

Выведите наибольшее число, которое можно образовать из всех n карточек. (В ответе число не должно содержать пробелов.)

Примеры

стандартный ввод	стандартный вывод
2 32 4	432
1 100	100

Система оценивания

Номер подзадачи	Баллы	Ограничения	Комментарии
		n, a_i	
1	30	$1 \leqslant n \leqslant 10^3$, $1 \leqslant a_i \leqslant 10^4$	Только баллы.
2	30	$1 \leqslant n \leqslant 10^3$, $1 \leqslant a_i \leqslant 10^6$	Только баллы. Требуемые подзадачи – 1.
3	40	$1 \leqslant n \leqslant 10^4$, $1 \leqslant a_i \leqslant 10^6$	Только баллы. Требуемые подзадачи – 1, 2.

Решения задач

Задача 1. Конфеты

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

Подсчитаем количество конфет, одинаковое для *всех* коробок. Для этого найдём общее количество конфет во всех коробках $sum = a_1 + a_2 + \ldots + a_n$ и разделим на число коробок n. Если значение этого среднего арифметического — целое, то есть остаток от деления sum на n равен нулю, то Карлсону ничего съедать и перекладывать не придётся. В общем случае, остаток $eat := sum \bmod n$ определяет число конфет, которое нужно съесть Карлсону. Теперь оставшиеся конфеты можно разложить поровну между коробками. Для этого из каждой коробки, где число конфет a_i больше среднего арифметического $mean := sum \operatorname{div} n$, нужно переложить $a_i - mean$ конфет в коробки с меньшим числом конфет. Общее число перекладываемых конфет равно сумме по всем таким i.

```
eat := sum mod n;
mean := sum div n;
sum := 0;
for i := 1 to n do
    if a[i] > mean then Inc(sum, a[i] - mean);
write (eat, ' ', sum - eat);
```

Задача 2. Похожие числа

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

Для чисел a и b подсчитаем количество вхождений каждой цифры m от 0 до 9. Обозначим их через $d[a, m]$ и $d[b, m]$ соответ-

ственно. Например, цифра 1 входит в число $a = 1213$ два раза, поэтому $d[a, 1] = 2$. Для каждой цифры m от 0 до 9 осталось сравнить значения $d[a, m]$ и $d[b, m]$. Если для какой-то цифры m они совпадают, к степени похожести добавляем единицу.

Приведём основной фрагмент кода на языке `Pascal`:

```
readln(n);
for i := 1 to 2 do
    for j := 1 to n do begin
        read(m);
        Inc(d[i, m]);
    end;
sum := 0;
for i := 0 to 9 do
    if d[1, i] > d[2, i] then Inc(sum, d[2, i]) else Inc(sum, d[1, i]);
write(sum);
```

Задача 3. Штабеля

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

Все числа каждого набора отсортируем по возрастанию. Теперь осталось проверить, что каждое число набора не меньше суммы всех предыдущих. Если это условие выполняется для всех k чисел набора (кроме первого), выводим `yes`; иначе — `no`.

Оценим сложность этого алгоритма. Для сортировки массива из k элементов понадобится $O(k \log k)$ операций, для проверки «прочности» штабеля нужно еще $O(k)$ операций. Действительно, когда проверяется условие «прочности» для очередного числа $weight[i + 1]$ из набора, нам нужно вычислить сумму $sum[i]$ всех стоящих слева от $weight[i + 1]$ чисел, причём $sum[i]$ получается из предыдущего значение суммы $sum[i - 1]$ добавлением единственного слагаемого $weight[i]$. Значит, для проверки условия $sum[i] \leqslant weight[i+1]$ на очередном шаге потребуется всего две дополнительные операции. Поэтому проверка прочности штабеля

(после сортировки чисел) потребует еще $O(k)$ операций. Поскольку таких наборов n, общая сложность алгоритма $O(n \cdot k \log k)$.

Приведём основной фрагмент кода на языке `Pascal`:

```
flag := true;
sum := weight[1];
for i := 2 to k do
    if weight[i] < sum then begin
        flag := false; break;
        end;
    else Inc(sum, weight[i]);
if flag then writeln('yes') else writeln('no');
```

Задача 4. Цепочка вычитаний

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

Прежде всего, отметим, что на любом шаге среди чисел ровно одно нечётное. Вначале — это число 1. На очередном шаге вычитаются либо два чётных числа, либо чётное и нечётное числа, так что количество нечётных снова остаётся прежним (и равным одному). Поэтому последнее оставшееся число всегда будет нечётным. Значит, если требуемое число k — чётное, выводим `-1`.

Покажем теперь, как получить *любое нечётное* число $k < 2^n$.

Предположим, что мы умеем получать все нечётные $k < 2^{n-1}$ с помощью чисел 1, 2, 2^2, …, 2^{n-1}, и пусть теперь задан набор 1, 2, 2^2, …, 2^{n-1}, 2^n. Рассмотрим нечётные числа в промежутке от 1 до 2^n, отметим его середину — число 2^{n-1}.

Если заданное нечётное $k < 2^{n-1}$, то по предположению его можно получить с помощью чисел 1, 2, 2^2, …, 2^{n-1}. Но тогда его можно получить и с помощью исходного набора 1, 2, 2^2, …, 2^{n-1}, 2^n. Для этого достаточно на первом шаге заменить пару чисел 2^{n-1} и 2^n их разностью, равной $2^n - 2^{n-1} = 2^{n-1}$, и мы вновь приходим к набору 1, 2, 2^2, …, 2^{n-1}, из которого число k уже получается.

Если же нечётное $k > 2^{n-1}$, то заменим его на число $k' = 2^n - k < 2^{n-1}$, которое — опять же по предположению — можно получить из набора $1, 2, 2^2, \ldots, 2^{n-1}$. Поэтому сначала с помощью $(n-1)$ операций вычитания получаем число $k' = 2^n - k$, а затем последней операцией заменяем пару чисел 2^n и k' их разностью $2^n - k' = 2^n - (2^n - k) = k$.

Осталось правильно реализовать указанный алгоритм. Это можно сделать, например, с помощью *стека*.

Задача 5. Скобочки

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

Решение задачи основано на использовании динамического программирования. Пусть $m[n]$ — количество правильных скобочных последовательностей, содержащих n пар скобок. Произвольная правильная последовательность скобок длины $2n$ получается по правилу (ST), причём S и T — правильные последовательности.

Если S — пустая строка, то правильная скобочная последовательность T состоит из $(n-1)$ пар скобок. Количество скобочных последовательностей этого типа, очевидно, равно $m[n-1]$.

Пусть теперь S — непустая строка и содержит $k \geqslant 1$ пар скобок. Строка S обязательно начинается с открывающейся скобки «(», которой соответствует некоторая закрывающаяся скобка «)». Между этими скобками расположена правильная скобочная последовательность из $(k-1)$ пар скобок; их количество равно $m[k-1]$. После строки S записана правильная скобочная последовательностей T из $n-k-1$ пар скобок. (Здесь мы учли пару внешних скобок в исходной строке (ST).) Количество строк T равно $m[n-k-1]$. По правилу произведения общее число всех таких комбинаций равно $m[k-1] \cdot m[n-k-1]$. Суммируя по всем k от 1 до $n-1$, получим окончательную формулу для подсчета чисел $m[n]$:

$$m[n] = m[n-1] + \sum_{k=1}^{n-1} m[k-1] \cdot m[n-k-1],$$

с начальным условием $m[0] = 1$. Поскольку числа $m[n]$ растут достаточно быстро, вычисления по этой формуле нужно выполнять по модулю $(10^9 + 9)$.

Замечание. Последовательность $m[n]$ играет важную роль в дискретной математике. Эти числа называются *числами Моцкина*; они описывают количество непрерывных ломаных в верхней полуплоскости, составленных из векторов (1; 0), (1; 1), (1; −1) и соединяющих начало координат с точкой $(2n; 0)$. Известна явная формула чисел Моцкина через биномиальные коэффициенты (http://oeis.org/A001006):

$$m[n] = \sum_{k=0}^{[n/2]} \frac{1}{k+1} C_n^{2k} C_{2k}^k.$$

Асимптотический рост чисел Моцкина оценивается величиной $O(3^n n^{-3/2})$.

Задача 6. Эх, дороги...

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

Математическая формулировка задачи выглядит следующим образом.

Дан неориентированный граф. Требуется найти такой его подграф, который содержит наименьшее число рёбер и в котором все его вершины, достижимые *до* удаления рёбер, остаются достижимыми и после удаления «лишних» рёбер.

Ясно, что вершины графа разбиваются на несколько групп так, что из любой вершины одной группы можно попасть в любую другую вершину этой группы, и между разными группами пути не существует. Такая группа вершин называется *компонентой связности*. У исходного графа может быть, конечно, несколько таких компонент. После удаления рёбер в каждой компоненте связности не должно быть циклов, иначе можно будет удалить по крайней мере ещё одно ребро с сохранением условия достижимости. Связный граф без циклов называется *деревом*, а совокупность деревьев — *лесом*.

Таким образом, задача сводится к построению для каждой компоненты связности соответствующего дерева, которое соединяет все вершины этой компоненты. Такое дерево называется *минимальным остовом*.

Для решения подзадачи 1 подходят переборные алгоритмы.

Для решения подзадач 2 и 3 можно воспользоваться алгоритмом Крускала для построения минимального остовного дерева.

Вначале каждую вершину из одной компоненты помещаем в своё дерево, а затем соединяем эти деревья, объединяя на каждой итерации два дерева некоторым ребром. В процессе объединения перебираются все рёбра от первого до последнего, и если у текущего ребра его концы принадлежат разным поддеревьям, эти поддеревья объединяются, а ребро добавляется к ответу. По окончании перебора всех рёбер все вершины окажутся принадлежащими одному остову, и ответ получен.

Простейшая реализация этого алгоритма имеет асимптотику $O(m^2)$. Используя структуру данных «система непересекающихся множеств», можно получить более быструю реализацию алгоритма Крускала с асимптотикой $O(n + m)$. Более подробно познакомиться с деталями этой реализации и структурой данных «система непересекающихся множеств» можно на сайте:

http://e-maxx.ru/algo/mst_kruskal_with_dsu

Региональный этап, 2017-2018

Задача 1. Улучшение успеваемости

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

Основные темы задачи — *целочисленная арифметика, вывод формулы, двоичный поиск.*

Пусть ученик получит x оценок в 5 баллов. Тогда общее число оценок будет равно $a + b + c + x$, а средняя оценка будет равна $(2a + 3b + 4c + 5x)/(a + b + c + x)$. По условию задачи должно выполняться условие

$$\frac{2a + 3b + 4c + 5x}{a + b + c + x} \geqslant 3.5\,.$$

Возможны два подхода для поиска минимального x, удовлетворяющего этому неравенству.

Подход 1. [Вывод формулы.] Умножив обе части неравенства на положительную величину $2(a + b + c + x)$, получим неравенство

$$4a + 6b + 8c + 10x \geqslant 7a + 7b + 7c + 7x, \quad \Longleftrightarrow \quad 3x \geqslant 3a + b - c.$$

Таким образом, минимальное подходящее значение x равно округлённому вверх числу $(3a+b-c)/3$. Необходимо также учесть, что x не может быть меньше 0. Для вычисления соответствующего значения, например, в языке C++, можно воспользоваться выражением

$$x = \max((3 * a + b - c + 2)/3, 0);$$

Подход 2. [Двоичный поиск.] Заметим, что каждая дополнительная оценка в 5 баллов увеличивает среднюю оценку; следовательно, для поиска минимального значения x, при котором среднее значение будет не меньше 3.5, можно применить двоичный поиск. Для того чтобы избежать переполнения целочисленного типа, при вычислениях в обоих подходах следует использовать 64-битный тип данных.

Решения, которые используют линейный поиск вместо двоичного, подходят для решения подзадач 1, 2, 3 и 4. Для решения подзадач 1 и 2 можно использовать более простые формулы. Для решения подзадачи 3 можно вычислять каждый раз значение средней оценки, суммируя все оценки по одной.

Задача 2. Квадраты и кубы

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

Основные темы задачи — линейный поиск, операции с вещественными числами.

Решение может быть получено перебором по переменной y. Поскольку $a \leqslant y^3 \leqslant b \leqslant 10^{18}$, достаточно ограничить перебор значений y до 10^6. Убедимся, что $a \leqslant y^3 \leqslant b$ и найдем количество подходящих значений x.

Для фиксированного y значение x^2 должно лежать в диапазоне от $L = \max(y^3 - k, a)$ до $R = \min(y^3 + k, b)$: $L \leqslant x^2 \leqslant R$. Извлекая квадратный корень из всех частей неравенства, получаем ограничение на x: $\sqrt{L} \leqslant x \leqslant \sqrt{R}$.

Подсчитаем количество целых значений в интервале от $\sqrt{L}$ до $\sqrt{R}$. Для этого округлим вверх значение $\sqrt{L}$ — обозначим его через $l = \lceil \sqrt{L} \rceil$, и округлим вниз значение $\sqrt{R}$, пусть $r = \lfloor \sqrt{R} \rfloor$. Количество подходящих значений x для данного y, таким образом, равно $r - l + 1$.

Перебирая значения y и подсчитывая только подходящие значения x, получим ответ.

Для поиска квадратного корня из числа можно воспользоваться либо функцией `sqrt` для извлечения корня из вещественного числа, с последующим округлением, либо двоичным поиском.

Для решения подзадач 1 и 2, где $k = 0$, можно заметить, что для искомых пар $(x; y)$ выполняется равенство $x^2 = y^3$, то есть x^2 и y^3 являются шестыми степенями некоторого целого числа z, поэтому достаточно перебрать z до 10^3 и посчитать количество

вариантов, для которых $a \leqslant z^6 \leqslant b$. При этом для подзадачи 1 можно, вместо этого, перебирать значения от a до b, проверяя, что они являются шестой степенью некоторого целого числа, либо просто заметить, что $z^6 \leqslant 1\,000$ выполнено только для $z = 1$, $z = 2$ и $z = 3$.

В подзадаче 3 можно перебрать все потенциальные пары чисел от a до b, проверяя каждый раз, будут ли они числами вида x^2 и y^3 для некоторых целых x и y.

Для решения подзадачи 4 достаточно перебором найти все требуемые варианты значений x и y.

В подзадаче 5 можно перебором найти все значения y, как это сделано в полном решении, при этом отдельно проверяя каждое число от L до R, будет ли оно точным квадратом.

В подзадаче 6 перебором находим все значения y, как в полном решении, а для поиска подходящих значений x используем метод двух указателей.

Задача 3. Лифт

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

Основные темы задачи — моделирование, структуры данных, обработка событий.

Будем использовать алгоритм обработки событий. Создадим события для каждого факта «сотрудник подошёл к лифту», для каждого такого события запомним его время и номер сотрудника. Будем хранить события в структуре данных, поддерживающей добавление и извлечение элемента с минимальным ключом, в качестве ключа будем использовать время события. Для этого можно воспользоваться, например, структурой данных из стандартной библиотеки C++ `std::set`, либо самостоятельно реализовать структуру данных типа «приоритетная очередь».

Кроме событий «сотрудник подошёл к лифту» нам потребуются события «лифт прибыл на интересный этаж». Интересным при движении вверх будет только этаж, на котором сделан актив-

ный вызов, а при движении вниз — каждый этаж ниже текущего положения лифта, на котором находится хотя бы один ожидающий лифта сотрудник, а также первый этаж. Для каждого этажа будем хранить множество ожидающих лифта сотрудников.

Лифт может находиться в трех состояниях:

- на первом этаже в ожидании вызова;
- лифт двигается вверх к активному вызову;
- лифт двигается вниз на первый этаж.

Будем рассматривать события в порядке возрастания их времени, при необходимости добавляя новые события. Рассмотрим действия при обработке очередного события для каждого состояния лифта. Для любого состояния лифта, когда сотрудник подходит к лифту, будем добавлять этого сотрудника в множество ожидающих лифт на соответствующем этаже.

Если лифт находится на первом этаже и ожидает вызова, то единственный возможный тип события — «сотрудник подошел к лифту». Пусть это происходит на этаже s. При обработке этого события данный вызов становится активным, а лифт переходит в состояние «двигается вверх к активному вызову». Если текущее время — максимум из времени события и времени, когда лифт последний раз освободился на первом этаже, — равно t, то добавим событие «лифт прибывает на этаж s» с временем $t + s$.

Если лифт двигается вверх к активному вызову, то при обработке события «лифт прибыл на интересный этаж» необходимо сделать следующее: добавим в очередь события «лифт прибыл на интересный этаж» для всех этажей, расположенных ниже текущего положения лифта, на которых находятся ожидающие лифта сотрудники. Если время текущего события t, а этаж, на котором находится лифт — s, то время для события «лифт прибыл на интересный этаж a» равно $t + s - a$. Добавим также событие для первого этажа. Затем переместим всех находящихся на текущем этаже сотрудников из множества ожидающих лифт на этом этаже в множество перемещающихся на лифте, и перейдем к обработке следующего события.

Если лифт двигается вниз, то обработка события «сотрудник подошёл к лифту на этаже a», на котором в этот момент нет дру-

гих сотрудников, происходит следующим образом. Если лифт в этот момент находится не ниже этажа a, то необходимо добавить в очередь событие «лифт прибыл на интересный этаж a». Если $a > 1$, то при обработке события «лифт прибыл на интересный этаж a» все ожидающие лифт сотрудники на этом этаже перемещаются с этажа в лифт. Если $a = 1$, все сотрудники из лифта помечаются как обработанные, и время их прибытия на первый этаж равно времени рассматриваемого события.

В заключение описания алгоритма отметим, что при сравнении событий с одинаковым временем события «лифт прибыл на интересный этаж» должны идти после событий «сотрудник подошёл к лифту», а события типа «сотрудник подошёл к лифту» следует обрабатывать в порядке возрастания номера сотрудника.

Для решения подзадачи 1 можно применить простую формулу — ответом для первого и единственного сотрудника будет число $t_1 + 2(a_1 - 1)$.

Для решения остальных подзадач необходимо с различной эффективностью промоделировать описанный алгоритм обработки событий.

В подзадаче 2 возможно пошаговое моделирование каждой секунды описанного в условии процесса. При этом можно каждую моделируемую секунду произвольным разумным образом обрабатывать входные данные, например, просматривая всех сотрудников и проверяя, не находится ли он на этаже, где находится лифт.

В подзадаче 3 событий мало, поэтому каждый раз для обработки события можно выбирать очередное событие проходом по всем имеющимся событиям, использование быстрых структур данных не требуется.

В подзадаче 4, как и в подзадаче 2, возможно пошаговое моделирование процесса, но необходимо аккуратное хранение информации о сотрудниках, чтобы суммарное время работы было $O(T + n)$ или $O(T + n \log n)$, где T — максимальное время, за которое может произойти событие (порядка 10^8 в этой подзадаче).

Наконец, для решения последней пятой подзадачи необходима эффективная реализация моделирования, описанного в разборе, время работы — $O(n \log n)$.

Задача 4. Мониторинг труб

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

Это самая сложная задача первого тура, она относится к темам: деревья, алгоритмы на графах, динамическое программирование.

Переформулировав задачу математически, получим следующее условие: задано корневое дерево, на каждом ребре которого написана буква. Требуется покрыть все ребра дерева словами из заданного множества, причем слово покрывает путь вниз по дереву, буквы на котором при прочтении вдоль пути образуют это слово.

У этой задачи есть два принципиально разных решения: первое использует алгоритм поиска остовного дерева минимального веса в ориентированном графе, а второе основано на методе динамического программирования на поддеревьях. Рассмотрим оба решения.

Первое решение. Построим вспомогательный ориентированный граф. Вершины графа будут совпадать с узлами заданного во входном файле дерева, а ориентированные взвешенные ребра построим следующим образом. Если, начав от вершины u, можно пройти вниз по дереву по буквам слова s_i и попасть в вершину v, то добавим в граф ребро $u\,v$ с весом w_i. Также для каждой вершины u добавим ребро веса 0 к её родителю в исходном дереве p_u.

Легко убедиться, что минимальная стоимость проверки всех труб равна стоимости минимального ориентированного остова в получившемся графе, с корнем в корне исходного дерева. Для поиска минимального остова можно использовать алгоритм Чу Йонджина и Лю Цзенхонга, известный в русскоязычной литературе как «алгоритм двух китайцев». Его время работы в самой простой реализации составляет $O(n^3)$, что достаточно для этой задачи.

Отметим, что построение графа в подзадачах 1, 3 и 4 можно выполнить наивно, попытавшись отложить каждое слово от каж-

дой вершины, получив сложность $O(n^2m)$. В остальных подзадачах требуется использовать структуру данных «бор». А именно, сложим в бор все заданные слова и при обходе дерева от каждой вершины будем помнить, в какой вершине бора мы находимся. При таком подходе время работы этой части решения составляет $O(\sum \texttt{len}(s_i) + n^2)$.

Второе решение. Как и в предыдущем решении, найдём все пути, которые можно покрыть каким-либо словом.

Теперь воспользуемся методом динамического программирования. Рассмотрим какой-нибудь ответ на задачу. Это множество путей, которое покрывает все ребра дерева. Возьмём некоторое поддерево и рассмотрим подмножество путей из ответа, нижние вершины которых лежат в этом поддереве. Заметим, что такое подмножество путей покрывает все ребра поддерева и также некоторый путь выше по дереву до корня поддерева. Значит, за состояние алгоритма можно принять пару вершин (u, v), одна из которых находится выше другой: нижняя обозначает корень поддерева, которое покрыто целиком, а верхняя обозначает вершину, в которой заканчивается покрытый путь из корня. Обозначим через $dp[u][v]$ минимальную стоимость покрыть поддерево вершины v так, чтобы одновременно покрыть путь от u до v.

Будем считать значения одновременно для всех состояний с фиксированным v. Для листа $dp[u][v]$ равно минимальной стоимости слова, которым можно покрыть путь от u до v, если такое слово есть. Для внутренней вершины необходимо найти сначала значения для её детей.

Итоговое время работы — $O(n^2)$.

Для решения подзадачи 1 заметим, что каждое ребро необходимо проверять независимо, и значит, для каждой буквы можно просто выбрать минимальную стоимость, для которой слово совпадает с этой буквой, и просуммировать соответствующие стоимости для всех труб. Итоговое время работы — $O(n + m)$.

В подзадаче 2 возможны различные подходы, основанные на динамическом программировании на прямой. Также возможно, построив предварительно бор, как это указано в разборе, вместо минимального остовного дерева найти кратчайший путь от корня до листа. Итоговое время работы — $O(n^3 + m)$.

В подзадаче 3 помимо описанных в разборе решений возможно решение перебором или динамическим программированием с состоянием «множество покрытых ребер». Итоговое время работы — $O(2^n\, n^2)$.

Наконец, учитывая технические сложности восстановления ответа в обоих подходах, решения, которые только находят минимальную стоимость покрытия, проходят все подзадачи, кроме подзадачи 6.

Задача 5. Удаление чисел

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

Основные темы задачи — моделирование, целочисленная арифметика, идея.

Определим номер позиции, на которой будет стоять число с позиции t после одной операции удаления. Заметим, что если число t делится на k, оно будет удалено в результате этой операции. В противном случае будет удалено $t \operatorname{div} k$ чисел, меньших k; следовательно, число с позиции t после операции удаления переместится на позицию $(t - t \operatorname{div} k)$, где div обозначает целочисленное деление.

Проведем моделирование процесса, считая количество выполненных операций удаления. Будем поддерживать только номер позиции, который имеет число n. Если на очередном шаге удаления номер позиции делится на k, ответ найден. В противном случае продолжаем моделирование. Заметим, что любое число в итоге будет удалено, за исключением чисел от 1 до $k-1$. Поскольку по условию $k < n$, такой случай рассматривать не требуется.

Оценим максимальное число операций удаления. После одной операции удаления n умножается на величину $1 - 1/k \geqslant 0.99$. Следовательно, количество операций в худшем случае составляет порядка $\log_{1/0.99} 10^{18} \approx 4\,200$.

Для решения подзадач 1 и 2 с $k = 2$ можно внимательнее изучить, как происходит удаление чисел в этом случае. На первом

шаге будут удалены числа, кратные 2, на втором шаге — числа, кратные 2, после целочисленного деления на 2, и так далее. Число n будет удалено на шаге $t+1$, где t — количество единиц на конце двоичной записи числа n.

В подзадачах 1 и 3 можно непосредственно промоделировать описанный в условии процесс. При этом можно каждый раз сдвигать оставшиеся элементы, заполняя освободившиеся позиции.

В подзадаче 4 необходимо моделировать процесс более эффективно, например, храня элементы в списке или отмечая удаленные элементы в массиве, но не сдвигая оставшиеся элементы на освободившиеся позиции.

Задача 6. Старая книга

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

Основные темы задачи — двоичный поиск, два указателя, вывод формулы.

Начнем с описания решения первых двух подзадач, где $k = 0$.

Выясним, какое минимальное количество страниц могло быть в книге. Для этого будем последовательно суммировать натуральные числа, пока сумма $1 + 2 + 3 + \ldots + n$ не окажется больше или равной s. Возможны два случая.

Случай 1. Если $1 + 2 + 3 + \ldots + n = s$, то в книге может вообще не быть иллюстраций, все страницы содержат текст и на каждой из них написан её номер. В этом случае ответ равен 0.

Случай 2. Пусть $1 + 2 + 3 + \ldots + n = t > s$, тогда $t - s < n$, и значит, в книге существует страница с номером t. Если на этой странице находится иллюстрация, а остальные страницы содержат текст, то сумма номеров оставшихся страниц равна s. Таким образом, в этом случае ответ равен 1.

Перейдем теперь к решению задачи в случае $k > 0$. Пусть в книге $k+z$ иллюстраций; будем перебирать значение z, начиная с 0. Первые k иллюстраций по условию находятся на первых k страницах Выясним, можно ли разместить остальные z иллюстраций

так, чтобы сумма номеров страниц с текстом оказалась равной s. Пусть в книге n страниц, тогда сумма номеров страниц с текстом будет минимально возможной, если все z оставшихся иллюстраций находятся на последних z страницах, а максимальная — если эти иллюстрации занимают страницы с номерами $k+1$ по $k+z$. Таким образом, сумма номеров страниц с текстом t удовлетворяет условию:

$$(k+1)+(k+2)+\ldots+(n-z) \leqslant t \leqslant (k+z+1)+(k+z+2)+\ldots+n.$$

Заметим, что любое t в указанном диапазоне может быть получено как сумма номеров страниц с текстом. Действительно, начав с ситуации, когда все страницы с иллюстрациями имеют максимальные возможные номера, будем постепенно перемещать иллюстрации к началу, увеличивая каждый раз сумму номеров страниц с текстом на 1. Процесс завершится, когда все иллюстрации окажутся на первых страницах, а сумма номеров страниц с текстом будем максимальной возможной.

Для выполнения обоих неравенств будем перебирать z и n. Максимальное значение z имеет порядок $O(\sqrt{s})$, поэтому выполняя действия при фиксированном z за $O(1)$ или $O(\log s)$, мы получим достаточно эффективное по времени решение. Искомое значение n можно искать двоичным поиском за $O(\log s)$, либо, учитывая, что при увеличении z значение n также увеличивается, использовать метод двух указателей, параллельно увеличивая z и n.

В подзадачах 1 и 3 маленькие ограничения на k и s позволяют выполнять все действия с меньшей эффективностью.

Задача 7. Красота фейерверка

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

Основные темы задачи — динамическое программирование, обработка деревьев, обход в глубину.

Самый длинный путь в графе называется его *диаметром*.

Красота салюта, таким образом, равна числу вершин в диаметре заданного во входных данных дерева.

В первых двух подзадачах $m = 1$ и требуется найти диаметр явно заданного дерева. В первой подзадаче это можно сделать наивно. Запустим обход в глубину от каждого листа, выберем самый длинный путь, который в нём начинается, выберем самый длинный из рассмотренных путей.

Во второй подзадаче требуется более эффективное решение. Мы рассмотрим два способа поиска диаметра, один проще в реализации (хотя его корректность требует отдельного доказательства, которое мы оставим как упражнение), зато второй допускает обобщение, которое потребуется в полном решении.

Способ 1. Сначала запустим обход в глубину от любой вершины, например, от корня дерева и найдем наиболее удаленную от него вершину u. Затем найдем наиболее удаленную от u вершину v, запустив еще один обход в глубину, на этот раз от вершины u. Путь между u и v является диаметром дерева. Доказательство этого факта оставим как упражнение, основная идея доказательства — рассмотреть диаметр $x - y$ дерева и убедиться, что если u находится не ближе к корню, чем y, то путь $x - u$ не короче, чем путь $x - y$.

Способ 2. Используем динамическое программирование. Обозначим через $down[u]$ — длину максимального пути вниз по дереву, начинающегося в вершине u, а через $down2[u]$ — длину самого длинного из путей, начинающихся в вершине u, идущих вниз по дереву, и использующих в качестве первого ребро, отличное от того, которое использует самый длинный путь.

Тогда $down[u]$ равно $\max(1 + down[v])$, где максимум берется по всем v — детям вершины u, а $down2[u]$ равно второму максимуму среди этих величин. Если у вершины u нет детей, полагаем $down[u] = down2[u] = 0$; аналогично считаем $down2[u] = 0$ для вершин, у которых только один ребенок.

Ответ на задачу — это число вершин в диаметре дерева, и оно равно максимальному значению $down[u] + down2[u] + 1$ по всем вершинам u.

Перейдем теперь к полному решению для $m > 1$.

Диаметр любого дерева устроен следующим образом: он со-

ставлен из двух путей, ведущих от некоторой вершины x до листа, либо представляет собой путь от корня до листа.

Рассмотрим первый случай, когда диаметр составлен из двух путей. Заметим, что оптимально составить путь следующим образом: в том экземпляре дерева T, в котором находится вершина x, необходимо взять диаметр, а копиях T, которые подвешены к листьям, являющихся концами диаметра, необходимо взять максимальные по длине пути, начинающиеся в корне. Аналогично следует поступить и в копиях T, подвешенных к концам этих путей, и так далее. Таким образом, если количество вершин в диаметре дерева T равно d, а максимальный по длине путь от корня до листа содержит p вершин, то ответ равен $d + 2(m - 1)p$.

Если предположить, что диаметр представляет собой путь от корня до листа, то его оптимальная длина равна mp, заметим, что это значение не может превышать $d + 2(m - 1)p$ при $m > 1$, поэтому этот вариант не может быть лучше.

Диаметр дерева мы научились находить при решении второй подзадачи, а значение p равно $down[r]$ для корня дерева r.

При решении подзадач 3 и 4 можно воспользоваться наивным способом поиска диаметра дерева, подзадача 3 также допускает решение с помощью других алгоритмов динамического программирования.

Задача 8. Обработка больших данных

Автор задачи :	*Кунявский П.Е.*
Разработчик :	*Путилин М.*
Разбор задачи:	*Станкевич А.С.*

Основные темы задачи — динамическое программирование, структуры данных, оптимизации динамического программирования, операции с множествами.

Структура корректных отрезков, описанная в условии, напоминает устройство структуры данных «дерево отрезков». Будем строить дерево отрезков на числах от 0 до $2^n - 1$, объединяя вершины, соответствующие отрезкам, которые целиком находятся внутри заданных (во входных данных) отрезков с одинаковыми

значениями, сделав их листьями. Ясно, что нет смысла вызывать `STORE` от более мелких отрезков.

Пусть вершина u соответствует отрезку $[L, R]$. Докажем индукцией по $R - L$, что для заполнения всех значений в ячейках памяти отрезка $[L, R]$ и только их, можно в качестве первой выполнить операцию `STORE` для всего отрезка $[L, R]$. Если вершина u соответствует листу, то утверждение очевидно. Пусть у вершины u есть два ребенка v и w, которые соответствуют отрезкам $[L, M]$ и $[M + 1, R]$. Тогда, если одна и та же операция `STORE` используется для присваивания значений в обоих для v и w отрезках, то она должна быть первой и должна быть выполнена для отрезка $[L, R]$. В противном случае операции заполнения отрезков $[L, M]$ и $[M + 1, R]$ проводятся независимо, и по индукционному предположению первая операция `STORE` для отрезка $L, M]$ будет операцией инициализации для всего этого отрезка. Выполним вместо нее операцию `STORE` для отрезка $[L, R]$ с таким же значением, а затем остальные операции для отрезка $[L, M]$ и все операции для отрезка $[M+1, R]$. Заметим, что результат выполнения будет тем же, поскольку операции внутри отрезка $[M + 1, R]$ полностью заменят присвоенные этой операцией значения, а для отрезка $[L, M]$ эти операции неотличимы. Таким образом утверждение доказано.

Рассмотрим теперь решение с использованием динамического программирования. Обозначим через $opt[u][c]$ минимальное количество операций `STORE`, которое необходимо выполнить для заполнения отрезка памяти $[L, R]$, соответствующего вершине u в дереве отрезков, требуемыми значениями, при этом первым действием мы выполняем операцию `STORE` для отрезка $[L, R]$ и значения c.

Если у вершины u нет детей, то $opt[u][c] = 1$, если в итоговом заполнении памяти на позициях, соответствующих отрезку вершины u должно стоять значение c, и $opt[u][c] = 2$ в противном случае.

Пусть теперь у вершины u два ребенка v и w, тогда $opt[u][c]$ выбирается среди следующих значений: $opt[v][c]+opt[w][c]-1$ (единица вычитается, потому что мы выполняем одну операцию заполнения отрезка значением c вместо двух в отрезках детей), минимум величины $opt[v][c] + opt[w][d]$ по всем возможным значени-

ям d и минимум величины $opt[v][d] + opt[w][c]$ по всем возможным значениям d.

Время работы такого решения равно $O(nm^2 \log n)$, поскольку число вершин в рассматриваемом фрагменте дерева отрезков есть $O(n \log n)$, в каждой вершине хранится m значений и пересчет осуществляется за $O(m)$.

В таком варианте решение проходит только подзадачи 1, 2 и 3. Избавимся от одного из множителей m, чтобы решить подзадачу 4. Заметим, что $\min(opt[v][c] + opt[w][d])$ по всем возможным значениям d равен $opt[v][c] + min(opt[w][d])$, второе слагаемое не зависит от c и может быть посчитано один раз. Аналогично для $\min(opt[v][d] + opt[w][c])$. Получаем решение за $O(nm \log n)$, которое проходит подзадачи 1, 2, 3 и 4.

Для решения последних двух подзадач необходимо отметить ещё один важный факт. Величины $opt[u][c]$ для одного значения u и разных c различаются не более чем на 1.

Действительно, пусть $opt[u][d] = t$. Значит если первым действием мы заполняем весь отрезок, соответствующий вершине u, значениями d, то все значения могут быть получены за t операций. Но тогда если первым действием заполнить весь отрезок значениями c, а затем применить последовательность операций из оптимального решения для d, мы получим решение за $t + 1$ операцию. Значит, $opt[u][c] \leqslant opt[u][d] + 1$, что и требовалось.

Следовательно, вместо хранения значений $opt[u][c]$ для каждой вершины и каждого возможного значения, достаточно хранить $best[u]$, равное минимуму $opt[u][c]$ по всем значениям c и множество значений $bestv[u] = \{ c_1, c_2, \ldots, c_s \}$, для которых $opt[u][c_i]$ минимально и равно $best[u]$. Для остальных значений d значение $opt[u][d]$ равно $best[u] + 1$.

Если вершина u является листом, который должен быть заполнен значениями c, то $best[u] = 1$, $bestv[u] = \{ c \}$.

Иначе, пусть v и w — дети вершины u. Если есть значение c, которое входит одновременно в множества $bestv[v]$ и $bestv[w]$, то

$$best[u] = best[v] + best[w] - 1,$$
$$bestv[u] = bestv[v] \cap bestv[w].$$

Если же множества $bestv[v]$ и $bestv[w]$ не пересекаются, то $best[u] = best[v] + best[w]$, $bestv[u] = bestv[v] \cup bestv[w]$.

Заметим, что каждое значение попадает в *bestv* от листа, и далее может попасть только в множество *bestv* вершин на пути от листа до корня дерева отрезков, количество элементов которого имеет порядок $O(k)$. Пересечение и объединение множеств осуществляется за время, пропорциональное их размеру, таким образом, время работы решения равно $O(kn \log n)$.

В заключение отметим, что для решения подзадачи 1 можно использовать также переборные решения, а в подзадачах 2 и 5 вместо дерева отрезков — динамическое программирование для всех различных корректных отрезков.

Задача 9. Поменяй жизнь!

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

Основные темы задачи — графы, обходы графов в ширину и в глубину, поиск компонентов связности.

Математическая постановка задачи следующая. Задан *граф дружбы* жителей города. За один шаг можно удалить все рёбра, выходящие из выбранной вершины a графа, и добавить рёбра, ведущие из a в остальные вершины. Требуется за наименьшее число шагов соединить рёбрами *все* вершины графа. Вывести соответствующий этому процессу список вершин.

Если граф дружбы сделать полным нельзя, вывести `-1`.

Пусть все жители города подружились за s дней.

За день до события «все подружились» ровно один житель a враждовал со всеми оставшимися. Пусть M — группа жителей, уже подружившихся между собой, и N — группа жителей, враждующих с ними. За 2 дня до события «все подружились» еще один житель b из M ссорится со всеми жителями из M и переходит в группу N (рис. 4).

Продолжим этот процесс: в любой момент каждый житель входит в M или в N, причём *все* жители из M дружат между собой и *все* жители из N дружат между собой. Повторяя этот процесс, приходим к необходимому и достаточному условию для выполнения события «все подружились»:

Событие «все подружились» может наступить только тогда, когда исходный граф дружбы состоит ровно из двух полных подграфов. Наименьшее число дней, когда все подружатся, равно числу жителей в наименьшем полном подграфе (компоненте связности).

Реализация проверки указанного условия основана на обходе заданного графа в ширину или в глубину. Выбираем произвольную вершину графа a и помечаем все вершины a_1, a_2, …, a_d, связанные с вершиной a, где d — количество «друзей» вершины a. Затем аналогичным образом помечаем все вершины, связанные с каждой вершиной a_i, и так далее. Этот процесс продолжается до тех пор, пока мы не пометим все вершины, входящие в компоненту связности вершины a. Если после этого какие-то вершины графа остались непомеченными, выбираем любую из них и запускаем обход в глубину из этой вершины.

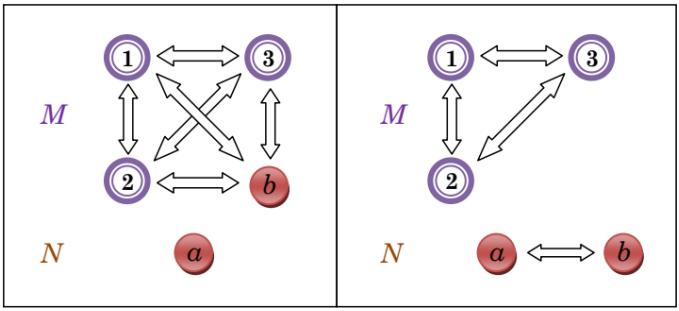


Рис. 4. Переход вершины b из группы «друзья» в группу «враги» вершины b.

Таким образом, мы сможем подсчитать количество компонент связности нашего графа. Если оно отлично от двух, выводим ответ `-1`. Если же компонент связности ровно две, остаётся проверить, что каждая компонента связности представляет собой полный подграф.

Для этого достаточно убедиться, что количество «друзей» каждой вершины, входящей в компоненту связности, на единицу меньше числа вершин этой компоненты. Это и будет означать, что все вершины из одной компоненты связаны ребром «дружбы»

друг с другом. Если это так, то ответ в задаче положительный — событие «все подружились» возможно. (Для этого все вершины из одной компоненты связности должны поменять свою жизнь, то есть «перейти» в другую компоненту связности.) Наименьшее количество дней, необходимое для этого, равно количеству вершин в наименьшей компоненте связности.

Время работы указанного решения — $O(n^2)$.

Задача 10. Карандаши

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

Основные темы задачи — алгоритмы перебора, дерево отрезков, два указателя.

Пусть n — общее число карандашей, k — число наборов.

Все карандаши имеют два параметра a и b — цвет и размер, сопоставим каждому карандашу пару чисел (a, b).

Подзадача 1. [«Грубое» решение.] Сравним координаты каждой пары с координатами всех других пар. Если существуют хотя бы два карандаша с разными первыми и разными вторыми координатами, то соответствующие два карандаша образуют бракованную пару. Иначе бракованных пар нет. Сложность этого алгоритма $O(k \cdot n^2)$. Это решение набирает 20 баллов.

Подзадача 2. [Сортировка.] Отсортируем все пары (a_i, b_i), сначала — в порядке возрастания первых координат, затем — в порядке возрастания вторых координат. Если первая и последняя пары после упорядочивания совпадают, бракованных пар в наборе нет. Если же у них разные первые и разные вторые координаты, соответствующая им пара карандашей бракованная. Сложность алгоритма $O(k \cdot n \log n)$. Это решение на 50 баллов.

Подзадача 3. [Предподсчёт.] Для каждого карандаша заранее найдём ближайший следующий, который образует бракованную пару с ним. При очередном запросе для каждого карандаша на отрезке смотрим, попадает ли следующий бракованный

с ним в отрезок. Сложность алгоритма $O(k \cdot n)$. Это решение набирает 70 баллов.

Подзадача 4. [Оптимизация.] Для каждого карандаша заранее найдём следующий $(nexta_i, nextb_i)$, бракованный с ним. При очередном запросе для каждого карандаша на отрезке с L по R рассмотрим три элемента:

$$L, \quad nexta_L \quad \text{и} \quad nextb_L.$$

Если $nexta_L \leqslant R$ и $nextb_L \leqslant R$, то из этих трёх элементов найдётся подходящая пара, иначе — такой пары нет. Сложность алгоритма $O(k + n)$. Это решение на 100 баллов.

Муниципальный этап, 2018-2019

Задача 1. Карточная игра

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

Основные темы этой простой задачи — счётчики, операции с циклами, целочисленное деление.

В этой игре первый игрок выиграет в том случае, когда общее количество сделанных ходов будет *нечётным*. Иначе выиграет второй игрок. Для заданного набора карточек подсчитаем количество возможных ходов. Для этого подсчитаем, сколько раз входит в исходный набор та или иная карточка. Это можно сделать двумя способами.

Подзадачи 1-2. Выберем одну из карточек и будем просматривать все карточки, расположенные справа от неё. Если найдём совпадающую с ней, увеличим значение счётчика на единицу. После прохода по всему массиву мы будем знать, сколько раз число входит в данный набор. Поделив его на 2, найдём количество ходов, которое можно сделать с этим числом. Затем перейдём к следующей карточке и так далее, пока не просмотрим все числа. Просуммировав все количества ходов, мы определим общее число ходов. Осталось проверить его чётность. Если это число чётное, выигрывает Джерри, выводим `2`. Иначе — выводим `1`.

Сложность такого алгоритма $O(n^2)$, поэтому это решение проходит первые две группы тестов, в которых значение n не превосходит 50 000.

Подзадача 3. Определим массив `count[]` из 10^5 чисел, в котором значение `count`$[i]$ равно количеству чисел, равных i. Это значение будем подсчитывать сразу при считывании чисел, записанных на карточках набора. После этого, как уже отмечалось, нужно подсчитать общее число ходов и определить его чётность.

Приведем фрагмент кода основной части программы на языке `Pascal`:

```
for i := 1 to n do begin
    read(a);
    inc(count[a]);
end;
sum := 0;
for a := 1 to 100 000 do
    sum := sum + count[a] div 2;
write(2 - sum mod 2);
```

Задача 2. Супердвоичная система счисления

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

Основные темы задачи — системы счисления, работа с массивами цифр.

Найдём запись в супердвоичной системе счисления нескольких первых натуральных чисел.

n	1	2	2^2	2^3	2^4
1	+				
2		+			
3	−		+		
4			+		
5	+		+		
6		−		+	
7	−			+	
8				+	
9	+			+	
10		+		+	
11	−		+		+
12			−		+
13	+		−		+
14		−			+
15	−				+

Заметим, что если число n — чётное, то запись числа $n = 2m$ получается из супердвоичного представления числа m «сдвигом» на один разряд (то есть добавлением нуля к записи числа m).

Если n — нечётно, то $a_0 = \pm 1$, и $a_1 = 0$. Цифра a_0 выбирается так, чтобы число $n - a_0$ делилось на 4. Супердвоичная запись числа $n = 4m + a_0$ получается из записи меньшего числа m «сдвигом» на два разряда и добавлением справа цифры a_0.

Во всех этих случаях единственность представления числа n следует из единственности представления меньшего числа m. (Единственность представления числа 1 очевидна.)

Приведем фрагмент кода основной части программы на язы-

ке `Pascal`:

```
k := 0;
while n > 0 do begin
    inc(k);
    r := n mod 2;
    if r = 1 then  a[k] := -n mod 4 + 2;
    if r = 0 then  a[k] := 0;
    n := (n - a[k]) div 2;
end;
for i := k downto 1 do
    write(a[i], ' ');
```

Задача 3. Популярный рейтинг

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

Основные темы задачи — эффективный поиск подстроки в строке, поиск шаблона, алгоритм Бойера-Мура.

Подзадачи 1-2. Выберем одно из чисел и будем просматривать все числа справа от него, подсчитывая на каждом шаге количество совпадающих с ним чисел. Как только обнаружим число, которое встречается более чем $n/2$ раз, задача решена. Таким образом, понадобится порядка n^2 операций. Это решение проходит первые две группы тестов.

Подзадача 3. Отсортируем исходный массив чисел и будем сравнивать соседние числа массива. В случае их совпадения будем увеличивать значение счётчика на единицу, подсчитывая, таким образом, количество вхождений в массив данного числа. Если значение счётчика оказывается больше $n/2$, популярный рейтинг найден. В противном случае, переходим к просмотру следующего числа. Поскольку сортировка требует $O(n \log n)$ операций, и затем еще $O(n)$ операций для просмотра элементов отсортированного массива, в итоге получаем алгоритм за $O(n \log n)$ операций. Это решение проходит первые три группы тестов.

Подзадача 4. [Алгоритм Боера-Мура.] Рассмотрим первое число массива, будем считать его «претендентом» на звание популярного числа.

1. Присвоим значение 1 счётчику числа вхождений этого числа в массив.
2. Увеличим значение счетчика на 1, если очередное считываемое число, совпадает с «претендентом» и уменьшаем на 1, если не совпадает. Так делаем, пока счётчик не станет равным 0, или не закончится поток.
3. Если после какого-то шага счетчик принимает значение 0, следующим шагом выполняем шаг 1.
4. То число, которое является «претендентом» в момент окончания потока, и есть искомый популярный элемент.

Это решение проходит все 4 группы тестов. Приведём основной фрагмент кода на языке `Pascal`:

```
for i := 1 to n do begin
    read(r);
    if counter = 0 then begin
      majority := r;
      counter := 1;
    end
    else
      if r = majority then inc(counter);
      else dec(counter);
end;
write(majority);
```

Задача 4. Шестерёночки

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

Задача составлена по мотивам фольклорной задачи Санкт-Петербургской олимпиады школьников 1995 года. Основные те-

мы задачи — графы, обходы в ширину и в глубину, компоненты связности.

Сначала составим граф, описывающий соединение шестерёнок. Будем говорить, что две шестерёнки связаны между собой *ребром*, если они соединены друг с другом. Если одна из шестерёнок вращается по часовой стрелке, то все соединенные с ней шестерёнки должны вращаться против часовой стрелки. Шестерёнки, соединённые с ними, — снова по часовой, следующие — против, и так далее.

Перейдем к исследованию графа соединения шестерёнок. С помощью *обхода в ширину* или *в глубину* назначим метки всем вершинам графа. Выберем одну из вершин в качестве начальной и присвоим ей метку `0`. Вершины, соединённые с ней ребром, получат метку `1`, а вершины, соединенные с ними, — снова метку `0`; они вращаются в том же направлении, что и начальная шестерёнка.

При входе в очередную вершину мы помечаем, что мы в ней были, и *рекурсивно* попадаем во все соседние с ней вершины, в которых ещё не были. Если мы попали в вершину, в которой уже побывали, сравниваем её метку с той, которую она должна получить. Если они не совпадают, система вращаться не может, и мы выводим `-1`. Иначе, мы получили *компоненту связности* графа, соответствующую исходной начальной вершине. (Это подсистема шестерёнок, которые вращаются после запуска начальной шестерёнки.) Выводим на печать номер начальной шестерёнки, с которой начинали обход графа, и продолжаем просмотр непомеченных вершин, пока не пометим все вершины графа.

Оценим сложность алгоритма. Для составления и считывания графа нам понадобится $O(m)$ шагов. Для нахождения всех вершин, связанных с заданной вершиной, то есть находящихся в одной с ней компоненте связанности, понадобится $O(n)$ операций. Такое же (по порядку) количество шагов нам придётся сделать, чтобы просмотреть все остальные компоненты связанности. Таким образом, итоговая сложность алгоритма — $O(n + m)$.

Задача 5. Сортировочная

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

Основные темы задачи — специальные сортировки массива, графы, кратчайшие пути в графе.

Подзадача 1. Решение для первой группы тестов предполагает несложный перебор возможных вариантов расположения чисел и аккуратное программирование списка переставляемых пар (не более восьми шагов).

Подзадача 2. Если среди чисел исходного набора встречается единица, то переставить любые два элемента a и b, не меняя положение остальных чисел, можно за три шага:

Положение чисел:	Переставляем пары:
$a \ldots b \ldots 1$	a 1
$1 \ldots b \ldots a$	1 b
$b \ldots 1 \ldots a$	1 a
$b \ldots a \ldots 1$	

Теперь упорядочить числа набора можно, например, с помощью алгоритма сортировки *пузырьком*. Сначала найдём число 1 и поставим его на первое место. Затем найдём следующее (после 1) *наименьшее* число и поставим его на второе место указанным выше способом — с помощью трёх перестановок. При этом все остальные числа, включая 1, не меняют своего положения. Затем поставим на третье место следующее наименьшее число и так далее. Сложность этого алгоритма — $O(n^2)$. Конечно, это решение можно улучшить, используя быструю сортировку, в этом случае его сложность будет $O(n \log n)$.

Полное решение. Рассмотрим граф, в котором вершины соответствуют данным числам, а рёбра — отношению делимости между ними. Другими словами, две вершины соединим ребром в том и только в том случае, если для соответствующих им чисел a и b или a делится на b или b делится на a. Создадим вектор-копию

`copy[]` исходного набора и отсортируем его числа по возрастанию. Теперь легко определить, на какое место в графе нужно переместить каждое число исходного набора.

Пусть a и b — две вершины в графе, для которых соответствующие числа нужно поменять местами так, чтобы одно из этих чисел заняло требуемое место, определяемое отсортированной копией набора. С помощью обхода в ширину находим *кратчайший путь* от a до b. (Если такого пути не существует, то есть число в вершине a нельзя поставить на требуемое место b, выводим `-1`.) Пусть этот путь состоит из s рёбер:

$$a - v_1 - v_2 - v_3 - \ldots - v_{s-1} - b.$$

Опишем два рекурсивных алгоритма перестановки чисел, соответствующих вершинам a и b.

Алгоритм за 2^{s-1} шагов. Несложно найти способы перестановки для небольших значений длины пути между вершинами a и b. Например, если длина пути равна 1, то это можно сделать за один шаг. Если длина пути равна 2, это можно сделать за 3 шага. Предположим, что за k шагов мы умеем решать задачу перестановки для пути длиной s. Тогда в пути

$$(a - v_1 - v_2 - v_3 - \ldots - v_{s-1} - v_s) - b$$

из $s+1$ рёбер выделим путь от a до v_s, переставим числа в вершинах a и v_s (k шагов): $v_s \quad v_1 - v_2 - v_3 - \ldots - v_{s-1} \quad a \quad b$, затем за один шаг переставим числа b и v_s:

$$b \quad \overbrace{v_1 - v_2 - v_3 - \ldots - \underbrace{v_{s-1} \quad a}} \quad v_s,$$

Теперь мы можем ещё раз воспользоваться рекурсией и ещё за k шагов поменять числа в вершинах a и v_s пути

$$a - v_1 - v_2 - v_3 - \ldots - v_{s-1} - v_s,$$

Таким образом, после $k + 1 + k = 2k + 1$ шагов мы приходим к расположению:

$$b - v_1 - v_2 - v_3 - \ldots - v_{s-1} - v_s - a.$$

Несложно вывести формулу для подсчета количества шагов в общем случае. Для $s = 3$ получим $2 \cdot 3 + 1 = 7$ шагов, для $s = 4$ получим $2 \cdot 7 + 1 = 15$ шагов и так далее. Общее число шагов равно $2^s - 1$.

Алгоритм за $2s - 1$ шагов. Приведём более эффективный рекурсивный способ перестановки двух чисел, соответствующих вершинам a и b. Основная идея состоит в циклическом сдвиге чисел вдоль пути между вершинами. Проиллюстрируем её на примере пути длины $s = 3$:

$$a - v_1 - v_2 - b\,.$$

Сначала за 2 шага осуществим циклический сдвиг вправо первых трёх элементов: от $a - v_1 - v_2$ придём к расположению $v_2\ a - v_1$:

Положение чисел:	Переставляем пары:
$a - v_1 - v_2 - b$	$a \quad v_1$
$v_1 - a \quad v_2 - b$	$v_1 \quad v_2$
$v_2 \quad a - v_1 \quad b$	

Теперь поменяем местами числа в вершинах v_2 и b:

$$b \quad a - v_1 - v_2\,,$$

и затем ещё за 2 шага осуществим циклический сдвиг влево чисел вдоль пути $a - v_1 - v_2$, то есть получим расположение $b \quad v_1 - v_2 \quad a$, общее число шагов $2 + 1 + 2 = 2 \cdot s - 1$.

Оценим сложность алгоритма. Сортировка исходного набора требует $O(n \log n)$ операций перестановок. Каждая перестановка двух элементов требует порядка $O(s) = O(n)$ шагов, поэтому итоговая сложность — $O(n^2 \log n)$.

Региональный этап, 2018-2019

Задача 1. Два измерения

Автор задачи :	*Сафонов И.*
Разработчик :	*Анопренко М., Станкевич А.С.*
Разбор задачи :	*Станкевич А.С.*

Для удобства оценки времени работы решений обозначим через $n = r - l + 1$ количество возможных моментов, в которые можно проводить измерения.

Для решения первой подзадачи достаточно перебрать все пары часов, в которые проводятся измерения: первое — в час i от l до $r - a$, второе — в час j от $i + a$ до r. Для каждой пары проверим, что разность $(j - i)$ делится на a, и в случае положительного результата проверки увеличим ответ на 1. Время работы этого решения — $O(n^2)$.

Избавимся от перебора времени второго измерения. Будем перебирать время первого измерения i, при этом второе измерение могло проводиться в моменты $i + a$, $i + 2a$, ..., которые не превышают r. Количество таких моментов равно $\lfloor (r - i)/a \rfloor$, где $\lfloor x \rfloor$ означает округление числа x вниз. С точки зрения реализации это означает, что $(r - i)$ следует нацело поделить на a. Такое решение имеет сложность $O(n)$, и оно подходит для первых двух подзадач.

```
long long ans = 0;
for (int i = l; i <= r; i++) {
    ans += (r - l) / a;
}
```

Следует обратить внимание на использование 64-битного типа, поскольку суммарное количество пар может быть порядка 10^{18}.

Заметим, что возможных значений i всё еще слишком «много» для того, чтобы уложиться в ограничение по времени для третьей подзадачи. Оптимизируем предложенное выше решение.

Рассмотрим два значения i_1 и $i_2 = i_1 + a$. Заметим, что множества всех подходящих пар измерений вида $(i_1; j)$ и $(i_2; j)$ различаются только наличием в первом случае пары $(i_1; i_2)$.

Пар вида $(i_1; j)$ подходит $k = \lfloor (r - i)/a \rfloor$, следовательно пар вида $(i_2; j)$ подходит $(k - 1)$. Значит, чтобы учесть в ответе все пары измерений с $i = i_1$, $i = i_1 + a$, $i = i_1 + 2a$, и так далее, надо прибавить к ответу $k + (k - 1) + (k - 2) + \ldots + 1 = \frac{1}{2}k(k + 1)$.

Будем перебирать i от l до $l + a - 1$. Для каждого значения i воспользуемся описанным выше методом, чтобы обработать все значения вида $i + ta$. Это решение имеет сложность $O(a)$, но и оно все еще недостаточно производительное.

```
long long ans = 0;
for (int i = l; i <= r && i < l + a; i++) {
    long long k = (r - i) / a;
    ans += k * (k + 1) / 2;
}
```

Сделаем теперь заключительное наблюдение. Для первого значения i выполняется $k = \lfloor (r - l)/a \rfloor$, для последнего значения $k = \lfloor (r - (l + a - 1))/a \rfloor$. Эти числа различаются не более чем на 1. Уменьшение значения на 1 происходит после того, как разность $(r - i)$ становится кратной a. Таким образом, количество чисел i, для которых используется исходное значение k, равно $(r - l + 1)$ `mod` a; остальные используют значение k на единицу меньше. С учётом сказанного получаем решение сложности $O(1)$:

```
long long big = (r - l + 1) % a;
long long small = a - big;
long long k = (r - l + 1) / a;
long long ans = big * k * (k + 1) / 2 + small * k * (k - 1) / 2;
```

Задача 2. Полные квадраты

Автор задачи : *Христенко О.Б.*
Разработчик : *Филиппов Д.*
Разбор задачи: *Станкевич А.С.*

В условии задачи отмечается тождество для суммы последовательных *нечётных* чисел $1+3+\ldots+(2n-1)=n^2$. Пусть k — заданное целое число, которое прибавляется к каждому элементу исходной последовательности. Предположим, что требуемый квадрат целого числа достигается после добавления числа k к сумме из a последовательных нечётных натуральных чисел, то есть $k+a^2=b^2$. Тогда $b^2-a^2=(b+a)(b-a)=k$, причём $a\geqslant 0$ и $b\geqslant 0$.

Сразу отметим, что в случае $k=0$ подходит ответ $b=0$.

Пусть теперь $k\neq 0$. В подзадачах 1–3 значение k больше 0, и значит, $b>a$, то есть $b\geqslant a+1$. Следовательно, $k=(b+a)(b-a)\geqslant \geqslant b+a\geqslant a$. Таким образом, достаточно рассмотреть значения a, не превосходящие k.

В подзадаче 1 будем перебирать значения a по возрастанию, для каждого получившегося числа будем проверять, является ли оно полным квадратом; первое найденное число и будет ответом. Из приведенного рассуждения следует, что если среди первых k чисел полного квадрата не оказалось, то ответа не существует. Время $O(k^2)$ или $O(k^3)$.

В подзадаче 2 необходимо выполнять тот же процесс немного эффективнее. Будем перебирать значения a по возрастанию, однако при этом, во-первых, мы не будем каждый раз пересчитывать сумму, а во-вторых, будем следить, как в методе двух указателей, за значением максимального квадрата целого числа, меньшим или равным значению текущей суммы. Совпадение этих двух значений означает, что ответ найден. Время работы $O(k)$.

Аналогичные рассуждения остаются верными и для подзадач 4 и 5, однако требуется оценить максимальное возможное значение a для отрицательных k. Поскольку $k<0$, $k=(b+a)(b-a)$, то $b<a$. Значит, $|\,k\,|=(b+a)(a-b)\geqslant b+a\geqslant a$. Таким образом, достаточно осмысленно перебирать значения a только до $|\,k\,|$.

Отметим в заключение этой части, что для получения баллов за подзадачи 1, 2, 4 и 5 можно и не доказывать оценку на максимальное значение a и просто выполнить, например, 10^6 итераций вычисления суммы.

Полное решение. Напомним, что требуется найти *минимальное* неотрицательное целое b, для которого существует a, та-

кое что $k = (b + a)(b - a)$. Пусть вначале $k > 0$, как в подзадачах 1–3. Найдём все делители числа k, это можно сделать за $O(\sqrt{k})$.

Числа $b + a$ и $b - a$ — делители числа k. Обозначим $b + a = v$ и $b - a = u$, тогда $v \cdot u = k$ и $u \leqslant v$. Отсюда $b = \frac{1}{2}(u + v)$, $a = \frac{1}{2}(v - u)$. Значения a и b будут целыми, только если делители u и v числа k имеют одинаковую чётность. (Из этого, в частности, следует, что если $k \bmod 4 = 2$, то решения нет, хотя этот факт и не потребуется для полного решения.)

```
long long ans;
void relax(long long u, long long v, long long k)    {
    if (u * v != k) return;
    a = (v - u) / 2;
    b = (v + u) / 2;
    if (a >= 0 && b >= 0 && b < ans)    {
        ans = b;
    }
}
. . . . . .
for (long long u = 1; u * u <= abs(k); u++)    {
    if (k % u == 0)    {
        long long v = abs(k) / u;
        if (u % 2 == v % 2)    {
            relax( u,  v, k);
            relax( u, -v, k);
            relax(-u,  v, k);
            relax(-u, -v, k);
            relax( v,  u, k);
            relax( v, -u, k);
            relax(-v,  u, k);
            relax(-v, -u, k);
        }
    }
}
```

Итак, если целые неотрицательные u и v имеют одинаковую чётность, $u \leqslant v$ и $uv = k$, то число $b = \frac{1}{2}(u + v)$ — один из полных квадратов, которые встречаются в последовательности.

Переберём все такие варианты и выберем минимальный.

Наконец, рассмотрим случай $k < 0$. Пусть снова u и v целые, $u \cdot v = k$, $u < v$, u и v имеют одинаковую чётность. Заметим, что на этот раз числа u и v имеют разные знаки, и значит, $u < 0$, $v > 0$. Тогда число $b = \frac{1}{2}(u + v)$ — один из полных квадратов, встречающихся в последовательности, переберём такие варианты и выберем из них минимальный.

Время работы получившихся решений $O(\sqrt{k})$.

Отметим, что для обхода трудностей с рассмотрением случаев можно проверять обе подходящие комбинации знаков u и v, как, например, в приведенном выше коде.

Задача 3. Автоматизация склада

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

В подзадаче 1 карты в отсеке для хранения уже лежат в том порядке, в котором роботу необходимо открывать помещения. Каждый раз помещая карту на последнее место, он не испортит порядок остальных карт; поэтому для решения этой подзадачи достаточно n раз вывести число n.

В подзадаче 2 карта от первого помещения, которое необходимо открыть, лежит последней, поэтому перед тем как её вынуть, необходимо вынуть все остальные карты. Они в свою очередь лежат в обратном порядке, следовательно, если в процессе перемещения изменить порядок следования карт, далее каждый раз очередная карта будет первой в отсеке и лишних действий выполнено не будет. Для того, чтобы выполнить разворот последовательности, извлекая i-ю по счёту карту, робот должен вернуть её на $(n - i + 1)$-е место. После того, как эта операция будет проделана $n - 1$ раз, карты в отсеке лежат в том порядке, в котором необходимо открывать помещения, и следует действовать как в подзадаче 1.

Для решения оставшихся подзадач необходимо сделать ключевое наблюдение. Заметим, что после того как робот взял из отсека карту, он всегда может положить её обратно в отсек та-

ким образом, чтобы его нужно было вынимать из отсека только чтобы открыть дверь в соответствующее помещение.

Чтобы достичь этого, будем действовать следующим образом. Пусть робот извлек карту от помещения номер i из отсека для их хранения и необходимо вернуть её обратно. Рассмотрим все номера помещений, которые будут открыты до очередного момента, когда необходимо будет открыть помещение с номером i, пусть это помещения j_1, j_2, …, j_k. Рассмотрим места, где в отсеке для хранения лежат карты от этих помещений. Поместим нашу карту сразу после последней из них. Несложно доказать по индукции, что при таком алгоритме возвращения карт никакая карта не будет извлечена из отсека для хранения, кроме как для открывания двери, более одного раза.

Наивная реализация описанного алгоритма работает за $O(n^2)$ и решает подзадачу 3. Для решения подзадач 4, 5 и 6 необходимо выполнять указанные действия эффективнее.

Первым действием построим последовательность, в которой из отсека будут извлекаться карты. Для этого рассмотрим по очереди все помещения, которые надо открыть. Если карта для этого помещения уже извлекалась, то наш алгоритм разместит её таким образом, чтобы надо было извлечь только её. Иначе будем извлекать карты из начала отсека, пока не дойдём до карты от требуемого помещения.

```
vector<int> k;
vector<bool> used(n);
for (auto x : a)    {
    if (!used[x])    {
        while (true)    {
            int y = b.front();
            bool end = y == x;
            k.push_back(y);
            used[y] = true;
            b.pop();
            if (end)  break;
        }
    } else k.push_back(x);
}
```

Мы получили массив $[k_1, k_2, \ldots, k_t]$, в котором хранится информация, в каком порядке карты будут извлекаться из отсека. Теперь рассмотрим очередную карту k_i, которая была извлечена; пусть следующее её вхождение в массив k на позиции j: $k_j = k_i$, $j > i$. Тогда позиция, на которую её следует поместить, в точности равна количеству различных значений между позициями i и j, включительно.

Получили множество запросов вида «найти количество различных значений на отрезке» в постановке оффлайн. Это стандартная задача, которую можно решить с использованием корневой декомпозиции, дерева отрезков или декартова дерева. В зависимости от эффективности реализации тесты для подзадачи 6 могут пройти или не пройти по времени.

Наконец, дополнительное ограничение подзадачи 4, что все значения a_i различны, позволяет немного упростить реализацию в части построения множества отрезков, на которых необходимо искать количество различных элементов.

Задача 4. Машинное обучение

Автор задачи :	*Филиппов С.*
Разработчик :	*Саутин Д., Филиппов С.*
Разбор задачи :	*Станкевич А.С.*

Немного переформулируем условие эффективности плана. Будем требовать, чтобы равенство $(a_i \texttt{ and } a_{i+1}) = a_i$ было верно для любого $1 \leqslant i \leqslant n - 1$. Нетрудно видеть, что данное условие равносильно условию исходной задачи.

Теперь легко понять, какой вид имеют все эффективные планы. Массив, описывающий эффективный план, разбивается на последовательные блоки, состоящие из одинаковых чисел, причем битовые представления элементов последующих блоков являются надмасками битовых представлений элементов предыдущих блоков. Также заметим, что количество блоков не может превышать $\log_2 k$ — двоичного логарифма числа k, так как битовое представление элемента очередного блока содержит строго больше единиц, чем битовые представления элементов предыдущих блоков.

Этого достаточно, чтобы научиться решать подзадачи 1 и 4 с помощью идеи динамического программирования. Для подзадачи 1 можно посчитать значения $dp[i][j]$ — количество искомых массивов длины i, таких что последний элемент равен j. Для подзадачи 4 придётся добавить еще один параметр в динамическое программирование и посчитать значения $dp[i][j][k]$ — количество искомых массивов длины i, таких что последний элемент снова равен j, а k — индекс последнего элемента не равного j. Данной информации достаточно, чтобы при пересчете новых значений через предыдущие гарантировать выполнение дополнительных требований к массиву, а именно требований неравенства некоторых пар элементов.

Для решения остальных подзадач необязательно хранить значение последнего элемента в явном виде. Ключевым наблюдением является то, что про последнее число достаточно знать количество единиц в его битовом представлении. Предположим, что мы научились считать значение $dp[i][j]$ — количество массивов длины i, которые удовлетворяют всем условиям задачи, причём количество единиц в битовом представлении последнего числа в массиве равно j. Важно, что мы не уточняем, какое именно число стоит на последней позиции; мы считаем, что оно фиксировано, причём нам не важна величина этого числа. Тогда для решения задачи достаточно перебрать x в диапазоне от 0 до k — последнее число в искомом массиве, и прибавить к ответу значение $dp[n][cnt(x)]$, где функция $cnt(x)$ возвращает количество единиц в битовом представлении x.

Научимся считать значения $dp[i][j]$ из предыдущего абзаца.

```
dp[i][j] = 0;
if (check(1, u))  dp[i][j] += 1;
for (int p = 1; l < i; p++) {
    if (check(p + 1, i))  {
        for (int h = 0; h < j; h++) {
            dp[i][j] += dp[p][h] * C[j][h];
        }
    }
}
```

Функция check(l, r) проверяет, можно ли сделать все элементы отрезка $[l, r]$ равными, не нарушив требований неравенства некоторых пар элементов. Для этого достаточно проверить, что не существует такого требования i, что $l \leqslant l_i < r_i \leqslant r$.

Изначально проверяется, можно ли сделать все элементы с 1-го по i-й равными. Если — да, то прибавим 1 к $dp[i][j]$. Таким образом мы учли массивы, состоящие из одного блока одинаковых элементов. Теперь считаем, что массив состоит из нескольких блоков и будем перебирать значения p — конца предыдущего блока. С помощью функции check$(p+1, i)$ проверим, может ли конец предыдущего блока находиться в позиции p, и если это так, переберем значения h — количество единиц в битовом представлении числа предыдущего блока и добавим к ответу величину $dp[p][h] \cdot C(j, h)$, где функция $C(n, k)$ возвращает C_n^k — количество способов выбрать k элементов из n. В самом деле, при фиксированном числе с j единицами в битовом представлении существует ровно C_j^h чисел, которые содержат h единиц в битовом представлении и при этом являются подмаской данного числа.

Мы научились решать задачу при больших k, но все еще делаем это медленно, чтобы решить остальные подзадачи. Наша цель — избавиться от хранения количества единиц в битовом представлении последнего числа. Для этого добавим в динамическое программирование еще один параметр, который будет отвечать за количество блоков, на которые разбивается искомый массив. Теперь у нас считываются значения $dp[i][j][c]$ — количество массивов длины i, которые удовлетворяют всем требованиям и при этом состоят из c блоков одинаковых элементов, причём элемент последнего блока имеет j единиц в битовом представлении. Формулы пересчёта почти не меняются по сравнению с предыдущим решением; если раньше мы вычисляли значение $dp[i][j]$ через $dp[p][h]$, то теперь будем пересчитывать $dp[i][j][c]$ через значения $dp[p][h][c-1]$.

Теперь удалим параметр j из динамического программирования и перестанем при пересчете чисел $dp[i][c]$ умножать их на $C(j, h)$.

Код примет такой вид:

```
for (int i = 1; i <= n; i++) {
    if (check(1, i)) {
        dp[i][1] += 1;
    }
    for (int j = 2; j <= 60; j++) {
        for (int p = 1; p < i; p++) {
            if (check(p + 1, i)) {
                dp[i][j] += dp[p][j - 1];
            }
        }
    }
}
```

В приведённом коде подсчитывается количество разбиений массива на заданное число блоков, так чтобы выполнялись все требования на неравенства заданных пар элементов. Предыдущие варианты решения помогут нам понять, на что нужно умножить получившиеся значения, чтобы решить исходную задачу. Раньше, когда мы к блоку элементов с h единицами в битовом представлении дописывали блок элементов с j единицами в битовом представлении, то мы домножали количество способов на C_j^h. Значит, если мы теперь зафиксируем количество блоков b и массив $(c_1, c_2, \ldots, c_b)$, означающий количество единиц в битовом представлении элемента соответствующего блока, а также число c с b единицами в битовом представлении, которое будет последним элементов искомого массива, то к ответу нужно добавить величину $dp[n][b] \cdot C_{c_2}^{c_1} \cdot C_{c_3}^{c_2} \cdot \ldots \cdot C_{c_b}^{c_{b-1}}$.

Посчитаем массив $\texttt{cnt}[j][b]$, в котором будут храниться значения сумм $C_{c_2}^{c_1} \cdot C_{c_3}^{c_2} \cdot \ldots \cdot C_{c_b}^{c_{b-1}}$ по всем возможным массивам $(c_1, c_2, \ldots, c_b)$. Такие массивы должны удовлетворять двум условиям: элементы в них должны идти по возрастанию, и последний элемент должен быть равен j. Это также можно сделать с помощью динамического программирования. Теперь количество эффективных массивов, состоящих из b блоков, у которых в последнем блоке стоит зафиксированное число с j единицами в битовом представлении, равно $dp[n][b] \cdot \texttt{cnt}[j][b]$. Если еще посчитать

значения `cntBits`$[j]$ — количество чисел от 0 до k с j единицами в битовом представлении, то ответ на задачу может быть посчитан в следующей лаконичной форме:

```
int ans = 0;
for (int j = 0; j <= 60; j++) {
    for (int b = 1; b <= min(n, j + 1); b++) {
        ans += dp[n][b] * cnt[j][b] * cntBits[j];
    }
}
```

Для того, чтобы получить полное решение, достаточно научиться быстро считать массив `cntBits`, быстро вычислять функцию `check`(l, r) и избавиться от вложенного цикла по переменной p в вычислении $dp[i][j]$, заметив, что нас интересует сумма значений $dp[p][j-1]$ для p, принадлежащих некоторому промежутку.

В итоге можно получить решение, требующее $O(n \cdot \log k + \log^2 k)$ времени и $O(n \cdot \log k)$ памяти, которое можно оптимизировать до $O(n)$ памяти.

Отметим, что в приведённых блоках кода опущено взятие по модулю, в действительности при любом вычислении нужно брать результат по модулю и использовать 64-битный тип данных, чтобы избежать переполнения.

Задача 5. Неисправный марсоход

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

Для решения подзадачи 1 можно использовать полный перебор вариантов очередного сигнала. Время работы полного перебора составляет $O(2^{b-a})$ и не превышает 2^{15}.

Для решения подзадачи 2 можно использовать динамическое программирование. Обозначим через $dp[i]$ минимальное число сигналов, необходимое, чтобы мощность батареи марсохода составляла i.

Тогда $dp[i] = \texttt{min}(dp[i-1], dp[i-2]) + 1$, если i не кратно c, и $dp[i] = +\infty$, если i кратно c. Время работы этого решения $O(b-a)$.

Для подзадачи 3, где $c = 2$, оптимальна следующая последовательность: необходимо $\frac{1}{2}(b-a) = 2$ раз отправить на марсоход сигнал `Y`; поскольку b и a оба нечётны, все промежуточные значения будут нечётными.

Перейдем теперь к полному решению.

Рассмотрим два соседних значения, кратных c и числа между ними: ck, $ck+1$, $ck+2$, …, $c(k+1)$. Поскольку величина мощности ck запрещено, первое из значений, которое возможно для мощности батареи, равно $ck+1$. Будем далее отправлять на марсоход сигналы `Y`, пока мощность не примет одно из двух значений: $ck+k-2$ или $ck+k-1$. В первом случае далее следует подать сигнал `X`, чтобы мощность приняла значение $ck+k-1$. Теперь, подав сигнал `Y`, мы переводим мощность в следующий отрезок чисел между кратными c.

Всего для преодоления такого отрезка потребуется $\lfloor (c+1)/2 \rfloor$ сигналов. (Здесь $\lfloor x \rfloor$ — целая часть числа x, то есть наименьшее целое, не превосходящее x.)

Осталось разобраться с начальным отрезком от a до первого числа, кратного c, и с заключительным отрезком от последнего числа, кратного c, до b. Это можно сделать аналогично. Наконец, следует иметь ввиду случай, когда между a и b совсем нет чисел, кратных c; в этом случае ответ равен $\lfloor (b-a)/2 \rfloor$.

Задача 6. Интервальные тренировки

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

Для решения подзадачи 1 можно применить рекурсивный перебор вариантов.

Тот же метод решает и подзадачу 2 при использовании аккуратного перебора с отсечениями. Если для n порядка 30 перебор не успевает найти ответ за отведенное ограничение по времени, можно применить метод предподсчёта: заранее на своём компью-

тере найти ответы для всех тестов с $1 \leqslant k \leqslant n \leqslant 30$ и отправить на проверку программу, которая выводит найденный заранее ответ.

Для решения подзадач 3 и 4 необходимо применить динамическое программирование. Рассмотрим сначала решение за $O(n^3)$, которое решает подзадачу 3.

Обозначим через $up[n][k]$ количество тренировочных планов, которые удовлетворяют описанным ограничениям с суммарной нагрузкой n, с нагрузкой k в первый день и нагрузкой во второй день строго больше нагрузки в первый. Аналогично введём величину $down[n][k]$, для количества таких планов, где нагрузка во второй день строго меньше нагрузки в первый день.

Заметим, что $up[n][n] = down[n][n] = 1$. Для $k < n$ значения можно вычислить по формулам:

$$up[n][k] = \sum_{i=k+1}^{n-k} down[n-k][i],$$

$$down[n][k] = \sum_{i=1}^{k-1} up[n-k][i].$$

Величина, которую требуется найти по условию задачи, равна $up[n][k] + down[n][k]$; не забудем также отдельный случай $n = k$, когда ответ равен 1.

Для подзадачи 4 решение за $O(n^3)$ слишком медленное, для её решения можно использовать два подхода.

Подход 1. [Другая формула пересчёта.] Заметим, что формула для $up[n][k]$ отличается от $up[n][k+1]$ только слагаемым $down[n-k][k+1]$, поэтому

$$up[n][k] = up[n][k+1] + down[n-k][k+1],$$

Аналогично,

$$down[n][k] = down[n][k-1] + up[n][k-1],$$

Теперь значения вычисляются за $O(1)$ и время подсчёта всех значений есть $O(n^2)$. Следует обратить внимание, что значения

$up[n][k]$ следует вычислять по убыванию k, а значения $down[n][k]$ — по возрастанию k.

Подход 2. [Префиксные суммы.] Введём дополнительные величины:

$$sumUp[n][k] = \sum_{i=1}^{k} up[n][i],$$

$$sumDown[n][k] = \sum_{i=1}^{k} down[n][i].$$

Тогда

$$up[n][k] = sumDown[n-k][n-k] - sumDown[n-k][k],$$
$$down[n][k] = sumUp[n][k-1].$$

Отметим, что во всех случаях необходимо аккуратно следить за тем, чтобы индексы значений в формулах не выходили за пределы массивов, и за порядком вычислений.

Задача 7. Экспедиция

Автор задачи :	*Андреева Е.В.*
Разработчик :	*Будин Н.*
Разбор задачи :	*Станкевич А.С.*

Для решения подзадачи 1 можно перебрать все подмножества кандидатов и для каждого подмножества проверить, удовлетворяет ли оно требованиям к составу экспедиции. Время работы такого решения — $O(n2^n)$.

В подзадаче 2 в экспедицию можно взять либо всех кандидатов с нечётными номерами, либо всех кандидатов с чётными номерами. Первое выгоднее, в этом случае в экспедицию отправятся $\lfloor (n+1)/2 \rfloor$ кандидатов.

Для решения подзадач 3, 4 и 5 переформулируем задачу в терминах теории графов. Построим ориентированный граф, у которого вершинами будут кандидаты в экспедицию; проведём из вершины i ребро в вершину a_i, если $a_i \neq -1$. Поскольку в экспедицию нельзя одновременно брать кандидатов i и a_i, множество отправленных в экспедицию кандидатов должно образовывать *независимое множество* в графе.

В подзадаче 3 получается подвешенное дерево, где корень — вершина 1, а рёбра ориентированы к корню. Для поиска независимого множества в дереве можно использовать жадный алгоритм или динамическое программирование. Опишем идею жадного алгоритм. Удалим ориентацию рёбер в графе и будем действовать следующим образом. Выберем все листья дерева в множество, удалим их и все инцидентные им вершины, а затем повторим этот процесс. Докажем, что предложенный алгоритм — корректный. Предположим, что лист i не входит в множество. Если соседняя к нему вершина a_i также не выбрана, то этот лист i можно включить в независимое множество, увеличив его размер. Если же соседняя вершина a_i включена в независимое множество, то выберем лист i вместо a_i, при этом размер независимого множества не изменится.

Рассмотрим теперь решение с использованием динамического программирования на подвешенном дереве. И хотя это решение избыточное для задачи на дереве, оно пригодится нам для решения подзадач 4 и 5. Пусть $with[u]$ — размер максимального независимого множества в поддереве вершины u, содержащего вершину u, и пусть $out[u]$ — размер максимального независимого множества в поддереве вершины u, не содержащего вершину u. Тогда

$$with[u] = \sum_{v-\text{сын } u} out[v],$$

$$out[u] = \sum_{v-\text{сын } u} \texttt{max}(with[v], out[v]).$$

Для листа $with[u] = 1$, $out[u] = 0$.

Теперь ответ для дерева равен $\texttt{max}(with[r],\ out[r])$ для корня дерева r.

Перейдем теперь к решению задачи для подзадач 4 и 5, где граф не обязан быть деревом. Заметим, что из каждой вершины выходит не более одного ребра. Каждая компонента связности такого графа представляет собой либо дерево, либо цикл, к каждой вершине которого может быть подвешено дерево. Решим задачу независимо для каждой компоненты связности и сложим ответы для них. Для компонент, которые представляют собой дерево, решение уже описано выше.

Рассмотрим теперь компоненту, которая содержит цикл. Удалим рёбра цикла и решим задачу для каждого из получившихся деревьев. Осталось решить следующую задачу на цикле: в каждой вершине есть два числа: $with[u]$ и $out[u]$ требуется для каждой вершины цикла одно из этих чисел,причём для двух соседних вершин нельзя одновременно выбрать $with$. Для решения этой задачи можно использовать динамическое программирование, аналогичное решению задачи для дерева: разрежем одно из рёбер цикла, переберём, какое из двух значений взято для первой вершины, посчитаем значения вдоль получившегося пути, и найдем ответ.

В зависимости от реализации всех описанных методов — за $O(n^2)$ или $O(n)$ — решение получает баллы за подзадачи 4 и 5.

Задача 8. Разбиение на пары

Автор задачи : *Сафонов И.*
Разработчик : *Путилин М.*
Разбор задачи: *Станкевич А.С.*

Рассмотрим искомое разбиение на пары. Для каждого j есть $n/2$ индексов i, таких что $a_{i,j} \leqslant b_j$ и $n/2$ индексов i, таких что $a_{i,j} \geqslant b_j$.

Таким образом, если решение существует, то в качестве b_j можно выбрать величину $\texttt{median}(a_{1,j}, \ldots, a_{n,j})$, где `median` — медиана набора чисел. В наборе чётное количество чисел, поэтому медиана — это среднее арифметическое $(n/2)$-й и $(n/2+1)$-й порядковых статистик значений соответствующего параметра, хотя можно выбрать и просто $(n/2)$-ю порядковую статистику.

Для решения подзадачи 1 можно перебрать все разбиения на пары и для каждого проверить корректность.

В подзадаче 2 параметр всего один, поэтому артефакты можно разбить на пары жадно: первый и последний, второй и предпоследний, и так далее. Решение всегда существует.

В подзадаче 3 все значения каждого параметра различны. Для каждого артефакта построим битовый вектор длины k, у которого j-я координата равна 0 или 1, в зависимости от того, боль-

ше или меньше значение j-го параметра, чем соответствующее медианное значение. Заметим, что артефакты требуется разбить на пары, которым соответствуют битовые вектора, являющиеся отрицанием друг друга. Это можно попытаться сделать жадным алгоритмом, если решение не найдено, то его не существует.

Трудность решения оставшихся подзадач заключается в том, что все параметры, кроме первого, могут принимать равные значения. Каждый артефакт, значение параметра которого равно соответствующей медиане, может быть как меньшим, так и большим по этому параметру в паре.

В подзадаче 4 параметров два. Разделим артефакты по первому параметру на два множества, условно назовём их *красным* и *синим*: первые $n/2$ по первой координате и вторые $n/2$ — по первой координате. В каждую пару должен войти один красный и один синий артефакт. Сопоставим каждому артефакту значение 0, 1 или -1, в зависимости от того, равно, больше или меньше значение его второго параметра медианному. Получаем три красных множества R_0, R_1, R_{-1} и три синих множества B_0, B_1, B_{-1}. Артефакты из R_1 должны войти в пары с B_{-1}, из R_{-1} — в пары с B_1. Артефакты из R_0 могут войти в пары с любыми синими, а из B_0 — с любыми красными. Их можно распределить жадно.

Приведём альтернативное решение. Отсортируем артефакты по второму параметру (при равенстве — произвольным образом), и первые $n/2$ назовём *малыми*, а последние $n/2$ — *большими*. Тогда красные большие входят в пары с синими малыми, а красные малые — с синими большими. Легко показать, что их количества соответственно совпадают.

В подзадачах 5 и 6 параметров больше и жадный алгоритм перестаёт работать. Рассмотрим решение на основе алгоритмов поиска паросочетаний в двудольных графах и потоков.

Решение подзадачи 5. Для каждой пары артефактов проверим, можно ли создать из них пару. Проведём ребро между ними в случае, если это сделать можно. Как и прежде разделим артефакты на красные и синие, в зависимости от первого параметра. Заметим, что ребра соединяют артефакты разного цвета, следовательно, построенный граф — двудольный. Найдём в получившемся графе полное паросочетание алгоритмом Куна. В графе $O(n^2)$ рёбер, поэтому такое решение имеет сложность — $O(n^3)$.

Решение подзадачи 6. Сопоставим каждому артефакту строку длины k из 0, 1 и -1, в которой j-е число равно -1, если $a_{i,j} < b_j$, равно 1, если $a_{i,j} > b_j$, и равно 0, если $a_{i,j} = b_j$. Заметим, что первое число в строке не может быть равно 0. Таким образом, точки разбиваются на $2 \cdot 3^{k-1}$ групп. Создадим вершину для каждой группы. Вершины, которые соответствуют точкам с условием $a_{i,1} < b_1$, составляют первую долю, остальные — вторую. Также установим *исток* и *сток*. В вершины первой доли проведём рёбра из истока с весом, равным числу точек в соответствующей группе. Из вершин второй доли в сток — аналогично. Между вершинами из разных долей проведём рёбра весом $+\infty$, если соответствующие им точки можно объединить в пары. Найдём в таком графе *максимальный* поток. Для получения ответа осталось найти *декомпозицию* этого потока.

Задача 9. Успешные пары

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

Основные темы задачи — логика, разбор случаев. Это простая задача, решение которой доступно большинству участников олимпиады.

Для решения задачи достаточно перебрать все возможные варианты разбиения учеников на пары и подсчитать количество «успешных» пар. Из всех вариантов выбрать оптимальный.

Примерный вариант кода решения:

```
int a, b, c, d, L;
cin >> a >> b >> c >> d >> L;
if ((a + b >= L && c + d >= L) || (a + c >= L && b + d >= L) ||
    (a + d >= L && b + c >= L)) cout << 2;
else if (a + b >= L || c + d >= L || a + c >= L || b + d >= L ||
     a + d >= L || b + c >= L) cout << 1;
else cout << 0;
```

Задача 10. Наибольшее число

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

Основная идея решения задачи состоит в специальной сортировке исходного массива чисел.

Сначала рассмотрим случай $n = 2$.

Первый способ. [Более длинный.] Если два числа a и b начинаются с разных цифр, наибольшее число получится так: первым запишем то число, которое начинается с *большей* цифры, а вторым — число, которое начинается с *меньшей*. Например, если на карточках числа $a = 123$ и $b = 7$, то наибольшим числом, которое можно составить из них будет 7 123.

Если же числа a и b начинаются с одной и той же цифры, то рассуждения нужно применить ко вторым цифрам этих чисел. Если у них одинаковые и вторые цифры, то сравниваем третьи цифры, и так далее. Такое сравнение чисел a и b очень похоже на *лексикографическое*. Например, если на карточках записаны числа $a = 123$ и $b = 1200$, то наибольшим числом, составленным из них, будет 123 1200.

Однако не всё так просто. Если при очередном сравнении у одного из чисел a или b соответствующей цифры нет, то приходится обращать внимание на следующую цифру более длинного числа. Если эта цифра больше самой первой (слева) цифры, то более длинное число нужно ставить перед более коротким, и наоборот — если иначе. Например, если даны числа $a = \mathbf{7}23$ и $b = 723\mathbf{8}$, то первым нужно записать число b, поскольку первые три цифры у них одинаковые, а 4-я цифра более длинного числа b больше самой первой цифры 7. Наибольшим число будет 723**8** **7**23.

Если же эта следующая цифра совпадает с первой цифрой чисел a и b, сравниваем её со второй цифрой и так далее.

Второй способ. [Совсем короткий.] Преобразуем числа a и b в строки, и рассмотрим два способа соединения этих строк: «a» + «b» и «b» + «a». Сделаем обратное преобразование — конвертируем строки в числа. То число, которое окажется больше, определяет способ соединения карточек. Например, для указан-

ного выше примера $a = 723$ и $b = 7238$:

$$«a» + «b» = «723» + «7238» = «723\,7238»,$$
$$«b» + «a» = «7238» + «723» = «7238\,723».$$

Во втором случае получается большее число, он и определяет способ соединения карточек.

Подзадачи 1 и 2. Будем просматривать массив чисел, записанных на карточках. Среди них найдём «наибольшее» число a, для которого выполняется условие: число, определяемое строкой $«a» + «b_i»$, не меньше числа, определяемого строкой $«b_i» + «a»$, для *всех* чисел исходного массива. Затем найдем следующее «по величине» число, и так далее. Примем во внимание, что при конвертации строк $«a» + «b»$ может получиться число порядка 10^{13}, поэтому в `C++` используем функцию `stoll` вместо `stoi`. Сложность алгоритма — $O(n^2)$.

Полное решение. Воспользуемся идеей алгоритма быстрой сортировки. Составим процедуру `Qsort`, единственным отличием которой от стандартной процедуры `qsort`, будет сравнение чисел. Вместо оператора сравнения чисел $a > b$ используем сравнение «по строкам»: если строка $«a» + «b»$ определяет число, большее чем у строки $«b» + «a»$, считаем, что $a \succ b$. В этом случае сложность будет $O(n \log n)$.

Оглавление