Перебор

1613. Два числа и четыре действия

Над двумя целыми положительными числами были выполнены следующие четыре действия:

- их сложили;
- вычли из большего меньшее;
- перемножили;
- разделили большее на меньшее.

После сложения всех результатов получили число n. Вам следует найти все пары таких чисел.

Вход. Одно натуральное число n ($1 \le n \le 10^{12}$).

Выход. В отдельных строках вывести по одной паре чисел $x \le y$, удовлетворяющих условию задачи. Пары должны быть упорядочены по возрастанию x. Если нет ни одной пары чисел, удовлетворяющей условию, вывести “NO SOLUTION” (без кавычек).

Пример входа 1
```
4
```

Пример выхода 1
```
1 1
```

Пример входа 2
```
1
```

Пример выхода 2
```
NO SOLUTION
```

Пример входа 3	**Пример выхода 3**
243	2 54 8 24

Пусть x, y ($x \le y$) – входные числа. В результате четырех действий получим следующие результаты:

- $x + y$: их сложили;
- $y - x$: вычли из большего меньшее;
- $x * y$: перемножили;
- y / x : разделили большее на меньшее.

Просуммируем четыре полученных числа:

$$2y + x * y + y / x = y \cdot \frac{2x + x^2 + 1}{x} = y \cdot \frac{(x+1)^2}{x} = n$$

Переберем значение x от 1 до $\sqrt{n}$. Если n делится на $(x + 1)^2$, то получим пару, в которой

$$y = \frac{n \cdot x}{(x+1)^2}$$

Реализация алгоритма

Читаем значение n.

```
scanf("%lld", &n);
```

Если станет *flag* = 1, то хотя бы одна пара (x, y) будет найдена.

```
flag = 0;
for (x = 1; x * x <= n; x++)
{
  if (n % ((x + 1) * (x + 1)) == 0)
  {
    flag = 1;
    y = n / ((x + 1) * (x + 1)) * x;
    printf("%lld %lld\n", x, y);
  }
}
```

Если ни одна пара (x, y) не найдена, выводим “NO SOLUTION”.

```
if (!flag) puts("NO SOLUTION");
```

2371. Черный квадрат

Вдохновленный шедевром Казимира Малевича "Черный квадрат", Петр Палевич решил создать собственную версию картины. Он приготовил полотно в виде прямоугольной сетки с $m \times n$ белыми квадратами – m строк по n ячеек каждая.

Петр покрасил некоторые клетки в черный цвет так, что черные ячейки сформировали квадрат размером $s \times s$ ячеек. Но на следующий день Петр разочаровался в своем творении и уничтожил его, разрезав полотно горизонтальными полосами размера $1 \times n$, после чего сжег их в камине.

На следующее утро Петр передумал и решил восстановить картину. Он попытался найти ее останки в камине, и, к счастью, одну из полос, а именно k-ую сверху, огонь не тронул.

Теперь Петр задумался, можно ли восстановить картину на основе этой полосы. Помогите ему сделать это.

Вход. Первая строка содержит четыре целых числа: m, n, s и k ($1 \le m, n \le 5000$, $1 \le s \le \min(m, n)$, $1 \le k \le m$).

Вторая строка содержит n символов и описывает k-ую строку картины, '.' означает белую клетку, '*' означает черную клетку.

Выход. Если изображение может быть однозначно восстановлено, то следует вывести "Unique". Если существует несколько вариантов восстановления картины, то вывести "Ambiguous". Если ни одной соответствующей картины не существует, вывести "Impossible".

Пример входа 1
```
4 4 1 2
..*.
```

Пример выхода 1
```
Unique
```

Пример входа 2
```
4 4 2 2
..**
```

Пример выхода 2
```
Ambiguous
```

Вычислим количество черных клеток в k-ой строке картины. Все они должны располагаться рядом, их количество должно равняться s.

Далее переберем все возможные позиции (i, j) левого верхнего угла квадрата и посчитаем количество вариантов расположения квадрата. Это число вариантов может равняться 0 (ответ Impossible), 1 (ответ Unique), или быть больше 1 (ответ Ambiguous).

Реализация алгоритма

k-ую строку картины будем хранить в массиве ss.

```
#define MAX 5010
char ss[MAX];
```

Читаем входные данные. Уменьшим значение k на 1 чтобы нумерация строк начиналась с нуля.

```
scanf("%d %d %d %d\n",&m,&n,&s,&k);
k--;
gets(ss);
```

Определим положение черных клеток: самая левая клетка будет в позиции *start*, самая правая в *finish*. Вычислим общее количество черных клеток *sum*.

```
sum = 0; start = finish = -1;
for(i = 0; i < n; i++)
  if (ss[i] == '*')
  {
    sum++;
    if (start == -1) start = i;
    finish = i;
  }
```

Рассмотрим случай, когда в *k*-ой строке имеются черные клетки. Тогда их количество *sum* должно равняться длине отрезка [*start*, *finish*], а также значению *s*.

```
if ((sum > 0) && ((sum != finish - start + 1) || (sum != s)))
{
  puts("Impossible");
  return 0;
}
```

Переберем все возможные позиции (*i*, *j*) левого верхнего угла квадрата и посчитаем количество вариантов *pos* расположения квадрата.

```
pos = 0;
for (i = 0; i < m - s + 1; i++)
for (j = 0; j < n - s + 1; j++)
```

Если *k*-ая строка расположена среди строк [*i*, *i* + *s* – 1], то она должна содержать черные клетки, причем самая левая позиция *start* черных клеток должна равняться *j*.

```
  if ((i <= k) && (k <= i + s - 1))
  {
    if (j == start) pos++;
  }
  else
  {
```

Иначе *k*-ая строка не должна содержать черные клетки (в этом случае *start* остается равным -1).

```
    if (start == -1) pos++;
  }
```

В зависимости от значения *pos* выводим ответ.

```
if (pos == 0)
  puts("Impossible");
else if (pos == 1)
  puts("Unique");
else
  puts("Ambiguous");
```

3839. Продуктовый магазин

Кассир продуктового магазина, как оказалось, не умеет различать символы умножения и сложения. Для облегчения процесса оплаты, Вы хотите купить товары таким образом, чтобы произведение их цен равнялось их сумме.

Конечно, если Вы купите только один товар, то условие всегда верно. Покупка двух или трех товаров является довольно скучным занятием для Вас. Поэтому Вы заинтересовались в поиске возможных цен из четырех товаров таких, что сумма четырех цен равна их произведению. Цены следует рассматривать в € с двумя знаками после запятой. Отметим, что каждое изделие стоит по крайней мере один цент.

Вход. Содержит одно действительное число с двумя десятичными знаками – сумму S ($5 \le S \le 25$).

Выход. Вывести все решения, в которых сумма четырех изделий не более S €. Для каждого решения выведите в отдельной строке цены четырех изделий в неубывающем порядке, разделяя их одним пробелом. Решения следует выводить по возрастанию стоимости первого изделия. Если у нескольких решений стоимость первого изделия одинакова, то следует выводить их по возрастанию стоимости второго изделия. В случае равенства стоимости первого и второго изделий, следует выводить решения по возрастанию стоимости третьего изделия. Каждое решение следует выводить только один раз.

Пример входа	**Пример выхода**
6.7	1.00 1.75 1.90 2.00 1.10 1.50 2.00 2.00 1.25 1.25 1.92 2.21 1.25 1.40 1.86 2.00 1.25 1.60 1.75 1.84

Пусть a', b', c', d' – стоимости четырех изделий. Тогда необходимо найти все такие четверки (a', b', c', d'), для которых $a' \le b' \le c' \le d'$, $a' + b' + c' + d' = a' * b' * c' * d'$, а также $a' + b' + c' + d' \le S$. Для избежания ошибок при работе с действительными числами, переведем исходную сумму S в копейки. Положим $a = a' * 100$, $b = b' * 100$, $c = c' * 100$, $d = d' * 100$. Соотношение $a' + b' + c' + d' = a' * b' * c' * d'$ примет вид:

$$a / 100 + b / 100 + c / 100 + d / 100 = (a / 100) * (b / 100) * (c / 100) * (d / 100)$$

или то же самое что

$$100^3 (a + b + c + d) = a * b * c * d$$

Остается перебрать значения a, b, c, d вплоть до копейки, для которых также выполняются соотношения $a \le b \le c \le d$ и $a + b + c + d \le 100 * S$.

Реализация алгоритма

Читаем входную сумму *ss*. Переведем ее в копейки. Далее будем искать решения, в которых сумма четырех изделий не превышает *maxPrice* = 100 * *ss* копеек.

```
scanf("%lf",&ss); maxPrice = (int)(100*ss + 1e-7);
```

Поскольку $a + b + c + d \le maxPrice$, то $a \le maxPrice / 4$, $b \le (maxPrice - a) / 3$, $c \le (maxPrice - a - b) / 2$. Используем эти ограничения при переборе.

```
for (int a = 1; a <= maxPrice/4; a++)
for (int b = a; b <= (maxPrice - a) / 3; b++)
for (int c = b; c <= (maxPrice - a - b) / 2; c++)
{
```

Из соотношения $1000000 * (a + b + c + d) = a * b * c * d$ выразим d:

$$d = 1000000 * (a + b + c) / (a * b * c - 1000000)$$

Значение d должно быть неотрицательным и целым.

```
  int s = 1000000 * (a + b + c), p = a * b * c - 1000000;
  if (p <= 0 || s % p != 0) continue;
  int d = s / p;
```

Должны выполняться соотношения $c \le d$ и $a + b + c + d \le maxPrice$.

```
  if (d < c || a + b + c + d > maxPrice) continue;
```

Выводим найденное решение – четверку (a', b', c', d') = (a / 100, b / 100, c / 100, d / 100).

```
  printf("%.2f %.2f %.2f %.2f\n",a/100.0,b/100.0,c/100.0,d/100.0);
}
```

9064. Старик и шахматная доска

За время своего путешествия Кратос побывал в множестве разных мест. Так, сегодня он забрел в маленькую деревушку, где его приютил седой старик, накормил и дал место для ночлега. Взамен старик попросил всего одну вещь – сделать для него шахматную доску, ведь он так любит эту игру.

У старика есть n белых и m черных квадратиков 1 * 1, из которых он хочет сделать не обычную доску 8 * 8, а наибольшую возможную, которая во-первых будет квадратной, а во-вторых будет иметь шахматную раскраску, то есть где любые две соседние по стороне клетки будут разных цветов (при этом угловые клетки могут быть как белого, так и черного цвета, в отличие от обычной шахматной доски). Кратос не совсем понял, зачем старику такая доска, но спорить не стал, и принялся за работу. Однако, с математикой у нашего титана совсем плохо, поэтому найти длину стороны квадрата, которая в итоге должна получиться, для него оказалось непосильной задачей, и он обратился за помощью

к вам. Помогите ему – найдите максимальную длину шахматной доски, которую можно составить из имеющихся квадратиков.

Вход. Два целых числа n и m ($0 \le n, m \le 10^9$) – количество белых и черных квадратиков соответственно. Гарантируется, что $n + m > 0$.

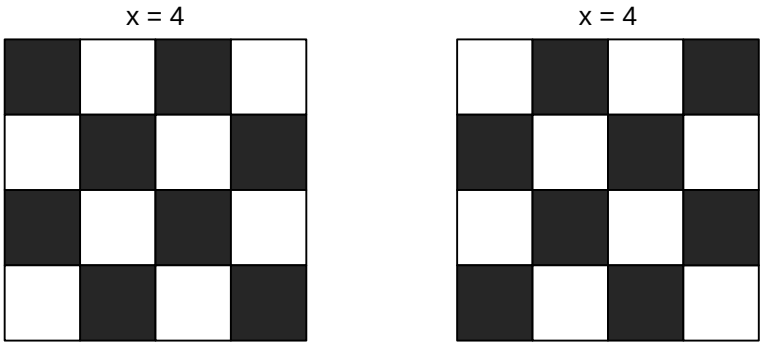
Выход. Выведите длину стороны максимального возможного квадрата, имеющего шахматную раскраску, который можно составить из имеющихся у старика квадратиков. Квадратики, конечно же, необязательно использовать все.

Пример входа 1	**Пример выхода 1**
8 9	4

Пример входа 2	**Пример выхода 2**
15 12	5

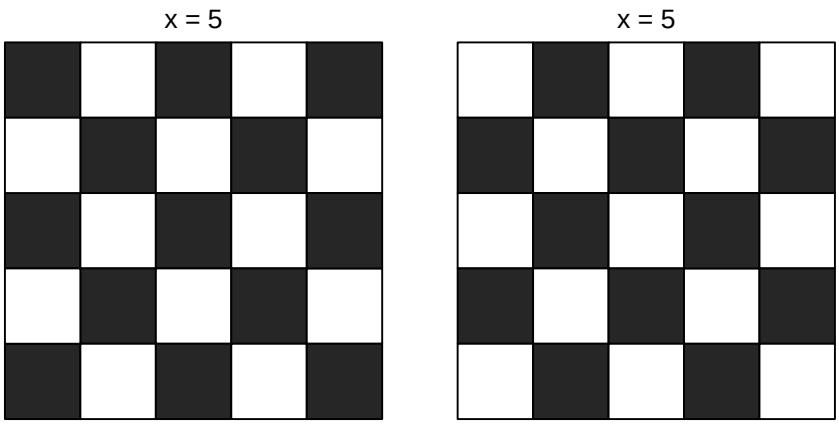
Квадрат, содержащий $n + m$ клеток, имеет длину стороны не более $\sqrt{n+m}$. Пусть x – длина стороны квадрата.

Если x четное, то в таком квадрате количество черных и белых клеток одинаково и равно $x^2 / 2$. Если $x^2 / 2 \le n$ и $x^2 / 2 \le m$, то квадрат со стороной x можно составить из имеющихся у старика квадратиков.



Если x нечетное, то в таком квадрате количество черных и белых клеток отличается на 1 и в зависимости от цвета покраски одного из углов может равняться:

- $(x^2 – 1) / 2$ белых и $(x^2 + 1) / 2$ черных;
- $(x^2 – 1) / 2$ черных и $(x^2 + 1) / 2$ белых;



Если количество белых и черных квадратов не более n и m, то квадрат со стороной x составить можно.

Перебираем значение x от $\sqrt{n+m}$ до 1 и выводим первое такое число x, для которого существует искомый квадрат.

Реализация алгоритма

Читаем входные данные.

```
scanf("%d %d", &n, &m);
x = sqrt(n + m);
```

Перебираем возможную длину стороны квадрата.

```
for (i = x; i > 0; i--)
{
  if (i % 2 == 0)
  {
```

Длина стороны квадрата i четная. Если в наличии имеются i^2 / 2 белых и черных квадратов, то ответ найден.

```
    q = i * i / 2;
    if (n >= q && m >= q) break;
  }
  else
  {
```

Длина стороны квадрата i нечетная. Если в наличии имеются (x^2 – 1) / 2 белых и (x^2 + 1) / 2 черных квадратиков или (x^2 – 1) / 2 черных и (x^2 + 1) / 2 белых квадратиков, то ответ найден.

```
    q = (i * i - 1) / 2;
    if (n >= q && m >= q + 1) break;
    if (n >= q + 1 && m >= q) break;
  }
}
```

Выводим ответ.

```
printf("%d\n", i);
```

Реализация алгоритма – формула
Читаем входные данные. Меняем местами n и m так чтобы было $n \le m$.

```
scanf("%d %d", &n, &m);
if (n > m) swap(n, m); // n <= m
```

Пусть длина x стороны квадрата четная. Поскольку $x^2 / 2 \le n$, то $x \le \sqrt{2n}$. Вычислим $x = \sqrt{2n}$ и проверим, является ли x четным. Если x нечетно, то уменьшим его на 1. Шахматную доску со стороной x можно составить.

```
x = int(sqrt(2 * n));
if (x % 2 == 1) x--;
res = x;
```

Пусть длина x стороны квадрата нечетная. Количество черных и белых квадратов должно отличаться на 1. Если $n = m$, то уменьшим n на 1.

```
if (n == m) n--;
```

Имеет место неравенство: $(x^2 - 1) / 2 \le n$, то есть $x \le \sqrt{2n+1}$. Вычислим $x = \sqrt{2n+1}$ и проверим, является ли x нечетным. Если x четное, то уменьшим его на 1. Шахматную доску со стороной x можно составить.

```
x = int(sqrt(2 * n + 1));
if (x % 2 == 0) x--;
if (x > res) res = x;
```

Выводим ответ.

```
printf("%d\n", res);
```

10029. Корона2020

Зия подозревает, что заразился коронавирусом. В связи с этим он ведёт исследование на своём ДНК, и, в результате вычислений, выясняет, что три различных числа a, b и c связаны с его ДНК. Зия верит, что, если подставляя в выражение a <> b <> c вместо знаков (<>) + или -, можно получить число 2020, то он не заразился коронавирусом, иначе, если это сделать невозможно, то он заразился. Помогите Зие выяснить, заразился ли он коронавирусом.

Вход. В одной строке заданы числа a, b и c ($1 \le a, b, c \le 10^8$).

Выход. Если Зия не заразится, выведите выражение *a* <> *b* <> *c*, которое даёт в результате 2020, иначе выведите слово CORONA. При выводе выражения, между числами и операторами не должно быть пробелов.

Пример входа 1	**Пример выхода 1**
2019 2020 2021	2019-2020+2021

Пример входа 2	**Пример выхода 2**
2019 2020 2022	CORONA

Переберем все возможные знаки между числами *a*, *b*, *c*. Если значение полученного выражения равно 2020, то выводим выражение. Иначе выводим слово CORONA.

Реализация алгоритма

Читаем входные данные.

```
scanf("%d %d %d", &a, &b, &c);
```

Перебираем все возможные знаки между числами. В зависимости от результата выводим ответ.

```
if (a + b + c == 2020) printf("%d+%d+%d", a, b, c); else
if (a + b - c == 2020) printf("%d+%d-%d", a, b, c); else
if (a - b + c == 2020) printf("%d-%d+%d", a, b, c); else
if (a - b - c == 2020) printf("%d-%d-%d", a, b, c); else
                       printf("CORONA\n");
```

10170. A * B + C

Задано натуральное число *n*. Сколько существует троек (A, B, C) натуральных чисел, удовлетворяющих равенству A * B + C = *n*?

Вход. Одно натуральное число *n* ($2 \le n \le 10^8$).

Выход. Выведите искомое количество троек.

Пример входа	**Пример выхода**
3	3

Поскольку числа A, B, C натуральные, то количество троек, являющихся решением A * B + C = *n*, равно числу пар (A, B), удовлетворяющих неравенству A * B < *n*. Это неравенство эквивалентно A * B ≤ *n* – 1, или B ≤ (*n* – 1) / A. Для каждого значения A ($1 \le A \le n - 1$) существует в точности $\lfloor (n-1)/A \rfloor$ таких B. Количество искомых троек равно

$$\sum_{i=1}^{n-1}\left\lfloor \frac{n-1}{i} \right\rfloor$$

Пример

Для $n = 3$ имеется в точности три тройки:

- (1, 1, 2), так как 1 * 1 + 2 = 3;
- (1, 2, 1), так как 1 * 2 + 1 = 3;
- (2, 1, 1), так как 2 * 1 + 1 = 3.

Искомая сумма равна

$$\sum_{i=1}^{2}\left\lfloor \frac{2}{i} \right\rfloor = \left\lfloor \frac{2}{1} \right\rfloor + \left\lfloor \frac{2}{2} \right\rfloor = 3$$

Пусть $n = 7$. Тогда A * B < 7 или A * B ≤ 6.
- если A = 1, то B ≤ 6 / 1 = 6. Пары (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6).
- если A = 2, то B ≤ 6 / 2 = 3. Пары (2, 1), (2, 2), (2, 3).
- если A = 3, то B ≤ 6 / 3 = 2. Пары (3, 1), (3, 2).
- если A = 4, то B ≤ 6 / 4 = 1. Пара (4, 1).
- если A = 5, то B ≤ 6 / 5 = 1. Пара (5, 1).
- если A = 6, то B ≤ 6 / 6 = 1. Пара (6, 1).

Искомая сумма равна

$$\sum_{i=1}^{6}\left\lfloor \frac{6}{i} \right\rfloor = \left\lfloor \frac{6}{1} \right\rfloor + \left\lfloor \frac{6}{2} \right\rfloor + \left\lfloor \frac{6}{3} \right\rfloor + \left\lfloor \frac{6}{4} \right\rfloor + \left\lfloor \frac{6}{5} \right\rfloor + \left\lfloor \frac{6}{6} \right\rfloor = 6 + 3 + 2 + 1 + 1 + 1 = 14$$

Упражнение

Решите задачу для $n = 11$. Найдите количество троек, являющихся решением уравнения A * B + C = 11 для натуральных A, B, C.

Реализация алгоритма

Читаем значение n.

```
scanf("%d", &n);
```

Вычисляем результирующую сумму.

```
res = 0;
for (i = 1; i < n; i++)
  res += (n - 1) / i;
```

Выводим ответ.

```
printf("%d\n", res);
```

10275. А + Б = С?

Пете скучно решать простые задачки, где нужно находить сумму двух чисел, поэтому его преподавательница программирования дала сложную задачу, которая звучит следующим образом:

Заданы три целых числа a, b и c. Следует определить, существует ли такое число среди них, что оно равно сумме двух других чисел.

Вход. Первая строка содержит три целых числа a, b и c ($0 \le a, b, c \le 100$).

Выход. Выведите “Yes”, если такое число существует, иначе выведите “No”.

Пример входа 1	**Пример выхода 1**
`0 3 3`	`Yes`
Пример входа 2	**Пример выхода 2**
`1 10 15`	`No`
Пример входа 3	**Пример выхода 3**
`12 20 8`	`Yes`

Для решения задачи для каждого из трех чисел следует проверить, равно ли оно сумме двух других. То есть для трех чисел a, b, c следует проверить выполнение каждого из следующих равенств:

$$a = b + c,\ b = a + c,\ c = a + b$$

Если одно из них выполняется, то ответ “Yes”. Иначе ответ “No”.

Пример

В первом тесте имеет место равенство: $0 + 3 = 3$.
Во втором тесте $1 + 10 \neq 15$.
В третьем тесте имеет место равенство: $12 + 8 = 20$.

Реализация алгоритма

Читаем входные данные.

```
scanf("%d %d %d", &a, &b, &c);
```

Проверяем, не равно ли каждое из чисел сумме двух других. Выводим ответ.

```
if (a == b + c || b == a + c || c == a + b)
  puts("Yes");
else
  puts("No");
```

10329. Игра с камушками

Чтобы скоротать время, корова Бесси и ее подруга Элси любят играть в аналог игры, которую они видели на окружной ярмарке.

Для начала Бесси кладет три перевернутые ракушки на стол и кладет под одну из них небольшой круглый камешек (по крайней мере, она думает что это камешек, так как нашла его на земле на одном из пастбищ). Затем Бесси меняет пары ракушек, а Элси пытается угадать местонахождение камешка.

Стандартная версия игры, которую видели коровы на ярмарке графства, позволяла игроку видеть начальное местоположение камешка, а затем требовалось угадать его окончательное местоположение после завершения всех обменов.

Однако коровы любят играть в версию, в которой Элси не знает начального местоположения камешка и где она может угадывать местоположение камешка после каждого обмена. Бесси, зная правильный ответ, в конце ставит Элси оценку, равную количеству правильных предположений, которые она сделала.

Учитывая обмены и предположения, но не исходное местоположение камешка, определите максимальный балл, который могла заработать Элси.

Вход. Первая строка содержит целое число n ($1 \le n \le 100$) – количество обменов. Каждая из следующих n строк описывает шаг игры и содержит три целых числа a, b и g, указывающих, что ракушки a и b были переставлены Бесси, после чего Элси предположила что камень находится под ракушкой g. Среди трех целых чисел встречаются только 1, 2 или 3. Известно, что $a \ne b$.

Выход. Выведите максимальное количество очков, которое могла бы заработать Элси.

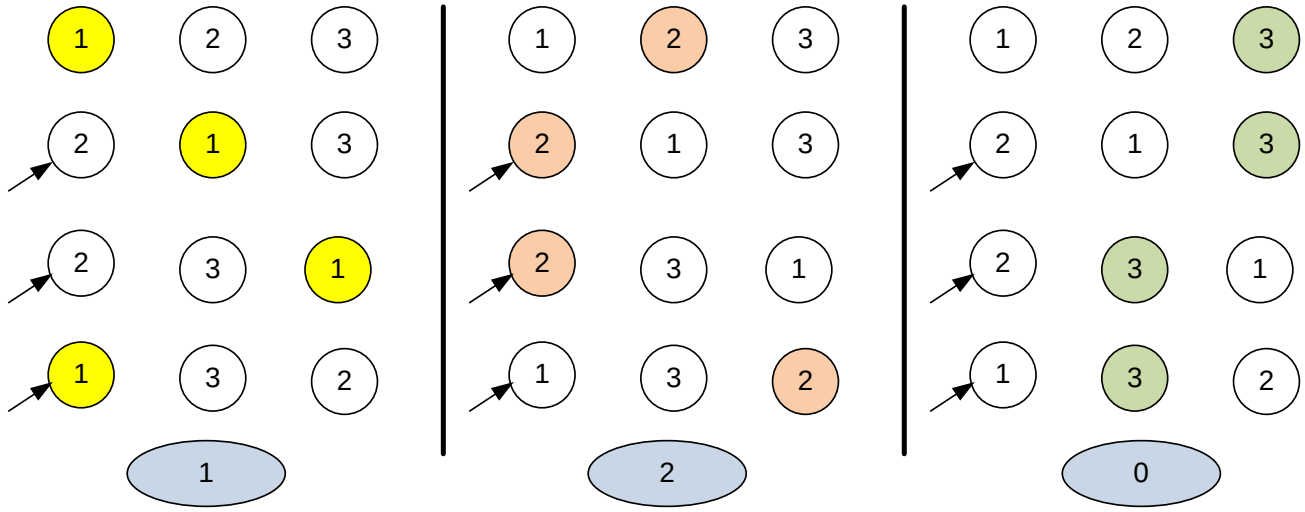
Пример. В приведенном примере Элси может заработать не более 2 очков. Если камешек изначально находился под ракушкой 1, то она угадала бы ровно один раз (в последнем предположении). Если камешек изначально был под ракушкой 2, то она угадает дважды (в первых двух попытках). Если камешек изначально был под ракушкой 3, то она не делает правильных предположений.

Пример входа	**Пример выхода**
3 1 2 1 3 2 1 1 3 1	2

Камень может изначально находиться под первой, второй или третьей ракушкой. Рассмотрим каждый из этих трех случаев и для каждого из них вычислим количество очков, которое заработает Элси. Максимальное из них и будет ответом. Задача сводится к моделированию игры.

Пример

Промоделируем игру для трех случаев: когда камешек сначала находится под первой, второй и третьей ракушкой. Стрелками указаны предположения Элси.



В первом случае Элси угадает 1 раз, во втором – два раза, в третьем – 0 раз. Максимальное количество очков, которое может заработать Элси, равно 2. Это будет в случае, когда камешек изначально находится под ракушкой номер 2.

Реализация алгоритма

Игра содержит не более 100 шагов, сохраним их в массивах a, b и g.

```
int a[100], b[100], g[100];
```

Функция ***num_correct*** возвращает количество очков, которое получит Элси при условии что камешек изначально находится под ракушкой номер *i*.

```
int num_correct(int starting_shell)
{
```

Переменная *current_shell* содержит номер ракушки, под которой перед каждой итерацией находится камешек.

```
  int current_shell = starting_shell, correct = 0;
  for (int i = 0; i < n; i++)
  {
```

Если камешек находится в позиции a_i, то его следует переместить в позицию b_i (и наоборот).

```
    if (a[i] == current_shell) current_shell = b[i];
    else if (b[i] == current_shell) current_shell = a[i];
```

Если после обмена камешек находтся в позиции g_i, то Элси угадывает его положение и зарабатывает одно очко.

```
    if (current_shell == g[i]) correct++;
  }
  return correct;
}
```

Основная часть программы. Читаем входные данные.

```
scanf("%d", &n);
for (i = 0; i < n; i++)
  scanf("%d %d %d", &a[i], &b[i], &g[i]);
```

Перебираем все возможные начальные положения камня: i = 1, 2, 3. Для каждого из них вычисляем количество очков Элси. В переменной *best* находим максимум среди них.

```
int best = 0;
for (i = 1; i <= 3; i++)
  best = max(best, num_correct(i));
```

Выводим ответ.

```
printf("%d\n", best);
```

Реализация алгоритма – моделирование

Шаги игры сохраним в массивах a, b и g.

```
int a[100], b[100], g[100];
```

Читаем входные данные.

```
scanf("%d", &n);
for (i = 0; i < n; i++)
  scanf("%d %d %d", &a[i], &b[i], &g[i]);
```

В переменной *best* вычисляем максимальный балл, который могла заработать Элси.

```
int best = 0;
```

Перебираем все возможные начальные положения камня: i = 1, 2, 3.

```
for (i = 1; i <= 3; i++)
{
```

Начальное положение игры кодируем в массиве m. Изначально камень лежит под ракушкой i: установим m[i] = 1, остальные ячейки массива m обнулим.

```
  int m[] = { 0, 0, 0, 0 };
  m[i] = 1;
```

В переменной *cnt* вычисляем количество правильных предположений Элси.

```
  cnt = 0;
```

В массиве m моделируем n шагов игры. Меняем местами ракушки номер a[i] и b[i]. Если m[g[i]] = 1, то предположение Элси верно.

```
  for (j = 0; j < n; j++)
  {
    swap(m[a[j]], m[b[j]]);
    cnt += m[g[j]];
  }
```

Находим максимум среди всех значений *cnt*.

```
  if (cnt > best) best = cnt;
}
```

Выводим ответ.

```
printf("%d\n", best);
```

10331. Отгадай животное

Когда коровам стало скучно играть в игру ракушки, Бесси и ее подруга Элси любят играть в другую игру, называемую “угадай животное”.

Первоначально Бесси думает о каком-то животном (чаще всего это животное – корова, что делает игру довольно скучной, но иногда Бесси проявляет изобретательность и думает о чем-то другом). Затем Элси задает серию вопросов, чтобы выяснить, какое животное выбрала Бесси. Каждый вопрос спрашивает, есть ли у животного какие-то определенные характеристики, и Бесси отвечает на каждый вопрос “да” или “нет”. Например:

Elsie: “Животное летает?”
Bessie: “нет”
Elsie: “Животное ест траву?”
Bessie: “да”
Elsie: “Животное дает молоко?”
Bessie: “да”
Elsie: “Животное говорит муу?”
Bessie: “да”
Elsie: “В таком случае я думаю, что это корова.”
Bessie: “Правильно! ”

Назовем “допустимым множеством” набор всех животных с характеристиками, согласующимися с вопросами Элси. Тогда Элси продолжает задавать вопросы до тех пор, пока возможный набор не будет содержать только одно животное, после чего она объявляет это животное в качестве своего ответа. В каждом вопросе Элси выбирает характеристику какого-нибудь животного из возможного набора, чтобы спросить о нем (даже если эта характеристика не сможет помочь ей сузить возможный набор в дальнейшем). Она никогда не спрашивает дважды об одной и той же характеристике.

Зная всех животных, которых знают Бесси и Элси, а также их характеристики, определите максимальное количество ответов “да”, которые Элси могла бы получить, прежде чем она узнает правильное животное.

Вход. Первая строка содержит количество животных n ($2 \le n \le 100$). Каждая из следующих n строк описывает одно животное. Строка начинается с имени животного, затем целого числа k ($1 \le k \le 100$), и k характеристик этого животного. Имена и характеристики животных представляют собой строки, содержащие до 20 строчных букв (a .. z). Нет двух животных с абсолютно одинаковыми характеристиками.

Выход. Выведите максимальное количество ответов “да”, которое Элси могла бы получить до окончания игры.

Пример входа	**Пример выхода**
4 bird 2 flies eatsworms cow 4 eatsgrass isawesome makesmilk goesmoo sheep 1 eatsgrass goat 2 makesmilk eatsgrass	3

Перебором найдем двух животных с максимальным количеством общих признаков *temp*. Тогда максимальное количество ответов “да”, которые может получить Элси, равно *temp* + 1.

Пример

Корова и коза имеют два общих признака: *makesmilk*, *eatsgrass*. На эти запросы Элси даст ответ “да”. Третьим запросом с ответом “да” будет запрос о признаке коровы, отличный от этих двух.

Реализация алгоритма

Характеристики животного номер *i* будем хранить в массиве ch[*i*].

```
vector<string> ch[100];
```

Функция ***common*** вычисляет количество общих признаков для животных с номерами *i* и *j*.

```
int common(int i, int j)
{
  int cnt = 0;
  for (int x = 0; x < ch[i].size(); x++)
  for (int y = 0; y < ch[j].size(); y++)
    if (ch[i][x] == ch[j][y]) cnt++;
  return cnt;
}
```

Основная часть программы. Читаем входные данные.

```
cin >> n;
for (i = 0; i < n; i++)
{
  cin >> s >> k;
  for (j = 0; j < k; j++)
  {
    cin >> s;
    ch[i].push_back(s);
  }
}
```

Перебираем все пары животных. Для каждой пары (*i*, *j*) вычисляем количество их общих признаков *temp*. Среди всех таких значений *temp* вычисляем наибольшее значение *res*.

```
res = 0;
for (i = 0; i < n; i++)
for (j = i + 1; j < n; j++)
{
  temp = common(i, j);
  if (temp > res) res = temp;
}
```

Выводим ответ.

```
cout << res + 1 << endl;
```

10481. A*B*C

Для заданного натурального числа k найдите количество троек натуральных чисел (a, b, c) таких что $a * b * c \le k$. Две тройки, которые отличаются только порядком, считаются разными.

Вход. Одно целое число k ($1 \le k \le 2 * 10^5$).

Выход. Выведите количество троек натуральных чисел (a, b, c) таких что $a * b * c \le k$.

Пример входа 1	**Пример выхода 1**
2	4
Пример входа 2	**Пример выхода 2**
10	53

Переберем значение a от 1 до k. Переберем b от 1 и до тех пор пока $a * b \le k$. Для каждой такой пары (a, b) существует в точности $\lfloor k/(a \cdot b) \rfloor$ значений c таких что $a * b * c \le k$. Осталось просуммировать эти значения c.

Реализация алгоритма
Читаем входное значение *k*.

```
scanf("%lld", &k);
res = 0;
```

Запускаем перебор по *a* и по *b*.

```
for (a = 1; a <= k; a++)
for (b = 1; ; b++)
{
```

Если *a* * *b* > *k*, то продолжаем перебор по *a*.

```
  if (a * b > k) break;
```

Прибавим к результату *res* значение $\lfloor k/(a \cdot b) \rfloor$.

```
  c = k / (a * b);
  res += c;
}
```

Выводим ответ.

```
printf("%lld", res);
```

10568. Коллекционер алмазов (бронза)

Беси собрала n алмазов различных размеров. И хочет разместить их специальным образом в амбаре. Она не будет включать в размещение два алмаза, если их размеры отличаются более чем на k. По данному k определите максимальное количество алмазов, которые Беси разместит в амбаре.

Вход. Первая строка содержит n ($n \le 1000$) и k ($0 \le k \le 10000$). Каждая из следующих n строк содержит целое число, определяющее размер одного из алмазов. Все размеры – положительные числа, не превышающие 10000.

Выход. Выведите одно положительное целое число – максимальное количество алмазов, которое Беси сможет показать.

Пример входа	**Пример выхода**
5 3 1 6 4 3 1	4

Анализ алгоритма

Отсортируем массив с размерами алмазов. Для каждого значения i рассмотрим последовательности m_i, m_{i+1}, …, m_j такие что $| m_j - m_i | \le k$, но при этом $| m_{j+1} - m_i | > k$. Среди всех таких последовательностей найдем ту, которая содержит наибольшее количество элементов.

Реализация алгоритма

Объявим массив для хранения размеров алмазов.

```
int m[1000];
```

Читаем входные данные.

```
scanf("%d %d", &n, &k);
for (i = 0; i < n; i++)
  scanf("%d", &m[i]);
```

Отсортируем массив.

```
sort(m, m + n);
```

Максимальное количество алмазов, которое Беси сможет показать, подсчитываем в переменной *res*.

```
res = 0;
```

Для каждой позиции i рассматриваем последовательность m_i, m_{i+1}, …, m_j наибольшей длины такую что $| m_j - m_i | \le k$. Длину последовательности подсчитываем в переменной *cnt*.

```
for (i = 0; i < n; i++)
{
  cnt = 0;
  for (j = i; j < n; j++)
  {
```

Как только станет $| m_j - m_i | > k$, выходим их цикла.

```
    if (m[j] > m[i] + k) break;
    cnt++;
  }
```

Среди длин всех последовательностей находим наибольшую.

```
  if (cnt > res) res = cnt;
}
```

Выводим ответ.

```
printf("%d\n", res);
```