**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЕТ**

**по лабораторной работе № 3**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант №3**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8301 | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Бобров А.Б. |
| Преподаватель | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Тутуева А.В. |

Санкт-Петербург

2020

### Цель работы

Дан список возможных авиарейсов в текстовом файле. Реализовать алгоритм поиска наиболее эффективного по стоимости перелёта из одного города в другой, используя алгоритм Флойда-Уоршелла и матрицу смежности.

### Описание реализуемого класса и методов

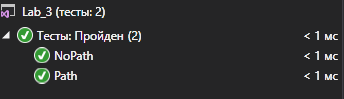
Matrix(List<string>\* data) — класс в котором реализуются построение матрицы смежности и основной метод Флойда-Уоршелла.

string Floyd\_Warshall(string From\_City, string To\_City) — основной метод в котором реализован алгоритм нахождения длин кратчайших путей между всеми парами вершин во взвешенном ориентированном графе.

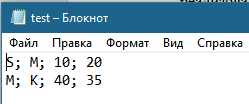
### Оценка временной сложности метода

string Floyd\_Warshall(string From\_City, string To\_City) – **O()**

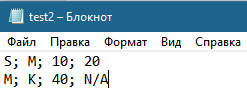
### Описание реализованных Unit-тестов



NoPath – проверяет алгоритм в случае наличия пути.

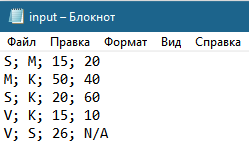


Path – тестирует метод, когда искомый путь отсутствует.

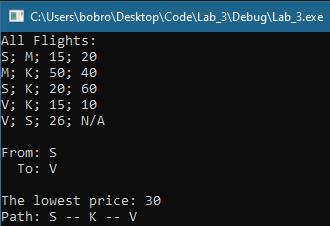


### Пример работы программы

Входные данные, приведены в файле.



Результат выполнения программы:



### Листинг

### main.cpp:

#include <iostream>

#include <fstream>

#include <string>

#include "Input.h"

#include "Matrix.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUS");

ifstream input("input.txt");

List<string>\* Flights = new List<string>();

string From;

string To;

Input(Flights, input);

cout << "All Flights:" << endl;

for (int i = 0; i < Flights->get\_size(); i++)

cout << Flights->at(i) << endl;

cout << "\nFrom: ";

getline(cin, From);

cout << " To: ";

getline(cin, To);

Matrix\* matrix = new Matrix(Flights);

cout << matrix->Floyd\_Warshall(From, To) << endl;

system("pause");

}

### Matrix.h:

#pragma once

#include"List.h"

#include"Map.h"

#include<string>

class Matrix

{

public:

Matrix(List<string>\* data)

{

CityToIndex = new Map<string, int>();

IndexToCity = new Map<int, string>();

int N = data->get\_size();

int index = 0;

for (int i = 0; i < N; i++)

{

string str\_cur = data->at(i);

int cur = str\_cur.find(';');

int cur1 = str\_cur.find(';', cur + 1);

string CityFrom = str\_cur.substr(0, cur);

string CityTo = str\_cur.substr(cur + 1, cur1 - cur - 1);

CityTo.erase(0, 1);

if (!CityToIndex->find\_is(CityFrom))

{

CityToIndex->insert(CityFrom, index);

IndexToCity->insert(index, CityFrom);

index++;

}

if (!CityToIndex->find\_is(CityTo))

{

CityToIndex->insert(CityTo, index);

IndexToCity->insert(index, CityTo);

index++;

}

}

//creating of path

size\_of\_matrix = index;

matrix = new double\*[size\_of\_matrix];

for (int i = 0; i < size\_of\_matrix; i++)

matrix[i] = new double[size\_of\_matrix];

for (int i = 0; i < size\_of\_matrix; i++)

for (int j = 0; j < size\_of\_matrix; j++)

matrix[i][j] = INF;

//input matrix path

for (int i = 0; i < N; i++)

{

int price\_1\_to\_2 = INF;

int price\_2\_to\_1 = INF;

string str\_cur = data->at(i);

int cur = str\_cur.find(';');

int cur1 = str\_cur.find(';', cur + 1);

int cur2 = str\_cur.find(';', cur1 + 1);

int cur3 = str\_cur.find(';', cur2 + 1);

string City1 = str\_cur.substr(0, cur);

string City2 = str\_cur.substr(cur + 1, cur1 - cur - 1);

City2.erase(0, 1);

if (str\_cur.substr(cur1 + 2, cur2 - 2 - cur1) != "N/A")

price\_1\_to\_2 = stof(str\_cur.substr(cur1 + 2, cur2 - 2 - cur1));

if (str\_cur.substr(cur2 + 2, cur3 - 1) != "N/A")

price\_2\_to\_1 = stoi(str\_cur.substr(cur2 + 2, cur3 - 2 - cur2));

matrix[CityToIndex->find(City1)][CityToIndex->find(City2)] = price\_1\_to\_2;

matrix[CityToIndex->find(City2)][CityToIndex->find(City1)] = price\_2\_to\_1;

}

}

string Floyd\_Warshall(string From\_City, string To\_City)

{

string cur;

while (!CityToIndex->find\_is(From\_City))

{

cout << "Wrong 'From' city name, please enter again" << endl;

cin >> From\_City;

}

while (!CityToIndex->find\_is(To\_City))

{

cout << "Wrong 'To' city name, please enter again" << endl;

cin >> To\_City;

}

int index\_from = CityToIndex->find(From\_City);

int index\_to = CityToIndex->find(To\_City);

int\*\* pre = new int\*[size\_of\_matrix];

for (int i = 0; i < size\_of\_matrix; i++)

{

pre[i] = new int[size\_of\_matrix];

for (int j = 0; j < size\_of\_matrix; j++)

pre[i][j] = i;

}

for (int k = 0; k < size\_of\_matrix; ++k)

for (int i = 0; i < size\_of\_matrix; ++i)

for (int j = 0; j < size\_of\_matrix; ++j)

{

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

{

matrix[i][j] = matrix[i][k] + matrix[k][j];

pre[i][j] = pre[k][j];

}

}

if (matrix[CityToIndex->find(From\_City)][CityToIndex->find(To\_City)] != INF)

{

int Price = matrix[CityToIndex->find(From\_City)][CityToIndex->find(To\_City)];

Price = round(Price);

cur = "\nThe lowest price: " + to\_string(Price) + "\nPath: ";

print\_path(index\_from, index\_to, pre, IndexToCity, cur);

cur.erase(cur.size() - 3);

}

else

{

cur = "There is no path, sorry :(";

}

return cur;

}

private:

void print\_path(int i, int j, int\*\* p, Map<int, string>\* IndexToCity, string&cur)

{

if (i != j)

print\_path(i, p[i][j], p, IndexToCity, cur);

cur = cur + IndexToCity->find(j) + " -- ";

}

double\*\* matrix;

int size\_of\_matrix;

Map<string, int>\* CityToIndex;

Map<int, string>\* IndexToCity;

const int INF = 1000000000;

};

### Unit-тесты:

#include "pch.h"

#include "CppUnitTest.h"

#include <fstream>

#include<string>

#include"../Lab\_3/Matrix.h"

#include"../Lab\_3/Input.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Lab3Test

{

TEST\_CLASS(Lab3Test)

{

public:

TEST\_METHOD(Path)

{

ifstream input("test.txt");

List<string>\* Flights = new List<string>();

string From = "S";

string To = "K";

Input(Flights, input);

Matrix\* matrix = new Matrix(Flights);

string str = "The lowest price: 50\nPath: S -- M -- K";

Assert::AreEqual(matrix->Floyd\_Warshall(From, To), str);

}

TEST\_METHOD(NoPath)

{

ifstream input("test2.txt");

List<string>\* Flights = new List<string>();

string From = "K";

string To = "M";

Input(Flights, input);

Matrix\* matrix = new Matrix(Flights);

string str = "There is no path, sorry :(";

Assert::AreEqual(matrix->Floyd\_Warshall(From, To), str);

}

};

}