Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ 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 Фадеев Артём Владимирович
(Фамилия Имя Отчество)
Проверила Страдина Марина Владимировна
(Фамилия Имя Отчество)

Санкт-Петербург, 2020 г.

Код:

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

using namespace std;

int edge;
vector<set<int>> g;

int edges () {
    edge = 0;
    for (int i = 0; i < (int) g.size(); i++) {
        edge += (int) g[i].size();
    }
    return (edge / 2);
}

void insulators() {
    cout << '\n';
    vector<int> all;
    for (int i = 0; i < (int) g.size(); i++) {
        if (g[i].empty()) {
            all.push_back(i);
        }
    }
    cout << "-  " << (int) all.size() << "\t| ";
    for (int i = 0; i < (int)all.size(); i++) {
        cout << all[i] << " ";
    }
    cout << '\n';
}

void degree_max () {
    int max_degree = 0;
    for (int i = 0; i < (int) g.size(); i++) {
        max_degree = max(max_degree, (int) g[i].size());
    }
    cout << max_degree << '\n';
    for (int i = 0; i < (int) g.size(); i++) {
        if ((int)g[i].size() == max_degree) {
            cout << "-  " << i << "\t| ";
            for (auto j = g[i].begin(); j != g[i].end(); j++) {
                cout << *j << " ";
            }
            cout << '\n';
        }
    }
}
vector<bool> used;
vector<int> d;
vector<int> p;

int total;

void bfs (int s) {
    used[s] = true;
    queue<int> q;
    q.push(s);
    p[s] = -1;
```

```
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (auto i = g[v].begin(); i != g[v].end(); i++) {
            int to = *i;
            if (!used[to]) {
                used[to] = true;
                total++;
                q.push(to);
                d[to] = d[v] + 1;
                p[to] = v;
            }
        }
    }
}



int diameter () {
    // Counting components
    int n = (int) g.size();
    used.assign(n, false);
    d.assign(n, 0);
    p.assign(n, 0);
    vector<pair<int, int>> components;
    for (int i = 0; i < n; ++i) {
        if (!used[i]) {
            total = 1;
            bfs(i);
            components.push_back({i, total});
        }
    }
    int start = 0, score = 0;
    for (int i = 0; i < (int) components.size(); i++) {
        if (components[i].second > score) {
            start = components[i].first;
            score = components[i].second;
        }
    }
    used.assign(n, false);
    d.assign(n, 0);
    p.assign(n, 0);
    vector<int> longest;
    bfs(start);
    for (int i = 0; i < n; i++) {
        if (used[i]) {
            longest.push_back(i);
        }
    }
    int h_max = 0;
    for (int i = 0; i < (int)longest.size(); i++) {
        total = 1;
        used.assign(n, false);
        d.assign(n, 0);
        p.assign(n, 0);
        bfs(longest[i]);
        for (int j = 0; j < n; j++) {
            h_max = max(h_max, d[j]);
        }
    }
    return h_max;
}

void path (int v, int u) {
```

```
    int n = (int) g.size();
    total = 1;
    used.assign(n, false);
    d.assign(n, 0);
    p.assign(n, 0);
    bfs(v);
    if (!used[u]) {
        cout << "No path: [" << v << " : " << u << "]" << '\n';
    }
    else {
        vector<int> path;
        for (int w = u; w != -1; w = p[w]) {
            path.push_back(w);
        }
        reverse(path.begin(), path.end());
        cout << "Path: [" << v << " : " << u << "]" << '\n';
        cout << "-  " << (int) path.size() - 1 << "\t| ";
        for (int i = 0; i < (int) path.size(); i++) {
            cout << path[i] << " ";
        }
        cout << '\n';
    }
}

void delete_ (int v) {
    vector<int> elem;
    for (auto i = g[v].begin(); i != g[v].end(); i++) {
        elem.push_back(*i);
    }
    g[v].clear();
    for (int i = 0; i < (int)elem.size(); i++) {
        g[elem[i]].erase(v);
        for (int j = 0; j < (int)elem.size(); j++) {
            if (g[elem[j]].find(elem[i]) == g[elem[j]].end() && elem[i] != elem[j]) {
                g[elem[j]].insert(elem[i]);
            }
        }
    }
}

void erase_() {
    /*
        Delete all vertexes % 17
        + [224, 932, 478, 459, 13, 26, 862]
    */
    for (int v = 0; v < 1000; v += 17) {
        delete_(v);
    }
    vector<int> need_erase = {224, 932, 478, 459, 13, 26, 862};
    for (int i = 0; i < (int) need_erase.size(); i++) {
        delete_(need_erase[i]);
    }
}

void print() {
    for (int i = 0; i < (int) g.size(); i++) {
        cout << i << " | ";
        for (auto j = g[i].begin(); j != g[i].end(); j++) {
            cout << *j << " ";
        }
        cout << '\n';
    }
}
```

```
int main() {
    setlocale(LC_ALL, "Russian");
    ifstream in("graphedges240.txt");
    in.is_open();
    int x, y;
    g.resize(1000);
    while (in >> x >> y) {
        g[x].insert(y);
        g[y].insert(x);
    }

    cout << "1. Edges in graph: " << edges() << '\n';

    cout << '\n' << "2. Insulators: ";
    insulators();

    cout << '\n' << "3. Max degree of vertex is: ";
    degree_max();

    cout << '\n' << "4. Diameter of the biggest component: " << diameter() << '\n';

    int A = 455, B = 521;
    cout << '\n' << "5. ";
    path(A, B);

    cout << '\n' << "6. ";
    int C = 311, D = 638;
    path(C, D);

    cout << '\n' << "7. ";
    int E = 406, F = 681;
    path(E, F);

    erase_();

    cout << '\n' << "8. Edges in graph(2): " << edges() << '\n';

    cout << '\n' << "9. Insulators(2): ";
    insulators();

    cout << '\n' << "10. Max degree of vertex is(2): ";
    degree_max();

    cout << '\n' << "11. Diameter of the biggest component(2): " << diameter() <<
'\n';

    cout << '\n' << "12. ";
    path(A, B);

    cout << '\n' << "13. ";
    path(C, D);

    cout << '\n' << "14. ";
    path(E, F);
}
```

Ход работы:

PS.

Практически везде я использую обход графа в ширину.

Диаметр.

Диаметр графа я считаю для максимальной компаненты связности графа (с наибольшим количеством вершин). Для поиска самой компаненты запускаю “n” - bfs-ов из еще не посещённых вершин, предварительно запоминая, где я начал и какое количество вершин посетил.

Подсчет самого диаметра провожу при помощи запуска bfs-ов из всех вершин моей компаненты связности и при каждом запуске считаю максимальную глубину, которую я мог получить.

Поиск кратчайшего расстояния.

Расстояние между двумя вершинами ищу точно также: запускаю bfs из вершины, запоминая для каждой вершины её предка. Так как мы имеем вершину, в которую нужно прийти и знаем всех её предков, то восстановить путь не составит труда.

Удаление вершины.

Имеем вершину и все вершины, которые с ней связаны.

Добавим ко всем вершинам, у которых есть связь с исходной, все вершины исходной вершины, если такой вершины уже не имеется и удалим связи между исходной вершиной со всеми другими.

Сложность:

Поиск: O(logn)

Вставка: O(logn)

Удаление: O(logn)

* n – количество вершин, с которыми связана исходная вершина.

Полученный результат:

1. Edges in graph: 2427

2. Insulators:

- 12 | 84 114 181 213 228 257 299 395 438 465 498 700

3. Max degree of vertex is: 12

- 224 | 154 298 448 565 587 610 705 787 937 948 956 979

- 712 | 39 58 131 195 326 372 405 495 526 553 727 900

4. Diameter of the biggest component: 9

5. Path: [455 : 521]

- 4 | 455 223 13 478 521

6. Path: [311 : 638]

- 5 | 311 63 189 862 977 638

7. Path: [406 : 681]

- 5 | 406 26 459 869 932 681

8. Edges in graph(2): 2871

9. Insulators(2):

- 77 | 0 13 17 26 34 51 68 84 85 102 114 119 136 153 170 181 187 204 213 221 224 228 238 255 257 272 289 299 306 323 340 357374 391 395 408 425 438 442 459 465 476 478 493 498 510 527 544 561 578 595 612 629 646 663 680 697 700 714 731 748 765 782 799 816 833 850 862 867 884 901 918 932 935 952 969 986

10. Max degree of vertex is(2): 25

- 948 | 92 106 135 154 165 298 308 406 448 461 489 565 581 587 602 610 638 705 787 869 877 937 956 965 979

11. Diameter of the biggest component(2): 8

12. Path: [455 : 521]

- 2 | 455 223 521

13. Path: [311 : 638]

- 4 | 311 1 554 135 638

14. Path: [406 : 681]

- 2 | 406 869 681