Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P3233
Богатов Александр Сергеевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2022

Задача 1450: Российские газопроводы

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

int min_dist = -2147483647;

class Wire {
    public:
        int from, to, benefit;

        Wire(int from, int to, int benefit) {
            this->from = from;
            this->to = to;
            this->benefit = benefit;
        }

        Wire() {

        }
};

int main()
{
    int n,m,s,f;

    cin >> n >> m;

    Wire* wire_array[m];

    for (int i = 0; i < m; i++) {
            int a,b,c;
            cin >> a >> b >> c;
            wire_array[i] = new Wire(a,b,c);
    }

    cin >> s >> f;

    vector<int> path_dijkstra(n+1, min_dist);

    path_dijkstra[s] = 0;
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (path_dijkstra[wire_array[j]->from] != min_dist &&
path_dijkstra[wire_array[j]->to] < wire_array[j]->benefit +
path_dijkstra[wire_array[j]->from]) {
                path_dijkstra[wire_array[j]->to] =
path_dijkstra[wire_array[j]->from] + wire_array[j]->benefit;
            }
        }
    }

    if (path_dijkstra[f] > min_dist)
        cout << path_dijkstra[f] << endl;
    else cout << "No solution";
```

```
    return 0;
}
```

Пояснение к примененному алгоритму:

Задача решается модифицированным алгоритмом Дейкстры, в котором мы ищем не минимальные пути, а максимальные.

Задача 1806: Мобильные телеграфы

```
#include <iostream>
#include <queue>
#include <vector>
#include <set>
#include <map>
#include <cmath>
#include <list>
#include <algorithm>
using namespace std;

int limit_prefix = 10;
int n;

class Telegraph
{
	public:
		int cost, idx;

	Telegraph()
	{
		this->cost = 0;
		this->idx = 0;
	}

	Telegraph(int cost, int idx)
	{
		this->cost = cost;
		this->idx = idx;
	}
};

class compare
{
	public:
		bool operator()(Telegraph *a, Telegraph *b)
		{
			return a->cost > b->cost;
		}
};

int common_prefix_length(int64_t idx1, int64_t idx2)
{
	int len = limit_prefix;
	while (idx1 != idx2)
	{
		idx1 /= limit_prefix;
		idx2 /= limit_prefix;
```

```
                len--;
        }

        return len;
}

void dijkstra(multiset<Telegraph*> &graph, vector<Telegraph*> &vmap,
priority_queue< Telegraph *, vector< Telegraph*>, compare> &ord,
Telegraph *current)
{
        for (Telegraph* tg : graph)
        {
                int weight = current->cost + tg->cost;
                if (vmap[tg->idx]->cost == -1 || vmap[tg->idx]->cost >
weight)
                {
                        ord.push(new Telegraph(weight, tg->idx));
                        vmap[tg->idx]->cost = weight;
                        vmap[tg->idx]->idx = current->idx;
                }
        }
}

vector<Telegraph*> build_graph(vector<int64_t> &tgphs, map<int64_t,
int> &connections, priority_queue< Telegraph *, vector< Telegraph*>,
compare> &ord, vector< int > &speed)
{
        vector<Telegraph*> vmap(n);
        for (int i = 0; i < n; i++)
        {
                vmap[i] = new Telegraph(-1, -1);
        }

        ord.push(new Telegraph(0, 0));
        while (!ord.empty())
        {
                Telegraph *current = ord.top();
                ord.pop();
                if (current->idx == n - 1)
                        break;
                multiset<Telegraph*> graph;
                int64_t x = tgphs[current->idx];
                int64_t replaced_x, test1, test2;
                for (int i = 0; i < limit_prefix; i++)
                {
                        test1 = (int64_t) pow(10, i);
                        int prefix_el = x / test1 % 10;
                        for (int j = 0; j < limit_prefix; j++)
                        {
                                replaced_x = x - prefix_el *test1 + j *
test1;
                                auto iter = connections.find(replaced_x);
                                if (iter != connections.end() && x !=
replaced_x)
                                {
                                        int pref_len =
common_prefix_length(x, replaced_x);
```

```
                                        graph.insert(new
Telegraph(speed[pref_len], iter->second));
                                }
                        }

                        for (int j = i + 1; j < limit_prefix; j++)
                        {
                                test2 = (int64_t) pow(10, j);
                                int prefix_el_2 = x / test2 % 10;
                                replaced_x = x - prefix_el *test1 +
prefix_el_2 *test1 - prefix_el_2 *test2 + prefix_el * test2;
                                auto iter = connections.find(replaced_x);
                                if (iter != connections.end() && x !=
replaced_x)
                                {
                                        int pref_len =
common_prefix_length(x, replaced_x);
                                        graph.insert(new
Telegraph(speed[pref_len], iter->second));
                                }
                        }
                }

                dijkstra(graph, vmap, ord, current);
        }

        return vmap;
}

int main()
{
        vector<int> speed(limit_prefix);
        cin >> n;
        for (int i = 0; i < limit_prefix; i++)
        {
                cin >> speed[i];
        }

        vector<int64_t> tgphs(n);
        map<int64_t, int> connections;
        priority_queue<Telegraph*, vector<Telegraph*>, compare> ord;
        for (int i = 0; i < n; i++)
        {
                cin >> tgphs[i];
                connections[tgphs[i]] = i;
        }

        vector<Telegraph*> vmap = build_graph(tgphs, connections, ord,
speed);


        cout << vmap[n - 1]->cost << endl;
        if (vmap[n - 1]->cost == -1)
                return 0;

        vector<int> path;

        path.push_back(n);
```

```
        int current = n - 1;

        while (current > 0)
        {
               path.push_back(vmap[current]->idx + 1);
               current = vmap[current]->idx;
        }

        cout << path.size() << endl;

        reverse(path.begin(), path.end());

    for (int i = 0 ; i < path.size(); i++) {
        cout << path[i] << " ";
    }
        return 0;
}
```

Пояснение к примененному алгоритму:

На основе вычисления различных комбинаций префиксов мы ищем самые выгодные по весу и на основе этого создаем неориентированный граф. На каждом этапе построения графа используем алгоритм Дейкстры для поиска кратчайшего по времени маршрута и записываем его, при наличии.