**BYNARY SEARCH TREE**

Задача 0.0

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Дано бинарное поисковое дерево, в котором ключи вершин являются целыми числами в диапазоне от −231 до 231 − 1 включительно. Найдите сумму ключей всех вершин дерева.

**Формат входных данных**

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Ключи задаются в формате по одному в строке.

В поисковом дереве все ключи по определению уникальны, поэтому при попытке добавить в дерево ключ, который там уже есть, он игнорируется.

**Формат выходных данных**

Выведите единственное число — сумму ключей в построенном дереве.

Задача 0.1. Построить дерево

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

По набору ключей постройте бинарное поисковое дерево и выполните его прямой левый обход.

**Формат входных данных**

Входной файл содержит последовательность чисел — ключи вершин в порядке добавления в дерево. Ключи задаются в формате по одному в строке.

В поисковом дереве все ключи по определению уникальны, поэтому при попытке добавить в дерево ключ, который там уже есть, он игнорируется.

**Формат выходных данных**

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом дерева.

Задача 0.2. Удалить из дерева

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

По набору ключей постройте бинарное поисковое дерево. Удалите из него ключ (правым удалением), если он есть в дереве. Выполните прямой левый обход полученного дерева.

**Формат входных данных**

В первой строке записано целое число — ключ, который нужно удалить из дерева.

Вторая строка пустая.

Последующие строки содержат последовательность чисел — ключи вершин в порядке добавления в дерево. Ключи задаются в формате по одному в строке. Дерево содержит хотя бы две вершины.

Напомним, что в поисковом дереве все ключи по определению уникальны, поэтому при попытке добавить в дерево ключ, который там уже есть, он игнорируется.

**Формат выходных данных**

Выведите последовательность ключей вершин, полученную прямым левым обходом дерева.

Задача 0.3. Является ли бинарное дерево поисковым?

Имя входного файла: bst.in  
Имя выходного файла: bst.out  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 МБ

Задано бинарное дерево. Необходимо проверить, является ли оно поисковым.

Будем предполагать, что в бинарном поисковом дереве могут быть вершины с одинаковыми ключами. Тогда, для того, чтобы дерево было поисковым, должно выполняться следующее требование: для каждой вершины xxx все ключи в левом поддереве вершины xxx меньше ключа вершины xxx, а все ключи в правом поддереве больше либо равны ключу вершины xxx.

**Формат входных данных**

Первая строка входного файла содержит единственное целое число n (1≤n≤8⋅10^5) – количество вершин в дереве.

Следующая строка содержит одно целое число m (−2^31≤m≤2^31−1)– значение в корневой вершине дерева.

В каждой из последующих n−1 строк через пробелы перечисляются три параметра m, p и c, которые задают какую-либо вершину дерева.

m – целое число (−2^31≤m≤2^31−1), значение, записанное в вершине.

p – целое число (1≤p≤n−1), номер строки входного файла, в которой был задан родитель текущей вершины (нумерация строк с нуля). Гарантируется, что p меньше, чем номер текущей строки.

c может принимать одно из двух значений `L' или `R'. Значение `L' указывает на то, что текущая вершина присоединена к родительской слева, `R' – справа.

Гарантируется, что совокупность всех строк задает корректное бинарное дерево.

**Формат выходных данных**

В единственной строке выведите `YES', если заданное дерево является бинарным деревом поиска, и `NO' в противном случае.

**10**

 Задача 34

Имя входного файла: tst.in  
Имя выходного файла: tst.out  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Найти и удалить (если такая вершина есть, правым удалением) максимальную по значению среди вершин, для которых расстояние до любой другой вершины дерева строго меньше *k* и через которые не проходят полупути длины *k* (длина полупути и расстояние между двумя вершинами измеряются числом дуг).

**Формат входных данных**

В первой строке находится целое число *k* (0 ≤ *k* ≤ 109).

В последующих строках, в каждой по одному числу, находятся ключи вершин исходного дерева (от 0 до 231 − 1) в порядке добавления в дерево.

Число вершин произвольное — от 0 до 1000.

**Формат выходных данных**

Выходной файл должен содержать последовательность ключей вершин, полученную прямым левым обходом итогового дерева.

Если в результате дерево удалилось полностью, то необходимо вывести в выходной файл сообщение Empty.

**DYNAMIC PROGRAMMING**

Задача 0.1. Порядок перемножения матриц

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Дана последовательность из *s* матриц *A*1, *A*2, …, *As*. Требуется определить, в каком порядке их следует перемножать, чтобы число атомарных операций умножения было минимальным. Матрицы предполагаются совместимыми по отношению к матричному умножению (т. е. число столбцов матрицы *Ai*− 1 совпадает с числом строк матрицы *Ai*).

Будем считать, что произведение матриц — операция, которая принимает на вход две матрицы размера *k* × *m* и *m* × *n* и возвращает матрицу размера *k* × *n*, затратив на это *kmn* атомарных операций умножения. (Базовый тип позволяет хранить любой элемент итоговой и любой возможной промежуточной матрицы, поэтому умножение двух элементов требует одной атомарной операции.)

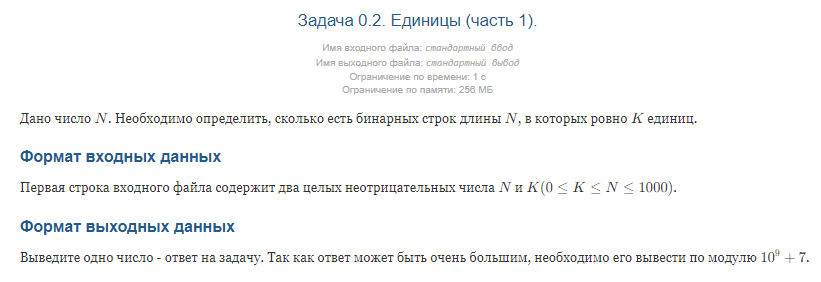
Так как перемножение матриц ассоциативно, итоговая матрица не зависит от порядка выполнения операций умножения. Другими словами, нет разницы, в каком порядке расставляются скобки между множителями, результат будет один и тот же.

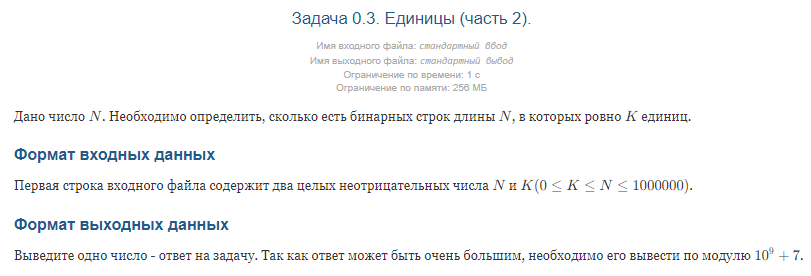
**Формат входных данных**

В первой строке задано число *s* матриц (2 ≤ *s* ≤ 100). В последующих *s* строках заданы размеры матриц: строка *i* + 1 содержит через пробел число *ni* строк и число *mi* столбцов матрицы *Ai* (1 ≤ *ni*, *mi* ≤ 100). Гарантируется, что *mi* совпадает с *ni*+ 1 для всех индексов *i* от 1 до *s* − 1.

**Формат выходных данных**

Выведите минимальное число атомарных операций умножения, необходимое для перемножения *s* матриц.





**7**

 Задача 6. Строго возрастающая без разрывов подпоследовательность

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Необходимо из заданной числовой последовательности *A*, состоящей из *n* элементов, вычеркнуть минимальное число элементов так, чтобы оставшиеся элементы образовали строго возрастающую подпоследовательность элементов. Построенный алгоритм должен иметь трудоёмкость O(*n* log *n*).

***Замечание***

Возрастание *без разрывов* подразумевает, что каждый следующий элемент подпоследовательности строго больше предыдущего.

**Формат входных данных**

Первая строка входного файла содержит число *n* (1 ≤ *n* ≤ 700 000). Следующая строка содержит *n* элементов последовательности *A*, которые разделены пробелами (элементы последовательности — целые числа, не превосходящие по модулю 1 000 000 000).

**Формат выходных данных**

Выведите одно число — длину строго возрастающей подпоследовательности элементов.

**7**

 Задача 20. Палиндром

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Вводится непустая строка *S*, которая имеет длину не более 7000 символов и состоит только из строчных латинских букв. Необходимо удалить из строки минимальное число символов так, чтобы получился палиндром (строка символов, которая читается слева направо и справа налево одинаково).

**Формат входных данных**

В первой строке записана исходная строка *S*.

**Формат выходных данных**

Выведите в первой строке длину получившегося палиндрома, а во второй строке сам палиндром (если палиндромов несколько, то выведите только один из них).

**6**

 Задача 25. Преобразование строк

Имя входного файла: in.txt  
Имя выходного файла: out.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

На вход подаются две символьные последовательности *A* и *B*, каждая последовательность состоит из маленьких латинских букв и имеет длину не более 1000 символов. Необходимо преобразовать последовательность *A* в последовательность *B* с минимальным суммарным штрафом, который определяется следующим образом:

* удаление символа из строки *A* равно *x* баллов;
* вставка символа в строку *A* равна *y* баллов;
* замена символа в строке *A* на любой другой равна *z* баллов.

## Формат входных данных

В первой строке находится число *x*. Во второй строке — число *y*. В третьей строке — число *z*. В следующих двух строках находятся символьные последовательности *A* и *B* (тип элементов последовательности string).

## Формат выходных данных

Выведите минимальный суммарный штраф.

**6**

 Задача 69.2. Путь лягушки

Имя входного файла: *стандартный ввод*  
Имя выходного файла: *стандартный вывод*  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

В одном очень длинном и узком пруду по кувшинкам прыгает лягушка. Кувшинки в пруду расположены в один ряд. Лягушка начинает прыгать с первой кувшинки ряда и хочет закончить на последней. Но в силу вредности характера лягушка согласна прыгать только вперед через одну или через две кувшинки. Например, с кувшинки номер 1 она может прыгнуть лишь на кувшинки номер 3 и номер 4.

На некоторых кувшинках сидят комарики. А именно, на *i*-й кувшинке сидят *ai* комаров. Когда лягушка приземляется на кувшинку, она съедает всех комариков, сидящих на ней. Лягушка хочет спланировать свой маршрут так, чтобы съесть как можно больше комаров. Помогите ей: скажите, какие кувшинки она должна посетить на своем пути.

**Формат входных данных**

Первая строка входа содержит *n* — число кувшинок в пруду (1 ≤ *n* ≤ 100 000). Вторая строка содержит *n* чисел, разделенных одиночными пробелами. *i*-е число сообщает, сколько комаров сидит на *i*-й кувшинке (1 ≤ *i* ≤ *n*). Все числа целые, неотрицательные и не превосходят 1000.

**Формат выходных данных**

В первой строке выведите одно число — максимальное число комаров, которые может съесть лягушка. Во второй строке выведите последовательность чисел — номера тех кувшинок, на которых должна побывать лягушка, в возрастающем порядке. Если решений несколько, выведите любое.

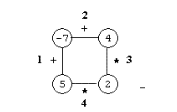
Если лягушка не может добраться до последней кувшинки, то выведите одно число −1.

**9**

 Задача 15. Полигон

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: от 1 с до 2 с  
Ограничение по памяти: нет

Существует игра для одного игрока, которая начинается с задания Полигона с *n* вершинами. Пример графического представления Полигона показан на рис. 1, где *n* = 4. Для каждой вершины Полигона задаётся значение — целое число, а для каждого ребра — метка операции + (сложение) либо × (умножение). Ребра Полигона пронумерованы от 1 до *n*.

  
Рис. 1. Графическое представление Полигона

Первым ходом в игре удаляется одно из рёбер. Каждый последующий ход состоит из следующих шагов:

* выбирается ребро *e* и две вершины *v*1 и *v*2, которые соединены ребром *e*;
* ребро *e* стягивается, то есть вершины *v*1 и *v*2 удаляются и заменяются новой вершиной *v* со значением, равным результату выполнения операции, определённой меткой ребра *e*, над значениями вершин *v*1 и *v*2. В рёбрах, отличных от *e* и инцидентных вершинам *v*1 и *v*2, эти две вершины заменяются на вершину *v*.

Игра заканчивается, когда больше нет ни одного ребра. Результат игры — это число, равное значению оставшейся вершины.

## Пример игры

## 

Напишите программу, которая по заданному Полигону, вычисляет максимальное значение оставшейся вершины и выводит список всех тех рёбер, удаление которых на первом ходу игры позволяет получить это значение.

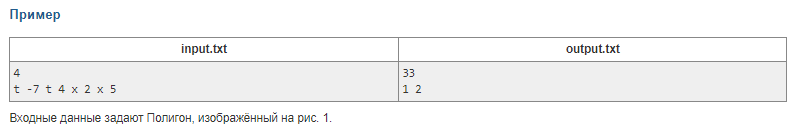
## Формат входных данных

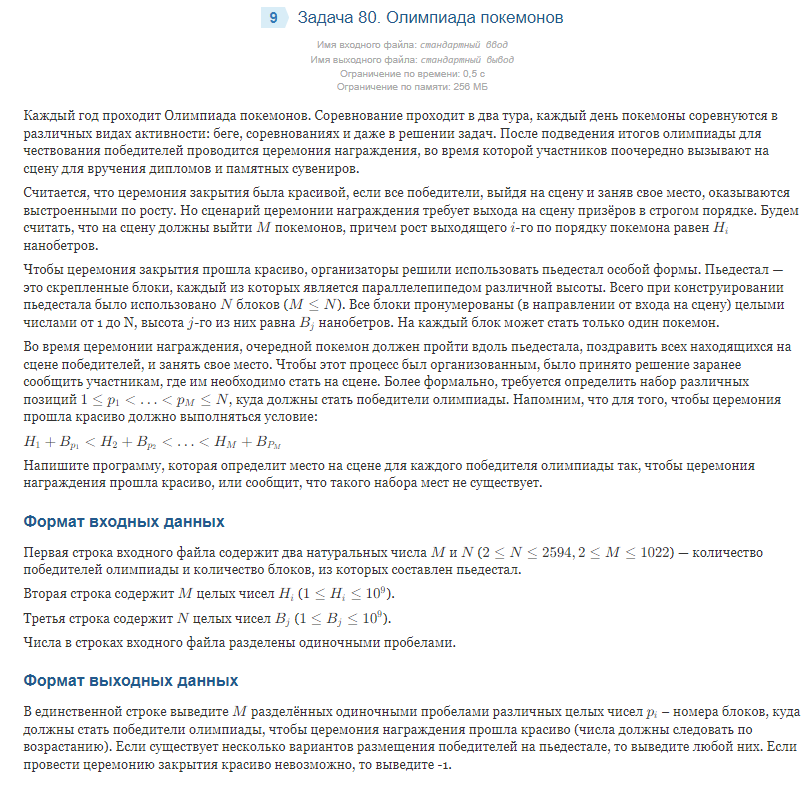
В первой строке записано число *n* (3 ≤ *n* ≤ 500). Вторая строка содержит метки рёбер с номерами 1, …, *n*, между которыми записаны через пробел значения вершин (первое из них соответствует вершине, инцидентной рёбрам 1 и 2, следующее — инцидентной рёбрам 2 и 3, и так далее, последнее — инцидентной рёбрам *n* и 1). Метка ребра — это буква t, соответствующая операции +, или буква x, соответствующая операции ×.

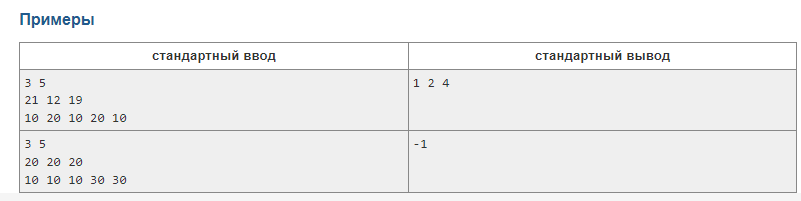
Гарантируется, что при любой последовательности ходов значения вершин находятся в пределах от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

## Формат выходных данных

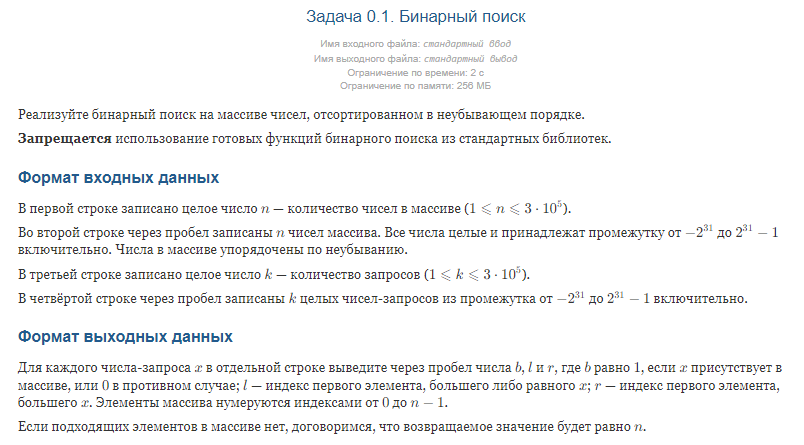
В первой строке выведите максимальное значение оставшейся вершины, которое может быть достигнуто для заданного Полигона. Во второй строке — список всех тех рёбер, удаление которых на первом ходу, позволяет получить это значение. Номера рёбер должны быть записаны в возрастающем порядке и отделяться друг от друга одним пробелом.

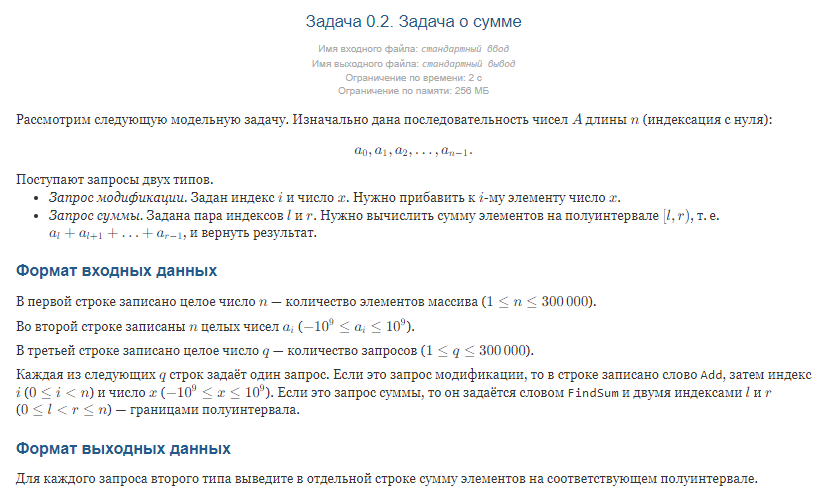
****

****

****

**DATA STRUCTURES**

****

****

Задача 0.3. Бинарная куча?

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Напомним, что для компактного хранения в памяти компьютера полного бинарного дерева на *n* вершинах можно использовать одномерный массив длины *n*. Ключи вершин являются значениями элементов массива, индексация начинается с единицы, при этом у элемента с индексом *i* сыновьями являются элементы с индексами 2*i* и 2*i* + 1 (если такие элементы существуют).

Бинарная куча — полное бинарное дерево, для которого выполняется основное свойство структуры данных «куча»: элементы организованы таким образом, что приоритет любой вершины не ниже приоритета каждого из её сыновей.

Будем предполагать, что среди элементов тот элемент имеет наибольший приоритет, у которого ключевое значение меньше (*Min-Heap*).

Задан массив из *n* чисел. Необходимо проверить, может ли он представлять бинарную кучу.

**Запрещается использовать специальные методы проверки из стандартных библиотек** (std::is\_heap в C++ и пр.).

**Формат входных данных**

В первой строке записано число *n* элементов в массиве (1 ≤ *n* ≤ 100 000). Во второй строке через пробел записаны *n* целых чисел, не превосходящих по модулю 109, — элементы массива.

**Формат выходных данных**

Выведите Yes, если существует бинарная куча, которая в памяти представляется заданным массивом, или No иначе.

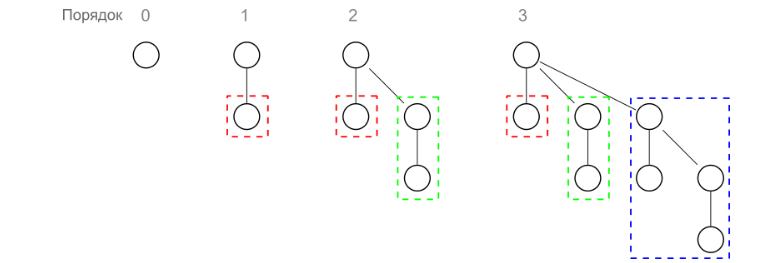
Задача 0.4. Биномиальная куча

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 МБ

*Биномиальное дерево* высоты *k* обозначается через *Bk* и определяется следующим образом.

* Биномиальное дерево *B*0 состоит из одной единственной вершины.
* Биномиальное дерево *Bk* образуется присоединением биномиального дерева высоты *k* − 1 к корню другого биномиального дерева высоты *k* − 1 (во время присоединения корень одного из деревьев полагается одним из сыновей корня другого дерева).

Таким образом, биномиальное дерево *Bk* имеет корневую вершину, сыновьями которой являются корни биномиальных деревьев *B*0, *B*1, *B*2, …, *Bk*− 1.



*Биномиальная куча* состоит из набора биномиальных деревьев попарно различной высоты. Другими словами, для любого *k* ≥ 0 биномиальное дерево *Bk* либо не входит в состав кучи, либо входит ровно один раз.

Дано общее число *n* вершин в биномиальной куче. Определите, из каких биномиальных деревьев может состоять эта куча.

## Формат входных данных

На входе задано одно число *n* вершин в куче (1 ≤ *n* ≤ 1018).

## Формат выходных данных

Для каждого биномиального дерева *Bk* в составе кучи выведите в отдельной строке одно целое число — высоту *k*. Деревья следует вывести по возрастанию высоты. Все деревья должны иметь различные порядки, а сумма порядков должна быть *n*.

Если решений несколько, выведите любое. Если решений нет и биномиальную кучу с заданным числом вершин построить нельзя, выведите −1.

Задача 0.5. Хеш-таблица

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Хеш-таблица состоит из *m* ячеек (ячейки нумеруются целыми числами от 0 до *m* − 1). Для разрешения коллизий используется метод открытой адресации. Функция

*h*(*x*, *i*) = ((*x* mod *m*) + *c* ⋅ *i*) mod *m*

задает линейную последовательность проб свободных ячеек, где *x* — ключ, *i* — номер попытки (попытки нумеруются с нуля), *c* — константа.

В таблицу было последовательно добавлено *n* ключей. В случае, если ключ в таблице уже есть, повторного добавления не происходит.

Определите, какой ключ хранится в каждой ячейке таблицы по окончании выполнения всех операций.

**Формат входных данных**

В первой строке через пробел записаны три целых числа: размер *m* таблицы (2 ≤ *m* ≤ 10 000), константа *c* (1 ≤ *c* ≤ *m* − 1) и число *n* ключей для добавления (0 ≤ *n* ≤ 10 000). Число *c* взаимно просто с числом *m*. Далее в последующих *n* строках записаны ключи, которые добавляются в хеш-таблицу, — целые числа из промежутка от 0 до 109. Ключи могут повторяться, но гарантируется, что в таблице достаточно места для размещения всех уникальных ключей из *n* заданных.

**Формат выходных данных**

Выведите ровно *m* чисел, *i*-е число описывает *i*-ю ячейку (0 ≤ *i* ≤ *m* − 1). Если ячейка пуста, выведите −1, иначе выведите ключ, который в ней содержится.

Задача 0.6. Дерево отрезков

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Имеется последовательность *s*0, …, *sn*− 1, состоящая из нулей. На этой последовательности могут выполняться запросы следующих типов:

1. Установить значение *si* равным *v*.
2. Прибавить к каждому элементу с индексом из отрезка [*a*, *b*] число *v*.
3. Найти сумму элементов с индексами из отрезка [*a*, *b*].
4. Найти минимум среди элементов с индексами из отрезка [*a*, *b*].
5. Найти максимум среди элементов с индексами из отрезка [*a*, *b*].

Требуется написать программу, которая будет обрабатывать указанные запросы.

## Формат входных данных

Первая строка содержит одно целое число *n* (1 ≤ *n* ≤ 300 000) — длину последовательности. Последующие строки имеют следующий формат: первое число — тип запроса (см. условие), далее для запроса типа 1 два числа: *i*, индекс элемента в массиве, и значение *v*; для запроса типа 2 три числа: *a*, *b*, *v*; для запросов типа 3–5 два числа: *a* и *b*. Вход заканчивается строкой с единственным числом 0. Число строк не превосходит 300 003. Абсолютная величина числа *v* в каждом запросе не превышает 1 000 000. Числа *a*, *b* и *i* в каждом запросе удовлетворяют неравенствам 0 ≤ *a* ≤ *b* < *n*, 0 ≤ *i* < *n*.

## Формат выходных данных

Для каждого запроса типа 3–5 выведите ответ на запрос.

**8**

 Задача 43.1. Кодирование Хаффмана

Имя входного файла: huffman.in  
Имя выходного файла: huffman.out  
Ограничение по времени: 1 с  
Ограничение по памяти: 128 МБ

Кодирование Хаффмана (D. A. Huffman) относится к префиксному кодированию, позволяющему минимизировать длину текста за счёт того, что различные символы кодируются различным числом битов.

Напомним процесс построения кода. Вначале строится дерево кода Хаффмана. Пусть исходный алфавит состоит из *n* символов, *i*-й из которых встречается *pi* раз во входном тексте. Изначально все символы считаются активными вершинами будущего дерева, *i*-я вершина помечена значением *pi*. На каждом шаге мы берём две активных вершины с наименьшими метками, создаём новую вершину, помечая её суммой меток этих вершин, и делаем её их родителем. Новая вершина становится активной, а двое её сыновей из списка активных вершин удаляются. Процесс многократно повторяется, пока не останется только одна активная вершина, которая полагается корнем дерева.

Заметим, что символы алфавита представлены листьями этого дерева. Для каждого листа (символа) длина его кода Хаффмана равна длине пути от корня дерева до него. Сам код строится следующим образом: для каждой внутренней вершины дерева рассмотрим две дуги, идущие от неё к сыновьям. Одной из дуг присвоим метку 0, другой — 1. Код каждого символа — последовательность из нулей и единиц на пути от корня к листу.

Задача состоит в том, чтобы вычислить длину текста после его кодирования методом Хаффмана. Сам текст не дан, известно лишь, сколько раз каждый символ встречается в тексте. Этого достаточно для решения задачи, поскольку длина кода зависит только от частоты появления символов.

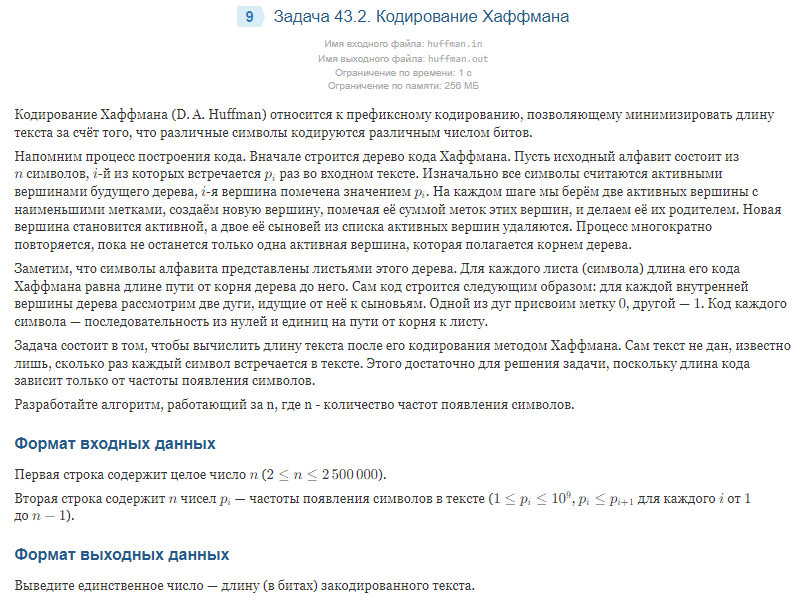
**Формат входных данных**

Первая строка содержит целое число *n* (2 ≤ *n* ≤ 500 000).

Вторая строка содержит *n* чисел *pi* — частоты появления символов в тексте (1 ≤ *pi* ≤ 109, *pi* ≤ *pi*+ 1 для каждого *i* от 1 до *n* − 1).

**Формат выходных данных**

Выведите единственное число — длину (в битах) закодированного текста.

****

**9**

 Задача 40. Ленивый программист

Имя входного файла: lazy.in  
Имя выходного файла: lazy.out  
Ограничение по времени: 1 с  
Ограничение по памяти: 32 МБ

Недавно образовавшаяся студия веб-дизайна SMART (Simply Masters of ART) имеет в своём активе пока только двух работников: дизайнера, по совместительству занимающего должность директора студии, и программиста. Благодаря расторопности директора студия сразу после своего образования получила *n* заказов на разработку веб-сайтов. Для *i*-го заказа известен крайний срок *di*, до которого студия должна выполнить его (измеряемый в некоторых условных единицах времени с начала образования студии).

Известно, что программист очень ленив и не торопится выполнять заказы. На выполнение *i*-го заказа ему необходимо *bi* единиц времени. К счастью, программист согласен ускорить темп своей работы за дополнительную плату. В частности, если директор заплатит ему *xi* долларов, то он выполнит *i*-й заказ за *bi* − *ai* ⋅ *xi* единиц времени (время выполнения остальных заказов не изменится — за каждый заказ нужно доплачивать отдельно).

Как видите, программист настолько любит деньги, что способен выполнить заказ даже мгновенно, если ему заплатят достаточную сумму. Но выполнять заказы за отрицательное время он пока ещё не научился, поэтому не имеет смысла доплачивать за выполнение *i*-го заказа более чем *bi* / *ai* долларов.

Теперь перед директором студии стоит непростая задача: организовать рабочее время и поощрение программиста так, чтобы он выполнил все заказы вовремя, и при этом доплатить программисту минимальную сумму денег.

**Формат входных данных**

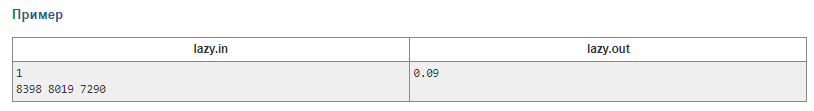
Первая строка содержит целое число *n* заказов (1 ≤ *n* ≤ 100 000).

Следующие *n* строк описывают заказы: *i*-я строка описывает *i*-й заказ и содержит целые числа *ai*, *bi*, *di* (1 ≤ *ai*, *bi* ≤ 10 000; 1 ≤ *di* ≤ 109).

**Формат выходных данных**

В единственной строке выведите минимальную сумму денег, которую нужно доплатить программисту, чтобы он, при правильной организации своего рабочего времени, выполнил все заказы вовремя.

Абсолютная погрешность ответа не должна превосходить 10−2.

****

**9**

 Задача 41. Тройное доминирование

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 МБ

В командной олимпиаде ФПМИ по программированию участвуют *n* команд. Каждая команда состоит из трёх участников. Так как олимпиада является командной, то, даже располагая сведениями о силах каждого из участников команды, довольно сложно делать какие-либо выводы о силе команды. Поэтому затруднительно делать какие-нибудь прогнозы относительно того, кто станет победителем олимпиады. Однако буквально за несколько дней до командной олимпиады ФПМИ прошла личная олимпиада ФПМИ, и её результаты уже известны. Таким образом, для каждого участника командной олимпиады известно место, которое он занял в личной. Имея эту информацию, можно попытаться сузить круг кандидатов в победители командной олимпиады.

Пронумеруем команды от 1 до *n* и обозначим через *xi*, *yi* и *zi* места, которые заняли участники *i*-й команды в личной олимпиаде. Переставлять участников в команде нельзя, т. е. (*xi*, *yi*, *zi*) — это не множество, а упорядоченная тройка.

Будем говорить, что команда *i* *доминирует над командой j*, если выполняются неравенства *xi* < *xj*, *yi* < *yj* и *zi* < *zj*. Понятно, что если команда *i* доминирует над командой *j*, то у команды *j* вряд ли есть шанс стать победителем олимпиады. Поэтому имеет смысл назвать команду *i* кандидатом в победители, если никакая другая команда не доминирует над ней.

Напишите программу, которая по результатам личной олимпиады определяет число команд – кандидатов в победители.

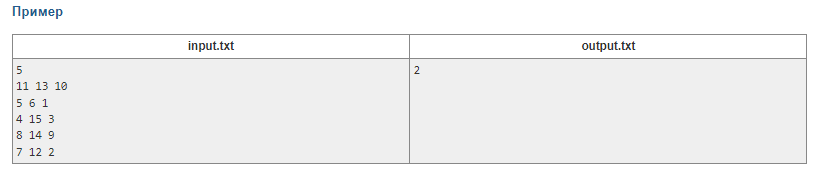
**Формат входных данных**

Первая строка содержит число *n* команд (1 ≤ *n* ≤ 100 000).

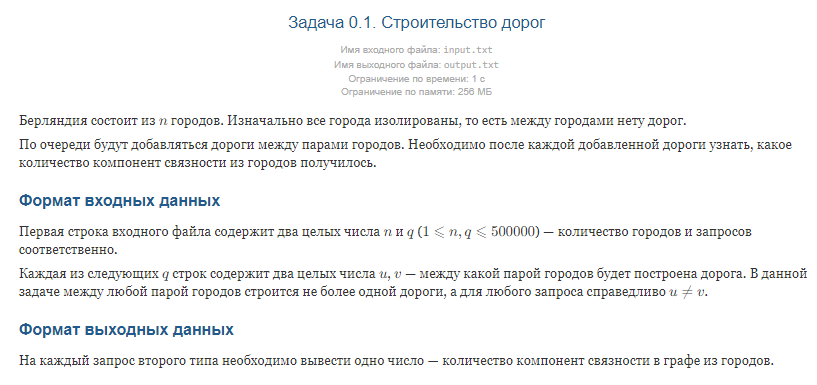
Следующие *n* строк описывают места, которые заняли в личной олимпиаде участники каждой из команд. Каждая из этих строк содержит три числа *xi*, *yi* и *zi* (1 ≤ *xi*, *yi*, *zi* ≤ 3*n*). Никакие два участника личной олимпиады не разделили место между собой.

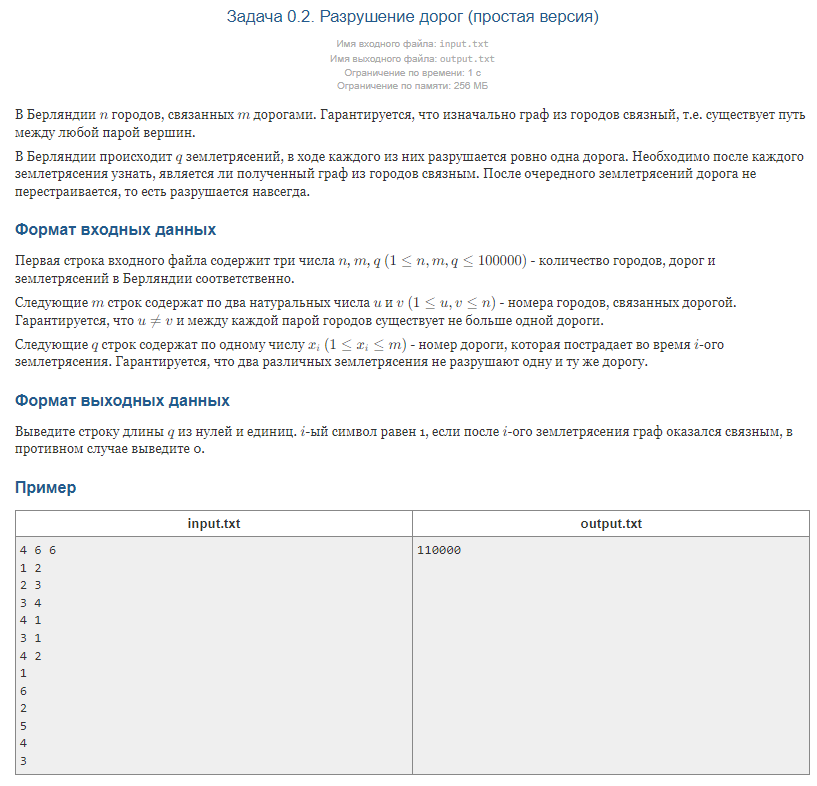
**Формат выходных данных**

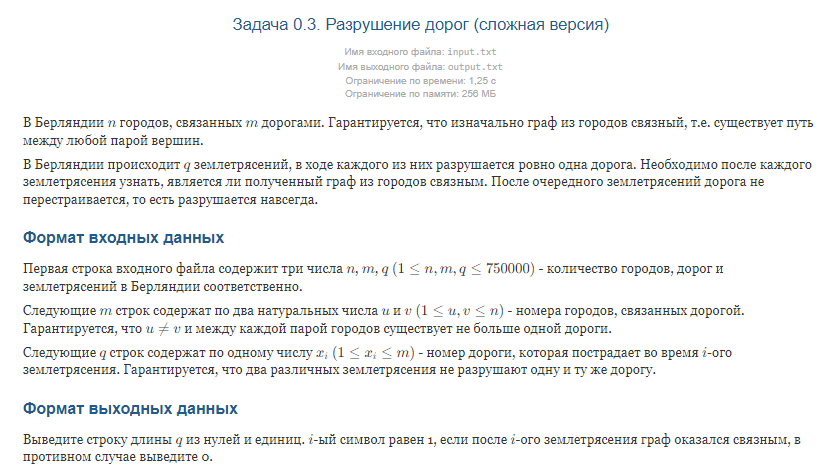
Выведите искомое число команд-кандидатов.

****

**GRAPHS**

****

****

****

Задача 0.4. Построить матрицу смежности

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 МБ

Неориентированный граф задан списком рёбер. Постройте матрицу смежности этого графа. Гарантируется, что в графе нет петель и кратных рёбер.

**Формат входных данных**

В первой строке записаны два целых числа *n* и *m* — число вершин (1 ≤ *n* ≤ 100) и число рёбер графа (0 ≤ *m* ≤ *n*(*n* − 1) / 2). Вершины графа пронумерованы числами от 1 до *n*. В следующих *m* строках описаны рёбра: в каждой строке заданы через пробел две вершины *u* и *v* (1 ≤ *u*, *v* ≤ *n*, причём *u* и *v* различны), которые соединены ребром.

**Формат выходных данных**

Выведите матрицу смежности графа: *n* строк по *n* чисел, каждое число — 0 или 1, числа в строках разделены пробелами.

Задача 0.5. Канонический вид (по списку дуг)

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Задан ориентированный граф, который является корневым деревом. Вершины графа пронумерованы целыми числами от 1 до *n*. Дуги направлены от корня к листьям. Выведите канонический способ задания этого дерева.

*Каноническим способом* представления корневого дерева на *n* вершинах называется массив *P* размера *n*, где *pi* — отец вершины *i*, а если вершина *i* является корнем дерева, то *pi* полагается равным нулю.

**Формат входных данных**

В первой строке записано целое число *n* (1 ≤ *n* ≤ 100 000). В последующих *n* − 1 строках заданы дуги графа. Дуга, идущая из вершины *u* в вершину *v*, задаётся парой чисел *u* и *v*, записанных через пробел (1 ≤ *u*, *v* ≤ *n*). Гарантируется, что граф представляет собой корневое дерево.

**Формат выходных данных**

Выведите *n* чисел — элементы массива *P*.

Задача 0.6. Построить список смежности

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 128 МБ

Неориентированный граф задан списком рёбер. Постройте список смежности этого графа. Гарантируется, что в графе нет петель и кратных рёбер.

**Формат входных данных**

В первой строке записаны два целых числа *n* и *m* — число вершин (1 ≤ *n* ≤ 100 000) и число рёбер графа (0 ≤ *m* ≤ 200 000). Вершины графа пронумерованы числами от 1 до *n*. В следующих *m* строках описаны рёбра: в каждой строке заданы через пробел две вершины *u* и *v* (1 ≤ *u*, *v* ≤ *n*, причём *u* ≠ *v*), которые соединены ребром.

**Формат выходных данных**

Выведите список смежности графа в виде *n* строк. В *i*-й строке выведите число вершин, смежных с *i*-й вершиной, а затем номера этих вершин через пробел в произвольном порядке.

Задача 0.7. Канонический вид (по матрице смежности)

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Задан ориентированный граф, который является корневым деревом. Вершины графа пронумерованы целыми числами от 1 до *n*. Дуги направлены от корня к листьям. Выведите канонический способ задания этого дерева.

*Каноническим способом* представления корневого дерева на *n* вершинах называется массив *P* размера *n*, где *pi* — отец вершины *i*, а если вершина *i* является корнем дерева, то *pi* полагается равным нулю.

**Формат входных данных**

В первой строке записано целое число *n* (1 ≤ *n* ≤ 100). В последующих *n* строках задана матрица смежности графа. Если в графе есть дуга, ведущая из вершины *i* в вершину *j*, то в *i*-й строке матрицы *j*-е число равно 1, иначе 0. Числа разделяются одиночными пробелами. Гарантируется, что граф на входе представляет собой корневое дерево.

**Формат выходных данных**

Выведите *n* чисел — элементы массива *P*.

Задача 0.8. Поиск в ширину

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Для ориентированного графа, заданного матрицей смежности, выполнили поиск в ширину. При неоднозначности выбора очередной вершины всегда выбирали вершину с меньшим номером. В процессе поиска в ширину в момент посещения вершины ей присваивали метку (нумерация меток начинается с единицы). Определите, какую метку получила каждая из вершин графа (поиск в ширину выполняли до тех пор, пока все вершины не получили метки).

**Формат входных данных**

В первой строке записано целое число *n* вершин орграфа (1 ≤ *n* ≤ 100). Вершины нумеруются числами от 1 до *n*. Далее записана матрица смежности графа: *n* строк по *n* чисел, каждое из которых — 0 или 1. Числа разделяются одиночными пробелами. Гарантируется, что все элементы на главной диагонали нулевые.

**Формат выходных данных**

Выведите *n* чисел, *i*-е из которых — метка, которую получила вершина *i*.

Задача 0.9. Поиск в глубину

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Для ориентированного графа, заданного матрицей смежности, выполнили поиск в глубину. При неоднозначности выбора очередной вершины всегда выбирали вершину с меньшим номером. В процессе поиска в глубину в момент посещения вершины ей присваивали метку (нумерация меток начинается с единицы). Определите, какую метку получила каждая из вершин графа.

**Формат входных данных**

В первой строке записано целое число *n* вершин орграфа (1 ≤ *n* ≤ 100). Вершины нумеруются числами от 1 до *n*. Далее записана матрица смежности графа: *n* строк по *n* чисел, каждое из которых — 0 или 1. Числа разделяются одиночными пробелами. Гарантируется, что все элементы на главной диагонали нулевые.

**Формат выходных данных**

Выведите *n* чисел, *i*-е из которых — метка, которую получила вершина *i*.

**4**

 Задача 0.10. Алгоритм Дейкстры

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: 64 МБ

Задан неориентированный взвешенный псевдограф, вершины которого пронумерованы числами от 1 до *n*. Требуется найти кратчайшую цепь между вершинами 1 и *n*. Гарантируется, что (1, *n*)-цепь существует.

**Формат входных данных**

В первой строке содержатся целые числа *n* и *m*, где *n* — число вершин, а *m* — число рёбер в псевдографе (1 ≤ *n* ≤ 200 000, 0 ≤ *m* ≤ 300 000).

Далее в *m* строках содержатся рёбра, по одному в строке. Каждое ребро задаётся тремя целыми числами *ui*, *vi*, *wi*, где *ui*, *vi* — концы ребра, а *wi* — его длина (1 ≤ *ui*, *vi* ≤ *n*, 0 ≤ *wi* ≤ 11 111).

**Формат выходных данных**

Выведите длину кратчайшей цепи.

Задача 0.11. Максимальный поток (простая версия)

Имя входного файла: *стандартный ввод*  
Имя выходного файла: *стандартный вывод*  
Ограничение по времени: 1 с  
Ограничение по памяти: 256 МБ

Задан ориентированный граф *G*, содержащий *n* вершин и *m* дуг, в котором каждой дуге (*u*, *v*) приписана пропускная способность *cuv*.

Требуется найти в этой сети поток максимальной величины из источника (вершины 1) в сток (вершину *n*).

**Формат входных данных**

Первая строка содержит два числа *n* и *m* — число вершин и дуг в графе соответственно. Гарантируется, что 1 ≤ *n*, *m* ≤ 200.

Каждая из следующих *m* строк содержит три числа *ui*, *vi*, *wi* — две вершины, которые соединены дугой и пропускная способность этой дуги. Гарантируется, что *ui* ≠ *vi*, 1 ≤ *ui*, *vi* ≤ *n*, 1 ≤ *wi* ≤ 1000000 для каждой дуги.

**Формат выходных данных**

Выведите одно число — максимально возможную величину потока в заданном графе.

Задача 0.12. Максимальный поток (сложная версия)

Имя входного файла: *стандартный ввод*  
Имя выходного файла: *стандартный вывод*  
Ограничение по времени: от 1 с до 10 с  
Ограничение по памяти: 256 МБ

Задан ориентированный граф *G*, содержащий *n* вершин и *m* дуг, в котором каждой дуге (*u*, *v*) приписана пропускная способность *cuv*.

Требуется найти в этой сети поток максимальной величины из источника (вершины 1) в сток (вершину *n*).

**Формат входных данных**

Первая строка содержит два числа *n* и *m* — число вершин и дуг в графе соответственно. Гарантируется, что 1 ≤ *n* ≤ 200000, 0 ≤ *m* ≤ 200000.

Каждая из следующих *m* строк содержит три числа *ui*, *vi*, *wi* — две вершины, которые соединены дугой и пропускная способность этой дуги. Гарантируется, что *ui* ≠ *vi*, 1 ≤ *ui*, *vi* ≤ *n*, 1 ≤ *wi* ≤ 1000000 для каждой дуги.

**Формат выходных данных**

Выведите одно число — максимально возможную величину потока в заданном графе.

**9**

 Задача 19. Пирамида Хеопса

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Внутри пирамиды Хеопса имеется *N* комнат, в которых установлены 2*M* модулей, составляющих *M* устройств. Каждое устройство включает два модуля, располагающихся в разных комнатах, и предназначено для перемещения между парой комнат, в которых установлены эти модули. Перемещение происходит за 0,5 условных единиц времени. В начальный момент времени модули всех устройств переходят в подготовительный режим. Каждый из модулей имеет некоторый свой целочисленный период времени, в течение которого он находится в подготовительном режиме. По истечении этого времени модуль мгновенно включается, после чего опять переходит в подготовительный режим. Устройством можно воспользоваться только в тот момент, когда одновременно включаются оба его модуля. Индиана Джонс сумел проникнуть в гробницу фараона. Обследовав её, он включил устройства и собрался уходить, но в этот момент проснулся хранитель гробницы. Теперь Джонсу необходимо как можно быстрее попасть в комнату с номером *N*, в которой находится выход из пирамиды. При этом из комнаты в комнату он может попадать только при помощи устройств, так как проснувшийся хранитель закрыл все двери в комнатах пирамиды.

Необходимо написать программу, которая получает на входе описание расположения устройств, их характеристик и выдаёт значение оптимального времени и последовательность устройств, которыми надо воспользоваться, чтобы попасть из комнаты 1 в комнату *N* за это время.

## Формат входных данных

В первой строке содержится число *N* комнат (2 ≤ *N* ≤ 10 000). Во второй строке — число *M* устройств (0 ≤ *M* ≤ 100 000). В следующих *M* строках находится информация об устройствах. Каждая строка содержит числа *R*1,*i*, *T*1,*i*, *R*2,*i*, *T*2,*i* — информацию об устройстве *i*:

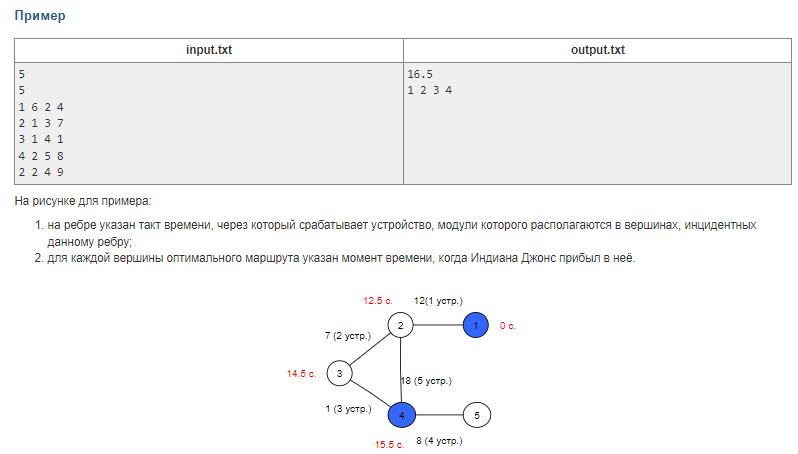
* *R*1,*i* и *R*2,*i* — номера комнат, в которых располагаются модули устройства *i*;
* *T*1,*i*, *T*2,*i* — периоды времени, через которые включаются соответствующие модули (*R*1,*i* — в момент времени *T*1,*i* и *R*2,*i* — в момент времени *T*2,*i*) (1 ≤ *T*1,*i*, *T*2,*i* ≤ 100 000, целые числа).

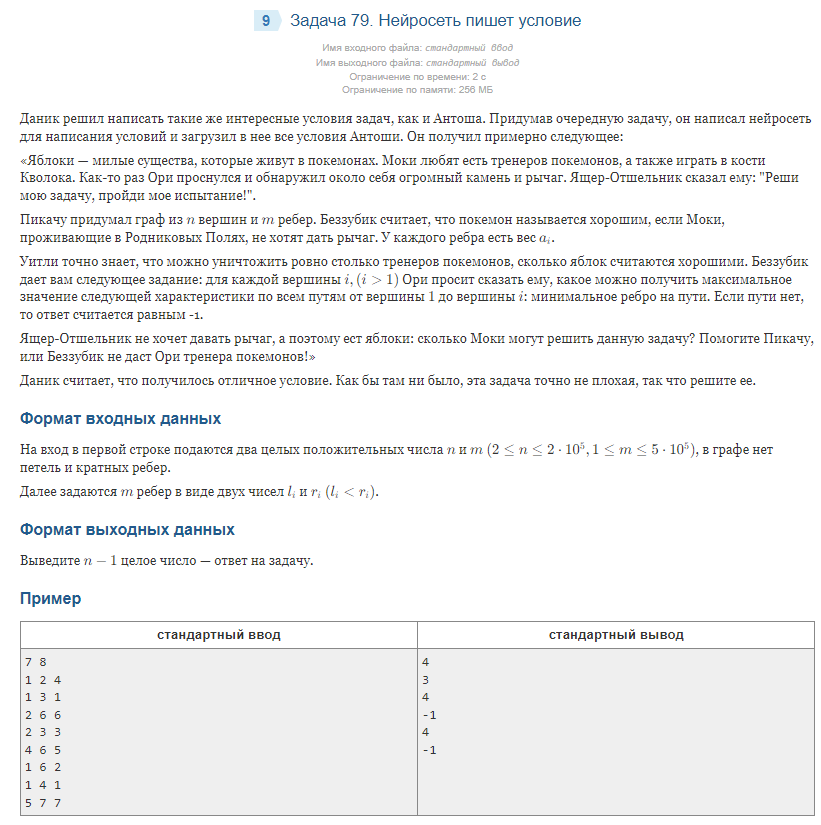
Каждым устройством можно пользоваться из обеих комнат в оба направления.

## Формат выходных данных

Первая строка выходного файла содержит оптимальное время (с абсолютной погрешностью не более 0,1).

Во второй строке указана последовательность устройств, которыми надо воспользоваться, чтобы попасть из комнаты 1 в комнату *N* за минимальное время. Если решений несколько, выведите любое. Гарантируется, что решение существует.

****

****

**TEST 1**

**5**

 Задача 53. Проход в матрице из (1, 1) в (n, m)

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Задана матрица *A* натуральных чисел размера *n* × *m*. За каждый проход через элемент (*i*, *j*) взимается штраф *ai*,*j*. Необходимо минимизировать штраф и пройти из элемента (1, 1) в (*n*, *m*), при этом из текущего элемента можно перейти в любой из трёх соседних, стоящих в строке с номером, на 1 большим (т.е. из элемента (*i*, *j*) можно перейти в один из элементов (*i*, *j* − 1), (*i*, *j*), (*i*, *j* + 1), если таковые существуют).

**Формат входных данных**

Первая строка содержит два числа: *n* и *m*, где *n* — число строк матрицы *A*, а *m* — число столбцов матрицы *A*.

Следующие *n* строк задают саму матрицу *A*: каждая строка содержит по *m* целых чисел, разделённых одиночными пробелами.

Ограничения: 2 ≤ *n* ≤ 200, 1 ≤ *m* ≤ 1 000, 0 ≤ *ai*,*j* ≤ 1 000 000.

**Формат выходных данных**

Выведите одно целое число — минимальную стоимость пути или −1 (минус единицу), если пути нет.

**6**

 Задача 56. Фишка

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Фишка может двигаться по полю длины *n* только вперёд. Длина хода фишки не более *k*. Необходимо найти число различных маршрутов, по которым фишка может пройти поле от позиции 1 до позиции *n*.

***Замечание***

Разработанный алгоритм должен иметь трудоемкость O(*n*).

**Формат входных данных**

Первая строка содержит число *t* наборов входных данных (1 ≤ *t* ≤ 100). Следующие *t* строк содержат по два числа: *n* и *k*, где *n* — длина текущего поля, а *k* — максимальная длина хода фишки (1 ≤ *k* < *n* ≤ 100 000).

**Формат выходных данных**

Необходимо для каждого набора входных данных вывести остаток от деления числа различных маршрутов на 1 000 000 007.

**9**

 Задача 45. Балансировка камней

Имя входного файла: input.txt  
Имя выходного файла: output.txt  
Ограничение по времени: 1 с  
Ограничение по памяти: нет

Вам даны рычажные весы. На левой чаше весов лежит один камень с весом *w* < 2*n*. У вас есть множество камней с весами 20, 21, 22, …, 2*n*−1. Определите, сколькими способами можно расположить некоторые из этих камней на чашах весов так, чтобы уравновесить их. Выведите это значение по модулю *d*.

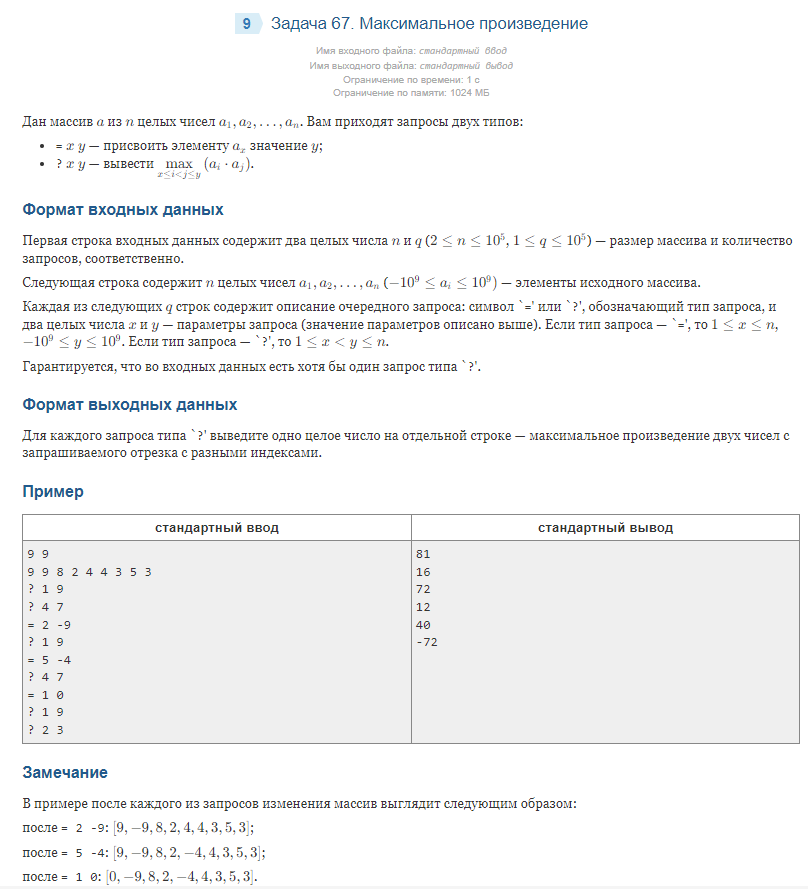
**Формат входных данных**

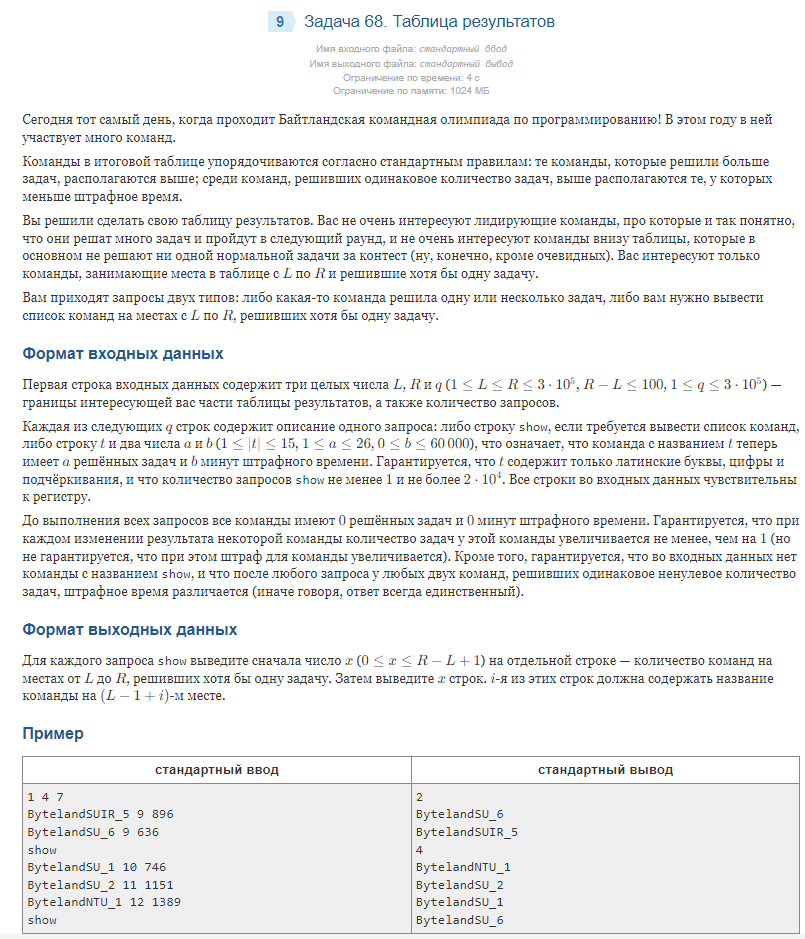
В первой строке записаны три числа: *n*, *k* и *d* (1 ≤ *k* ≤ *n* ≤ 5 000 000, 1 ≤ *d* ≤ 1000). Далее во второй строке ровно *k* символов — двоичная запись числа *w*.

**Формат выходных данных**

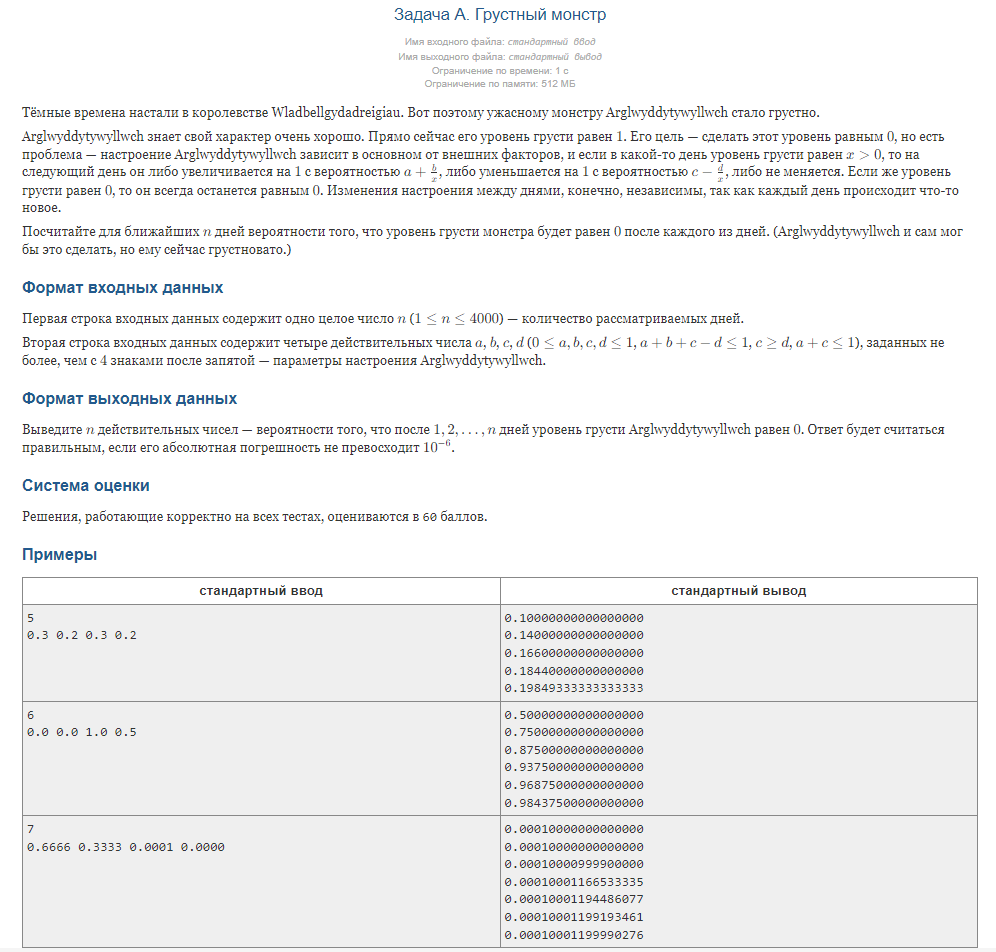
Выведите искомое число способов, взятое по модулю *d*.

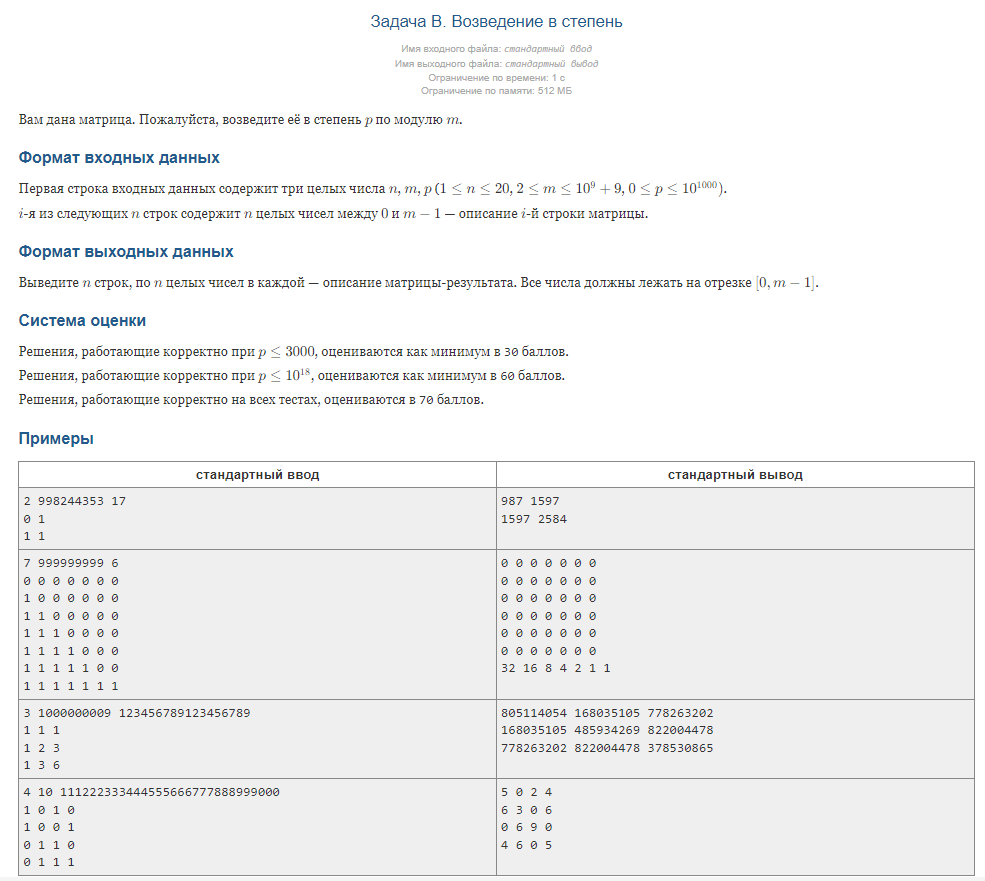
**TEST 2**

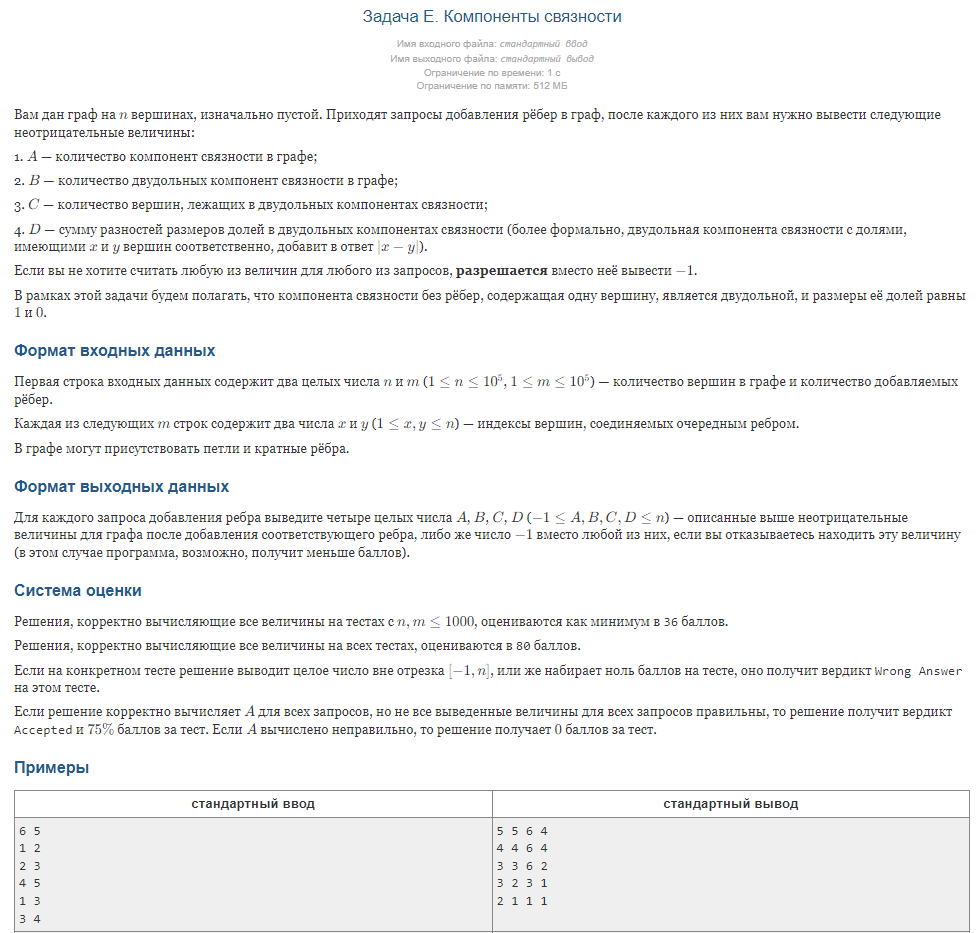
****

****

**CONTEST**

****

****

****