# Задача A. Основная теорема арифметики

| | |
|---|---|
| Имя входного файла: | `arithm.in` |
| Имя выходного файла: | `arithm.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Основная теорема арифметики гласит, что любое натуральное число $x$ большее единицы может быть единственным образом представлено в виде $x = p_1^{a_1} \cdot \ldots \cdot p_k^{a_k}$, где $p_1 < \ldots < p_k$ — простые числа, $a_1, \ldots, a_k$ — положительные целые числа.

Ваша задача — написать программу, находящую такое представление для заданного числа $x$. Напомним, что натуральное число называется простым, если оно больше единицы и делится только на единицу и на себя.

## Формат входного файла

Входной файл содержит целое число $x$ ($2 \le x \le 2 \cdot 10^9$).

## Формат выходного файла

В выходной файл выведите искомое представление числа $x$. В качестве знака умножения выводите `*` («звездочка»). Сомножитель вида $p^a$ выводите как `p^a`. Если $a = 1$, то выводите просто `p`. Отделяйте сомножители от знака умножения пробелом.

## Примеры

| arithm.in | arithm.out |
|---|---|
| 12 | 2^2 * 3 |
| 16 | 2^4 |
| 36 | 2^2 * 3^2 |

# Задача B. Астрономическая задача

| | |
|---|---|
| Имя входного файла: | `astronomy.in` |
| Имя выходного файла: | `astronomy.out` |
| Ограничение по времени: | 3 секунды |
| Ограничение по памяти: | 64 мегабайта |

Флатландские астрономы собираются начать непрерывное наблюдение за $n$ звездами. Вселенная, где располагается Флатландия — двумерная, поэтому каждая звезда характеризуется двумя координатами (координаты измеряются в парсеках).

Ученые хотят выбрать точку, в которой следует установить наблюдательную станцию. Однако если две звезды будут наблюдаться из этой точки под небольшим углом относительно друг друга, то наблюдать за этими звездами будет неудобно. Поэтому астрономы ищут точку, при наблюдении из которой минимальный угол между двумя звездами будет как можно больше.

Также не следует располагать станцию слишком близко к звезде нельзя. Расстояние до ближайшей звезды должно быть не меньше одного парсека.

Помогите ученым выбрать точку, в которой следует построить станция.

## Формат входного файла

Первая строка входного файла содержит $n$ — количество звезд ($3 \le n \le 10$). Следующие $n$ содержат по два целых числа — координаты звезд (координаты по модулю не превышают $10^3$).

## Формат выходного файла

Выведите два целых числа — координаты точки, в которой ученым следует построить станцию. Если существует несколько решений, выведите любое.

Проверяющая программа будет использовать сравнения вещественных чисел с точностью $10^{-5}$.

## Пример

| astronomy.in | astronomy.out |
|---|---|
| 4<br>0 0<br>10 0<br>0 10<br>10 10 | 5 5 |

# Задача C. Игра с шариками

| | |
|---|---|
| Имя входного файла: | `balls.in` |
| Имя выходного файла: | `balls.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Действие одной очень популярной компьютерной игры, которая часто бывает установлена на карманных компьютерах происходит на квадратном поле размером 11 на 11, разбитом на 121 маленький квадратик.

Изначально в каждом квадратике находится шарик одного из пяти цветов: красного (обозначается символом `R`), синего (`B`), зеленого (`G`), желтого (`Y`), фиолетового (`V`). Назовем *связной областью*, содержащий данный шарик, все шарики, до которых можно добраться из данного, двигаясь каждый раз на один квадратик по вертикали или горизонтали, не выходя за границы игрового поля и проходя только по шарикам того же цвета, что и данный.

При выборе некоторого шарика автоматически выбираются все шарики, лежащие в одной связной области с ним. Если эта связная область содержит хотя бы 2 шарика, то эти шарики исчезают и игроку начисляется $n \cdot (n-1)$ очков, где $n$ — количество шариков в связной области.

Задано начальное расположение шариков. Необходимо для каждого цвета определить, какое максимальное количество очков можно набрать за первый ход, выбрав один шарик такого цвета.

## Формат входного файла

Входной файл содержит 11 строк по 11 символов в каждой — описание игрового поля.

## Формат выходного файла

Для каждого цвета шариков в выходной файл выведите максимальное количество очков, которое можно набрать, выбрав шарик этого цвета. Следуйте формату, приведенному в примере.

## Примеры

| balls.in | balls.out |
|---|---|
| RRRRRBBBGGG<br>RRRRRBBBGGG<br>RRRRRBBBGGG<br>RRRRRBBBGGG<br>RRRRRBBBGGG<br>RRRRRBBBGGG<br>RRRRRBBBGGG<br>YYYYYYYYYYY<br>YYYYYYYYYYY<br>YYYYYYYYYYY<br>VRVRVBVBVGV | R: 1190<br>G: 420<br>B: 420<br>Y: 1056<br>V: 0 |

# Задача D. Две окружности

| | |
|---|---|
| Имя входного файла: | `circles.in` |
| Имя выходного файла: | `circles.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Заданы две окружности. Необходимо найти количество точек с целыми координатами, которые лежат строго внутри обеих окружностей.

## Формат входного файла

Входной файл содержит две строки, каждая из которых описывает одну из окружностей. Описание окружности состоит из двух целых чисел: $x$, $y$ и $r$, соответственно координат ее центра и ее радиуса ($-10^9 \le x, y \le 10^9$, $1 \le r \le 10^3$).

## Формат выходного файла

В выходной файл выведите ответ на задачу.

## Примеры

| circles.in | circles.out |
|---|---|
| 0 0 1<br>0 0 3 | 1 |

# Задача E. Скидка

| | |
|---|---|
| Имя входного файла: | `discount.in` |
| Имя выходного файла: | `discount.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Недавно сеть гиперпупермаркетов Неда-Мол объявила специальную акцию для своих клиентов. Каждый клиент, который покупает в моле что-нибудь действительно дорогое, получает $n$ специальных карточек со скидкой, которые он может использовать, чтобы снизить стоимость покупки. Каждая карточка имеет две стороны, на лицевой стороне написано “$-a_i$”, а на обратной — “$-b_i\%$” ($a_i$ и $b_i$ — целые числа, они могут быть различными на различных карточках).

Пусть начальная стоимость покупки равна $x$. Чтобы уменьшить стоимость покупки, клиент может использовать карточки со скидкой. Карточки можно использовать в произвольном порядке. Каждая карточка может быть использована одним из двух способов: пусть на сторонах карточки написано “$-a_i$ и “$-b_i\%$”, соответственно, тогда она может либо уменьшить текущую цену на $a_i$ (использование лицевой стороной), либо умножить текущую цену на $1 - b_i/100$ (использование обратной стороной). Все действия выполняются с вещественными числами. Если цена становится отрицательной, то покупка отдается покупателю бесплатно.

Петя собирается купить в Неда-Моле Феррари. Цена Феррари до скидок равна $x$. Помогите Пете выяснить, как оптимально использовать свои карточки, чтобы окончательная цена Феррари была как можно меньше.

## Формат входного файла

Первая строка входного файла содержит два целых числа: $n$ ($1 \le n \le 50$) и $x$ ($1 \le x \le 10^9$). Следующие $n$ строк описывают карточки со скидкой, каждая строка содержит два целых числа $a_i$ и $b_i$ ($1 \le a_i \le 10\,000$, $1 \le b_i \le 99$).

## Формат выходного файла

Выведите в выходной файл $n$ строк. Строки должны описывать карточки со скидками в том порядке, в котором они должны быть использованы. Каждая строка должна содержать номер карточки, за которым через пробел должно следовать слово “`front`”, если карточку следует использовать лицевой стороной (“$-a_i$”), либо слово “`reverse`”, если карточку следует использовать обратной стороной (“$-b_i\%$”).

Конечная цена, которая получится при использовании карточек в соответствии с вашим выходным файлом, должна отличаться от оптимальной цены не больше чем на $10^{-9}$ (хотя бы по одной из абсолютной или относительной погрешностей).

## Пример

| discount.in | discount.out |
|---|---|
| 3 1000<br>10 1<br>20 1<br>10 2 | 3 reverse<br>1 front<br>2 front |

После применения третьей карточки Петя получает скидку 2%, то есть 20, цена становится 980. После этого он в любом порядке использует лицевой стороной первую и вторую карточки и получает еще $10 + 20 = 30$ скидки, итого цена становится 950.

# Задача F. Деление на два

| | |
|---|---|
| Имя входного файла: | `divide.in` |
| Имя выходного файла: | `divide.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Понятие *мощности множества* играет важную роль в математике. Будем говорить, что два множества $A$ и $B$ *равномощны* и писать $|A| = |B|$, если существует взаимно-однозначная функция $\phi : A \to B$ (биекция между множествами). Обозначим как 2 множество, состоящее из двух элементов 0 и 1. Легко показать, что если $|A| = |B|$, то $|2 \times A| = |2 \times B|$. (Множество $X \times Y$ представляет собой множество упорядоченных пар, где первый элемент выбирается из множества $X$, а второй — из множества $Y$. Например, $\{1, 2\} \times \{a, b, c\} = \{\langle 1, a\rangle, \langle 1, b\rangle, \langle 1, c\rangle, \langle 2, a\rangle, \langle 2, b\rangle, \langle 2, c\rangle\}$.

Немного более интересно обратное утверждение: если $|2 \times A| = |2 \times B|$, то $|A| = |B|$. Простое доказательство этого факта использует так называемую *аксиому выбора*, и поэтому является *неконструктивным* — по заданной биекции между $2\times A$ и $2\times B$ не строится в явном виде биекция между $A$ и $B$, а лишь доказывается ее существование.

Элегантная конструкция Конвея и Дойля позволяет построить искомую биекцию $A$ между $B$ конструктивно по заданной биекции между $2 \times A$ и $2 \times B$. Мы опишем метод построения этой биекции для конечных множеств, хотя его можно распространить и на бесконечные множества.

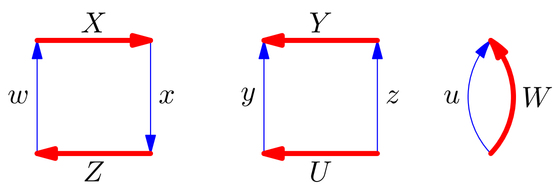
Пусть задана биекция $\phi : 2 \times A \to 2 \times B$. Мы хотим построить биекцию $\xi : A \to B$.

Для начала давайте изобразим биекцию $\phi$ в графическом виде. Изобразим элементы $A$ и $B$ как синие и красные стрелки, соответственно.


Теперь соединим их в соответствии с заданной биекцией $\phi$. Пусть $a \in A$. Если $\phi(\langle 0, a\rangle) = \langle 0, b\rangle$, то соединим начало синей стрелки, соответствующей $a$ с началом красной стрелки, соответствующей $b$. Аналогично, если $\phi(\langle 0, a\rangle) = \langle 1, b\rangle$, соединим начало стрелки $a$ с концом стрелки $b$. Аналогичным образом используем $\phi(\langle 1, a\rangle)$ для соединения концов синих стрелок с началами или концами красных стрелок.

Следующий рисунок показывает пример изображения биекции между $2 \times \{u, w, x, y, z\}$ $2 \times \{U, W, X, Y, Z\}$, заданной таблицей слева.

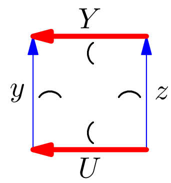
| Элемент $\alpha$ | $\phi(\alpha)$ |
|---|---|
| $\langle 0, u\rangle$ | $\langle 0, W\rangle$ |
| $\langle 1, u\rangle$ | $\langle 1, W\rangle$ |
| $\langle 0, w\rangle$ | $\langle 1, Z\rangle$ |
| $\langle 1, w\rangle$ | $\langle 0, X\rangle$ |
| $\langle 0, x\rangle$ | $\langle 1, X\rangle$ |
| $\langle 1, x\rangle$ | $\langle 0, Z\rangle$ |
| $\langle 0, y\rangle$ | $\langle 1, U\rangle$ |
| $\langle 1, y\rangle$ | $\langle 1, Y\rangle$ |
| $\langle 0, z\rangle$ | $\langle 0, U\rangle$ |
| $\langle 1, z\rangle$ | $\langle 0, Y\rangle$ |




Как видно, изображение биекции представляет собой чередующиеся циклы, состоящие из красных и синих стрелок (если бы множества были бесконечными, то могли бы также получиться бесконечные чередующиеся пути, но в конечном случае получаются только циклы). Теперь мы опишем конструктивный метод как построить биекцию между синими и красными стрелками.

Будем рассматривать циклы по отдельности. Если у цикла больше стрелок направлено в одном направлении, чем в другом, то будем считать это направление *основным направлением* цикла. В этом случае каждой синей стрелке из этого цикла сопоставим красную стрелку, которая следует за ней в основном направлении. Так, на предыдущем рисунке это определяет значения биекции для $w$ и $x$: $\xi(w) = X$, $\xi(x) = Z$.

Если же в цикле равное число стрелок указывает в каждом из направлений, то используем другой способ. Нарисуем рядом с каждой стрелкой скобку, закрывающей стороной в сторону, в которую направлена стрелка. Если мы теперь обойдем цикл, то мы получим последовательность скобок. В зависимости от начала обхода, может получиться правильная скобочная последовательность, либо нет. Найдется хотя бы одна позиция, начав обход из которой, мы получим правильную скобочную последовательность. Выберем любую такую позицию и рассмотрим соответствующую последовательность. Сопоставим каждой синей стрелке ту красную стрелку, рядом с которой записана парная ей скобка.




Например, на приведенном рисунке мы можем начать обход против часовой стрелки, начав с $y$. Получается скобочная последовательность `(())`. Это дает значения для биекции: $\xi(y) = Y$, $\xi(z) = U$.

Аналогично получаем $\xi(u) = W$.

По заданной биекции $\phi$ между $2 \times A$ и $2 \times B$, постройте по приведенному алгоритму биекцию $\xi$ между $A$ и $B$.

## Формат входного файла

Первая строка входного файла содержит $n$ — количество элементов в каждом из множеств $A$ и $B$ $(1 \le n \le 50\,000)$. Будем обозначать элементы $A$ и $B$ числами от 1 до $n$.

Следующие $2n$ строк описывают заданную биекцию. Первая строка содержит $\phi(\langle 0, 1\rangle)$, вторая — $\phi(\langle 1, 1\rangle)$, третья — $\phi(\langle 0, 2\rangle)$, и т.д.

## Формат выходного файла

Выведите $n$ целых чисел — $\xi(1)$, $\xi(2)$, и т.д.

## Пример

| divide.in | divide.out |
|---|---|
| 5<br>0 2<br>1 2<br>1 5<br>0 3<br>1 3<br>0 5<br>1 1<br>1 4<br>0 1<br>0 4 | 2<br>3<br>5<br>4<br>1 |

# Задача G. Сферический конь в вакууме

| | |
|---|---|
| Имя входного файла: | `knight.in` |
| Имя выходного файла: | `knight.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Задача об обходе конем шахматной доски — одна из классических задач, на ней часто демонстрируют различные методы ускорения перебора. Рассмотрим трехмерную версию этой задачи.

Задан параллелепипед размера $a \times b \times c$. Шахматный конь может находится в любом единичном кубике этого параллелепипеда. Требуется, начав из одной из угловых клеток параллелепипеда, обойти конем параллелепипед так, чтобы конь побывал в каждом ее кубике ровно один раз.

Будем ссылаться на единичные кубики по трем координатам, которые будут изменяться от 1 до $a$, от 1 до $b$ и от 1 до $c$, соответственно. Конь может переместиться с кубика на другой, если одна из координат при этом остается прежней, одна изменяется ровно на один, а одна — ровно на два. Путь коня должен начинаться в кубике с координатами $(1, 1, 1)$.

## Формат входного файла

Входной файл содержит три целых числа $a$, $b$ и $c$ ($1 \le a, b, c \le 5$).

## Формат выходного файла

Если искомый путь существует, выведите “`YES`” на первой строке выходного файла. После этого выведите координаты кубиков на пути коня в том порядке, в котором он на них побывает.

Если искомого пути не существует, выведите “`NO`” на первой строке выходного файла.

## Пример

| knight.in | knight.out |
|---|---|
| 4 3 2 | YES<br>1 1 1<br>3 2 1<br>1 3 1<br>2 1 1<br>4 1 2<br>2 2 2<br>4 2 1<br>2 3 1<br>3 1 1<br>1 2 1<br>3 3 1<br>4 1 1<br>4 3 2<br>3 1 2<br>1 2 2<br>3 3 2<br>2 1 2<br>1 3 2<br>3 2 2<br>1 1 2<br>2 3 2<br>4 3 1<br>2 2 1<br>4 2 2 |

# Задача H. И снова ладьи...

| | |
|---|---|
| Имя входного файла: | `rooks.in` |
| Имя выходного файла: | `rooks.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

На прямоугольной доске размера $m \times n$ требуется расставить $k$ ладей. Размещение ладей будем называть *корректным*, если ни одна ладья не находится между двумя другими ладьями по вертикали или горизонтали.

По заданным $m$, $n$ и $k$ найдите количество корректных размещений ладей на доске. Поскольку это число может быть достаточно большим, выведите его по модулю 10003.

## Формат входного файла

Входной файл содержит $m$, $n$ и $k$ ($1 \le m, n \le 50$, $1 \le k \le mn$).

## Формат выходного файла

Выведите одно число — количество корректных размещений $k$ ладей на доске размера $m \times n$, по модулю 10003.

## Пример

| rooks.in | rooks.out |
|---|---|
| 3 2 3 | 18 |

# Задача I. Квадратный корень

| | |
|---|---|
| Имя входного файла: | `sqroot.in` |
| Имя выходного файла: | `sqroot.out` |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 64 мегабайта |

Введем в рассмотрение так называемые *0-1 матрицы* размером 4 на 4. Такая матрица — это квадратная таблица, содержащая 16 чисел $a_{i,j}$ ($i = 1 \dots 4$, $j = 1 \dots 4$), каждое из которых равно 0 или 1.

*Произведением* двух матриц $A$ и $B$ называется матрица $A \cdot B = C$, элементы которой вычисляются по формуле $c_{i,j} = (\sum_{k=1}^{4} a_{i,k} \cdot b_{k,j}) \bmod 2$. *Квадратным корнем* из матрицы $A$ называется 0-1 матрица $B$, такая что $B \cdot B = A$.

Задана некоторая 0-1 матрица размера 4 на 4. Вычислите ее квадратный корень или установите, что его не существует.

## Формат входного файла

Входной файл содержит четыре строки, каждая из которых содержит четыре числа (каждое из этих чисел — либо 0, либо 1). $j$-ое число $i$-ой строки соответствует элементу $a_{i,j}$ заданной матрицы $A$.

## Формат выходного файла

Выведите в выходной файл квадратный корень из заданной матрицы в формате, аналогичном входному файлу. Если квадратного корня не существует — выведите в выходной файл `NO SOLUTION`. Если решений несколько, выведите любое.

## Примеры

| sqroot.in | sqroot.out |
|---|---|
| 1 0 0 0<br>0 1 0 0<br>0 0 1 0<br>0 0 0 1 | 1 0 0 0<br>0 1 0 0<br>0 0 1 0<br>0 0 0 1 |
| 0 0 0 1<br>0 1 0 1<br>0 1 0 1<br>1 0 0 0 | NO SOLUTION |