**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**

Студентка гр. 8302 \_\_\_\_\_\_\_\_\_\_\_\_\_ Колмакова С.П.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ 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

• Констуктор – 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 различные стандартные и нестандартные случаи введенных данных, такие как:

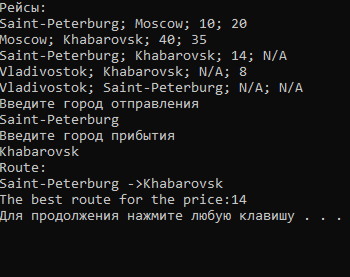
- Введение существующего кратчайшего пути

- Введение маршрута, которого нет в списке маршрутов

- Введение маршрута, в котором начальная точка является конечной

-Введение такого начального и конечного городов, что кратчайший маршрут будет представлять собой больше 1 города.

Пример работы программы

**Пример**  


Листинг

**DijkstraAlgo.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);//второ 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++;

}

}

size = index\_city;//длина основного списка

graph = new List<Vertex>[size];

//заполняем список цен(смежности)

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);

}

}

}

string DijkstraAlgo(string city\_Start, string city\_End) {

if (city\_Start != city\_End) {

string answer;

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\* distance = new int[size];//расстояния от стартовой вершины

for (int i = 0; i < size; i++) {//заполняем бесконечно большими числами(пока несуществкюшие расстояния) и метки посещения обнуляем

distance[i] = INF;

visited[i] = false;

}

distance[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 vertex = -1;

for (int j = 0; j < size; ++j)

if (!visited[j] && (vertex == -1 || distance[j] < distance[vertex]))

vertex = j;

if (distance[vertex] == INF)

break;

visited[vertex] = true;

for (size\_t j = 0; j < graph[vertex].get\_size(); ++j) {

int to = graph[vertex].at(j).index\_City,

len = graph[vertex].at(j).price;

if (distance[vertex] + len < distance[to]) {

distance[to] = distance[vertex] + len;

path[to] = vertex;

}

}

}

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);

if (distance[End\_index] != INF) {

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();

answer += "Route:\n";

string cur;

for (int i = 0; i < path\_current->get\_size(); i++) {

cur += "->";

cur += map\_index\_to\_name\_City->find(path\_current->at(i)) + ' ';

}

cur.erase(0, 2);

answer += cur + "\nThe best route for the price:" + to\_string(distance[End\_index]) + '\n';

}

else {

answer = "This route can't be built, try waiting for the flight schedule for tomorrow!";

}

return answer;

}

else throw exception("End city can't be equal to start city");

}

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;

};

**UnitTestDijkstra.cpp**

#include "pch.h"

#include "CppUnitTest.h"

#include"../AaDS Lab 3/DijkstraAlgo.h"

#include <fstream>

#include"../AaDS Lab 3/input\_data.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTestForDijkstraAlgorithm

{

TEST\_CLASS(UnitTestForDijkstraAlgorithm)

{

public:

TEST\_METHOD(TestAvailablePath)

{

ifstream vvod("C:\\Users\\sveta\\Desktop\\lab3\\DijkstraAlgorithm\\input.txt");

List<string>\* list\_fly = new List<string>();

string city\_Start = "Moscow";

string city\_End = "Saint-Peterburg";

InputDataFromFile(list\_fly, vvod);

AdjList\* adj = new AdjList(list\_fly);

string cur = "Route:\nMoscow ->Saint-Peterburg \nThe best route for the price:20\n";

Assert::AreEqual(adj->DijkstraAlgo(city\_Start, city\_End), cur);

}

TEST\_METHOD(TestUnavailablePath)

{

ifstream vvod("C:\\Users\\sveta\\Desktop\\lab3\\DijkstraAlgorithm\\input.txt"");

List<string>\* list\_fly = new List<string>();

string city\_Start = "Vladivostok";

string city\_End = "Moscow";

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(TestPathExeption)

{

try {

ifstream vvod("C:\\Users\\sveta\\Desktop\\lab3\\DijkstraAlgorithm\\input.txt");

List<string>\* list\_fly = new List<string>();

string city\_Start = "Vladivostok";

string city\_End = "Vladivostok";

InputDataFromFile(list\_fly, vvod);

AdjList\* adj = new AdjList(list\_fly);

}

catch (exception& ex) {

Assert::AreEqual(ex.what(), "End city can't be equal to start city");

}

}

TEST\_METHOD(TestePath)

{

ifstream vvod("C:\\Users\\sveta\\Desktop\\lab3\\DijkstraAlgorithm\\input.txt");

List<string>\* list\_fly = new List<string>();

string city\_Start = "Moscow";

string city\_End = "Khabarovsk";

InputDataFromFile(list\_fly, vvod);

AdjList\* adj = new AdjList(list\_fly);

string cur = "Route:\nMoscow ->Saint-Peterburg ->Khabarovsk \nThe best route for the price:34\n";

Assert::AreEqual(adj->DijkstraAlgo(city\_Start, city\_End), cur);

}

};

}

Вывод

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