Задача A. Покрытие

Имя входного файла:	`cliquecovering.in`
Имя выходного файла:	`cliquecovering.out`
Ограничение по времени:	3 секунды (5 для Java)
Ограничение по памяти:	256 мебибайт

Дан неориентированный граф G без петель и кратных рёбер.

Кликой в графе называется **непустое** множество вершин, внутри которого проведены рёбра между всеми парами вершин.

Анти-кликой в графе называется **непустое** множество вершин, внутри которого нет ни одного ребра.

Ваша задача — для каждой пары целых чисел (p, q) проверить, можно ли разбить граф на p клик и q анти-клик.

Формат входных данных

В первой строке даны два числа n и m ($1 \leqslant n \leqslant 20$) — количество вершин и рёбер в графе соответственно. В следующих m строках даны пары чисел a и b ($1 \leqslant a, b \leqslant n$) — номера вершин, соединённых очередным ребром.

Гарантируется, что в графе нет петель и кратных рёбер.

Формат выходных данных

Выведите $n + 1$ строку. В каждой строке выведите $n + 1$ символ без пробелов. Если нумеровать строки и символы в строке с нуля, то в i-й строке j-й символ должен быть равен «`1`», если граф можно разбить на i клик и j анти-клик, и «`0`» в противном случае.

Пример

cliquecovering.in	cliquecovering.out
5 4 1 2 1 3 2 3 3 4	000111 011110 011100 111000 110000 100000

Задача B. Доставка воды к хижинам

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

Поверхность космической станции имеет торическую форму. На ней расположено три хижины, населенных биороботами, и три колодца: с живой водой, с мертвой водой и минералкой. Доставка воды осуществляется монорельсовыми тележками каждый день, поэтому во избежание столкновений нужно провести пути от каждого дома до каждого колодца таким образом, чтобы они не пересекались.

Для целей задачи, будем представлять тор как квадратную сетку, у которой склеены противоположные стороны. Введем на сетке систему координат так, чтобы верхний левый угол сетки имел координаты $(0, 0)$, а правый нижний $(8, 8)$. Формально, склейка означает, что мы отождествляем точки $(x, 8)$ и $(x, 0)$, а также точки $(8, y)$ и $(0, y)$. В результате у любой точки (x, y) есть ровно 4 соседа — точки с координатами $(x \pm 1 \bmod 8, y \pm 1 \bmod 8)$.

Вам даны координаты трех хижин и трех колодцев на поверхности станции. Все 6 точек попарно различны. Надо провести 9 путей — от каждой хижины до каждого колодца. Два пути могут пересекаться только по хижине или колодцу, являющемуся одним из концов обоих путей. В частности, каждый путь проходит ровно через одну хижину и один колодец, которые являются его началом и концом.

Для прокладывания путей тор покрывается более мелкой сеткой. Формально, это значит, что вы можете выбрать положительное целое число d, и ввести новую систему координат, в которой у точки (x, y) соседями являются точки с координатами $(x \pm 1 \bmod (8 \cdot d), y \pm 1 \bmod (8 \cdot d))$, а точка с координатами (x, y) в старой системе координат имеет координаты $(d \cdot x, d \cdot y)$. Вы можете остаться в старой системе координат, выбрав $d = 1$.

Необходимо выбрать число d и вывести пути в новой системе координат. Соседние клетки пути должны быть соседними по стороне клетками в новой системе координат. Пути не должны пересекаться, кроме как по концам.

Формат входных данных

В первых трех строках дано два целых числа hx_i, hy_i — координаты i-той хижины с исходной системе координат $(1 \leqslant hx_i, hy_i \leqslant 6)$. В каждой из следующих трех строк даны два целых числа sx_i, sy_i — координаты i-того колодца с исходной системе координат $(1 \leqslant sx_i, sy_i \leqslant 6)$.

Формат выходных данных

В первой строке выведите во сколько раз вы решили уменьшить размер клетки сетки — число d $(1 \leqslant d \leqslant 1000)$. После этого выведите 9 строк. В каждой строке необходимо вывести один из путей. Пути надо выводить в следующем порядке

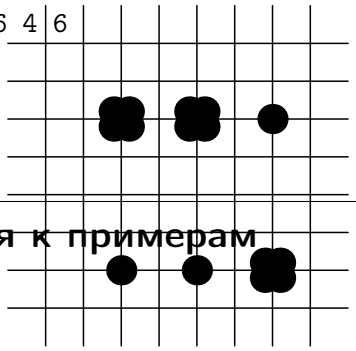
- От 1 домика к 1 колодцу
- От 1 домика ко 2 колодцу
- От 1 домика к 3 колодцу
- От 2 домика к 1 колодцу
- От 2 домика ко 2 колодцу
- От 2 домика к 3 колодцу
- От 3 домика к 1 колодцу
- От 3 домика ко 2 колодцу
- От 3 домика к 3 колодцу

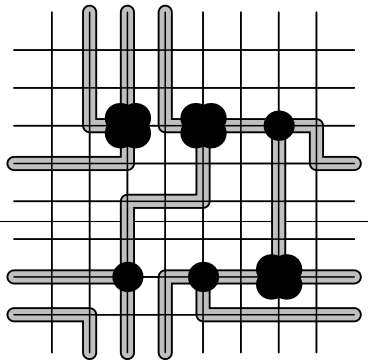
В каждой из строк сначала выведите длину пути l, потом l пар целых чисел x_j, y_j $(0 \leqslant x_j, y_j < 8)$ — последовательность координат узлов сетки в пути. Первая точка пути должна совпадать с координатами хижины в новой системе координат, последняя — с координатами колодца в новой системе координат. Две соседние точки пути должны являться соседними в новой системе координат.

Пример

hovels.in
2 2 4 2 6 6 6 2 2 6 4 6

hovels.out
1 7 2 2 2 3 1 3 0 3 7 3 7 2 6 2 5 2 2 2 1 2 0 2 7 2 6 11 2 2 1 2 1 1 1 0 1 7 0 7 7 7 6 7 5 7 4 7 4 6 3 4 2 5 2 6 2 7 4 2 4 3 4 4 3 4 2 4 2 5 2 6 7 4 2 3 2 3 1 3 0 3 7 3 6 4 6 5 6 6 6 5 6 4 6 3 6 2 5 6 6 7 6 0 6 1 6 2 6 3 6 6 5 6 4 6





Пояснения к примерам

Задача C. Археологические раскопки

Имя входного файла:	`intersect-hull.in`
Имя выходного файла:	`intersect-hull.out`
Ограничение по времени:	4 секунды (6 для Java)
Ограничение по памяти:	256 мебибайт

Археологи занимаются раскопками древнего города «Двоя». Цель раскопок — оценить площадь города и сделать всю предполагаемою территорию древнего города национальным наследием, неприкосновенным культурным заповедником. Для простоты рассчётов археологи приближают территорию раскопок плоскостью. Считается, что если в точках $(x_1, y_1), \ldots, (x_k, y_k)$ были найдены древние артефакты, то город занимал ровно всю территорию выпуклой оболочки данных k точек. В процессе раскопок последовательно случаются события вида:

1. В точке (x, y) откопали новый древний артефакт.
2. Артефакт, откопанный ранее, признан обычным мусором.
3. Государство рассматривает план постройки прямой дороги, идущей ровно по прямой $ax + by + c = 0$. Им интересно оценить предполагаемый урон культурному наследию, который нанесёт постройка данной дороги. Величину урона оценивают длиной пересечения предполагаемой территории древнего города и дороги.

До момента начала раскопок никаких артефактов найдено не было. Вам даны события в том порядке, в котором они происходили. Для каждого события вида «план дороги» найдите длину пересечения.

Формат входных данных

В первой строке задаётся число событий n ($1 \leqslant n \leqslant 50\,000$).

Каждая из следующих n строк содержит описание события:

1. `+ x y` — новый древний артефакт.
2. `- i` — артефакт, заданный i-м из n событий, оказался мусором.
3. `? a b c` — найти длину пересечения с дорогой $ax + by + c = 0$ ($a^2 + b^2 > 0$).

События нумеруются числами от 1 до n. В одной точке может появиться несколько разных артефактов. Каждый артефакт признаётся мусором не более одного раза. Все координаты x, y, а так же числа a, b, c — целые числа, не превосходящие 10^8 по модулю.

Формат выходных данных

Для каждого запроса вида `? a b c` на отдельной строке выведите одно вещественное число — ответ. Относительная или абсолютная погрешность не должна превышать 10^{-9}.

Пример

intersect-hull.in	intersect-hull.out
12 + -1 0 + 1 0 + 0 1 + 0 -1 + 0 0 ? 1 -1 0 - 2 ? 1 -1 0 - 4 ? 1 -1 0 + 10 10 - 11	1.414213562373095145 0.707106781186547573 0.000000000000000000

Задача D. Воздушные змеи

Имя входного файла:	`kites.in`
Имя выходного файла:	`kites.out`
Ограничение по времени:	2 секунды (3 для Java)
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 E. 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 нечетное 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 который 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 F. Разносчик пиццы

Имя входного файла:	pizza.in
Имя выходного файла:	pizza.out
Ограничение по времени:	2 секунды (3 для Java)
Ограничение по памяти:	256 мебибайт

Вы подрабатываете разноской пиццы. У вас есть рюкзак размера S и огромный заказ на n пицц, i-я их которых имеет размер a_i. Разумеется, доставить пиццу требуется как можно скорее. К сожалению, у вас нет ни машины, ни друзей, которые могли бы помочь, так что единственный способ перевозки — распределить все пиццы в стопки размера не более S каждая и доставлять стопки по очереди. Вам надо распределить все пиццы из заказа в минимально возможное количество стопок.

Формат входных данных

Входной файл состоит из t тестов ($1 \leqslant t \leqslant 10$). Первая строка файла содержит число t, далее следуют описания тестов. Каждый тест описывается двумя строчками: на первой располагаются целые числа n ($1 \leqslant n \leqslant 20$) и S ($1 \leqslant S \leqslant 10^9$), на второй располагаются целые числа $a_1, a_2, \ldots, a_n$ ($1 \leqslant a_i \leqslant S$).

Формат выходных данных

Для каждого теста выведите на отдельной строке минимальное число стопок m, а на следующих m строчках — описание стопок. i-я из последующих строк должна содержат количество пицц в i-й стопке k_i и список из k_i номеров пицц. Каждая пицца должна встречаться ровно в одной стопке. Если есть несколько оптимальных решений, выведите любое из них.

Примеры

pizza.in	pizza.out
3 1 10 10 2 10 10 10 4 10 5 7 5 7	1 1 1 2 1 1 1 2 3 1 2 2 1 3 1 4

Задача G. Перевёрнутая цепная дробь

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

Конечная цепная дробь — это последовательность вида $[a_0; a_1, a_2, \ldots, a_n]$. На элементы цепной дроби и её размер накладываются следующие ограничения:

- n — неотрицательное конечное целое число,
- элементы a_0, a_1, a_2, …, a_n — целые числа,
- $a_i > 0$ при $i > 0$,
- $a_n > 1$, если $n > 0$.

Эти ограничения позволяют установить взаимно однозначное соответствие между рациональными числами и конечными цепными дробями: каждому рациональному числу x соответствует единственная цепная дробь $[a_0; a_1, a_2, \ldots, a_n]$ такая, что

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}.$$

Для обозначения соответствия используется знак равенства: $x = [a_0; a_1, a_2, \ldots, a_n]$. Например,

$$\frac{17}{25} = 0 + \frac{1}{\frac{25}{17}} = 0 + \frac{1}{1 + \frac{8}{17}} = 0 + \frac{1}{1 + \frac{1}{\frac{17}{8}}} = \mathbf{0} + \frac{1}{\mathbf{1} + \frac{1}{\mathbf{2} + \frac{1}{\mathbf{8}}}},$$

поэтому мы пишем $\frac{17}{25} = [0; 1, 2, 8]$.

Вам дано рациональное число x ($0 < x \leqslant \frac{1}{2}$). Пусть $x = [0; a_1, a_2, \ldots, a_n]$. Найдите рациональное число, равное $[0; a_n, a_{n-1}, \ldots, a_1]$.

Формат входных данных

В единственной строке даны два положительных целых числа p и q ($1 \leqslant p < q \leqslant 10^9$). Гарантируется, что $\frac{p}{q}$ — несократимая дробь, причём $0 < \frac{p}{q} \leqslant \frac{1}{2}$.

Формат выходных данных

Выведите искомое число в виде несократимой дроби. Числитель и знаменатель дроби следует выводить через пробел.

Примеры

reversed-fraction.in	reversed-fraction.out
3 7	2 7
2 7	3 7
2 5	2 5

Задача H. Техническая поддержка

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

Ярослав работает агентом технической поддержки социальной сети «НаСвязи». Ярослав лучший в своем деле. Он блестяще отвечает на самые каверзные вопросы от пользователей соцсети.

Сегодня один из пользователей задал такой вопрос: «Рассмотрим все квадратные уравнения вида $x^2 + px + q = 0$, в которых $p + q = 218$. Сколько их них имеют целочисленные корни?». Так как Ярослав лучший в своем деле, он решил сразу подготовиться к ответу на все подобные вопросы. Он заменил число 218 на произвольное число A и написал программу, принимающую A в качестве параметра и отвечающую на такой вопрос. А вам слабо?

Формат входных данных

В единственной строке дано целое число A, по модулю не превосходящее $2 \cdot 10^9$.

Формат выходных данных

Если уравнений с целыми корнями конечное число, выведите их количество, иначе выведите слово «`infinity`» без кавычек.

Примеры

sqr-equations.in	sqr-equations.out
218	4
-9	4

Задача I. Машина со стеком и продпрограммами

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

Рассмотрим следующую машину и язык для её программирования. Есть стек, в котором в каждый момент может быть занято от 0 до 5 ячеек. Изначально стек пуст. Кроме того, есть программа, которая записывается как строка и выполняется последовательно слева направо.

Символы строки имеют следующие значения:

- `0..9` — положить значение, равное соответствующей цифре, на верх стека.
- `+-` — арифметические действия: взять с верха стека второй операнд, взять с верха стека первый операнд, положить на верх стека результат операции.
- Блок `[<program>]a..z` — определяет подпрограмму `program` и называет её одной маленькой буквой английского алфавита.
- `a..z` — вызов подпрограммы, соответствующей букве.

В **момент определения** подпрограммы или всей программы сохраняется её *простой текст*: все вызовы других подпрограмм внутри неё заменяются на простой текст этих подпрограмм, сохранённый ранее. Заметьте, что в простом тексте могут встречаться только цифры и знаки арифметических действий.

Блоки из квадратных скобок могут быть вложены друг в друга. При вызове подпрограммы её определение ищется сначала в самом вложенном блоке из квадратных скобок, содержащем этот вызов, затем в блоке предыдущей вложенности, и так далее до уровня всей программы. Из определений в одном и том же блоке выбирается то, которое встретилось в исходной строке позже всего, но до вызова.

По данной программе выведите, что окажется в стеке после её выполнения.

Формат входных данных

Первая и единственная строка ввода содержит от 1 до 160 символов включительно. Гарантируется, что эта строка задаёт корректную программу для описанной машины, в том числе, при выполнении программы не будет происходить переполнения или исчерпания стека.

Формат выходных данных

В первой строке выведите число s — количество значений в стеке после выполнения программы. Во второй строке выведите сами эти числа через пробел, начиная **с верха** стека.

Примеры

stacksub.in	stacksub.out
23+	1 5
23-	1 -1
[9+]a9aaaaa	1 54

Задача J. Нарисуй прямой обход

Имя входного файла:	`xlrto2d.in`
Имя выходного файла:	`xlrto2d.out`
Ограничение по времени:	2 секунды
Ограничение по памяти:	32 мебибайта (48 для Java)

Из википедии мы знаем:

Двоичное дерево поиска (англ. binary search tree, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- *Оба поддерева — левое и правое — являются двоичными деревьями поиска.*
- *У всех узлов левого поддерева произвольного узла x значения ключей данных меньше, нежели значение ключа данных самого узла x.*
- *В то время, как значения ключей данных у всех узлов правого поддерева (того же узла x) больше, нежели значение ключа данных узла x.*

Рассмотрим следующую процедуру вывода BST:

```
void outTree( Node* v ) {
  if (v == 0) return;
  cout << v->x << " ";
  outTree(v->left);
  outTree(v->right);
}
outTree(root);
```

Вам дан вывод непустого дерева, сделанный этой функцией. Известно, что по этому выводу можно однозначно восстановить структуру исходного дерева. Преобразуйте его в красивую 2D картинку.

Обозначим 2D-изображение дерева с корнем в `v` как `rect(v)`. `rect(v)` — прямоугольник, получаемый вызовом рекурсивной функции. Возьмём `rect(v.left)`, `rect(v.right)` и строку `s`, задающую `v.x` (`s` тоже прямоугольник, её высота 1, а длина равна длине десятичного представления числа `v.x`). Нарисуем слева направо `rect(v.left)`, `s`, `rect(v.right)` таким образом, что правый верхний угол `rect(v.left)` совпадает с левым нижним углом `s` и верхний левый угол `rect(v.right)` совпадает с правым нижним углом `s`. Прямоугольник, ограничивающий полученное изображение, и есть `rect(v)`.

Формат входных данных

В единственной строке вывод функции `outTree`. Ключи дерева — целые числа от 0 до 10^9. Все ключи различны.

Формат выходных данных

Выведите 2D-картинку. Гарантируется, что размер вывода не превосходит 5 мегабайт. Строки не должны содержать хвостовые пробелы. Каждая строка должна оканчиваться переводом строки. В примере пробелы сделаны видимыми, но на самом деле следует выводить обычные пробелы с ASCII-кодом 32.

Пример

xlrto2d.in	xlrto2d.out
2 1 400 3 500	␣2 1␣␣400 ␣␣3␣␣␣500