МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
Заключительный этап
Профиль Информационные технологии
Междисциплинарные задачи

I Вариант

Задача 1 (20 баллов)

На входе сначала строка с числом N (N=5), а потом текст из N строк. В каждой строке N чисел, разделенных пробелами. Данный текст представляет собой отношения между населенными пунктами (матрицу смежности) и расстояниями между ними (если пути нет – длина равна 0). Населенные пункты не имеют названий и пронумерованы от 1 до N.

Входные данные не гарантируются, программа должна завершаться при некорректном вводе и вывести «Repeat».

Программа должна выполнять следующие задания:

Произвести ввод строк текста из потока ввода, вывести матрицу на экран. Вывести является ли описываемый матрицей граф ориентированным (directed) или неориентированным (undirected).

Представить статистику, выведя на экран общую длину и количество дорог длиной менее 3.

Формат выходных данных соблюдать как показано в примере.

Пример входных данных	Пример выходных данных
5 0 1 1 0 4 1 0 1 1 3 1 1 0 1 1 0 1 1 0 1 4 3 1 1 0	0 1 1 0 4 1 0 1 1 3 1 1 0 1 1 0 1 1 0 1 4 3 1 1 0 undirected 7 7

Пример решения:

```
#include <iostream>
#include <vector>

int main()
```

```
{
   int n = 0;
   if (!(std::cin >> n)) return 1;
   if (n!=5) { std::cout << "repeat\n";
   } else {
   std::vector<std::vector<int>> matr(n);
   for (size_t i = 0; i < n; i++)
   {
      matr[i] = std::vector<int>(n);
   }

   for (size_t i = 0; i < n; i++)
   {
      for (size_t j = 0; j < n; j++)
      {
         if (!(std::cin >> matr[i][j])) return 1;
      }
   }

   std::vector<std::vector<int>> matrT(n);
   for (size_t i = 0; i < n; i++)
   {
      matrT[i] = std::vector<int>(n);
   }

   for (size_t i = 0; i < n; i++)
   {
      for (size_t j = 0; j < n; j++)
      {
         matrT[j][i] = matr[i][j];
      }
   }

   bool flag_dir = false;
   bool flag_weight = false;
   for (size_t i = 0; i < n; i++)
   {
      if (!flag_dir || !flag_weight)
      {
         for (size_t j = 0; j < n; j++)
         {
            if (matrT[i][j] != matr[i][j])
            {
```

```
                flag_dir = true;
            }
            if (matr[i][j]>1)
            {
                flag_weight = true;
            }
        }
    }
    else break;
}


//ВЫВОД МАТРИЦЫ
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
    std::cout << matr[i][j] << " ";
 std::cout << "\n";
}
if (!flag_dir) std::cout << "un";
std::cout << "directed\n";
//if (!flag_weight) std::cout << "un";
//std::cout << "weighted\n";

//СЧИТАЕМ КОЛИЧЕСТВО ДОРОГ и ДЛИНУ НЕ БОЛЕЕ ЧЕМ 2
int k = 1;
int summa = 0;
int kol = 0;
for (int i = 0; i < n; i++)
{
for (int j = k; j < n; j++){
if ((matr[i][j]<3)&&(matr[i][j]>0)){
kol++;
summa = summa + matr[i][j];
}
}
 k++;
}
std::cout << summa;
std::cout << "\n";
std::cout << kol;
std::cout << "\n";
}
```

}

Критерии оценивания:

0 - решение не прошло тестирование;

6 - решение прошло только тесты базовой группы;

12 - решение прошло тесты базовой и средней групп;

20 - решение прошло все тесты.

Задача 2 (30 баллов)

В период пандемии рациональное распределение времени работы врача – одна из важнейших задач. Крайне важно чтобы в отделении больницы всегда был хотя бы один врач. Также важно чтобы врачи хорошо отдыхали, поэтому в отделении должно всегда находится минимальное возможное количество врачей. Всего в отделении работает N врачей. От Вас не требуется составить график работы отделения больницы, а лишь определить по данному графику на период от a до b, удовлетворяет ли он заданным требованиям. У каждого врача известны моменты времени заступления на дежурство и окончания дежурства s и t. В момент времени s считается, что врач уже находится на дежурстве, а в момент времени t дежурство уже закончено. Время дается в пределах одного календарного года, в часах с начала года.

Входные данные

В первой строке даны два целых неотрицательных числа a и b $(0 \le a < b \le 8760)$

Во второй строке дано единственное целое положительное число N $(1 \le N \le 10^5)$ – количество врачей, работающих в отделении больницы.

В каждой из последующих N строк содержится два целых неотрицательных числа s_i и t_i $(a \le s_i < t_i \le b)$ – время заступления на дежурство и окончания дежурства i-го врача.

Выходные данные

В единственно строке выходного потока необходимо вывести «YES», если график удовлетворяет требованиям, «NO» - в противном случае.

Пример

стандартный ввод	стандартный вывод
0 24 4 0 8 8 16 14 22 18 24	YES
0 24 4 0 8 10 16 14 22 18 24	NO
0 24 5 0 8 8 16 14 22 15 21 18 24	NO

Пример решения:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>

using namespace std;

struct Point {
```

```
int x; //time
int type; //open == 0 or close == 1
int num; //segment's number
int len; //segment's length
};

//operator for srict wick order
bool operator<(Point& p1, Point& p2) {
if (p1.x == p2.x) {
if (p1.type == 0 && p2.type == 0) {
if (p1.len == p2. len) {
return p1.num < p2.num;
}
return p1.len > p2.len;
}
else if (p1.type == 1 && p2.type == 1) {
if (p1.len == p2. len) {
return p1.num > p2.num;
}
return p1.len < p2.len;
}
return p1.type < p2.type;
}
else {
return p1.x < p2.x;
}
}

void fastIO() {
ios_base::sync_with_stdio(false);
cin.tie(NULL);
}

int main() {
fastIO();
int s, t, n;
cin >> s >> t >> n;
vector<Point> points(2*n);

for (int i = 0; i < n; i++) {
cin >> points[2*i].x;
points[2*i].type = 0;
points[2*i].num = i;
```

```
cin >> points[2*i + 1].x;
points[2*i + 1].type = 1;
points[2*i + 1].num = i;
points[2*i + 1].len = points[2*i].len = points[2*i + 1].x - points[2*i].x;
}
sort(points.begin(), points.end());

int count = 0;
bool flag = true;
int lastOpened;
for (int i = 0; i < points.size(); i++) {
if (points[i].type == 0) {
lastOpened = points[i].num;
count++;
if (count == 1 && points[i].x > s) {
flag = false;
break;
}
if (count > 2) {
flag = false;
break;
}
}
else {
count--;
if (lastOpened == points[i].num && count > 0 || count == 0 && points[i].x < t) {
flag = false;
break;
}
}
}
if (flag) {
cout << "YES";
}
else {
cout << "NO";
}
}
```

Критерии оценивания:

0 - решение не прошло тестирование;

10 - решение прошло только тесты базовой группы;

20 - решение прошло тесты базовой и средней групп;

30 - решение прошло все тесты.

Задача 3 (50 баллов)

Для составления слов языка Тумбу - Юмбу используются n букв. Буквы в слове не повторяются, а их порядок не важен ("abc" и "cba" считаются одним и тем же словом). Слово может состоять из одной и более букв.

В столбик записано m разных слов из языка племени Тумбу-Юмбу. Вы дописывает в конец столбика ещё одно слово длины x (оно может повторять уже имеющиеся). И перемещаете буквы из слова в слово так, чтобы получившиеся после перестановок слова были одинаковой длины. Слова, получившиеся после перестановок, могут повторять друг друга.

В качестве входных данных вы получаете:

n - число букв языка Тумбу - Юмбу

m - количество слов записанных в столбик

l_1, l_2 .... l_m - длины слов записанных в столбик

Ваша задача рассчитать:

x - длину слова, которое вы дописали

y - наименьшее количество ходов, необходимых для перестановок букв в словах.

Перемещение буквы в соседнее слово происходит за 1 ход, через одно слово за 2 хода, через два слова за 3 хода и т.д.

Имейте в виду, что входные данные, могут описывать ситуацию, не отвечающую условиям задачи,

в этом случае вы должны вывести:

0

0

Ограничения

Для решения необходимо использовать стандартные потоки ввода вывода.

Число n лежит в интервале [1, 20]

Число n лежит в интервале [1, 200]

Примеры ввода/вывода

Ввод	**Вывод**
4 5 1 1 1 1 1	0 0
4 5 4 2 3 1 3	5 5

Пример решения:

Для начала следует убедиться, что входные данные удовлетворяют условию задачи.

Есть два варианта, при которых данные можно считать некорректными:

По условию задачи, буквы в слове повторяться не могут, значит максимальная длинна слова равна числу букв в алфавите (n).

По условию задачи, изначально записанные в столбик слова не могут повторяться. Значит слов конкретной длинны k, может быть ограниченное количество. Сколько слов длины k мы можем составить из алфавита размерности n? Ответом будет количество размещений из n по k:

$$A_n^k = \frac{n!}{(n-k)!}$$

С учётом того, что порядок букв в слове не важен ("abc" и "cba" считаются одним и тем же словом), получается, что разных слов длины k может быть:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

.

Необходимо понять, какой длины мы получим слова после всех перестановок, от этого зависит и длина слова, которое мы допишем.

Разделим число всех букв в исходных словах, на число слов. Если результат деления целое число, то переставляя буквы в уже имеющихся словах, мы можем получить слова одинаковой длины и к исходным словам нам нужно дописать слово такой же длины.

Если результат нецелое число, то переставляя буквы в исходных словах мы получим слова двух длин. Если всего символов в исходных словах w, то у нас получится (w mod m) слов длины (w div m +1) и (m – w mod m) слов длины (w div m). Соответственно в слове, которое мы допишем должно быть (w div m +1) + (m – w mod m) букв.

Теперь, остаётся понять сколько ходов нам потребуется для перемещения букв и уравнивания длин всех слов. Мы можем беспрепятственно перемещать буквы из более длинных слов в более короткие, поскольку буквы в слове не повторяются, в длинном слове всегда есть буквы, которых нет в коротком.

Посчитаем для каждого слова, включая дописанное, сколько символов в слове не хватает либо сколько из него нужно переместить. Запишем результаты подсчётов в столбик, также как слова. Причём, недостаток символов будем записывать с знаком «-», а избыток с знаком «+». Например, для слов с длинами [2, 4, 2, 3, 4] получится [-1, 1, -1, 0, 1]. Для каждой линии, разделяющей слова, мы можем посчитать сколько букв должны пересечь для уравнивания длин слов. Чтобы посчитать это, достаточно найти сумму чисел справа от линии. Для нашего случая [-1 | 1, -1, 0, 1], сумма чисел справа от линии равна 1 + (-1) + 0 + 1, т.е. 1. Это значит, что из правой части в левую, нужно переместить одну букву, это один ход. Если бы сумма была -2, это значило-бы, что переместить надо две буквы, но из левой части в правую, это будет 2 хода. Для каждой линии нужно посчитать

число ходов, а затем просуммировать их. Это и будет число ходов, необходимое для уравнивания длин слов.

```
# coding: utf-8
import math
import numpy as np

n = int(input())  # букв в алфавите
m = int(input())  # слов перед нами
nums = map(int, input().split())
words = np.array(list(nums))
# print(words)

if n < np.max(words):  # есть слово, длина которого больше числа букв в алфавите
    print(0)
    print(0)
    exit()

words_len, words_count = np.unique(words, return_counts=True)
i = 0
while i < len(words_len):  # слов конкретной длины больше, чем их может быть составленно из алфивита
    if math.factorial(n) / math.factorial(n - words_len[i]) / math.factorial(words_len[i]) < words_count[i]:
        print(0)
        print(0)
        exit()
    i += 1

# исходные данные верны!
# рассчитываем длину дополнительного слова
all_symbol = np.sum(words)
long_words = all_symbol % m  # столько слов получится длины new_len
short_words = m - long_words  # столько слов получится длины new_len - 1
if long_words == 0:
    new_len = all_symbol // m
    print(new_len)  # длина дополнительного слова
    # не дописываем новое слово к столбцу - оно не скажется на числе ходов, необходимых для перестановок
else:
    new_len = all_symbol // m + 1
```

```
    x = short_words + new_len
    print(x)  # длина дополнительного слова
    words = np.append(words, x)  # допишем новое слово в конец столбца

# рассчитываем число ходов
balance = np.zeros_like(words)
balance = words - new_len
# print(balance)
moves = 0
i = 1
while i < len(balance):
    left = np.sum(balance[:i])
    # print("left", balance[:i], left)
    # right = np.sum(balance[i:])
    # print("right", balance[i:], right)
    moves += abs(left)
    i += 1

print(moves)
```

Критерии оценивания:

0 - решение не прошло тестирование;

16 - решение прошло только тесты базовой группы;

32 - решение прошло тесты базовой и средней групп;

50 - решение прошло все тесты.

МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
Заключительный этап
Профиль Информационные технологии
Междисциплинарные задачи

II Вариант

Задача 1 (20 баллов)

На входе сначала строка с числом N (N=6), а потом текст из N строк. В каждой строке N чисел, разделенных пробелами. Данный текст представляет собой отношения между населенными пунктами (матрицу смежности) и расстояниями между ними (если пути нет – длина равна 0). Населенные пункты не имеют названий и пронумерованы от 1 до N.

Входные данные не гарантируются, программа должна завершаться при некорректном вводе и вывести «Repeat».

Программа должна выполнять следующие задания:

Произвести ввод строк текста из потока ввода, вывести матрицу на экран. Вывести является ли описываемый матрицей граф взвешенным (weighted) или невзвешенным (unweighted).

Вывести на экран общую длину и количество дорог длиной более 2.

Формат выходных данных соблюдать как показано в примере:

Пример входных данных	Пример выходных данных
6 0 2 3 4 5 6 2 0 4 5 6 1 3 4 0 6 1 2 4 5 6 0 2 3 5 6 1 2 0 4 6 1 2 3 4 0	0 2 3 4 5 6 2 0 4 5 6 1 3 4 0 6 1 2 4 5 6 0 2 3 5 6 1 2 0 4 6 1 2 3 4 0 weight 46 10

Пример решения:

```
#include <iostream>
#include <vector>

int main()
```

```
{
    int n = 0;
    if (!(std::cin >> n)) return 1;
    if (n!=6) { std::cout << "repeat\n";
    } else {
    std::vector<std::vector<int>> matr(n);
    for (size_t i = 0; i < n; i++)
    {
        matr[i] = std::vector<int>(n);
    }

    for (size_t i = 0; i < n; i++)
    {
        for (size_t j = 0; j < n; j++)
        {
            if (!(std::cin >> matr[i][j])) return 1;
        }
    }

    std::vector<std::vector<int>> matrT(n);
    for (size_t i = 0; i < n; i++)
    {
        matrT[i] = std::vector<int>(n);
    }

    for (size_t i = 0; i < n; i++)
    {
        for (size_t j = 0; j < n; j++)
        {
            matrT[j][i] = matr[i][j];
        }
    }

    bool flag_dir = false;
    bool flag_weight = false;
    for (size_t i = 0; i < n; i++)
    {
        if (!flag_dir || !flag_weight)
        {
            for (size_t j = 0; j < n; j++)
            {
                if (matrT[i][j] != matr[i][j])
                {
```

```
                flag_dir = true;
            }
            if (matr[i][j]>1)
            {
                flag_weight = true;
            }
        }
    }
    else break;
}


    //ВЫВОД МАТРИЦЫ
    for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
        std::cout << matr[i][j] << " ";
     std::cout << "\n";
}
    //if (!flag_dir) std::cout << "un";
    //std::cout << "directed\n";
    if (!flag_weight) std::cout << "un";
    std::cout << "weighted\n";

    //СЧИТАЕМ КОЛИЧЕСТВО ДОРОГ и ДЛИНУ  БОЛЕЕ ЧЕМ 2
    int k = 1;
    int summa = 0;
    int kol = 0;
    for (int i = 0; i < n; i++)
{
    for (int j = k; j < n; j++){
if (matr[i][j]>2){
kol++;
summa = summa + matr[i][j];
}
}
     k++;
}
std::cout << summa;
std::cout << "\n";
std::cout << kol;
std::cout << "\n";
    }
```

}

Критерии оценивания:

0 - решение не прошло тестирование;

6 - решение прошло только тесты базовой группы;

12 - решение прошло тесты базовой и средней групп;

20 - решение прошло все тесты;

Задача 2 (30 баллов)

Каждому хирургу во время операции ассистирует медицинская сестра, которая помогает проводить операцию. Например, следит за тем, чтобы на столе рядом с хирургом всегда находился тот инструмент, который понадобиться ему в данный момент. Всего имеется N различных инструментов. Проблема в том, что стол для инструментов может вместить не более K инструментов, поэтому медицинской сестре приходится убирать какой-то инструмент, а на его место класть необходимый. Для автоматизации этого процесса был создан специальный робот-манипулятор, который, анализируя с помощью алгоритмов компьютерного зрения действия хирурга, может предугадывать последовательность, в которой хирургу понадобится инструменты. Таким образом робот-манипулятор способен минимизировать количество операций перемещения инструментов, что позволит как можно меньше отвлекать хирурга от операции. Ваша задача состоит в том, чтобы определить минимальное количество операций перемещений инструментов, зная последовательность, в которой хирургу понадобятся инструменты. Перед началом операции стол был пустой.

Входные данные

В первой строке даны три числа N, K и $P (1 \leq \mathrm{K} \leq \mathrm{N} \leq 100000, 1 \leq \mathrm{P} \leq 500000)$.

В следующих P строках записаны номера инструментов в том порядке, в котором они понадобятся хирургу.

Выходные данные

В единственно строке выходного потока необходимо вывести единственное целое число: минимальное количество операций перемещения инструментов.

Пример

стандартный ввод	стандартный вывод
3 2 7 1 2 3 1 3 1 2	4

Примечание к примеру тестов

Операция 1: положить на стол инструмент 1

Операция 2: положить на стол инструмент 2

Операция 3: убрать со стола инструмент 2 и положить на стол инструмент 3

Операция 4: убрать со стола инструмент 3 или 1 и положить на стол инструмент 2.

Пример решения:

```
#include <iostream>
#include <queue>
#include <unordered_set>
#include <vector>
#include <list>

using namespace std;

int main() {
int n, k, p;
cin >> n >> k >> p;
int INF = p + 1;
```

```
vector<int> seq(p);
vector<queue<int, list<int> > > next(n + 1);
for (int i = 0; i < p; i++) {
cin >> seq[i];
next[seq[i]].push(i);
}
for (int i = 1; i <= n; i++) {
if (!next[i].empty()) {
next[i].pop();
next[i].push(INF);
}
}
int moves = 0;
priority_queue<pair<int, int> > maxPQ;
unordered_set<int> onTable;
for (int i = 0; i < p; i++) {
//remove intrument from tabel
if (onTable.size() >= k && onTable.find(seq[i]) == onTable.end()) {
onTable.erase(maxPQ.top().second);
maxPQ.pop();

}

// update priority
maxPQ.push({ next[seq[i]].front(), seq[i] });
next[seq[i]].pop();
if (onTable.find(seq[i]) == onTable.end()) {
onTable.insert(seq[i]);
moves++;
}
}
cout << moves;
return 0;
}
```

Критерии оценивания:

0 - решение не прошло тестирование;

10 - решение прошло только тесты базовой группы;

20 - решение прошло тесты базовой и средней групп;

30 - решение прошло все тесты.

МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
Заключительный этап
Профиль Информационные технологии
Междисциплинарные задачи

Задача 3 (50 баллов)

Для составления слов языка Тумбу - Юмбу используются n букв. Буквы в слове не повторяются, а их порядок не важен ("abc" и "cba" считаются одним и тем же словом). Слово может состоять из одной и более букв.

В столбик записано m разных слов из языка племени Тумбу-Юмбу. Вы дописывает в конец столбика ещё одно слово длины x (оно может повторять уже имеющиеся). И перемещаете буквы из слова в слово так, чтобы получившиеся после перестановок слова были одинаковой длины. Слова, получившиеся после перестановок, могут повторять друг друга.

В качестве входных данных вы получаете:

n - число букв языка Тумбу - Юмбу

m - количество слов записанных в столбик

l_1, l_2 .... l_m - длины слов записанных в столбик

Ваша задача рассчитать:

x - длину слова, которое вы дописали

y - наименьшее количество ходов, необходимых для перестановок букв в словах.

Перемещение буквы в соседнее слово происходит за 1 ход, через одно слово за 2 хода, через два слова за 3 хода и т.д.

Имейте в виду, что входные данные, могут описывать ситуацию, не отвечающую условиям задачи,

в этом случае вы должны вывести:

0

0

Ограничения

Для решения необходимо использовать стандартные потоки ввода вывода.

Число n лежит в интервале [1, 20]

Число n лежит в интервале [1, 200]

Примеры ввода/вывода

МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
Заключительный этап
Профиль Информационные технологии
Междисциплинарные задачи

Ввод	**Вывод**
4 5 1 1 1 1 1	0 0
4 5 4 2 3 1 3	5 5

Пример решения:

Для начала следует убедиться, что входные данные удовлетворяют условию задачи.

Есть два варианта, при которых данные можно считать некорректными:

По условию задачи, буквы в слове повторяться не могут, значит максимальная длинна слова равна числу букв в алфавите (n).

По условию задачи, изначально записанные в столбик слова не могут повторяться. Значит слов конкретной длинны k, может быть ограниченное количество. Сколько слов длины k мы можем составить из алфавита размерности n? Ответом будет количество размещений из n по k: $A_n^k = \frac{n!}{(n-k)!}$.

С учётом того, что порядок букв в слове не важен ("abc" и "cba" считаются одним и тем же словом), получается, что разных слов длины k может быть:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

.

Необходимо понять, какой длины мы получим слова после всех перестановок, от этого зависит и длина слова, которое мы допишем.

МОСКОВСКАЯ ПРЕДПРОФЕССИОНАЛЬНАЯ ОЛИМПИАДА ШКОЛЬНИКОВ
Заключительный этап
Профиль Информационные технологии
Междисциплинарные задачи

Разделим число всех букв в исходных словах, на число слов. Если результат деления целое число, то переставляя буквы в уже имеющихся словах, мы можем получить слова одинаковой длины и к исходным словам нам нужно дописать слово такой же длины.

Если результат нецелое число, то переставляя буквы в исходных словах мы получим слова двух длин. Если всего символов в исходных словах w, то у нас получится (w mod m) слов длины (w div m +1) и (m – w mod m) слов длины (w div m). Соответственно в слове, которое мы допишем должно быть (w div m +1) + (m – w mod m) букв.

Теперь, остаётся понять сколько ходов нам потребуется для перемещения букв и уравнивания длин всех слов. Мы можем беспрепятственно перемещать буквы из более длинных слов в более короткие, поскольку буквы в слове не повторяются, в длинном слове всегда есть буквы, которых нет в коротком.

Посчитаем для каждого слова, включая дописанное, сколько символов в слове не хватает либо сколько из него нужно переместить. Запишем результаты подсчётов в столбик, также как слова. Причём, недостаток символов будем записывать с знаком «-», а избыток с знаком «+». Например, для слов с длинами [2, 4, 2, 3, 4] получится [-1, 1, -1, 0, 1]. Для каждой линии, разделяющей слова, мы можем посчитать сколько букв должны пересечь для уравнивания длин слов. Чтобы посчитать это, достаточно найти сумму чисел справа от линии. Для нашего случая [-1 | 1, -1, 0, 1], сумма чисел справа от линии равна 1 + (-1) + 0 + 1, т.е. 1. Это значит, что из правой части в левую, нужно переместить одну букву, это один ход. Если бы сумма была -2, это значило-бы, что переместить надо две буквы, но из левой части в правую, это будет 2 хода. Для каждой линии нужно посчитать число ходов, а затем просуммировать их. Это и будет число ходов, необходимое для уравнивания длин слов.

```
# coding: utf-8
import math
import numpy as np

n = int(input())  # букв в алфавите
m = int(input())  # слов перед нами
nums = map(int, input().split())
```

```
words = np.array(list(nums))
# print(words)

if n < np.max(words):  # есть слово, длина которого больше числа букв в алфавите
    print(0)
    print(0)
    exit()

words_len, words_count = np.unique(words, return_counts=True)
i = 0
while i < len(words_len):  # слов конкретной длины больше, чем их может быть составленно из алфивита
    if math.factorial(n) / math.factorial(n - words_len[i]) / math.factorial(words_len[i]) < words_count[i]:
        print(0)
        print(0)
        exit()
    i += 1

# исходные данные верны!
# рассчитываем длину дополнительного слова
all_symbol = np.sum(words)
long_words = all_symbol % m  # столько слов получится длины new_len
short_words = m - long_words  # столько слов получится длины new_len - 1
if long_words == 0:
    new_len = all_symbol // m
    print(new_len)  # длина дополнительного слова
    # не дописываем новое слово к столбцу - оно не скажется на числе ходов, необходимых для перестановок
else:
    new_len = all_symbol // m + 1
    x = short_words + new_len
    print(x)  # длина дополнительного слова
    words = np.append(words, x)  # допишем новое слово в конец столбца

# рассчитываем число ходов
balance = np.zeros_like(words)
balance = words - new_len
# print(balance)
moves = 0
i = 1
while i < len(balance):
    left = np.sum(balance[:i])
```

```
    # print("left", balance[:i], left)
    # right = np.sum(balance[i:])
    # print("right", balance[i:], right)
    moves += abs(left)
    i += 1

print(moves)
```

Критерии оценивания:

0 - решение не прошло тестирование;

16 - решение прошло только тесты базовой группы;

32 - решение прошло тесты базовой и средней групп;

50 - решение прошло все тесты;