Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P3231

Савон Г.К.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2021

Задача 1162 «Мы будем делать бизнес, мы будем делать бабки»

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

using namespace std;

int n, m, s;
double a, b, v, rab, rba, cab, cba;
vector<vector<float> > ribs;
vector<float> dist;

int main()
{
    cin >> n >> m >> s >> v;
    s--;
    ribs.resize(2 * m);
    dist.resize(n);
    for (int i = 0; i < 2 * m; i = i + 2) {
        cin >> a >> b >> rab >> cab >> rba >> cba;
        a--;
        b--;
        ribs[i].push_back(a);
        ribs[i].push_back(b);
        ribs[i].push_back(rab);
        ribs[i].push_back(cab);
        ribs[i + 1].push_back(b);
        ribs[i + 1].push_back(a);
        ribs[i + 1].push_back(rba);
        ribs[i + 1].push_back(cba);
    }
    for (int j = 0; j < n; j++) {
        dist[j] = 0;
    }
    dist[s] = v;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < ribs.size(); j++) {
            dist[ribs[j][1]] = max(dist[ribs[j][1]], (dist[ribs[j][0]] - ribs[j][3]) *
ribs[j][2]);
        }
    }
    for (int i = 0; i < ribs.size(); i++) {
        if ((dist[ribs[i][0]] - ribs[i][3]) * ribs[i][2] > dist[ribs[i][1]] + 0.000001) {
            cout << "YES";
            return 0;
        }
    }
    cout << "NO";
    return 0;
}
```

Пояснение к примененному алгоритму:

Важно понять, что вершинами в решении этой задачи будут валюты, а ребрами – пункты обмена. И нам нужно соответственно, выбирать из имеющихся как можно более выгодные пункты обмена, то есть получать в вершине пройдя по выбранному ребру как можно больше денег в данной валюте. Нам надо найти цикл, в котором будет расти значение в исходной волюте. И поскольку мы ищем какбы наибольший путь, то это аналогично задаче по поиску кратчайшего пути, когда есть отрицательный цикл. Для такой задачи используется алгоритм Форда-Беллмана, соответственно он подойдет и нам. На каждой итерации будем пытаться по средствам каждого из имеющихся ребер улучшить значение на втором конце этих ребер.

Сложность: O(nm)

Задача 1806 «Мобильный телеграф»

```
#include <iostream>
#include <vector>
#include <set>
#include <unordered_map>
#include <queue>
#include <stdint.h>
#include <math.h>

using namespace std;

int n, k, inf;
string s;
vector<int> koeff;
unordered_map<long long, int> numbers;
vector<vector<int> > numbers_vec;
vector<int> dist;
set<int> not_used;
vector<vector<int> > parent;
set<vector<int> > que;
vector<vector<int> > now_vec;
vector<int> vec;
vector<vector<int> > res;
vector<int> find_vec;

long long make_num(vector<int> vec)
{
    long long res = 0;
    for (int i = 0; i < 10; i++) {
        res = res * 10 + vec[i];
    }
    return res;
}

vector<vector<int> > make_connections(int num)
{ //weight, number
    res.clear();
    int swichch;
    int find_num;
```

```
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            if (j != numbers_vec[num][i]) {
                find_vec = numbers_vec[num];
                find_vec[i] = j;
                long long number = (make_num(find_vec));

                if (numbers.find(number) != numbers.end()) {
                    find_num = numbers.at(number);
                    if (not_used.find(find_num) != not_used.end()) {
                        vec.clear();
                        vec.push_back(numbers.at(number));
                        vec.push_back(koeff[i]);
                        res.push_back(vec);
                    }
                }
            }
        }
        for (int j = i + 1; j < 10; j++) {
            find_vec.clear();
            find_vec = numbers_vec[num];
            swichch = find_vec[i];
            find_vec[i] = find_vec[j];
            find_vec[j] = swichch;
            long long number = make_num(find_vec);
            if (numbers.find(number) != numbers.end()) {
                find_num = numbers.at(number);
                if (not_used.find(find_num) != not_used.end()) {
                    vec.clear();
                    vec.push_back(numbers.at(number));
                    vec.push_back(koeff[i]);
                    res.push_back(vec);
                }
            }
        }
    }
    return res;
}

int dextra(int num)
{
    while (!que.empty()) {
        num = (*que.begin())[1];
        not_used.erase(num);
        que.erase(que.begin());
        if (num == n - 1)
            return dist[num];
        now_vec = make_connections(num);
        for (int i = 0; i < now_vec.size(); i++) {
            if (dist[now_vec[i][0]] > dist[num] + now_vec[i][1]) {
                vec.clear();
                vec.push_back(dist[now_vec[i][0]]);
                vec.push_back(now_vec[i][0]);
                que.erase(vec);
                dist[now_vec[i][0]] = dist[num] + now_vec[i][1];
                parent[now_vec[i][0]][0] = num;
                parent[now_vec[i][0]][1] = parent[num][1] + 1;
```

```
                vec[0] = dist[now_vec[i][0]];
                que.insert(vec);
            }
        }
    }
    return -1;
}

void print_parents(int num)
{
    if (num != 0) {
        print_parents(parent[num][0]);
        cout << parent[num][0] + 1 << " ";
    }
}

int main()
{
    cout.tie(0);
    iostream::sync_with_stdio(0);
    cin >> n;
    inf = INT32_MAX;
    dist.resize(n);
    parent.resize(n);
    vector<int> vec;
    numbers_vec.resize(n);
    for (int i = 0; i < 10; i++) {
        cin >> k;
        koeff.push_back(k);
    }
    for (int i = 0; i < n; i++) {
        vec.clear();
        dist[i] = inf;
        not_used.insert(i);
        cin >> s;
        for (int j = 0; j < 10; j++)
            vec.push_back((s[j]) - '0');
        numbers_vec[i] = vec;
        numbers[make_num(vec)] = i;
        parent[i].resize(2);
        parent[i][0] = -1;
        parent[i][1] = 1;
    }
    dist[0] = 0;
    que.clear();
    vec.clear();
    vec.push_back(0);
    vec.push_back(0);
    que.insert(vec);
    int ans = dextra(0);
    cout << ans << endl;
    if (ans != -1) {
        cout << parent[n - 1][1] << endl;
        print_parents(n - 1);
        cout << n;
    }
    return 0;
```

```
}
```

Пояснение к примененному алгоритму:

Логика решения очень проста:

1) На основании номеров строим взвешенный граф
2) Находим в этом графе кратчайший путь (алгоритмом Дейкстры)

А вот с реализацией было больно да. Оказалось, что если хранить весь граф – память страдает, но к счастью для Дейкстры достаточно связей текущей вершины. А из-за определения связей страдало время выполнения (как же долго искать номераааааа). Но сдаваться нелья, зато можно сначала перестать сравнивать все номера попарно и начать искать все за БОЛЬШУЮ константу, расстроиться, что это тоже долго и переделать хранилище этих номеров под unordered и радоваться, что наконец-то все прокатило.

Сложность: O(mlog(n))