Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

**Отчёт по задачам блока №4**

**По дисциплине «Алгоритмы и структуры данных»**

Группа: P3212

Выполнил: Балин А. А.

Проверил: Косяков М. С.

# Задача N

## Решение

Ошибочно было использовать в качестве модели ориентированный граф, в таком случае я так и не справился с тем, что k считается несколько раз подряд. Общая идея состоит в том, чтобы построить матрицу смежности по данным о копилках, ребро строилось, если ключ лежит в копилке. Немного порисовав, я понял, что неважно, в какую сторону смотрит ребро, главное пройти по всем вершинам, находящемся в текущей компоненте связности графа. Именно количество компонент связности и будет ответом в задаче, поэтому обходим в глубину вершины по очереди и отмечаем, что по ним прошлись. Если идти не куда, значит компонента связности пройдена целиком, можно переходить к следующей.

## Код

#include <iostream>

#include <list>

#include <vector>

using namespace std;

void put\_used(vector<vector<bool>>\* vec, int pointer, vector<bool>\* used,

int n) {

(\*used)[pointer] = true;

for (int i = 0; i < n; i++) {

if ((\*vec)[pointer][i]) {

if (!(\*used)[i]) {

put\_used(vec, i, used, n);

}

}

}

}

int main() {

int n, k;

cin >> n;

vector<vector<bool>> swinka(n);

vector<bool> is\_used(n);

for (int i = 0; i < n; ++i) {

swinka[i] = vector<bool>(n);

}

for (int i = 0; i < n; ++i) {

int temp;

cin >> temp;

temp--;

swinka[i][temp] = true;

swinka[temp][i] = true;

is\_used[i] = false;

}

k = 0;

for (int i = 0; i < n; ++i) {

if (!is\_used[i]) {

k++;

put\_used(&swinka, i, &is\_used, n);

}

}

cout << k;

}

# Задача O

## Решение

Изначально была идея искать циклы, ведь именно они могут сломать разделение, но выяснилось, что если вершина присоединяется к циклу с исходящим ребром, то все в цикле списывали с него, и это было довольно трудно реализовать программно (еще циклы могут пересекаться…). В связи с этим после долгих попыток осознать, какую задачу на графах имплементировать здесь, я пришел к удивительно простой задаче, которую мы проходили на теории графов – раскраски графа. Если раскрасить граф в два цвета получится, причём не важно в данном случае, какой цвет кого обозначает, мы сможем разделить людей на две группы. Поэтому мы красим граф, проходя в глубь него, проверяем, что нет ребер, соединяющих два одинаковых цвета – тогда нам нельзя разделить на две группы.

## Код

#include <iostream>

#include <map>

#include <set>

#include <vector>

using namespace std;

bool mark\_used(vector<vector<bool>>\* m, vector<int>\* nodes, int node, int mode,

int n) {

(\*nodes)[node] = mode;

for (int i = 0; i < n; ++i) {

if ((\*m)[node][i]) {

if ((\*nodes)[i] == 0) {

if (mode == 1) {

if (mark\_used(m, nodes, i, 2, n)) {

return true;

}

}

else {

if (mark\_used(m, nodes, i, 1, n)) {

return true;

}

}

}

else {

if ((\*nodes)[i] == (\*nodes)[node]) {

return true;

}

}

}

}

return false;

}

int main() {

int n, m;

cin >> n;

cin >> m;

vector<vector<bool>> matrix(n);

vector<int> nodes(n);

for (int i = 0; i < n; ++i) {

matrix[i] = vector<bool>(n);

for (int j = 0; j < n; j++) {

matrix[i][j] = false;

}

nodes[i] = 0;

}

for (int i = 0; i < m; ++i) {

int good, bad;

cin >> good;

cin >> bad;

good--;

bad--;

matrix[good][bad] = true;

matrix[bad][good] = true;

}

for (int i = 0; i < n; ++i) {

if (nodes[i] == 0) {

if (mark\_used(&matrix, &nodes, i, 1, n)) {

cout << "NO";

return 0;

}

}

}

cout << "YES";

return 0;

}