

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра САПР**

**ОТЧЕТ**  
**по практической работе №3**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Алгоритмы на графах**

Студент гр. 8302

Халитов Ю.Р.

Преподаватель

Тутуева А.В.

Санкт-Петербург

2020

## Краткое описание реализуемого алгоритма и используемых структур данных

Алгоритм Флойда–Уоршелла — алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе. Краткое описание шагов алгоритма:

1. Перенумеровать вершины графа от 1 до  $N$  целыми числами, определить матрицу  $D$ , каждый элемент  $d_{ij}$  которой есть длина кратчайшей дуги между вершинами  $i$  и  $j$ . Если такой дуги нет, положить значение элемента равным  $\infty$  (в случае компьютерных вычислений возьмем какое-нибудь очень большое число). Кроме того, положить значения диагонального элемента  $d_{i,i}$  равным 0.
2. Для целого  $m$ , последовательно принимающего значения  $1 \dots N$  определить по элементам матрицы  $D^{m-1}$  элементы  $D^m$
3. Алгоритм заканчивается получением матрицы всех кратчайших путей  $D^N$ , где  $N$  — число вершин графа.

Описание реализуемых структуры данных:

class **Graph** – класс для хранения графа; граф представлен матрицей смежности.

## Оценки временной сложности реализуемых алгоритмов

Таблица 1 – Оценки временной сложности алгоритмов

Метод	Сложность
<code>void Floyd();</code>	$O(n^3)$
<code>vector&lt;string&gt; route(string, string);</code>	$O(n)$ (без учета сопоставления названию города его индекса, сложность которого $O(\log(n))$ )
<code>int cost(string, string);</code>	$O(1)$ ( $O(\log(n))$ , если учитывать, что чтобы сопоставить названию города индекс, нужно воспользоваться поиском в словаре)

$n$  – количество узлов в графе

## Примеры работы программы

1. data0.csv:

Saint-Petersburg;Moscow;10;20

Moscow;Habarovsk;40;35

Saint-Petersburg;Habarovsk;14;N/A

Vladivostok;Habarovsk;13;8

Vladivostok;Saint-Petersburg;N/A;40

```
Enter csv file name: data0.csv
Cities:
Habarovsk
Moscow
Saint-Petersburg
Vladivostok
Departure city: Moscow
Arrival city: Habarovsk
Route: Moscow Saint-Petersburg Habarovsk
Cost: 34
```

2. data1.csv:

London;Prague;3;N/A

Moscow;Kiev;21;14

Prague;Kiev;23;20

Kiev;London;13;N/A

```
Enter csv file name: data1.csv
Cities:
Kiev
London
Moscow
Prague
Departure city: London
Arrival city: Kiev
Route: London Prague Kiev
Cost: 26
```

## Листинг

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <map>
#include <string>
#include <algorithm>

class Graph
{
    std::vector<std::vector<int>> matrix;
    std::vector<std::vector<int>> next;
    std::map<std::string, int> cities_map;
    std::vector<std::string> cities;
    int size;
public:
    Graph(std::vector<std::vector<std::string>>);
    void Floyd();
    std::vector<std::string> route(std::string, std::string);
    int cost(std::string, std::string);
    std::vector<std::vector<int>> get_matrix();
    std::vector<std::string> get_cities();
    std::map<std::string, int> get_cities_map();
};

Graph::Graph(std::vector<std::vector<std::string>> rows)
{
    for (auto row : rows) cities_map[row[0]] = cities_map[row[1]]
= 0;
    int i = 0;
    for (auto& city : cities_map) city.second = i++;
    size = cities_map.size();
    matrix = std::vector<std::vector<int>>(size,
std::vector<int>(size, 100000));
    next = std::vector<std::vector<int>>(size,
std::vector<int>(size, -1));
    for (auto row : rows)
    {
        int i = cities_map[row[0]];
        int j = cities_map[row[1]];
        if (row[2] != "N/A")
        {
            matrix[i][j] = stoi(row[2]);
            next[i][j] = j;
        }
        if (row[3] != "N/A")
        {
            matrix[j][i] = stoi(row[3]);
            next[j][i] = i;
        }
    }
    for (int i = 0; i < size; ++i)
    {
        matrix[i][i] = 0;
        next[i][i] = i;
    }
}
```

```

        cities = std::vector<std::string>(cities_map.size());
        for (auto& city : cities_map) cities[city.second] =
city.first;
}

void Graph::Floyd()
{
    for (int k = 0; k < size; ++k)
        for (int i = 0; i < size; ++i)
            for (int j = 0; j < size; ++j)
            {
                if (matrix[i][j] > matrix[i][k] + matrix[k]
[j])
                {
                    matrix[i][j] = matrix[i][k] + matrix[k]
[j];
                    next[i][j] = next[i][k];
                }
            }
}

std::vector<std::string> Graph::route(std::string dep_city,
std::string arr_city)
{
    std::vector<std::string> path;
    int dep = cities_map[dep_city];
    int arr = cities_map[arr_city];
    if (next[dep][arr] == -1) return path;
    path.push_back(cities[dep]);
    while (dep != arr)
    {
        dep = next[dep][arr];
        path.push_back(cities[dep]);
    }
    return path;
}

std::vector<std::vector<int>> Graph::get_matrix()
{
    return matrix;
}

std::vector<std::string> Graph::get_cities()
{
    return cities;
}

std::map<std::string, int> Graph::get_cities_map()
{
    return cities_map;
}

int Graph::cost(std::string dep, std::string arr)
{
    return matrix[cities_map[dep]][cities_map[arr]];
}

using namespace std;

```

```

vector<vector<string>> read_csv(string file_name)
{
    fstream file;
    file.open(file_name, ios::in);
    vector<vector<string>> rows;
    int i = 0;
    string row, word;
    while (getline(file, row)) {
        stringstream ss(row);
        rows.push_back(vector<string>());
        while (getline(ss, word, ',')) rows[i].push_back(word);
        ++i;
    }
    file.close();
    return rows;
}

using namespace std;

int main()
{
    string file_name;
    cout << "Enter csv file name: ";
    cin >> file_name;
    vector<vector<string>> rows = read_csv(file_name);
    Graph g(rows);
    map<string, int> cities_map = g.get_cities_map();
    cout << "Cities:\n";
    for (auto city : cities_map) cout << city.first << "\n";
    g.Floyd();

    string dep_city;
    do
    {
        cout << "Departure city: ";
        cin >> dep_city;
    } while (cities_map.count(dep_city) == 0);

    string arr_city;
    do
    {
        cout << "Arrival city: ";
        cin >> arr_city;
    } while (cities_map.count(arr_city) == 0);
    int cost = g.cost(dep_city, arr_city);
    auto route = g.route(dep_city, arr_city);
    cout << "Route: ";
    for (auto city : route) cout << city << " ";
    cout << "\nCost: " << cost;
}

```