# Задача 1. Кинотеатр

На 60 баллов можно было написать решение, суммирующее количество мест в каждом ряду: $N$, $N+1$, $N$, $N+1$, ... пока сумма станет не меньше $K$.

Для решения на полный балл заметим, что в двух соседних рядах сумма мест равна $2N+1$, и если разбить ряды на пары, то можно определить, в какую пару рядов попадает $K$-е место, поделив и взяв остаток от деления на $2K+1$. Также нужно потом рассмотреть один из двух случаев: в какой ряд (нечётный или чётный) попало наше место, сравнив остаток от деления на $2K+1$ с числом мест в первом ряду $N$.

Заметим, что в таких задачах удобней нумеровать объекты с 0. В приведённом ниже решении осуществляетя переход в 0-нумерацию, вычитанием числа 1, а при выводе ответа прибавляется 1.

Пример решения.

```
n = int(input())
k = int(input())
k -= 1
rows2 = k // (2 * n +  1)
k %= 2 * n + 1
if k < n:
    print(2 * rows2 + 1, k + 1)
else:
    print(2 * rows2 + 2, k + 1 - n)
```

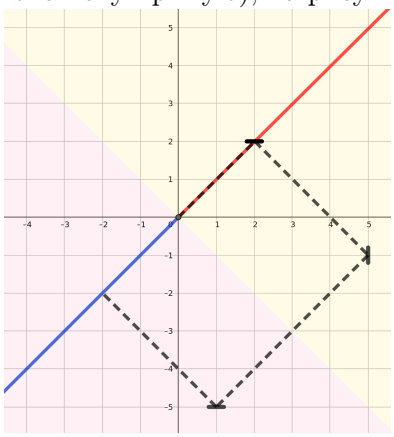
# Задача 2. Лазерная пушка

В этой задаче нужно аккуратно разобрать все возможные случаи.

Заметим, что мы можем попасть только в те целочисленные точки, у которых сумма $x+y$ — чётная. Если сумма нечётная, то ответ «-1».

Для поражения точек, лежащих на луче $y=x$, $x>0$ зеркала не нужны, и ответ в этом случае «0». На рисунке этот луч закрашен красным цветом.

В любом другом случае первое зеркало необходимо поставить на этом луче. Тогда луч можно отразить так, чтобы он прошёл через точки полуплоскости, лежащей выше прямой $y=-x$ (не включая эту прямую), на рисунке ниже эта область закрашена жёлтым цветом.



При этом для того, чтобы поразить в этой области цель ниже прямой $y=x$ (т.е. ниже луча, после его выхода из пушки),нужно поставить горизонтальное зеркало, а для того, чтобы поразить цель выше прямой $y=x$, необходимо вертикальное зеркало. В решении должны быть разобраны эти два случая.

Для поражения оставшейся части плоскости, кроме луча $y=x$, $x<0$ понадобится два зеркала. Эта область покрашена на рисунке розовым цветом. Второе зеркало должно быть ориентировано не так, как первое. Два зеркала также понадобятся для точек на границе жёлтой и розовой областей (это прямая $y=-x$).

Наконец, для поражения цели на луче $y=x$, $x<0$, то есть на продолжении того луча, который выходит из пушки, в противоположную сторону (на рисунке он покрашен в синий цвет), понадобится три зеркала.

На рисунке изображён пример расстановки трёх зеркал для поражения цели на этом луче. Луч изображён пунктирной линией, зеркала — короткими отрезками. Рисунок также иллюстрирует примеры расстановки зеркал и для других случаев.

Пример решения.

```
x = int(input())
y = int(input())

if (x + y) % 2 == 1:  # Нет решения
    print(-1)
elif x == y > 0:  # Нет зеркал
    print(0)
elif -x < y < x:  # Одно зеркало, ниже прямой y = x
    print(1)
    print((x + y) // 2, (x + y) // 2, 'H')
elif y > -x: # Одно зеркало, выше прямой y = x
    print(1)
    print((x + y) // 2, (x + y) // 2, 'V')
elif y < x:  # Два зеркала, ниже прямой y = x
    print(2)
    print(1, 1, 'H')
    print(1 + (x - y) // 2, 1 - (x - y) // 2, 'V')
elif y > x:  # Два зеркала выше прямой y = x
    print(2)
    print(1, 1, 'V')
    print(1 - (y - x) // 2, 1 + (y - x) // 2, 'H')
else:  # Три зеркала, на луче y = x, x < 0
    print(3)
    print(1, 1, 'H')
    print(2, 0, 'V')
    print(x + 1, y - 1, 'H')
```

# Задача 3. Не был предателем...

Для решения на 60 баллов необходимо явно построить строку $S$, повторённую $n$ раз, и найти в ней все вхождения строки $T$ при помощи любого алгоритма.

Заметим, что метод строк `count` языка Python не подходит для такого решения, т.к. он считает количество непересекающихся вхождений, поэтому на языке Python сделаем так. После умножения строки $S$ на $n$ будем находить первое вхождение строки $T$ при помощи метода `find(T)`, а затем будем менять строку на срез этой строки, начиная со следующего символа от начала найденного вхождения $T$ до конца этой строки. Пример такого решения.

```
T = input()
S = input()
n = int(input())
S = S * n
ans = 0
while T in S:
    ans += 1
    S = S[S.find(T) + 1:]
print(ans)
```

Такое решение не пройдёт большие тесты, т.к. строка $S \times n$ будет очень большой и программе даже не хватит памяти для построения такой строки.

Для полного решения заметим, что достаточно изучить те вхождения строки $T$, начало которых находится где-то внутри первого повторения строки $S$ внутри строки $S \times n$, а затем умножить эти вхождения на какое-то число.

Здесь нужно быть аккуратным, если, например, строка $T$ целиком содержится внутри строки $S$, то в строку $S \times n$ она будет входить $n$ раз. В тесте из условия, например, это три вхождения строки «ABA» в строку «BABABABABABA», а именно «BABABABABABA».

Если же строка $T$ не входит целиком в строку $S$, но начинается внутри первого повторения строки $S$, а заканчивается внутри второго повторения строки $S$, то в этом случае строка $T$ попадает "на границу" между соседними повторениями строк $S$, и к ответу нужно добавить $n - 1$. В тесте из условия это соответствует вхождениям «BABABABABABA».

Но возможны и более сложные случаи, когда длина строки $T$ превышает длину строки $S$. Тогда для одного вхождения строки $T$ понадобится не два, а более соседних повторений строки $S$ и к ответу нужно будет добавить $n - 2$ или меньшее число.

Необходимо изучить только те вхождения строки $T$, когда начало строки $T$ находится внутри первого вхождения строки $S$. В этом случае достаточно сделать не $n$ копий строки $S$, а столько копий, чтобы длина получившейся строки была бы не меньше, чем сумма длин строк $S$ и $T$.

Затем переберём все вхождения строки $T$, начало которых находится внутри первого повтора строки $S$, и подсчитаем, сколько повторов строки $S$ "затрагивает" это вхождение строки $T$. В решении ниже это значение равно $k$. Далее к ответу нужно добавить $n + 1 - k$.

Пример такого решения.

```
T = input()
S = input()
n = int(input())

Sn = ""
while len(Sn) < len(T) + len(S):
    Sn += S

ans = 0
for i in range(len(S)):
    if Sn[i:i + len(T)] == T:
        k = (i + len(T) + len(S) - 1) // len(S)
        ans += max(n + 1 - k, 0)

print(ans)
```

# Задача 4. Железная дорога

30 баллов можно набрать, если посекундно моделировать перемещения поездов: внутри цикла изменять координату каждого поезда на $\pm 1$ и считать сумму расстояний поездов до станции (это сумма модулей координат поездов).

Моделирование достаточно продолжать 100 шагов, т.к. если первоначальные координаты поездов по модулю не превосходят 100, то время прохождения каждого поезда мимо станции не превосходит 100, а, значит, через 100 секунд все поезда будут удаляться от станции. Или можно остановить моделирование в тот момент, когда суммарное расстояние всех поездов до станции начнёт увеличиваться.

Пример такого решения.

```
n = int(input())
m = int(input())
a = [int(input()) for i in range(n)]
b = [int(input()) for i in range(m)]

sum = 10 ** 18
prev_sum = 2 * sum;

t = 0
while sum < prev_sum:
```

```
    prev_sum = sum
    sum = 0
    for i in range(n):
        sum += abs(a[i])
        a[i] += 1
    for i in range(m):
        sum += abs(b[i])
        b[i] -= 1
    t += 1
print(t - 2)
```

Это решение не будет работать при больших значениях координат поездов, т.к. при посекундном моделировании количество шагов может достигать $10^9$.

Для прохождения второй группы тестов (на 60 баллов) необходимо заметить, что каждый поезд либо приближается к станции, либо удаляется от станции, и если количество приближающихся поездов больше, чем количество удаляющихся, то суммарное расстояние будет уменьшаться. Изменение количества приближающихся и удаляющихся поездов будет происходить в тот момент, когда поезд проходит через станцию, поэтому в качестве возможного ответа нужно рассматривать не все возможные моменты времени, а только те моменты времени, когда какой-то из поездов проходит через станцию. Таких моментов не более $n$. Переберём все эти моменты и для каждого такого "интересного" момента посчитаем расстояние от всех поездов до станции, найдём тот интересный момент, для котороо эта сумма минимальна.

Какие моменты будут интересными? Из тех поездов, которые движутся в положительном направлении, мимо станции пройдут только те поезда, начальные координаты которых $a_i < 0$, и такой $i$-й поезд пройдёт мимо станции в момент $|a_i| = -a_i$. Среди поездов, которые движутся в отрицательном направлении, наоборот, мимо станции пройдут те поезда, у которых $b_j > 0$, и такой поезд пройдёт мимо станции в момент $b_j$.

Такое решение будет иметь сложность $O(n^2)$. Пример такого решения.

```
n = int(input())
m = int(input())
a = [int(input()) for i in range(n)]
b = [int(input()) for i in range(m)]


def calc_sum(t):
    s = 0
    for i in range(n):
        s += abs(a[i] + t)
    for i in range(m):
        s += abs(b[i] - t)
    return s


ans_t = 0
ans_sum = calc_sum(0)

for i in range(n):
    if a[i] < 0:
        t = -a[i]
        s = calc_sum(t)
        if s <= ans_sum:
            ans_sum = s
            ans_t = t
```

```
for i in range(m):
    if b[i] > 0:
        t = b[i]
        s = calc_sum(t)
        if s <= ans_sum:
            ans_sum = s
            ans_t = t

print(ans_t)
```

Улучшим и это решение. Если количество приближающихся к станции поездов больше, чем количество удаляющихся от станции поездов, то суммарное расстояние всех поездов до станции будет уменьшаться, а если больше — то увеличиваться. Минимум этого расстояния будет достигаться, когда половина всех поездов прошла через станцию.

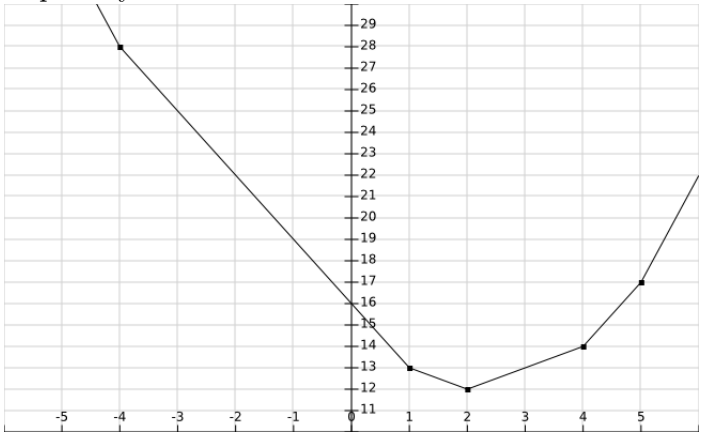
Для каждого поезда определим момент времени, когда этот поезд проедет станцию. Для поездов, движущихся в положительном направлении, это время равно $-a_i$, для поездов, движущихся в отрицательном направлении, это время равно $b_j$. Составим список из всех времён прохождения поездов через станцию, то есть список из значений $-a_i$ и $b_j$, упорядочим его и возьмём средний элемент, то есть элемент с индексом $(n+m)/2$. Получим такое решение, которое имеет сложность $O(n \log n)$

```
n = int(input())
m = int(input())
a = [int(input()) for i in range(n)]
b = [int(input()) for i in range(m)]
times = sorted([-x for x in a] + b)
print(max(0, times[(n + m) // 2]))
```

В этом решении времена прохода некоторых поездов могут быть отрицательными, то есть поезд прошёл через станцию “в прошлом”, а в момент $t = 0$ этот поезд уже движется от станции. Если “медианный” поезд (который и является ответом) имеет отрицательное время прохождения через станцию, то в этом случае уже в момент $t = 0$ большая часть поездов удаляется от станции, поэтому нужно вывести «0». Поэтому возьмём максимум из двух величин: времени прохождения через станцию “медианного” поезда и нуля.

Это решение имеет сложность $O((n+m)\log(n+m))$, так как использует сортировку массива из $n+m$ элементов и набирает 100 баллов. Но сложность можно уменьшить до $O(n+m)$, если заметить, что нам нужно упорядочить конкатенацию (объединение) двух уже упорядоченных массивов, а этом можно сделать за линейную сложность, используя метод двух указателей. Так же следует поступать и в том случае, если используемый язык не содержит функции быстрой сортировки массива.

Рассмотрим и алгебраическую интерпретацию этой задачи. График расстояния поезда до станции от времени — это функция вида $f(x) = |x - x_0|$, где $x_0$ — время прохождения поезда через станцию. Сумма расстояний всех поездов до станции —- это сумма нескольких таких графиков. Такой график будет кусочно-линейной функцией, например, на этом рисунке приведён график для примера из условия.



Этот график имеет изломы в точках, соответствующих временам прохождения поездов через станцию. Для примера из условия эти времена равны $-4$, 1, 2, 4, 5, и минимум будет достигаться в медианной из этих точек. Поэтому нужно взять среднюю точку из всех времён прохождения поездов через станцию, и вывести это значение или число «0», если это значение отрицательно.

В случае чётного числа поездов у этого графика есть горизонтальный участок, и ответом является любое число на отрезке между временами прохождения двух медианных поездов через станцию.

# Задача 5. Джерримендеринг

$X$ выигрывает в каком-то избирательном округе, если сумма числа жителей в данном округе положительна, при суммировании чисел так, как они даны во входных данных, то есть отрицательные значения соответствуют голосующим против $X$.

Таким образом, задача сводится к следующей: данный массив из $N$ элементов нужно разбить на три непустых части размерами $N_1$, $N_2$, $N_3$ так, чтобы минимум в двух частях сумма элементов массива была бы положительной.

Самое простое решение — переберём границы частей (границу между первой и второй частью и границу между второй и третьей частью) двумя вложенными циклами, для каждого полученного разбиения посчитаем сумму чисел в этом разбиении. Если две из трёх сумм будут положительными, то ответ найден.

Сложность такого решения зависит от того, как считать сумму чисел. Если каждый раз пересчитывать сумму заново, то такое решение будет иметь сложность $O(N^3)$ (сложность перебора всех разбиений $O(N^2)$, для каждого разбиения проверка разбиения делается за $O(N)$).

Такое решение набирает 40 баллов. Пример такого решения на языке C++.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n, 0);
    for (int i = 0; i < n; ++i)
        cin >> a[i];
    for (int i = 0; i < n - 2; ++i) {
        for (int j = i + 1; j < n - 1; ++j) {
            int sum1 = 0, sum2 = 0, sum3 = 0;
            for (int k = 0; k <= i; ++k) {
                sum1 += a[k];
            }
            for (int k = i + 1; k <= j; ++k) {
                sum2 += a[k];
            }
            for (int k = j + 1; k < n; ++k) {
                sum3 += a[k];
            }
            if ((sum1 > 0 && sum2 > 0) || (sum2 > 0 && sum3 > 0) || (sum1 > 0 && sum3 > 0)) {
                cout << i + 1 << " " << j - i << " " << n - j - 1 << endl;
                return 0;
            }
        }
    }
    cout << 0 << endl;
}
```

Такое решение можно ускорить, если не пересчитывать все суммы заново для каждого разбиения,

а заметить, что при передвижении границы между частями на один элемент одна сумма уменьшается на значение этого элемента, а другая — увеличивается. Это позволит проверку одного разбиения делать за $O(1)$, используя значения сумм для предыдущего разбиения, а общая сложность решения сократится до $O(N^2)$. Такое решение будет набирать 70 баллов.

Пример такого решения на языке C++.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n, 0);
    int total_sum = 0;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        total_sum += a[i];
    }
    int sum1 = 0, sum2 = 0, sum3 = 0;
    for (int i = 0; i < n - 2; ++i) {
        sum1 += a[i];
        sum3 = 0;
        for (int j = n - 2; j > i; --j) {
            sum3 += a[j + 1];
            sum2 = total_sum - sum1 - sum3;
            if ((sum1 > 0 && sum2 > 0) || (sum2 > 0 && sum3 > 0) || (sum1 > 0 && sum3 > 0)) {
                cout << i + 1 << " " << j - i << " " << n - j - 1 << endl;
                return 0;
            }
        }
    }
    cout << 0 << endl;
}
```

Решение на полный балл имеет сложность $O(N)$. Для этого разберём три возможных варианта ответа: когда положительными будут суммы в первой и второй части (обозначим этот вариант «$+ + -$»), во второй и третьей части (обозначам этот вариант «$- + +$»), в первой и третьей части («$+ - +$»). Решение разбирает эти три случая по отдельности.

Самый простой случай «$+ - +$». Нам нужно найти префикс (начальную часть массива) и суффикс (конечную часть массива) с положительной суммой. При этом между ними должна оказаться непустая вторая часть. Найдём минимальный положительный префикс и минимальный положительный суффикс, если это удалось и между ними есть непустая третья часть, то задача решена. Будем идти от начала массива, считая сумму всех чисел на префиксе, пока она не станет положительной, затем сделаем то же самое с конца.

Первая часть решения на полный балл, разбирающая случай «$+ - +$».

```
import sys

n = int(input())
a = [int(input()) for i in range(n)]

# Case: + - +
sum_prefix = 0
i = 0
while i < n and sum_prefix <= 0:
```

```
    sum_prefix += a[i]
    i += 1

sum_suffix = 0
j = n - 1
while j >= 0 and sum_suffix <= 0:
    sum_suffix += a[j]
    j -= 1

if i <= j:
    print(i, j + 1 - i, n - j - 1)
    sys.exit(0)
```

Случаи «+ + −» и «− + +» сложнее. Посмотрим на случай «+ + −». Здесь нам нужно найти два префикса, оба с положительной суммой, причём сумма элементов между первым и вторым префиксом тоже должна быть положительной. Пусть $i$ — индекс последнего элемента второй части. Мы знаем сумму элементов на префиксе до $i$-го элемента. Тогда найдётся разбиение вида «+ + −», если до $i$-го элемента есть меньший префикс с положительной суммой, причём эта сумма должна быть меньше, чем сумма на префиксе до $i$-го элемента. Тогда из всех меньших префиксов нам достаточно запомнить только один префикс: сумма на котором будет минимальной положительной.

Поэтому будем запоминать из всех просмотренных префиксов тот, у которого сумма будет положительной и минимальный. Эту сумму будем хранить в переменной `best_sum_prefix`, а индекс границы этого префикса — в переменной `best_prev_i`. Когда мы рассматриваем очередной префикс (до $i$-го элемента), сумма на этом префиксе равна `sum_prefix`, и если эта сумма строго больше, чем значение `best_sum_prefix`, то ответ найден. Иначе нужно обновить значение `best_sum_prefix` и `best_prev_i`, если текущая сумма положительна и меньше `best_sum_prefix`.

Фрагмент решения для этого случая (продолжение предыдущего решения).

```
# Case: + + -
sum_prefix = 0
best_sum_prefix = 10 ** 9 + 1
best_prev_i = 0
i = 0
while i < len(a) - 1:
    sum_prefix += a[i]
    if sum_prefix - best_sum_prefix > 0:
        print(best_prev_i + 1, i - best_prev_i, n - 1 - i)
        sys.exit(0)
    if best_sum_prefix > sum_prefix > 0:
        best_sum_prefix = sum_prefix
        best_prev_i = i
    i += 1
```

Случай «− + +» обрабатывается аналогично, только рассматриваются суффиксы. Фрагмент решения для этого случая, с выводом числа 0, если решение не найдно.

```
# Case: - + +
sum_suffix = 0
best_sum_suffix = 10 ** 9 + 1
best_prev_i = n - 1
i = n - 1
while i > 0:
    sum_suffix += a[i]
    if sum_suffix - best_sum_suffix > 0:
        print(i, best_prev_i - i, n - best_prev_i)
        sys.exit(0)
```

```
    if best_sum_suffix > sum_suffix > 0:
        best_sum_suffix = sum_suffix
        best_prev_i = i
    i -= 1

print(0)
```