

**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 1**

Студент гр. 8301

Преподаватель

\_\_\_\_\_  
\_\_\_\_\_

Пчёлко В.А.

Тутуева А.В.

Санкт-Петербург

2020

## **Цель работы**

Реализовать программу принимающую список рейсов и цены за прямой и обратный и рейс и, в которой пользователь в свою очередь вводит город отправления и назначения и получает самый выгодный рейс или получает информацию о невозможности совершения перелёта. Для решения задачи использовать алгоритм Дейкстры

## **Описание реализуемых и вспомогательных классов**

Класс `DijkstraAlgo` – это представление графа путей списком смежностей. Класс содержит подкласс `Vertex` – представление вершин графа и методы

- Конструктор – который получает на вход список строк, и строит список типа `Vertex`/
- Деструктор – реализован деструктор, который вызывает метод `clear`(на основе обычного удаления двоичного дерева).
- `DijkstraAlgo` – рализация алгоритма Дейкстры. Функция также реализует вывод в консоль результат алгоритма.

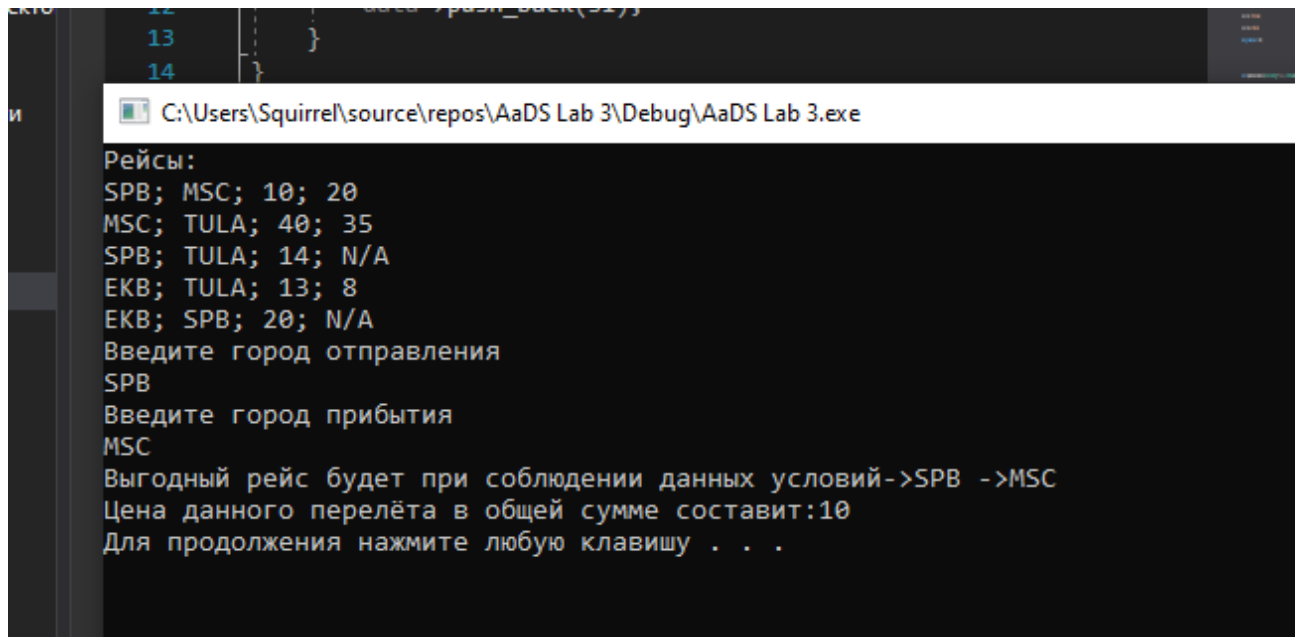
## **Оценка временной сложности алгоритмов**

- *`DijkstraAlgo` –  $O(n^2)$*

## **Описание реализованных unit-тестов**

Реализованные мною тесты проверяют правильное нахождение выгодного перелёта. Я рассмотрел ситуации, когда перелёт возможен, когда нет и когда город отправления и прибытия совпадает.

## Пример работы программы



The screenshot shows a console window titled "C:\Users\Squirrel\source\repos\AaDS Lab 3\Debug\AaDS Lab 3.exe". The output text is as follows:

```
Рейсы:  
SPB; MSC; 10; 20  
MSC; TULA; 40; 35  
SPB; TULA; 14; N/A  
EKB; TULA; 13; 8  
EKB; SPB; 20; N/A  
Введите город отправления  
SPB  
Введите город прибытия  
MSC  
Выгодный рейс будет при соблюдении данных условий->SPB ->MSC  
Цена данного перелёта в общей сумме составит:10  
Для продолжения нажмите любую клавишу . . .
```

## Код программы

### AdjList+Algo.h

```
#pragma once  
#include<string>  
#include "list.h"  
#include "map.h"  
class AdjList {  
    class Vertex {  
    public:  
        Vertex(int index_City = 0, double price = 0) :index_City(index_City),  
price(price) {}  
        int index_City;  
        double price;  
    };  
public:  
    AdjList(List<string>* data) {  
        graph = new List<Vertex>();  
        map_index_to_name_City = new Map<int, string>();  
        map_City_name_to_index = new Map<string, int>();  
        int N = data->get_size();  
        int index_city = 0;  
        for (int i = 0; i < N; i++) {  
            //заполнили все индексы разных городов и считаем  
            их кол-во  
            string CurrentString = data->at(i);  
            int current = CurrentString.find(';');  
            //первое вхождение ;  
            int current1 = CurrentString.find(';', current + 1);  
            //второе вхождение ;  
            string str_name_city1 = CurrentString.substr(0, current);  
            //получаем  
            первый город  
            string str_name_city2 = CurrentString.substr(current + 1, current1 -  
current - 1);  
            //получаем второй город  
            str_name_city2.erase(0, 1);  
            //удаляем пробел  
            if (!map_City_name_to_index->is_in_map(str_name_city1)) {  
                map_City_name_to_index->insert(str_name_city1, index_city);  
                map_index_to_name_City->insert(index_city, str_name_city1);  
                index_city++;  
            }  
            if (!map_City_name_to_index->is_in_map(str_name_city2)) {  
                map_City_name_to_index->insert(str_name_city2, index_city);  
                map_index_to_name_City->insert(index_city, str_name_city2);  
                index_city++;  
            }  
        }  
    }  
};
```

```

        map_index_to_name_City->insert(index_city, str_name_city2);
        index_city++;
    }
}
////////////////////////////////////
size = index_city;//длина основного списка
graph = new List<Vertex>[size];

////////////////////////////////////заполня
ем список цен(смедности)
    for (int i = 0; i < N; i++) {
        int price_1_to_2 = INF;
        int price_2_to_1 = INF;
        string CurrentString = data->at(i);
        int current = CurrentString.find(';');//первое вхождение ;
        int current1 = CurrentString.find(';', current + 1);//второе вхождение
;
        int current2 = CurrentString.find(';', current1 + 1);//3 вхождение ;
        int current3 = CurrentString.find(';', current2 + 1);//4 вхождение ;
        string str_name_city1 = CurrentString.substr(0, current);//получаем
первый город
        string str_name_city2 = CurrentString.substr(current + 1, current1 -
current - 1);//получаем второй город
        str_name_city2.erase(0, 1);//удаляем пробел
        //cout << stof(CurrentString.substr(current1 + 2, current2 - 2 -
current1)) << 'f';
        if (CurrentString.substr(current1 + 2, current2 - 2 - current1) !=
"N/A")
            price_1_to_2 = stof(CurrentString.substr(current1 + 2, current2 -
2 - current1));
        if (CurrentString.substr(current2 + 2, current3 - 1) != "N/A")
            price_2_to_1 = stoi(CurrentString.substr(current2 + 2, current3 -
2 - current2));
        if (price_1_to_2 != INF) {
            Vertex v1(map_City_name_to_index->find(str_name_city2),
price_1_to_2);//временная вершина для добавления
            graph[map_City_name_to_index-
>find(str_name_city1)].push_back(v1);
        }
        if (price_2_to_1 != INF) {
            Vertex v2(map_City_name_to_index->find(str_name_city1),
price_2_to_1);//временная вершина для добавления
            graph[map_City_name_to_index-
>find(str_name_city2)].push_back(v2);
        }
    }
}

void DijkstraAlgo(string city_Start, string city_End) {
    while (!map_City_name_to_index->is_in_map(city_Start)) {
        cout << "The departure city is missing, enter it again" << endl;
        cin >> city_Start;
    }
    while (!map_City_name_to_index->is_in_map(city_End)) {
        cout << "The arrival city is missing, enter it again" << endl;
        cin >> city_End;
    }
    int index_city = 0;
    int index_start_vertex = map_City_name_to_index->find(city_Start);//находим
индекс города отправления
    bool* visited = new bool[size];//заполнить все false
    int* d = new int[size];//расстояния от стартовой вершины
    for (int i = 0; i < size; i++) { //заполняем бесконечно большими числами(пока
несуществующие расстояния) и метки посещения обнуляем
        d[i] = INF;

```

```

        visited[i] = false;
    }
    d[index_start_vertex] = 0; //вершина начала пути всегда ноль(откуда
отсчитываем)
    int* path = new int[size]; //предки (последовательность краткого пути)
    ///////////////////////////////////////////////////////////////////
    ///////////////////////////////////////////////////////////////////
    for (int i = 0; i < size; ++i) {
        int v = -1;
        for (int j = 0; j < size; ++j)
            if (!visited[j] && (v == -1 || d[j] < d[v]))
                v = j;
        if (d[v] == INF)
            break;
        visited[v] = true;
        for (size_t j = 0; j < graph[v].get_size(); ++j) {
            int to = graph[v].at(j).index_City,
                len = graph[v].at(j).price;
            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                path[to] = v;
            }
        }
    }
    int End_index = map_City_name_to_index->find(city_End); //поиск индекса города
прибытия
    int Start_index = map_City_name_to_index->find(city_Start);
    List<int>* path_current = new List<int>();
    for (int v = End_index; v != Start_index; v = path[v])
        path_current->push_back(v);
    path_current->push_back(Start_index);
    path_current->reverse();
    cout << "Выгодный рейс будет при соблюдении данных условий";
    for (int i = 0; i < path_current->get_size(); i++) {
        cout << "->";
        cout << map_index_to_name_City->find(path_current->at(i)) << ' ';
    }
    cout << endl << "Цена данного перелёта в общей сумме составит:" <<
d[End_index] << endl;
}
private:
    List<Vertex>* graph;
    Map<string, int>* map_City_name_to_index;
    Map<int, string>* map_index_to_name_City;
    int size;
    const int INF = INT_MAX;
};

```

## Код Unit-тестов

```

#include "stdafx.h"
#include "CppUnitTest.h"
#include <fstream>
#include <string>
#include "../AaDS Lab 3/list.h"
#include "../AaDS Lab 3/DijkstraAlgo.h"
#include "../AaDS Lab 3/input_data.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
    TEST_CLASS(UnitTest1){
    public:

```

```

TEST_METHOD(TestAvailablePath1)
{
    ifstream
vvod("C:\\Users\\Squirrel\\Desktop\\AISDlab3var3\\l3input.txt");
    List<string>* list_fly = new List<string>();
    string city_Start = "SPB";
    string city_End = "EKB";
    InputDataFromFile(list_fly, vvod);
    AdjList* adj = new AdjList(list_fly);
    string cur = "Route:\\nSPB ->TULA ->EKB \\nThe best route for the
price:22\\n";
    Assert::AreEqual(adj->DijkstraAlgo(city_Start, city_End), cur);
}
TEST_METHOD(TestUnavailablePath1)
{
    ifstream
vvod("C:\\Users\\Squirrel\\Desktop\\AISDlab3var3\\l3input.txt");
    List<string>* list_fly = new List<string>();
    string city_Start = "TULA";
    string city_End = "SPB";
    InputDataFromFile(list_fly, vvod);
    AdjList* adj = new AdjList(list_fly);
    string cur = "This route can't be built, try waiting for the flight
schedule for tomorrow!";
    Assert::AreEqual(adj->DijkstraAlgo(city_Start, city_End), cur);
}
TEST_METHOD(TestSameCity)
{
    ifstream
vvod("C:\\Users\\Squirrel\\Desktop\\AISDlab3var3\\l3input.txt");
    List<string>* list_fly = new List<string>();
    string city_Start = "SPB";
    string city_End = "SPB";
    InputDataFromFile(list_fly, vvod);
    AdjList* adj = new AdjList(list_fly);
    string cur = "Route:\\nSPB \\nThe best route for the price:0\\n";
    Assert::AreEqual(adj->DijkstraAlgo(city_Start, city_End), cur);
}
TEST_METHOD(TestPath1)
{
    ifstream
vvod("C:\\Users\\Squirrel\\Desktop\\AISDlab3var3\\l3input.txt");
    List<string>* list_fly = new List<string>();
    string city_Start = "EKB";
    string city_End = "TULA";
    InputDataFromFile(list_fly, vvod);
    AdjList* adj = new AdjList(list_fly);
    string cur = "Route:\\nEKB ->TULA \\nThe best route for the price:13\\n";
    Assert::AreEqual(adj->DijkstraAlgo(city_Start, city_End), cur);
}
TEST_METHOD(TestePath2)
{
    ifstream
vvod("C:\\Users\\Squirrel\\Desktop\\AISDlab3var3\\l3input.txt");
    List<string>* list_fly = new List<string>();
    string city_Start = "TULA";
    string city_End = "MSC";
    InputDataFromFile(list_fly, vvod);
    AdjList* adj = new AdjList(list_fly);
    string cur = "This route can't be built, try waiting for the flight
schedule for tomorrow!";
    Assert::AreEqual(adj->DijkstraAlgo(city_Start, city_End), cur);
}

```

```
}  
};
```

### **Вывод**

В данной лабораторной работе я познакомился алгоритмом Дейкстры и смог применить в реальной ситуации на примере авиарейсов и нахождение выгодного пути.