До удаления:

1. Кол-во ребер = 2488
2. Кол-во изолятов = 6
Изоляты: 104 326 339 609 757 770
3. Максимальная степень вершины = 13
Вершины с этой степенью: 469 540
4. Диаметр самой большой компоненты связности = 10
5. Длина кратчайшего пути от A(705) до B(823) = 4
Путь - 705 42 495 22 823
6. Длина кратчайшего пути от C(253) до D(649) = 4
Путь - 253 655 389 190 649
7. Длина кратчайшего пути от E(396) до F(90) = 5
Путь - 396 828 591 510 327 90

После удаления:

8. Кол-во ребер = 2287
9. Кол-во изолятов = 6
Изоляты: 104 326 339 609 757 770
10. Максимальная степень вершины = 12
Вершины с этой степенью: 195 256 800
11. Диаметр самой большой компоненты связности = 10
12. Длина кратчайшего пути от A(705) до B(823) = 5
Путь - 705 86 787 122 874 823
13. Длина кратчайшего пути от C(253) до D(649) = 5
Путь - 253 427 560 55 256 649
14. Длина кратчайшего пути от E(396) до F(90) = 6
Путь - 396 828 725 195 224 327 90

Программа на c++

```cpp
#include <bits/stdc++.h>
using namespace std;

///Немного глобальных переменных
int n = 1000, m = 0;
vector<vector<int>> G(n);
vector<int> Deleted(n);

///Объявление функций
int bfs(vector<vector<int>>& Colors, vector<int>& Visited, int x);
pair<int, int> bfs2(int x);
vector<int> bfs3(int x, int f);

///Наша основная функция, которую мы сначала запустим для полного графа, затем
```

```
для удаленного
void solve(int task = 0) {
    ///m - количество ребер
    cout << 1 + task << ". Кол-во ребер = " << m << endl;

    ///Если вершину еще не удалили и из нее не исходит ребер, то это изолят.
    ///Вместе с этим найдем максимальную степень вершины
    vector<int> Isolates;
    int maxN = 0;
    for (int i = 0; i < n; i++) {
        if (Deleted[i]) continue;
        if (G[i].empty()) {
            Isolates.push_back(i);
        }
        maxN = max(maxN, int(G[i].size()));
    }
    cout << 2 + task << ". Кол-во изолятов = " << Isolates.size() << endl <<
"Изоляты:\n";
    sort(Isolates.begin(), Isolates.end());
    for (auto isolate : Isolates) {
        cout << isolate << " ";
    } cout << endl;

    ///Если вершину еще не удалили, а ее степень равна максимальной, то мы
должны будем ее вывести
    vector<int> MaxNodes;
    for (int i = 0; i < n; i++) {
        if (Deleted[i]) continue;
        if (G[i].size() == maxN) {
            MaxNodes.push_back(i);
        }
    }
    cout << 3 + task << ". Максимальная степень вершины = " << maxN << endl <<
"Вершины с этой степенью:\n";
    for (auto node : MaxNodes) {
        cout << node << " ";
    } cout << endl;

    ///Заведем массив массивов Colors. В Colors[i] будут находиться все
вершины из компоненты связности под номером i
    vector<vector<int>> Colors;
    vector<int> Visited(n);
    int maxC = 0;
    ///Если вершину еще не удалили и не посетили, то найдем ее компоненту
связности
    for (int i = 0; i < n; i++) {
        if (Deleted[i]) continue;
        if (!Visited[i]) {
            int ans = bfs(Colors, Visited, i);
            if (ans > Colors[maxC].size()) {
                maxC = Colors.size() - 1;
            }
        }
    }
    ///Найдем диаметр самой большой(по кол-ву вершин в ней) компоненты
```

```
связности
    cout << 4 + task << ". Диаметр компоненты связности = " <<
bfs2(bfs2(Colors[maxC][0]).first).second << endl;

    int A = 705, B = 823, C = 253, D = 649, E = 396, F = 90;

    ///Находим и выводим пути
    vector<int> Path = bfs3(A, B);
    if (Path.empty()) {
        cout << 5 + task << ". Пути между A(" << A << ") и B(" << B << ") не
существует" << endl;
    } else {
        cout << 5 + task << ". Длина кратчайшего пути от A(" << A << ") до B("
<< B << ") = " << Path.size() - 1 << endl;
        cout << "Путь - ";
        for (auto x : Path) {
            cout << x << " ";
        } cout << endl;
    }

    Path = bfs3(C, D);
    if (Path.empty()) {
        cout << 6 + task << ". Пути между C(" << C << ") и D(" << D << ") не
существует" << endl;
    } else {
        cout << 6 + task << ". Длина кратчайшего пути от C(" << C << ") до D("
<< D << ") = " << Path.size() - 1 << endl;
        cout << "Путь - ";
        for (auto x : Path) {
            cout << x << " ";
        }
        cout << endl;
    }

    Path = bfs3(E, F);
    if (Path.empty()) {
        cout << 7 + task << ". Пути между E(" << E << ") и F(" << F << ") не
существует" << endl;
    } else {
        cout << 7 + task << ". Длина кратчайшего пути от E(" << E << ") до F("
<< F << ") = " << Path.size() - 1 << endl;
        cout << "Путь - ";
        for (auto x : Path) {
            cout << x << " ";
        }
        cout << endl;
    }
}


signed main()
{
    int x, y;
    ifstream fin ("/home/vadim/CLionProjects/test/graphedges224.txt");
```

```
    ///Считываем ребра и запускаем solve()
    while (fin >> x >> y) {
        G[x].push_back(y);
        G[y].push_back(x);
        m++;
    }
    solve();


    ///Удаляем вершины кратные 17 и те, что нам дали в файле. Запускаем
solve()
    int remainder = 0;
    for (int i = 0; i < n; i += 17) {
        Deleted[i] = 1;
        m -= (G[i].size() >> 1) + remainder;
        remainder ^= G[i].size() & 1;
    }
    for (auto x : {389, 42, 655, 469, 22, 540, 510}) {
        Deleted[x] = 1;
        m -= (G[x].size() >> 1) + remainder;
        remainder ^= G[x].size() & 1;
    }
    solve(7);
}


///Функция для поиска пути из вершины s в вершину f
vector<int> bfs3(int s, int f) {
    int x = s;
    deque<int> O = {x};
    vector<int> Dist(n);
    Dist[x] = 1;
    ///Обычный bfs
    while (!O.empty()) {
        x = O.front();
        O.pop_front();
        for (auto y : G[x]) {
            if (Deleted[y]) continue;
            if (!Dist[y]) {
                Dist[y] = Dist[x] + 1;
                O.push_back(y);
            }
        }
        if (Dist[f]) {
            break;
        }
    }
    ///Если пути нет, то возвращаем пустой вектор, иначе восстанавливаем путь
с вершины f до s
    if (!Dist[f]) {
        return {};
    }
    vector<int> Path = {f};
    while (f != s) {
```

```
        for (auto y : G[f]) {
            if (Dist[y] == Dist[f] - 1) {
                f = y;
                break;
            }
        }
        Path.push_back(f);
    }
    reverse(Path.begin(), Path.end());
    return Path;
}

///Функция для поиска дальнейшей вершины и расстояния до нее от вершины x
pair<int, int> bfs2(int x) {
    deque<int> O = {x};
    vector<int> Dist(n);
    Dist[x] = 1;
    ///Обычный bfs
    while (!O.empty()) {
        x = O.front();
        O.pop_front();
        for (auto y : G[x]) {
            if (Deleted[y]) continue;
            if (!Dist[y]) {
                Dist[y] = Dist[x] + 1;
                O.push_back(y);
            }
        }
    }
    return {x, Dist[x]};
}

///Функция для нахождения компоненты связности, в которой находится вершина x
int bfs(vector<vector<int>>& Colors, vector<int>& Visited, int x) {
    deque<int> O = {x};
    ///Создаем пустую компоненту связности
    Colors.emplace_back();
    Visited[x] = 1;
    ///Обычный bfs
    while (!O.empty()) {
        x = O.front();
        O.pop_front();
        for (auto y : G[x]) {
            if (Deleted[y]) continue;
            if (!Visited[y]) {
                Visited[y] = 1;
                O.push_back(y);
                Colors.back().push_back(y);
            }
        }
    }
    return Colors.back().size();
}
```