Задача 1. Заяц-топтун

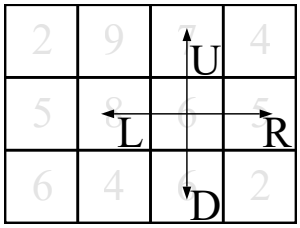
Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Заяц-топтун любит вытаптывать поля. Он только что нашёл очередное засеянное поле. На карте поле представляет собой прямоугольник, состоящий из H клеток в высоту и W клеток в ширину. Попадая в определённую клетку поля, заяц вытаптывает её. На вытаптывание он тратит некоторое время, которое задано отдельно для каждой клетки.

Заяц не определился, с какой клетки начать вытаптывать поле, но он уже придумал, какую последовательность передвижений ему нужно совершить. План передвижений он записал на бумаге, обозначив буквой каждое элементарное передвижение.

Передвижения могут быть следующие:

- буква `'U'`: перейти из текущей клетки поля в соседнюю с ней клетку, находящуюся сверху непосредственно над ней;
- буква `'R'`: перейти в соседнюю клетку справа;
- буква `'L'`: перейти в соседнюю клетку слева;
- буква `'D'`: перейти в соседнюю клетку снизу.



Для каждой из клеток поля заяц хочет узнать, какое время он потратит на реализацию своего плана, если начнёт свои передвижения с этой клетки. Заяц тратит время в начальной клетке и в каждой из клеток, куда он попадает, даже если он там уже побывал.

Заяц не собирается выходить за пределы поля. Если, начав с некоторой клетки, он в процессе своих передвижений покинул бы поле, то такую клетку он считает не подходящей для начала. Для таких клеток заяц просит выдавать ответ 0.

Формат входных данных

В первой строке входного файла задано целое число T — количество тестовых случаев ($1 \leqslant T \leqslant 100$). Далее следует T блоков.

В первой строке блока дано два целых числа H и W — высота и ширина поля соответственно ($1 \leqslant H, W \leqslant 100$).

Во второй строке блока записана непустая последовательность передвижений зайца. Каждое передвижение представлено одним из символов 'L', 'R', 'U', 'D'. Число передвижений не превосходит 100.

Следующие H строк содержат по W целых чисел — время, которое заяц тратит в соответствующей клетке поля. Эти числа лежат в диапазоне от 1 до 1 000.

Суммарное количество клеток по всем тестовым случаям не превосходит 10^5.

Формат выходных данных

Для каждого тестового случая выведите H строк по W целых чисел — ответ для каждой из клеток поля.

Примеры

input.txt	output.txt
1 3 4 RDRUL 2 9 7 4 5 8 6 5 6 4 6 2	41 38 0 0 37 33 0 0 0 0 0 0
2 1 1 D 1 5 5 RLDULRUD 4 7 8 5 8 7 9 9 5 7 3 8 5 3 6 1 6 6 9 3 5 2 1 6 1	0 0 0 0 0 0 0 76 72 49 0 0 63 51 40 0 0 47 51 63 0 0 0 0 0 0

Пояснение к примеру

На рисунке в условии изображено поле из первого примера и направления переходов.

Задача 2. Буферизация видео

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

Дано видео в формате MPEG-2. Для каждого кадра известно, сколько времени нужно на его декодирование. Требуется определить минимальный размер буфера, при котором можно показать всё видео без задержек и пропаданий кадров.

В видео-файле задана последовательность кадров в том порядке, в котором их необходимо показывать на экран. Видео нужно показывать с постоянной частотой кадров: в видео-файле задано время d между двумя соседними кадрами. Если пронумеровать кадры начиная с нуля, то нулевой кадр нужно показать в момент времени 0, первый кадр — в момент времени d, второй кадр — в момент времени $2d$, третий — в момент времени $3d$, и так далее.

Чтобы показать на экране кадр видео, нужно сначала этот кадр декодировать. Все декодированные и декодирующиеся кадры требуется хранить в буфере, за исключением тех кадров, которые уже точно никогда не понадобятся. Размер буфера указывается в кадрах и фиксирован на время всего показа. Обычно кадр можно удалить из буфера сразу после его показа, но не всегда, потому что для декодирования некоторых кадров нужно обязательно иметь какие-то другие кадры в буфере в декодированном виде.

Чтобы начать декодирование кадра f, необходимо:

1. иметь в буфере в декодированном виде все кадры, от которых зависит кадр f, и
2. иметь в буфере свободное место, куда будет помещён кадр на время декодирования.

Время, в течение которого происходит декодирование кадра, задано отдельно для каждого кадра. Разрешается декодировать не более одного кадра одновременно.

Удалить декодированный кадр f из буфера можно только если:

1. этот кадр f уже показан, и
2. все кадры, которые зависят от f, уже декодированы.

Кадры в видео-файле бывают трёх типов:

- `I`-кадр. Этот кадр ни от чего не зависит.
- `P`-кадр. Он зависит от предыдущего кадра типа `I` или `P`, причем ближайшего такого кадра.
- `B`-кадр. Он зависит от предыдущего кадра типа `I` или `P`, а также от следующего кадра типа `I` или `P`, причем ближайших таких кадров.

Например, если есть пять кадров с типами `IBBBP`, то первый кадр не зависит ни от чего, последний кадр зависит от первого, а все остальные кадры зависят от обоих этих кадров.

Последовательность декодирования кадров строго определена. Кадры нужно декодировать в том же порядке, в котором их надо показывать на экране. Единственное исключение: когда попадается `B`-кадр, который зависит от более позднего кадра, необходимо сначала декодировать этот более поздний кадр, если он ещё не декодирован. Например, если заданы кадры с типами `IBBPBBI`, то следует декодировать кадры в порядке: 0, 3, 1, 2, 6, 4, 5.

Нужно определить, какой минимальный размер буфера достаточен, чтобы можно было показать все кадры видео точно в положенное время.

Разрешается начать декодирование кадров за любое время до начала показа видео. Считаем, что показ кадра на экран, размещение кадра в буфере и удаление кадра из буфера происходят мгновенно.

Формат входных данных

В первой строке входного файла записаны два целых числа: N — количество кадров и d — время между соседними кадрами в микросекундах ($3 \leqslant N \leqslant 2 \cdot 10^5$, $1 \leqslant d \leqslant 10^9$).

В остальных N строках описаны кадры в порядке их показа. Каждый кадр задаётся своим типом (`I`, `P` или `B`) и тем, сколько микросекунд уходит на его декодирование. Время декодирования целое и лежит в диапазоне от 1 до 10^9 включительно.

Гарантируется, что первый кадр имеет тип `I`, а последний кадр имеет тип `I` или `P`.

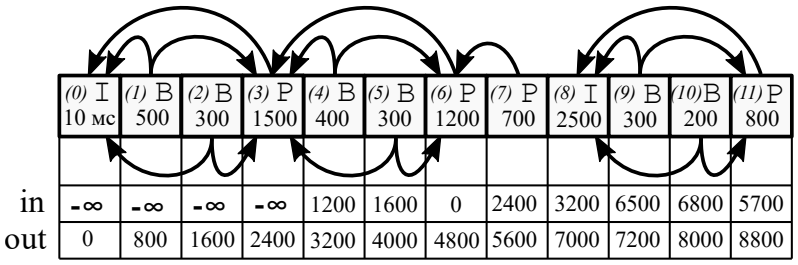
Формат выходных данных

В выходной файл требуется ввести одно целое число: минимальный размер буфера в кадрах, при котором всё видео удаётся показать вовремя.

Примеры

input.txt	output.txt
12 800 I 10000 B 500 B 300 P 1500 B 400 B 300 P 1200 P 700 I 2500 B 300 B 200 P 800	4

Пояснение к примеру



порядок декодирования: 0, 3, 1, 2, 6, 4, 5, 7, 8, 11, 9, 10;

На картинке показано, как можно проиграть видео используя буфер на 4 кадра. Стрелками показаны зависимости кадров, а в строках in/out показано, когда кадр помещается в буфер, и когда он удаляется из него. Кадры пронумерованы, начиная с нуля.

Задача 3. Пакеты

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

Для установки приложений Вася в своей операционной системе Vinux использует современный пакетный менеджер Vum.

Каждое приложение имеет несколько версий, которые можно установить. Из всех версий одного приложения можно одновременно установить только одну версию.

Будем называть конкретную версию конкретного приложения словом *пакет*. Поскольку отдельные версии разных приложений могут быть несовместимы между собой, то в Vum существуют *конфликты*. Каждый конфликт — это множество пакетов, из которых можно одновременно установить только один. Попытка установить более одного пакета из перечисленных в конфликте приведёт к тому, что в системе Vinux всё сломается, а Вася этого не хочет.

Версии любого приложения могут входить в конфликт, а могут и не входить. От одного приложения в конфликте могут участвовать несколько версий, в том числе и все версии этого приложения.

Поскольку система описания конфликтов в Vum достаточно примитивна, то каждый пакет может находиться не более, чем в одном конфликте.

Вася хочет установить N различных приложений, не сломав Vinux. Нужно выбрать версии приложений, не нарушая описанные выше условия, а переборные алгоритмы разрешения зависимостей в Vum не могут с этим справиться. Поэтому Вася обращается к вам с просьбой помочь его пакетному менеджеру.

Формат входных данных

В первой строке входного файла записано целое число N — количество приложений, которые Вася хочет поставить ($1 \leqslant N \leqslant 200$).

В следующих N строках даны описания приложений, включающие списки их версий. Описание приложения состоит из следующих полей, перечисленных через пробел:

- имя приложения — строка из строчных латинских букв длиной от 1 до 10 символов;
- количество версий — положительное целое число, не превышающее 1 000;
- номера версий — положительные целые числа, перечисленные в порядке возрастания.

Номера версий приложений не превосходят 100 000. Имена приложений не совпадают.

В следующей строке записано число K — количество конфликтов ($0 \leqslant K \leqslant 500$).

В следующих K строках описаны конфликты. Описание конфликта начинается с положительного целого числа T — количества пакетов, входящих в данный конфликт. Далее через пробел перечислено T пакетов. Для каждого пакета указано сначала имя приложения, а потом номер версии. Гарантируется, что перечисленные в конфликтах пакеты не повторяются, и что все версии были упомянуты в описании приложений.

Формат выходных данных

Если установить все N приложений без нарушения конфликтов нельзя, то в единственной строке выходного файла должно быть записано слово `No`. А если можно, то в первой строке должно быть записано слово `Yes`, а во второй — перечислены через пробел номера версий, который следует установить для достижения этого.

Версии для N приложений нужно выводить в том порядке, в котором приложения описаны во входном файле. Если существует несколько способов выбрать версии приложений,

то можно вывести любой из них.

Примеры

input.txt	output.txt
3 vim 3 1 2 3 nano 2 5 8 python 2 2 3 2 2 vim 3 nano 8 3 vim 1 nano 5 vim 2	Yes 2 8 2
2 firefox 1 38 chrome 1 46 1 2 firefox 38 chrome 46	No

Пояснение к примеру

В первом примере есть два конфликта, первый из которых запрещает одновременно поставить 3-ю версию приложения vim и 8-ю версию приложения nano. Второй конфликт говорит, что из 1-й версии vim, 5-й версии nano и 2-й версии vim в систему можно поставить лишь что-то одно, или не поставить вообще ничего. Правильными ответами будут:

- 1 8 2
- 1 8 3
- 2 8 2
- 2 8 3
- 3 5 2
- 3 5 3

Во втором примере есть два приложения, у каждого по одной версии. Однако, единственный конфликт запрещает поставить одновременно оба приложения.

Задача 4. Васин граф

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

У Васи есть граф. В этом графе целых N вершин, но вот рёбер пока нет. Васе небезразлично, какую структуру будет иметь его граф: он знает K пар вершин $\{u_j, v_j\}$ таких, что если в графе будет путь между ними, то произойдет *непоправимое*. Этого Вася допустить не может.

Вася составил список из M **неориентированных** рёбер. Вася будет рассматривать рёбра в заданном порядке и обязательно вставлять очередное ребро в граф, если может. Если при добавлении в граф очередного ребра произойдёт *непоправимое*, то такое ребро Вася просто выкинет. Ваша задача — определить, какие рёбра Вася возьмёт в граф, а какие не возьмёт.

Формат входных данных

В первой строке входного файла задано три целых числа N, K и M ($1 \leqslant N \leqslant 10^5$, $0 \leqslant K, M \leqslant 10^5$).

Далее следуют K строк, в i-й из которых даны два целых числа u_i и v_i — номера конфликтных вершин, наличия путей между которыми допускать нельзя ($1 \leqslant u_i < v_i \leqslant N$). Пары конфликтных вершин не повторяются.

Затем следуют M строк, в i-й из которых даны два целых числа $\tilde{u}_i$ и $\tilde{v}_i$ — номера вершин ребра, которое Вася может добавить в граф ($1 \leqslant \tilde{u}_i < \tilde{v}_i \leqslant N$). Эти рёбра даны в порядке, в котором Вася будет их рассматривать. Рёбра в списке не повторяются.

Формат выходных данных

В первую строку выходного файла требуется вывести количество рёбер, которые Вася возьмёт в граф. Во второй строке должны быть записаны через пробел номера рёбер в порядке возрастания.

Примеры

input.txt	output.txt
3 1 3 1 2 1 2 2 3 1 3	1 2
5 2 6 1 2 2 3 1 3 2 4 3 4 1 4 4 5 1 5	3 1 2 5

Задача 5. Квадратный трёхчлен

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

Фёдор недавно поступил в университет. Поступить удалось лишь после успешной сдачи тестовых экзаменов. При подготовке к экзаменам Федя решил несколько тысяч однотипных задач. Например, он раз за разом находил корни различных уравнений. Теперь он решил придумать свои задачи, ведь после сдачи экзаменов он может делать что угодно!

Он задумался, почему всегда в задачах просят найти корни уравнений, а не наоборот – найти уравнение по корню? Для начала он решил поэкспериментировать с полюбившимися с 9-го класса квадратными уравнениями. Однако Фёдор столкнулся с новой проблемой – знаний ему не хватает, поскольку такая задача для него нетипичная.

Помогите решить Феде новую задачу.

Формат входных данных

В первой строке входного файла задано целое число T — количество тестовых случаев ($1 \leqslant T \leqslant 100$). Далее следует T строк – по одной для каждого случая.

Каждая строка содержит одно ненулевое целое число Y — корень квадратного уравнения($1 \leqslant |Y| \leqslant 10^6$).

Формат выходных данных

В выходной файл необходимо вывести T строк, в i-й строке — ответ на i-й тестовый случай. Ответ должен содержать три **целых** числа A_i, B_i, C_i, разделённые символом пробела, такие, что i-ое число ввода является корнем квадратного уравнения $A_iX^2 + B_iX + C_i = 0$.

Все коэффициенты должны быть **ненулевыми** и **по модулю не должны превышать** 10^6. Если искомое уравнение не единственно, нужно вывести коэффициенты любого из подходящих. Гарантируется, что в каждом случае хотя бы одно такое уравнение существует.

Примеры

input.txt	output.txt
2 1 -2	2 -3 1 3 2 -8

Задача 6. Выравнивание структур

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	1 секунда
	3 секунды (для Java)
Ограничение по памяти:	**64 мегабайта**

Петя изучает язык C, и только что дошёл до структур. Его особенно заинтересовал тот факт, что поля структуры не всегда размещаются в памяти подряд: иногда между соседними полями появляются дополнительные «пустоты» (padding). Из-за этого получается, что размер структуры может зависеть от того, в каком порядке программист перечислит поля структуры!

Дано описание структуры на языке C. Оно начинается с ключевого слова `struct`, после которого в фигурных скобках описаны поля структуры, а в самом конце стоит точка с запятой. В описании поля сначала стоит тип поля, затем имя поля, и наконец точка с запятой. Тип поля гарантированно отделён от имени поля хотя бы одним пробелом. Имя поля — это непустая строка, состоящая из латинских букв любого регистра, цифр и символов подчёркивания, причём имя точно **не** начинается с цифры.

В качестве типа может быть указан либо примитивный тип, либо указатель на примитивный тип. Разрешены следующие примитивные типы:

- `char` имеет размер 1 байт.
- `short` имеет размер 2 байта.
- `int` и `float` имеют размер 4 байта.
- `int64_t` и `double` имеют размер 8 байт.

Когда поле является указателем, то в типе сначала записан какой-либо примитивный тип, а затем одна или несколько звёздочек. Независимо от примитивного типа и количества звёздочек, любой указатель имеет размер 8 байт (т.е. мы полагаем 64-битную платформу).

Структура размещается в памяти следующим образом. Первое поле структуры помещается по адресу с максимальным выравниванием: будем считать, что его адрес делится нацело на 8. Каждое следующее поле размещается по такому адресу, что:

1. оно расположено после предыдущего,
2. оно корректно выровнено, то есть его адрес делится на его размер, и
3. его адрес минимален при выполнении первых двух условий.

При этом между соседними полями может появиться пустое место (padding) размером от 1 до 7 байт.

После последнего поля также может быть добавлено несколько байт пустоты. Это необходимо, чтобы можно было хранить массив таких структур, то есть располагать в памяти много одинаковых структур друг за другом. В конце структуры добавляется минимальное количество байт пустоты, так чтобы в массиве любой длины все поля всех структур были корректно выровнены. При этом размер всей структуры равен сумме размеров всех полей и размеров всех имеющихся в структуре пустот.

Петя может переставлять местами поля структуры. Он хочет узнать, какой при этом может получиться минимальный размер структуры, и какой максимальный размер. Кроме того, Пете интересно, какой в среднем получится размер структуры, если он переставит все поля в случайном порядке.

Формат входных данных

Во входном файле описана одна структура согласно указанным в условии правилам. В

любом месте описания может быть добавлено любое количество пробелов и/или переводов строк, если их добавление не разрывает на части название примитивного типа, имя поля или ключевое слово.

Гарантируется, что в структуре от 1 до 100 полей, а общее количество символов в описании не превышает 5 000. Имена всех полей структуры отличаются друг от друга.

Формат выходных данных

В единственную строку выходного файла нужно вывести три числа через пробел: минимально возможный размер структуры, максимально возможный размер и размер структуры в среднем. Первые два числа должны быть целыми, а последнее число должно быть вещественным.

Абсолютная или относительная погрешность третьего числа не должна превышать 10^{-9}.

Пример

input.txt	output.txt

`struct  {`
`  int x;`
`  int64_t* *   Y;`
 	
`  char   _temp1;`
`} ;`
	`16 24 18.66666666666666`

Пояснение к примеру

Поле `x` имеет размер 4 байта, поле `Y` размера 8 байт, а поле `_temp1` занимает 1 байт.

Если не менять порядок полей, тогда поле `x` будет занимать байты 0-3. Придётся добавить 4 байта пустоты, чтобы поле `Y` было выровнено по 8 байтам, заняв байты 8-15. Поле `_temp1` будет иметь адрес 16 и занимать один байт, но после него нужно добавить ещё 7 байтов пустоты. Без этого поле `Y` второй структуры в массиве не может быть корректно выровнено. Получается общий размер 24 байта.

Если выбирать порядок полей, то размер 24 байта получается только когда поле `Y` находится между другими двумя полями. Так получается в 2 способах среди всех 6 спопобов выбора порядка. В остальных случаях размер структуры равен 16 байтам.

Задача 7. Ракета

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

В конструкторском бюро секретной организации X, далее *бюро*, разрабатывается амбициозный проект: строительство ракеты, которая долетит до Солнца и сядет на его поверхность. Устройство ракеты очень сложное, однако бюро уже составило все чертежи, и набор необходимых для строительства деталей утверждён.

Для i-й детали бюро считает подходящими K_i базовых материалов, каждый из которых характеризуется двумя величинами m_{ij} и c_{ij}. Здесь m_{ij} — это масса детали, если изготовить её целиком из этого базового материала, а c_{ij} — это её стоимость в этом случае.

Материалом для i-й детали может быть либо один из K_i базовых материалов, либо сплав двух базовых материалов. В случае сплава двух базовых материалов с массами m_1, m_2 и стоимостями c_1, c_2 в пропорции $\alpha \in (0;1)$, получается деталь с массой $\alpha m_1 + (1-\alpha)m_2$ и стоимостью $\alpha c_1 + (1-\alpha)c_2$. В устойчивости к нагрузкам сплава из трёх базовых материалов конструкторы не уверены. Чего уж нам об этом задумываться!

Ваша задача — решить для каждой из N деталей, из какого материала её следует изготовить, чтобы минимизировать стоимость всей ракеты. Стоимость всей ракеты равняется сумме стоимостей всех её деталей. Ракета, построенная из выбранных материалов, должна взлететь с Земли, а для этого суммарная масса всех деталей не должна превышать M.

Формат входных данных

В первой строке входного файла записано два целых положительных числа N и M — количество деталей, необходимых для строительства ракеты и максимальная допустимая суммарная масса деталей ($1 \leqslant M \leqslant 10^9$).

Далее следует N блоков строк. В i-м блоке дано описание i-й детали. Описание i-го блока начинается со строки, содержащей целое положительное число K_i. Далее следуют K_i строк, в j-й из которых даны целые числа m_{ij} и c_{ij} — масса и стоимость i-й детали, выполненной целиком из j-го предложенного для неё базового материала ($1 \leqslant m_{ij}, c_{ij} \leqslant 10^9$).

Гарантируется, что $\sum\limits_{i=1}^{N} K_i \leqslant 10^5$. Также гарантируется существование решения.

Формат выходных данных

В первую строку выходного файла требуется вывести минимальную возможную стоимость ракеты как вещественное число. В i-ю из следующих строк требуется вывести описание материала, из которого следует изготовить i-ю деталь ($1 \leqslant i \leqslant N$).

Описание материала для i-й детали должно выглядеть следующим образом.

Если i-ю деталь нужно изготовить из базового материала, то строка описания имеет формат “1 A”, где целое число A — номер базового материала ($1 \leqslant A \leqslant K_i$).

Если i-ю деталь нужно делать из сплава двух базовых материалов, то строка описания имеет формат “2 A B X Y”, где целые числа A и B — номера базовых материалов ($1 \leqslant A, B \leqslant K_i$), а целые числа X и Y — числитель и знаменатель пропорции сплава ($0 < X < Y \leqslant 10^9$, $\alpha = \frac{X}{Y}$).

Требуется, чтобы пропорции всех сплавов в вашем ответе имели одинаковый знаменатель Y. Гарантируется, что решение, удовлетворяющее этому требованию, существует. Относительная или абсолютная погрешность ответа (минимальной стоимости ракеты) не должна превышать 10^{-12}.

Примеры

input.txt	output.txt
2 11 3 4 3 6 3 7 8 4 9 5 10 3 6 5 7 6	7.5000000000 1 1 2 3 2 3 4
2 4 2 1 2 2 3 2 3 2 2 5	4.0 1 1 1 1

Задача 8. Капуста

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

«Голодные» дети Поволжья, как известно, очень любят кислую капусту. Но вот каждый из них согласен есть капусту только одного сорта. Правда, сорт этот может быть для каждого из детей совершенно любым. Разные «голодающие» могут есть как разные сорта капусты, так и одинаковые. Чтобы никого не обижать, порции у детей должны быть одинаковыми. При этом заведующий Альхен хочет сделать порции как можно больше.

У Альхена изначально есть некоторое количество кислой капусты каждого сорта, также имеется некоторая сумма денег. На эти деньги он может докупить какое-то количество этой самой капусты, своё для каждого сорта. Цена каждого сорта капусты известна. А вот продавать уже имеющуюся капусту нельзя.

Какими нужно сделать порции для своих подопечных — заведующий Альхен не знает. Помогите ему в этом разобраться.

Формат входных данных

В первой строке входного файла задано целое число T — количество тестовых случаев ($1 \leqslant T \leqslant 100$). Далее следует T блоков.

В первой строке блока записаны три целых числа: N — количество сортов кислой капусты ($1 \leqslant N \leqslant 10^5$), M — количество детей Поволжья ($1 \leqslant M \leqslant 10^5$), S — сумма денег для покупки дополнительной кислой капусты ($1 \leqslant S \leqslant 10^9$).

Во второй строке блока содержится M целых чисел T_i, где T_i — номер сорта капусты, который ест i-ое дитя Поволжья ($1 \leqslant T_i \leqslant N$).

В каждой из следующих N строк содержится по два целых числа: A_i — имеющееся изначально количество капусты i-ого сорта, в килограммах ($0 \leqslant A_i \leqslant 10^4$), и C_i — цена одного килограмма капусты данного сорта ($1 \leqslant C_i \leqslant 10^4$).

Сумма M по всем тестовым случаям не превосходит 10^5, а также сумма N по всем тестовым случаям не превосходит 10^5.

Формат выходных данных

В выходной файл выведите T строк, в i-й строке — ответ на i-й тестовый случай. Ответом на тест является максимально возможный размер порции в килограммах.

Абсолютная или относительная погрешность каждого ответа не должна превышать 10^{-9}.

Примеры

input.txt	output.txt
1 3 7 37 3 3 2 3 1 2 3 2 2 1 6 3 1	2.777777777778
2 2 3 17 1 2 1 50 3 0 2 1 2 1 1 1 1 1	8.5 1

Задача 9. Отрезки

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

Иннокентий по долгу службы пишет разные алгоритмы для обработки треугольных сеток. В последней задаче, над которой он бьётся, нужно уметь быстро строить сечения триангуляции параллельными плоскостями. Иннокентий смекнул, что ключевая сложность задачи легко сводится к следующей одномерной задаче. Дан набор отрезков на координатной прямой, и нужно обрабатывать запросы: по заданной точке определить все отрезки, в которые эта точка попадает. Помогите Иннокентию решить эту задачу.

Чтобы отразить характер реальных триангуляций, входные данные в этой задаче генерируются случайным образом. Всего дано N отрезков разной длины, причём коротких отрезков много, а длинных — мало. Каждый отрезок задан своим центром C и радиусом R, и покрывает интервал от $C - R$ до $C + R$ включительно. Центр каждого отрезка C сгенерирован с равномерным распределением в диапазоне от 0 до N. Радиус R генерируется при помощи формулы:

$$R = \min\left(\frac{1}{\mathrm{rnd}}, N\right)$$

где rnd возвращает случайное число от 0 до 1 с равномерным распределением.

Кроме того, дано N точек-запросов. Каждый запрос задан координатой в диапазоне от 0 до N, которая также генерируется случайным образом с равномерным распределением. Для каждого такого запроса нужно определить все отрезки, в которые входит точка, и вывести их количество.

В реальной задаче Иннокентия ответ нужно вернуть сразу после получения точки-запроса. Чтобы смоделировать подобную ситуацию, сделаем задачу более динамической.

Во-первых, будем считать, что изначально на прямой нет никаких отрезков. Во входных данных перечислено $2N$ запросов, из которых половина — это запросы на добавление отрезка, а половина — это точки-запросы. Нужно обрабатывать эти запросы в порядке перечисления, и при обработке точки-запроса рассматривать лишь те отрезки, которые уже добавлены на прямую к этому моменту.

Во-вторых, обработка каждой точки-запроса изменяет все последующие входные данные следующим образом. Допустим, все отрезки, в которые попадает точка-запрос P, имеют номера $k_0, k_1, k_2, \ldots, k_{t-1}$. Пусть $S = (k_0 + k_1 + \ldots + k_{t-1})$ — сумма этих номеров. Тогда нужно изменить координату всех последующих точек-запросов следующим образом: если X — текущая координата точки, то новая координата равна $(X + S) \bmod N$. Иными словами, нужно циклически сдвинуть их на S, приводя все значения к диапазону от 0 до N, исключая значение N. Аналогичным образом нужно изменить центры всех ещё не добавленных на прямую отрезков.

Отрезки нумеруются подряд номерами от 0 до $N - 1$ включительно.

Формат входных данных

В первой строке входного файла записано целое число N — количество отрезков и количество точек ($3 \leqslant N \leqslant 10^5$). В остальных $2N$ строках описаны запросы в порядке их выполнения: всего ровно N запросов на добавление отрезка и N точек-запросов.

Точка-запрос начинается с цифры 1, после которой через пробел записана координата X точки как вещественное число ($0 \leqslant X < N$). Запрос на добавление отрезка начинается с цифры 0, после которой через пробел записано два вещественных числа: C — центр отрезка и R — радиус отрезка ($0 \leqslant C < N$, $1 \leqslant R \leqslant N$).

Все вещественные числа генерируются согласно указанным в условии правилам, а затем округляются до некоторого количества знаков после десятичной точки. Это количество знаков может быть в диапазоне от одного до шести.

Формат выходных данных

В выходной файл необходимо вывести N ответов на точки-запросы в порядке их обработки, по одному ответу в строке. Для каждой точки-запроса нужно вывести только количество найденных отрезков t. Сами же найденные отрезки, точнее их номера, нужно использовать для корректирования оставшихся запросов, как описано в условии.

Пример

input.txt	output.txt
5 0 3.1 5.0 1 2.2 0 0.8 1.3 0 0.5 1.3 0 3.6 1.2 1 3.8 1 2.0 1 3.8 0 3.6 1.6 1 2.5	1 2 3 2 4

Пояснение к примеру

Сначала добавляется отрезок с концами $[-1.9; 8.1]$. Идущая сразу после этого точка-запрос в этом отрезке лежит, поэтому первый ответ равен 1. Поскольку номер отрезка равен нулю, никакого сдвига к будущим запросам делать не нужно.

Далее добавляется три отрезка ровно с теми центрами и радиусами, какие записаны в файле. Затем проверяется точка-запрос $x = 3.8$, для которой находятся отрезки с номерами 0 и 3. Теперь ко всем последующим данным нужно применять сдвиг на 3.

Следующая точка-запрос задаётся в файле равной 2.0, однако после выполнения циклического сдвига на 3 она попадает в 0. Точку $x = 0$ содержат отрезки с номерами 0, 1 и 2. Нужно добавить циклический сдвиг ещё на 3, что с учётом уже имеющегося сдвига даёт в сумме циклический сдвиг на 1.

Следующая точка-запрос после сдвига равна $x = 4.8$, и она лежит на отрезках с номерами 0 и 3. Обратите внимание, что точка попадает точно на правую границу отрезка 3. После этого запроса суммарный циклический сдвиг равен 4.

Далее добавляется отрезок с центром в $c = 2.6$ после учёта циклического сдвига. Наконец, последняя точка-запрос в $x = 1.5$ попадает в отрезки 0, 1, 2 и 4.

Задача 10. Civilization

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

Действия пошаговой компьютерной стратегии Civilization V происходят на гексагональном поле — двумерной сетке, состоящей из клеток в форме правильных одинаковых шестиугольников.

Для каждой клетки поля заданы рельеф и покрытие. Рельеф может быть равниной, холмом или горой, а покрытие — лугом, лесом или болотом. По границам клеток могут протекать реки.

По клеткам поля могут передвигаться юниты. Юнит занимает ровно одну клетку поля. Игра Civilization является пошаговой и на каждом ходу юниту даётся определённое количество очков движения (ОД). По умолчанию, на переход с любой клетки поля на соседнюю клетку нужно потратить 1 ОД. Однако, в зависимости от клетки, с которой юнит уходит, и клетки, на которую он переходит, стоимость передвижения может увеличиться:

- при переходе с клетки, являющейся равниной, на клетку, являющейся холмом, стоимость передвижения увеличивается на 1 ОД;
- при переходе в клетку, являющуюся лесом или болотом, стоимость передвижения увеличивается на 1 ОД;
- при переходе через реку тратятся все ОД текущего хода.

Клетки с горами являются непроходимыми — на них переходить нельзя.

Пока значение ОД юнита строго положительно, он в течение хода может перемещаться между клетками. Если стоимость очередного перемещения больше имеющегося количества ОД юнита, то ему разрешено сделать это перемещение. Как только количество ОД становится равным нулю или отрицательным, то текущий ход для юнита заканчивается. В этом случае продолжить передвижение юнит сможет только на следующем ходу, когда количество ОД восстановится на начальное значение.

Требуется найти кратчайший путь от стартовой клетки до конечной за минимальное количество ходов. Учитываются как завершённые ходы, на которых были потрачены все ОД, так и последний ход, на котором может остаться положительное количество ОД. При этом оставшиеся ОД на последнем ходу максимизировать **не** требуется.

Формат входных данных

В первой строке входного файла дано два целых числа W и H — ширина и высота поля ($1 \leqslant W, H \leqslant 100$). Юниты не могут выходить за пределы игрового поля. В игровое поле входят клетки с координатами (x, y) при $0 \leqslant x < W$ и $0 \leqslant y < H$ и никакие другие.

В пояснении к примеру приложена иллюстрация, на которой изображено гексагональное поле и нумерация клеток. Поле состоит из гексагональных клеток, которые образуют H рядов. Каждый ряд состоит из W клеток, при этом все клетки ориентированы так, что сверху и снизу расположены вершины шестиугольника, а слева и справа рёбра, которыми клетка соединена с соседними клетками в ряду. Каждый следующий ряд смещён относительно предыдущего на половину клетки. Ось Ox направлена вдоль верхнего ряда клеток слева направо. Ось Oy направлена под углом 120 градусов к оси Ox.

В следующих H строках содержится описание клеток игрового поля. Каждая строка содержит описание W клеток, записанных через пробел. Описание клетки состоит из двух символов, первый из которых описывает рельеф и может иметь одно из следующих значений:

- ‘`p`’ — равнина,
- ‘`h`’ — холм,
- ‘`m`’ — гора,

а второй описывает покрытие:

- ‘`g`’ — луг,
- ‘`f`’ — лес,
- ‘`m`’ — болото.

Следующая строка содержит одно целое число R — количество рек на поле ($0 \leqslant R \leqslant 10^4$). Следующие R строк содержат по 4 целых числа x_A, y_A, x_B и y_B — координаты двух соседних клеток A и B ($0 \leqslant x_A, x_B < W$, $0 \leqslant y_A, y_B < H$), между которыми протекает река по их общей стороне.

В предпоследней строке записано целое число M ($1 \leqslant M \leqslant 10^5$) — количество ОД, которые имеются у юнита на текущем, первом ходу. До этого же значения будут восстанавливаться ОД юнита на каждом новом ходу.

В последней строке записано 4 целых числа x_s, y_s, x_f и y_f ($0 \leqslant x_s, x_f < W$, $0 \leqslant y_s, y_f < H$) — координаты начальной (x_s, y_s) и конечной (x_f, y_f) клетки. Гарантируется, что данные координаты не лежат на клетках с горами и что стартовая клетка не совпадает с конечной.

Формат выходных данных

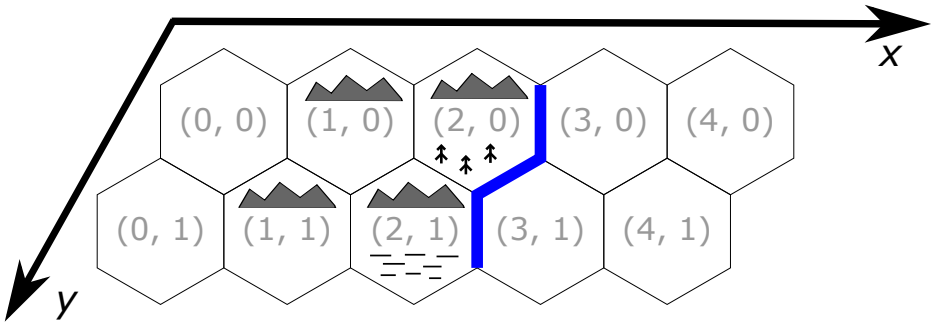
В первую строку выходного файла требуется вывести фразу `Come this way`, если существует путь между стартовой и конечной клетками и фразу `They shall not pass` в противном случае.

Если путь существует, то в следующую строку нужно записать минимальное количество ходов, за которое возможно дойти до требуемой клетки.

Следующая строка должна содержать N — количество пройденных на данном пути клеток, включая стартовую и конечную. А далее в N строках должны быть записаны координаты клеток в порядке от стартовой к требуемой. Если существует несколько способов доехать до конечной клетки с минимальным количеством ходов, то можно вывести любой из них.

Примеры

Поле из третьего примера выглядит следующим образом:



(Таблица с примерами приведена на следующей странице)

input.txt	output.txt
3 1 pg pm hf 0 7 0 0 2 0	Come this way 1 3 0 0 1 0 2 0
3 1 pf mg hm 0 7 0 0 2 0	They shall not pass
5 2 pg hg hf pg pg pg hg hm pg pg 3 2 0 3 0 2 0 3 1 2 1 3 1 4 0 1 4 0	Come this way 3 6 0 1 0 0 1 0 2 0 3 0 4 0

Пояснения к примерам

В первом примере единственный путь от стартовой клетки до конечной требует 5 ОД:

- Переход из $(0, 0)$ в $(1, 0)$ требует 2 ОД из-за болота в клетке $(1, 0)$.
- Переход из $(1, 0)$ в $(2, 0)$ требует 3 ОД из-за леса и холма в конечной клетке.

Таким образом, перемещение может завершиться на первом же ходу.

Во втором примере из-за непроходимой горы в конечную клетку попасть невозможно.

Поле из третьего примера нарисовано на предыдущей странице условия. Один из вариантов перемещения до конечной клетки:

- Переход из $(0, 1)$ в $(0, 0)$ требует 1 ОД (базовая стоимость).
- Переход из $(0, 0)$ в $(1, 0)$ требует 2 ОД из-за холма в клетке $(1, 0)$.
- Переход из $(1, 0)$ в $(2, 0)$ требует 2 ОД из-за леса в клетке $(2, 0)$. Здесь нет увеличения стоимости за переход в клетку с холмом, т.к. клетка $(1, 0)$ так же является холмом. На этом первый ход заканчивается.
- Из-за отрицательного количества ОД, чтобы продолжить движение юнит должен начать второй. Переход из $(2, 0)$ в $(3, 0)$ заканчивает второй ход из-за перехода через реку.
- Далее начинается третий ход. Переход из $(3, 1)$ в $(4, 0)$ требует 1 ОД (базовая стоимость).

Задача 11. Эклиптика

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

Беспилотный космический летательный аппарат «Лучик» находится в северном полушарии на широте ϕ градусов и готов стартовать с поверхности планеты в любой момент. Планируется вывод «Лучика» на солнечную орбиту, поэтому аппарат нужно запустить в такой момент времени, когда его стартовая площадка наиболее близка к плоскости эклиптики.

Планета вращается вокруг своего солнца по круговой орбите. Плоскостью эклиптики называют ту плоскость, в которой лежит эта орбита.

Планета имеет форму шара и вращается вокруг своей оси. Угол между осью вращения планеты и нормалью к плоскости эклиптики составляет α градусов. Планета выполняет один оборот вокруг своей оси ровно за 24 часа.

Командный пункт находится на экваторе, на той же долготе, что и «Лучик». Он пересекает плоскость эклиптики каждые 12 часов. В момент времени *HH:MM:SS* командный пункт сообщил о том, что он пересёк плоскость эклиптики. После этого он и «Лучик» стали **отделены друг от друга** плоскостью эклиптики. Требуется найти первый момент времени после этого сообщения, когда «Лучик» будет наиболее близок к плоскости эклиптики.

Формат входных данных

В первой строке входного файла дано целое число T — количество тестов ($1 \leqslant T \leqslant 5 \cdot 10^4$). Далее в каждой из следующих T **пар** строк идет описание очередного теста.

В первой строке описания теста дано два вещественных числа α и ϕ. Число α — это угол между осью вращения планеты и перпендикуляром к плоскости эклиптики, заданный в градусах; число ϕ — это широта, на которой находится летательный аппарат, заданная в градусах ($1.0 \leqslant \alpha, \phi \leqslant 89.0$). Оба вещественных числа заданы с не более чем пятью знаками после десятичной точки.

Во второй строке дано время, когда пришло сообщение из командного пункта. Время задаётся в 24-часовом формате *HH:MM:SS*, где *HH* — это количество часов от 0 до 23, *MM* — количество минут от 0 до 59, и *SS* — количество секунд от 0 до 59. Каждое из трёх чисел записано ровно двумя цифрами, возможно с использованием ведущего нуля.

Формат выходных данных

В выходной файл требуется вывести ответы для тестов в порядке их задания во входных данных, по одному ответу в строке.

Ответ на тест — это ближайший момент времени, когда аппарат может вылететь. Время требуется вывести в том же формате, что и время сообщения командного пункта. Выведенное время должно отличаться от точного ответа не более чем на одну секунду.

Примеры

input.txt	output.txt
2 23.44 15.0 12:00:00 20.0 16.11 23:00:00	14:32:41 02:30:04