**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ОТЧЁТ

**по лабораторной работе №3**

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

**Тема: «Алгоритмы на графах»**

**Вариант 2.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Бабкин И.А. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи**

Дан список возможных авиарейсов в текстовом файле. Необходимо найти наиболее эффективный по стоимости перелет из города ***i*** в город ***j***.

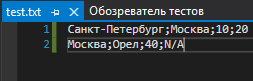
## Описание и оценка временной сложности функций

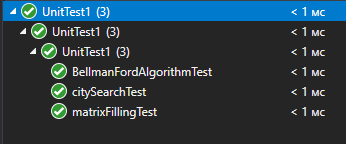
|  |  |  |
| --- | --- | --- |
| Функция | Описание | Временная сложность |
| void citySearch | Ищет уникальные города и заносит их в список | O(n) |
| void matrixFilling | Заполняет матрицу смежности | O(n) |
| void BellmanFordAlgorithm | Ищет наиболее эффективные по стоимости перелеты при помощи алгоритма Беллмана-Форда | O(n^3) |

## Также в лабораторной работе был использован LinkedList

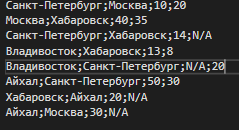
## Описание реализованный unit-тестов

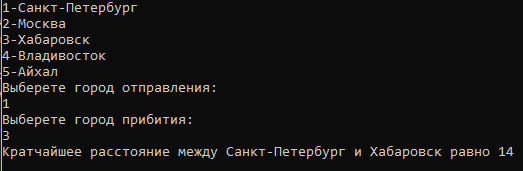
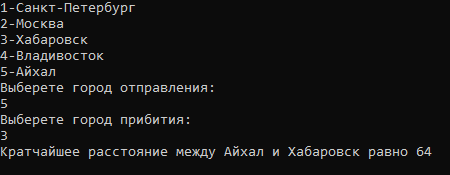
|  |
| --- |
| Название теста |
| citySearchTest |
| matrixFillingTest |
| BellmanFordAlgorithmTest |

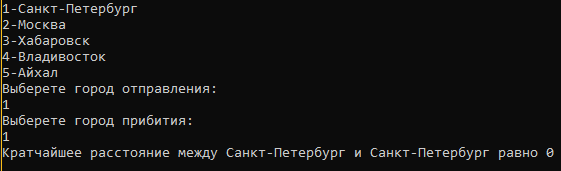
В тестах использовался данный текстовый файл:  




## Пример работы





## Листинг

**BellmanFord.h**

#pragma once

#include "list.h"

void citySearch(LinkedList<string>\* listOfCities, ifstream& fin);

void matrixFilling(LinkedList<string>\* listOfCities, ifstream& fin, int\*\* adjacencyMatrix);

void BellmanFordAlgorithm(int\*\* adjacencyMatrix, int\*\* shortestPathMatrix, int n, int start);

**BellmanFord.cpp**

#include <iostream>

#include <fstream>

#include "list.h"

#include <string>

#include "BellmanFord.h"

using namespace std;

void citySearch(LinkedList<string>\* listOfCities, ifstream& fin) {

string str;

string tempStr;

int counter, index;

while (!fin.eof()) {

getline(fin, str);

counter = 0;

for (size\_t i = 0; i < str.size(); i++) {

tempStr = "";

while (str[i] != ';' && i != str.size()) {

tempStr += str[i++];

}

index = listOfCities->find(tempStr);

if (index == -1) {

listOfCities->push\_back(tempStr);

}

counter++;

if (counter == 2) {

break;

}

}

}

}

void matrixFilling(LinkedList<string>\* listOfCities, ifstream& fin, int\*\* adjacencyMatrix)

{

string str;

string tempStr;

int indexI, indexJ;

int counter;

while (!fin.eof()) {

getline(fin, str);

counter = 0;

for (size\_t i = 0; i < str.size(); i++) {

tempStr = "";

while (str[i] != ';' && i != str.size()) {

tempStr += str[i++];

}

if (counter == 0) {

indexI = listOfCities->find(tempStr);

}

if (counter == 1) {

indexJ = listOfCities->find(tempStr);

}

if (counter == 2 && tempStr != "N/A") {

adjacencyMatrix[indexI][indexJ] = atoi(tempStr.c\_str());

}

if (counter == 3 && tempStr != "N/A") {

adjacencyMatrix[indexJ][indexI] = atoi(tempStr.c\_str());

}

counter++;

}

}

}

void BellmanFordAlgorithm(int\*\* adjacencyMatrix, int\*\* shortestPathMatrix, int n, int start)

{

for (int i = 0; i < n; i++) {

shortestPathMatrix[i][start] = 0;

}

for (int k = 1; k < n; k++) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (shortestPathMatrix[k-1][j] + adjacencyMatrix[j][i] < shortestPathMatrix[k][i] && adjacencyMatrix[j][i] != 0) {

shortestPathMatrix[k][i] = shortestPathMatrix[k-1][j] + adjacencyMatrix[j][i];

}

}

}

}

}