Задача 1. Муравьи

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Что только не придумаешь в процессе утомительной погони за сокровищами! Например, невероятно увлекательную игру «Муравьи».

Суть игры следующая: задано игровое поле размера $n \times m$, в некоторых клетках которого находятся стулья. Также на поле есть муравейник. Из него, начиная с первой секунды, с интервалом в одну секунду выползают муравьи, которые устремляются каждый к своему любимому стулу по кратчайшей траектории. За одну секунду муравей переходит из клетки, где он находится, в любую смежную с ней по стороне. Достигнув своего любимого стула, муравей радостно аннигилирует. Всё, что остаётся игроку, — это сидеть и считать, в какую секунду сколько муравьёв закончит свой жизненный путь.

Муравей достаточно мал, стулья его нисколько не задерживают. Заметим, что стул может стоять в любой клетке поля, в том числе, и на муравейнике.

Формат входных данных

В первой строке входного файла записаны пять целых чисел n, m, k, r, c — соответственно, размеры поля, количество стульев и координаты муравейника ($1 \leqslant n, m \leqslant 100$, $1 \leqslant k \leqslant n \cdot m$, $1 \leqslant r \leqslant n, 1 \leqslant c \leqslant m$).

Далее приводится описание поля — это прямоугольная таблица из n строк и m столбцов. В каждой клетке таблицы содержится некоторое неотрицательное целое число i ($i \leqslant k$). Если $i = 0$, то это пустая клетка, иначе — в данной клетке находится стул с номером i.

Гарантируется, что все целые числа от 1 до k встречаются в этой таблице по одному разу. Любимый стул муравья, который выползает из муравейника в секунду i, имеет номер i.

Формат выходных данных

В первую строку выходного файла необходимо вывести целое число e — количество событий, описывающих самоликвидацию муравьёв. Далее в e строках требуется описать собственно события — по два целых числа — номер секунды и количество взорвавшихся в эту секунду муравьёв. Моменты времени в событиях должны строго возрастать, а количество муравьёв в каждом событии должно быть положительным.

Примеры

input.txt	output.txt
3 5 4 2 5 0 0 1 0 0 0 0 0 0 3 0 4 0 0 2	3 3 2 4 1 8 1

Задача 2. Blend

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

В одной популярной программе для проектирования зданий есть операция по созданию твёрдого тела под названием “Blend”. В данную операцию передаются два контура, лежащие в параллельных плоскостях, и она строит тело, расположенное между этими контурами. Область, которую ограничивает каждый контур в своей плоскости, становится гранью-крышкой построенного тела, а между контурами достраиваются боковые грани.

Боковая поверхность строится следующим образом (см. рисунок 1 после примеров). На каждом контуре выбирается некоторая начальная точка. Кроме того, на каждом контуре заранее выбрана ориентация, то есть выбрано одно из двух направлений обхода. Выбранные начальные точки соединяются *активным отрезком*. Далее концы активного отрезка ведут вдоль контуров: конец на нижнем контуре проводят вдоль нижнего контура, а конец на верхнем контуре проводят вдоль верхнего контура. В ходе ведения активный отрезок заметает боковую поверхность. Чтобы получить таким образом всю боковую поверхность искомого тела, нужно провести каждый конец отрезка по соответствующему контуру в точности на один полный оборот. Вести конец активного отрезка против выбранного направления обхода контура запрещено.

В данной задаче мы будем считать, что каждый контур является замкнутой ломаной, каждое звено которой является отрезком. Можно заметить, что описанное выше построение неоднозначно: можно построить самые разные боковые поверхности для заданных контуров, изменяя скорости ведения концов активного отрезка. Чтобы разрешить эту неоднозначность, вводятся дополнительные правила. Когда один конец активного отрезка находится в вершине ломаной, то и второй конец также должен находиться в вершине (другой) ломаной. Такие положения активного отрезка определяют *боковые рёбра* построенного тела, а части боковой поверхности между соседними боковыми рёбрами называются *боковыми гранями*.

Получается, что каждая боковая грань построенного тела должна либо соединять одно звено одной ломаной с вершиной другой ломаной (треугольник), либо соединять одно звено одной ломаной с одним звеном другой (четырёхугольник). Четырёхугольник может даже быть криволинейным: в этом случае его точная геометрия зависит от скорости ведения концов активного отрезка. В реальной практике эти скорости также нормируются, но для данной задачи это уже значения не имеет.

Получается, что результирующее тело чётко определено, если выбрать, какие построить боковые рёбра, то есть как сопоставить друг другу вершины заданных ломаных. В общем случае это сопоставление определяет инженер — пользователь программы. Однако для простоты программа также предлагает сопоставление по умолчанию. Оно выбирается таким образом, чтобы суммарная длина всех боковых рёбер была минимально возможной. От Вас требуется написать программу, которая будет вычислять это сопоставление по умолчанию по заданным входным контурам.

Обратите внимание, что данное описание ничего не говорит о самопересечениях и самокасаниях боковой поверхности. Естественно, при наличии подобных проблем построенное тело, строго говоря, не будет твёрдым телом. Такие аномалии вполне возможны и разрешены при построении тела с помощью операции “Blend”, в том числе при выборе боковых рёбер по умолчанию. Обнаружением и исправлением подобных проблем занимаются совсем другие алгоритмы.

Формат входных данных

В первой строке записано три целых числа: M — количество вершин в нижней ломаной, N — количество вершин в верхней ломаной и H — высота тела ($3 \leqslant M, N \leqslant 300, 1 \leqslant H \leqslant 10^6$).

В следующих M строках задаются координаты вершин нижней ломаной. В каждой строке указано два целых числа: x- и y- координаты очередной вершины. Вершины описаны в выбранном порядке обхода ломаной, после последней заданной вершины в обходе следует первая заданная. В оставшихся N строках записаны координаты верхней ломаной в том же формате.

Все координаты не превышают 10^6 по абсолютной величине. Нижняя ломаная лежит в плоскости $z = 0$, а верхняя — в плоскости $z = H$. У каждой ломаной все вершины различны, самопересечения и самокасания разрешены.

Формат выходных данных

В первую строку выходного файла необходимо вывести вещественное число A — суммарная длина боковых рёбер в вашем решении, и целое число K — количество боковых рёбер в нём ($\max(M, N) \leqslant K \leqslant M + N$).

В оставшихся K строках требуется вывести боковые рёбра в том порядке, в котором они заметаются активным отрезком. Начинать можно с любого бокового ребра. Для каждого бокового ребра в отдельной строке выведите два целых числа i и j — номера вершин в нижней и верхней ломаной соответственно, которые соединяет это ребро ($1 \leqslant i \leqslant M$, $1 \leqslant j \leqslant N$).

Абсолютное или относительно отклонение значения A в вашем решении от оптимального не должно превышать 10^{-9}.

Примеры

input.txt	output.txt
3 3 1 0 0 2 0 1 1 3 -1 1 2 -1 -1	4.878315177510850 3 1 3 2 1 3 2
9 3 3 0 2 1 0 2 0 7 0 8 1 8 5 3 7 2 7 0 6 6 2 2 6 2 2	33.210944197060996 9 1 3 2 3 3 3 4 1 5 1 6 1 7 2 8 2 9 2

Иллюстрация

(на отдельной странице)

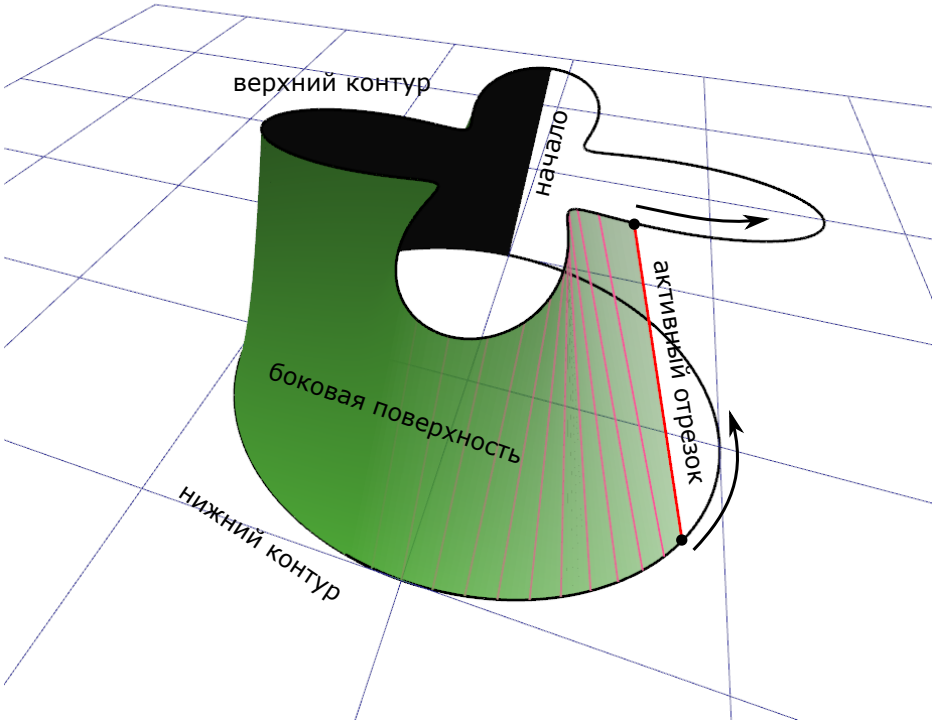


Рис. 1: Заметание боковой поверхности активным отрезком.

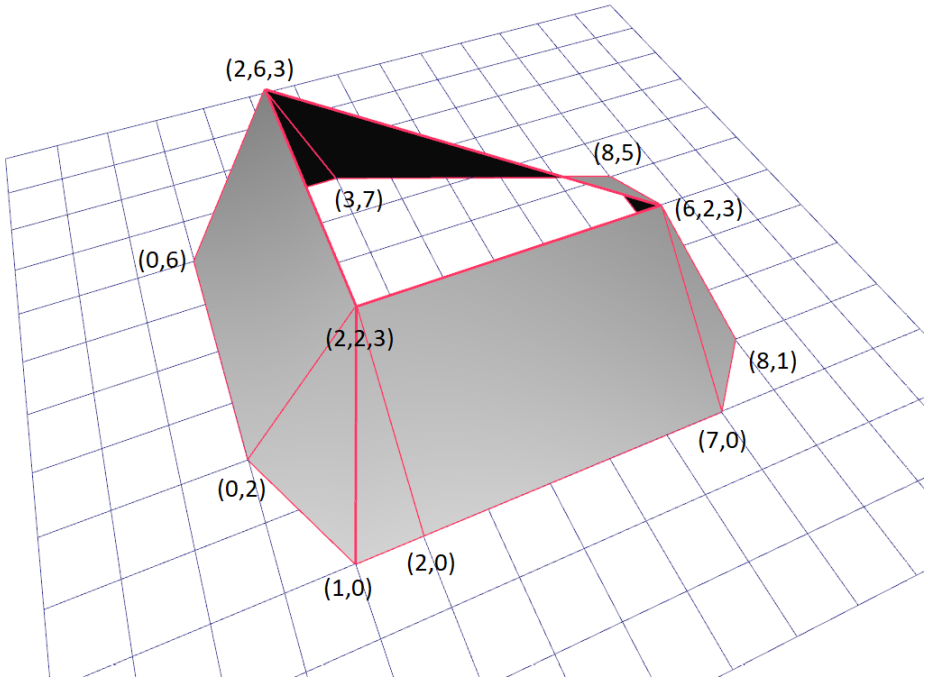


Рис. 2: Второй пример: боковые рёбра и грани.

Задача 3. Резервное копирование

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Известно множество стратегий резервного копирования. Самые простые для понимания из них – это дифференциальное и инкрементальное резервное копирование.

При инкрементальном резервном копировании, мы создаем копию данных, измененных с момента предыдущего бэкапа: например, в воскресенье мы делаем полную копию, в понедельник копируем данные, измененные за понедельник, во вторник – за вторник, и т.д. При таком копировании, объем копируемых данных минимален, но для восстановления может потребоваться много файлов. Например, если мы восстанавливаем данные в пятницу, нам нужны будут файлы резервной копии за воскресенье (полный бэкап), понедельник, вторник, среду и четверг.

При дифференциальном резервном копировании мы периодически (например, каждое воскресенье) создаем полную копию. Затем, в каждый день недели мы создаем копию всех данных, измененных с даты последнего полного копирования: в понедельник за понедельник, во вторник – за понедельник и вторник, и т.д. При этом, общий объем копируемых данных относительно (по сравнению с инкрементальным копированием) велик, но для восстановления данных нужно всего два файла: последний дифференциальный и последний полный бэкап.

Популярная бесплатная утилита для резервного копирования dump(1) вводит понятие уровня бэкапа, при помощи которого можно воспроизводить как инкрементальное или дифференциальное копирование, так и сложные стратегии, не сводимые ни к тому, ни к другому. Сама по себе идея уровня копирования очень проста: когда мы делаем бэкап уровня N, мы копируем все файлы, измененные с момента последнего копирования с уровнем меньше N. Если бэкапа меньшего уровня ранее не делалось, мы делаем полную копию всех данных.

Напишите программу, которая на основании расписания резервного копирования определяет, какие файлы бэкапа необходимо использовать для восстановления.

Формат входных данных

Первая строчка входного файла содержит число M – общее количество тестов в файле ($0 < M \leqslant 100\,000$).

Каждая следующая строка входного файла состоит из восьми целых чисел, разделенных пробелами. Первые семь из них – расписание резервных копий, то есть уровни бэкапа N_i, которые делаются в соответствующие дни недели i: в воскресенье, понедельник, вторник и т.д. ($0 \leqslant N_i \leqslant 9$). Гарантируется, что в воскресенье делается бэкап с нулевым уровнем, то есть первое число равно нулю.

Восьмое число – это день d, в который необходимо восстановить данные ($0 \leqslant d \leqslant 6$). Здесь 0 соответствует воскресенью, 6 – субботе.

Формат выходных данных

Для каждой строки входного файла, выведите последовательность чисел d_j, соответствующих дням, в которые были сделаны резервные копии, которые необходимо использовать при восстановлении ($0 \leqslant d_j \leqslant 6$). Обратите внимание, что всегда сначала восстанавливают более раннюю копию, а затем более позднюю, поэтому должны соблюдаться требования: $d_j < d_{j+1}$ и $N_{d_j} < N_{d_{j+1}}$. Также, восстановление всегда заканчивается самой поздней копией, поэтому последнее число в строке вывода должно быть равно d.

Примеры

input.txt	output.txt
4 0 0 0 0 0 0 0 4 0 8 8 8 8 8 8 3 0 2 3 4 5 6 7 4 0 7 2 6 3 5 4 4	4 0 3 0 1 2 3 4 0 2 4

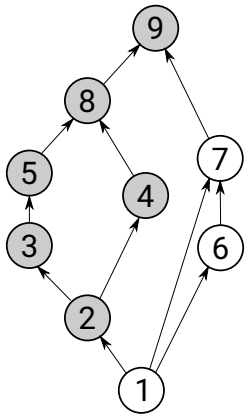
Задача 4. bit gisect

Имя входного файла:	`стандартный поток ввода`
Имя выходного файла:	`стандартный поток вывода`
Ограничение по времени:	3 секунды
	5 секунд (для Java)
Ограничение по памяти:	256 мегабайт

Программист Вася для хранения своего кода использует систему контроля версий bit. Историю изменения кода в bit можно представить в виде ориентированного графа без контуров. Вершины данного графа называются ревизиями.

Когда Вася хочет что-то изменить в коде, он может взять произвольную ревизию A, изменить код и получить новую ревизию D. В этом случае будем говорить, что ревизия D зависит от ревизии A. Каждая такая зависимость в bit выражена дугой от вершины ревизии A к вершине ревизии D. Особым типом ревизий в bit являются слияния: Вася может взять две произвольные различные ревизии A и B и объединить их в одной новой ревизии D. В этом случае ревизия D будет зависеть от двух ревизий: A и B. В репозитории существует только одна «начальная» ревизия: это ревизия, которая не зависит ни от какой другой ревизии. Все остальные ревизии в bit зависят от одной или двух других ревизий. Проект у Васи находится в законченном состоянии: существует ровно одна ревизия, от которой никакая другая ревизия не зависит.

И вот как-то раз случилась беда, и где-то в коде закрались баги, а Вася всё никак не может их найти. Баги у Васи работают следующим образом: если в какой-то ревизии A появился баг, то этот баг проявляет себя во всех ревизиях, которые достижимы в графе bit из A: это сама ревизия A, все ревизии, которые зависят от A, все ревизии, которые зависят от ревизий, зависящих от A и т.д. Для поиска бага Вася может переключаться между ревизиями и проверять его наличие в разных ревизиях, таким образом сужая список ревизий, в одну из которых закрался баг, и в результате найти именно ту ревизию, в которой этот баг был допущен.



Васе от пользователей поочерёдно поступают сообщения о найденных багах, и он должен последовательно найти все баги. Помогите Васе как можно скорее исправить код.

На рисунке изображен пример из девяти ревизий. Ревизия `1` является начальной. Ревизии `7`, `8` и `9` являются слияниями и зависят от двух ревизий. Серым цветом выделена ревизия `2`, в которой допущен баг, и все ревизии, которые подвержены данному багу.

Протокол взаимодействия

Это интерактивная задача, и в ней вам предстоит работать не с файловым вводом-выводом, а со специальной программой — интерактором. Взаимодействие с ней осуществляется через стандартные потоки ввода-вывода.

При старте вашей программе в стандартный поток ввода подаётся информация о ревизиях. Первая строка содержит целое число N — количество ревизий в bit ($1 \leqslant N \leqslant 1\,000$). Следующие N строк содержат описания ревизий, каждое из которых начинается с её номера — строки из 6 шестнадцатиричных цифры (цифры от `0` до `9` и строчные латинские буквы от ‘`a`’ до ‘`f`’). Гарантируется, что в данном описании нет ревизии с номером `000000`. Затем идёт целое число k — количество ревизий, от которых зависит текущая ревизия ($0 \leqslant k \leqslant 2$) и через пробел указано k различных номеров ревизий. Гарантируется, что на момент описания

очередной ревизии были даны описания всех ревизий, от которых она зависит.

Далее Васе предстоит находить баги. Гарантируется, что всего в коде Вася допустил не более 10 000 багов.

Поиск очередного бага начинается с того, что вашей программе на вход подаётся строка `bug` и через пробел задаётся номер ревизии, в которой обнаружен очередной баг. Если в качестве номера ревизии дана строка `000000`, то это означает, что все баги были найдены и вашей программе следует завершить работу. Иначе начинается процесс поиска бага.

Далее ваша программа должна отправлять запросы на стандартный поток вывода. Каждый запрос должен состоять из одной строки, в которой через пробел записаны команда и номер ревизии, над которой требуется выполнить команду. Команда может быть одной из:

- `check` — проверить указанную ревизию на наличия в ней бага. В ответ вам будет дана строка `bad`, если данная ревизия подвержена багу и `good`, если в ней бага нет.
- `fix` — означает, что вы нашли ревизию, в которой закрался баг. После команды `fix` ваша программа должна перейти к поиску следующего бага.

На каждый баг ваша программа может сделать не более 20 запросов, иначе решение получит вердикт Wrong Answer.

Убедитесь, что вы выводите символ перевода строки и очищаете буфер потока вывода (команда `flush` языка) после каждого выведенного запроса. Иначе решение может получить вердикт Timeout.

Пример

Для удобства чтения команды в примере разделены пустыми строками.

стандартный поток ввода	стандартный поток вывода
9 000001 0 000002 1 000001 000003 1 000002 000004 1 000002 000005 1 000003 000006 1 000001 000007 2 000006 000001 000008 2 000004 000005 000009 2 000007 000008 bug 000008 bad good bad bug 000009 bad good bug 000000	check 000004 check 000001 check 000002 fix 000002 check 000007 check 000006 fix 000007

Задача 5. Слоты

Имя входного файла:	`input.bin`
Имя выходного файла:	`output.bin`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

В программировании компьютерных игр есть свои общеизвестные способы решения часто встречающихся задач, которые также называются «паттернами». Один такой паттерн связан с размещением игровых объектов в массиве.

Допустим, имеется массив A фиксированного размера N, состоящий из *слотов* с номерами от 1 до N. В ходе игры можно создавать и уничтожать игровые *объекты*. Каждый объект с момента создания должен быть прописан в некоторый слот в массиве A. После уничтожения объекта занимаемый им слот освобождается, и в дальнейшем этот слот может быть переиспользован для вновь создаваемых объектов.

Помимо этого, у каждого объекта также есть целочисленный *идентификатор*, который уникален для объекта в течение всей игры. Идентификаторы выдаются создаваемым объектам последовательно, начиная с единицы.

Наконец, чтобы создание новых объектов выполнялось за $O(1)$, дополнительно поддерживается стек всех свободных на текущий момент слотов массива A.

При создании нового объекта выполняются следующие действия:

1. Извлечь номер слота x с вершины стека свободных слотов.
2. Прописать новый объект в слот x.
3. Присвоить новому объекту идентификатор, который на единицу больше последнего выданного идентификатора, либо 1, если это самый первый объект в игре.

При уничтожении объекта выполняются следующие действия:

1. Слот x, в котором лежал уничтожаемый объект, становится свободным.
2. Номер слота x добавляется на вершину стека свободных слотов.

Такая система позволяет размещать игровые объекты по слотам массива. Причём операции создания и уничтожения работают за константное время, а благодаря идентификаторам можно хранить «слабые ссылки» на объекты — то есть такие ссылки, по которым можно в любой момент узнать, жив ещё соответствующий объект, или он уже был уничтожен.

В данной задаче требуется определить, можно ли получить заданную конфигурацию в описанной системе слотов, и, если можно, то как сделать это за минимальное количество операций. Разрешается выполнять два типа операций: создать новый объект и уничтожить объект в заданном слоте.

Про каждый слот известно, должен ли он быть свободным или занятым в конце игры. Если слот должен быть занятым, то известно, какой идентификатор должен быть у лежащего в нём объекта.

Изначально все слоты пусты, живых объектов нет, игра только началась. В стеке свободных слотов лежат все слоты по порядку, первый слот массива A лежит на вершине стека.

Внимание: в данной задаче входной файл **input.bin** и выходной файл **output.bin** хранят **бинарные данные**! Все числа записаны в нативном для проверяющих машин виде, порядок байтов little-endian.

Формат входных данных

Входной файл состоит из $(N + 1)$ целых 32-битных чисел. Первое число равно N — количеству слотов ($1 \leqslant N \leqslant 10^6$). Остальные N чисел определяют конфигурацию, которую нужно получить. Каждое число показывает, что должно быть в соответствующем слоте:

- 0 — означает, что слот должен быть пустым,
- $k > 0$ — означает, что в слоте должен быть объект с идентификатором k.

Гарантируется, что все записанные идентификаторы различны и не превышают $2 \cdot 10^6$.

Формат выходных данных

Если получить требуемую конфигурацию невозможно, то в выходной файл нужно вывести одно 32-битное целое число -1.

Иначе, нужно вывести $(M + 1)$ целых 32-битных чисел. Первое из этих чисел равно M — минимальному количеству операций, за которое можно сделать нужную конфигурацию. Остальные M чисел задают сами операции в порядке их выполнения.

Каждая операция кодируется одним числом:

- 0 — означает, что нужно создать новый объект,
- $1 \leqslant x \leqslant N$ — означает, что нужно уничтожить объект, который лежит в слоте x.

Все операции должны быть корректными: нельзя уничтожать объект из свободного слота, и нельзя создавать объект, когда свободных слотов нет.

Примеры

В примерах показано содержимое входного и выходного файлов в шестнадцатеричном виде. В системе тестирования файлы будут в бинарном виде. Примеры в бинарном виде можно скачать на вкладке «Новости» рядом с условиями.

input.bin
0A 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00 05 00 00 00 0A 00 00 00 09 00 00 00 08 00 00 00 00 00 00 00 0C 00 00 00
output.bin
0F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 09 00 00 00
input.bin
05 00 00 00 05 00 00 00 04 00 00 00 03 00 00 00 02 00 00 00 01 00 00 00
output.bin
FF FF FF FF

Пояснение к примеру

В первом примере последовательно создаются 8 объектов, которые занимают первые 8 слотов с идентификаторами от 1 до 8. Далее объекты в слотах 6 и 7 (с идентификаторами 6 и 7 соответственно) уничтожаются, причём в таком порядке, чтобы на вершину стека свободных слотов попал слот 7. После этого создаются два объекта: они получают следующие по порядку идентификаторы 9 и 10, и попадают в слоты 7 и 6 соответственно. Наконец, создаётся ещё два объекта с идентификаторами 11 и 12, попадающие в последние два слота, после чего объект с идентификатором 11 в слоте 9 уничтожается.

Во втором примере получить требуемую конфигурацию нельзя.

Задача 6. Сортировка чая

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	3 секунды
Ограничение по памяти:	256 мегабайт

Устав от тяжёлого труда, Стёпа решил попить чаю — так сказать, устроить «пятичасовой чай». Ведь вы знаете, как программисты любят чай, причём самый разный: чёрный, зелёный, красный, белый, коричневый и так далее. Зайдя на общую кухню, он заглянул в ящик с чаем, и обнаружил, что кто-то перемешал в ящике пачки чая разных сортов. Стёпа любит порядок: он хочет, чтобы пачки чая были уложены в точности так, как он привык.

В ящике лежат N пачек чая, уложенные в K рядов. В каждом ряду пачки лежат последовательно, от самой ближней пачки к самой дальней. Но ящик очень низкий, поэтому достать можно только самую ближнюю пачку в каждом ряду. Таким образом, единственная операция, которую можно сделать — это достать самую ближнюю пачку из некоторого ряда i и положить её в ряд j, где она станет самой ближней. Как говорят программисты на своём птичьём языке, каждый ряд — это стек, он же «LIFO».

На каждой пачке написан сорт чая в виде целого числа. Стёпа любит, когда:

1. в каждом ряду одинаковое количество пачек,
2. сорт пачки из ряда i меньше или равен сорту пачки из ряда j (при любых $i < j$),
3. в каждом ряду пачки упорядочены по возрастанию сорта в порядке от самой ближней пачки к самой дальней.

Нужно предложить план, в котором не более $13 \cdot N$ операций перекладывания пачки с места на место, после выполнения которого всё лежит «как Стёпа любит». При этом минимизации количества операций не требуется.

Формат входных данных

В первой строке входного файла записано два целых числа N и K — сколько всего пачек чая и во сколько рядов они уложены ($1 \leqslant N \leqslant 10^5$, $11 \leqslant K \leqslant 111$).

Далее записано K строк, в каждой r-ой строке задаётся r-ый ряд пачек чая. Для каждого ряда записано сначала целое число T_r — сколько пачек в этом ряду, а затем T_r целых чисел — сорта чая в пачках этого ряда ($0 \leqslant T_r \leqslant N$). Пачки в ряду перечисляются в порядке от самой ближней до самой дальней.

Гарантируется, что сумма всех T_r равна N, и что N делится нацело на K. Целые числа, обозначающие сорта чая, не превышают 10^9 по абсолютной величине.

Формат выходных данных

В первую строку выходного файла необходимо вывести число M — количество операций в вашем плане ($0 \leqslant M \leqslant 13 \cdot N$).

В оставшиеся M строк требуется вывести операции в порядке их выполнения. Для каждой операции нужно вывести через пробел два целых числа: i — номер ряда, из которого нужно взять ближнюю пачку, и j — номер ряда, куда следует её положить ($1 \leqslant i, j \leqslant K$).

Примеры

input.txt	output.txt
33 11 3 -1 1 1 3 1 1 1 3 1 2 2 3 3 3 3 3 3 4 4 3 5 7 7 3 7 7 7 3 7 8 8 3 8 9 9 3 9 9 9 3 9 9 9	0
33 11 3 -1 1 1 3 1 1 1 3 1 2 2 0 5 3 4 3 3 4 4 5 7 7 3 3 7 7 7 3 7 8 8 3 8 9 9 3 9 9 9 3 9 9 9	13 6 7 6 7 6 7 6 4 7 6 7 6 7 6 5 4 5 11 5 4 5 4 11 5 4 5

Пояснение к примеру

В первом примере все пачки лежат в точности так, как Стёпа любит, так что делать ничего не нужно.

Во втором примере ряд 4 пустой, а те пачки, которые должны там лежать, лежат в рядах 5 и 6. Первые семь операций показанного плана извлекают чай сорта 3 из ряда 6. Обратите внимание, что эта пачка самая дальняя, поэтому чтобы её достать, нужно сначала вытащить три пачки перед ней. Эти пачки перебрасываются в ряд 7, куда они помещаются в обратном порядке, а после убирания пресловутой пачки сорта 3 они перекладываются обратно, снова в правильном порядке. Остальные шесть операций убирают из ряда 5 в ряд 4 две пачки чая сорта 3. Вместе с этим пачка чая сорта 4 в этом ряду сдвигается подальше.

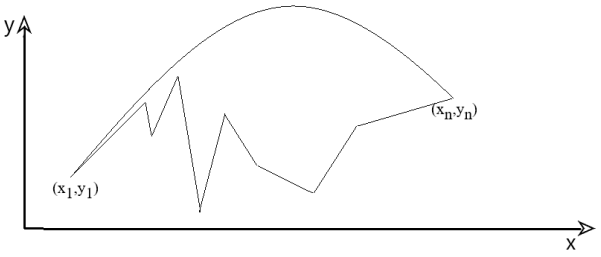
Задача 7. Стрельба

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Отец Фёдор с куском похищенной колбасы скрылся в горах, и поэтому концессионерам не остается ничего другого, кроме как пытаться поразить его камнем (не драгоценным, конечно, а обычным) в глаз, как советуют старые мудрые люди.

Остап и Киса находятся у подножия горы и бросают камень, целясь похитителю точно в глаз. Ваша задача — найти минимальную начальную скорость, с которой они могут бросить камень так, чтобы он поразил отца Фёдора, минуя все препятствия на пути.

Для простоты рельеф местности, где располагаются Остап, Киса и отец Фёдор, представляет собой ломаную линию на плоскости. Координата y на рассматриваемой плоскости обозначает высоту. Размерами людей можно пренебречь, поэтому Остап и Киса представляются одной точкой на рельефе, а глаз отца Фёдора — другой точкой.



Камень летит по параболе под действием ускорения свободного падения, которое следует принять равным $g = 10$. Сопротивлением воздуха пренебрегаем. На своём пути камень должен лететь выше ломаной.

Формат входных данных

В первой строке входного файла задано число n — количество вершин ломаной ($2 \leqslant n \leqslant 10^5$). Далее во входном файле задаётся n строк. В i-ой из них содержится два целых числа x_i, y_i — координаты i-ой вершины ломаной ($0 \leqslant x_i, y_i \leqslant 10^7$, $1 \leqslant i \leqslant n$). Гарантируется, что последовательность x_i строго возрастает, то есть $x_i < x_j$ при $i < j$.

Известно, что Остап и Киса находятся в первой точке ломаной (x_1, y_1), а отец Фёдор находится в последней точке (x_n, y_n).

Формат выходных данных

В выходной файл нужно вывести одно вещественное число — минимально возможную начальную скорость камня, позволяющую попасть в отца Фёдора.

Относительная или абсолютная погрешность ответа не должна превышать 10^{-6}.

Примеры

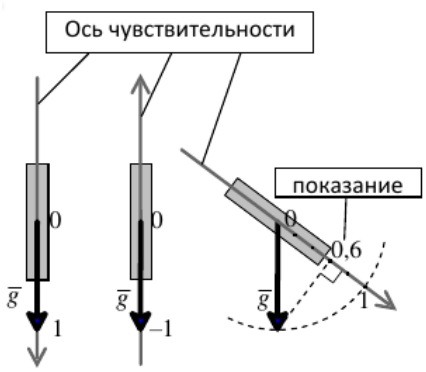
input.txt	output.txt
6 1 2 4 4 5 0 6 3 8 4 11 2	10
3 0 0 1 3 5 2	10.4963363572

Задача 8. Калибровка акселерометров

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Современные умные электронные устройства умеют выполнять функции навигатора, шагомера, распознавать, как передвигается пользователь, — идет, бежит, едет в такси или в автобусе, умеют автоматически ориентировать изображение на экране. При решении всех этих и многих других задач используются, так называемые, акселерометры. С простейшим одноканальным акселерометром связано некоторое направление — его ось чувствительности; показания неподвижного акселерометра позволяют вычислить отклонение его оси от направленной вниз вертикали, т.е. от направления силы тяжести. Если на устройстве закреплено несколько одноканальных акселерометров, то по положениям их осей можно определить ориентацию в пространстве всего устройства в целом.

Идеальный акселерометр измеряет косинус угла между своей осью чувствительности и направлением силы тяжести. Если ось чувствительности направлена вниз, то есть вдоль вектора силы тяжести $\overline{g}$, то акселерометр показывает значение 1, а если вверх — то показывает число -1. Если же ось направлена под углом к вертикали, то показание идеального акселерометра равно проекции на неё единичного вектора силы тяжести $\overline{g}$:



При изготовлении миниатюрных акселерометров невозможно полностью избежать дефектов. Дефекты датчиков приводят к таким ошибкам показаний:

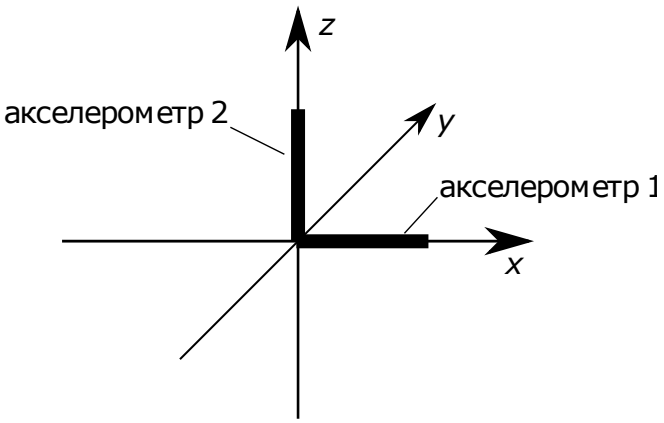
1. отклонение оси чувствительности акселерометра от положенного по проекту направления (дефект крепления);
2. пропорциональное изменение показаний, т.е. увеличение или уменьшение всех показаний согласно некоторому множителю;
3. систематический сдвиг всех показаний на определенную величину.

Если на устройстве установлено несколько акселерометров, то описанные ошибки проявляются по-своему в каждом из них. Однако, если рассмотреть серию показаний одного акселерометра в разных положениях, то оказывается, что ошибки влияют одинаково на все его показания.

Чтобы выяснить, как именно акселерометры искажают свои показания, и произвести цифровую коррекцию этих искажений, проводят процедуру калибровки. Один из способов калибровки — это считывание показаний акселерометров в нескольких точно зафиксированных положениях устройства и определение по этим данным параметров вышеописанных

дефектов для каждого акселерометра. Эти параметры в дальнейшем можно применять для определения ориентации устройства в произвольном положении.

В данной задаче мы будем рассматривать устройство с двумя жёстко закреплёнными акселерометрами. Когда устройство ориентировано в пространстве стандартным образом, то согласно проекту, один акселерометр должен быть направлен вдоль оси X вправо, а второй — вдоль оси Z вверх (противоположно силе тяжести). Пользователь может вращать устройство: мы будем считать, что вращать можно только в плоскости XZ. В таком случае ориентация устройства полностью определяется углом поворота вокруг оси Y от стандартного положения против часовой стрелки (то есть от оси X к оси Z).



Для калибровки такого акселерометра производят замеры показаний акселерометров при повороте устройства на углы, кратные 90 градусам, что позволяет определить все параметры дефектов. Вам требуется на основе предоставленных наборов данных для калибровки определить параметры дефектов и научиться по показаниям акселерометров определять реальное положение устройства.

Формат входных данных

Первые четыре строки входного файла содержат показания датчиков для углов поворота устройства, равных 0, 90, 180 и 270 градусам соответственно.

Следующая строка входного файла содержит целое число T — количество показаний, для которых требуется определить положение устройства ($1 \leqslant T \leqslant 1\,000$). Каждая из оставшихся T строк содержит показания датчиков для угла поворота устройства, который необходимо определить.

Каждое показание датчика содержит по два вещественных числа, записанных через пробел — показания первого, направленного вдоль оси X, и второго, направленного вдоль оси Z, акселерометров соответственно. Оба числа задаются с 15-ю знаками после десятичной точки.

Гарантируется, что показания во входном файле получены по описанной в условии модели измерений и ошибок. Ошибки в показаниях не могут быть сильно большими: отклонение оси чувствительности от проектного никогда не превышает 30°, пропорциональное изменение показаний изменяет их не более чем в два раза, а смещение не превышает 5 по абсолютной величине.

Формат выходных данных

В выходной файл требуется вывести T чисел, по одному в каждой строке. Каждое число — это угол поворота устройства в градусах для соответствующих показаний акселерометров.

Угол поворота должен быть в диапазоне от 0° до 360° включительно. Относительная или абсолютная погрешность каждого ответа не должна превышать 10^{-6}.

Примеры

input.txt	output.txt
0 -1 -1 0 0 1 1 0 5 -0.707106781186547 -0.707106781186547 -0.707106781186547 0.707106781186547 0.707106781186547 0.707106781186547 0.707106781186547 -0.707106781186547 -0.866025403784438 -0.5	45 135.00 225 315 60
-0.091012995433623 -0.946575228282571 -0.983288528313429 0.028440168472892 0.291012995433623 0.846575228282571 1.183288528313429 -0.128440168472892 5 -0.801067248717891 -0.628508848717885 -0.530934079986151 0.639439998807080 1.001067248717891 0.528508848717886 0.730934079986151 -0.739439998807080 -0.933661882864499 -0.430356435566630	45.000000000000043 135.000000000000000 225.000000000000000 315.000000000000000 59.999999999999986

Пояснение к примеру

В первом примере у акселерометров отсутствуют какие-либо дефекты.

Во втором примере на каждый из двух акселерометров влияют все три компонента ошибки, причём в разной мере.

Задача 9. Деревья

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Программист Вася продолжает изучать теорию графов. Он прочитал главу про деревья и уже несколько дней ломает голову над следующей задачей: необходимо построить корневое дерево с N вершинами, каждая из которых, кроме листьев, имеет K детей. Причем ответ нужно записать в виде списка рёбер, а среди всех возможных вариантов ответа надо найти лексикографически минимальный.

Список рёбер графа записывается в строку следующим образом. Каждое ребро графа описывается парой целых чисел — номерами вершин, которые это ребро соединяет. Эти два числа должны быть записаны без ведущих нулей, и между ними должен стоять ровно один символ пробела. Строка состоит из описаний всех $N-1$ рёбер графа, записанных подряд и отделённых друг от друга одним пробелом. Предполагается, что все вершины пронумерованы числами от 1 до N, причём корень имеет номер 1.

Васе нужно найти лексикографически минимальную строку, которую можно получить таким образом по корневому дереву требуемого вида. При лексикографическом сравнении следует считать, что пробел как символ меньше всех цифр.

Например, пусть требуется построить дерево из 5 вершин, все нелистовые вершины которого должны иметь по 2 ребёнка. Под заданное требование подходит дерево с рёбрами $(1,4),(1,5),(4,3),(4,2)$. Список его рёбер можно записать в виде строки разными способами:

- `4␣2␣4␣3␣1␣4␣1␣5`
- `2␣4␣3␣4␣1␣4␣1␣5`
- `1␣4␣1␣5␣2␣4␣3␣4`

Здесь каждый следующий вариант записи лексикографически меньше предыдущего, но ни один не является оптимальным. При таких значениях N и K лексикографически минимальную строку `1␣2␣1␣3␣2␣4␣2␣5` даёт другое дерево.

Помогите Васе решить задачу перед контрольной работой по теории графов.

Формат входных данных

В первой строке входного файла записано два целых числа N и K, где N — количество вершин в искомом дереве, K — количество детей у нелистовых вершин ($2 \leqslant N \leqslant 10^5$, $1 \leqslant K \leqslant 10^5$).

Формат выходных данных

Если дерева с заданными параметрами не существует, то выведите слово `No` в единственную строку выходного файла.

Иначе в первую строку выходного файла нужно вывести слово `Yes`, а во вторую — искомую лексикографически минимальную строку.

Примеры

input.txt	output.txt
5 2	Yes 1 2 1 3 2 4 2 5
4 10	No

Задача 10. Порядок команд

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	5 секунд
Ограничение по памяти:	256 мегабайт

Во Всесибирской олимпиаде по программированию собирается участвовать N команд, но ни одна из них ещё не зарегистрировалась. После регистрации команды сортируются в лексикографическом порядке по их названиям. Жюри олимпиады хочет понять, в каком порядке команды будут идти в списке. Для этого было проведено следующее исследование. Изучая выступления команд на других соревнованиях, жюри собрало список возможных названий для каждой команды. Выяснилось, что команда с номером i могла использовать для регистрации любое из своих любимых названий $S_{i1}, \ldots, S_{iK_i}$.

Теперь жюри задалось интереснейшим вопросом. Верно ли, что их исследование не принесло им абсолютно никакой пользы? Иначе говоря, верно ли, что команды могут оказаться в списке в любом порядке? Если это не так, то жюри хочет узнать хотя бы один порядок команд, который получиться **не** может.

Формат входных данных

В первой строке входного файла записано целое число N — количество команд ($1 \leqslant N \leqslant 350$). Далее идёт N блоков строк, каждый из которых описывает очередную команду в порядке их номеров от 1 до N.

В первой строке описания i-й команды дано целое положительное число K_i. Блок описания i-й команды состоит из K_i+1 строки, включая строку с самим числом K_i. Следующие K_i строк содержат возможные названия i-й команды по одному в строке: $S_{i1}, \ldots, S_{iK_i}$. Название команды может состоять только из маленьких латинских букв. Каждое название непусто и имеет длину не больше 100 символов. Все K_i названий различны.

Гарантируется, что у разных команд нет совпадающих названий. Также гарантируется, что $\sum\limits_{i=1}^{N} K_i \leqslant 350$.

Формат выходных данных

Если команды могут оказаться в списке в любом порядке, в выходной файл необходимо вывести слово `YES`.

Иначе, в первую строку нужно вывести слово `NO`, а во вторую — любую перестановку номеров команд, которая получиться не могла — это N чисел, разделённых пробелами.

Примеры

input.txt	output.txt
3 1 teamname 2 vanechka ivan 4 albatross teddybear vitalya pythonists	YES

input.txt	output.txt
2 2 geometrylovers epsiszero 1 speedcoderz	NO 2 1

Задача 11. Сон Остапа

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	3 секунды
Ограничение по памяти:	256 мегабайт

Остапу снится страшный сон, в котором он находится внутри выпуклого многоугольника с N вершинами. Граница этого многоугольника разбита на три непрерывные части, при этом каждая сторона многоугольника входит ровно в одну часть. Если Остап окажется около одной части границы многоугольника, то его может поймать сладкая вдовушка. Около второй части его поджидает обманутый им архивариус Коробейников, а около третьей дежурит его злейший конкурент отец Федор.

Остап мечтает оказаться далеко от всех троих. Все три угрозы одинаково серьёзные, поэтому он хочет, чтобы расстояние до каждой части многоугольника было одинаковое. Найдите такую точку.

Формат входных данных

В первой строке входного файла записано целое число N — количество вершин ($3 \leqslant N \leqslant 40\,000$).

В каждой из следующих N строк записаны два целых числа X_i и Y_i — координаты очередной вершины многоугольника. Координаты не превосходят 10^6 по абсолютной величине. Вершины задаются в порядке обхода против часовой стрелки. Гарантируется, что многоугольник строго выпуклый.

В последней строке записаны три различных целых числа C_1, C_2, C_3, определяющих разбиение сторон многоугольника на части ($1 \leqslant C_j \leqslant N$). У каждой из трёх вершин с этими номерами одна инцидентная сторона многоугольника лежит в одной части, а другая — в другой части. Вершины занумерованы от 1 до N в том порядке, в каком они заданы.

Формат выходных данных

Если такая точка существует в первую строку выходного файла выведите слово `Yes`. В следующую строку выведите два вещественных числа X_c и Y_c — координаты точки, в которую хочет переместиться Остап. Расстояния от этой точки до трёх частей границы многоугольника должны отличаться друг от друга не более, чем на 10^{-6}, в абсолютном или относительном смысле.

Если такой точки не существует, в единственную строку выходного файла выведите слово `No`.

Примеры

input.txt	output.txt
4 8 0 5 3 3 3 0 0 4 3 2	Yes 3.62132025 1.49999996