Министерство науки и образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский университет ИТМО

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

ЛАБОРАТОРНАЯ РАБОТА №5

По дисциплине ‹‹Введение в цифровую культуру и программирование››

Работа с графом

Выполнил Круглов Георгий Николаевич

(Фамилия Имя Отчество)

Проверила Страдина Марина Владимировна

(Фамилия Имя Отчество)

Санкт-Петербург, 2020 г.

НОМЕР ВАРИАНТА: 226

Всего рёбер: 2486
Всего изолированных вершин: 5
Их номера: 23 115 416 598 945

Наибольшая степень: 14
Номера вершин имеющих эту степень: 14
Диаметр графа: 9
Наидлиннейший кратчайший путь: 771 822 204 516 590 240 874 144 681 69

Расстояние от 281 до 608 - 4
Путь: 608 902 440 237 281
Расстояние от 953 до 600 - 4
Путь: 600 850 26 969 954
Расстояние от 777 до 584 - 3
Путь: 584 607 340 777

Количество рёбер после удаления: 2186
Количество изолированных вершин: 6
Их номера: 23 115 244 416 598 945

Наибольшая степень: 12
Номера вершин имеющих эту степень: 508
Диаметр графа: 9
Наидлиннейший кратчайший путь : 411 4 928 689 122 264 440 11 999 66

Расстояние от 281 до 608 - 5
Путь: 608 109 763 834 269 281
Расстояние от 953 до 600 - 5
Путь: 600 19 447 20 692 954
Расстояние от 777 до 584 - 6
Путь: 584 597 719 284 705 975 777

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#include <algorithm>

#define INF 1e9

using namespace std;

vector<vector<int> > graph;
vector<vector<int> > dist;

vector<pair<int, int> > degree;

vector<bool> visited;
vector<int> previous;

int dfs(int index, int length) {
    visited[index] = true;

    for (auto &destination : graph[index]) {
        if (!visited[destination])
            return dfs(destination, length + 1);
    }

    return length;
}

void bfs(int index) {
    queue<int> q;
    q.push(index);

    dist[index][index] = 0;
    previous[index] = -1;

    while (!q.empty()) {
        int observed = q.front();
        q.pop();

        for (int i = 0; i < graph[observed].size(); i++) {
            int destination = graph[observed][i];

            if (dist[index][destination] > dist[index][observed] + 1) {
                dist[index][destination] = dist[index][observed] + 1;
                previous[destination] = observed;
                q.push(destination);
            }
        }
    }
}

void way(int a, int b) {
    previous.clear();
    previous.resize(1000);

    dist.clear();
    dist.resize(1000, vector<int>(1000, INF));

    bfs(a);

    cout << dist[a][b] << '\n';
```

```
    while (b != -1) {
        cout << b << ' ';
        b = previous[b];
    }
    cout << '\n';
}

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

    freopen("data.txt", "r", stdin);

    int edgesCount = 0;

    graph.resize(1000);
    previous.resize(1000);

    for (int i = 0; i < 1000; i++) {
        degree.emplace_back(0, i);
    }

    int a;
    while (cin >> a) {
        int b;
        cin >> b;

        graph[a].push_back(b);
        graph[b].push_back(a);

        degree[a].first++;
        degree[b].first++;

        edgesCount++;
    }

    cout << "Всего рёбер: " << edgesCount << '\n';

    ///Изолированные вершины
    vector<int> isolated;

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

    for (int i = 0; i < 1000; i++) {
        if (degree[i].first == 0) {
            isolated.push_back(degree[i].second);
        }
    }

    sort(isolated.begin(), isolated.end());
    cout << "Всего изолированных вершин: " << isolated.size();
    cout << "\nИх номера: ";

    for (auto &node : isolated) {
        cout << node << ' ';
    }
    cout << '\n';


    ///Вершины с наибольшей степенью
    cout << "Наибольшая степень: " << degree.back().first;
    cout << "\nНомера вершин имеющих эту степень: ";
```

```
    for (int i = 999; i >= 0; i--) {
        if (degree[i].first == degree[999].first)
            cout << degree[i].first << ' ';
    }
    cout << '\n';


    ///Компоненты
    visited.resize(1000);

    int maxLength = 0;
    int vertex = -1;

    for (int i = 0; i < 1000; i++) {
        if (!visited[i]) {
            int length = dfs(i, 0);
            if (length > maxLength) {
                vertex = i;
                maxLength = length;
            }
        }
    }


    visited.clear();
    visited.resize(1000);
    dist.resize(1000, vector<int>(1000, INF));


    for (int i = 0; i < 1000; i++) {
        bfs(i);
    }

    int diameter = -1;
    int vertex2;

    for (int i = 0; i < 1000; i++) {
        for (int j = 0; j < 1000; j++) {
            if (dist[i][j] == INF) {
                dist[i][j] = -INF;
            }
            if (dist[i][j] > diameter) {
                diameter = dist[i][j];
                vertex = i;
                vertex2 = j;
            }
        }
    }
    cout << "Диаметр графа: " << diameter << '\n';

    way(vertex, vertex2);

    ///Вопрос 5
    way(281, 608);

    ///Вопрос 6
    way(954, 600);

    ///Вопрос 7
    way(777, 584);


    ///Удаление
```

```
    set<int> deleted;

    deleted.insert(70);
    deleted.insert(874);
    deleted.insert(237);
    deleted.insert(340);
    deleted.insert(607);

    for (auto &i : graph) {
        for (int j = 0; j < i.size(); j++) {
            int deleting = i[j];
            auto iterator = deleted.find(deleting);

            if (deleting % 17 == 0 || iterator != deleted.end()) {
                i.erase(i.begin() + j, i.begin() + j + 1);
                graph[deleting].clear();
                j--;
            }
        }
    }

    ///Пересчёт характеристик графа
    edgesCount = 0;
    degree.clear();
    for (int i = 0; i < 1000; i++) {
        degree.emplace_back(0, i);
    }
    for (int i = 0; i < 1000; i++) {
        edgesCount += graph[i].size();
        degree[i].first = graph[i].size();
    }

    cout << "Количество рёбер после удаления: " << edgesCount / 2 << '\n';

    ///Изолированные вершины
    isolated.clear();
    sort(degree.begin(), degree.end());

    for (auto &i : degree) {
        auto iterator = deleted.find(i.second);

        if (i.first == 0 && iterator == deleted.end() && i.second % 17 != 0) {
            isolated.push_back(i.second);
        }
    }

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

    cout << "Количество изолированных вершин: " << isolated.size() << '\n';

    for (auto &i : isolated)
        cout << i << ' ';
    cout << '\n';


    ///Вершины с наибольшей степенью
    cout << "Наибольшая степень: " << degree.back().first;
    cout << "\nНомера вершин имеющих эту степень:\n";
    for (auto &node : degree) {
        if (node.first == degree.back().first)
            cout << node.second << ' ';
    }
    cout << '\n';
```

```
    ///Компоненты
    visited.resize(1000);

    maxLength = 0;
    vertex = -1;

    for (int i = 0; i < 1000; i++) {
        if (!visited[i]) {
            int length = dfs(i, 0);
            if (length > maxLength) {
                vertex = i;
                maxLength = length;
            }
        }
    }

    visited.clear();
    visited.resize(1000);
    dist.resize(1000, vector<int>(1000, INF));

    for (int i = 0; i < 1000; i++) {
        bfs(i);
    }

    diameter = -1;
    vertex2 = -1;

    for (int i = 0; i < 1000; i++) {
        for (int j = 0; j < 1000; j++) {
            if (dist[i][j] == INF) {
                dist[i][j] = -INF;
            }
            if (dist[i][j] > diameter) {
                diameter = dist[i][j];
                vertex = i;
                vertex2 = j;
            }
        }
    }

    cout << "Диаметр графа: " << diameter << '\n';

    way(vertex, vertex2);

    ///Вопрос 12
    way(281, 608);

    ///Вопрос 13
    way(954, 600);

    ///Вопрос 14
    way(777, 584);
}
```