# Задача 1. Метро

Минимальное время, в течение которого на первом пути можно было наблюдать $n$ поездов равно $n+a(n-1)$ ($n$ поездов стоят на платформе по одной минуте, и $n-1$ интервал между ними по $a$ минут). Максимальное время, в течение которого на первом пути можно было наблюдать ровно $n$ поездов, равно $n+a(n+1)$ (добавились два интервала по $a$ минут). Таким образом, время нахождения Тани на платформе принадлежит отрезку $[n+a(n-1); n+a(n+1)]$.

Аналогично, Таня наблюдала $m$ поездов на второй платформе, поэтому время нахождения Тани на платформе принадлежит отрезку $[m+b(m-1); m+b(m+1)]$.

Необходимо найти пересечение данных отрезков и вывести левую и правую границы пересечения. Если пересечение пусто, то нужно вывести число $-1$.

Пример решения.

```
a = int(input())
b = int(input())
n = int(input())
m = int(input())
l = max(n + (n - 1) * a, m + (m - 1) * b)
r = min(n + (n + 1) * a, m + (m + 1) * b)
if l <= r:
    print(l, r)
else:
    print(-1)
```

# Задача 2. Площадь

Для решения задачи на 60 баллов (сторона площади не превышает 1000) можно написать перебор по ответу: циклом увеличиваем значение ширины дорожки $d$, пока количество плиток в дорожке не будет превосходить $t$. Пример такого решения.

```
n = int(input())
m = int(input())
t = int(input())
c = 2 * (n + m) - 4
d = 0
while c <= t and c > 0:
    d += 1
    t -= c
    c -= 8
print(d)
```

В этом решении в переменной $d$ хранится количество выложенных дорожек шириной в одну клетку, в переменной $c$ — количество плиток в одной дорожке. Для внешней дорожки $c = 2(n+m)-4$, каждая следующая дорожка содержит на 8 плиток меньше, чем предыдущая.

Для решения на полный балл посчитаем количество плиток в дорожке ширины $d$. Оно равно $nm-(n-2d)(m-2d)$. Решим неравенство

$$nm-(n-2d)(m-2d) \leqslant t.$$

Из этого неравенства $d \leqslant \frac{1}{4}\left(n+m-\sqrt{(n+m)^2-4t}\right)$. Вычислим значение этого выражения и возьмём от него целую часть.

Пример решения на 100 баллов.

```
n = int(input())
m = int(input())
t = int(input())
print(int(((n + m) - ((n + m) ** 2 - 4 * t) ** 0.5) / 4))
```

На 100 баллов возможно также решение, в котором используется двоичный поиск для нахождения наибольшего возможного значения $d$. Пример такого решения.

```
n = int(input())
m = int(input())
t = int(input())
left = 0
right = min(n // 2, m // 2) + 1
while right - left > 1:
    d = (left + right) // 2
    if n * m - (n - 2 * k) * (m - 2 * k) <= t:
        left = d
    else:
        right = d
print(left)
```

# Задача 3. Много пирожных

Пусть было выбрано $k$ видов пирожных и взято $s$ пирожных каждого выбранного вида. Очевидно, что значение $s$ совпадает с одним из значений $a_i$. Тогда можно перебрать все значения $a_i$, для каждого из них посчитать значение $k$ — количество видов пирожных, которых есть не менее $s$ штук, и выбрать такую пару $(k, s)$, при которой произведение $ks$ будет наибольшим. Пример такого решения:

```
n = int(input())
a = [int(input()) for i in range(n)]
ans_s = 0
ans_k = 0
for i in range(n):
    s = a[i]
    k = 0
    for j in range(n):
        if a[j] >= s:
            k += 1
    if k * s > ans_k * ans_s:
        ans_k = k
        ans_s = s
print(ans_k, ans_s)
```

Такое решение имеет сложность $O(n^2)$ и набирает 60 баллов.

Чтобы набрать 100 баллов можно отсортировать массив $a_i$ по невозрастанию значений. Тогда для выбранного значения $a_i$ виды пирожных, которых есть не менее $a_i$ штук будут идти раньше $a_i$, значит, их количество будет равно $i + 1$ (если нумеровать элементы массива с нуля). Это позволит избавиться от вложенного цикла и получить решение, сложность которого будет определяться сложностью алгоритма сортировки. Если использовать быстрые алгоритмы сортировки (присутствующие во всех современных языках программирования), сложность которых $O(n \log n)$, то такое решение будет набирать 100 баллов. Пример решения, использующего сортировку.

```
n = int(input())
a = [int(input()) for i in range(n)]
a.sort(reverse=True)
ans_s = 0
ans_k = 0
for i in range(n):
    if a[i] * (i + 1) > ans_k * ans_s:
        ans_k = i + 1
        ans_s = a[i]
print(ans_k, ans_s)
```

Также вместо алгоритма сортировки можно использовать идею подсчёта, воспользовавшись тем, что значения $a_i$ не превышают $10^5$. Приведём пример такого решения на языке Pascal.

```
var n, i, s, ans_k, ans_s, num: longint;
    count: array[1..100000] of longint;
begin
    readln(n);
    for i := 1 to 100000 do
        count[i] := 0;
    for i := 1 to n do
    begin
        readln(s);
        inc(count[s]);
    end;
    ans_k := 0;
    ans_s := 0;
    k := 0;
    for s := 100000 downto 1 do
    begin
        k := k + count[s];
        if k * s > ans_k * ans_s then
        begin
            ans_k := k;
            ans_s := s;
        end
    end;
    writeln(ans_k);
    writeln(ans_s)
end.
```

# Задача 4. Космические шахматы

Для решения на 16 баллов можно в тексте программы разобрать все 8 соседних клеток (разбор каждого случая оценивается в 2 балла).

У участников, знакомых с алгоритмами на графах, могло возникнуть желание написать алгоритм обхода в ширину для поиска кратчайшего пути. Но такое решение будет набирать 40 баллов, потому что для нахождения пути в отдалённую клетку этому алгоритму понадобится посетить слишком много клеток. Кроме того, в задаче не требуется найти кратчайший путь, достаточно найти какой-нибудь путь.

Заметим, что за 3 хода можно переставить коня в соседнюю клетку. Напишем алгоритм, передвигающий коня по одной в соседнюю клетку, пока не будет достигнута нужная клетка. Отдельно разберём случаи перемещения коня в соседнюю клетку в кажом из четырёх направлений. Пример такого решения.

```
x_finish = int(input())
y_finish = int(input())
x = 0
y = 0
while x < x_finish:
    print(x + 1, y + 2)
    print(x - 1, y + 1)
    print(x + 1, y)
    x += 1
while x > x_finish:
    print(x - 1, y + 2)
    print(x + 1, y + 1)
    print(x - 1, y)
    x -= 1
```

```
while y < y_finish:
    print(x + 2, y + 1)
    print(x + 1, y - 1)
    print(x, y + 1)
    y += 1
while y > y_finish:
    print(x + 2, y - 1)
    print(x + 1, y + 1)
    print(x, y - 1)
    y -= 1
```

Такое решение будет использовать $3(|x| + |y|)$ ходов для достижения клетки $(x; y)$, поэтому оно будет набирать только 70 баллов (в ограничениях на 70 баллов $|x|, |y| \leqslant 15\,000$, поэтому число ходов не превышает $90\,000$).

Для полного решения необходимо соптимизировать это решение, совершая меньшее число ходов для перемещения в нужном направлении. Пусть нам нужно попасть в клетку $(x; y)$. Если $|x| \geqslant |y|$, то сделаем перемещение на две клетки по оси $OX$ и на одну клетку по оси $OY$. Иначе сделаем перемещение на одну клетку по оси $OX$ и на две клетки по оси $OY$. При этом нужно учитывать направления (знаки чисел $x$ и $y$). Например, если коню нужно попасть в клетку $(1000; 0)$, то он будет чередовать ходы $(2; 1)$ и $(2; -1)$. А если нужно попасть в клетку $(1000; 1000)$, то будет чередование ходов $(2; 1)$ и $(1; 2)$.

Такими перемещениями можно попасть в клетку, близко расположенную к конечной. Если при этом мы не попали точно в конечную, то будем использовать перемещения в соседнюю клетку за три хода, как в предыдущем решении. Пример решения.

```
x_finish = int(input())
y_finish = int(input())

def sign(x):
    if x >= 0:
        return 1
    else:
        return -1

x = 0
y = 0
while abs(x_finish - x) > 1 or abs(y_finish - y) > 1:
    if abs(x_finish - x) >= abs(y_finish - y):
        dx = 2 * sign(x_finish - x)
        dy = sign(y_finish - y)
    else:
        dx = sign(x_finish - x)
        dy = 2 * sign(y_finish - y)
    x += dx
    y += dy
    print(x, y)

while x < x_finish:
    print(x + 1, y + 2)
    print(x - 1, y + 1)
    print(x + 1, y)
    x += 1
while x > x_finish:
    print(x - 1, y + 2)
    print(x + 1, y + 1)
```

```
    print(x - 1, y)
    x -= 1
while y < y_finish:
    print(x + 2, y + 1)
    print(x + 1, y - 1)
    print(x, y + 1)
    y += 1
while y > y_finish:
    print(x + 2, y - 1)
    print(x + 1, y + 1)
    print(x, y - 1)
    y -= 1
```

Оценим количество выполненных перемещений. Выполняя пару ходов $(2;1)$ и $(1;2)$ конь смещается на три клетки за два хода. Чтобы попасть из начала координат в клетку $(10^5;10^5)$ понадобится около $\frac{2}{3}10^5$ ходов.

# Задача 5. Городские центральные диаметры

Прежде всего избавимся от параметра $t$ — заложим время опоздания в график движения поездов, увеличив все значения $b_i$ и $d_j$ на $t$.

Далее задача решается моделированием событий вида «отправление поезда» и «прибытие поезда». Будем смотреть, что происходит на станциях и считать количество поездов, находящихся на каждой из двух станций. Если поезд прибывает на станцию, то увеличиваем количество поездов, находящихся на этой станции на 1. Если поезд отправляется от станции, то уменьшаем счётчик количества поездов на 1. При этом если поезд должен отправиться, а на станции в этот момент поездов нет, то необходимо ввести дополнительный поезд, что увеличит значение ответа на 1.

Все события нужно перебирать в порядке возрастания времён, при этом если в один и тот же момент происходит два события (отправление и прибытие поезда), то сначала необходимо обработать прибытие, а потом отправление поезда, так как прибывший поезд сразу же может отправиться в новый рейс.

Пример решения.

```
t = int(input())
trains = []
n = int(input())
for i in range(n):
    a = int(input())
    b = int(input())
    trains.append([a, 1])
    trains.append([b + t, -2])
m = int(input())
for i in range(m):
    a = int(input())
    b = int(input())
    trains.append([a, 2])
    trains.append([b + t, -1])
ans = 0
free = [0, 0, 0]
trains.sort()
for t, event in trains:
    if event < 0:
        free[-event] += 1
    else:
        if free[event] == 0:
            ans += 1
```

```
            free[event] = 1
        free[event] -= 1
print(ans)
```

В этом решении список `trains` содержит информацию об имеющихся событиях. В этот список кладутся пары значений, первый элемент пары — время события (отправление или прибытие поезда), второй элемент пары — тип события. Возможные типы событий следующие: 1 — отправление от станции 1, 2 — отправление от станции 2, $-1$ — прибытие на станцию 1, $-2$ — прибытие на станцию 2. При сортировке этого списка события упорядочатся сначала по первому элементу (времени), а при равном значении времени — по второму элементу (типу события), поэтому при равном значении времени сначала будут обработаны все прибытия поездов.

Затем события сортируются (можно использовать любой, в том числе и неэффективный алгоритм сортировки), затем последовательно обрабатываются, с изменением количества свободных поездов на станциях, которое хранится в списке `free`.