Задача 1. Лыжные гонки

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

В городе N наступила зима, и уже скоро состоятся первые соревнования по лыжным гонкам. В этом году регистрация проходила через интернет — каждый участник ввёл свои данные и выбрал себе номер из номеров, ещё не выбранных другими лыжниками. Из-за большого числа зарегистрировавшихся, организаторы решили проводить несколько стартов на гонку.

Чтобы определить счастливчиков, которые выходят на трассу в первом старте, они придумали простое правило — лыжник с номером X выходит на старт, если не существует другого лыжника, номер которого делится нацело на X.

Помогите организаторам написать программу, которая определит номера тех, кто должен стартовать первыми.

Формат входных данных

В первой строке входного файла записано целое число K — количество зарегистрированных участников ($1 \leqslant K \leqslant 10^5$).

Во второй строке через пробел заданы K целых чисел A_i — номера, которые выбрали участники при регистрации ($1 \leqslant A_i \leqslant 10^7$). Все числа A_i различны.

Формат выходных данных

В выходной файл необходимо вывести единственную строку, которая содержит в порядке возрастания номера всех участников, стартующих первыми. Числа должны разделяться символом пробела.

Пример

input.txt	output.txt
3 4 8 12	8 12

Задача 2. Стулья

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

Остап и Киса попали на выставку-продажу стульев. Перед ними стоит две задачи: как можно быстрее покинуть эту выставку, а то конкурирующая организация в лице отца Федора не дремлет, ну и при этом собрать все имеющиеся на выставке стулья.

План павильона выставки представляет собой прямоугольную таблицу из N строк и M столбцов. В некоторых клетках этой таблицы отмечены стулья, которые надо собрать. Первоначально концессионеры находятся в верхнем левом углу таблицы — клетке с координатами $(1, 1)$, а выход находится в правом нижнем углу — клетке с координатами (N, M). За один ход концессионеры перемещаются из текущей клетки в любую соседнюю, смежную по стороне, клетку. Проходя через клетку со стулом, они забирают этот стул с собой.

Требуется найти для концессионеров путь минимальной длины из начальной клетки в конечную. Среди таких кратчайших путей нужно выбрать путь, который проходит через все клетки со стульями, либо выяснить, что такого пути не существует.

Формат входных данных

В первой строке входного файла записано три целых числа N, M, K — соответственно размеры таблицы и количество стульев ($2 \leqslant N, M \leqslant 100$, $0 \leqslant K \leqslant 1000$).

В следующих K строках приведено по два целых числа в каждой: X_i — номер строки таблицы, в которой находится i-ый стул, и Y_i — номер столбца таблицы, в котором находится i-ый стул ($1 \leqslant X_i \leqslant N$, $1 \leqslant Y_i \leqslant M$). Ни в одной клетке не может находиться более одного стула.

Формат выходных данных

Если собрать все стулья ни на каком кратчайшем пути невозможно, то в выходной файл нужно выдать одно слово «`Impossible`» (без кавычек).

Если же такой маршрут существует, то нужно вывести его в виде строки из последовательности ходов, каждый ход кодируется одним символом в соответствии со следующими обозначениями:

- `D` — ход вниз;
- `R` — ход вправо.

Если возможных ответов несколько, то требуется вывести **лексикографически наименьший** из них.

Пример

input.txt	output.txt
3 3 2 1 2 3 3	RDDR
3 3 2 1 2 2 1	Impossible

Задача 3. Треугольник

Имя входного файла:	`stdin`
Имя выходного файла:	`stdout`
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

После коварного исчезновения Корейко с заветным чемоданчиком перед Остапом встала практичеки неразрешимая задача — найти, хотя бы приблизительно, район, где он может находиться.

Из некоторых соображений известно, что область поисков представляет собой треугольник на плоскости, координаты вершин которого — целые числа. Для определения координат вершин великий комбинатор шлет наркомфину телеграмму странного содержания (a, b, c), и в ответ получает не менее странный ответ (p, q). На самом деле, три числа из телеграммы Остапа — это коэффициенты уравнения прямой $a \cdot x + b \cdot y = c$, а полученный им ответ — пропорция, в которой эта прямая разрезает треугольник на части (по площади).

Известно, что координаты всех вершин треугольника — целые числа, по модулю не превышающие 1 000. Также известно, что треугольник невырожденный, и все углы треугольника составляют не менее 5 градусов.

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.

Ваша программа должна отправлять запросы, на которые интерактор будет давать ответы. На каждый запрос, являющийся коэффициентами уравнения прямой, вам будет сообщено, в какой пропорции данная прямая разрезает треугольник.

После вывода ответа на задачу решение участника должно завершить работу.

Требуется не более чем за 1 000 запросов определить координаты вершин треугольника. Если вы не сможете определить координаты вершин треугольника или выведите их неправильно, то вы получите вердикт Wrong Answer.

Формат выходных данных

Формат запроса Остапа: "`?` a b c", где a, b и c — коэффициенты в уравнении прямой, записанные как вещественные числа с не более чем 15 знаками после десятичной точки ($|a|, |b| \leqslant 2 \cdot 10^3$, $|c| \leqslant 4 \cdot 10^6$, $\frac{1}{4 \cdot 10^6} \leqslant a^2 + b^2 \leqslant 4 \cdot 10^6$).

Формат ответа задачи: "`!` x_1 y_1 x_2 y_2 x_3 y_3", где x_i, y_i — целочисленные координаты вершины искомого треугольника. Три вершины треугольника можно записывать в любом порядке.

Убедитесь, что каждый запрос, включая ответ на задачу, заканчивается переводом строки, и что вы очищаете буфер потока вывода (команда `flush` языка). В противном случае решение может получить вердикт Deadlock, т.е. решение не уложилось в отведённое астрономическое время.

Формат входных данных

Формат ответа наркомфина: "p q", где вещественные числа p и q определяют пропорцию, в которой заданная в запросе прямая разрезает треугольник ($0 \leqslant p, q \leqslant 1$, $p + q = 1$). Число p равно доле площади треугольника, для которой выполнено неравенство $a \cdot x + b \cdot y \leqslant c$. Числа p и q выводятся с 15 знаками после десятичной точки.

Пример

stdin
? 0 1 0 ? 0 2 1 ? 0 1 0 ? 1 0 1 ? 0 3 1 ? 2 0 1 ? 2 0 3 ! 0 0 2 0 1 1
stdout
0.000000000000000 1.000000000000000 0.750000000000000 0.250000000000000 0.000000000000000 1.000000000000000 0.500000000000000 0.500000000000000 0.555555555555555 0.444444444444445 0.125000000000000 0.875000000000000 0.875000000000000 0.125000000000000

Задача 4. Провода

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

Сотрудники одной очень большой и очень секретной конторы работают в большой прямоугольной комнате. При этом N сотрудников первого отдела располагаются у окон вдоль одной стены и столько же сотрудников второго отдела — вдоль противоположной стены. Однажды поступило важное и столь же секретное предписание — соединить компьютеры сотрудников этих двух отделов между собой так, чтобы компьютер каждого сотрудника первого отдела был бы связан с соответствующим ему компьютером сотрудника второго отдела отдельным проводом.

Было составлено техническое задание, которое включало план комнаты. На этом плане комната представляется прямоугольником размера $A \times B$: левая и правая стороны имеют длину A, а верхняя и нижняя — длину B. На левой стороне прямоугольника расположено N контактов-входов, которые соответствуют расположению компьютеров сотрудников первого отдела, а на правой — N контактов-выходов для компьютеров сотрудников второго отдела. По заданному взаимно-однозначному соответствию входов и выходов требуется соединить проводом каждый контакт-вход с соответствующим ему контактом-выходом.

При прокладке проводов нужно придерживаться следующих правил:

1. Провода не могут разветвляться, каждый провод начинается на контакте-входе и заканчивается на контакте-выходе.
2. Каждый провод может идти как внутри прямоугольника, так и снаружи (все контакты доступны с обеих сторон прямоугольника).
3. Провод **не** может пересекать границу прямоугольника.
4. Провода **не** могут пересекаться друг с другом, то есть один провод не может проходить над другим.

Требуется найти, какая минимальная суммарная длина проводов потребуется, чтобы соединить контакты требуемым образом, или определить, что это невозможно. Можно считать, что толщина проводов пренебрежимо мала: можно прокладывать провода сколь угодно близко друг к другу.

Напишите программу, которая вычислит минимальную суммарную длину проводов.

Формат входных данных

В первой строке входного файла записано три целых числа: A — длина левой и правой сторон прямоугольника, B — длина верхней и нижней сторон прямоугольника и N — количество контактов-входов и контактов-выходов ($1 \leqslant A, B \leqslant 10^8$, $1 \leqslant N \leqslant 10^5$).

Во второй строке описаны положения всех N контактов-входов. Для каждого k-ого контакта-входа записано целое число L_k — расстояние от левого нижнего угла прямоугольника до контакта ($0 \leqslant L_k \leqslant A$). Гарантируется, что все L_k различны.

В третьей строке описаны положения N контактов-выходов. Для каждого k-ого контакта-выхода записано целое число R_k — расстояние от правого нижнего угла прямоугольника до контакта ($0 \leqslant R_k \leqslant A$). Гарантируется, что все R_k различны.

Требуется соединить каждый k-ый в порядке описания контакт-вход с k-ым в порядке описания контактом-выходом.

Формат выходных данных

В выходной файл необходимо вывести одно вещественное число — минимальную суммар-

ную длину всех проводов при корректном способе соединения. Относительная или абсолютная погрешность ответа не должна превышать 10^{-9}.

Если корректно проложить провода требуемым образов невозможно, необходимо вывести единственное число -1.

Пример

input.txt	output.txt
6 7 3 1 3 5 5 1 3	27.560219778561036542194604983054

Комментарий

Строго с математической точки зрения, минимальная суммарная длина проводов может не достигаться ни на каком корректном плане соединения из-за того, что толщина проводов бесконечно мала. В таком случае требуется найти точную нижнюю грань (infimum) среди возможных суммарных длин.

Задача 5. Голосование

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	2 секунды
	5 секунд (для Java)
Ограничение по памяти:	256 мегабайт

Политическая обстановка в Берляндии изменилась. Благодаря победе на последних президентских выборах кандидата от оппозиции, многоуровневую систему выборов наконец-то отменили. Теперь президента в Берляндии определяют путём единого всеобщего голосования. Однако консерваторы активно внушают населению мысль, что с новой системой выборов повлиять на результат стало ещё проще, чем раньше. Чтобы опровергнуть эти утверждения, президент поручил оценить стоимость изменения результатов голосования путём подкупа избирателей.

Всего в Берляндии имеется N избирателей и K кандидатов. Каждый из избирателей может либо отдать свой голос за одного конкретного кандидата, либо воздержаться, например, не придя на выборы. После того, как все избиратели так или иначе проголосовали, подсчитывается количество голосов, отданных за каждого кандидата. В выборах побеждает кандидат, набравший строго больше голосов, чем любой другой кандидат. Если такого кандидата нет, то выборы признаются несостоявшимися.

Вам предлагается написать программу на основе следующих положений. Для каждого отдельного избирателя известно, за кого он планирует голосовать. Разрешается изменить его предпочтение на любое другое, затратив на это некоторую сумму денег. Необходимо сделать так, чтобы заданный кандидат победил на выборах. Требуется минимизировать сумму денег, которую необходимо для этого потратить.

Формат входных данных

В первой строке входного файла дано три целых числа: N — количество избирателей в Берляндии, K — количество кандидатов на выборах, T — номер кандидата, победу которого необходимо обеспечить ($1 \leqslant N \leqslant 200$, $2 \leqslant K \leqslant 10$, $1 \leqslant T \leqslant K$). Как все избиратели, так и все кандидаты пронумерованы последовательными целыми числами, начиная с единицы.

Далее записана матрица стоимостей из N строк и $K + 1$ столбца. Элемент $C_{i,j}$ матрицы определяет, сколько денег нужно потратить, чтобы i-ый избиратель проголосовал за j-ого кандидата ($1 \leqslant i \leqslant N$, $1 \leqslant j \leqslant K$). Последний в строке элемент $C_{i,K+1}$ определяет, сколько нужно потратить денег, чтобы i-ый избиратель воздержался и не пришел на выборы.

Гарантируется, что все стоимости $C_{i,j}$ целые и удовлетворяют неравенству $0 \leqslant C_{i,j} \leqslant 10^9$. Кроме того, для каждого i ровно одно из чисел $C_{i,1}, C_{i,2}, \ldots, C_{i,K+1}$ равно нулю: этот ноль означает, что избиратель изначально планирует голосовать соответствующим образом.

Формат выходных данных

В первую строку выходного файла требуется вывести одно целое число — минимально возможную сумму денег, которую нужно потратить на изменение предпочтений избирателей.

Во вторую строку нужно вывести N целых чисел. i-ое из этих чисел V_i указывает, что i-ый избиратель должен голосовать за V_i-ого кандидата ($1 \leqslant V_i \leqslant K + 1$). Особое значение $V_i = K + 1$ означает, что i-ый избиратель должен воздержаться от голосования.

Если оптимальных решений несколько, разрешается вывести любое из них.

Пример

input.txt	output.txt
5 2 2 0 9 2 0 10 1 7 8 0 0 2 1 3 0 1	3 1 3 3 2 2

Пояснение к примеру

В примере предлагается изменить предпочтение второго избирателя, чтобы он не пришёл на выборы (обойдется в одну денежную единицу), а также предпочтение четвёртого избирателя, чтобы он проголосовал за желаемого кандидата (необходимо заплатить ещё две денежные единицы). В результате за первого кандидата проголосует только первый избиратель, а за второго проголосуют четвёртый и пятый избиратели.

Задача 6. Конечный автомат

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

Сегодня Вася узнал, что такое «детерминированный конечный автомат» (ДКА), и теперь он жаждет всем об этом рассказать.

Оказывается, в ДКА имеется N состояний. В любой момент в ходе своей работы автомат находится в одном из них. На вход автомату подаётся произвольная строка, а в конце работы он сообщает, является ли она *допустимой*.

Процесс работы устроен следующим образом:

1. В начале работы ДКА находится в *начальном* состоянии, в автомате оно обязательно отмечено.
2. Автомат *считывает* по одному все символы в строке в порядке слева направо. При прочтении каждого символа он может перейти в другое состояние (подробности ниже).
3. Когда вся строка прочитана, автомат определяет ответ, исходя из того, в каком состоянии он оказался.

Для каждого состояния u автомата и каждого возможного символа c в автомате указано, в каком состоянии он будет находиться после считывания символа c, если до этого он находился в состоянии u. Это новое состояние может как совпадать с u, так и отличаться от него. Кроме того, для каждого состояния в автомате указано, какой ответ нужно выдать (допустимая строка или нет), если автомат оказался в нём по завершении работы.

На первом семинаре по этой теме Вася строил самые разные ДКА, а «на дом» Васе задали следующую задачу. Требуется построить ДКА, на вход которому подаётся целое неотрицательное число, записанное в B-ичной системе счисления, и который определяет как допустимые такие и только такие числа, которые нацело делятся на заданный модуль M.

Для простоты Вася предположил, что подаваемое на вход число:

- начинается со старших разрядов (они записаны слева);
- может иметь ведущие нули;
- может быть пустым: в таком случае оно равно нулю и точно делится на M.

Вася, по природе своей, — перфекционист, и он хочет научиться строить ДКА, удовлетворяющий условию задачи, с **минимальным** возможным количеством состояний. За помощью он обратился к вам.

Формат входных данных

В единственной строке входного файла записано два целых числа B и M: B — основание позиционной системы исчисления, в которой задаётся входное число и M — значение модуля, на который должны делиться все допустимые и только допустимые числа ($2 \leqslant B \leqslant 16, 2 \leqslant M \leqslant 10^5$).

Формат выходных данных

В выходной файл требуется вывести описание минимального ДКА, являющегося решением задачи.

В первой строке должно быть записано два целых числа: N — количество состояний в автомате ($N \geqslant 2$) и S — номер состояния, являющегося начальным ($0 \leqslant S < N$). Все состояния автомата пронумерованы подряд, начиная с нуля.

Во второй строке через пробел должно быть записано N символов. k-ый из этих символов определяет, какой ответ должен возвращать автомат, если он оказался в k-ом состоянии по

завершении работы ($0 \leqslant k < N$). Символ равен ‘G’, если строка должна быть признана допустимой, и ‘B’ — в противном случае.

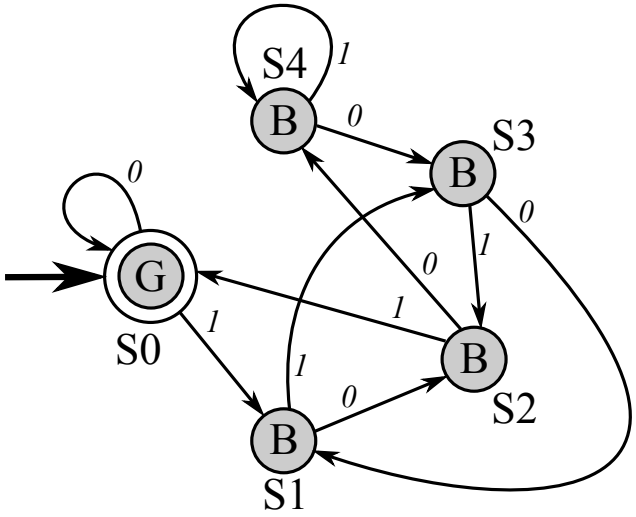
Далее должно быть записано N строк, по B целых чисел в каждой. k-ое число в i-ой из этих строк содержит номер состояния, в котором должен оказаться автомат после считывания цифры k, если до её считывания он находился в состоянии i ($0 \leqslant i < N$, $0 \leqslant k < B$). Это число может быть любым целым в пределах от 0 до $N - 1$ включительно.

Пример

input.txt	output.txt
2 5	5 0 G B B B B 0 1 2 3 4 0 1 2 3 4

Пояснение к примеру

Ниже изображен ДКА, который показан в примере. Рядом с каждым состоянием находится буква “S” и его номер. В начальное состояние ведёт жирная стрелка. Каждая обычная стрелка показывает, в какое состояние должен перейти ДКА, после считывания цифры, записанной около неё.



Задача 7. Управление сценой

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	3 секунды
	4 секунды (для Java)
Ограничение по памяти:	256 мегабайт

Весь программистский мир захлестнула мода на веб-приложения. Теперь заказчики хотят запускать в браузере всё подряд, несмотря на логику и здравый смысл. Увы, веб не обошёл стороной даже Стёпу, работа которого целиком и полностью лежит в области систем автоматизированного проектирования (САПР). Теперь он тоже вынужден писать код на языке javascript, в котором можно взять обещание о выполнении работы, но нельзя дождаться её завершения.

Особенно грустно обстоят дела с производительностью в мире javascript, в том числе в области компьютерной графики. В то время как весь мир с упоением играет в новый Doom, включив многопоточную отрисовку на низкоуровневом графическом API "Vulkan", Стёпа вынужден работать с three.js — обёрткой над API "WebGL", который является утяжелённым урезанным вариантом OpenGL-я десятилетней давности. И неудивительно, что, в результате, рисование модели с несколькими тысячами элементов тормозит даже на Core i7 с любой видеокартой.

Специфика графики в области САПР заключается в том, что состояние объектов на сцене изменяется очень редко. Стёпа реализовал собственный отрисовщик, который может улучшать производительность за счёт объединения объектов и их отрисовки «блоками». Происходящие на сцене изменения нужно вносить в отрисовщик, вызывая методы «добавить объект», «удалить объект», «обновить объект». К сожалению, коллеги Степана не хотят вызывать эти методы вручную: они привыкли к богатому интерфейсу three.js, да и уже написано много кода, использующего three.js напрямую. Так что Стёпа был вынужден написать прослойку, которая автоматически отслеживает изменения на three.js-сцене и вносит их в отрисовщик.

В three.js сцена представляет собой корневое дерево, каждая вершина которого является объектом, который надо рисовать. У каждого объекта имеется уникальный идентификатор (ID) — целое положительное число, отличное от идентификаторов других объектов. У каждого объекта есть набор детей, порядок которых не имеет значения. Также в каждом объекте есть набор свойств, которые влияют на внешний вид всех объектов в поддереве (например, трансформация координат). Пользователи three.js могут выполнять над объектами следующие операции:

- `A.add(B)`: добавить объект B в качестве нового ребёнка к объекту A.
- `A.remove(B)`: убрать объект B из набора детей объекта A.
- `A.modify()`: изменить какие-либо свойства в объекте A.

Следует заметить, что операция `remove` не удаляет объекты, а лишь отсоединяет поддерево от дерева сцены (физически объекты в javascript программист удалить вообще не может — для этого есть сборщик мусора). Отсоединённые объекты пользователь может в будущем присоединить обратно к сцене при помощи операции `add`. Таким образом, в общем случае в любой момент времени в программе может быть несколько отдельных деревьев объектов: одно дерево сцены, которое необходимо рисовать, и произвольное количество "отсоединённых" поддеревьев, которые рисовать не нужно.

Пользователи гарантируют корректность выполнения операций над объектами:

1. При выполнении `A.add(B)` гарантируется, что объект B **не** имеет родителя, и что объект A **не** является потомком B (и не совпадает с ним).

2. При выполнении `A.remove(B)` гарантируется, что объект B содержится в списке детей объекта A.
3. Корневой объект сцены никогда **не** добавляется в качестве ребёнка к другой вершине.

Благодаря последней гарантии всегда можно легко определить, какое из деревьев является сценой, ведь в дереве сцены корневой объект никогда не меняется.

Чтобы «подружить» дерево сцены three.js с новым отрисовщиком, Стёпа реализовал прослойку следующим образом. Каждый кадр, при вызове отрисовщика, прослойка выполняет полный обход сцены. Далее она сравнивает имеющиеся объекты с теми, которые были при отрисовке предыдущего кадра, и вызывает все необходимые методы отрисовщика. К сожалению, из-за медлительности javascript полный обход сцены для большой модели занимает несколько миллисекунд процессорного времени, и Стёпа очень хочет избавиться от необходимости выполнять его каждый кадр. Он полагает, что можно легко определить все изменения, произошедшие с момента предыдущего кадра, благодаря возможности отслеживать все операции, выполняемые пользователями на объектами three.js,

Необходимо реализовать новый вариант прослойки, который будет запускаться каждый кадр, принимать на вход последовательность всех операций, выполненных после отрисовки предыдущего кадра, и выдавать на выход три списка объектов:

- `added`: объекты, которые входят в сцену на этом кадре, но **не** входили на предыдущем кадре.
- `removed`: объекты, которые **не** входят в сцену на этом кадре, но входили на предыдущем кадре.
- `modified`: объекты, которые входят в сцену на этом кадре и входили на предыдущем, и которые *нуждаются в обновлении*.

Объект A *нуждается в обновлении*, если верно одно из:

1. После отрисовки предыдущего кадра один из предков A или сам A был отсоединён от своего родителя операцией `remove`. (т.е. путь до корня сцены был хотя бы раз разорван).
2. Для объекта A или одного из его предков на сцене была вызвана операция `modify` между этим и предыдущим кадрами (т.е. была модификация где-то на пути до корня сцены).

Можно заметить, что объект **не** *нуждается в обновлении* тогда и только тогда, когда на пути от него до корня сцены не было никаких изменений: ни структурных изменений дерева, ни изменений внутренних свойств объектов.

Формат входных данных

В начале входных данных описывается изначальное состояние дерева сцены. Объектов вне сцены изначально нет.

В первой строке задано целое число N — количество объектов в дереве сцены ($1 \leqslant N \leqslant 100\,000$). В последующих N строках описываются объекты сцены в произвольном порядке, по одному в строке. Для каждого объекта сначала указан его идентификатор — целое положительное число в пределах от 1 до 10^9. Далее указано количество его детей, а затем через пробел перечислены идентификаторы всех этих детей (в произвольном порядке, без повторений).

Гарантируется, что все описанные объекты образуют одно дерево, а корень этого дерева является корневым объектом сцены.

В следующей строке задано целое число M — количество выполняемых команд ($1 \leqslant M \leqslant 200\,000$). В следующих M строках описаны команды, по одной в строке.

В зависимости от типа команда описывается в формате:

- `add A B`

- `remove A B`
- `modify A`
- `render`

Описание команд `add`, `remove`, `modify` дано выше в условии задачи. Вместо `A` и `B` в каждой команде записаны идентификаторы объектов, над которыми выполняется операция. Гарантируется, что все эти объекты изначально присутствовали в дереве сцены. Команда `render` запускает отрисовку очередного кадра.

Гарантируется, что в любой момент времени **высота любого дерева объектов не превышает** 100.

Формат выходных данных

Для каждой команды `render` нужно вывести все изменения, произошедшие на сцене после предыдущей команды `render`. Если же до этой команды других команд `render` не было, то нужно вывести изменения, вызванные всеми командами до неё.

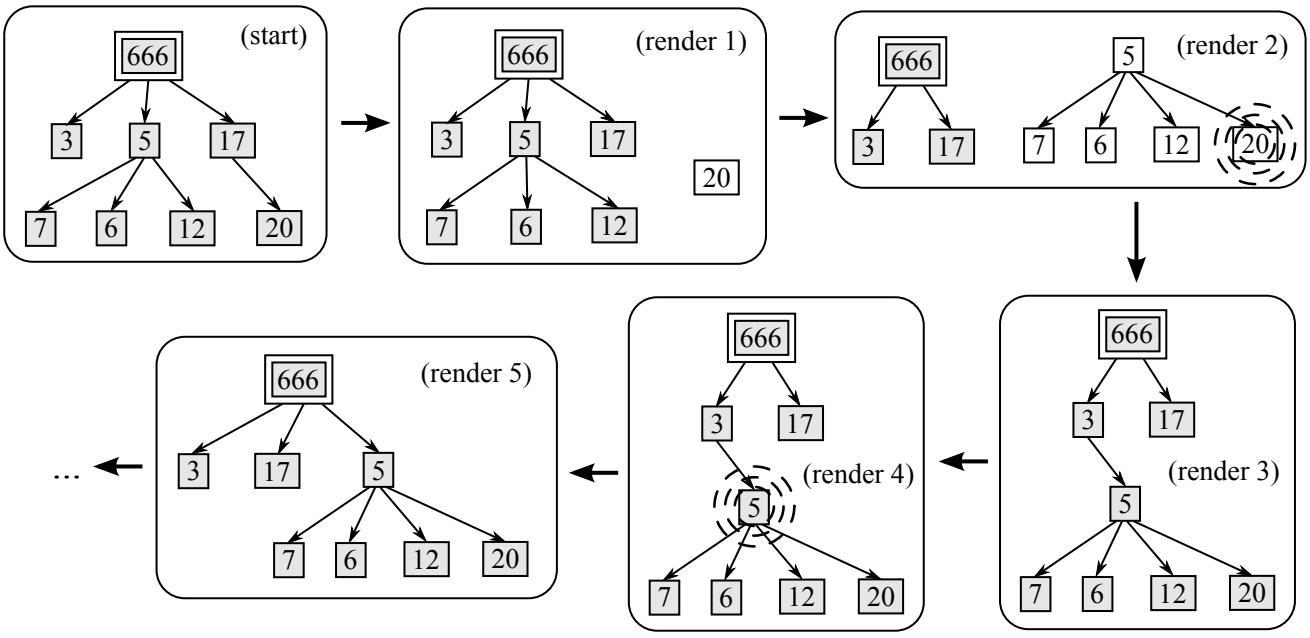
Изменения нужно выводить в виде трёх списков объектов: `added`, `removed`, `modified`. Определение этих списков дано в условии задачи выше. Для каждого списка нужно вывести в одной строке через пробел: название списка, количество объектов в списке, идентификаторы всех объектов в списке.

Изменения для команд `render` нужно выводить в порядке выполнения этих команд. В каждом описании изменений списки нужно выводить строго в порядке: `added`, `removed`, `modified`. В каждом списке идентификаторы нужно выводить в порядке увеличения.

Гарантируются следующие ограничения сверху на объём выходного файла. Суммарное количество объектов в ответе (по всем кадрам и по всем спискам) не превышает 300 000. Количество команд `render` не превышает 20 000.

Пояснение к примеру

На иллюстрации к примеру можно увидеть состояние объектов в момент вызова любой команды render. Сам пример приведён далее.



Пример

input.txt	output.txt
8 3 0 5 3 7 6 12 17 1 20 7 0 6 0 12 0 20 0 666 3 3 5 17 14 remove 17 20 render remove 666 5 add 5 20 modify 20 render add 3 5 render modify 5 render remove 3 5 add 666 5 render modify 666	added 0 removed 1 20 modified 0 added 0 removed 4 5 6 7 12 modified 0 added 5 5 6 7 12 20 removed 0 modified 0 added 0 removed 0 modified 5 5 6 7 12 20 added 0 removed 0 modified 5 5 6 7 12 20

Задача 8. Система весов

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

На полу стоит сложная система рычажных весов и гирь.

Рычажные весы состоят из опоры, перекладины и двух чаш (см. иллюстрацию к первому примеру). Перекладина жёстко **не** закреплена, и опирается на опору только в одной своей точке так, что она может свободно вращаться вокруг неё в вертикальной плоскости. Благодаря точному выбору точки опоры перекладина находится в горизонтальном положении неустойчивого равновесия. Чаши прикреплены к концам перекладины, и на них обычно ставят предметы, веса которых требуется сопоставить. Расстояние от точки опоры до чаши называется плечом рычага. Правило рычага гласит, что при достижении равновесия плечи рычага относятся так же, как относятся веса предметов на чашах.

Система рычажных весов и гирь устроена следующим образом. На каждой чаше каждых весов либо стоит гиря, либо стоят другие рычажные весы. Одни рычажные весы стоят непосредственно на полу, все остальные весы стоят на чашах других весов. Веса всех гирь известны, а веса самих весов (опор, перекладин и чаш) пренебрежимо малы по сравнению с весами гирь. Все весы всегда находятся в положении равновесия благодаря правильному выбору точки опоры. Размеры гирь, чаш и опор по сравнению с длинами перекладин также пренебрежимо малы.

Требуется обработать последовательность запросов двух типов:

1. Изменить вес заданной гири.
2. Узнать положение точки опоры у заданных весов.

После каждого изменения веса какой-либо гири все весы в системы заново уравновешиваются, при этом точки опоры некоторых весов смещаются.

Формат входных данных

В первой строке входного файла записано два целых числа: N — количество весов в системе ($1 \leqslant N \leqslant 5 \cdot 10^4$) и K — количество запросов ($1 \leqslant K \leqslant 10^5$).

Все весы пронумерованы последовательными целыми числами, начиная с единицы. Весы с номером 1 стоят на полу. Все гири так же пронумерованы последовательным целыми числами, начиная с единицы.

Во второй строке записано $(N+1)$ целых чисел: t-ое из этих чисел W_t определяет начальный вес гири с номером t ($1 \leqslant W_t \leqslant 10^9$).

В следующих N строках описываются весы. В i-ой из этих строк записано три целых числа: S_i — длина перекладины i-ых весов ($1 \leqslant S_i \leqslant 10^4$), L_i — номер весов, стоящих на левой чаше i-ых весов и R_i — номер весов, стоящих на правой чаше i-ых весов. Если на левой чаше стоят весы, то $i < L_i \leqslant N$, а если гиря — то L_i равно номеру гири со знаком «минус», при этом $1 \leqslant -L_i \leqslant N+1$. Аналогично, R_i задаёт либо номер стоящих весов ($i < R_i \leqslant N$), либо номер гири со знаком «минус» ($1 \leqslant -R_i \leqslant N+1$).

Далее во входном файле записано K строк, в каждой j-ой из которых описан один запрос. Описание запроса начинается с целого числа t_j, определяющего тип запроса ($1 \leqslant t_j \leqslant 2$). Если $t_j = 1$, то далее указано два целых числа: k_j — номер гири, вес которой изменяется ($1 \leqslant k_j \leqslant N+1$) и V_j — новый вес гири ($1 \leqslant V_j \leqslant 10^9$). Если $t_j = 2$, то далее указано одно целое число k_j — номер весов, у которых требуется узнать положение точки опоры ($1 \leqslant k_j \leqslant N$). Других типов запросов нет.

Формат выходных данных

Для каждого запроса на определение положения точки опоры требуется вывести в отдельной строке выходного файла одно вещественное число — расстояние от левой чаши заданных весов до точки опоры. Ответы нужно выдавать в порядке упоминания соответствующих запросов во входных данных.

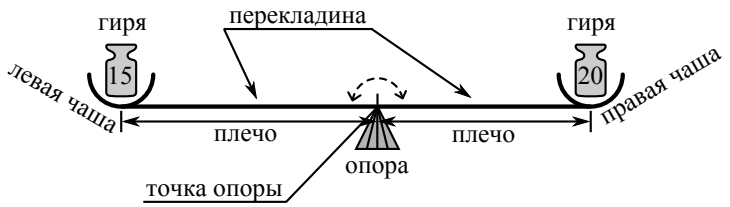
Абсолютная или относительная погрешность каждого ответа не должна превышать 10^{-13}.

Пример

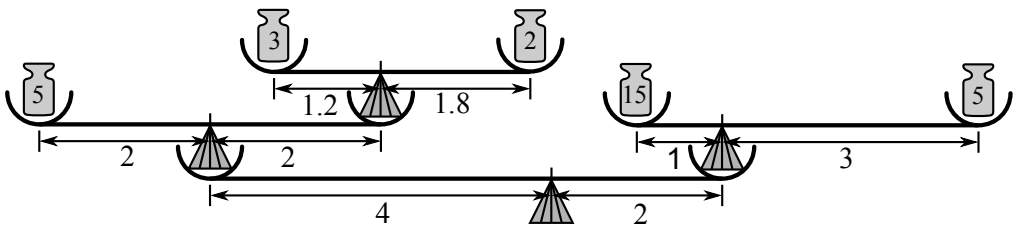
input.txt	output.txt
1 1 15 20 10 -1 -2 2 1	5.714285714285714285714285714285
4 9 3 2 5 5 15 6 2 3 4 -3 4 4 -5 -4 3 -1 -2 2 1 2 2 2 3 2 4 1 4 45 2 3 1 5 45 2 3 2 1	4 2 1 1.2 3 2 5.4

Пояснение к примеру

Первый пример и общая схема рычажных весов:



Начальное состояние системы из второго примера:



Задача 9. Кармон болеть

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

Вася очень любит ловить кармонов. Каждый кармон обладает числовым параметром *БС* (боевой силой). Чем этот параметр больше, тем кармон сильнее и, соответственно, ценнее.

Недавно Вася заболел и попросил своего друга Петю пойти половить кармонов вместо него. Петя согласился, но Вася высказал еще одну просьбу: он хочет, чтобы Петя, поймав каждого очередного кармона, сообщал ему сумму *БС* для K самых сильных кармонов, пойманных к этому моменту. Пете эта просьба показалась немного странной, но чего не сделаешь ради больного друга? Впрочем, он решил уточнить, что же делать, если он еще не собрал K кармонов. Вася подумал и решил, что в этом случае ничего сообщать не надо.

Помогите Пете написать программу, которая получит на вход список *БС* пойманных кармонов и выдаст значения, которые необходимо сообщать Васе.

Формат входных данных

В первой строке входного файла дано два целых числа: N — общее число пойманных кармонов и K — количество кармонов, сумму *БС* которых необходимо сообщать ($10 \leqslant N \leqslant 100\,000$, $2 \leqslant K \leqslant \min(N, 1000)$).

Во второй строке задано N целых чисел, определяющих *БС* кармонов в порядке их поимки. Все они находятся в диапазоне от 1 до 10 000 включительно.

Формат выходных данных

В единственную строку выходного файла необходимо вывести $(N - K + 1)$ целых чисел — суммы *БС* для K самых сильных пойманных кармонов после поимки каждого кармона, начиная с K-ого.

Пример

input.txt	output.txt
14 4 1 2 3 4 5 6 7 8 9 10 1 1 1 1	10 14 18 22 26 30 34 34 34 34 34

Задача 10. Battle City Online

Имя входного файла:	`input.txt`
Имя выходного файла:	`output.txt`
Ограничение по времени:	12 секунд
Ограничение по памяти:	512 мегабайт

Петя и Вася с детства являются большими поклонниками NES-игры «*Battle City*», более известной под названием «*Танчики*». Недавно у них появился повод для радости — официально объявлено о выходе современной онлайн-версии этой игры «*Battle City Online 2016*» («*BCO 2016*»). Они сразу же заполучили себе такую новинку и приступили к знакомству с игровым процессом. Каково же было их удивление, когда на старте игры они обнаружили себя единственными игроками на сервере. На экране красовалась лишь игровая карта, являющаяся клетчатым прямоугольным полем размера 8×8, на которой находился один единственный танк. Этим танком можно управлять с помощью имеющихся у ребят контроллеров. На каждом из двух абсолютно идентичных контроллеров имелось по восемь кнопок, которые отвечают за управление танком: кнопки движения в каждом из четырех направлений, а также кнопки выстрела в каждом из четырех направлений.

Каждая из клеток поля является либо пустой, либо занята участком кирпичной или бронированной стены, либо содержит участок водоема. Изначально танк находится в пустой клетке. При обработке нажатия на клавишу движения выполняется перемещение в указанном направлении на соседнюю по стороне клетку в случае, если она является пустой. Если соседней клетки в заданном направлении не существует, или она не является пустой, то после обработки нажатия танк не меняет своего местоположения. При нажатии на клавишу выстрела определяется участок стены, наиболее близкий в заданном направлении к танку. Если такой участок стены существует и при этом является кирпичным, то он уничтожается, т.е. клетка становится пустой до конца игры.

Казалось бы, Петя и Вася должны были окончательно расстаться с надеждами получить хотя бы какое-то удовольствие от игрового процесса, но в последний момент Петя заметил, что создатели игры не предусмотрели одновременное управление танком с использованием сразу двух контроллеров. Если в один момент времени оба игрока нажимают по одной кнопке на своих контроллерах, то будет обработано только одно из двух нажатий. Причем контроллер, нажатие на котором будет учтено, выбирается равновероятно.

Петя и Вася не растерялись и придумали следующую игру. Они закрывают глаза, берут в руки контроллеры. После чего, каждый из них последовательно выполняет по N нажатий на кнопки своего контроллера — по одному нажатию в секунду. Таким образом, в каждую секунду происходят одновременные нажатия на кнопки двух контроллеров, но обработано будет только одно из них. Как можно догадаться, ребята заранее делают ставки, на какой клетке игровой карты будет находиться танк после обработки нажатий во все моменты времени. Чья позиция окажется ближе к итоговой — тот и победил.

Вот только Вася не намерен честно играть в эту игру. Он прекрасно знает, какие кнопки на своем контроллере собирается нажать Петя. Сам же он давно решил, какие кнопки он нажмет на своем контроллере. Для каждой из клеток игровой карты ему необходимо знать вероятность того, что в конце игры танк окажется именно в ней. Посчитать такие вероятности он просит Вас. Ваша задача — написать программу, которая максимально точно определит эти значения.

Формат входных данных

Первая строка входного файла содержит единственное целое число N — количество нажа-

тий каждого из игроков ($1 \leqslant N \leqslant 30$). Вторая строка содержит N символов, описывающих нажатия Пети. Аналогично, третья строка содержит N символов, задающих описание нажатий Васи.

Каждый из символов второй и третьей строки может быть одним из следующих:

- «u» — переместиться вверх (в предыдущую строку),
- «d» — переместиться вниз (в следующую строку),
- «l» — переместиться влево (в предыдущий столбец),
- «r» — переместиться вправо (в следующий столбец),
- «U» — выстрелить вверх,
- «D» — выстрелить вниз,
- «L» — выстрелить влево,
- «R» — выстрелить вправо.

Далее во входном файле следует восемь строк по восемь символов в каждой, задающие игровую карту. Каждый из символов игровой карты может быть одним из следующих:

- «.» — пустая клетка,
- «B» — участок кирпичной стены,
- «A» — участок бронированной стены,
- «W» — участок с водоемом,
- «T» — стартовая позиция танка.

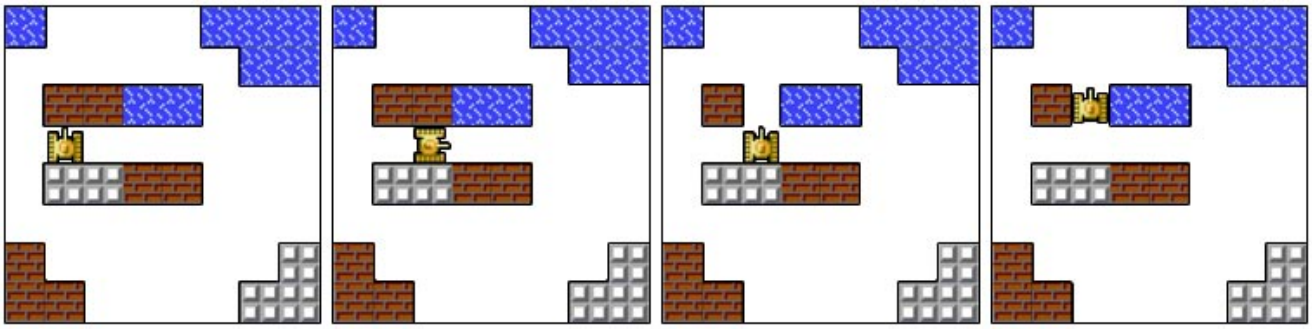
Формат выходных данных

Выходной файл должен содержать восемь строк, в каждую из которых следует вывести по восемь значений вероятностей в виде несократимых натуральных дробей P_{ij}/Q_{ij}. При выводе j-ое значение вероятности в i-й строке должно соответствовать вероятности того, что танк окажется в клетке на пересечении i-й строки и j-го столбца.

Знаменатель следует выводить всегда, даже если он равен единице. Обратите внимание на приведенные примеры для большей ясности.

Пояснение к примеру

Приведенное изображение демонстрирует один из возможных в приведенном примере маршрутов «DrDUu». Первым и третьим ходом выполняется выстрел в бронированный участок стены, а четвертым — уничтожается кирпичный участок.



В пример выходного файла добавлены дополнительные пробельные символы для лучшего визуального восприятия.

Пример

input.txt

```
5
rrrru
DUDUd
W....WWW
......WW
.BBWW...
.T......
.AABB...
........
B......A
BB....AA
```

output.txt

```
0/1  0/1  0/1  0/1  0/1  0/1  0/1  0/1
0/1  0/1  0/1  0/1  0/1  0/1  0/1  0/1
0/1 1/32 3/32  0/1  0/1 1/32  0/1  0/1
0/1 1/32 5/32 11/32 1/4  0/1  0/1  0/1
0/1  0/1  0/1  1/32 0/1 1/32  0/1  0/1
0/1  0/1  0/1  0/1  0/1  0/1  0/1  0/1
0/1  0/1  0/1  0/1  0/1  0/1  0/1  0/1
0/1  0/1  0/1  0/1  0/1  0/1  0/1  0/1
```

Задача 11. Генерация тестов

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

Недавно Паша придумал достаточно тривиальную задачу для одной из тренировок по олимпиадному программированию. Входными данными в этой задаче являлись строка S, состоящая из N цифр, а также три целых числа L, R и P ($1 \leqslant L \leqslant R \leqslant N$, P — простое). В качестве выходных данных было необходимо определить остаток от деления числа, образованного цифрами на позициях от L до R включительно, на число P. Стоит заметить, что иногда число, составленное из цифр на позициях от L до R, могло содержать лидирующие нули. Паша подготовил условие задачи, написал решение, а также подготовил много тестов для проверки решений.

Перед тренировкой Паша обнаружил, что T файлов со входными тестовыми данными пропали, а остались лишь соответствующие файлы с ответами на эти тесты. Он помнит, что строка S во всех этих тестах была абсолютно одинакова, более того, Паша прекрасно помнит эту строку. Аналогично, Паша помнит значение P, которое также совпадало во всех утерянных тестах. Теперь, чтобы восстановить утерянные входные данные, Паша просит Вас написать программу, на вход которой подается строка S длины N, числа P и T, а также T значений A_i — ответы на утерянные тестовые данные. Программа должна для каждого из значений A_i определить количество различных пар $\{L_i, R_i\}$ ($1 \leqslant L_i \leqslant R_i \leqslant N$) — пар допустимых значений из входного файла, а также найти одну из таких пар.

Формат входных данных

Первая строка входного файла содержит строку S, состоящую из N десятичных цифр ($1 \leqslant N \leqslant 10^5$). Во второй строке записано два целых числа T и P — количество пропавших тестовых данных и простое число, остаток от деления на которое требовалось определить ($1 \leqslant T \leqslant 100$, $11 \leqslant P \leqslant 10^9 + 33$, P — простое). Далее следует T строк, i-ая из которых содержит единственное целое число A_i — ответ на i-ый тестовый набор входных данных ($0 \leqslant A_i < P$).

Формат выходных данных

Для каждого из T ответов в выходной файл необходимо вывести в отдельную строку три целых числа C_i, L_i и R_i — количество различных допустимых пар входных значений и значения одной из таких пар соответственно. Если Паша ошибся при подготовке тестов, и для какого-то ответа не существует ни одной допустимой пары, то для этого ответа необходимо вывести три нуля.

Пример

input.txt	output.txt
923813 4 17 5 3 15 13	2 1 3 3 3 3 0 0 0 3 4 5