# Разбор задач муниципального этапа олимпиады по информатике

## 1. Пирожки (все классы)

Тема: вывод формулы
Сложность: простая

Во всех подзадачах необходимо найти минимальное такое C, что C*N <= X (X рублей достаточно на покупку N пирожков), но C*(N+1) > X (X рублей недостаточно на покупку N+1 пирожка).

Подзадача 1

Можно заметить, что C<=X. В противном случае C*N при любом натуральном N было бы больше X, т.е. не удалось бы купить ни одного пирожка.

Значит, искомая величина лежит в пределах отрезка [1; X]. Переберем все натуральные числа до X в порядке возрастания. Первое же подходящее является ответом. Если ни одно из чисел не подходит, выведем -1. Временная сложность такого решения - O(X).

Если завершать перебор сразу же после нахождения первого подходящего решения, то получим вариант с временной сложностью O(C). Поскольку C - натуральное, и C*N <= X, то C не превосходит целой части X/N, и O(C) = O(X/N).

Подзадача 2

С одной стороны, C не превосходит целой части X/N. С другой стороны, поскольку C*(N+1) > X, то C > X/(N+1). То есть, C принадлежит полуинтервалу (X/(N+1); X/N].

Его длина X/N - X/(N+1) = X/(N*(N+1)). Значит, если мы переберем целые значения в этом полуинтервале и выберем минимальное подходящее, временная сложность такого варианта будет O(X/(N*(N+1))). При X<=10^12 и N>=10^3 эта величина не превосходит 10^6.

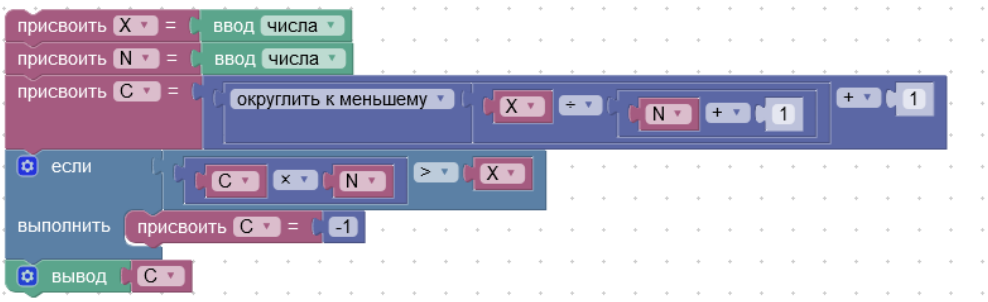
Подзадача 3

От нас требуется найти минимальное целое C, для которого C*(N+1) > X, то есть C > X/(N+1). Заметим, что таким числом является C = [X/(N+1)] + 1, где [] обозначает взятие целой части числа. Для доказательства этого утверждения достаточно рассмотреть случаи X, кратного и не кратного N+1. Если найденное таким образом C удовлетворяет и второму условию задачи (C*N <= X), то оно является ответом. В противном случае, ни одного подходящего числа не существует.

Временная сложность решения — O(1)

Пример реализации:

```
x = int(input())
n = int(input())
c = x // (n+1) + 1
if c*n > x:
    c = -1
print(c)
```



## 2. Алгоритм (7-8 класс)

Тема: реализация программы по схеме алгоритма
Сложность: простая

Пример реализации:

```
n=int(input())
k=0
while not((n==1) or (n==4)): # not так как на схеме выход по ”да”
  k+=1
  m=0
  while n>0:
    m+=(n%10)**2
    n=n//10
  n=m
print(k,n)
```

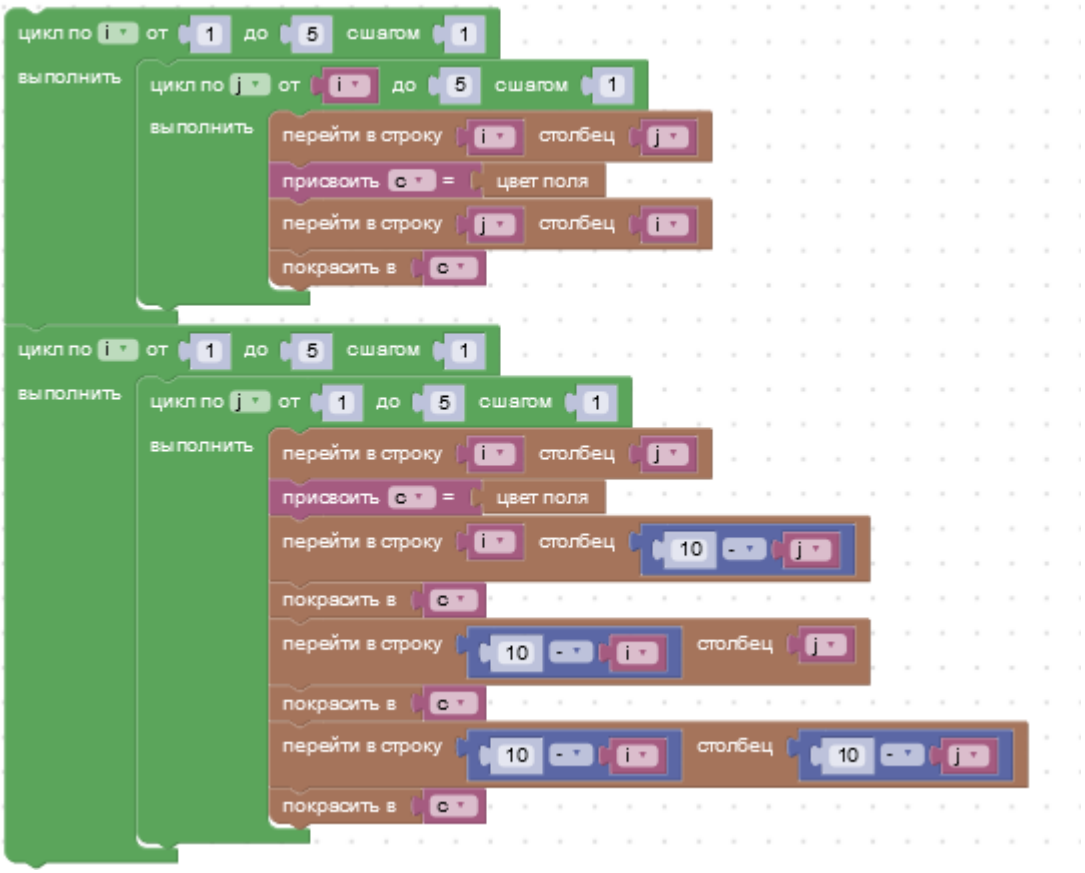
## 3. Рисование снежинки (7-8 классы)

Тема: исполнители, циклы
Сложность: простая

В подзадаче 2 достаточно считать цвет начальной клетки и двойным циклом закрасить все поле.

Для упрощения решения лучше выделить каждое отражение в отдельный цикл, сначала диагональное, затем вертикальное и затем горизонтальное. Можно сэкономить два цикла, выполняя горизонтальное и вертикальное отражение вместе.

Пример реализации:



## 2 или 4. Склад (все классы)

Тема: сортировка, жадность
Сложность: ниже средней

Каждая цепь, отделенная по способу Марины, дает звено, а каждая цепь, собранная из одиночных звеньев, уменьшает количество отделяемых цепей, поэтому выгоднее всего использовать одиночные звенья для формирования маленьких цепей. В подзадаче 1 можно искать самую длинную цепь, затем искать самую короткую цепь и уменьшать ее длину на 1.

Для решения подзадачи 2 необходимо применить сортировку. Тогда все короткие цепи будут в начале списка, а длинные — в конце и вместо поиска можно просто передвигать индекс. Если по завершению цикла остается одна цепь, то ее придется отделять старым способом (при этом добавляя к ней оставшиеся одиночные звенья)

Пример реализации:

```
n=int(input())
a=list(map(int,input().split()))
a.sort()
i=n-1
j=0
ans=0
while j<i:
  a[j]-=1
  if a[j]==0:
    j+=1
  ans+=1
  i-=1
if i==j:
  ans+=1
print(ans)
```

## 3. Коттеджный поселок (9-11 класс)

Тема: разбор случаев, конструктив
Сложность: средняя

Подзадача 1.

Периметр поселка равен 2*(H+W), а внутри него можно проложить от 0 до W-1 включительно внутренних проездов длины 1. Каждый проезд добавляет к общей длине забора 2.
Отсюда алгоритм: находим разность между желаемым периметром и периметром поселка. Разделив пополам (если не делится, то ответ -1), получаем требуемое число проездов.
Если проездов слишком много, ответ также -1. Иначе проводим разрезы произвольным образом, например, отсекая квадратные участки размера 1*1 с одного из краев поселка (левого или правого).

Подзадача 2.

Аналогично подзадаче 1, вычислим общую длину проездов X как половину разности между требуемой длиной заборов и периметром поселка.
Заметим, что если проложить в поселке единственный проезд длины 1, то хотя бы один из его концов будет тупиком, т.е. поселок не разбивается им на прямоугольные участки.
То есть, если требуемая для достижения нужной длины заборов длина проездов равна 1, то получить соответствующий план невозможно, и ответ равняется -1.
Покажем, как достичь любой другой длины проездов, не превышающей максимально возможной.
Выразим длину проездов в виде X = 2*P + Q, где P и Q - неполное частное и остаток от деления X на 2 соответственно. Проведем P вертикальных проездов, отделяющих от поселка прямоугольники высотой 2 и шириной 1. А затем, если Q > 0, то разрежем один из таких прямоугольников на 2 квадрата размера 1*1.

Подзадача 3.

Отдельно рассмотрим случай, когда L=2H+2W. Далее будем предполагать, что поселок разбит хотя бы на 2 прямоугольника-участка.

Лемма 1. Существует пара противоположных сторон поселка, к каждой из которых примыкает хотя бы 1 внутренний проезд.
Доказательство. Предположим обратное: пусть в каждой паре противоположных сторон есть сторона, от которой не отходит ни одного проезда. Тогда есть 2 смежных по углу стороны поселка (одна - из одной пары противоположных сторон, другая - из второй), от которых не отходят проезды. Значит, эта пара сторон целиком принадлежит одному участку. Поскольку участок прямоугольный и имеет стороны, одна из которых равна H, а другая - W, то он совпадает с целым поселком. Получили противоречие, значит предположение неверно.

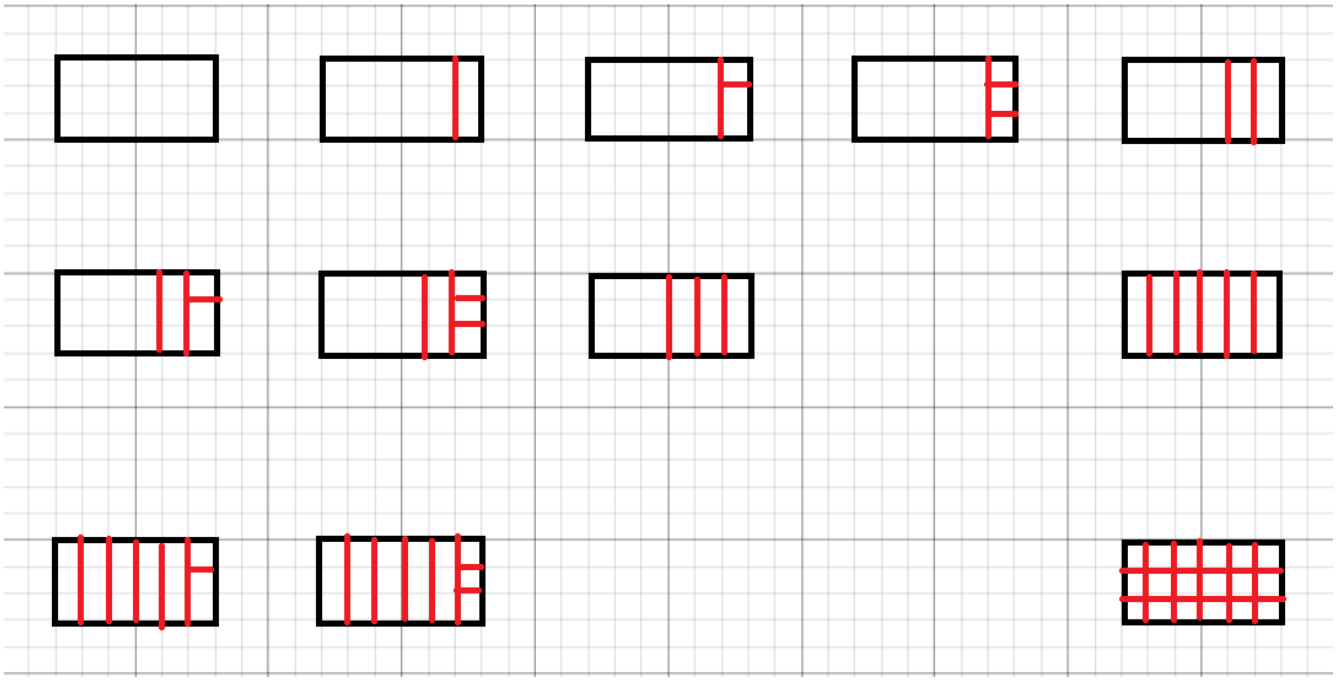
Лемма 2. Существует проезд (не обязательно прямой), который соединяет 2 противоположные стороны поселка.
По лемме 1, имеем 2 противоположные стороны, к каждой из которых подходит хотя бы по 1 проезду. Пусть это верхняя и нижняя сторона (для левой и правой доказательство аналогичное). Пусть эти стороны не связаны между собой проездами. Тогда существует цельный участок, сверху ограниченный проездами, достижимыми от верхней стороны, а снизу - проездами, достижимыми от нижней стороны. Такой участок касается одновременно левой и правой сторон поселка и является прямоугольником - значит, верхняя и нижняя его границы являются прямыми, соединяющими противоположные (левую и правую) стороны прямоугольника. Значит, хотя бы 1 пара противоположных сторон связана между собой.

Лемма 3. Минимальный суммарный периметр, которого можно достичь при разбиении прямоугольника H*W, равен 2H+2W+2*min(H,W)
Минимальная длина проезда, соединяющего противоположные стороны прямоугольника, равна min(H,W). Оценка достигается при наличии единственного прямого проезда, соединяющего 2 противоположные стороны поселка и разрезающего его на 2 прямоугольника.

Лемма 4. Максимальный суммарный периметр, которого можно достичь при разбиении прямоугольника H*W, равен 4*H*W
Если провести все возможные внутренние проезды, получим H*W квадратиков 1*1. Их суммарный периметр равен 4*H*W.

Отсюда следует корректность следующего алгоритма:
1. Если L = 2H+2W, выведем разбиение из 1 участка, совпадающего с целым поселком.
2. Если L нечетна, то ответ -1.
3. Если H>W, то отразим поселок относительно биссектрисы 1 координатной четверти и решим симметричную задачу (при выводе отразим координаты участков обратно).
3. Если L > 4*H*W либо L < 4H+2W, то ответ -1.
4. Вычислим периметр D=2*W*(H+1), который получится, если в прямоугольнике проведены все возможные вертикальные проезды:
4.1 Если требуемый периметр L больше или равен D, то проведем все возможные вертикальные проезды. Затем будем разрезать полученные полосы на квадратики 1*1, пока не достигнем требуемого числа горизонтальных проездов.
4.2 Если требуемый периметр L меньше D, то выразим требуемую длину внутренних проездов (она не превосходит H*(W-1)) в виде H*P + Q (где Q<H, P<W). Проведем P вертикальных проездов, отделяющих от поселка прямоугольники высотой H и шириной 1. Отсечем от крайнего левого из них Q квадратов размера 1*1.

Таким образом мы получим разбиение для любой величины периметра в отрезке до минимального до максимального достижимого значения. Для лучшего понимания процедуры см. рисунок. Общее время работы — O(H*W).



Пример реализации:

```
w, h, val = map(int, input().split())

if val == 2*(h+w):
    print(1)
    print(0, 0, w, h)
    exit(0)

rotate = False
if h > w:
    rotate = True
    h, w = w, h

p_min = 2 * (w + h) + 2*h
p_max = 4*h*w
if val % 2 or val > p_max or val < p_min:
    print(-1)
    exit(0)

res = []

#  общий периметр, если внутри сделаны все возможные вертикальные разрезы
#  имеем W вертикальных полос, каждая размера 1*H, т.е. с периметром 2(H+1)
p_diff = 2*w * (h+1)
if val >= p_diff:
    #  все возможные вертикальные разрезы сделаны,
    #  проводим требуемое количество горизонтальных
    #  каждый раз отрезая квадраты 1*1
    val -= p_diff
    for c in range(w):
        for r in range(h):
            #  горизонтальные разрезы больше не нужны
            #  используем вертикальную полосу (или её часть) целиком
            if val == 0:
                res.append((c, r, c+1, h))
                break

            #  горизонтальные разрезы еще нужны
            #  отрезаем квадрат 1*1 от текущей полосы
            res.append((c, r, c+1, r+1))
            if r < h - 1:
                val -= 2
else:
    #  длина внутренних разрезов
    val -= 2*(h+w)
    val //= 2
    #  сколько из них вертикальных и горизонтальных
    n_vert = val // h
    n_hor = val % h

    #  отделяем квадраты 1*1 горизонтальными разрезами от крайней полосы
    for r in range(n_hor):
        res.append((0, r, 1, r+1))
    res.append((0, n_hor, 1, h))

    #  отделяем полосы 1*H вертикальными разрезами
    for c in range(1, n_vert):
        res.append((c, 0, c+1, h))
    res.append((n_vert, 0, w, h))

print(len(res))
for rect in res:
    if rotate:
        print(rect[1], rect[0], rect[3], rect[2])
    else:
        print(rect[0], rect[1], rect[2], rect[3])
```

## 5. Справедливые подмассивы-1 (7-8 класс)

Тема: частичные суммы, ассоциативные массивы
Сложность: средняя

Подзадача 1

Переберем все подмассивы входного массива. Для каждого подмассива пройдем по его элементам и вычислим отдельно суммы четных и нечетных его элементов. Если суммы оказались равны, значит подмассив - справедливый, и ответ нужно увеличить на 1. Временная сложность решения - O(N^3).

Подзадача 2

Массив является справедливым тогда и только тогда, когда знакочередующаяся сумма его элементов равна нулю, т.е. a[L] - a[L+1] + a[L+2] - a[L+3] +... = 0. Чтобы находить значение подобной знакочередующейся суммы для любого подмассива за O(1), подсчитаем префиксные знакочередующиеся суммы для входного массива:

prefix[0] = 0
prefix[1] = prefix[0] + a[0] = a[0]
prefix[2] = prefix[1] - a[1] = a[0] - a[1]
prefix[3] = prefix[2] + a[2] = a[0] - a[1] + a[2]
...

Тогда значение суммы для подмассива от L до R равно разности prefix[R+1] - prefix[L]. Значит, ответ равен количеству пар индексов (L; R) Таких, что prefix[L] = prefix[R+1]. Переберем все пары индексов и подсчитаем среди них количество подходящих. Временная сложность решения — O(N^2).

```
n = int(input())
a = list(map(int, input().split()))
pref = [0]
for i in range(n):
    pref.append(pref[-1] + (a[i] if i % 2 else -a[i]))
cnt = 0
for left in range(n):
    for right in range(left, n):
        if pref[right+1] == pref[left]:
            cnt += 1
print(cnt)
```

Подзадача 3

Аналогично предыдущей подзадаче, подсчитаем массив префиксных знакочередующихся сумм для исходного массива и попробуем найти количество пар индексов с совпадающими значениями этих сумм быстрее, чем за O(N^2). Суммы принимают значения из отрезка [-N*max(a_i); N*max(a_i)]. Подсчитаем, сколько раз префиксные суммы принимают каждое из значений X в этом отрезке и сохраним результат в элементе массива cnt[X + N*max(a_i)] (добавление N*max(a_i) требуется для того, чтобы сделать индекс неотрицательным). Тогда для каждого X возможно cnt[X]*(cnt[X]-1)/2 вариантов отрезка с концами, в которых префиксная сумма имеет соответствующее значение. Переберем все возможные значения суммы и для каждой из них вычислим ответ по формуле. Временная сложность решения - O(N*max(a_i)).

Подзадача 4

Поступим аналогично предыдущей подзадаче, но вместо массива счетчиков размером 2*N*max(a_i) +1 заведем ассоциативный массив (словарь). Ключом в словаре будут являться значения знакочередующейся префиксной суммы, значением - сколько раз такая сумма встречается в массиве. В словаре будет O(N) элементов, которые можно перебрать и вычислить для каждого количество соответствующих подмассивов по формуле. Временная сложность решения - O(N) либо O(NlogN) в зависимости от используемой реализации словаря.

Пример реализации:

```
n = int(input())
a = list(map(int, input().split()))
cnt = dict()
pref = 0
cnt[0] = 1
for i in range(n):
    pref += a[i] if i % 2 else -a[i]
    if pref in cnt:
        cnt[pref] += 1
    else:
        cnt[pref] = 1
result = 0
for c in cnt.values():
    result += c*(c-1)//2
print(result)
```

# 4. Справедливые подмассивы-2 (9-11 класс)

Тема: частичные суммы, ассоциативные массивы, преобразование условия
Сложность: выше средней

Подзадачи 1-4 аналогично 7-8 классу

Подзадача 5

Пройдем по массиву слева направо и будем поддерживать K отдельных префиксных сумм: prefix[i][j] (j = 0...K-1) будет являться суммой элементов на всех позициях слева от текущей позиции i, номера которых сравнимы с j по модулю K. Вычислим в каждой позиции K-1 попарную разность между префиксными суммами в этой позиции, например:

prefix[i][1] - prefix[i][0]
prefix[i][2] - prefix[i][0]
...
prefix[i][K-1] - prefix[i][0]

Если для пары позиций наборы таких разностей совпадают, то заключенный между ними подмассив - справедливый. Чтобы показать это, достаточно вычислить разность между суммами 2х людей на этом подмассиве, выразив их как разности соответствующих префиксных сумм и перегруппировав слагаемые в полученном выражении.

При движении по массиву будем поддерживать словарь, ключом в котором будет являться набор разностей, а значением - счетчик, сколько раз такой набор разностей уже встречался. Тогда общее количество подходящих подмассивов получим суммированием по формуле, аналогично предыдущей подзадаче.

Пример реализации:

```
n, k = map(int, input().split())
a = list(map(int, input().split()))
sums = [0]*k
tmp = [0]*(k-1)
cnt = {tuple(tmp): 1}
result = 0
for i in range(n):
    sums[i % k] += a[i]
    for j in range(1, k):
        tmp[j-1] = sums[j] - sums[0]
    tmp2 = tuple(tmp)
    if tmp2 in cnt:
        result += cnt[tmp2]
        cnt[tmp2] += 1
    else:
        cnt[tmp2] = 1

print(result)
```

```
#include <iostream>
#include <vector>
#include <map>
using namespace std;
typedef long long ll;
int main() {
    int n, k;
    cin >> n >> k;
    vector<ll> a(n);
    for (auto& t : a)
        cin >> t;
    vector<ll> sum(k, 0);
    map<vector<ll>, int> cnt;
    ll ans = 0;
    vector<ll> tmp(k-1, 0);
    cnt[tmp] = 1;
    for (int i = 0; i < n; ++i) {
        sum[i % k] += a[i];
        for (int j = 1; j < k; j++)
            tmp[j-1] = sum[j] - sum[0];
        ans += cnt[tmp]++;
    }
    cout << ans << "\n";
}
```

# 5. Круговая порука (9-11 класс)

Тема: моделирование, перебор комбинаций
Сложность: выше средней

Подзадача 1

Рассмотрим 2 случая: первый человек - честный, и первый человек - коррупционер.

Случай 1: если честный человек называет число 0, то следующий за ним - тоже честный. Если называет число 1, следующий за ним - коррупционер.

Случай 2: если коррупционер называет число 0 или число 2, то следующий за ним - тоже коррупционер. Если же он называет число 1, то за ним находится честный человек.

Таким образом, рассмотрев 2 варианта, всех остальных далее мы однозначно определяем по цепочке, одного за другим. На последнем шаге (при замыкании круга) может возникнуть противоречие в одном из вариантов - значит, этот вариант из двух невозможен. Выберем минимальное и максимальное количество коррупционеров среди двух (или одного, если возникло противоречие) вариантов.

Подзадача 2

Переберем все возможные 2^N комбинаций честных и коррумпированных людей на N местах. Для каждой комбинации подсчитаем, сколько коррупционеров сидит справа от каждого из людей за O(N*K) либо O(N+K) операций (непосредственным вычислением, либо с использованием скользящего окна). Далее просто проверим, не противоречат ли полученные числа ответам людей: для честных ответы должны быть в точности равны, для коррумпированных - отличаться ровно на 1. Среди всех вариантов, в которых противоречий не обнаружилось, выберем минимальный и максимальный.

Подзадача 3

Переберем все возможные комбинации первых K человек за 2^K. Далее, зная K человек подряд, всегда можем однозначно восстановить следующего:

- если 1ый из известных K - честный, то просто смотрим на разность между названным им количеством коррупционеров и реальным количеством среди следующими за ним K-1 людьми.
- если 1ый - коррупционер, то из двух возможных вариантов (честный, коррумпированный) подходить всегда будет ровно 1 (т.к. ответы коррупционера будут как минимум отличаться четностью от ответов честного человека).

Если в какой-то момент данные противоречивы, то соответствующий вариант просто перестаем рассматривать.

Общее время работы O(2^K * N*K). В реальности не существует теста, в котором сразу все возможные комбинации не будут приводить к противоречиям, поэтому реальное время работы при немедленном отсечении неподходящих вариантов будет гораздо меньше.

Пример реализации

```
n, k = map(int, input().split())
a = list(map(int, input().split()))

min_res = n+1
max_res = -1
for mask in range(2**k):
    good = True
    c = [0] * (n+k)
    for i in range(k):
        if mask & (1 << i):
            c[i] = 1

    for i in range(n):
        s = c[i+1:i+k].count(1)
        if c[i] == 0:
            c[i+k] = a[i] - s
            if c[i+k] < 0 or c[i+k] > 1:
                good = False
                break
        else:
            opts = [a[i] - 1, a[i] + 1]
            if s+1 in opts:
                c[i+k] = 1
            elif s in opts:
                c[i+k] = 0
            else:
                good = False
                break
```

```
        if i + k >= n and c[i+k] != c[i+k-n]:
            good = False
            break

    if good:
        c = c[:n]
        cnt = c.count(1)
        #  print(c)
        min_res = min(min_res, cnt)
        max_res = max(max_res, cnt)

print(min_res, max_res)
```