11 Класс. Задачи 1-3

Защитный экран

Условие.

В будущем, Объединенные корпорации Земли работают над строительством новых видов атомных реакторов. Ведущая атомная корпорация взяла на себя важную задачу: возведение защитных конструкций вокруг нового реактора — огромного, стабильного и безопасного, который обещает стать источником чистой энергии для всех планет в Солнечной системе.

Однако безопасность — превыше всего. Вокруг реактора должен быть возведен многоуровневый защитный барьер, один из ключевых компонентов которого — **энергетический забор**, состоящий из **n защитных панелей**. Каждая панель покрыта специальным цветовым барьером, что позволяет эффективно рассеивать радиацию и защищать реактор от внешних угроз.

Но возникает проблема: распределение цветовых барьеров на панелях детально просчитано и должно строго следовать сложной научной логике, определенной спроектированной **матрицей безопасности**.

Выяснилось, что для оптимальной работы защиты используются **4 цвета барьеров**, каждый из которых может следовать за другим согласно строгим правилам. Эту информацию инженеры Росатома закодировали в специальную **матрицу допусков** размером 4×4. В этой матрице, на позиции (i, j), стоит **1**, если цвет **j** допускает, чтобы за ним следовал цвет **i** (т. е. последовательность **ji** разрешена). Позиция (i, j) находится на пересечении i-ой строки и j-ого столбца (строки по горизонтали, столбцы по вертикали).

Задача, поставленная перед вами проста, но крайне важна: вычислить **количество возможных способов покраски**, следуя строгим правилам матрицы для цветов на панелях забора. Результат нужно предоставить по модулю **1e9 + 7**, чтобы избежать переполнения.

Помните, каждый неправильный расчет может поставить под угрозу безопасность реактора и, как следствие, всей Земли!

Формат ввода.

В первой строке вводится одно число n $(1 \leq n \leq 100000)$ $-$ количество панелей, которые нужно будет использовать для построения забора.
В последующих четырех строках вводится матрица 4x4, описывающая, какие цвета могут следовать друг за другом.

Формат вывода.

В единственной строке выведите количество возможных корректных последовательностей покраски энергетического забора из **n панелей по правилам матрицы допустимости цветов** по модулю ($1e9 + 7$).

Пример 1

Ввод	Вывод
4 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	2

Пример 2

Ввод	Вывод
5 0 1 1 0 0 1 1 0 1 1 1 1 0 1 1 0	208

Примечание

4
1 0 0 0
0 1 0 0
0 0 0 0
0 0 0 0

В данном тесте после первого цвета может идти только первый, после второго — только второй. Два существующих варианта: 1111, 2222.

Решения

Язык C++

```cpp
#include <cassert>
#include <iostream>
#include <stdint.h>
#include <vector>

#define iom
\
  std::cin.tie(0);
\
  std::cout.tie(0);
\
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;

const int MOD = 1e9 + 7;
std::vector<int> permitterPrevColor[4];

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

void solve(std::istream &cin, std::ostream &cout) {

  int n;
  cin >> n;

  for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
      int cur;
      cin >> cur;
      assert(cur >= 0);
      assert(cur <= 1);
      if (cur == 1) {
        permitterPrevColor[i].push_back(j);
      }
    }
  }

  std::vector<std::vector<int>> dp(n, std::vector<int>(4, 0));
```

```
  for (int i = 0; i < 4; ++i) {
    dp[0][i] = 1;
  }

  for (int i = 1; i < n; ++i) {
    for (int color = 0; color < 4; ++color) {
      for (auto prevColor : permitterPrevColor[color]) {
        add(dp[i][color], dp[i - 1][prevColor]);
      }
    }
  }

  int ans = 0;
  for (int color = 0; color < 4; ++color) {
    add(ans, dp[n - 1][color]);
  }

  cout << ans << endl;
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

Язык Python

```
MOD = int(1e9+7)
permitedPrevColor = [[] for i in range(4)]

n = int(input())
for prevColor in range(4):
    line = list(map(int, input().split()))
    for curColor in range(4):
        if line[curColor] == 1:
            permitedPrevColor[curColor].append(prevColor)

dp = [[0 for i in range(4)] for j in range(n)]
```

```
for color in range(4):
    dp[0][color] = 1

for i in range(1, n):
    for color in range(4):
        for prevColor in permitedPrevColor[color]:
            dp[i][color] = (dp[i][color] + dp[i - 1][prevColor]) % MOD

ans = 0
for color in range(4):
    ans = (ans + dp[n - 1][color]) % MOD
print(ans)
```

Привет, это мониторинг

Условие.

Далекое будущее. Страна Z стала мировым лидером в использовании передовых атомных технологий. Корпорация **R** разработала новый тип сверхмощных термоядерных реакторов, которые питают огромные мегаполисы и промышленные комплексы по всей планете. Работа этих реакторов требует ювелирной точности и осторожности: сюда входит контроль температуры, плотности плазмы и многомиллиардных данных, поступающих от тысяч датчиков в реальном времени.

Для управления и мониторинга состояния термоядерной установки инженеры-контролеры разработали специальную систему, которая отображает важные параметры на так называемом **матриц-экране** — суперкомпьютере с огромной матрицей данных размером **n × m**. Каждый элемент матрицы содержит уникальную информацию о текущем состоянии элементов реактора. Числа, расположенные в этой матрице, выражают критические данные, такие как температура, давление, напряжение и пр.

Система управления получает множественные запросы от операторов, которые отвечают за стабилизацию различных частей реактора. Каждому запросу соответствует объединение данных по определённым строке и столбцу — набор данных создается путём объединения информации из одной строки и одного столбца (значение на пересечении строки и столбца берется **один** раз). По результатам такого объединения нужно быстро находить **k-ый по величине важный параметр** из полученного множества данных.

Формат ввода.

В первой строке через пробел вводятся два числа n, m ($1 \le n \times m \le 10^5$) $-$ количество строк и столбцов в матрице данных.
В следующих n строках через пробел вводятся по m чисел. Каждая ячейка содержит целое число x_{ij} ($1 \le x_{ij} \le 10^9$).
В $(n+2)$-ой строке вводится число q ($1 \le q \le 10^5$) $-$ количество запросов.
В следующих q строках содержатся по три числа i, j, k ($0 \le i < n$, $0 \le j < m$, $1 \le k \le n + m - 1$), где

- i - номер строки;
- j - номер столбца;
- k - позиция в отсортированном массиве данных, который получается при объединении i-ой строки и j-ого столбца.

Формат вывода.

В q строках выведите ответ на задачу $-$ **k-ое по величине значение**, находящиеся в объединении указанных строки и столбца.

Пример.

Ввод	Вывод
4 3 30 84 22 78 76 64 80 8 87 3 55 84 8 3 2 1 3 0 5 3 2 6 0 1 4 1 2 3 2 2 4 1 2 6 1 1 6	3 80 87 55 76 80 87 84

Примечание

4 3

```
30 84 22
78 76 64
80 8 87
```

`3 55 84`

Для первого запроса: `3 2 1` построим и отсортируем объединенный массив: [3 22 55 64 84 87]. Элемент на 1 позиции - `3`.
Для третьего запроса: `3 2 6` аналогичный массив, как для первого. Элемент на 6 позиции - `87`.
Для четвертого запроса: `0 1 4` построим и отсортируем объединенный массив: [8 22 30 55 76 84]. Элемент на 4 позиции - `55`.

Решения

Язык C++

```
#include <algorithm>
#include <iostream>
#include <stdint.h>
#include <vector>

#define iom
\
  std::cin.tie(0);
\
  std::cout.tie(0);
\
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"
using namespace std;

using ll = long long;
using ull = unsigned long long;

int n, m;
vector<vector<int>> matrix, rowMatrix, columnMatrix;

int bin_search(int val, const vector<int> &v, int l, int r) {
  while (l < r) {
    auto mid = (l + r) >> 1;
    if (v[mid] <= val) {
      l = mid + 1;
    } else {
      r = mid;
    }
  }
```

```
  return l;
}

int calc_ans(int i, int j, int k) {
  const auto &row = rowMatrix[i];
  int l_row = 0, r_row = row.size();

  const auto &column = columnMatrix[j];
  int l_column = 0, r_column = column.size();

  int l = min(row[0], column[0]);
  int r = max(row[row.size() - 1], column[column.size() - 1]);

  while (r > l) {
    int mid = (l + r) >> 1;
    auto less_or_equal_in_rows = bin_search(mid, rowMatrix[i], l_row,
r_row);
    auto less_or_equal_in_columns =
        bin_search(mid, columnMatrix[j], l_column, r_column);

    auto total = less_or_equal_in_rows + less_or_equal_in_columns;
    if (matrix[i][j] <= mid) {
      total--;
    }

    if (total < k) {
      l = mid + 1;

      l_row = less_or_equal_in_rows;
      l_column = less_or_equal_in_columns;
    } else {
      r = mid;

      r_row = less_or_equal_in_rows;
      r_column = less_or_equal_in_columns;
    }
  }

  return l;
}

void solve(std::istream &cin, std::ostream &cout) {
  cin >> n >> m;

  matrix.clear();
  rowMatrix.clear();
```

```
  columnMatrix.clear();

  matrix.resize(n, vector<int>(m));
  rowMatrix.resize(n, vector<int>(m));
  columnMatrix.resize(m, vector<int>(n));

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      cin >> matrix[i][j];

      rowMatrix[i][j] = matrix[i][j];
      columnMatrix[j][i] = matrix[i][j];
    }
  }

  for (int i = 0; i < n; ++i) {
    sort(rowMatrix[i].begin(), rowMatrix[i].end());
  }
  for (int j = 0; j < m; ++j) {
    sort(columnMatrix[j].begin(), columnMatrix[j].end());
  }

  int q;
  cin >> q;

  while (q--) {
    int i, j, k;
    cin >> i >> j >> k;

    int ans = calc_ans(i, j, k);
    cout << ans << endl;
  }
}

int32_t main() {
  solve(std::cin, std::cout);
  out;
  return 0;
}
```

Язык Python

```
def bin_search(val, v, l, r):
    while l < r:
```

```
        mid = (l + r) >> 1
        if v[mid] <= val:
            l = mid + 1
        else:
            r = mid
    return l

def calc_ans(i, j, k):
    global n
    global m
    global matrix
    global rowMatrix
    global columnMatrix

    row = rowMatrix[i]
    l_row = 0
    r_row = len(row)

    column = columnMatrix[j]
    l_column = 0
    r_column = len(column)

    l = min(row[0], column[0])
    r = max(row[r_row - 1], column[r_column - 1])

    while r > l:
        mid = (l + r) >> 1
        less1 = bin_search(mid, row, l_row, r_row)
        less2 = bin_search(mid, column, l_column, r_column)

        total = less1 + less2
        if matrix[i][j] <= mid:
            total -= 1

        if total < k:
            l = mid + 1
            l_row = less1
            l_column = less2
        else:
            r = mid

            r_row = less1
            r_column = less2

    return l
```

```
n, m = map(int, input().split())
matrix = [list(map(int, input().split())) for i in range(n)]

rowMatrix = [[-1 for j in range(m)] for i in range(n)]
columnMatrix = [[-1 for j in range(n)] for i in range(m)]
for i in range(n):
    for j in range(m):
        rowMatrix[i][j] = matrix[i][j]
        columnMatrix[j][i] = matrix[i][j]

for i in range(n):
    rowMatrix[i].sort()
for j in range(m):
    columnMatrix[j].sort()

q = int(input())
for _ in range(q):
    i, j, k = map(int, input().split())

    ans = calc_ans(i, j ,k)
    print(ans)
```

Код доступа

Условие.

В одной из секретных лабораторий корпорации **R** произошла аварийная ситуация — в реакторной системе произошел сбой, который может привести к серьезным последствиям. Для предотвращения катастрофы необходимо ввести специальный код доступа, который управляет системой охлаждения реактора и стабилизирует положение.

Однако код доступа был зашифрован и разбит на части для дополнительной безопасности. После накопления данных со всех секций, хранящих зашифрованные сегменты кода, удалось получить несколько частей строки, но они пришли перемешанными и частично пересекающимися.

Код представлял собой строку длины **N**, состоящую из строчных символов латинского алфавита, но после шифровки и передачи она была разбита на **n-k+1** частей с длиной каждого сегмента, равной **k**. Сегменты были получены из исходной строки следующим образом: сначала взяли первые **k** элементов, потом вторые **k** элементов и так далее.

Каждая часть состоит из последовательных символов оригинальной строки, но пришла в хаотичном порядке. Без восстановления исходной строки невозможно ввести код и спасти ситуацию в реакторе.
Используя все доступные данные (n-k+1 строк), вычислите оригинальную строку длины **N**.

Формат ввода.

В первой строке через проблем вводятся два числа n и k ($2 \leq n \leq 10^5$, $2 \leq k \leq min(500, n)$) $-$ длина исходной строки и одного из сегментов.
В следующей $(n - k + 1)$ строке вводится один из сегментов исходной строки.

Формат вывода.

В первой строке выведите полученную строку длины n.
Во второй строке выведите через пробел номера (индексация с единицы) строк из входной последовательности, с помощью которых эта строка была получена (если существует несколько ответов, будет принят любой из них).

Пример 1

Ввод	Вывод
7 2 ab ba ac ca ab ba	cababac 4 1 2 5 6 3

Пример 2

Ввод	Вывод
10 4 abob boba obaa baab aabe abeb beba	abobaabeba 1 2 3 4 5 6 7

Примечание

```
7 2
ab
ba
ac
ca
ab
ba
```

Решение:

```
cababac
ca – 4
 ab – 1
  ba – 2
   ab – 5
    ba – 6
     ac – 3
```

Также в тестирующей системе будут засчитаны решения `babacab` и `abacaba`.

Решения

Язык C++

```
#include <algorithm>
#include <cassert>
#include <iostream>
#include <stack>
#include <string>
#include <unordered_map>
#include <vector>

#define iom
\
  std::cin.tie(0);
\
  std::cout.tie(0);
\
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
```

```cpp
#define endl "\n"

using ll = long long;
using ull = unsigned long long;
using namespace std;

int n, k;
vector<int> powers1, powers2;
const int P1 = 31;
const int P2 = 53;
const int MA = 1e9 + 7;
const int MB = 1e9 + 9;

void precalc() {
  powers1.clear();
  powers2.clear();
  powers1.resize(k + 1);
  powers2.resize(k + 1);

  powers1[0] = powers2[0] = 1;
  for (int i = 1; i <= k; ++i) {
    powers1[i] = (powers1[i - 1] * P1) % MA;
    powers2[i] = (powers2[i - 1] * P2) % MB;
  }
}

struct num {
  int a, b;

  num() {}
  num(int x) : a(x), b(x) {}

  explicit num(const char c, const char start_c, int i) {
    a = ((c - start_c + 1) * powers1[i]) % MA;
    b = ((c - start_c + 1) * powers2[i]) % MB;
  }

  num(int a, int b) : a(a), b(b) {}

  num operator+(const num &x) const {
    return num((a + x.a) % MA, (b + x.b) % MB);
  }
  num operator-(const num &x) const {
    return num((a + MA - x.a) % MA, (b + MB - x.b) % MB);
  }
  num operator*(int x) const { return num(((ll)a * x) % MA, ((ll)b * x) %
```

```
MB); }
  num operator*(const num &x) const {
    return num(((ll)a * x.a) % MA, ((ll)b * x.b) % MB);
  }
  bool operator==(const num &x) const { return a == x.a && b == x.b; }

  explicit operator ll() const { return (ll)a * MB + b + 1; } // > 0
};

struct Node {
  string s;

  num start;
  num finish;

  Node(string _s) : s(std::move(_s)) {
    int start_power = 0;
    int finish_power = 0;

    start = num(s[0], 'a', start_power++);
    finish = num(0);

    for (int i = 1; i + 1 < k; ++i) {
      start = start + num(s[i], 'a', start_power++);
      finish = finish + num(s[i], 'a', finish_power++);
    }

    finish = finish + num(s[k - 1], 'a', finish_power++);
  }
};

vector<Node> strings;
unordered_map<ll, int> indeg;
unordered_map<ll, int> outdeg;
unordered_map<ll, int> used;
unordered_map<ll, vector<pair<ll, int>>>
    graph; // endNode: [(startNode, nodeIdx)]

void readGraph(std::istream &cin, std::ostream &cout) {
  strings.clear();
  indeg.clear();
  outdeg.clear();
  graph.clear();

  for (int _ = 0; _ < n - k + 1; ++_) {
    string s;
```

```
    cin >> s;
    strings.emplace_back(s);

    ll startll = (ll)strings.back().start;
    if (graph.find(startll) == graph.end()) {
      graph[startll] = vector<pair<ll, int>>();
    }

    ll finishll = (ll)strings.back().finish;
    if (graph.find(finishll) == graph.end()) {
      graph[finishll] = vector<pair<ll, int>>();
    }

    graph[startll].emplace_back(finishll, strings.size() - 1);

    indeg[finishll]++;
    outdeg[startll]++;
  }
}

pair<ll, ll> findStartAndInitUsed() {
  used.clear();

  ll startNode = -1, finishNode = -1;
  for (const auto &[node, rebs] : graph) {
    used[node] = 0;

    if (outdeg[node] - indeg[node] == 1) {
      startNode = node;
    } else if (indeg[node] - outdeg[node] == 1) {
      finishNode = node;
    }
  }
  if (startNode == -1) {
    startNode = (*graph.begin()).first;
  }

  return {startNode, finishNode};
}

vector<int> getNodesOrder(ll startNode) {
  stack<pair<ll, int>> st;
  vector<int> nodes;
  st.emplace(startNode, -1);

  while (!st.empty()) {
```

```
    auto [v, stringIdx] = st.top();
    const auto &rebs = graph[v];

    if (used[v] == rebs.size()) {
      if (stringIdx != -1) {
        nodes.push_back(stringIdx);
      }

      st.pop();
      continue;
    }

    auto next = rebs[used[v]++];
    st.emplace(next);
  }
  reverse(nodes.begin(), nodes.end());

  return nodes;
}

string restoreString(const vector<int> &nodes) {
  string s(n, '\0');
  int s_idx = 0;
  for (int fIdx = 0; fIdx < strings[nodes[0]].s.size(); ++fIdx) {
    s[s_idx++] = strings[nodes[0]].s[fIdx];
  }
  for (int idx = 1; idx < nodes.size(); ++idx) {
    s[s_idx++] = strings[nodes[idx]].s.back();
  }
  return s;
}

void solve(std::istream &cin, std::ostream &cout) {
  cin >> n >> k;

  precalc();

  readGraph(cin, cout);

  auto [startNode, finishNode] = findStartAndInitUsed();

  auto nodes = std::move(getNodesOrder(startNode));
  auto str = std::move(restoreString(nodes));

  cout << str << endl;
  for (int i = 0; i < nodes.size(); ++i) {
```

```
    cout << nodes[i] + 1;
    if (i + 1 != nodes.size()) {
      cout << " ";
    }
  }
  cout << endl;
}

int32_t main() {
  iom;
  solve(std::cin, std::cout);
  out;
  return 0;
}
```

Язык Python

```
MOD_A = int(1e9 + 7)
MOD_B = int(1e9 + 9)
P1 = 31
P2 = 53

n, k = 0, 0
powers1, powers2 = [], []

def precalc():
    global powers1, powers2
    powers1 = [1] * (k + 1)
    powers2 = [1] * (k + 1)

    for i in range(1, k + 1):
        powers1[i] = (powers1[i - 1] * P1) % MOD_A
        powers2[i] = (powers2[i - 1] * P2) % MOD_B

class num:
    def __init__(self, a=0, b=0):
        self.a = a
        self.b = b

    @staticmethod
    def from_char(c, start_c, i):
        _a = ((ord(c) - ord(start_c) + 1) * powers1[i]) % MOD_A
        _b = ((ord(c) - ord(start_c) + 1) * powers2[i]) % MOD_B
        return num(_a, _b)
```

```
    def __add__(self, other):
        return num((self.a + other.a) % MOD_A, (self.b + other.b) % MOD_B)

    def __sub__(self, other):
        return num((self.a - other.a + MOD_A) % MOD_A, (self.b - other.b +
MOD_B) % MOD_B)

    def __mul__(self, x):
        if isinstance(x, int):
            return num((self.a * x) % MOD_A, (self.b * x) % MOD_B)
        else:
            return num((self.a * x.a) % MOD_A, (self.b * x.b) % MOD_B)

    def __eq__(self, other):
        return self.a == other.a and self.b == other.b

    def __int__(self):
        return self.a * MOD_B + self.b + 1

    def __hash__(self):
        return int(self)


class Node:
    def __init__(self, s):
        self.s = s
        start_power = finish_power = 0

        self.start = num.from_char(s[0], 'a', start_power)
        start_power += 1
        self.finish = num(0, 0)

        for i in range(1, k - 1):
            self.start += num.from_char(s[i], 'a', start_power)
            start_power += 1
            self.finish += num.from_char(s[i], 'a', finish_power)
            finish_power += 1

        self.finish += num.from_char(s[k - 1], 'a', finish_power)


strings = []
indeg, outdeg, used = {}, {}, {}
graph = {}

def readGraph():
```

```
    global strings, indeg, outdeg, graph
    strings.clear()
    indeg.clear()
    outdeg.clear()
    graph.clear()

    for _ in range(n - k + 1):
        s = input().strip()
        strings.append(Node(s))

        startll = int(strings[-1].start)
        finishll = int(strings[-1].finish)

        if startll not in graph:
            graph[startll] = []
        if finishll not in graph:
            graph[finishll] = []

        graph[startll].append((finishll, len(strings) - 1))
        indeg[finishll] = indeg.get(finishll, 0) + 1
        outdeg[startll] = outdeg.get(startll, 0) + 1

def findStartAndInitUsed():
    global used
    used.clear()

    start_node = finish_node = -1
    for node in graph:
        used[node] = 0
        if outdeg.get(node, 0) - indeg.get(node, 0) == 1:
            start_node = node
        elif indeg.get(node, 0) - outdeg.get(node, 0) == 1:
            finish_node = node
    if start_node == -1:
        start_node = list(graph.keys())[0]
    return start_node, finish_node

def getNodesOrder(start_node):
    st = [(start_node, -1)]
    nodes = []

    while st:
        v, string_idx = st[-1]
        rebs = graph[v]

        if used[v] == len(rebs):
```

```
            if string_idx != -1:
                nodes.append(string_idx)
            st.pop()
            continue

        next_node = rebs[used[v]]
        used[v] += 1
        st.append(next_node)

    nodes.reverse()
    return nodes

def restore_string(nodes):
    s = [''] * n
    s_idx = 0

    for fIdx in range(len(strings[nodes[0]].s)):
        s[s_idx] = strings[nodes[0]].s[fIdx]
        s_idx += 1

    for idx in range(1, len(nodes)):
        s[s_idx] = strings[nodes[idx]].s[-1]
        s_idx += 1

    return ''.join(s)

def solve():
    global n, k
    n, k = map(int, input().split())

    precalc()
    readGraph()

    start_node, finish_node = findStartAndInitUsed()
    nodes = getNodesOrder(start_node)
    result_str = restore_string(nodes)

    print(result_str)
    print(" ".join(str(x + 1) for x in nodes))


if __name__ == "__main__":
    solve()
```

9-11 Класс. Задачи 4-5

Разведка

Условие.

В мире, где технологии развиваются стремительными темпами, появилась новая и захватывающая вершина научной мысли — технология Квантовой Синтезации. Она предлагает возможности, которые могут кардинально преобразить промышленность, медицинские исследования и даже повседневную жизнь людей. Но с такой мощью приходит и невероятный интерес, особенно со стороны частных коммерческих организаций.

В отсутствие жёсткого контроля со стороны государства, технологии Квантовой Синтезации стали объектом острейшей конкуренции между десятками крупных корпораций и тысячами стартапов. Эти компании вступают в агрессивную гонку за лидерство, стараясь выпустить инновации первыми и завоевать рынок. В условиях жестокого капитализма не является редкостью ситуация, когда одна корпорация выкупает другую, забирая себе её наработки и специалистов.

Социалистический лагерь начинает координировать работу своих агентов и шпионов. Их миссия заключается в том, чтобы разведывать корпоративные тайны, обеспечивать свои родные страны новейшими разработками, и в перспективе — внедрять идеи социальной справедливости и сотрудничества в области технологий.

Агенты социалистического лагеря действуют скрытно: они внедрены в коммерческие компании, где занимаются сбором информации, защищая технологии от эксплуатации во вредоносных целях и стараясь поддерживать баланс в нарастающей корпоративной войне. Разведка помогает союзным странам создать свои собственные версии технологий, которые будут использоваться в интересах общества, а не только ради прибыли.

Ваша задача — дать ответы на вопросы партии исходя из получаемых от агентов данных.

Формат ввода.

В первой строке вводится число N ($3 \leq N \leq 5 * 10^5$) - число организаций, в которых работают наши агенты.
Во второй строке вводятся через пробел N чисел a_i ($1 \leq a_i \leq 10^{10}$), где a_i - уровень квалификации агента номер i.
В третьей строке вводится число Q ($1 \leq Q \leq 5 * 10^5$) - число событий, которое необходимо

обработать.
В следующих Q строках содержатся описания событий. Есть 3 вида событий:

- 1 X ($1 \leq X \leq N$). Агент X уволен, так как не справился с поставленной задачей. Гарантируется, что в этот момент агент X не уволен.
- 2 X Y ($1 \leq X, Y \leq N, X \neq Y$). Агенты X и Y доложили, что их организации объединились в одну. Гарантируется, что в этот момент агенты X и Y не уволены.
- 3 X ($1 \leq X \leq N$) . Руководство интересуется тремя лучшими агентами в организации, в которой работает агент X. Гарантируется, что в этот момент агент X не уволен. Гарантируется что хотя бы один запрос 3 типа будет.

Гарантируется, что во всем списке событий будет не более 20 увольнений.

Формат вывода.

На каждый запрос третьего вида выведите через пробел 3 числа - уровни квалификации трех лучших агентов запрошенной организации. Числа должны быть расположены по возрастанию.

Если в запрошенной организации работают 2 человека, первым числом вывести "-1". Например, в запрошенной организации работают агенты квалификации 30 и 20. Правильный ответ на такой запрос будет: "-1 20 30".

Если в запрошенной организации работает 1 человек, первыми двумя числами вывести "-1". Например, в запрошенной организации работает один агент квалификации 100. Правильный ответ на такой запрос будет: "-1 -1 100".

Пример.

Ввод	Вывод
10 5 4 5 6 10 10 2 7 8 5 9 3 1 2 1 2 3 1 2 3 4 3 3 2 2 4 3 4 1 3 3 2	-1 -1 5 -1 4 5 -1 5 6 5 5 6 4 5 6

Примечание

Пояснение к тесту 1. После каждого события скобками обозначены составы организаций, их элементы - рейтинги агентов.

Запрос 1. Вывод ответа.
Запрос 2. ([5, 4], [5], [6], [10], [10], [2], [7], [8], [5]).
Запрос 3. Вывод ответа.
Запрос 4. ([5, 4], [6, 5], [10], [10], [2], [7], [8], [5]).
Запрос 5. Вывод ответа.
Запрос 6. ([6, 5, 5, 4], [10], [10], [2], [7], [8], [5]).
Запрос 7. Вывод ответа.
Запрос 8. ([6, 5, 4], [10], [10], [2], [7], [8], [5]).
Запрос 9. Вывод ответа.

Система оценивания

Группа 1. Гарантируется, что $(1 \leq N, Q \leq 100)$. За эту группу вы получите 21 балл.
Группа 2. Гарантируется, что не будет увольнений. За эту группу вы получите 35 баллов. Будет тестироваться только при прохождении группы 1.
Группа 3. Без дополнительных ограничений. За эту группу вы получите 44 балла. Будет тестироваться только при прохождении группы 2.

Решения

Язык C++

```cpp
#include <algorithm>
#include <array>
#include <iostream>
#include <unordered_map>
#include <vector>

using namespace std;

using ll = long long;

int get_main(int who, vector<int> &parent) {
  if (who == parent[who]) {
    return who;
  }
  return parent[who] = get_main(parent[who], parent);
}
```

```
void unite_two(int le, int re, vector<int> &parent, vector<int> &priority,
               vector<array<ll, 3>> &top_3) {
  int le_main = get_main(le, parent);
  int re_main = get_main(re, parent);
  if (le_main == re_main) {
    return;
  }
  if (priority[le_main] < priority[re_main]) {
    swap(le_main, re_main);
  }
  parent[re_main] = le_main;
  priority[le_main] = max(priority[le_main], priority[re_main] + 1);
  vector<ll> new_top = {top_3[le_main][0], top_3[le_main][1],
                        top_3[le_main][2], top_3[re_main][0],
                        top_3[re_main][1], top_3[re_main][2]};
  sort(new_top.begin(), new_top.end());
  top_3[le_main][0] = new_top[3];
  top_3[le_main][1] = new_top[4];
  top_3[le_main][2] = new_top[5];
}

void remove_him(int del_person, vector<int> &parent, vector<int> &priority,
                vector<array<ll, 3>> &top_3, const vector<ll> &qual) {
  unordered_map<int, vector<int>> current_teams;
  int n = parent.size();
  for (int i = 0; i < n; ++i) {
    if (parent[i] != -1) {
      current_teams[get_main(i, parent)].push_back(i);
    }
  }
  parent[del_person] = -1;
  for (int i = 0; i < n; ++i) {
    if (parent[i] != -1) {
      parent[i] = i;
      priority[i] = 0;
      top_3[i] = {-1, -1, qual[i]};
    }
  }
  for (const auto &[_, group] : current_teams) {
    int prev_pers = -1;
    for (int chel : group) {
      if (parent[chel] != -1) {
        if (prev_pers == -1) {
          prev_pers = chel;
        } else {
```

```
          unite_two(prev_pers, chel, parent, priority, top_3);
        }
      }
    }
  }
}

vector<ll> get_top_3(int person, vector<int> &parent,
                     vector<array<ll, 3>> &top_3) {
  int person_main = get_main(person, parent);
  vector<ll> ans = {top_3[person_main][0], top_3[person_main][1],
                    top_3[person_main][2]};
  return ans;
}

void solve() {
  int n;
  cin >> n;
  vector<ll> qual(n);
  vector<int> parent(n);
  vector<int> priority(n);
  vector<array<ll, 3>> top_3(n);
  for (int i = 0; i < n; ++i) {
    cin >> qual[i];
    const ll MAX_VAL = 1e10;
    top_3[i][0] = -1;
    top_3[i][1] = -1;
    top_3[i][2] = qual[i];
    parent[i] = i;
  }
  int q;
  cin >> q;
  while (q--) {
    int t;
    cin >> t;
    if (t == 1) {
      int x;
      cin >> x;
      --x;
      remove_him(x, parent, priority, top_3, qual);
    } else if (t == 2) {
      int x, y;
      cin >> x >> y;
      --x;
      --y;
      unite_two(x, y, parent, priority, top_3);
```

```
    } else if (t == 3) {
      int x;
      cin >> x;
      --x;
      vector<ll> top_3_tmp = get_top_3(x, parent, top_3);
      for (ll el : top_3_tmp) {
        cout << el << ' ';
      }
      cout << '\n';
    }
  }
}

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

Язык Python

```
import sys
from typing import List, Tuple
from array import array

def get_main(who: int, parent: List[int]) -> int:
    assert parent[who] != -1
    if who == parent[who]:
        return who
    parent[who] = get_main(parent[who], parent)
    return parent[who]

def unite_two(le: int, re: int, parent: List[int], priority: List[int],
top_3: List[array], qual: List[int]) -> None:
    le_main = get_main(le, parent)
    re_main = get_main(re, parent)
    if le_main == re_main:
        return
    if priority[le_main] < priority[re_main]:
        le_main, re_main = re_main, le_main
    parent[re_main] = le_main
    priority[le_main] = max(priority[le_main], priority[re_main] + 1)
```

```
    new_top = [top_3[le_main][0], top_3[le_main][1], top_3[le_main][2],
top_3[re_main][0], top_3[re_main][1], top_3[re_main][2]]
    new_top.sort()
    top_3[le_main][0] = new_top[3]
    top_3[le_main][1] = new_top[4]
    top_3[le_main][2] = new_top[5]

def remove_him(del_person: int, parent: List[int], priority: List[int],
top_3: List[array], qual: List[int]) -> None:
    current_teams = {}
    n = len(parent)
    for i in range(n):
        if parent[i] != -1:
            main = get_main(i, parent)
            if main not in current_teams:
                current_teams[main] = []
            current_teams[main].append(i)
    parent[del_person] = -1
    for i in range(n):
        if parent[i] != -1:
            parent[i] = i
            priority[i] = 0
            top_3[i] = array('l', [-1, -1, qual[i]])
    for group in current_teams.values():
        prev_pers = -1
        for chel in group:
            if parent[chel] != -1:
                if prev_pers == -1:
                    prev_pers = chel
                else:
                    unite_two(prev_pers, chel, parent, priority, top_3,
qual)

def get_top_3(person: int, parent: List[int], top_3: List[array]) ->
List[int]:
    person_main = get_main(person, parent)
    return [top_3[person_main][0], top_3[person_main][1], top_3[person_main]
[2]]

def solve():
    input = sys.stdin.readline
    n = int(input())
    qual = list(map(int, input().split()))
    parent = list(range(n))
    priority = [0] * n
    top_3 = [array('l', [-1, -1, qual[i]]) for i in range(n)]
```

```
    q = int(input())
    for _ in range(q):
        READ_RES = list(map(int, input().split()))
        t = READ_RES[0]
        if t == 1:
            x = READ_RES[1] - 1
            remove_him(x, parent, priority, top_3, qual)
        elif t == 2:
            x, y = READ_RES[1], READ_RES[2]
            unite_two(x - 1, y - 1, parent, priority, top_3, qual)
        elif t == 3:
            x = READ_RES[1] - 1
            top_3_tmp = get_top_3(x, parent, top_3)
            print(' '.join(map(str, top_3_tmp)))
        else:
            assert False

if __name__ == '__main__':
    solve()
```

Строим веранду

Условие.

Эстет Геннадий решил построить веранду на своей даче. Геннадий очень любит единообразие, поэтому цвета дивана, стола и обшивки веранды должны отличаться друг от друга как можно меньше. Цвет - это некоторое число X из диапазона $(1 \leq X \leq 10^9)$.

Бюджет Геннадия ограничен, на закупку материалов у него есть N рублей. Пусть цвет выбранного дивана $c1$, стола - $c2$, веранды - $c3$.Помогите Геннадию, найдите минимальную разность цветов ($minimize(max(c1, c2, c3) - min(c1, c2, c3))$), которой можно достичь, потратив не более N рублей.

Формат ввода.

В первой строке вводятся 2 числа N, X $(1 \leq N \leq 3 * 10^5, 1 \leq X \leq 10^9)$ - количество материалов каждого вида и бюджет Геннадия.
Вторая строка содержит N чисел a_i $(1 \leq a_i \leq 10^9)$ - цвета диванов.
Третья строка содержит N чисел b_i $(1 \leq b_i \leq 10^9)$ - цены диванов.
Четвертая строка содержит N чисел c_i $(1 \leq c_i \leq 10^9)$ - цвета столов.
Пятая строка содержит N чисел d_i $(1 \leq d_i \leq 10^9)$ - цены столов.

Шестая строка содержит N чисел e_i $(1 \leq e_i \leq 10^9)$ - цвета обшивок.
Седьмая строка содержит N чисел f_i $(1 \leq f_i \leq 10^9)$ - цены обшивок.

Формат вывода.

Выведите единственное число - минимальную разность цветов дивана, стола и обшивки.
Если на данный бюджет ничего купить нельзя, выведите -1.

Пример 1

Ввод	Вывод
3 30 10 20 30 10 11 10 8 19 32 10 10 10 10 20 30 10 10 10	2

Пример 2

Ввод	Вывод
1 10 3 3 3 4 3 4	-1

Примечание

Пояснение к тесту 1. Для получения ответа 2 мы можем купить первый диван (цвет 10, цена 10); первый стол (цвет 8, цена 10); первую обшивку (цвет 10, цена 10). Этот же ответ можно получить, купив третьи стол, диван и обшивку.
Номера дивана, стола и обшивки НЕ ОБЯЗАНЫ СОВПАДАТЬ!
Если бы бюджет был 31, то оптимальным выбором были бы второй диван, второй стол и вторая обшивка, разность их цветов была бы 1.

Система оценивания

Группа 1. Гарантируется, что ($1 \leq N \leq 100$). За эту группу вы получите 8 баллов.
Группа 2. Гарантируется, что ($1 \leq N \leq 3000$). За эту группу вы получите 27 баллов. Будет тестироваться только при прохождении группы 1.
Группа 3. Без дополнительных ограничений. За эту группу вы получите 65 балла. Будет тестироваться только при прохождении группы 2.

Решения

Язык C++

```cpp
#include <algorithm>
#include <array>
#include <cassert>
#include <iostream>
#include <vector>

using namespace std;

using ll = long long;

class TMinQueue {
public:
  TMinQueue(vector<array<ll, 2>> &tmp_arr) : arr(tmp_arr), size(0) {}

  int size;

  void set_interval(int min_val, int max_val) {
    while (first_not_in < arr.size() && arr[first_not_in][0] <= max_val) {
      push_back(arr[first_not_in][1]);
      ++first_not_in;
    }
    while (first_to_remove < first_not_in &&
           arr[first_to_remove][0] < min_val) {
      pop_front();
      ++first_to_remove;
    }
  }

  int get_min() const {
    assert(size > 0);
    int ans = 1e9 + 7;
    if (front_stack.size()) {
      ans = min(front_stack.back()[1], ans);
    }
    if (back_stack.size()) {
```

```
      ans = min(back_stack.back()[1], ans);
    }
    return ans;
  }

private:
  void push_in_stack(int x, vector<array<int, 2>> &stack) {
    int new_min = x;
    if (stack.size() && stack.back()[1] < new_min) {
      new_min = stack.back()[1];
    }
    stack.push_back({x, new_min});
  }

  void push_back(int x) {
    push_in_stack(x, back_stack);
    ++size;
  }

  void pop_front() {
    if (front_stack.empty()) {
      while (back_stack.size()) {
        push_in_stack(back_stack.back()[0], front_stack);
        back_stack.pop_back();
      }
    }
    assert(front_stack.size() > 0);
    front_stack.pop_back();
    --size;
  }

private:
  vector<array<ll, 2>> &arr;
  int first_not_in = 0;
  int first_to_remove = 0;
  vector<array<int, 2>> front_stack, back_stack; // value, min
};

bool enough_if_min(int n, int X, vector<array<ll, 2>> &min_el,
                   vector<array<ll, 2>> &b, vector<array<ll, 2>> &c, int
mid) {
  TMinQueue qb = TMinQueue(b);
  TMinQueue qc = TMinQueue(c);
  for (int id = 0; id < n; ++id) {
    int max_col = min_el[id][0] + mid;
    qb.set_interval(min_el[id][0], max_col);
```

```
    qc.set_interval(min_el[id][0], max_col);
    if (qb.size && qc.size) {
      int qb_min = qb.get_min();
      int qc_min = qc.get_min();
      if (X - min_el[id][1] >= qb_min + qc_min) {
        return true;
      }
    }
  }
  return false;
}

bool enough(int n, int X, vector<array<ll, 2>> &a, vector<array<ll, 2>> &b,
            vector<array<ll, 2>> &c, int mid) {
  return enough_if_min(n, X, a, b, c, mid) ||
         enough_if_min(n, X, b, c, a, mid) || enough_if_min(n, X, c, a, b,
mid);
}

void solve() {
  int n;
  ll X;
  cin >> n >> X;
  vector<array<ll, 2>> a(n), b(n), c(n);
  vector<array<ll, 2>> best(3, {-1, X + 1}); // color, cost
  for (int i = 0; i < n; ++i) {
    cin >> a[i][0];
  }
  for (int i = 0; i < n; ++i) {
    cin >> a[i][1];
    if (a[i][1] < best[0][1]) {
      best[0] = a[i];
    }
  }
  for (int i = 0; i < n; ++i) {
    cin >> b[i][0];
  }
  for (int i = 0; i < n; ++i) {
    cin >> b[i][1];
    if (b[i][1] < best[1][1]) {
      best[1] = b[i];
    }
  }
  for (int i = 0; i < n; ++i) {
    cin >> c[i][0];
  }
```

```
  for (int i = 0; i < n; ++i) {
    cin >> c[i][1];
    if (c[i][1] < best[2][1]) {
      best[2] = c[i];
    }
  }
  if (X - best[2][1] < best[0][1] + best[1][1]) {
    cout << "-1" << endl;
    return;
  }
  sort(a.begin(), a.end());
  sort(b.begin(), b.end());
  sort(c.begin(), c.end());
  int lb = -1;
  int rb = 1e9 + 1;
  while (rb - lb > 1) {
    int mid = (lb + rb) >> 1;
    if (enough(n, X, a, b, c, mid)) {
      rb = mid;
    } else {
      lb = mid;
    }
  }
  cout << rb << endl;
}

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

Язык Python

```
import sys
from collections import deque
from typing import List, Tuple

class MinQueue:
    def __init__(self, arr: List[Tuple[int, int]]):
        self.arr = arr
        self.size = 0
        self.first_not_in = 0
```

```
        self.first_to_remove = 0
        self.front_stack = deque()
        self.back_stack = deque()

    def set_interval(self, min_val: int, max_val: int):
        while self.first_not_in < len(self.arr) and
self.arr[self.first_not_in][0] <= max_val:
            self.push_back(self.arr[self.first_not_in][1])
            self.first_not_in += 1
        while self.first_to_remove < self.first_not_in and
self.arr[self.first_to_remove][0] < min_val:
            self.pop_front()
            self.first_to_remove += 1

    def get_min(self) -> int:
        assert self.size > 0
        ans = int(1e9+7)
        if self.front_stack:
            ans = min(self.front_stack[-1][1], ans)
        if self.back_stack:
            ans = min(self.back_stack[-1][1], ans)
        return ans

    def push_in_stack(self, x: int, stack: deque):
        new_min = x
        if stack and stack[-1][1] < new_min:
            new_min = stack[-1][1]
        stack.append((x, new_min))

    def push_back(self, x: int):
        self.push_in_stack(x, self.back_stack)
        self.size += 1

    def pop_front(self):
        if not self.front_stack:
            while self.back_stack:
                val, _ = self.back_stack.pop()
                self.push_in_stack(val, self.front_stack)
        assert self.front_stack
        self.front_stack.pop()
        self.size -= 1

def enough_if_min(n, X, min_el, b, c, mid) -> bool:
    qb = MinQueue(b)
    qc = MinQueue(c)
    for id in range(n):
```

```
        max_col = min_el[id][0] + mid
        qb.set_interval(min_el[id][0], max_col)
        qc.set_interval(min_el[id][0], max_col)
        if qb.size and qc.size:
            qb_min = qb.get_min()
            qc_min = qc.get_min()
            if X - min_el[id][1] >= qb_min + qc_min:
                return True
    return False

def enough(n, X, a, b, c, mid) -> bool:
    return (enough_if_min(n, X, a, b, c, mid) or
            enough_if_min(n, X, b, c, a, mid) or
            enough_if_min(n, X, c, a, b, mid))

def solve():
    input = sys.stdin.readline
    n, X = map(int, input().split())

    a_colors = list(map(int, input().split()))
    a_prices = list(map(int, input().split()))
    b_colors = list(map(int, input().split()))
    b_prices = list(map(int, input().split()))
    c_colors = list(map(int, input().split()))
    c_prices = list(map(int, input().split()))

    a = [(a_colors[i], a_prices[i]) for i in range(n)]
    b = [(b_colors[i], b_prices[i]) for i in range(n)]
    c = [(c_colors[i], c_prices[i]) for i in range(n)]

    best_a = min(a, key=lambda x: x[1])
    best_b = min(b, key=lambda x: x[1])
    best_c = min(c, key=lambda x: x[1])

    if X - best_c[1] < best_a[1] + best_b[1]:
        print("-1")
        return

    a.sort()
    b.sort()
    c.sort()
    lb, rb = -1, int(1e9) + 1
    while rb - lb > 1:
        mid = (lb + rb) // 2
        if enough(n, X, a, b, c, mid):
            rb = mid
```

```
        else:
            lb = mid

    print(rb)

if __name__ == '__main__':
    solve()
```