Федеральное государственное
автономное образовательное учреждение
высшего образования

«Национальный исследовательский
технологический университет «МИСиС»

Федеральное государственное
автономное образовательное учреждение
высшего образования

«Московский физико-технический
институт (государственный университет)»

Открытая олимпиада школьников по программированию «Когнитивные технологии»

2017/2018 учебного года

Задачи и решения

Москва, 2018

Оглавление

Введение

Отборочный этап Открытой олимпиады школьников по программированию «Когнитивные технологии» 2018 проводился в ноябре-декабре 2017 года. Заключительный этап проходил 14 января 2018 года на площадках НИТУ «МИСиС» и МФТИ. Оба тура проводились на системе Яндекс.Контест. Задачи отборочного и заключительного этапов были подготовлены авторской группой Центра Развития IT-образования МФТИ в составе Филиппа Дмитриевича Руховича, Евгения Александровича Белых, Степана Александровича Калинина, Артёма Микаэловича Комендантяна, Михаила Владимировича Макарова, Владислава Сергеевича Невструева, Александра Денисовича Проскурина и Андрея Владимировича Сергунина

Глава 1

Отборочный этап

1.1 Результаты

Первое место в отборочном этапе олимпиады занял участник, решивший 7 задач. По 6 задач решили 2 участника, занявшие соответственно второе и третье место. Остальные участники решили 5 и менее задач.

1.2 Задачи

Задача A. Конец света

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Много-много лет назад древние Майя предсказали, что вчера должен был произойти конец света. Свято веря в это предсказание, простой работник телеканала «Последний» по имени Петя решил, что в последний день своей жизни терять нечего, и выкрикнул прямо в лицо директору телеканала Стрэну Льву Константиновичу: «MTV!», оставив того в полном недоумении.

К счастью или к несчастью, конец света так и не наступил, а Петя теперь оказался в крайне неловком положении. Чтобы срочно загладить свою вину, Пете ничего не остаётся, кроме как немного приврать. Он решил сказать начальнику, что имел в виду вовсе не популярный телеканал, а что-нибудь другое. Например: «Мы тобой восхищаемся!» или «Мощное телевидение воз-

главляешь!». Более того, Пете не просто нужно придумать расшифровку аббревиатуры «MTV» – ему также нужно, чтобы она состояла из трёх подряд идущих слов из списка любимых слов Льва Константиновича.

Строго говоря, у Пети есть список из n любимых слов Льва Константиновича. Ему интересно узнать, сколько существует различных способов придумать расшифровку аббревиатуры «MTV», если слова в расшифровке должны принадлежать списку любимых слов и должны идти в нём подряд ровно в порядке: слово на букву «M», слово на букву «T», слово на букву «V». Более того, в списке любимых слов Льва Константиновича некоторые слова могут повторяться. Поэтому два способа расшифровки, состоящие из одинаковых слов, но стоящих на разных позициях в списке, считаются различными. Список слов может оказаться довольно длинным, так что Петя просит вас помочь ему решить эту задачу.

Формат входных данных

В первой строке дается единственное число n ($1 ::: n ::: 10^5$) - количество чисел в списке любимых слов Льва Константиновича.

В следующих n строках перечислены слова списка, по одному на строке.

Гарантируется, что сумма длин слов во входных данных не превосходит 10^5.

Формат выходных данных

Выведите единственное число - количество возможных расшифровок аббревиатуры «MTV».

Примеры

стандартный ввод	стандартный вывод
6 Mi Toboi Voshishaemsya Moshnoe Televidenie Vozglavlyaesh	2
10 Agniev Mi Toboi Voshishaemsya moshnoe televidenie vozglavlyaesh Mi toboi Voshishaemsya	1
6 Mi Toboi Voshishaemsya Mi Toboi Voshishaemsya	2

Разбор

Нам нужно было посчитать, сколько раз в заданном списке слов подряд стоят три слова, первое из которых начинается на букву «M», второе - на букву «T», а третье - на букву «V». Для этого было достаточно считать все слова как строки в массив, а затем пробежаться по нему и посчитать, сколько раз выполнено условие:

word[*i*][0] == $^{1}M^{1}$ && *word*[*i* + 1][0] == $^{1}T^{1}$ && *word*[*i* + 2][0] == $^{1}V^{1}$.

Также нужно было внимательно следить за тем, чтобы не вылезти за границы массива.

Задачу решило 147 участников, всего по задаче было сделано 432 попытки.

Авторское решение

```
#include <bits/stdc++.h>

using namespace std;

#define ld double
#define mp make_pair
#define ll long long

const ll inf = (ll) 1e9 + 7;
const ll mod = (ll) 1e9 + 7;
const ll maxn = 200001;

int n, ans;
vector < string > v;
char s[maxn];
string str;

int main()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", s);
        str = "";
        int m = strlen(s);
        for (int j = 0; j < m; j++) {
            str += s[j];
        }
        v.push_back(str);
        if (i > 1) {
            if (v[i - 2][0] == 'M' && v[i - 1][0] == 'T' && v[i][0] == 'V') {
                ans++;
            }
        }
    }
    cout << ans;
    return 0;
}
```

Задача B. Очень интересная новость

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

В некоторой социальной сети зарегистрировано n пользователей. Однажды Миша услышал очень интересную новость и решил выложить её на своей странице в этой социальной сети. Известно, что i-й пользователь просматривает страницы людей, на которых он подписан, в моменты времени t такие, что $t - i$ делится на n. Если человек впервые видит на чей-либо странице очень

интересную новость, он незамедлительно выкладывает её на своей странице. Для каждого пользователя найдите момент времени, когда он выложит очень интересную новость на своей странице.

Формат входных данных

В первой строке даны два целых числа n и m ($1 ::: n, m ::: 200000$) – количество пользователей и количество подписок. Миша имеет номер 1 и выкладывает новость в момент времени 1.

Каждая из следующих n строк содержит два целых числа u_i и v_i ($1 ::: u_i, v_i ::: n, u_i \neq v_i$), означающие, что пользователь u_i подписан на поль- зователя v_i.

Формат выходных данных

Выведите n целых чисел $d_1, \ldots, d_n$, где d_i – время, когда i-й пользователь выложит очень интересную новость, или -1 если этого не произойдет никогда.

Примеры

стандартный ввод	стандартный вывод
4 4 2 1 3 2 4 3 1 4	1 2 3 4
4 4 1 2 2 3 3 4 4 1	1 10 7 4
3 2 2 1 2 3	1 2 -1

Разбор

Построим ориентированный граф, вершинами которого будут пользователи. Если i-ый человек подписан на j-ого, то проведем из j в i ребро веса 1, если $i > j$, и ребро веса 0 иначе. Найдем кратчайшие расстояния от первого пользователя до остальных. Тогда ответ для i-го пользователя будет $d_i \cdot n + i - 1$, где d_i – посчитанное расстояние.

Асимптотика решения есть $O(n + m)$, если искать расстояние с помощью 0-1-bfs и $O(m \log n)$, если использовать алгоритм Дейкстры « с кучей ».

Задачу решили 47 участников, всего по задаче было сделано 397 попыток.

Авторское решение

```
#include <bits/stdc++.h>

#define all(a) (a).begin(), (a).end()
#define pb push_back
#define sz(a) (int)(a).size()

using namespace std;

typedef long long ll;
typedef pair < int, int >pii;
typedef pair < ll, ll > pll;
typedef long double ld;

const int INF = 1e9;

struct Edge {
    int to;
    int w;
};

vector < vector < Edge >> g;

vector < int > bfs(int v)
{
    int n = sz(g);
    vector < int >dist(n, INF);
    dist[v] = 0;
    deque < int >deq = { v };
    while (!deq.empty()) {
        int u = deq.front();
        deq.pop_front();
      for (Edge e:g[u]) {
            if (dist[e.to] > dist[u] + e.w) {
                dist[e.to] = dist[u] + e.w;
                if (e.w) {
                    deq.push_back(e.to);
                } else {
                    deq.push_front(e.to);
                }
            }
        }
    }
    return dist;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
```

```
    int n, m;
    cin >> n >> m;
    g.resize(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        --u;
        --v;
        g[v].push_back( { u, (u < v ? 1 : 0) } );
    }

    vector < int >dist = bfs(0);
    for (int i = 0; i < n; ++i) {
        if (dist[i] == INF) {
            cout << -1 << " ";
        } else {
            ll res = (ll) dist[i] * n + i + 1;
            cout << res << " ";
        }
    }
    cout << endl;
}
```

Задача C. Костюмы для актеров

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Актерская труппа одно очень известного театра готовится к премьере. Они уже давно придумали сценарий, распределили роли, каждый актер уже даже выучил свой текст! Осталось разобраться только с костюмами.

Выбор подходящих костюмов - вовсе не простая задача, как может показаться. Каждый актер порой вынужден переодеваться прямо во время спектакля! Конечно, это требует времени, так что иногда успеть вовремя выйти на сцену в новом костюме гораздо труднее, чем правильно сыграть свою роль.

К счастью, в премьерной постановке каждый костюм будет состоять из не более чем 30 занумерованных от 0 до 29 частей, каждая из которых покрашена либо в белый, либо в черной цвет. Актеры заметили, что чем сильнее отличается новый костюм от уже надетого, тем дольше придется переодеваться. Чтобы как-то оценить, какие костюмы труднее всего надевать, а какие легче всего, они решили сопоставить каждому костюму свое число по следующему принципу: пусть части костюма с номерами $y_1, y_2, \ldots, y_k$ имеют черный цвет, тогда данному костюму соответствует число $2^{y_1} + 2^{y_2} + \ldots + 2^{y_k}$.

Введя таким образом соответствие между числами и костюмами, актеры заметили, что время переодевания одного костюма в другой равно битовому

исключающему ИЛИ (т.е. xor-у) соответствующих им чисел. Например, если один костюм соответствует числу 6, а другой числу 5, то поменять один из них на другой актер может за $6 \oplus 5 = 3$ секунды (так как $6 = 2^2 + 2^1$, а $5 = 2^2 + 2^0$, то переодеть потребуется части с номерами 0 и 1, что займет $2^0 + 2^1 = 3$ секунды); в случае же костюмов с номерами 15 и 21 на переодевание потребуется $2^4 + 2^3 + 2^1 = 24$ секунд (ибо $15 = 2^3 + 2^2 + 2^1 + 2^0$, $21 = 2^4 + 2^2 + 2^0$).

Теперь им интересно узнать для каждого имеющегося костюма: если актер наденет этот костюм, то какое наименьшее и наибольшее время на переодевание он может потратить? Так как вся труппа очень занята репетициями, они попросили вас помочь им решить эту задачу.

Формат входных данных

В первой строке находится единственное целое число n ($2 ::: n ::: 2 * 10^5$) – количество костюмов, имеющихся в театре.

Во второй строке через пробел перечислены n чисел, соответствующие имеющимся костюмам: $x_1, x_2, \ldots, x_n$ ($0 ::: x_i < 2^{30}$).

Формат выходных данных

Выведите n строк, i-я из которых должна содержать два числа, записанных через пробел: минимальное и максимальное время, необходимое для переодевания i–го костюма.

Примеры

стандартный ввод	стандартный вывод
2 58 40	18 18 18 18
3 8 7 8	0 15 15 15 0 15

Разбор

Посмотрим на задачу с математической точки зрения. Нам дано множество чисел $a[]$; для i-го из них нужно среди остальных чисел выбрать такие два, чтобы xor с первым из них был минимально возможным, а со вторым – максимально возможным. Заметим также, что вывести нужно было не сами числа, а xor-ы с ними.

Для того чтобы быстро находить число с минимальным или максимальным xor-ом, воспользуемся структурой данных «битовый бор». Проще говоря, переведем все числа в двоичную систему счисления и представим их в виде строк из 0 и 1. Также нужно привести все числа к одной длине. Так как они не превосходят 2^{30}, то нужно привести их к длине не менее 31, добавив в начало

ведущие нули. Затем построим на этих строках бор. Кроме того, нам нужно в каждой вершине бора поддерживать размер ее поддерева.

Покажем сначала, как удалять число из такого бора. Для этого просто пройдемся по бору по пути, соответствующему данному числу, и уменьшим размер поддерева в каждой вершине этого пути на 1. Также будем теперь говорить, что в боре есть переход по 0 или 1, не просто если есть ребро в некоторую вершину, но если также размер поддерева этой вершины больше 0.

Также ясно, что добавляя число в бор, мы также будем изменять размер поддеревьев. Имеется в виду, что если добавить несколько одинаковых чисел, то мы корректно сохраним их количество, прибавляя по 1 на соответствующем им пути каждый раз при добавлении.

Теперь посмотрим, как с помощью бора находить число, xor с которым максимален для некоторого числа $a[i]$. Представим $a[i]$ в двоичной системе счисления как строку из 0 и 1 и добавим в нее ведущие нули, чтобы его длина стала равна 31. Сначала нужно удалить это число из бора, чтобы ни в каком из случаев мы не нашли xor с самим собой. Теперь посмотрим на старший бит числа. Если он 0, то для того чтобы xor был как можно больше, нам нужно взять число с 1 в этом бите. То есть, если из корня есть переход по 1 (т.е. есть переход и у вершины, достижимой по этому переходу, размер поддерева больше 0), то перейдем по нему. Если же перейти по 1 нельзя, то нам ничего не остается, кроме как перейти по 0. Иначе, если в старшем бите $a[i]$ стоит 1, мы делаем все то же самое, только пытаемся в первую очередь переходить по 0. Заметим, что по условию в множестве всегда есть хотя бы два числа, так что какой-то переход сделать мы точно сможем. Таким образом, мы сделаем первый переход по одному из битов, и перейдем в аналогичную ситуацию, но уже не в корне. Теперь мы будем рассматривать второй по старшинству бит и абсолютно аналогично решать, куда переходить дальше. Дойдя до листа бора, мы получим искомое число, xor с которым максимален. После этого нужно не забыть добавить обратно в бор удаленное число $a[i]$.

Заметим, что число, xor с которым минимален, ищется абсолютно аналогично, только теперь нужно пытаться переходить не по противоположному по значению биту, а по такому же. Разумеется, нужно также не забыть сначала удалить число из бора, а затем добавить.

Таким образом, мы научились находить для произвольного $a[i]$ числа из множества, xor с которыми для него минимален и максимален, так что осталось только вывести значения xor-ов с найденными числами.

Задачу решили 23 участника, всего по задаче было сделано 197 попыток.

Авторское решение

```
#include <bits/stdc++.h>
```

```
using namespace std;

#define ld double
#define mp make_pair
#define ll long long

const ll inf = (ll) 1e9 + 7;
const ll mod = (ll) 1e9 + 7;
const ll maxn = 200001;
const ld eps = 1e-12;
const int MAXLOG = 31;

struct trie {
    int go[2];
    int cnt;

     trie() {
        go[0] = go[1] = -1;
        cnt = 0;
}};

trie a[7000001];
int x[maxn], sz = 1, n;
vector < int >tv;

void add(int x)
{
    tv.clear();
    for (int i = 0; i < MAXLOG; i++) {
        tv.push_back(x & 1);
        x >>= 1;
    }
    reverse(tv.begin(), tv.end());
    int cur = 0;
    for (int i = 0; i < tv.size(); i++) {
        if (a[cur].go[tv[i]] == -1) {
            a[cur].go[tv[i]] = sz;
            a[sz].cnt++;
            cur = sz;
            sz++;
        } else {
            cur = a[cur].go[tv[i]];
            a[cur].cnt++;
        }
    }
}

void del(int x)
{
    tv.clear();
    for (int i = 0; i < MAXLOG; i++) {
        tv.push_back(x & 1);
```

```
        x >>= 1;
    }
    reverse(tv.begin(),  tv.end());
    int cur = 0;
    for (int i = 0; i < tv.size(); i++) {
        cur = a[cur].go[tv[i]];
        a[cur].cnt--;
    }
}

int findmi(int x)
{
    tv.clear();
    for (int i = 0; i < MAXLOG; i++) {
        tv.push_back(x & 1);
        x >>= 1;
    }
    reverse(tv.begin(), tv.end());
    int res = 0;
    int cur = 0;
    int pos = MAXLOG - 1;
    for (int i = 0; i < tv.size(); i++) {
        int nxt0 = a[cur].go[0];
        int nxt1 = a[cur].go[1];
        if (tv[i] == 0 && nxt0 != -1 && a[nxt0].cnt) {
            cur = nxt0;
        } else if (tv[i] == 1 && nxt1 != -1 && a[nxt1].cnt) {
            cur = nxt1;
        } else {
            res |= (1 << pos);
            if (nxt0 != -1 && a[nxt0].cnt) {
                cur = nxt0;
            } else {
                cur = nxt1;
            }
        }
        pos--;
    }
    return res;
}

int findma(int x)
{
    tv.clear();
    for (int i = 0; i < MAXLOG; i++) {
        tv.push_back(x & 1);
        x >>= 1;
    }
    reverse(tv.begin(), tv.end());
    int res = 0;
    int cur = 0;
    int pos = MAXLOG - 1;
```

```
    for (int i = 0; i < tv.size(); i++) {
        int nxt0 = a[cur].go[0];
        int nxt1 = a[cur].go[1];
        if (tv[i] == 1 && nxt0 != -1 && a[nxt0].cnt) {
            cur = nxt0;
            res |= (1 << pos);
        } else if (tv[i] == 0 && nxt1 != -1 && a[nxt1].cnt) {
            cur = nxt1;
            res |= (1 << pos);
        } else {
            if (nxt0 != -1) {
                cur = nxt0;
            } else {
                cur = nxt1;
            }
        }
        pos--;
    }
    return res;
}

int main()
{

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &x[i]);
        add(x[i]);
    }
    for (int i = 0; i < n; i++) {
        del(x[i]);
        if (x[i] == 1) {
            x[i] = x[i];
        }
        printf("%d %d\n", findmi(x[i]), findma(x[i]));
        add(x[i]);
    }
    return 0;
}
```

Задача D. Сосиска в массиве

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	64 мегабайта

Петя купил в мясном отделе два массива a и b длины n. Сосиской называется такая пара индексов l, r, $l ::: r$, что существует такие позиции u, ($l ::: u ::: r$) и d, ($l ::: d ::: r$), что подмассив a с l по u не убывает, подмассив a с u по r не возрастает, подмассив b с l по d не возрастает и подмассив b с d по r не

убывает. Число $r - l + 1$ называется размером сосиски.

Подмассивом массива m с l по r называется массив m^1 длины $r - l + 1$ такой, что $\forall i, 1 \vdots\vdots i \vdots\vdots r - l + 1 : m^1[i] = m[i + l - 1]$.

Пете стало интересно, каков максимальный размер сосиски в его массивах.

Формат входных данных

В первой строке входного файла содержится одно целое число $n, 1 \vdots\vdots n \vdots\vdots 10^5$ - длина Петиных массивов. Во второй строке записаны n целых чисел $a_i, |a_i| \vdots\vdots 10^9$ - массив a. В третей строке записаны n целых чисел $b_i, |b_i| \vdots\vdots 10^9$ - массив b.

Формат выходных данных

Выведите размер наибольшей сосиски в массивах Пети.

Примеры

стандартный ввод	стандартный вывод
4 2 5 1 3 1 4 3 2	2
16 1 3 4 5 5 4 3 1 1 3 4 5 5 4 3 1 -1 -3 -4 -5 -5 -4 -3 -1 -1 -3 -4 -5 -5 -4 -3 -1	9

Разбор

Назовём пару индексов l, r верхней полусосиской, если существует такая позиция u, $(l \vdots\vdots u \vdots\vdots r)$, что подмассив a с l по u не убывает, подмассив a с u по r не возрастает. Заметим, что если l, r – верхняя полусосиска, то l, i, $(l \vdots\vdots i \vdots\vdots r)$ – тоже верхняя полусосиска.

Определим $c[l] := max\{m \mid l, l + m - 1$ – верхняя полусосиска $\}$. Заметим, что если $a[l - 1] > a[l]$, то любая полусосиска с правым концом l будет невозрастающим подмассивом a. Найдём для каждой позиции i число $hdown[i]$ – максимальную длину невозрастающего подмассива a с началом в i, например с помощью обратной динамики по массиву a, используя следующую формулу:

$$hdown[i] = \begin{cases} hdown[i + 1] + 1, & \text{если } i < n \ \& \ a[i] \geq a[i + 1]; \\ 1, & \text{иначе.} \end{cases}$$

n чисел $hdown[i]$, таким образом, могут быть вычислены за время $O(n)$.

Аналогично можно найти и n чисел $hup[i]$ - максимально возможную длину неубывающего подмассива a с началом в i.

Теперь заметим, что чтобы набрать максимальной длины верхнюю полусосиску с началом в позиции i, нужно набрать максимально возможный неубывающий подмассив a с началом в i, а из его конца - максимальной длины невозрастающий подмассив a. Отсюда следует, что если $hup[i] = v$, то, то $c[i] = i + v - 1 + hdown[i + v - 1] - 1$.

Аналогичным образом можно определить нижнюю полусосиску в массиве b и насчитать массив $d[l] := max\{m / l, (l + m - 1) - $ нижняя полусосиска$\}$. Теперь длина сосиски нетрудно ищется как $max\{min(d[i], c[i]) / 1 ::: i ::: n\}$. Время работы такого решения есть $O(n)$.

Задачу решили 22 участника, всего по задаче было сделано 77 попыток.

Авторское решение

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

using namespace std;

void calc(vector < int >&a, vector < int >&ln)
{
    ln.assign(a.size(), 1);
    for (int i = ((int)a.size()) - 2; i >= 0; --i) {
        if (a[i] >= a[i + 1])
            ln[i] = ln[i + 1] + 1;
    }
    for (int i = ((int)a.size()) - 2; i >= 0; --i) {
        if (a[i] <= a[i + 1])
            ln[i] = max(ln[i], ln[i + 1] + 1);
    }
}

int main()
{
    iostream::sync_with_stdio(false);
    cin.tie(0);
    int n;
    cin >> n;
    vector < int >a(n), b(n); for
    (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    for (int i = 0; i < n; ++i) {
        cin >> b[i];
        b[i] = -b[i];
    }
    vector < int >ln1, ln2;
    calc(a, ln1);
    calc(b, ln2);
    int ans = 1;
    for (int i = 0; i < n; ++i) {
```

```
        ans = max(ans, min(ln1[i], ln2[i]));
    }
    cout << ans << endl;
    return 0;
}
```

Задача E. Петя, треугольники и контест

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Петя участвует в работе жюри одной Очень Известной Олимпиады по Информатике (ОИОИ). В круг обязанностей Пети входит подготовка задачи со следующим условием:

«Вам даны n различных точек на плоскости ($0 ::: n ::: 1000$). Выведите количество равнобедренных треугольников с вершинами в этих точках?».

Подготовка задачи включает в себя в том числе и подготовку тестов, и Петя обратился к Вам за помощью в этом деле. Сейчас Петя хочет сделать тест, ответ к которому будет ровно m, то есть будет существовать ровно m невырожденных равнобедренных треугольников с вершинами в этих точках. Ваша задача заключается в том, чтобы придумать такой тест.

Формат входных данных

В первой строке дано единственное число m ($0 ::: m ::: 10^5$) – требуемое количество равнобедренных треугольников.

Формат выходных данных

В первой строке выведите единственное целое неотрицательное число n, не превосходящее 1000 – количество точек в тесте.

В последующих n строках выведите координаты точек. На $i + 1$-й строке выведите через пробел 2 целых числа x_i и y_i – координаты i-й точки. Координаты не должны превосходить 10^4 по модулю, точки должны быть попарно различны.

Гарантируется, что ответ всегда существует.

Примеры

стандартный ввод	стандартный вывод
1	3 0 0 0 1 1 0
5	5 0 0 3 -4 3 4 5 0 6 3

Разбор

Так как вершина равнобедренного треугольника лежит на серединном перпендикуляре основания, возникает желание получить много пар точек с совпадающими серединными перпендикулярами. Для этого можно выбрать точки $(x, 0)$ и $(-x, 0)$, где x пробегает значения от 1 до 320. Также выберем точку $(0, 0)$ (зачем – будет понятно позже). Теперь каждая точка $(0, y), y > 0$ порождает 320 равнобедренных треугольников. Тогда, выбрав $\frac{m}{320}$ точек на оси Oy можно получить почти все нужные треугольники, а точнее – все, без $r = m \bmod 320$. Чтобы избежать появления треугольников, у которых основание не лежит на оси Ox, достаточно взять точки $(0, y)$ достаточно высоко, например, с $y \geq 1000$. Оставшиеся r треугольников можно получить, выбрав точку $(320 - r, 10000)$. Основания этих треугольников будут лежать на оси Ox и соединять точки вида $(320 - r - i, 0)$ и $(320 - r + i, 0)$, каковых ровно r, причём точка $(0, 0)$ также может понадобиться. Также несложно убедиться в том, что никаких равнобедренных треугольников с основанием не на оси Ox не возникнет.

Суммарное количество точек равно $641 + \left\lfloor \frac{m}{320} \right\rfloor + 1 \leq 641 + 312 + 1 < 1000$, что и требовалось.

Задачу решил 1 участник, всего по задаче было сделано 92 попытки.

Авторское решение

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

using namespace std;


int main()
{
```

```
    int m;
    cin >> m;
    vector < pair < int, int >>points;
    points.emplace_back(0, 0);
    for (int i = 0; i < 320; ++i) {
        points.emplace_back(i + 1, 0);
        points.emplace_back(-i - 1, 0);
    }
    for (int i = 0; i < m / 320; ++i) {
        points.emplace_back(0, 1000 + i);
    }
    if (m % 320 != 0) {
        points.emplace_back(320 - m % 320, 10000);
    }
    cout << points.size() << endl;
  for (auto p:points) {
        cout << p.first << ' ' << p.second << endl;
    }

    return 0;
}
```

Задача F. Сортировка слиянием

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Недавно Илья на лекции по программированию узнал про сортировку слиянием. Идея её проста - разбиваем массив на два примерно одинакового размера, каждый рекурсивно сортируем и затем сливаем при помощи двух указателей. Но заинтересовала его не реализация, а странная фраза преподавателя: "Сортировка работает, используя не более $O(n \cdot log(n))$ сравнений". Илья не совсем понял доказательство этого факта и поэтому решил проверить его на практике.

Он написал следующий код:

```
cnt := 0 //Количество сравнений во время сортировки
function letter (a, b):
    cnt := cnt + 1
    return a < b

array a[MAXN]       //Массив, который надо отсортировать
array buffer[MAXN] //Вспомогательный массив

procedure copy_to_buffer(l, r, ptr_buffer): //Копирует полуинтервал [l, r) из a в buffer
    while (l < r)
        buffer[ptr_buffer] := a[l]
        ptr_buffer := ptr_buffer + 1
```

```
        l := l + 1

procedure merge_sort(l, r): //В массиве a сортирует полуинтервал [l, r)
    n := r - l

    if (n <= 1) //Если массив имеет размер 1 или 0, то он уже отсортирован
        return

    m := l + n / 2 //Деление целочисленное, с округлением вниз
    merge_sort(l, m) //Сортируем левый и правый массивы
    merge_sort(m, r)

    ptr_left := l
    ptr_right := m
    ptr_buffer := l

    while (ptr_left < m and ptr_right < r) //Сливаем массивы при помощи двух указателей
        if (letter(a[ptr_left], a[ptr_right]))
            buffer[ptr_buffer] := a[ptr_left]
            ptr_left := ptr_left + 1
            ptr_buffer := ptr_buffer + 1
        else
            buffer[ptr_buffer] := a[ptr_right]
            ptr_right := ptr_right + 1
            ptr_buffer := ptr_buffer + 1

    copy_to_buffer(ptr_left, m, ptr_buffer) //Сливаем то, что осталось в левом
    copy_to_buffer(ptr_right, r, ptr_buffer) //Сливаем то, что осталось в правом
    //Хотя бы в одном массиве не осталось ничего, поэтому копирование работает корректно

    while (l < r) //Копируем ответ из буфера обратно в массив a
        a[l] := buffer[l]
        l := l + 1
```

Факт подтвердился, но теперь ему стало интересно следующее – при данных n и k, существует ли массив длины n, на котором его сортировка будет делать ровно k сравнений? Более того, если такой массив существует, то Илье интересно посмотреть на любой пример такого массива. Сам он не смог придумать, как решать эту задачу, поэтому просит вас помочь ему.

Формат входных данных

Вам даны два целых числа, разделённые пробелом – n и k ($1 \vdots n \vdots 10^5$; $0 \vdots k \vdots 10^9$)

Формат выходных данных

Если массива длины n, при сортировке которого merge_sort использует не более k сравнений, не существует, то выведите единственную строчку «No» (без кавычек).

Если же такой массив существует, то в первой строчке выведите слово «Yes» (без кавычек), а во второй строчке сам массив – n чисел, разделённых пробелами. Каждое число по модулю должно не превосходить 10^9. Гарантируется, что если подходящий массив существует, то существует и подходящий массив с данным ограничением.

Примеры

стандартный ввод	стандартный вывод
4 5	YES 1 3 2 4
4 0	NO

Разбор

Для начала для каждого $i \in 1 \ldots n$ определим минимальное и максимальное возможное количество сравнений, которое может получиться при сортировке i чисел. Для этого воспользуемся методом динамического программирования. Обозначим эти значения dp_min_i и dp_max_i соответственно. В силу того, что во время слияния двух половин большого массива хотя бы один из указателей ptr_left или ptr_right должен дойти до конца соответствующей половины, количество сравнений во время этого слияния не меньше размера меньшей половины, то есть хотя бы $\lfloor \frac{i}{2} \rfloor$. С другой стороны, количество сравнений не превосходит общего количества шагов, то есть оно меньше либо равно $\lceil \frac{i}{2} \rceil + \lfloor \frac{i}{2} \rfloor - 1 = i - 1$. Обе оценки достигаются: если все числа левой половины строго меньше всех чисел правой, то достигается оценка снизу; если же все числа левой половины, кроме самого большого строго меньше всех чисел правой, а самое большое число левой половины больше всех чисел правой, то достигается оценка сверху. Таким образом получается, что $dp\_min_i = dp\_min_{\lfloor \frac{i}{2} \rfloor} + dp\_min_{\lceil \frac{i}{2} \rceil}$ и $dp\_max_i = dp\_max_{\lfloor \frac{i}{2} \rfloor} + dp\_max_{\lceil \frac{i}{2} \rceil} + i - 1$, поскольку нам также необходимо отсортировать левую и правую половины.

Таким образом, если k не лежит в отрезке $dp\_min_n \ldots dp\_max_n$, то ответ «NO». Иначе существует пример.

Пример будет строить рекурсивная функция *build_example*, на вход которой будет подаваться строго возрастающий массив и число k, возвращать же она будет массив, который является перестановкой входного и на котором сортировка сделает ровно k сравнений.

Пусть размер данного на вход массив равен n. Если $n = 1$, то ничего делать не надо, так как k всегда лежит в допустимых пределах, то есть в данном случае оно равно 0. Иначе сортировка сделает не менее dp_min_n сравнений за счет сортировки левой и правой половины, а также их слияния. Осталось распределить $k_1 = k - dp\_min_n$ сравнений. Если $k_1 \leq dp\_max_{\lfloor \frac{i}{2} \rfloor} - dp\_min_{\lfloor \frac{i}{2} \rfloor}$, то вызовем рекурсивно функцию *build_example* от левой половины и числа $dp\_min_{\lfloor \frac{i}{2} \rfloor} + k_1$, а также от правой половины и числа $dp\_min_{\lceil \frac{i}{2} \rceil}$, и в итоге получим необходимый результат. Иначе останется $k_2 = k_1 - (dp\_max_{\lfloor \frac{i}{2} \rfloor} - dp\_min_{\lfloor \frac{i}{2} \rfloor})$ сравнений, которое мы аналогичным образом попытаемся удовлетворить при помощи сортировки правой половины. Если нам это не удастся, то останется $k_3 = k_2 - (dp\_max_{\lceil \frac{i}{2} \rceil} - dp\_min_{\lceil \frac{i}{2} \rceil})$ сравнений, которые необходимо выполнить во время слияния двух массивов. Если поменять местами самое большое число левой половины и k_3-е в порядке возрастания число правой, то во время слияния сначала ptr_left дойдет до конца левой половины, обеспечив $\lfloor \frac{i}{2} \rfloor - 1$ сравнение, затем он будет сравниваться с k_3 числами из второй половины, и в конце сравнится с одним числом, которое больше него. Итого получено необходимое число сравнений.

Обратите внимание, что после того, как мы поменяли местами самое большое число левой половины и k_3-е правой, нам надо поставить его в начало правой половины для поддержания упорядоченности массива, передаваемого функции *build_example*. Делается это за $O(n)$ путем последовательности транспозиций двух соседних элементов. Итоговая асимптотика составляет $O(n \cdot log(n))$, доказательство аналогично доказательству оценки времени работы сортировки слиянием.

Задачу решил 1 участник, всего по задаче была сделана 21 попытка.

Авторское решение

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

using  namespace  std;

vector  <  int  >  mn,  mx;

void  f(int  n)
{
    if  (n  <=  1)  {
        mn[n]  =  0;
        mx[n]  =  0;
        return;
    }
    if  (mn[n]  !=  0)  {
        return;
    }
    f(n  /  2);
    f(n  -  n  /  2);
    mn[n]  =  mn[n  /  2]  +  mn[n  -  n  /  2]  +  n  /  2;
    mx[n]  =  mx[n  /  2]  +  mx[n  -  n  /  2]  +  n  -  1;
}

void  gen(int  n,  int  k,  vector  <  int  >&a)
{
    if  (n  ==  1)  {
        return;
    }

    vector  <  int  >left,  right;
    for  (int  i  =  0;  i  <  n;  ++i)  {
        if  (i  <  n  /  2)  {
            left.push_back(a[i]);
        }  else  {
            right.push_back(a[i]);
        }
    }

    int  left_k  =  mn[n  /  2];
    int  right_k  =  mn[n  -  n  /  2];
    int  my_k  =  n  /  2;
    k  -=  left_k  +  right_k  +  my_k;

    int  d;

    d  =  min(k,  mx[n  /  2]  -  mn[n  /  2]);
    k  -=  d;
    left_k  +=  d;

    d  =  min(k,  mx[n  -  n  /  2]  -  mn[n  -  n  /  2]);
    k  -=  d;
    right_k  +=  d;
```

```
    d = min(k, n - 1 - n / 2);
    k -= d;
    my_k += d;

    if (my_k != n / 2) {
        int t = my_k - n / 2 - 1;
        swap(left.back(), right[t]);
        for (int i = t - 1; i >= 0; --i) {
            swap(right[i], right[i + 1]);
        }
    }
    gen(n / 2, left_k, left);
    gen(n - n / 2, right_k, right);
    for (int i = 0; i < n; ++i) {
        if (i < n / 2) {
            a[i] = left[i];
        } else {
            a[i] = right[i - n / 2];
        }
    }
}

int main()
{
    int n, k;
    cin >> n >> k;

    mn.resize(n + 1);
    mx.resize(n + 1);

    f(n);


    if (k < mn[n] || k > mx[n]) {
        cout << "NO" << endl;
        return 0;
    }

    vector < int >ans(n);
    for (int i = 0; i < n; ++i) {
        ans[i] = i + 1;
    }
    gen(n, k, ans);

    cout << "YES" << endl;
    for (int i = 0; i < n; ++i) {
        cout << ans[i] << " ";
    }
    cout << endl;

    return 0;
```

}

Задача G. Прогульщик Петя

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Петя учится в школе. Его класс занимается пять дней в неделю, однако каждый день у него шесть уроков. Пете кажется, что их у него слишком много, поэтому некоторые он решил прогулять. К сожалению, если по одному предмету он прогуляет больше одного урока, то преподаватели что-то заподозрят. Петру интересно, какое наибольшее количество уроков он сможет прогулять, не вызвав подозрений? Посчитайте это число для него.

Формат входных данных

Вам даны 30 строчек, разделённые на пять блоков по шесть строчек в каждой – петино расписание. В *i*-й строчке каждого блока записана строка, состоящая из строчных букв латинского алфавита – название предмета, который в данный день идёт *i*-м. Каждое название не превосходит по длине 15.

Формат выходных данных

В единственную строчку напишите ответ на задачу - максимальное количество уроков, которое сможет прогулять Петя.

Пример

стандартный ввод	стандартный вывод
algebra algebra geometry geometry biology topology philosophy algebra biology topology geography algebra literature chemistry philosophy algebra chemistry physics algebra topology geometry chemistry chemistry informatics informatics literature literature biology informatics algebra	10

Примечание

В тесте из условия Петя может прогулять десять уроков следующим образом: второй, третий и шестой уроки в понедельник; первый и пятый уроки во вторник; второй и шестой уроки в среду; последний урок в четверг, третий и четвёртый уроки в пятницу.

Разбор

Количество уроков, которые может прогулять Петя, совпадает с количеством предметов, которые у него есть. Это количество можно посчитать либо

с использованием std::set с асимптотикой $O(n \cdot log(n))$, либо каждый раз проверяя, что очередной предмет в расписании ранее не встречался. Асимптотика такого решения $O(n^2)$, что укладывается в ограничения.

Задачу решили 137 участников, всего по задаче было сделано 233 попытки.

Авторское решение

```
#include <iostream>
#include <set>
#include <string>

using namespace std;

set < string > st;

int main()
{
    int n = 30;
    string s;

    for (int i = 0; i < n; ++i) {
        cin >> s;
        st.insert(s);
    }

    cout << st.size() << endl;
}
```

Задача H. Александр учит стихи

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Александр считает себя поэтом. Более того - он даже пишет стихи! Недавно у него появилось желание выучить своё самое красивое стихотворение. Для этого он хочет использовать новую технику запоминания стихов. Суть её заключается в том, чтобы взять от каждой строчки некоторый, возможно, пустой префикс, затем склеить полученные префиксы в одну строку не меняя порядка и уже её учить. К сожалению, Александр может запомнить только строчку длины k. Также он является эстетом и поэтому хочет, чтобы полученная строка была лексикографически минимальна. Помогите ему в нахождении такой строки.

Префиксом длины l строки $S = S_0S_1 \ldots S_{n-1}$ называется строка $S_0S_1 \ldots S_{l-1}$, либо пустая строка, если $l = 0$.

Строка S лексикографически меньше строки S^1, если существует l, такое что префиксы длины l у обеих строк совпадают, а $S_l < S^1_l$, либо если S

является префиксом строки S^1, но не совпадает с S^1.

Формат входных данных

В первой строке входных данных находятся два целых числа – n и k ($1 ::: n ::: 5 \cdot 10^3; 1 ::: k ::: 5 \cdot 10^3$) – количество строк в произведении Александра и длина искомой строки соответственно. В следующих n строках содержатся строчки из стихотворения Александра. Каждая строчка суть слово из строчных букв латинского алфавита. Гарантируется, что суммарная длина всех слов не превосходит $5 \cdot 10^3$, и что ответ существует.

Формат выходных данных

Выведите единственную строку - ответ на задачу.

Примеры

стандартный ввод	стандартный вывод
3 7 aaaaa bbbbb aaaaa	aaaaaaa
3 4 ab abb bba	aabb
5 5 are you can do it	acadi

Примечание

В первом тестовом примере можно взять первые 4 буквы из первой строки и первые 3 из третьей.

Во втором тесте можно взять по первой букве из первой и второй строки, а из третьей взять первые два символа.

В третьем тесте следует взять первый символ из первой строки, первые два символа из третьей и по одному первому символу из двух последний строк.

Разбор

Будем посимвольно искать строку-ответ. Для этого будем поддерживать двумерный массив dp, где $dp_{i,j} = 1$ тогда и только тогда, когда из префиксов

строк с номерами, не превосходящими i, можно составить уже найденный префикс ответа, причем длина префикса, взятого из строки с номером i, должна быть равна j. Также необходимо, чтобы оставшихся символов хватало для дополнения построенной строки до нужной длины. Во всех остальных случаях в $dp_{i,j}$ будет храниться 0. Иными словами, $dp_{i,j}$ показывает, может ли построенная строка заканчиваться перед j-м символом i-й строки.

Легко видеть, что найденную строку можно продолжить j-м символом i-й строки тогда и только тогда, когда $dp_{i,j} = 1$. Значит, на каждом шаге из всех допустимых символов мы будем выбирать минимальный и им продолжать уже найденный ответ. dp пересчитывается следующим образом:

- При $j > 0$ $dp_{i,j} = 1$, если $dp_{i,j-1} = 1$ и j-й символ i-й строки равен найденному.
- При $j = 0$ $dp_{i,j} = 1$, если символов в строках, начиная с i-й хватает на то, чтобы дописать уже построенный префикс до нужной длины, а также если в предыдущей строке существует символ, такой что он равен найденному и соответствующее ему значение dp равно 1, либо если $dp_{i-1,j}$ существует и равно 1.

На каждом шаге мы просматриваем все символы во всех строках 2 раза – когда ищем оптимальный символ и когда пересчитываем dp. Пусть S – суммарная длина всех строк. Таким образом, на поиск очередного символа мы тратим $O(S)$ действий, и итоговая асимптотика решения составляет $O(S \cdot k)$.

Задачу решили 3 участника, всего по задаче было сделано 76 попыток.

Авторское решение

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

using namespace std;

vector < string > s;
vector < int >sum;
vector < vector < bool > >good[2];
vector < bool > good_empty_prefix[2];
vector < pair < int, int >>member[2];

bool previous_good(int i, int j, int id)
{
    if (id > 0) {
        return good[i][j][id - 1];
    } else {
        return  good_empty_prefix[i][j];
    }
```

```
}

int main()
{
    int n, k;
    cin >> n >> k;

    s.resize(n);
    sum.resize(n);
    good[0].resize(n);
    good[1].resize(n);
    good_empty_prefix[0].resize(n);
    good_empty_prefix[1].resize(n);

    for (int i = 0; i < n; ++i) {
        cin >> s[i];
        good[0][i].resize(s[i].size());
        good[1][i].resize(s[i].size());
    }
    sum.back() = s.back().size();
    for (int i = n - 2; i >= 0; --i) {
        sum[i] = sum[i + 1] + s[i].size();
    }

    for (int i = 0; i < n; ++i) {
        if (sum[i] >= k) {
            member[0].push_back( {
                                    i, -1}
            );
        } else {
            break;
        }
    }
    string ans = "";

    for (int id = 1; id <= k; ++id) {
        char min_ch = 'z';
        int id1 = (id + 1) % 2;
        for (int j = 0; j < member[id1].size(); ++j) {
            if (member[id1][j].second + 1 < s[member[id1][j].first].size()) {
                min_ch =
                    min(min_ch,
                        s[member[id1][j].first][member[id1][j].second +
                                                1]);
            }
        }
        ans += min_ch;
        int first1 = n + 1;
        member[id % 2].clear();
        for (int j = 0; j < member[id1].size(); ++j) {
            if (member[id1][j].second + 1 < s[member[id1][j].first].size()
                && min_ch ==
```

```
                s[member[id1][j].first][member[id1][j].second + 1]) {
                member[id % 2].push_back( {
                                        member[id1][j].first,
                                        member[id1][j].second + 1}
                );
                first1 = min(first1, member[id1][j].first);
            }
        }
        for (int i = first1 + 1; i < n; ++i) {
            if (sum[i] + id >= k) {
                member[id % 2].push_back( {
                                        i, -1}
                );
            } else {
                break;
            }
        }
    }
    cout << ans << endl;
    return 0;
}
```

Глава 2

Заключительный этап

2.1 Результаты

Первое место в заключительном этапе олимпиады занял участник, решивший 6 задач. На втором месте участник, решивший 5 задач, на третьем участник, также решивший 5 задач, но с худшим штрафным временем. Эти три участника объявлены победителями олимпиады. Призёрами олимпиады стали участники, решившие не менее 3 задач.

2.2 Задачи

Задача A. Шифровка клавиатурой

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Владислав очень любит смотреть фильмы. Наиболее понравившиеся ему фильмы он записывает в список. Чтобы никто, кроме него этот список не увидел, он защитил его паролем. На самом деле пароль Владислава есть название его любимого фильма, в котором каждую букву он заменил на некоторую другую. Для того, чтобы определить, по какому правилу будут меняться буквы, Владислав использует клавиатуру. Необходимая нам часть клавиатуры выглядит так:

qwertyuiop

asdfghjkl
zxcvbnm

Также для шифровки он выбирает некоторое целое неотрицательное число k. Затем, когда Владиславу требуется определить, на какую букву требуется заменить текущую, он находит ее на клавиатуре и отходит от нее на k шагов вправо, причем, если на очередном шаге справа нет клавиши, он переходит в начало строки. Например, при $k = 3$ символ «q» заменится на символ «r», а символ «n» на символ «x».

Александру известно название любимого фильма Владислава, а также число k, которое Владислав использует при вставке. Помогите Александру подобрать пароль от списка Владислава.

Формат входных данных

В первой строке записано название любимого фильма Владислава. Название — это непустая строка, состоящая только из строчных букв латинского алфавита. Длина строки не превосходит 100.

Во второй строке записано единственное целое число k ($0 ::: k ::: 200$) — число, которое Владислав использует для шифровки.

Формат выходных данных

В единственной строке выведите пароль Владислава.

Примеры

стандартный ввод	стандартный вывод
vforvendetta 3	mjwumyxhyiif
cloudatlas 2	bsqogdusdf
thegodfather 9	rhwgidfarhwe

Разбор

Заведем три строки:

- s_1 = qwertyuiop
- s_2 = asdfghjkl
- s_3 = zxcvbnm

Пусть нам требуется определить, на какой символ заменится символ c. Для этого найдем его позицию в наших строках. Пусть он находится в строке s_i

на j-й позиции. Несложно убедиться, что искомый символ находится в строке s_i на позиции с номером $(j + k) \mod |s_i|$, где $|s_i|$ – длина строки s_i.

Для решения задачи достаточно пройтись по исходной строке и для каждого символа определить, на какой символ он заменится. Итоговая асимптотика решения составляет $O(\alpha \cdot |s|)$, где $\alpha = |s_1| + |s_2| + |s_3|$ – размер алфавита, а $|s|$ – длина исходной строки.

Задачу решили 73 участника, всего по задаче было сделано 130 попыток.

Авторское решение

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

using namespace std;

vector < string > vec = {
"qwertyuiop", "asdfghjkl", "zxcvbnm"};

int main()
{
    string s;
    int k;
    cin >> s >> k;
    for (int i = 0; i < s.size(); ++i) {
        for (int j = 0; j < vec.size(); ++j) {
            int t;
            for (t = 0; t < vec[j].size(); ++t) {
                if (vec[j][t] == s[i]) {
                    break;
                }
            }
            if (t < vec[j].size()) {
                cout << vec[j][(t + k) % vec[j].size()];
            }
        }
    }
    return 0;
}
```

Задача B. Хитрая функция

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Даны целые числа a, b, p; p – простое, и функция $f(x, y)$. Эта функция устроена следующим образом. Сначала вычисляется число $v = ab + xy$. Если v делится на p, то $f(x, y)$ равно 1; иначе вычисляется число $u = ax + by$, а

итоговое значение $f(x,y)$ будет равным остатку от деления числа $u \cdot inv_p(v)$ на p.

Здесь запись $inv_p(a)$ означает взятие обратного по модулю p, то есть такого целого числа $b \in [0,\ldots,p-1]$, что $ab-1$ делится на p. Науке известно, что если a не делится на p, то число $a^{p-1}-1$ делится на p. Этот факт носит название "малая теорема Ферма". Из него, в частности, следует, что в качестве $inv_p(a)$ можно взять остаток от деления числа a^{p-2} на p.

Вам также дано число k. Требуется найти значение выражения

$$f(f(\ldots f(f(k,k-1),k-2),\ldots,2),1)$$

Формат входных данных

В первой строке даны четыре целых числа a,b,p,k ($2 ::: p ::: 10^9+7$, $1 ::: a,b ::: min(p-1,10^5)$, $2 ::: k ::: 10^{18}$, p – простое) – параметры функции и количество итераций.

Формат выходных данных

Выведите требуемое значение выражения.

Примеры

стандартный ввод	стандартный вывод
2 4 5 3	4
1 1 2 10	1

Разбор

Рассмотрим число $f(x,a)$ при некотором x. Есть два случая. Если $v = ab + ax$ делится на p, то $f(x,a) = 1$. Если же v не делится на p, то $f(x,a) = (ax+ab)\cdot inv_p(ab+ax) = 1$. Поэтому при $k > a$ выполнено равенство $f(f(\ldots f(f(k,k-1),k-2),\ldots,2),1) = f(f(\ldots f(f(1,a-1),a-2),\ldots,2),1)$. Функцию $f(x,y)$ можно считать за $\log p$ используя малую теорему Ферма. Асимптотика $O(min(a,k)\log p)$.

Задачу решил 1 участник, всего по задаче было сделано 33 попытки.

Авторское решение

```
#include <iostream>
#include <stdio.h>
#include <cstring>
#include <vector>
#include <algorithm>
#include <cstdlib>
#include <cmath>
#include <queue>
#include <stack>
#include <map>
```

```
#include <set>
#include <ctime>
#include <cassert>
#include <unordered_map>
#include <fstream>
#include <random>
#include <cstring>
#include <bitset>
#include <functional>

#define all(a) (a).begin(), (a).end()
#define pb push_back
#define sz(a) (int)(a).size()

using namespace std;

typedef long long ll;
typedef pair < int, int >pii;
typedef pair < ll, ll > pll;
typedef long double ld;

int p;

void add(int &a, int b)
{
    a += b;
    if (a >= p) {
        a -= p;
    }
}

void mul(int &a, int b)
{
    ll c = (ll) a * b;
    c %= p;
    a = c;
}

int binPow(int a, int b)
{
    if (b == 0) {
        return 1;
    }
    int res = binPow(a, b / 2);
    mul(res, res);
    if (b & 1) {
        mul(res, a);
    }
    return res;
}

int f(int a, int b, int x, int y)
```

```
{
    int tmp, u = 0, v = 0;

    tmp = a;
    mul(tmp, x);
    add(u, tmp);
    tmp = b;
    mul(tmp, y);
    add(u, tmp);

    tmp = a;
    mul(tmp, b);
    add(v, tmp);
    tmp = x;
    mul(tmp, y);
    add(v, tmp);

    if (v == 0) {
        return 1;
    }
    int res = u;
    mul(res, binPow(v, p - 2));
    return res;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    int a, b;
    ll k;
    cin >> a >> b >> p >> k;

    int cur = 0;

    if (a == 0) {
        cur = k % p;
        k = min(k, 2 + k % 2);
    } else {
        k = min((ll) a + 1, k);
        cur = k;
    }

    for (int i = k - 1; i >= 1; --i) {
        cur = f(a, b, cur, i);
    }

    cout << cur << endl;
    return 0;
}
```

Задача C. Список фильмов

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Владислав очень любит смотреть фильмы. Наиболее понравившиеся ему фильмы он записывает в список. Чтобы как-то упорядочить фильмы в списке, Владислав для каждого фильма определяет два параметра – игру актеров и качество съемки. Оба параметра он оценивает целыми числами от 1 до 10^9. В тот момент, когда Владислав просмотрел очередной фильм, он добавляет его в список по следующему алгоритму:

1. На первом шаге Владислав выбирает параметр - игру актеров или качество съемки.
2. Затем он находит в своем текущем списке первый фильм, у которого выбранный параметр больше либо равен соответствующему параметру нового фильма.
3. Наконец, Владислав вставляет новый фильм перед найденным либо в конец списка, если искомого фильма не существует.

Александру удалось получить доступ к этому списку. Теперь ему интересно – в каком порядке Владислав смотрел фильмы?

Формат входных данных

В первой строке входных данных вам дано единственное целое число n ($1 ::: n ::: 10^5$) – количество фильмов в списке Владислава. Далее в n строках записаны по два целых числа a_i и b_i ($1 ::: a_i, b_i ::: 10^9$) – оценка обоих параметров у фильма, который в списке Владислава имеет номер i.

Формат выходных данных

Если Владислав ошибся и не существует порядка просмотра фильмов, после которого они будут идти в том же порядке, что и во входных данных, в единственной строке выведите «NO» (без кавычек). Иначе в первой строке выведите «YES» (без кавычек), а в следующих n строках выведите по два числа p_i и t_i, где p_i – номер очередного фильма, а t_i – параметр, по которому этот тип следует вставлять в список. Если существует несколько правильных ответов, выведите любой.

Примеры

стандартный ввод	стандартный вывод
4 5 6 6 4 7 7 3 8	YES 2 1 4 2 1 1 3 2
7 2 3 7 6 6 9 4 10 5 13 8 1 3 7	YES 1 1 6 1 7 2 4 1 5 1 2 2 3 2

Разбор

Ключевым для решения задачи является следующее утверждение о том, что примера не существует тогда и только тогда, когда существует такое $i < n$, что $a_i > a_{i+1}$ и $b_i > b_{i+1}$.

Докажем такое утверждение. Пусть такое i существует. Пусть фильм с номером i был просмотрен раньше, чем фильм с номером $i + 1$. В момент вставки фильма с номером $i + 1$ на втором шаге алгоритма будет найден либо фильм с номером i, либо некоторый фильм с меньшим номером. Таким образом, фильм с номером $i + 1$ будет стоять левее фильма с номером i. Если же порядок просмотра этих фильмов был другим, то в момент вставки фильма с номером i на втором шаге алгоритм не остановится на фильме с номером $i + 1$, и фильм с номером i будет вставлен либо после фильма с номером $i + 1$, либо до него, однако в таком случае они не будут соседними.

Если такого i не существует, то существует пример. Будем смотреть фильмы в порядке убывания номеров. Фильм с номером n вставим по любому параметру. i-й фильм будем вставлять по тому параметру, который меньше соответствующего параметра фильма с номером $i + 1$. Такой параметр найдется в силу утверждения.

Таким образом, мы не только доказали утверждение, но и в случае существования решения, нашли алгоритм его построения. Итоговая асимптотика решения составляет $O(n)$.

Задачу решили 24 участника, всего по задаче было сделано 114 попыток.

Авторское решение

```
#include <iostream>
```

```
#include <vector>
#include <cstdio>

using namespace std;

vector < int >a, b;

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    int n;
    cin >> n;

    a.resize(n);
    b.resize(n);

    for (int i = 0; i < n; ++i) {
        cin >> a[i] >> b[i];
    }

    bool fl = 1;
    for (int i = 0; i + 1 < n; ++i) {
        if (a[i] > a[i + 1] && b[i] > b[i + 1]) {
            fl = 0;
            break;
        }
    }

    if (!fl) {
        cout << "NO" << endl;
        return 0;
    }
    cout << "YES" << endl;
    for (int i = n - 1; i >= 0; --i) {
        cout << i + 1 << " ";
        if (i == n - 1 || a[i] <= a[i + 1]) {
            cout << "1\n";
        } else {
            cout << "2\n";
        }
    }
    return 0;
}
```

Задача D. Плюсы и минусы последовательностей

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Как-то раз, от безделья, Влад решил написать всевозможные последовательности, в которых каждое из чисел от 1 до n встречается ровно по одному разу. Влад любит беспорядок, поэтому он вычеркнул все последовательности, в которых существует число k, которое стоит на k-м месте. Но на этом он не остановился. Рядом с каждой из оставшихся последовательностей он написал строку из плюсов и минусов длины n, где на k-м месте стоит '+', если число k в последовательности стоит правее k-й позиции, и '-', если левее.

Теперь Влад интересуется, сколько раз он выписал свою любимую строку s длины n, состоящую из плюсов и минусов. Помогите ему с этой задачей. Так как искомое число может быть очень большим, выведите его остаток по модулю $10^9 + 7$.

Формат входных данных

Единственная строка содержит строку s ($1 ::: /s/ ::: 5000$) – любимую строку Влада, состоящую из плюсов и минусов.

Формат выходных данных

Выведите одно число: количество раз, которые была выписана строка s. Так как ответ может быть очень большим, выведите не само число, а остаток от деления этого числа на $10^9 + 7$.

Примеры

стандартный ввод	стандартный вывод
+-+-	2
+-+-+	0
+-+-+—-	200

Разбор

Будем насчитывать величину $dp[k][e]$ – количество способов для каждого числа от 1 до k определить, будет ли оно стоять в перестановке на позициях $[1, k]$ (т.е. среди чисел $p[1]$, $p[2]$, . . ., $p[k]$) или нет, а если будет, то определить место, при этом ровно для e из этих чисел определено, что они не попадают в отрезок $[1, k]$. Заметим, что число e также равно числу незанятых числами $1..k$ позиций отрезка $[1, k]$.

Тогда если $s[k] =^{1} -^{1}$, то $dp[k][e] = dp[k-1][e] \cdot e + dp[k-1][e+1] \cdot (e+1)^2$. А именно, если на позициях $[1, k]$ не лежит e чисел множества $1..k$, то либо e,

либо $e + 1$ чисел из множества $1..k - 1$ не стоят на позициях $[1..k - 1]$. Если таких чисел e, т.е. место k не занято числами $k-1$, то число k можно поставить e способами. А если таких чисел $e + 1$, то k можно поставить $e + 1$ способом и еще на место k можно поставить одно из $e + 1$ чисел.

Аналогично выводится, что если $s[k] = {}^{1} + {}^{1}$, то $dp[k][e] = dp[k - 1][e - 1] + dp[k - 1][e] \cdot e$.

Ответ будет находиться в $dp[n][0]$. Асимптотика решения $O(n^2)$.

По задаче было сделано 2 попытки, ни одна из которых не была удачной.

Авторское решение

```
#include <bits/stdc++.h>

#define all(a) (a).begin(), (a).end()
#define sz(a) (int)(a).size()

using namespace std;

typedef long long ll;
typedef pair < int, int >pii;
typedef pair < ll, ll > pll;
typedef long double ld;

const int P = 1e9 + 7;

void add(int &a, int b)
{
    a += b;
    if (a >= P) {
        a -= P;
    }
}

void mul(int &a, int b)
{
    ll c = (ll) a * b;
    c %= P;
    a = c;
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    string s;
    cin >> s;
    int n = sz(s);

    if (s[0] == '-') {
```

```
        cout << 0 << endl;
        return 0;
    }

    vector < vector < int >>dp(n, vector < int >(n));

    dp[0][1] = 1;

    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j <= i + 1; ++j) {
            if (s[i + 1] == '-') {
                if (j > 0) {
                    int tmp = dp[i][j];
                    mul(tmp, j);
                    add(dp[i + 1][j], tmp);
                    mul(tmp, j);
                    add(dp[i + 1][j - 1], tmp);
                }
            } else {
                int tmp = dp[i][j];
                add(dp[i + 1][j + 1], tmp);
                mul(tmp, j);
                add(dp[i + 1][j], tmp);
            }
        }
    }

    cout << dp[n - 1][0] << endl;

    return 0;
}
```

Задача E. Подземелья, подземелья и ещё больше подземелий

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Диппер и дядя Форд играют в "Подземелья, подземелья и ещё больше подземелий". До победы им осталось совсем немного, всего лишь победить Вероятника Мерзейшего.

Условия схватки в следующем: Вероятник дал Дипперу и Форду многочлен с n целыми неотрицательными коэффициентами ($f(x) = a_n * x^n + a_{n-1} * x^{n-1} + ... + a_1 \cdot x + a_0$) (коэффициенты **могут** быть нулевыми, в т.ч. и a_n) и некоторую константу T. Дальше Вероятник называет q неотрицательных целых чисел x. Чтобы победить Вероятника, Диппер и Форд должны научиться быстро отвечать на вопрос: верно ли, что значение многочлена в точке x меньше либо равно T?

Формат входных данных

В первой строке через пробел заданы два числа: $n(0 ::: n ::: 10^5)$ и $T(0 ::: T ::: 10^9)$.

В следующей строке задано $n + 1$ число – коэффициенты многочлена, начиная с a_0 и заканчивая a_n $(0 ::: a_i ::: 10^9)$.

В следующей строке задано $q(1 ::: q ::: 10^5)$ – количество запросов. В следующих q строках заданы сами запросы – числа x_i $(0 ::: x_i ::: 10^9)$

Формат выходных данных

Выведите q строк – ответы на запросы Вероятника: "YES"(без кавычек) в случае, если $f(x_i) ::: T$, и "NO"(без кавычек) иначе.

Примеры

стандартный ввод	стандартный вывод
1 10 1 2 6 0 1 2 3 4 5	YES YES YES YES YES NO
2 10 1 0 1 4 5 3 1 7	NO YES YES NO

Разбор

Заметим, что для любого $x \rangle 0$ выполнено: $f(x) ::: f(x + 1)$, а если степень многочлена больше нуля, то $f(10^9 + 1) > 10^9$. Найдем наибольшее y из отрезка $[0, 10^9 + 1]$ такое, что $f(y) ::: T$ с помощью бинарного поиска (при вычислении $f(y)$ возможно переполнение типа, поэтому как только одно из слагаемых в процессе вычисления оказалось больше T, то вычисление нужно завершить и сообщить, что $f(y) > T$). Далее в каждом запросе достаточно сравнить данное число с y. Итоговая асимптотика решения $O(n * \log_2(n) + q)$.

Задачу решили 39 участников, всего по ней было сделано 674 попытки.

Авторское решение

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = (int) 1e5 + 10;
int v[maxn];

const int inf = (int) 1e9 + 10;

int calc(int n, int x)
{
    ll ans = 0;

    for (int i = n; i >= 0; i--) {
        ans *= x;
        ans += v[i];
        ans = min(ans, (ll) inf);
    }

    return (int) ans;
}

int main()
{
    int n;

    int t;

    cin >> n >> t;

    for (int i = 0; i <= n; i++) {
        scanf("%d", &v[i]);
    }

    int l = -1;
    int r = 1e9 + 10;

    while (l != r) {
        ll m = (l + r + 1) >> 1;

        if (calc(n, m) <= t) {
            l = m;
        } else {
            r = m - 1;
        }
    }

    int q;

    cin >> q;

    while (q--) {
```

```
        int x;

        scanf("%d", &x);

        if (x <= 1) {
            printf("YES\n");
        } else {
            printf("NO\n");
        }
    }

    return 0;
}
```

Задача F. Подарки

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	1 секунда
Ограничение по памяти:	256 мегабайт

Дипперу и Мейбл подарили много сладких подарков на Рождество, и им нужно разделить их между собой. Диппер решил, что просто поделить подарки пополам слишком скучно, и предложил Мейбл сыграть в следующую игру.

Все подарки разложили на ковёр, состоящий из $2*n+1$(занумерованных от 1 до $2*n+1$) расположенных в ряд кусков ткани. На каждом куске либо лежит ровно один подарок, либо ничего не лежит. Также Диппер дал Мейбл листок, на котором написана строка $w = w_0w_1...w_{n-1}$, кодирующая n последовательных действий, каждое из которых принимает одно из следующих значений:

- R - "сдвинуться вправо";
- L - "сдвинуться влево";
- T - "взять подарок, если он был изначально на текущем куске ткани и еще не забирался ранее".

Мейбл решила, что Диппер выбрал выгодную для него последовательность, и поэтому Диппер разрешил Мейбл взять любой циклический сдвиг действий с этого листка(напомним, что циклическим сдвигом строки $s = s_0s_1...s_{n-1}$ на число шагов k от 0 до $n - 1$ называется строка $s_ks_{k+1}...s_{n-1}s_0s_1...s_{k-1}$), посадить свою свинью Пухлю в центр ковра (на кусок ткани с номером $n + 1$) и заставить её выполнять этот циклический сдвиг действий с листка. Все подарки, которые соберёт Пухля, достаются Мейбл. Мейбл хочет получить как можно больше подарков и попросила вас помочь ей.

Формат входных данных

В первой строке дано единственное число $n(1 ::: n ::: 3 * 10^4)$.

В следующей строке дана строка s длины $2 * n + 1$ из 0 и 1, где $s_i = 1$, когда на i-ом куске ткани есть подарок, и 0 иначе.

В последней строке дана строка действий w длины n.

Формат выходных данных

Выведите единственное число: максимальное количество подарков, которые может получить Мейбл, выбрав произвольный циклический сдвиг строки действий w, если Пухля начнет на куске ткани с номером $n + 1$.

Примеры

стандартный ввод	стандартный вывод
3 0011000 TTL	2
4 000010000 TLRT	1

Примечание

В первом примере, можно получить циклический сдвиг TLT, который даст оптимальный ответ 2.

Во втором примере, есть 4 циклических сдвига строки:

$TLRT$: забираем подарок с 5 куска ткани, двигаемся влево, двигаемся обратно вправо, пытаемся забрать подарок(но его уже нет).

$LRTT$: Двигаемся влево, двигаемся обратно вправо, забираем подарок с 5 куска ткани, пытаемся забрать подарок(но его уже нет).

$RTTL$: Двигаемся вправо, пытаемся забрать подарок(но его там изначально не было), пытаемся забрать подарок(но его там изначально не было), двигаемся влево.

$TTLR$: Забираем подарок с 5 куска ткани, пытаемся забрать подарок(но его уже нет), двигаемся влево, двигаемся обратно вправо.

Итого ответ 1.

Разбор

Первое что мы научимся делать, это поддерживать позиции, с которых мы собираем подарки для очередного циклического сдвига строки и пересчитывать их через предыдущий циклический сдвиг. Для исходной строки просто найдем эти позиции, промоделировав алгоритм. Дальше нам надо уметь перемещать одну букву из начала в конец:

- если это буква L, то нетрудно заметить, что все места с которых мы собирали подарки сдвинуться на один вправо, так как мы изначально убрали один сдвиг влево, который был перед собиранием всех подарков.

- Аналогично, если это буква R, то все места, с которых мы собирали подарки сдвинутся на один влево.

- Осталось разобраться с буквой T; посмотрим, что может произойти при перенесении её с первой позиции строки на последнюю. Раньше мы гарантированно собирали подарок с позиции $n + 1$, теперь неизвестно, совершается ли это действие. Аналогично, пусть суммарно за все действия мы сдвигаемся на d от изначальной позиции, тогда раньше в позиции $n + 1 + d$ мы могли взять подарок, а могли и не взять (ибо этот подарок уже взят раньше); теперь же этот подарок гарантированно берем. Это все изменения, которые могли произойти. Тогда в позиции $n + 1 + d$ мы просто проставляем, что теперь оттуда берётся подарок, а для изначальной позиции надо понять, сколько раз мы пытались поднять с неё подарок. Если один, то надо сказать, что теперь с неё не собирается подарок, иначе – всё еще собирается.

Чтобы поддерживать для каждой позиции количество поднятий с неё подарка, можно воспользоваться, например, декартовым деревом (оно умеет поддерживать операции добавления, удаления, вычитания или прибавления ко всем элементам единицы, а также нахождение количества вхождения определенного элемента). А сами же позиции, с которых подбираются подарки, удобно хранить с помощью, например, структуры данных std::bitset в языке C++ или ее аналогов (в т.ч. самописных) в других языках программирования. Она (структура) умеет сдвигать все элементы влево или вправо на 1 и изменять произвольный элемент, и делает это за $O(len/32)$, где len есть длина нашего bitset. Тогда мы умеем для каждого циклического сдвига пересчитывать позиции, с которых собираются подарки, за время $O(n/32 + \log_2(n))$. Чтобы для очередного циклического сдвига найти количество подарков, которые мы подберем, запишем позиции, в которых лежат подарки, в другой bitset. Тогда количество подарков, которое мы соберем, есть в точности количество позиций, в которых и подарок лежит и собирается подарок, то есть в обоих bitset стоит 1. Чтобы оставить только такие позиции, надо применить операцию AND, а чтобы найти количество единиц у bitset, нужно использовать функцию count; обе этих функции работают за $O(len/32)$. Итоговая асимптотика решения, неформально говоря, $O(n * \log_2(n) + n * n/32)$.

Задачу решили 5 участников, всего по ней была сделана 231 попытка.

Авторское решение

```
#include <bits/stdc++.h>

using namespace std;

const int maxn = (int) 6e4 + 10;

char s[maxn];
bitset < maxn > b;
int cnt[maxn];
bitset < maxn > now;

int main()
{
    int n;

    cin >> n;

    for (int i = 0; i < 2 * n + 1; i++) {
        char x;

        scanf(" %c", &x);

        now[i] = x - '0';
    }

    scanf("%s", s);

    int pos = n;

    for (int i = 0; i < n; i++) {
        if (s[i] == 'T') {
            cnt[pos]++;
            b[pos] = 1;
        } else if (s[i] == 'R') {
            pos++;
        } else {
            pos--;
        }
    }

    int dx = pos - n;

    int ans = (now & b).count();

    int st = n;

    for (int i = 0; i < n - 1; i++) {
        if (s[i] == 'L') {
            b <<= 1;
            st--;
        } else if (s[i] == 'R') {
            b >>= 1;
```

```
            st++;
        } else {
            if (cnt[st] == 0) {
                throw 1;
            }

            cnt[st]--;
            if (cnt[st] == 0) {
                b[n] = 0;
            }

            cnt[st + dx]++;

            if (cnt[st + dx] == 1) {
                b[n + dx] = 1;
            }
        }

        ans = max(ans, (int) (b & now).count());
    }

    cout << ans << endl;

    return 0;
}
```

Задача G. Бегущие по стадиону

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

На стадионе бегают n спортсменов, занумерованных числами от 1 до n, все со скоростью 1. Стадион представляет из себя две параллельные дорожки длины l. По одной из них спортсмены бегут влево, по другой – вправо; все спортсмены бегут с одинаковой скоростью. Добежав до конца стадиона, спортсмен мгновенно меняет направление и дорожку.

Как-то раз Петя пришел на стадион и записал расстояние от каждого спортсмена до левого края стадиона и его направление бега. Через некоторое время пришел Вася и сказал, что все расстояния от левого края стадиона до спортсменов различны, а сами спортсмены образуют перестановку p, считая слева направо. Нужно определить, возможна ли ситуация, в которой оба мальчика правы.

Формат входных данных

В первой строке даны два целых числа n и l ($1 \vdots n \vdots 10^5$, $1 \vdots l \vdots 10^9$) – количество спортсменов и длина стадиона.

Каждая из следующих n строк содержит два целых числа x_i, v_i ($0 ::: x_i ::: l$, $v_i \in \{-1, 1\}$) – расстояние от i-го спортсмена до левого конца стадиона и его направление бега, которые записал Петя.

Последняя строка содержит перестановку p чисел от 1 до n - перестановка спортсменов, которую увидел Вася.

Гарантируется, что все пары $\{x_i, v_i\}$ различны, а также, что если $x_i = 0$, то $v_i = 1$, а если $x_i = l$, то $v_i = -1$.

Формат выходных данных

Если описанная ситуация возможна, выведите *Yes*. Иначе выведите *No*.

Примеры

стандартный ввод	стандартный вывод
2 10 5 1 5 -1 1 2	Yes
4 4 0 1 1 -1 2 1 3 -1 2 1 4 3	Yes
4 8 8 -1 7 -1 0 1 2 1 1 3 4 2	No

Разбор

Заметим, что состояния всех спортсменов изменяются циклично с периодом $2l$. Нам нужно определить, существует ли момент времени, когда для любого i спортсмен с номером $p[i]$ стоит левее спортсмена $p[i+1]$. Для фиксированного i найдем все моменты времени, лежащие в промежутке $[0, 2l]$, когда $p[i]$-й спортсмен левее $p[i+1]$-го. Это некоторый «циклический» отрезок, то есть либо обычный отрезок, либо два отрезка таких, что начало одного совпадает с 0, а конец второго – с $2l$, так как за время $2l$ спортсмены находятся в одной точке ровно два раза. Остается только определить, существует ли точка, покрытая всеми циклическими отрезками.

Заменим циклические отрезки второго типа на два обычных отрезка. Теперь нужно проверить, существует ли точка, покрытая n раз. Отсортируем

все концы отрезков, и пробежимся по ним в порядке возрастания координаты. Будем прибавлять к счетчику единицу, если встретили начало отрезка, и вычитать, если конец. В итоге, нужно определить существует ли момент, когда счетчик равен n.

Асимптотика решения $O(n \log n)$.

По задаче было сделано 23 попытки, ни одна из которых не оказалась успешной.

Авторское решение

```
#include <bits/stdc++.h>

#define all(a) (a).begin(), (a).end()
#define pb push_back
#define sz(a) (int)(a).size()

using namespace std;

typedef long long ll;
typedef pair < int, int >pii;
typedef pair < ll, ll > pll;
typedef long double ld;

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    int n, len;
    cin >> n >> len;

    vector < int >x(n), v(n); for

    (int i = 0; i < n; ++i) {
        cin >> x[i] >> v[i];
    }

    vector < int >p(n);
    for (int i = 0; i < n; ++i) {
        cin >> p[i];
        --p[i];
    }

    vector < pair < ll, int >>event;
    int cur = 0;

    for (int k = 0; k + 1 < n; ++k) {
        int i = p[k];
        int j = p[k + 1];
```

```
    ll tm = 0;

    if (v[i] == -1 && v[j] == -1) {
        tm = x[i] + x[j];
    } else if (v[i] == 1 && v[j] == 1) {
        tm = (len - x[i]) + (len - x[j]);
    } else if (v[i] == 1 && v[j] == -1) {
        tm = abs(x[i] - x[j]);
        tm = (x[i] < x[j] ? tm : 2 * len - tm);
    } else {
        tm = abs(x[i] - x[j]);
        tm = (x[j] < x[i] ? tm : 2 * len - tm);
    }

    if (x[i] < x[j] || (x[i] == x[j] && v[i] < v[j])) {
        ++cur;
        event.push_back( {
                        tm, -1}
        );
        event.push_back(  {
                        tm + 2 * len, 1}
        );
    } else {
        event.push_back(  {
                        tm, 1}
        );
        event.push_back(  {
                        tm + 2 * len, -1}
        );
    }
}

event.push_back(  {
                0, 0}
);
sort(all(event));

for (int i = 0; i < sz(event);) {
    int j = i;
    for (; j < sz(event) && event[i].first == event[j].first; ++j) {
        cur += event[j].second;
    }
    // cout << event[i].first << " " << cur << endl;

    if (cur == n - 1 && event[i].first != 4 * len) {
        cout << "Yes" << endl;
        return 0;
    }
    i = j;
}
cout << "No" << endl;
return  0;
```

}

Задача H. Свитера Мейбл

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

На Рождество Мейбл решила устроить в хижине чудес распродажу своих старых свитеров, а дядя Стен согласился ей в этом помочь.

Оказалось, что у Мейбл на продажу имеются свитера только трёх цветов – красного (R), зелёного (G) и синего (B). Мейбл решила, что свитера одинакового цвета будут стоить одинаково. Также оказалось, что каждому жителю Гравити Фолз хочется купить ровно один свитер, и что каждому нравится только два цвета из трёх возможных. (Получается, что есть всего 6 вариантов предпочтений жителей по цветам: RB, BR, RG, GR, BG, GB).

Если жителю предлагают цвет, который ему не нравится, то он расстраивается и не покупает свитер. Мейбл не любит, когда кто-то расстраивается, поэтому хочет, чтобы все получили свитера соответственно их предпочтениям. А дядя Стен просто хочет заработать как можно больше денег.

Помогите Мейбл продать свитера так, чтобы никто из жителей не был расстроен, и при этом заработать как можно больше.

Формат входных данных

В первых трёх строках заданы по два числа через пробел: стоимость $cost(1 ::: cost ::: 10^9)$ и количество свитеров $cnt(1 ::: cnt ::: 10^5)$ каждого цвета. (В первой строке красного, во второй – зелёного, в третьей – синего)

В следующей строке дано количество жителей Гравити Фолз $n(1 ::: n ::: 10^5)$

В следующих n строках заданы предпочтения по цветам свитеров жителей Гравити Фолз.

Формат выходных данных

В первой строке выведите "NO"(без кавычек), если Мейбл не может продать каждому жителю по свитеру.

Иначе выведите "YES"(без кавычек) и в следующей строке выведите, какую максимальную сумму денег Мейбл при этом может заработать.

Примеры

стандартный ввод	стандартный вывод
1 1 2 1 3 1 3 RG GB RB	YES 6
1 1 2 1 3 1 3 RB RB BR	NO

Примечание

В первом тесте, первому продаем красный свитер, второму зеленый, третьему синий.

Во втором тесте, жителям суммарно надо 3 синих или красных свитера, а у Мейбл есть только 2.

Разбор

Так как у жителей всего может быть три различных предпочтения (RB, RG, BG), то посчитаем сколько суммарно есть жителей каждого из трех типов. Упорядочим свитера по уменьшению стоимости. Пусть первый тип самый дорогой, а третий самый дешевый. Рассмотрим жителей, которые хотят свитера первого и второго типов, и переберем, сколько свитеров первого типа Мейбл им продаст; тогда остальные автоматически купят свитера второго типа. Теперь у нас остались жители, которые хотят первый и третий тип, или второй и третий тип. Так как первый тип свитеров дороже третьего, то все оставшиеся свитера первого типа Мейбл продаст жителям, которые хотят первый или третий тип, оставшиеся жители возьмут обязательно третий тип. Аналогично, так как второй тип дороже третьего, то все оставшиеся свитера второго типа Мейбл продаст жителям, которые хотят второй или третий тип, а оставшиеся жители возьмут снова обязательно третий тип. В итоге нам нужно проверить, что свитеров третьего типа хватит на всех, и изменить ответ, если он стал лучше. Итоговая асимптотика $O(n)$.

Задачу решило 12 участников, всего по задаче было сделано 139 попыток.

Авторское решение

```
#include <bits/stdc++.h>

#define ll long long
#define ld double
#define pii pair <int, int>
#define forn(i, n) for (int i = 0; i < (int)n; i++)
#define mp make_pair
#define ui unsigned ll
#define pll pair <ll, ll>

using namespace std;

vector < pair < pii, int >>v;

char s[10];

int cnt[3];
int f[3];

int main()
{
    for (int i = 0; i < 3; i++) {
        int x, y;

        scanf("%d %d", &x, &y);

        v.push_back(mp(mp(x, y), i));
    }

    sort(v.rbegin(), v.rend());

    for (int i = 0; i < 3; i++) {
        f[v[i].second] = i;
    }

    int n;

    cin >> n;

    for (int i = 0; i < n; i++) {
        scanf("%s", s);

        vector < int >g;

        if (s[0] == 'R' || s[1] == 'R') {
            g.push_back(0);
        }

        if (s[0] == 'G' || s[1] == 'G') {
            g.push_back(1);
        }
```

```
    if (s[0] == 'B' || s[1] == 'B') {
        g.push_back(2);
    }

    g[0] = f[g[0]];
    g[1] = f[g[1]];

    sort(g.begin(), g.end());

    if (g[0] == 0) {
        if (g[1] == 1) {
            cnt[0]++;
        } else {
            cnt[1]++;
        }
    } else {
        cnt[2]++;
    }
}

ll ans = -1;

for (int i = 0; i <= cnt[0]; i++) {
    if (i > v[0].first.second) {
        continue;
    }

    if (cnt[0] - i > v[1].first.second) {
        continue;
    }

    ll now = (ll) v[0].first.first * i;
    now += (ll) v[1].first.first * (cnt[0] - i);

    int a = v[0].first.second - i;
    int b = v[1].first.second - (cnt[0] - i);
    int c = v[2].first.second;

    int cn1 = cnt[1];
    int cn2 = cnt[2];

    if (cn1 <= a) {
        now += (ll) v[0].first.first * cn1;
    } else {
        now += (ll) v[0].first.first * a;
        cn1 -= a;

        if (c < cn1) {
            continue;
        }

        now += (ll) v[2].first.first * cn1;
```

```
            c -= cn1;
        }

        if (cn2 <= b) {
            now += (ll) v[1].first.first * cn2;
        } else {
            now += (ll) v[1].first.first * b;
            cn2 -= b;

            if (c < cn2) {
                continue;
            }

            now += (ll) v[2].first.first * cn2;
        }

        ans = max(ans, now);
    }

    if (ans == -1) {
        cout << "NO" << endl;
    } else {
        cout << "YES" << endl;
        cout << ans << endl;
    }

    return 0;
}
```

Задача I. Диппер и таблица

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

О нет, на Гравити Фолз обрушилась новая беда! На этот раз на городок напал брат хитроумного Билла Шифра по имени Фрэнк Тэйбл. Он, как и его злобный братец, превращает жителей города в каменные статуи и разрушает всё вокруг.

К счастью, Дипперу и Мэйбл удалось укрыться и теперь они пытаются найти способ одолеть злодея. Диппер внимательно изучил дневники дедушки Форда и обнаружил, что Фрэнка Тэйбла можно победить, если разгадать секрет загадочной таблицы чисел. Эта таблица состоит из N строк и M столбцов, а в каждой её ячейке записано некоторое целое число $X[i][j]$, такое что $1 ::: X[i][j] ::: C$, где C - магическое число, записанное на полях дневника.

Дедушка Форд подсказал ребятам, что разгадкой секрета являются два массива целых чисел $A[i]$ и $B[j]$ размеров N и M соответственно, причем все

числа в массивах положительны и не превосходят C, а также верно, что для любых $1 ::: i ::: N$ и $1 ::: j ::: M$ выполнено условие $X[i][j] = min(A[i], B[j])$.

К сожалению, Диппер и Мэйбл так долго разбирались с этой задачей, что теперь Фрэнк стал слишком силён. Из-за этого ребятам надо не просто найти по имеющейся таблице массивы $A[i]$ и $B[j]$, а посчитать количество пар подходящих под условия массивов. Эта задача оказалась совсем сложной для брата с сестрой, поэтому они просят вас помочь им, пока не стало слишком поздно!

Так как ответ на задачу может быть очень большим, выведите его остаток от деления на $10^9 + 7$.

Формат входных данных

В первой строке содержатся три целых числа N, M и C ($1 ::: N * M ::: 10^5$, $1 ::: C ::: 10^9$) - размеры таблицы. Далее следует описание таблицы - N строк, в i-й из которых содержатся M чисел $X[i][j]$ ($1 ::: X[i][j] ::: C$).

Формат выходных данных

Выведите остаток от деления числа возможных пар массивов $A[i]$ и $B[j]$ на $10^9 + 7$.

Примеры

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

Примечание

Рассмотрим первый тест из условия. В нём оба массива должны иметь длину 1 и все числа в них должны не превосходить 3. При этом, $min(A[1], B[1]) = 2$, значит нам подходят такие пары массивов:

- $A[1] = 2, B[1] = 2$
- $A[1] = 2, B[1] = 3$
- $A[1] = 3, B[1] = 2$

Ясно, что во всех других массивах длины 1 из элементов не превосходящих 3 не будет выполнено условие $min(A[1], B[1]) = 2$. Следовательно, ответ 3.

Разбор

Сначала скажем, что $A[i]$ = максимуму в i-ой строке, а $B[i]$ = максимуму в i-ом столбце. Действительно, если существует хотя бы одна подходящая пара массивов A и B, то такие массивы должны подойти. В самом деле, посмотрим на i-ую строку матрицы $X[][]$. Ее элемент $X[i][j]$ равен минимуму из $A[i]$ и $B[j]$. Таким образом, если очередной $B[j]$ меньше, чем $A[i]$, то $X[i][j] = B[j]$, иначе он равен $A[i]$. Это означает, что либо наибольший из элементов строки равен $A[i]$, либо все элементы строки равны $B[j]$. Но тогда мы можем сказать, что $A[i]$ равен наибольшему из них, также ничего не нарушив. Абсолютно аналогичные рассуждения можно провести для каждого столбца, чтобы обосновать построение массива $B[j]$.

Построив таким образом массивы $A[]$ и $B[]$, нужно проверить, что матрица $X[][]$ корректна и соответствует им. Если это не так, нужно сразу вывести 0 как ответ на задачу.

Теперь, чтобы посчитать число допустимых пар массивов, посмотрим, какие элементы в них можно менять. Если для некоторых i и j $A[i] \neq B[j]$, то меньшее из $A[i]$ и $B[j]$ меняться не может, так как тогда изменится значе- ние $X[i][j]$, чего быть не должно. Это значит, что мы можем менять только такие элементы $A[]$, которые больше всех элементов $B[]$, и такие элементы $B[]$, которые больше всех элементов $A[]$. Но по построению массивов $A[]$ и $B[]$, эти элементы одинаковы в обоих массивах и равны наибольшему значению в матрице $X[][]$.

Найдем теперь значение максимума в матрице $X[][]$ и количество таких максимумов в массивах $A[]$ и $B[]$. Обозначим эти значения соответственно за $maxval$, $k1$, $k2$. Заметим, что одновременно изменять значения в массивах $A[]$ и $B[]$ мы не можем, так как если мы изменим значения $A[i]$ и $B[j]$, то значение $X[i][j]$ станет некорректным.

Суммируя все вышесказанное, получаем, что изменять мы можем только наибольшие значения причем только в одном из массивов одновременно. Более того, это изменяемое наибольшее значение можно сделать сколь угодно большим, но не больше числа C из условия задачи. Это значит, что итоговый ответ можно вычислить по формуле: $(C - maxval + 1)^{k1} + (C - maxval + 1)^{k2} - 1$. Действительно, мы выбираем один из массивов, в нем каждый из $k1$ (или $k2$ в случае массива $B[]$) максимальных элементов может принять любое значение, большее либо равное своему текущему значению. 1 нужно вычесть, так как вариант, при котором мы ничего не изменяем, будет учтен в обоих слагаемых, то есть дважды.

По задаче было сделано 5 попыток, ни одна из которых не оказалась успешной.

Авторское решение

```
#include <bits/stdc++.h>

using namespace std;

#define ll long long
#define mp make_pair

const double pi = 3.14159265358979323846;
const double eps = 1e-6;
const ll maxn = 100010;
const ll inf = 1000000007;
const ll mod = 1000000007;
#define ld double

int n, m, x, c, a[maxn], b[maxn];
vector < vector < int >>v(maxn);

ll bpow(ll x, ll n)
{
    if (n == 0) {
        return 1;
    } else if (n == 1) {
        return x;
    } else if (n % 2 == 0) {
        ll tmp = bpow(x, n / 2);
        return (tmp * tmp) % mod;
    } else {
        return (x * bpow(x, n - 1)) % mod;
    }
}

int main()
{

    cin >> n >> m >> c;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> x;
            v[i].push_back(x);
            a[i] = max(a[i], x);
            b[j] = max(b[j], x);
        }
    }
    int mx = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (v[i][j] != min(a[i], b[j])) {
                cout << 0;
                return 0;
            }
            mx = max(mx, v[i][j]);
```

```
        }
    }
    int k1 = 0, k2 = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] == mx) {
            k1++;
        }
    }
    for (int i = 0; i < m; i++) {
        if (b[i] == mx) {
            k2++;
        }
    }
    ll ans = (bpow(c - mx + 1, k1) + bpow(c - mx + 1, k2) - 1 + mod) % mod;
    cout << ans;

    return 0;
}
```

Задача J. Региональный вопрос

Имя входного файла:	стандартный ввод
Имя выходного файла:	стандартный вывод
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

В Берляндии есть несколько городов, причем в некоторых из них находятся кинотеатры.

Между некоторыми парами городов проложены дороги таким образом, что от каждого города можно добраться до каждого и количество дорог на единицу меньше количества городов. Расстоянием между двумя городами назовем минимальное количество дорог, которые нужно пройти, чтобы добраться от одного из них до другого.

Для простоты пронумеруем все города и скажем, что столица имеет номер 1. Известно, что Берляндия разбита на регионы. Для них верны следующие факты:

- Каждый город принадлежит ровно одному региону.
- Пусть известно минимальное и максимальное расстояние от столицы до города из региона. Тогда все города, расстояния от которых до столицы находятся между этими величинами включительно, принадлежат тому же самому региону.
- Между двумя городами региона может не существовать пути, не выходящего за пределы этого региона.

- У всех граждан Берляндии развито чувство прекрасного и чувство справедливости, поэтому от каждого города Берляндии можно добраться до одинакового числа кинотеатров, не выходя за пределы региона.

Так как Берляндия сравнительно малоизвестная страна, а поля туристических брошюр слишком узки, то можно легко найти информацию про дорожную схему Берляндии и расположение кинотеатров, но сложно узнать что-то про регионы.

Владислав как истинный кинолюбитель планирует посетить Берляндию. Он не хочет иметь какие-либо проблемы с передвижением внутри страны, поэтому просит вас хотя бы найти количество вариантов разделить страну на регионы так, чтобы это не противоречило вышеупомянутым фактам.

Два варианта считаются различными, если есть пара городов, находящихся в одном регионе в первом варианте и не находящихся в одном регионе во втором варианте.

Ответ может оказаться очень большим, поэтому выведите его остаток от деления на $10^9 + 7$.

Формат входных данных

В первой строке даны целые числа n, k ($1 \vdots k \vdots n \vdots 10^5$) – количество городов и кинотеатров.

Во второй строке через пробел даны k различных чисел a_i ($1 \vdots a_i \vdots n$) – номера городов, в которых есть кинотеатры. Гарантируется, что все номера различны.

Дальше идет $n-1$ строка. В i-й из них даны два числа u_i, v_i ($1 \vdots u_i, v_i \vdots n$) - номера городов между которыми проведена дорога.

Формат выходных данных

Выведите остаток от деления на $10^9 + 7$ количества вариантов разделения Берляндии на регионы.

Примеры

стандартный ввод	стандартный вывод
7 3 5 2 6 1 2 2 3 3 4 4 5 5 6 6 7	4
8 6 1 2 5 6 7 8 2 1 2 3 2 4 5 3 6 3 4 7 4 8	3

Разбор

После разделения на регионы образуется несколько связных частей, в которых по условию должно быть одинаковое количество кинотеатров. Понятно, что это количество делит общее количество кинотеатров. Можно для каждого делителя d общего количества найти количество способов разделения на регионы так, чтобы в каждой связной части было d кинотеатров и потом просуммировать все эти числа.

Пусть мы зафиксировали d – количество кинотеатров в каждой части. Будем считать динамику dp_h - если мы начнем новый регион на высоте h, то сколько вариантов разделить на регионы все, что ниже.

Допустим на h-й высоте начинается cnt поддеревьев и сумма количеств кинотеатров в них sum. Тогда на высоте, где начнется следующий регион, сумма количеств кинотеатров в поддеревьях будет $nextSum = sum - cnt \cdot d$. Отсюда dp_h равно сумме dp_k, где на k-й высоте в поддеревьях будет $nextSum$ кинотеатров, или нулю, если начало нового региона отсекает сверху связные части с количествами кинотеатров, не равными d.

Осталось научиться проверять корректность начала нового региона. Достаточно проверить, что в каждом из cnt поддеревьев количество кинотеатров делится на d и что в каждой из связных частей будет хотя бы одна вершина. Если хотя бы один из этих фактов ложен, то присвоим dp_h ноль, иначе сумму dp_k. Тогда если оба факта истинны, но есть связная часть не с d кинотеатра-

ми, то нетрудно показать, что dp_h все равно будет равно нулю. Проверять эти факты можно, предподсчитав в начале размеры поддеревьев и самый близкий к корню каждого поддерева кинотеатр.

Итак, мы умеем находить для каждого делителя числа k количество способов за линейное время, что достаточно для решения задачи.

По задаче было сделано 8 попыток, ни одна из которых не оказалась успешной.

Авторское решение

```
#include <bits/stdc++.h>

using namespace std;

#define sz(x) ((int) (x).size())
const int maxn = 1e5 + 10;
const int mod = 1e9 + 7;
const int inf = 1e9;

int gcd(int a, int b)
{
    while (a && b) {
        if (a >= b)
            a %= b;
        else
            b %= a;
    }
    return a + b;
}

int n, k;
char marked[maxn];
vector < int >graph[maxn];
int size[maxn];
int h[maxn];
int maxh;
vector < int >layer[maxn];
int gcdLayer[maxn];
int closestMarked[maxn];
int closestMarkedOnLayer[maxn];
int sumSizeOnLayer[maxn];
int startSize[maxn];
int endSize[maxn];
int dp[maxn];
int suf[maxn];

void dfs(int v, int p)
{
    size[v] = (marked[v]) ? 1 : 0;
    closestMarked[v] = (marked[v]) ? h[v] : inf;
  for (auto to:graph[v]) {
```

```
        if (to != p) {
            h[to] = h[v] + 1;
            dfs(to, v);
            size[v] += size[to];
            closestMarked[v] = min(closestMarked[v], closestMarked[to]);
        }
    }
    maxh = max(maxh, h[v]);
    layer[h[v]].push_back(v);
    gcdLayer[h[v]] = gcd(gcdLayer[h[v]], size[v]);
    closestMarkedOnLayer[h[v]] =
        max(closestMarkedOnLayer[h[v]], closestMarked[v]);
    sumSizeOnLayer[h[v]] += size[v];
}

int solve(int d)
{
    dp[maxh + 1] = 1;
    for (int i = maxh; i >= 0; --i) {
        suf[i + 1] = suf[i + 2] + dp[i + 1];
        if (suf[i + 1] >= mod)
            suf[i + 1] -= mod;
        int nextSum = sumSizeOnLayer[i] - d * sz(layer[i]);
        if (gcdLayer[i] % d == 0 && nextSum >= 0
            && closestMarkedOnLayer[i] < startSize[nextSum]) {
            dp[i] = suf[startSize[nextSum]] - suf[endSize[nextSum] + 1];
            if (dp[i] < 0)
                dp[i] += mod;
        } else {
            dp[i] = 0;
        }
    }
    return dp[0];
}

signed main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> k;
    for (int i = 0; i < k; ++i) {
        int v;
        cin >> v;
        marked[v] = true;
    }
    for (int i = 1; i < n; ++i) {
        int a, b;
        cin >> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }
```

```
    dfs(1, -1);
    for (int i = 0; i < maxn; ++i)
        startSize[i] = endSize[i] = -1;
    for (int i = maxh; i >= 0; --i)
        startSize[sumSizeOnLayer[i]] = i;
    for (int i = 0; i <= maxh; ++i)
        endSize[sumSizeOnLayer[i]] = i;
    startSize[0] = endSize[0] = maxh + 1;

    int answer = 0;
    for (int d = 1; d <= k; ++d) {
        if (k % d == 0) {
            answer += solve(d);
            if (answer >= mod)
                answer -= mod;
        }
    }
    cout << answer << '\n';
}
```