Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3231

Савон Г.К.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2021

Задача M «Цивилизация»

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

vector<vector<int> > card;
vector<vector<int> > dist;
int n, m, x0, y0, x1, y1;
set<vector<int> > will_be_next;

void put_vec_for_set(int i, int j, int now_dist, int direction)
{
    if (i >= 0 && i < n && j >= 0 && j < m && dist[i][j] == -1) {
        if (card[i][j] < 3) {
            vector<int> vec;
            vec.push_back(now_dist + card[i][j]);
            vec.push_back(i);
            vec.push_back(j);
            will_be_next.insert(vec);
            dist[i][j] = vec[0];
            card[i][j] = direction;
        }
    }
}

void add_all(int i, int j, int now_dist)
{
    put_vec_for_set(i, j - 1, now_dist, -4);
    put_vec_for_set(i, j + 1, now_dist, -2);
    put_vec_for_set(i + 1, j, now_dist, -3);
    put_vec_for_set(i - 1, j, now_dist, -1);
}

void do_u_de_way(int i, int j)
{
    if (i != x0 || j != y0) {
        if (card[i][j] == -1) {
            do_u_de_way(i + 1, j);
            cout << "N";
        }
        if (card[i][j] == -2) {
            do_u_de_way(i, j - 1);
            cout << "E";
        }
        if (card[i][j] == -3) {
            do_u_de_way(i - 1, j);
            cout << "S";
        }
        if (card[i][j] == -4) {
            do_u_de_way(i, j + 1);
            cout << "W";
        }
    }
```

```
}

int main()
{
    char c;
    cin >> n >> m >> x0 >> y0 >> x1 >> y1;
    x0--;
    y0--;
    x1--;
    y1--;
    card.resize(n);
    dist.resize(m);
    for (int i = 0; i < n; i++) {
        card[i].resize(m);
        dist[i].resize(m);
        for (int j = 0; j < m; j++) {
            dist[i][j] = -1;
            cin >> c;
            if (c == '.')
                card[i][j] = 1;
            if (c == 'W')
                card[i][j] = 2;
            if (c == '#')
                card[i][j] = 3;
        }
    }
    add_all(x0, y0, 0);
    vector<int> now_vec;
    while (dist[x1][y1] == -1 && will_be_next.size() > 0) {
        now_vec = *will_be_next.begin();
        will_be_next.erase(now_vec);
        add_all(now_vec[1], now_vec[2], now_vec[0]);
    }
    cout << dist[x1][y1] << endl;
    do_u_de_way(x1, y1);
    return 0;
}
```

Пояснение к примененному алгоритму:

Для каждой из клеток (ну или почти для каждой мы выясняем, сколько до нее минимально идти). Рассматриваем все ребра, выходящие из уже достигнутых клеток, если они ведут не в болото или аут добавляем в сет ребер, указывая расстояние, которое получит клетка на конце ребра. Соответственно будем брать наименьшие значения из начала сета.

Ну и для восстановления пути надо еще запоминать в каждой клетке откуда же это в нее такой крутой маршрут построили.

Сложность: O(mlog(n)), где m-количество вершин, n-количество ребер(в задаче эти буквы другое означают просто)

Задача N «Свинки-копилки»

```
#include <iostream>
#include <set>
#include <vector>

using namespace std;

int n, parent_num, ans;
set<int> not_used_points;
set<int> now_circle;
vector<int> graph;

void find_circle(int point_num)
{
    now_circle.insert(point_num);
    if (not_used_points.find(graph[point_num]) != not_used_points.end()) {
        if (now_circle.find(graph[point_num]) != now_circle.end()) {
            ans++;
        }
        else {
            find_circle(graph[point_num]);
        }
    }
    not_used_points.erase(point_num);
}

int main()
{
    ans = 0;
    cin >> n;
    graph.resize(n);
    for (int i = 0; i < n; i++) {
        not_used_points.insert(i);
        cin >> parent_num;
        graph[i] = parent_num - 1;
    }
    while (not_used_points.size() > 0) {
        now_circle.clear();
        find_circle(*not_used_points.begin());
    }
    cout << ans;
    return 0;
}
```

Пояснение к примененному алгоритму:

По факту нам нужно найти количество циклов в графе.

В каждой итерации я смотрю еще нетронутую вершину и плавно возвращаюсь к истокам (к родителю, родителю родителя и тд).

Поскольку каждая свинья имеет родителя(вероятно даже самого себя, то когда-нибудь этот цикл замкнется)

-в цикле всегда можно разбить любую свинью и открыть всех остальных.

-если же ключ от свиньи лежит в ней самой, то очевидно – это тоже цикл, ее придется разбить.

Сложность: O(nlog(n))

Задача O «Долой списывание!»

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int m, n;
vector<vector<int> > graph;
vector<int> graph_color;
set<int> not_used_points;

bool make_it_color(int point, int color)
{
    if (not_used_points.find(point) == not_used_points.end()) {
        if (graph_color[point] != color) {
            return false;
        }
        return true;
    }
    else {
        not_used_points.erase(point);
        graph_color[point] = color;
        for (int i = 0; i < graph[point].size(); i++) {
            if (!make_it_color(graph[point][i], color * (-1))) {
                return false;
            }
        }
        return true;
    }
}

int main()
{
    cin >> n >> m;
    graph.resize(n);
    graph_color.resize(n);
    int men1, men2;
    for (int i = 0; i < m; i++) {
        cin >> men1 >> men2;
        men1--;
        men2--;
        graph[men1].push_back(men2);
        graph[men2].push_back(men1);
    }
    for (int i = 0; i < n; i++) {
        not_used_points.insert(i);
```

```
            graph_color[i] = 0;
        }
        while (not_used_points.size() > 0) {
            if (!make_it_color(*not_used_points.begin(), 1)) {
                cout << "NO";
                return 0;
            }
        }
        cout << "YES";
        return 0;
}
```

Пояснение к примененному алгоритму:

Забавно, что в задаче какбы и не выясняется, кто списывал, а кто этому способствовал, но ладно. Предполагается, что ученики делятся только на условно плохих или только на условно хороших. То есть каждый конкретный всегда либо списывал, либо давал списывать. Любой нечетный цикл специфических взаимоотношений школьников плох потому что кому-то придется быть и тем и другим. Пойдем по графу от любой вершины в ширину и будем поочередно распределять школьников «плохой, хороший». Если какой-то из них уже попадался ранее, проверим, относился ли он к той же группе. Если компонент в графе несколько – пройдемся по каждой из них.

Сложность: (m+n)

Задача P «Авиаперелеты»

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int n, litres, maxx, max_len;
vector<vector<int> > card;
set<int> used;
set<vector<int> > ribs; //length, i, j;
vector<int> shortest_point_rib;

int find_min(int now_point, int direction)
{
    used.insert(now_point);
    vector<int> vec;
    for (int i = 0; i < n; i++) {
        if (used.find(i) == used.end()) {
            vec.clear();
            if (direction == 1)
                vec.push_back(card[now_point][i]);
            else
```

```
                vec.push_back(card[i][now_point]);
            vec.push_back(i);
            int len = vec[0];
            if (shortest_point_rib[i] > len)
                ribs.insert(vec);
            if (shortest_point_rib[i] < maxx) {
                vec.clear();
                vec.push_back(shortest_point_rib[i]);
                vec.push_back(i);
                ribs.erase(vec);
                shortest_point_rib[i] = len;
            }
        }
    }
    if (used.size() == n)
        return -1;
    int new_one_point = (*ribs.begin())[1];
    int result = (*ribs.begin())[0];
    ribs.erase(ribs.begin());
    int answer = find_min(new_one_point, direction);
    if (answer < 0)
        return (result);
    return max(result, answer);
}

int main()
{
    cout.tie(0);
    iostream::sync_with_stdio(0);
    cin >> n;
    maxx = 1000000000;
    max_len = 0;
    card.resize(n);
    shortest_point_rib.resize(n);
    if (n == 1) {
        cout << 0;
        return 0;
    }
    for (int i = 0; i < n; i++) {
        card[i].resize(n);
        shortest_point_rib[i] = maxx;
        for (int j = 0; j < n; j++) {
            cin >> litres;
            if (litres > max_len)
                max_len = litres;
            card[i][j] = litres;
        }
    }
    int res1, res2;
    res1 = find_min(0, 1);
    ribs.clear();
    used.clear();
    if (res1 == max_len) {
        cout << res1;
        return 0;
    }
    for (int i = 0; i < n; i++) {
```

```
            shortest_point_rib[i] = maxx;
        }
        res2 = find_min(0, 0);
        cout << max(res1, res2);
        return 0;
}
```

Пояснение к примененному алгоритму:

По факту нам нужно из имеющихся ребер выбрать такие, чтобы граф остался связан в обе из сторон и чтобы эти ребра были как можно короче. Возьмем любую из вершин, мы обязаны иметь возможность влететь в нее и вылететь, для всех остальных вершин запишем минимальные ребра, которые выходят из уже присоединенных вершин в них, и обратно. По сути, тот же алгоритм Дейкстры, только мы смотрим не расстояние, а сами длины ребер.

Сложность: O(n^2log(n))