Задача 01. (a, b)-башня

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

Маленький Артём задался вопросом: как, записав всего несколько цифр, получить очень большое число? Конечно, один из способов — написать башню из степеней. Как известно, башня из степеней вычисляется сверху вниз: в выражении вида x^{y^z} сначала вычисляется y^z, а затем число x возводится в полученную степень. Например, короткая запись 2^{3^4} равна огромному числу $2^{3 \cdot 3 \cdot 3 \cdot 3} = 2^{81} = 2\,417\,851\,639\,229\,258\,349\,412\,352$.

Артём выписал целые числа от a до b по диагонали, двигаясь вверх и вправо. Получилась башня из степеней:

$$a^{(a+1)^{\cdot^{\cdot^{\cdot^{(b-1)^b}}}}}.$$

Помогите ему выяснить, чему равна последняя десятичная цифра этого — возможно, огромного — числа.

Формат входных данных

В первой строке записано два целых числа a и b — параметры башни ($1 \leqslant a \leqslant b \leqslant 10$).

Формат выходных данных

Выведите одно число — последнюю десятичную цифру числа

$$a^{(a+1)^{\cdot^{\cdot^{\cdot^{(b-1)^b}}}}}.$$

Примеры

abtower.in	abtower.out
2 4	2
3 3	3

Пояснения к примерам

В первом примере получается число $2^{3^4} = 2^{81} = 2\,417\,851\,639\,229\,258\,349\,412\,35\mathbf{2}$. Последняя десятичная цифра этого числа равна 2.

Во втором примере получается просто число **3**, последняя и единственная десятичная цифра которого равна 3.

Задача 02. Исправление адресов

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

В почтовой системе Города Роботов адреса записываются как последовательности из девяти десятичных цифр (000 000 000 – 999 999 999). Всего в городе n различных существующих адресов, по которым могут быть доставлены письма.

В главное почтовое отделение Города Роботов прибыли m конвертов. На каждом из конвертов написан адрес, также состоящий из девяти десятичных цифр. Однако некоторые адреса могут быть написаны с ошибками, а некоторые конверты даже могли попасть в почтовое отделение города по ошибке.

Ваша задача — восстановить правильное написание адресов. Считается, что адрес s на конверте *можно исправить* на адрес t в Городе Роботов, если s и t либо совпадают, либо различаются ровно в одной цифре из девяти. Для каждого конверта, адрес на котором исправляется однозначно, выведите исправленный адрес, по которому нужно доставить этот конверт. Если адрес на конверте нельзя исправить ни на какой адрес в городе, поменяв не более чем одну цифру — или, наоборот, существует больше одного адреса в городе, на который его можно исправить — следует также это выяснить.

Формат входных данных

В первой строке ввода задано целое число n — количество различных адресов в Городе Роботов ($1 \leqslant n \leqslant 100$). В следующих n строках записаны сами адреса, по одному на строке. Каждый адрес состоит ровно из девяти десятичных цифр без пробелов.

Следующая строка ввода содержит целое число m — количество конвертов ($1 \leqslant m \leqslant 100$). В следующих m строках записаны адреса, указанные на конвертах, по одному на строке. Каждый адрес также состоит ровно из девяти десятичных цифр без пробелов.

Имейте в виду, что, хотя все n адресов, существующих в Городе Роботов, различны, на нескольких или даже всех m конвертах может быть написан один и тот же адрес.

Формат выходных данных

Выведите m строк. В i-й из этих строк выведите адрес в Городе Роботов, по которому необходимо доставить i-й конверт. Помните, что адрес должен состоять ровно из девяти десятичных цифр без пробелов. Если подходящих адресов нет, выведите вместо адреса число `-1`. Если же подходящих адресов больше одного, выведите вместо адреса число `-2`.

Примеры

error.in	error.out
2 000000000 999999999 3 012345678 999999989 000000000	-1 999999999 000000000

error.in	error.out
3 123123123 124123123 134123123 2 124123123 135123123	-2 134123123

Пояснения к примерам

В первом примере (слева):

- в первом адресе на конверте пришлось бы поменять почти все цифры, чтобы получить какой-то из существующих в городе адресов;
- во втором адресе нужно поменять одну цифру;
- в третьем адресе вообще ничего менять не нужно.

Во втором примере (справа):

- первый адрес на конверте может быть исправлен на все три существующих адреса;
- второй адрес восстанавливается однозначно.

Задача 03. Воздушные змеи

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

Назовём *воздушным змеем* четырёхугольник, состоящий из двух равных прямоугольных треугольников, склеенных по гипотенузе так, чтобы получилась симметричная фигура относительно линии склейки. Эту линию будем называть *осью* змея.

На плоском столе лежало несколько воздушных змеев из бумаги. Каждый из них разрезали по оси. После этого получившиеся треугольники, возможно, подвинули и повернули (но не отразили) независимо друг от друга. Наконец, некоторые треугольники убрали со стола.

Зная, где находятся оставшиеся треугольники, выясните, какое минимальное количество воздушных змеев могло быть на столе исходно, а также из каких треугольников они могли состоять. Треугольники на столе могут накладываться произвольно.

Формат входных данных

В первой строке записано целое число n — количество треугольников ($1 \leqslant n \leqslant 100\,000$). Каждая из следующих n строк содержит шесть чисел x_1, y_1, x_2, y_2, x_3, y_3 — координаты трёх вершин очередного треугольника. Вершины треугольника могут быть перечислены в произвольном порядке. Гарантируется, что все треугольники — прямоугольные, а все координаты — целые числа, не превосходящие 10^6 по абсолютной величине.

Формат выходных данных

В первой строке выведите число m — минимальное количество воздушных змеев, которые могли находиться на столе исходно. Далее выведите возможное описание m воздушных змеев в любом порядке, по одному змею в строке. Каждое описание должно иметь вид k_1-k_2. Здесь k_1 и k_2 — номера треугольников, из которых состоит воздушный змей, в любом порядке. Треугольники на столе пронумерованы целыми числами от 1 до n в порядке их следования во входных данных. Убранные со стола треугольники обозначаются номером 0. Каждый треугольник на столе должен встречаться в описании ровно один раз.

Если возможных ответов с минимальным m несколько, выведите любой из них.

Пример

kites.in	kites.out
6 0 0 0 1 1 1 0 0 1 0 1 1 1 1 1 2 3 2 0 0 0 2 1 2 3 4 0 0 -8 6 0 5 0 0 -10 0	4 3-4 0-5 2-1 6-0

Задача G. Игра с окружностью

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

На листе бумаги нарисована окружность, на которой отмечено n различных точек. Гидран и Диореп играют в игру, делая ходы по очереди. Ход состоит в том, чтобы выбрать две различные отмеченные точки и провести между ними хорду; это можно сделать, только если новая хорда не имеет общих точек (в том числе и концов) со всеми ранее проведёнными хордами. Проигрывает тот, кто не может сделать ход. Первым ходит Гидран. Кто выигрывает при правильной игре?

Формат входных данных

На ввод подаётся несколько строк. Первая из них содержит целое число t — количество тестовых случаев ($1 \leqslant t \leqslant 50$). Каждая из следующих t строк описывает один тестовый случай. Описание тестового случая состоит из одного целого числа n — количества отмеченных точек на окружности ($3 \leqslant n \leqslant 10^9$).

Формат выходных данных

Для каждого заданного n выведите на отдельной строке «`53`», если выигрывает Гидран, и «`34`», если выигрывает Диореп. Ответы должны быть выведены в том же порядке, в котором числа n заданы во вводе.

Пример

circgame.in	circgame.out
2 4 5	53 34

Задача 04. Ханойские башни 2

Имя входного файла:	`hanoi2.in`
Имя выходного файла:	`hanoi2.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мебибайт

Ханойские башни — популярная головоломка. Она состоит из трёх стержней и n дисков различного диаметра. В центре каждого диска находится отверстие для того, чтобы нанизывать диск на стержень. Изначально все диски нанизаны на первый стержень, причём сверху расположен самый маленький диск, под ним диск побольше и так далее; снизу лежит самый большой диск. За одно перекладывание разрешается снять один диск сверху любого стержня и нанизать его сверху на другой стержень. При этом нельзя нанизывать диск на стержень, на котором верхний диск имеет меньший диаметр; на пустой стержень можно нанизывать любой диск. Цель — перекладывать диски таким образом, чтобы перенести их все на третий стержень.

Напишите программу, которая решает головоломку в общем виде: по заданному количеству дисков и данному начальному положению находит последовательность перекладываний, позволяющую перевести их в данное конечное положение. Количество перекладываний должно быть минимально возможным.

Формат входных данных

В первой строке записано целое число n — количество дисков ($1 \leqslant n \leqslant 15$). Во второй строке записано n целых чисел через пробел — номера стержней, на которых изначально лежат диски. В третьей строке также записано n целых чисел через пробел — номера стержней, на которых должны оказаться диски. В этих строках диски перечислены от маленьких к большим. Если несколько дисков в начальном или конечном положении лежат на одном стержне, это означает, что внизу лежит самый большой из них, на нём — самый большой из оставшихся, и так далее; наверху лежит самый маленький диск.

Формат выходных данных

В первой строке выведите m — минимальное число перекладываний. В следующих m строках выведите описания операций перекладывания по одному на строке. Описание перекладывания должно состоять из двух целых чисел, разделённых пробелом — номеров стержня, с которого снимается диск, и стержня, на который он нанизывается. Если существует несколько способов решить головоломку за минимальное число перекладываний, можно выводить любой из них. Если решения не существует, выведите в первой строке число -1.

Примеры

hanoi2.in	hanoi2.out
2 1 1 3 3	3 1 2 1 3 2 3
3 1 2 3 2 3 1	5 1 2 3 1 2 1 2 3 1 2

Задача 05. Деление

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

Алиса ходит в кружок по математике. Недавно на занятии кружка преподаватель Глеб Михайлович рассказывал про $\mathbb{Z}_m$ — *кольцо вычетов по модулю* m. $\mathbb{Z}_m$ состоит из m целых чисел: 0, 1, …, $m-1$. Складывать, вычитать и умножать такие числа совсем просто: нужно сделать операцию как с обычными числами, а потом взять остаток от деления ответа на m. Например, при $m=10$ сложение работает так: $2+3=5$, $3+5=8$, $8+5=3$ (поскольку ответ с обычными числами равен 13, а 3 — это остаток от деления 13 на 10). Вычитание работает так: $3-2=1$, $1-2=9$ (ответ с обычными числами равен -1, а 9 — это остаток от деления -1 на 10). Умножение работает так: $2\cdot 3=6$, $6\cdot 8=8$ (с обычными числами получилось бы 48, а 8 — это остаток от деления 48 на 10).

Иначе обстоит дело с делением в $\mathbb{Z}_m$. Деление определяют как операцию, обратную к умножению (так же, как вычитание — операция, обратная к сложению). А именно, результат деления a/b равен c, если $c\cdot b=a$. К сожалению, это определение не всегда однозначно: таких чисел c иногда несколько, а иногда нет вообще. Например, если $m=10$, то $6/2$ может дать как результат 3 (поскольку $3\cdot 2=6$), так и результат 8 (поскольку $8\cdot 2=6$; напомним, что все равенства в этой задаче записываются не для обычных чисел, а для чисел из $\mathbb{Z}_m$). А вот выражение $7/2$ вообще не может дать никакого результата: среди чисел 0, 1, …, 9 нет такого числа x, что $x\cdot 2=7$.

Глеб Михайлович достал с полки какой-то старый калькулятор и объяснил, что этот калькулятор работает не с обычными числами, а как раз с $\mathbb{Z}_m$, причём $m=2^{32}=4\,294\,967\,296$. Сложение, умножение и вычитание он выполняет правильно. А вот при делении, если результат неоднозначен, калькулятор может с одинаковой вероятностью выдать любой из возможных ответов. Если же деление вообще не может дать никакого результата, на экране калькулятора появляется надпись «`Error`» (ошибка).

Затем Глеб Михайлович передал калькулятор ребятам как наглядное пособие. Когда калькулятор попал в руки к Алисе, она ввела число x и стала делить его на два, результат — ещё на два, и так далее. Через некоторое время на экране возникла надпись «`Error`». Алиса снова ввела x и попробовала ещё раз: действительно, калькулятор работал точно так, как сказал преподаватель.

Теперь Алисе интересно, какие числа могут получаться на калькуляторе, если ввести x, после чего ноль или более раз совершить деление на два. Она составила список чисел y_1, y_2, …, y_n, которые ей было бы интересно получить из x. Для каждого из этих n чисел выясните, возможно ли это.

Формат входных данных

В первой строке входных данных записаны два целых числа x и n — начальное число и количество интересующих Алису чисел ($0\leqslant x\leqslant 4\,294\,967\,295$, $1\leqslant n\leqslant 50$). Во второй строке заданы n целых чисел y_1, y_2, …, y_n — числа, которые интересуют Алису ($0\leqslant y_i\leqslant 4\,294\,967\,295$).

Формат выходных данных

Выведите n строк. Если число y_i может получиться из x после нуля или более делений на два в кольце вычетов по модулю 2^{32}, выведите в i-й строке слово «`Yes`», иначе выведите в ней «`No`».

Примеры

division.in	division.out
16 5 4 3 2 32 16	Yes No Yes No Yes
123456789 3 123 456 789	No No No

Пояснения к примерам

В первом примере из числа 16 могут получиться, например, числа 16, 8, 4, 2 и 1. Могут получиться и некоторые другие числа, но 3 и 32 получить не удастся.

Во втором примере из числа 123 456 789 нельзя получить никакое другое число: после первого же деления калькулятор выведет «`Error`».

Задача 06. Таймер

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

Рассмотрим следующую конструкцию таймера. Таймер состоит из двух цилиндров: внутреннего и внешнего, который может вращаться вокруг внутреннего. На поверхность внутреннего цилиндра нанесены отметки и числа. Отметки соответствуют минутам. Внешний же цилиндр покрывает весь внутренний, оставляя видимым только маленькое окошко. Время, которое показывает таймер, можно узнать, посмотрев в центр этого окошка.

Всего на внутреннем цилиндре 60 отметок. Каждая пятая из них отмечена соответствующим числом: 0, 5, 10, 15, …, 55. Расстояния между любыми двумя соседними отметками, включая расстояние между последней и первой отметками, равны. Ширина окошка ровно в пять раз больше расстояния между соседними отметками.

Исходно в центре окошка во внешнем цилиндре находится отметка 0 на внутреннем цилиндре. Чтобы выставить время на таймере, внешний цилиндр поворачивают вокруг внутреннего по часовой стрелке. После этого он медленно поворачивается против часовой стрелки, пока опять не достигнет отметки 0, после чего останавливается. На таймере можно выставить любое время, строго меньшее, чем 60 минут.

Анна учит своего домашнего робота Берту узнавать время, которое показывает таймер. Берта может фотографировать окошко таймера. Помогите Анне написать программу для Берты, чтобы определять время по фотографии.

Фотография окошка — это растровое изображение 60 пикселов в ширину и 12 пикселов в высоту. Будем для простоты считать, что фотография — это идеальное изображение видимой части внутреннего цилиндра, уложенной на плоскость. В таком случае расстояние между соседними отметками будет ровно 12 пикселов. Также для простоты предположим, что фотография сделана в момент, когда время, которое показывает таймер, кратно 5 секундам. Поскольку окошко не совсем прямоугольной формы, небольшие уголки на фотографии всегда закрыты внешним цилиндром; точную форму уголков можно увидеть в примере. Все остальные пикселы фотографии либо полностью белые, либо полностью чёрные.

При указанных предположениях каждая отметка выглядит как чёрный квадрат 2×2 в первых двух строках фотографии; настоящий центр отметки находится между двумя столбцами этого квадрата. Каждое число печатается чёрным шрифтом размера 8×8 в строках с 4-й по 11-ю. Каждое число (из одной или двух цифр) горизонтально выровнено так, что центр соответствующего блока размера 8×8 или 16×8 расположен по центру отметки, при которой написано это число. Все пикселы, не покрытые уголками, отметками или цифрами, отображаются как белые.

Ниже показаны изображения отдельных цифр от 0 до 9 в используемом шрифте 8×8. Сам шрифт идентичен шрифту, использовавшемуся в видеокартах *Rendition Vérité 1000* в 1990-е годы. Изображения цифр в шрифте можно также скачать по следующему адресу:
`http://acm.math.spbu.ru/171226_b17/timer/`.

```
.xxxxx.. ...xx... .xxxxx.. .xxxxx.. ...xxx.. xxxxxxx. .xxxxx.. xxxxxxx. .xxxxx.. .xxxxx..
xx..xxx. ..xxx... xx...xx. xx...xx. ..xxxx.. xx...... xx...xx. xx...xx. xx...xx. xx...xx.
xx.xxxx. .xxxx... .....xx. .....xx. .xx.xx.. xxxxxx.. xx...... x....xx. xx...xx. xx...xx.
xxxx.xx. ...xx... ...xxx.. ..xxxx.. xx..xx.. .....xx. xxxxxx.. ....xx.. .xxxxx.. .xxxxxx.
xxx..xx. ...xx... .xxx.... .....xx. xxxxxxx. .....xx. xx...xx. ...xx... xx...xx. .....xx.
xxx..xx. ...xx... xx...... xx...xx. ....xx.. xx...xx. xx...xx. ..xx.... xx...xx. xx...xx.
.xxxxx.. .xxxxxx. xxxxxxx. .xxxxx.. ...xxxx. .xxxxx.. .xxxxx.. ..xx.... .xxxxx.. .xxxxx..
........ ........ ........ ........ ........ ........ ........ ........ ........ ........
```

Формат входных данных

Входные данные состоят ровно из 12 строк, каждая из которых содержит ровно 60 символов: фотография окошка таймера. Поскольку окошко не совсем прямоугольной формы, небольшие уголки на фотографии всегда закрыты символами «-» (знак минус); точную форму уголков можно увидеть в примере. Все остальные символы соответствуют изображению внутреннего цилиндра и равны либо «.» (точка) для белых пикселов, либо «X» (большая буква икс) для чёрных пикселов.

Гарантируется, что таймер корректно показывает некоторое время, строго меньшее 60 минут и кратное 5 секундам.

Формат выходных данных

Выведите время, которое показывает таймер, на отдельной строке в формате «MM:SS». Здесь «MM» — две цифры, соответствующие минутам, а «SS» — две цифры, соответствующие секундам.

Пример

timer.in

```
-------X.........XX.........XX.........XX.........-------
----..XX.........XX.........XX.........XX.........XX----
-.............................................................-
X.....................................................XXXXXXX..
XX....................................................XX.......
XX....................................................XXXXXX...
XX......................................................XX..
XX......................................................XX..
XX....................................................XX...XX..
-.....................................................XXXXX..-
----.........................................................----
-------.....................................................-------
```

timer.out

```
07:05
```

Пояснение к примеру

В этом примере центр окошка находится между отметками 10 и 5. Отметка, обозначенная числом 5, хорошо видна справа. Часть нуля из числа 10 можно увидеть слева. Более тщательное изучение фотографии позволяет установить, что время, которое показывает таймер, в точности равно 7 минутам и 5 секундам.

Задача 07. Поиск уникального элемента

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

Это интерактивная задача.

А вы слышали о такой задаче: «Дан массив чисел, в котором все числа кроме одного, встречаются два раза, а оставшееся — один раз. Найти это число.» ?

Вам предстоит решить почти такую же. В этой задаче массив чисел отсортирован, содержит все элементы кроме одного по два раза, оставшийся элемент содержит один раз, но он вам не дан! Вместо этого можно по одному запрашивать его элементы.

Протокол взаимодействия

В начале взаимодействия на вход вашей программе будет подано нечётное число n ($1 \leqslant n \leqslant 199\,999$) — длина массива.

После этого вы можете делать два типа запросов.

- «`?` x». Запросить элемент массива на позиции x ($1 \leqslant x \leqslant n$). Разрешено сделать не более 40 таких запросов. При превышении этого лимита решение получит вердикт «`Wrong Answer`».
- «`!` v». Ответить на задачу. Число v должно быть равно единственному элементу массива, который встречается один раз.

В ответ на запрос первого типа будет получена одна строка, в которой содержится соответствующий элемент массива. Это положительное целое число, не превосходящее 10^9.

После выполнения запроса второго типа решение должно корректно завершиться.

Каждый запрос следует выводить на отдельной строке. Чтобы предотвратить буферизацию вывода, после каждого выведенного запроса 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.

Пример

запросы участника	ответы проверяющей программы	Загаданный массив

`? 1`
`? 5`
`? 3`
`! 2`
	5
	1
	3
	2
		`1 1 2 3 3`

Задача 08. Кротовья нора

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

Кротовья нора состоит из пещерок, соединённых коридорами, причём из каждой пещерки выходит ровно три коридора. Коридорами и пещерками пользуется не только крот — в одной из пещерок обосновался жук-носорог, а в другую только что свалился жук-олень. Жук-олень чует своего соперника и спешит встретиться, чтобы выяснить, кто из них будет главным в кротовьей норе!

Выясните, каким минимальным количеством коридоров потребуется пройти жуку-оленю, чтобы попасть в пещерку, где его поджидает жук-носорог.

Формат входных данных

В первой строке заданы три числа: N ($4 \leqslant N \leqslant 1000$) — количество пещерок в кротовьей норе, X ($1 \leqslant X \leqslant N$) — номер пещерки жука-носорога и Y ($1 \leqslant Y \leqslant N$) — номер пещерки жука-оленя. В последующих N строках стоит по три числа в каждой: в $(i+1)$-й строке стоят номера пещерок, в которые из i-й пещерки ведут коридоры. Каждый коридор упоминается в строках обеих вершин, которые он соединяет. Никакой коридор не соединяет пещерку с самой собой, и никакие две пещерки не соединены более чем одним коридором. Гарантируется, что $X \neq Y$. Все числа во входных данных целые.

Формат выходных данных

Выведите одно целое число — минимальное количество коридоров, которыми требуется пройти, чтобы попасть из пещерки жука-оленя в пещерку жука-носорога. Если это оказалось невозможно, выведите число -1.

Примеры

burrow.in	burrow.out
4 1 4 2 3 4 3 4 1 4 1 2 1 2 3	1
6 2 3 4 5 6 4 5 6 4 5 6 1 2 3 1 2 3 1 2 3	2
8 5 4 2 3 4 1 3 4 1 2 4 1 2 3 6 7 8 5 7 8 5 6 8 5 6 7	-1

Задача 09. Максимум в цикле

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

Ничего хорошего не происходит после двух часов ночи. Но порой ночи так прекрасны (или так ужасны) и длинны, что в голову приходят замечательные идеи. Вот и сегодня профессору Фальсификационной и Геокосмической Академии (ФиГА) пришла в голову чудесная идея, не поддающаяся объяснению того, зачем она нужна кому-то, кроме него самого.

Начнём издалека: профессору очень интересно ездить по дорогам, особенно по кольцевым маршрутам, а ещё лучше — когда на дорогах много развязок, перекрёстков, закусочных, пробок и прочего. Каждая дорога характеризуется для профессора своей *ужасностью* — числом, зависящим от этих интересных черт дороги. Так получилось, что ужасности всех дорог различны.

Сеть дорог представляет собой набор двунаправленных дорог, каждая из которых соединяет какие-то две развязки. Из любой развязки можно добраться по дорогам до любой другой развязки. Нет ни одной дороги, соединяющей какую-либо развязку саму с собой.

Теперь профессор решил называть *ужасающими* те дороги, для которых существует маршрут без повторяющихся дорог, начинающийся и заканчивающийся в одной и той же развязке, в котором эта дорога — самая ужасная.

«А не станут ли все дороги ужасающими...» — подумал профессор, но это была настолько грустная мысль, что додумывать её не захотелось. Поэтому он поручил программистам выяснить, какие же дороги всё-таки не являются ужасающими.

Формат входных данных

В первой строке ввода даны два целых неотрицательных числа n и m — количество развязок и количество дорог ($0 \leqslant n, m \leqslant 10^5$). В следующих m строках даны дороги в виде пар чисел — номеров развязок, которые эти дороги соединяют. Развязки нумеруются целыми числами от 1 до n. Дороги перечислены в порядке увеличения ужасности.

Гарантируется, что из любой развязки можно добраться по дорогам до любой другой развязки, а также что нет ни одной дороги, соединяющей какую-либо развязку саму с собой.

Формат выходных данных

В первой строке выведите количество дорог, которые не являются ужасающими. В следующей строке выведите номера этих дорог через пробел в порядке возрастания. Дороги нумеруются с единицы в том порядке, в котором они заданы во вводе.

Примеры

maxincycle.in	maxincycle.out
4 6 1 2 2 1 2 3 2 3 3 4 4 3	3 1 3 5
4 4 1 2 2 3 3 4 4 1	3 1 2 3

Пояснения к примерам

В первом тесте дорога 2 является ужасающей, потому что она самая ужасная в маршруте из дорог 1 и 2. Аналогично дорога 4 — самая ужасная в маршруте из 3 и 4, а дорога 6 — самая ужасная в маршруте из 5 и 6. Есть и другие кольцевые маршруты без повторяющихся дорог, например, маршрут из дорог 1, 3, 4 и 2, но самой ужасной в них снова является одна из дорог 2, 4 и 6. Дороги 1, 3 и 5 не являются ужасающими.

Во втором тесте есть всего один кольцевой маршрут — из дорог 1, 2, 3 и 4. Самая ужасная дорога в нём имеет номер 4.

Задача 10. Гирлянда

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

Новогодняя гирлянда, установленная за окном Васиного дома, представляет из себя n лампочек, расположенных на одной прямой и пронумерованных подряд числами от 0 до $(n-1)$. Вася, глядя в окно, заинтересовался порядком включения лампочек. После внимательных наблюдений он установил следующее.

Изначально все лампочки выключены. Далее каждую секунду включается одна лампочка. В первую секунду включается лампочка с номером a, во вторую — с номером $a^2 \bmod n$, в третью — с номером $a^3 \bmod n$, ..., в k-ю — лампочка с номером $a^k \bmod n$.

Вася выяснил значения чисел a и n, и теперь его заинтересовал следующий вопрос. Рассмотрим лампочку, включённую в r-ю секунду (она имеет номер $a^r \bmod n$). Сколько лампочек слева от неё, то есть с меньшими номерами, уже включены? Вася считает, что именно свойства этих чисел для разных значений r и обеспечивают гирлянде особую красоту. Однако сам он будет слишком долго вручную их вычислять. Помогите Васе — в ответ на каждый заданный им вопрос о числе r выведите количество лампочек, зажжённых до r-й зажжённой лампочки и находящихся слева от неё.

Формат входных данных

В первой строке входных данных записаны целые числа a, n и q через пробел ($1 \leqslant a < n \leqslant 1\,000\,000$, $1 \leqslant q \leqslant 10\,000$). Следующие q строк содержат по одному целому числу r_i каждая ($1 \leqslant r_i \leqslant 100\,000$) и описывают Васины вопросы. Вася не знает, что будет, когда по этому алгоритму надо будет зажечь уже зажжённую лампочку, поэтому его вопросы таковы, что для любого i за первые r_i секунд никакая лампочка не должна будет загореться дважды.

Формат выходных данных

Выведите q строк, по одному числу в каждой строке. В i-й строке выведите количество лампочек, зажжённых раньше r_i-й зажжённой лампочки и имеющих меньшие номера.

Примеры

garland.in	garland.out
2 3 2 1 2	0 0
5 16 4 1 3 4 2	0 2 0 1
2 8 2 2 2	1 1

Пояснения к примерам

В первом примере порядок зажигания лампочек — 2, 1.

Во втором примере — 5, 9, 13, 1.

В третьем примере — 2, 4, 0.

Задача 11. Охрана артефакта

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

Королю Байтландии недавно удалось добыть мощный артефакт — Меч Вычислимости за Полиномиальное Время. С его помощью король наконец-то сможет принести мир и процветание своему королевству — за полиномиальное время. Однако другие наверняка захотят заполучить предмет с таким потенциалом. Так что артефакт должен быть обеспечен надёжной охраной.

С помощью придворных волшебников удалось построить магическую комнату для безопасного хранения артефакта. Однако для максимальной надёжности внутри комнаты должен находиться стражник. Конечно же, было бы лучше иметь в комнате с артефактом сразу несколько стражников, но её размеры таковы, что внутрь может поместиться лишь один человек. Поэтому король хочет нанять несколько стражников и составить план их дежурства, чтобы в каждый момент времени *ровно один* из них охранял артефакт. Смена стражников происходит мгновенно и не влияет на безопасность. Помимо прочего, артефакт предотвращает старение, поэтому пополнение стражи не предусмотрено — план дежурств должен быть составлен так, чтобы артефакт был под охраной одного и того же набора стражников вечно.

Всего доступно для найма n отважных и способных людей. Кандидат номер i может стоять на дежурстве не более a_i часов подряд, а после дежурства должен отдыхать не менее b_i часов подряд. Само собой, король хочет минимизировать число нанятых стражников, ведь чем больше стражников — тем больше издержки на зарплату и социальный пакет и тем выше вероятность предательства. Кроме того, по правилам профсоюза стражников никому из них не позволяется заступать на дежурство более одного раза между двумя последовательными дежурствами любого из остальных стражников (одним выходом на дежурство считается охрана артефакта в течение некоторого количества часов подряд).

Итак, вам, как советнику короля, необходимо нанять минимально возможное число стражников, чтобы было возможно организовать вечную и непрекращающуюся охрану артефакта, выполнив все вышеперечисленные требования. Удачи, или именно вам и придётся провести всю следующую вечность на дежурстве!

Формат входных данных

Первая строка входных данных содержит одно целое число n — количество кандидатов ($2 \leqslant n \leqslant 10^5$). Далее следуют n строк: i-я из них содержит два целых положительных числа a_i и b_i, разделенных пробелом — максимальное время непрерывной работы и минимальное время непрерывного отдыха i-го кандидата. Эти значения не превосходят 10^{12}.

Формат выходных данных

Выведите одно число — минимально возможное необходимое число стражников. Если подходящее расписание невозможно составить, используя доступных кандидатов, выведите «-1» (без кавычек).

Примеры

artifact.in	artifact.out
5 1 5 2 4 3 4 2 3 4 4	3
2 1 3 1 4	-1

Задача 12. Число из спичек

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

Вы умеете выкладывать на столе некоторые цифры от 0 до k с помощью какого-то количества спичек. Ваша задача — построить максимально большое число из n спичек. Например, если вам нужно 6 спичек для цифры 0, 7 спичек для цифры 1 и 8 спичек для цифры 2, то максимальное число, которое можно сложить из 21 спички — это 210.

Формат входных данных

В первой строке даны числа n и k ($1 \leqslant n \leqslant 50$, $0 \leqslant k \leqslant 9$). Во второй строке задано $k+1$ число от 1 до 50 — количество спичек, требуемое на цифры от 0 до k. Гарантируется, что из n спичек можно выложить хотя бы одну из цифр.

Формат выходных данных

Выведите максимальное число, которое можно построить. Лидирующие нули в выводе недопустимы. Можно использовать не все n спичек, а только часть.

Пример

matches.in
21 2 6 7 8
matches.out
210

matches.in
30 2 5 23 24
matches.out
20

matches.in
1 3 1 5 3 2
matches.out
0

matches.in
50 9 1 1 1 1 1 1 1 1 1 1
matches.out
99999999999999999999999999999999999999999999999999