Задача A. Devil's Hell deLivery

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

Компания Devil's Hell deLivery доставляет буквально всё: ящики, коробки, пакеты, датаграммы и так далее.

Процесс доставки из города A в город B устроен следующим образом. Пусть нужно доставить K предметов, используя N грузовиков. Вместимость грузовика с номером i равна c_i. Вес предмета с номером j равен w_j. Грузовики доставляют предметы за несколько шагов.

За один шаг некоторые предметы целиком загружаются в грузовики. Все предметы можно перевозить только целиком. Вместимость каждого грузовика должна быть не меньше суммарного веса предметов, которые в него загружены. Все грузовики отправляются одновременно.

По окончании шага все грузовики отправляются на исходную позицию. Если какие-то предметы остались не доставленными, предпринимается очередной шаг.

Ваша задача — распределить предметы по шагам и грузовикам так, чтобы минимизировать количество шагов.

Формат входных данных

Во вводе задано не более ста тестовых случаев.

Каждый тестовый случай начинается со строки с двумя целыми числами N и K ($1 \leqslant N \leqslant 5$, $1 \leqslant K \leqslant 9$): количеством грузовиков и количеством предметов, соответственно. В следующей строке записано N целых чисел $c_1, \ldots, c_N$ ($1 \leqslant c_i \leqslant 10^8$): вместимости грузовиков. В следующей строке записано K целых чисел $w_1, \ldots, w_K$ ($1 \leqslant w_i \leqslant 10^8$): веса предметов.

Формат выходных данных

Для каждого тестового случая, если доставка невозможна, выведите единственную строку с числом -1.

В остальных случаях сначала выведите строку с целым числом S — необходимым количеством шагов. Это число нужно минимизировать. Затем выведите S строк, по одной на каждый шаг. Описание шага должно начинаться с целого числа I_i — количества предметов, которые перевозятся на этом шаге. Далее запишите I_i пар a_j b_j. Каждая пара a_j b_j означает, что предмет a_j перевозится на грузовике b_j.

Если возможных оптимальных ответов несколько, выведите любой из них.

Пример

dhl.in	dhl.out
2 4 10 20 5 5 5 5 2 1 10 10 20	1 4 1 1 2 1 3 2 4 2 -1

Задача B. Доминошки на торе

Имя входного файла:	`domino-on-torus.in`
Имя выходного файла:	`domino-on-torus.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Из тянущейся клетчатой бумаги вырезан прямоугольник размера $A \times B$ по линиям сетки. Все клетки этого прямоугольника пронумерованы числами от 1 до $A \cdot B$. Из первого прямоугольника вырезан второй прямоугольник размера $C \times D$, тоже по линиям сетки, так чтобы стороны длины C были параллельны сторонам длины A. Склеим стороны длины B, а потом стороны длины A. Получится тор с прямоугольной дыркой из $C \times D$ клеток. (Тор — это поверхность бублика.)

Две клетки тора будем считать различными, если они имеют разные номера. Внешнюю поверхность этого тора с дыркой мы будем замощать тянущимися доминошками, каждая из которых состоит из двух соседних по стороне клеток разного цвета — белой и чёрной. Замощение должно удовлетворять следующему условию: если две соседние по стороне клетки принадлежат разным доминошкам, то они должны быть одного цвета — обе белые или обе чёрные.

Два замощения считаются различными, если выполнено хотя бы одно из двух следующих условий:

1. существует хотя бы одна такая клетка, что в одном замощении она является белой, а в другом — чёрной;
2. существует хотя бы одна такая клетка, что в одном замощении доминошка, которая её покрывает, покрывает также и клетку Y_1, а в другом — клетку Y_2, причём $Y_1 \neq Y_2$.

Формат входных данных

В первой строке ввода задано четыре целых числа A, B, C и D ($4 \leqslant A, B \leqslant 10^9$, $2 \leqslant C < A$, $2 \leqslant D < B$, все числа чётные).

Формат выходных данных

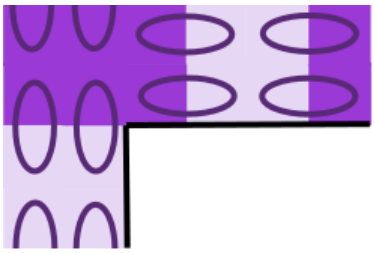
Выведите число замощений.

Пример

domino-on-torus.in	domino-on-torus.out
4 6 2 4	4

Пояснение к примеру

На рисунке показано одно из возможных замощений теста из примера.



Задача C. Поиск маленьких чисел

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Это — интерактивная задача.

Жюри загадало перестановку чисел от 1 до n. Ваша задача — найти позиции, в которых стоят числа от 1 до k. Для этого вы можете воспользоваться программой жюри, которая умеет сравнивать числа, стоящие на двух позициях в перестановке.

Формат входных данных

В первой строке будет задано два числа n и k — размер перестановки и количество чисел, позиции которых надо найти. Во всех тестах, кроме теста из примера, $n = 10\,000$, а $k \leqslant 10$.

Далее будут следовать ответы на ваши запросы по одному в строке. Если первое число из сравниваемых меньше, то в строке будет содержаться единственный символ «`<`», иначе — единственный символ «`>`».

Формат выходных данных

Если вы хотите сравнить числа на i-й и j-й позициях, необходимо вывести строку «`?` i j». При этом i и j должны быть различными целыми числами от 1 до n. Вы можете сделать не более 10 700 таких запросов.

Если вы нашли позиции всех чисел от 1 до k, то необходимо вывести «`!` pos_1 pos_2 ... pos_k», после чего завершить работу программы.

Чтобы предотвратить буферизацию вывода, после каждой выведенной строки следует вставить команду очистки буфера вывода: например, это может быть `fflush (stdout)` в C или C++, `System.out.flush ()` в Java, `flush (output)` в Pascal или `sys.stdout.flush ()` в Python.

Также не забывайте выводить символ перевода строки в конце каждой строки, которую вы выводите.

Пример

стандартный ввод	стандартный вывод
3 3 *(waiting for output)* < *(waiting for output)* > *(waiting for output)* <	*(reading input)* ? 1 2 *(reading input)* ? 3 1 *(reading input)* ? 2 3 *(reading input)* ! 1 2 3 *(terminating)*

Пояснение к примеру

В примере загадана перестановка 1 2 3.

Задача D. Лабиринт в лесу

Имя входного файла:	*стандартный ввод*
Имя выходного файла:	*стандартный вывод*
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Это — интерактивная задача. В ней нужно дойти от клетки входа в неизвестный лабиринт до клетки выхода из него за не слишком большое время.

Прославленный воин Малькольм, Герой Аркании, шёл ночью по тёмному лесу. Он шёл долго и уже думал, что заблудился, как вдруг наткнулся на каменные стены. По клейму на каменной плите, видневшемуся под опавшими листями, Малькольм понял: это лабиринт гномьей работы.

Герой совсем не разбирается в лесе и деревьях, растущих в нём. Зато он провёл много времени в компании гномов, так что теперь может многое сказать про лабиринт, посмотрев на одно лишь клеймо.

> На клейме этого лабиринта красуются символы «SK3». Символ «S» означает, что лабиринт квадратный, то есть его пол состоит из $n \times n$ квадратных клеток размера метр на метр. Вокруг этих клеток — тонкая сплошная стена. В двух местах этой стены — проёмы: вход в лабиринт, возле которого стоит герой, расположен в южной стене юго-западной угловой клетки лабиринта, выход — в северной стене северо-восточной угловой клетки.
>
> Между каждыми двумя клетками лабиринта, имеющими общую сторону, может быть либо проход, либо метровая секция тонкой внутренней стены. В лабиринте из любой клетки можно двигаться в любую соседнюю с ней по стороне, если между этими клетками нет секции стены. Также можно свободно двигаться через проём выхода. А проём входа открыт до тех пор, пока герой не ступит внутрь лабиринта, после чего он немедленно закрывается и открывается снова лишь тогда, когда герой благополучно доберётся до выхода.
>
> Символы «K3» говорят о том, что лабиринт имеет следующую структуру. Сначала изо всех возможных секций тонких внутренних стен (всего может быть $n \times (n-1)$ секций, идущих с севера на юг, и столько же секций, идущих с запада на восток) выбирается случайная треть (нецелое количество стен округляется вниз) и фиксируется их случайный порядок. После этого эти секции ставятся в выбранном порядке. При этом, если какая-то секция при её установке разбила бы квадрат, в котором строится лабиринт, на не связанные области, эта секция пропускается (не ставится). Таким образом гарантируется связность полученного лабиринта.

Малькольм обрадовался: зная, как устроен лабиринт, он наверняка сможет дойти до выхода из него. Его цель — начав движение из клетки леса рядом с входным проёмом, оказаться в клетке леса рядом с выходным проёмом лабиринта. Кроме того, следует поспешить: тому, кто пройдёт путь от входа до выхода достаточно быстро, откроется сокровище, спрятанное в лесу рядом с выходом — если, конечно, никто не добрался до него раньше.

Впрочем, некоторые трудности всё же предстоят. Во-первых, Малькольм не знает размера лабиринта: длина стороны квадрата n может быть любым целым числом метров от 10 до 200. Во-вторых, в лабиринте, как и в окружающем его лесу, темно, поэтому Малькольм двигается почти на ощупь. В начале каждого шага герой выбирает одно из четырёх направлений: на север, на восток, на юг или на запад. После этого он пытается переместиться на один метр в этом направлении. Если герой натыкается на препятствие, он остаётся там, где был. В противном случае он перемещается на соседнюю клетку в выбранном направлении. Время, потраченное на шаг, не зависит от результата этого шага, поэтому важно лишь количество шагов.

Помогите герою идти так, чтобы оказаться в требуемом месте достаточно быстро. Решение считается правильным, если герой сделает не более $5 \cdot n + 300$ шагов и окажется в клетке снаружи от выхода из лабиринта.

Протокол взаимодействия

В этой задаче программа жюри реагирует на команды, выведенные программой участника в стандартный поток вывода. Каждая команда определяет следующий шаг Малькольма. Команда выводится на отдельной строке и состоит из одной заглавной английской буквы: «`N`» для шага на север, «`E`» для шага на восток, «`S`» для шага на юг и «`W`» для шага на запад.

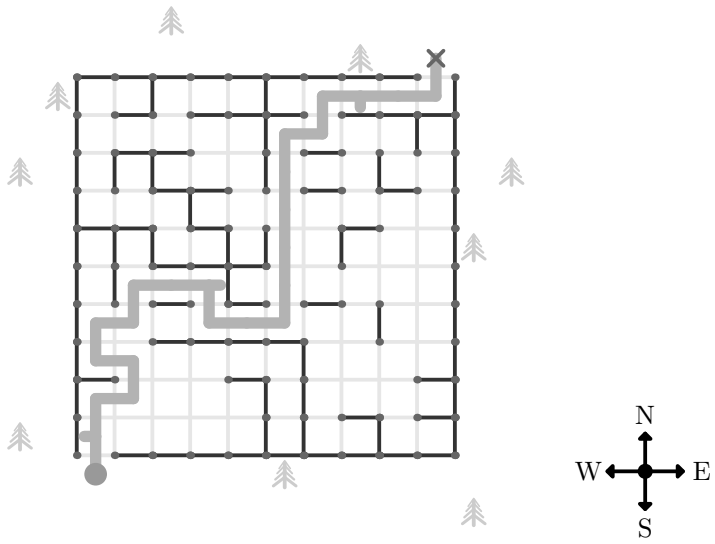
В ответ на каждую команду жюри сразу же передаёт участнику в стандартный поток ввода строку, содержащую одно слово, записанное маленькими буквами английского алфавита: «`no`», если шагу помешало препятствие, «`ok`», если перемещение удалось, но не привело героя на конечную клетку, и «`end`», если перемещение удалось и герой оказался на конечной клетке. После получения ответа «`end`» решение должно закончить свою работу, ничего больше не выводя.

Чтобы предотвратить буферизацию вывода, после каждой выведенной строки следует вставить команду очистки буфера вывода: например, это может быть `fflush (stdout)` в C или C++, `System.out.flush ()` в Java, `flush (output)` в Pascal или `sys.stdout.flush ()` в Python.

Лабиринт в каждом тесте, хотя и сгенерирован случайно, зафиксирован заранее.

Если решение сделало не менее 100 000 шагов, но ещё не превысило ограничения по времени и не пришло в конечную клетку, проверка завершается с вердиктом «`Wrong Answer`».

Пример



Пояснение к примеру

На картинке выше показан лабиринт, сгенерированный в первом тесте. Малькольм начинает в клетке, отмеченной кругом, и должен оказаться в клетке, отмеченной крестиком. Этот лабиринт можно пройти, например, следующей последовательностью команд (снизу от команд показаны результаты их выполнения):

N	W	N	E	N	W	N	E	N	E	E	E	S	E	E	N	N	N	N	N	E	N	E	S	E	E	N
ok	no	ok	ok	ok	ok	ok	ok	ok	ok	ok	no	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	no	ok	ok	end	

На картинке это решение представлено толстой серой линией.

Задача E. Грамматика

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

Формальной грамматикой называется способ описания формального языка, представляющий собой четвёрку $\Gamma = \langle \Sigma, N, S \in N, P \subset N^+ \times (\Sigma \cup N)^* \rangle$, где Σ — *алфавит*, элементы которого называют *терминалами*, N — множество, элементы которого называют *нетерминалами*, S — начальный нетерминал грамматики, P — набор *правил вывода* вида $\alpha \to \beta$.

Здесь N^+ — это все строки из одного или нескольких элементов множества N (непустые строки из нетерминалов), а $(\Sigma \cup N)^*$ — это все строки из нуля, одного или нескольких элементов множества $(\Sigma \cup N)$ (строки из терминалов и нетерминалов, включая пустую).

Грамматика называется *контекстно-свободной*, если в левой части каждого правила вывода всегда стоит только один нетерминал, то есть верно более сильное условие для правил: $P \subset N \times (\Sigma \cup N)^*$.

Для примера рассмотрим такую грамматику (второй тестовый случай примера) над алфавитом $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, множеством нетерминалов $N = \{S, A\}$ и двумя правилами вывода:

1. $S \to \mathtt{b}A$
2. $A \to \mathtt{aa}$

Она, очевидно, является контекстно-свободной.

Чтобы получить язык, описываемый грамматикой, нужно взять строку, составленную из одного начального нетерминала S, и применить к ней один или несколько раз какие-либо правила вывода. Применение правила вывода состоит в нахождении в текущей строке в каком-либо месте строки из левой части этого правила и замены её на правую часть этого же правила вывода. *Языком* грамматики Γ будет называться множество всех строк, состоящих **только** из терминальных символов и выводимых по такому принципу.

Например, в языке описанной выше грамматики есть строка `baa`, выводимая следующим образом: $S \to \mathtt{b}A \to \mathtt{baa}$. Более того, других строк в её языке нет.

Но бывают и грамматики, в языке которых бесконечное количество строк. А бывают и такие, язык которых пуст.

Вам дана контекстно-свободная грамматика, алфавит в которой состоит из двух терминалов «`a`» и «`b`». Ваша задача — проверить, есть ли в её языке строка, в которой символ «`a`» встречается строго большее число раз, чем символ «`b`».

Нетерминалы в этой задаче будут нумероваться числами от 1 до n. Начальный нетерминал всегда имеет номер 1.

Формат входных данных

Во вводе заданы один или несколько тестовых случаев.

В первой строке каждого тестового случая записаны два числа n и m — число нетерминалов и количество правил вывода ($1 \leqslant n \leqslant 100$, $1 \leqslant m \leqslant 50\,000$).

Далее следуют m строк, каждая из которых описывает одно правило вывода в следующем виде. Сначала заданы A_i и k_i — номер нетерминала в левой части ($1 \leqslant A_i \leqslant n$) и число объектов в правой части правила вывода ($0 \leqslant k_i \leqslant 100$), далее следует k_i объектов, каждый из которых — либо нетерминал $B_{i,j}$ ($1 \leqslant B_{i,j} \leqslant n$), либо один из терминалов «`a`» или «`b`». Объекты разделяются одиночными пробелами.

Общая сумма значений n по всем тестовым случаям не превосходит 1000, а общая сумма значений m не превосходит 50 000. Размер ввода не превышает 5 мегабайт.

Ввод завершается строкой из двух нулей.

Формат выходных данных

Для каждого из тестовых случаев необходимо вывести на отдельной строке «`YES`», если в языке данной грамматики есть строка, в которой количество букв «`a`» строго больше, чем количество букв «`b`», и «`NO`» в противном случае.

Пример

grammar.in	grammar.out
2 2 1 2 a 2 2 1 b 2 2 1 2 b 2 2 2 a a 2 2 1 2 b 2 2 3 a a 1 0 0	NO YES NO

Задача F. Ещё одна задача про поиск точки

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

Вам дано n точек на плоскости: $A_1, A_2, \cdots, A_n$. У i-й точки есть вес w_i. Найдите такую точку B, что значение максимума взвешенного расстояния $\max\limits_{i=1}^{n} w_i \cdot |A_i B|$ минимально.

Формат входных данных

Во вводе содержится один или несколько тестовых случаев.

В первой строке каждого тестового случая задано число n ($1 \leqslant n \leqslant 500\,000$) — количество точек. В каждой из следующих n строк задано по три числа: x_i, y_i и w_i. Все числа целые и не превосходят 10^7 по модулю. Вес каждой точки строго положителен.

Тестовые случаи следуют друг за другом без каких-либо пропусков. Ввод заканчивается строкой, содержащей одно число 0. Его не надо считать ещё одним тестовым случаем. Сумма чисел n во всех тестовых случаях не превосходит 500 000.

Гарантируется, что во вводе не более 1000 тестовых случаев.

Формат выходных данных

В ответ на каждый тестовый случай выведите два числа — координаты точки B. Ваш ответ будет считаться правильным, если абсолютная или относительная погрешность максимума взвешенного расстояния не будет превосходить 10^{-9}.

Пример

minwdist.in	minwdist.out
2 2 2 1 0 0 1 3 0 0 1 6 0 2 0 6 3 0	1.0 1.0 2.4 3.6

Задача G. Наножучки

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

В этой задаче нужно для набора жучков, каждый из которых является шпионом или агентом, показать, чему равно точное число жучков-шпионов, не раскрыв ни одного шпиона или агента.

Наножучки — интеллектуальные роботы, выполняющие волю своих хозяев. Обычно они заняты подслушиванием, подсматриванием или слежкой за другими наножучками. Все наножучки на вид совершенно одинаковые.

Бартош и Вивек — инженеры двух корпораций: Восточного Картеля и Южной Торговой Компании. Им поручена проверка помещения для переговоров в офисе Восточного Картеля, в которой представители их корпораций будут обсуждать новый контракт. Вместе они смогли обнаружить и поймать n наножучков. Каждый пойманный жучок — либо агент безопасности Восточного Картеля, либо шпион Южной Торговой Компании.

Андерс — эксперт по безопасности, работающий в компании СпайТек. Его работа на сегодня — помочь Бартошу и Вивеку определить, сколько среди пойманных наножучков агентов, а сколько шпионов. Андерс знает про каждого жучка, шпион он или агент, но никто другой не может этого определить.

Андерс видит, что среди жучков ровно a шпионов. Бартош и Вивек же смогли лишь выяснить при помощи стандартных приборов, что всего шпионов либо a, либо b, причём числа a и b отличаются ровно на единицу. Никакой другой информацией о жучках они не обладают.

Наножучки — дорогостоящие интеллектуальные механизмы, которые особенно ценят сохранение своего инкогнито. Конечно, Андерсу как эксперту известно, кто хозяин каждого из них. Но если про какого-то наножучка точная информация о том, на какую корпорацию он работает, станет известна инженерам или другим наножучкам, дальнейшая деятельность этого жучка (да и само его существование!) будет поставлена под угрозу.

Прибор, который должен помочь Андерсу в его работе — балансировщик. Это небольшая коробочка с двумя камерами и индикатором. Работать с балансировщиком легко: Андерс помещает каких-то наножучков в каждую из камер и нажимает на кнопку. После этого прибор для каждой корпорации проверяет, что количество наножучков, работающих на неё, одинаковое в обеих камерах. Если это верно для обеих корпораций, индикатор загорается зелёным, иначе — красным.

Андерс планирует произвести серию проверок с помощью балансировщика. Для этого он сначала поставит всех наножучков в ряд в том порядке, в каком захочет. В ходе каждой проверки Андерс будет помещать каких-то наножучков в одну камеру, каких-то в другую, нажимать на кнопку, показывать инженерам результат работы балансировщика, после чего возвращать всех наножучков на места в ряду с сохранением их прежнего порядка. При такой процедуре можно считать, что на время всей серии проверок все наножучки пронумерованы целыми числами от 1 до n в соответствии с их местами в ряду.

Помогите Андерсу доказать инженерам, что среди n пойманных наножучков ровно a шпионов, так, чтобы при этом каждый жучок сохранил своё инкогнито, или установите, что это невозможно.

Формат входных данных

В первой строке заданы три целых числа n, a и b ($20 \leqslant n \leqslant 50$, $1 \leqslant a, b \leqslant 4$, $|a - b| = 1$). Здесь n — общее количество наножучков. Бартош и Вивек знают, что среди них либо a, либо b шпионов, а задача Андерса — доказать, что на самом деле точное количество шпионов равно a.

Формат выходных данных

В первой строке выведите одно число — количество проверок m ($0 \leqslant m \leqslant 1000$) или -1, если выполнить все условия задачи невозможно. В случае положительного ответа выведите m строк, по одной строке на каждую проверку.

Описание каждой проверки содержит ровно ту информацию, которую Андерс показывает инженерам, и состоит из трёх частей. Сначала записывается, какие жучки помещаются в первую камеру:

сперва количество жучков, а затем их номера в любом порядке. После этого следует символ, обозначающий результат проверки: «=» (ASCII-код 61) для положительного результата (зелёный цвет индикатора) или «^» (ASCII-код 94) для отрицательного (красный цвет индикатора). Далее записывается, какие жучки помещаются во вторую камеру, в том же формате, что и для первой камеры. Всё это записывается в одну строку, причём соседние числа или символы разделяются одним или несколькими пробелами.

Один и тот же жучок не может быть упомянут дважды в ходе одной проверки.

Замечание

Заметим, что нигде не записывается, какие именно наножучки фактически были шпионами, а какие — агентами. Эта информация доступна только Андерсу и, возможно, вашему решению — никто другой не должен её узнать.

Пример

nanobugs.in	nanobugs.out
22 2 1	4 6 1 2 3 4 5 6 = 6 7 8 9 10 11 12 5 13 14 15 16 21 = 5 17 18 19 22 20 3 13 14 17 ^ 3 6 8 9 1 1 ^ 2 2 3

Пояснение к примеру

В примере инженеры знают, что из 22 пойманных наножучков либо один, либо два являются шпионами. На самом же деле шпионов ровно два, и задача Андерса — доказать это. В предлагаемом ответе он решил произвести серию из четырёх проверок.

Первая проверка говорит о том, что среди первых шести жучков столько же шпионов, сколько среди следующих шести жучков. Аналогичное утверждение здесь и далее верно про агентов: заметим, что, если в первой камере наножучков столько же, сколько во второй, количества шпионов в них равны тогда и только тогда, когда равны количества агентов.

После второй проверки можно утверждать, что среди жучков, имеющих номера 13, 14, 15, 16 и 21, столько же шпионов, сколько среди жучков с номерами 17, 18, 19, 22 и 20.

Третья проверка говорит о том, что количество шпионов среди жучков 13, 14 и 17 обязательно отличается от количества шпионов среди жучков 6, 8 и 9.

Наконец, четвёртая проверка не несёт никакой информации, так как проверка с разными количествами жучков в камерах всегда выдаст отрицательный результат.

Какие же жучки могли быть шпионами?

Пусть, например, Андерс перед проверками поставил двух шпионов на позиции 2 и 8. Тогда первая проверка выдала равенство, потому что в обеих камерах по одному шпиону и по пять агентов. Вторая проверка выдала равенство, так как в обеих камерах шпионов нет. Третья проверка выдала неравенство: среди наножучков 13, 14 и 17 нет шпионов, а среди 6, 8 и 9 — один шпион. Четвёртая проверка не зависит от положения шпионов. Получается, что все проверки выдали требуемые результаты. Значит, по результатам этих проверок *возможность* того, что шпионов ровно два, доказана.

Предположим теперь на месте инженеров, что шпион на самом деле один и имеет номер 1. Но в этом случае первая проверка выдала бы неравенство. Аналогично можно проверить, что, если шпион имеет номер 2, 3, ..., 22, хотя бы одна из проверок также выдала бы другой ответ. Значит, доказана *невозможность* того, что шпион ровно один. А поскольку Бартош и Вивек знают, что шпионов либо один, либо два, получается, что Андерс доказал, что их ровно два.

Остаётся проверить, что ни один агент и ни один шпион не раскрыт. Для этого можно предъявить такой набор вариантов расстановки шпионов, что каждый из этих вариантов выдаёт требуемые результаты проверок, а кроме того, каждый наножучок хотя бы в одном из вариантов является агентом и хотя бы в одном — шпионом. Таким набором вариантов являются, например, расстановки шпионов на позициях $(1,8)$, $(2,9)$, $(3,8)$, $(4,9)$, $(5,8)$, $(6,7)$, $(6,10)$, $(6,11)$, $(6,12)$, $(13,17)$, $(13,18)$, $(13,19)$, $(13,20)$, $(13,22)$, $(14,18)$, $(15,17)$, $(16,17)$ и $(17,21)$.

Задача H. Некратчайший путь

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

В этой задаче нужно пройти из одного угла клетчатого поля 4×4 в другой по любому простому пути, не являющемуся кратчайшим.

На плоскости нарисовано квадратное клетчатое поле из 4×4 клеток. Каждая клетка поля либо свободна, либо целиком занята стеной.

Робот Аврелий проходит тест на измерение уровня интеллекта, решая различные задачи на этом поле. Он уже успешно справился с поиском произвольного и кратчайшего пути из одной клетки в другую.

В очередном задании Аврелию следует пройти из левого верхнего угла поля в правый нижний по свободным клеткам. За один шаг из любой клетки можно переместиться в любую другую клетку, имеющую с ней общую сторону. При этом путь должен быть простым, то есть в одной и той же клетке нельзя оказываться дважды. Кроме того, длина пути должна быть строго больше, чем длина кратчайшего пути между этими углами на заданном поле. Длиной пути считается количество шагов в нём.

Напишите программу, которая позволит роботу Аврелию справиться с заданием.

Формат входных данных

На входе задан один или несколько тестовых случаев. Каждый тестовый случай задаётся на четырёх строках. Каждая из этих строк описывает один ряд поля и содержит ровно четыре символа, задающих клетки этого ряда. Пустая клетка обозначается символом «`.`» (точка, ASCII-код 46), а стена — символом «`#`» (решётка, ASCII-код 35). Соседние тестовые случаи отделяются друг от друга строкой, состоящей из четырёх символов «`-`» (минус, ASCII-код 45).

Гарантируется, что левая верхняя и правая нижняя клетки поля пусты. Также гарантируется, что тестовые случаи на входе не повторяются в пределах одного теста.

Формат выходных данных

В ответ на каждый тестовый случай выведите строку, определяющую путь. Строка должна состоять из символов, соответствующих командам перемещения в соседнюю клетку поля, в порядке их применения: «`D`» для движения вниз, «`U`» для движения вверх, «`L`» для движения влево и «`R`» для движения вправо. Если возможных путей несколько, выведите любой из них. Если же пути с нужными свойствами не существует, выведите вместо него число «`-1`».

Пример

nsp.in	nsp.out
.#.. .... ..#. .#.. ---- ..## ...# #... .#.. ---- ...# ..#. .#.. #...	DDRURURDDD -1 -1

Пояснение к примеру

В примере задано три тестовых случая.

В первом тестовом случае длина кратчайшего пути равна шести: кратчайшим является единственный путь «DRRRDD». Кроме того, есть три простых пути, не являющихся кратчайшими: это пути «DDRURRDD» и «DRRURDDD» длины 8, а также путь «DDRURURDDD» длины 10. Любой из них подходит в качестве ответа.

Во втором тестовом случае есть восемь различных простых путей, но все они имеют одинаковую длину, то есть являются кратчайшими.

В третьем тестовом случае путей из левого верхнего угла в правый нижний угол по свободным клеткам не существует.

Задача I. Композиция многочленов

Имя входного файла:	`polycomp.in`
Имя выходного файла:	`polycomp.out`
Ограничение по времени:	2 секунды (*3.5 секунды для Java*)
Ограничение по памяти:	256 мебибайт

Даны многочлены $f(x)$, $g(x)$, $h(x)$ над полем $\mathbb{Z}/2\mathbb{Z}$.

Найдите многочлен $f(g(x)) \bmod h(x)$.

Формат входных данных

Три строки ввода содержат многочлены f, g и h, по одному на строке. Каждый многочлен p описывается как n p_0 p_1 p_2 ... p_n ($1 \leqslant n \leqslant 4000$, $p_i \in \{0, 1\}$ для всех i, а $p_n = 1$). Сам многочлен $p(x)$ в таком случае равен $p_0 + p_1 x + p_2 x^2 + \cdots + p_n x^n$.

Формат выходных данных

Выведите ответ в том же формате.

Возможен ответ вида «`0 0`», обозначающий тождественный ноль.

Примеры

polycomp.in	polycomp.out
5 0 1 0 1 0 1 2 1 1 1 4 0 1 1 0 1	1 1 1
2 1 1 1 3 0 0 1 1 4 1 0 1 0 1	3 1 0 0 1

Замечание

Напомним несколько определений.

Поле $\mathbb{Z}/2\mathbb{Z}$ — это множество из двух чисел 0 и 1, в котором результаты сложения, вычитания, умножения и деления — это остатки по модулю 2 от аналогичных результатов для обычных чисел.

Многочлен $f(x)$ над этим полем — это объект вида $f_n \cdot x^n + f_{n-1} \cdot x^{n-1} + \ldots + f_1 x + f_0$, где коэффициенты f_n, ..., f_0 — числа из $\mathbb{Z}/2\mathbb{Z}$, и переменная x тоже может принимать значения из $\mathbb{Z}/2\mathbb{Z}$. Число n — максимальное такое, что $f_n \neq 0$ — называется степенью многочлена $p(x)$.

Многочлены $a(x) = \sum\limits_k a_k x^k$ и $b(x) = \sum\limits_k b_k x^k$ равны, если для любого k числа a_k и b_k равны.

Сложение и вычитание многочленов определяются покомпонентно: $a(x) \pm b(x) = \sum\limits_k (a_k \pm b_k) \cdot x^k$.

Произведение многочленов $a(x)$ и $b(x)$ определяется как $c(x) = \sum\limits_k c_k x^k$, где $c_s = \sum\limits_{t=0}^{s} (a_t \cdot b_{s-t})$.

Многочлены можно делить друг на друга. Если многочлен $b(x)$ не является тождественным нулём, говорят, что $a(x)/b(x) = q(x)$ и $a(x) \bmod b(x) = r(x)$, если $q(x) \cdot b(x) + r(x) = a(x)$, а степень $r(x)$ строго меньше степени $b(x)$. Несложно показать, что многочлены $q(x)$ и $r(x)$ определены однозначно.

Композиция $a(b(x))$ — это многочлен $\sum\limits_k a_k (b(x))^k$, где степень многочлена определяется через произведение: $(b(x))^0 = 1$, $(b(x))^1 = b(x)$, $(b(x))^p = b(x) \cdot (b(x))^{p-1}$ для $p > 1$. Коэффициенты композиции можно получить, если раскрыть скобки и сложить коэффициенты при одинаковых степенях переменной x.

Задача J. Потенциал

Имя входного файла:	`potential.in`
Имя выходного файла:	`potential.out`
Ограничение по времени:	3 секунды
Ограничение по памяти:	256 мебибайт

Дан взвешенный ориентированный граф. Пусть у каждой вершины есть потенциал Φ_i. Тогда к весу каждого ребра прибавляется потенциал начала и вычитается потенциал конца.

Требуется найти такие целые Φ_i, чтобы веса у всех рёбер были одинаковыми.

Формат входных данных

В первой строке задано целое число t — количество тестовых случаев.

В первой строке каждого тестового случая заданы целые числа n и m ($1 \leqslant n \leqslant 300\,000$, $0 \leqslant m \leqslant 300\,000$) — количество вершин и рёбер в графе. В следующих m строках задано по три целых числа x_i, y_i и w_i ($1 \leqslant x_i, y_i \leqslant n$, $-10^9 \leqslant w_i \leqslant 10^9$) — начало, конец и вес ребра. Гарантируется, что граф не содержит кратных рёбер и петель.

Также гарантируется, что сумма всех n и m по всем тестовым случаям не превосходит $600\,000$.

Формат выходных данных

Для каждого тестового случая выведите «`YES`», если существует целочисленное решение, и «`NO`» в противном случае.

Если ответ положительный, то в следующей строке выведите n целых чисел — потенциалы вершин. Все выведенные числа должны быть не больше 10^{18} по абсолютной величине. Гарантируется, что если ответ существует, то существует и ответ, удовлетворяющий этому ограничению.

Если возможных ответов несколько, выведите любой из них.

Пример

potential.in	potential.out
2 5 4 1 2 -1 2 3 2 3 4 1 4 5 179 5 5 1 2 1 2 3 1 3 4 1 4 5 0 5 1 2	YES 0 -1 1 2 181 YES 0 0 0 0 -1

Задача K. Что за Флекс?

Имя входного файла:	`wtflex.in`
Имя выходного файла:	`wtflex.out`
Ограничение по времени:	0.5 секунды
Ограничение по памяти:	256 мебибайт

В этой задаче вам нужно найти следующее в некотором порядке число с тем же набором простых делителей.

Археолог Крис изучает древнюю платформу Флекс, которую недавно обнаружили рядом с ручьём Аудоби. На этой платформе когда-то были записаны номера версий различных телепортов, которые были созданы Древнейшими. Каждый номер версии — это целое число от 1 до N. На платформе имеется несколько столбцов чисел, каждый из столбцов соответствует одному телепорту. В каждом столбце были записаны все когда-либо созданные версии этого телепорта в порядке постройки, от старых к новым. К сожалению, некоторые номера уже стёрлись.

Крис только что закончил изучение версии номер a телепорта b и готов приниматься за следующую версию этого телепорта. К сожалению, он не знает даже номера этой следующей версии, в отличие от любой программы, являющейся решением этой задачи. Разумеется, про телепорты и номера их версий кое-что известно:

1. Каждое число от 1 до N является номером какой-то версии какого-то телепорта.
2. Разные версии **одного** телепорта имеют разные номера.
3. Каждый телепорт однозначно характеризуется некоторым множеством простых чисел. Далее мы рассматриваем какой-нибудь фиксированный телепорт и будем считать, что он характеризуется множеством $X = \{p_1, p_2, \ldots, p_m\}$ (считаем, что $p_1 < p_2 < \cdots < p_m$).
4. Известно, что номер любой версии телепорта делится на каждое простое число из X и только на них, то есть представляется в виде $p_1^{k_1} p_2^{k_2} \ldots p_m^{k_m}$, где $k_i \geqslant 1$. Соответственно, каждая версия телепорта однозначно характеризуется упорядоченным набором из m положительных целых чисел $(k_1, k_2, \ldots, k_m)$.
5. Версия $\alpha = (k_1, k_2, \ldots, k_m)$ была построена раньше версии $\beta = (l_1, l_2, \ldots, l_m)$, если и только если существует такое i $(0 \leqslant i < m)$, что $k_1 = l_1$, $k_2 = l_2$, $\ldots$, $k_i = l_i$, а $k_{i+1} < l_{i+1}$. Можно сказать, что версии отсортированы лексикографически по своим наборам.

Формат входных данных

Единственная строка ввода содержит два числа — номер версии a некоторого телепорта и число N, характеризующее платформу $(1 \leqslant a \leqslant N \leqslant 10^{18})$.

Формат выходных данных

Если существует версия того же телепорта, которая была построена сразу после версии a, выведите её номер. В противном случае выведите «`-1`».

Примеры

wtflex.in	wtflex.out
6 13	12
12 13	-1