**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 3.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Квитко Д.В. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

**Постановка задачи**

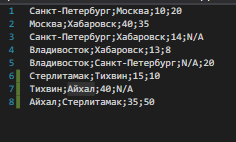
Найти наиболее эффективный по стоимости перелет из города i в город j с помощью алгоритма Флойда-Уоршелла и матрицы смежности

## Описание и оценка временной сложности функций

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Временная сложность |
| void reading(queue\* que, std::ifstream& text); | Создает очередь, в которую записываются все уникальные города | O(n) |
| void creating\_matrix(queue\* que, std::ifstream& text, int\*\* matrix, int N); | Создается матрица смежности | O(n) |
| void FU(int\*\* D, int N); | Строится матрица кратчайших путей с помощью алгоритма Флойда-Уоршелла | O(n^3) |

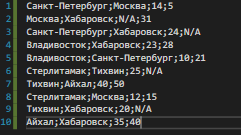
## Описание реализованный unit-тестов

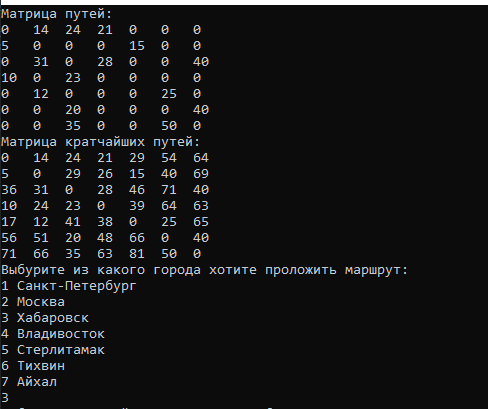
|  |  |
| --- | --- |
| Название теста |  |
| TestMethod1 | Проверяет корректно ли создается очередь с уникальными городами |
| TestMethod2 | Проверяет на правильность составления матрицы смежности |
| TestMethod3 | Проверка алгоритма Флойда-Уоршелла |

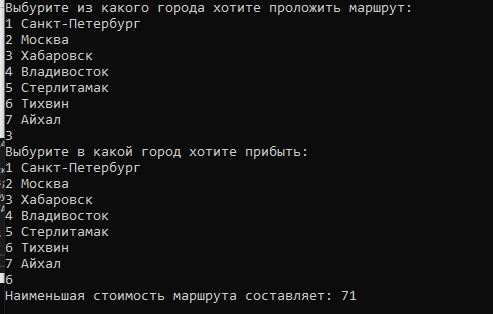


## Пример работы

Пример:







## Листинг

**queue.h**

ifndef queue\_H

#define queue\_H

#include <iostream>

#include <string> // подключаем строки

#include <fstream>

class queue

{

private:

class queueNode {

public:

queueNode\* next;

std::string city;

queueNode(queueNode\* next = nullptr)

{

this->next = next;

};

queueNode(std::string city, queueNode\* next = nullptr)

{

this->city = city;

this->next = next;

};

~queueNode()

{

}

};

public:

queueNode\* head;

queueNode\* tail;

unsigned int size;

queue() {

tail = nullptr;

head = nullptr;

size = 0;

}

~queue() {

clear();

}

unsigned int get\_size() {

return size;

}

void clear() {

while (size != 0)

{

pop();

}

}

void push(std::string city) {

if (size == 0) {

head = new queueNode(city);

tail = head;

}

else {

tail->next = new queueNode(city);

tail = tail->next;

}

size++;

}

void pop() {

if (size == 0) {

return;

}

if (size == 1) {

delete head;

reset\_list();

}

else {

queueNode\* current = head;

head = head->next;

delete current;

}

size--;

}

void reset\_list()

{

head = nullptr;

tail = nullptr;

}

std::string at(unsigned int index)

{

if (index >= size) {

throw "Index is greater than list size";

}

else {

queueNode\* current = head;

unsigned int counter = 0;

while (counter != index) {

current = current->next;

counter++;

}

return current->city;

}

}

};

#endif

**function.h**

#include"queue.h"

void reading(queue\* que, std::ifstream& text);

void creating\_matrix(queue\* que, std::ifstream& text, int\*\* matrix, int N);

void FU(int\*\* D, int N);

**function.cpp**

#include<iostream>

#include<fstream>

#include<string>

#include"queue.h"

#include"function.h"

void reading(queue\* que,std::ifstream& text) {

std::string str,city;

int count = 0;

bool flag = true;

while (!text.eof()) {

std::getline(text, str);

for (unsigned int i = 0; i < str.size(); i++) {

if (str[i]!=';')

{

city += str[i];

}

else if(count!=2) {

for (int j = 0; j < que->get\_size(); j++) {

if (city == que->at(j)) {

flag = false;

}

}

if (flag) {

que->push(city);

}

flag = true;

count++;

city.clear();

}

else {

city.clear();

break;

}

}

count = 0;

}

str.clear();

city.clear();

}

void creating\_matrix(queue\* que, std::ifstream& text, int\*\* matrix, int N) {

std::string str, word;

int count = 0;

int unit\_1 = 0, unit\_2 = 0;

while (!text.eof()) {

std::getline(text, str);

for (unsigned int i = 0; i < str.size() + 1; i++) {

if (str[i] != ';' && i < str.size())

{

word += str[i];

}

else if (count < 2) {

for (int j = 0; j < que->get\_size(); j++) {

if (word == que->at(j)) {

if (count == 0)

unit\_1 = j;

else

unit\_2 = j;

break;

}

}

count++;

word.clear();

}

else {

if (count == 2) {

if (word != "N/A") {

matrix[unit\_1][unit\_2] = atoi(word.c\_str());

}

else {

matrix[unit\_1][unit\_2] = 0;

}

}

else {

if (word != "N/A") {

matrix[unit\_2][unit\_1] = atoi(word.c\_str());

}

else {

matrix[unit\_2][unit\_1] = 0;

}

}

count++;

word.clear();

}

}

count = 0;

word.clear();

}

str.clear();

word.clear();

}

void FU(int\*\* D, int N){

int V = N;

for (int i = 0; i < V; i++) D[i][i] = 0;

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

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

for (int k = 0; k < V; k++)

if (D[j][i] && D[i][k] && j != k)

if (D[j][i] + D[i][k] < D[j][k] || D[j][k] == 0)

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

}

**main.cpp**

#include<iostream>

#include <limits>

#include<fstream>

#include<string>

#include<Windows.h>

#include<iomanip>

#include"queue.h"

#include"function.h"

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::ifstream file;

file.open("text.txt", std::ios::in);

int index\_i, index\_j;

queue\* que = new queue;

reading(que, file);

int size = que->get\_size();

int\*\* matrix, \*\*matrix\_1;

matrix = new int\* [size];

matrix\_1 = new int\* [size];

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

matrix[i] = new int[size];

}

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

matrix\_1[i] = new int[size];

}

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

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

matrix[i][j] = 0;

}

}

file.clear();

file.seekg(0);

creating\_matrix(que, file, matrix, size);

file.close();

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

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

matrix\_1[i][j] = matrix[i][j];

}

}

FU(matrix\_1, size);

std::cout.setf(std::ios::left);

std::cout << "Mатрица путей: " << std::endl;

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

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

std::cout << std::setw(3) <<matrix[i][j]<<" ";

}

std::cout << std::endl;

}

std::cout << "Mатрица кратчайших путей: " << std::endl;

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

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

std::cout<<std::setw(3)<< matrix\_1[i][j]<<" ";

}

std::cout << std::endl;

}

std::cout << "Выбурите из какого города хотите проложить маршрут: " << std::endl;

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

std::cout << i + 1 << " " << que->at(i) << std::endl;

}

std::cin >> index\_i;

std::cout << "Выбурите в какой город хотите прибыть: " << std::endl;

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

std::cout << i + 1 << " " << que->at(i) << std::endl;

}

std::cin >> index\_j;

std::cout << "Наименьшая стоимость маршрута составляет: " << matrix\_1[index\_i-1][index\_j-1]<< std::endl;

return 0;

}