

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

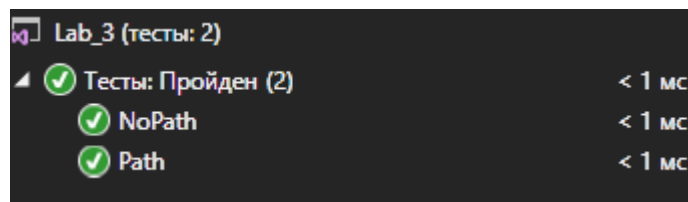
`Matrix(List<string>* data)` — класс в котором реализуются построение матрицы смежности и основной метод Флойда-Уоршелла.

`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.

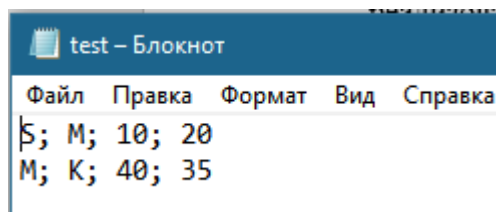
Оценка временной сложности метода

`string Floyd_Warshall(string From_City, string To_City) – $O(n^3)$`

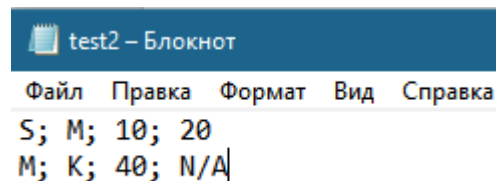
Описание реализованных Unit-тестов



NoPath – проверяет алгоритм в случае наличия пути.

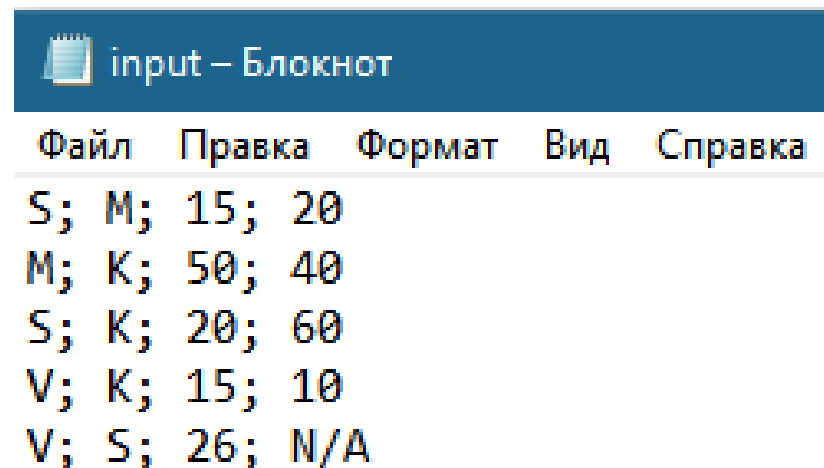


Path – тестирует метод, когда искомый путь отсутствует.



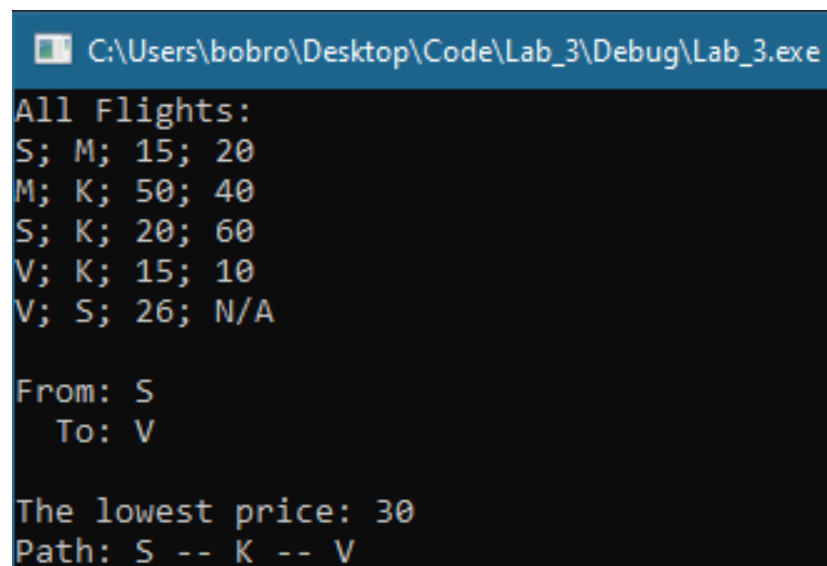
Пример работы программы

Входные данные, приведены в файле.



```
input – Блокнот
Файл  Правка  Формат  Вид  Справка
S; M; 15; 20
M; K; 50; 40
S; K; 20; 60
V; K; 15; 10
V; S; 26; N/A
```

Результат выполнения программы:



```
C:\Users\bobro\Desktop\Code\Lab_3\Debug\Lab_3.exe
All Flights:
S; M; 15; 20
M; K; 50; 40
S; K; 20; 60
V; K; 15; 10
V; S; 26; N/A

From: S
  To: V

The lowest price: 30
Path: S -- K -- V
```

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);
}

};

}

```