**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»**

**Тема: «Алгоритмы на графах»**

**Вариант 18(3)**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9302 |  | Плюснина Е.Ю. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2020

**Постановка задачи**

Найти наиболее эффективный по стоимости перелет из города i в город j по алгоритму Флойда-Уоршелла.

# Описание реализуемого класса, оценка временной сложности

BiList<string> Node, Information;– списки, где хранятся имена узлов графа и информация из текстового файла

int\*\* Matrix;– указатель на матрицу смежности

size\_t size;– размер матрицы смежности

**Функции:**

void input(string filename = "in.txt");– чтение из файла. Оценка временной сложности: О(n)

void clear(); - очистка. Оценка временной сложности: О(n)

void Summarize();– создание списка имён узлов и матрицы смежности. Оценка временной сложности: О(n2)

size\_t FloydWarshall(size\_t, size\_t);– алгоритм Флойда-Уоршелла. Оценка временной сложности: О(n3)

# Листинг

**List.h**

#pragma once

#include <cstddef>

#include <string>

using namespace std;

template<class Type>

class BiList

{

public:

BiList();

void push\_back(Type);

void push\_front(Type);

void pop\_back();

void pop\_front();

void insert(Type, size\_t);

Type at(size\_t);

void remove(size\_t);

size\_t get\_size();

void print\_to\_console();

void clear();

void set(size\_t, Type);

bool IsEmpty();

void reverse();

void Swap(size\_t, size\_t);

int find(Type);

~BiList();

private:

struct Node

{

Type data;

Node\* before = NULL;

Node\* next = NULL;

};

Node\* first;

Node\* last;

size\_t size;

};

**List.cpp**

#pragma once

#include "List.h"

#include <iostream>

template<class Type>

BiList<Type>::BiList()

{

first = NULL;

last = NULL;

size = 0;

}

template<class Type>

void BiList<Type>::push\_back(Type Value)

{

Node\* temp = new Node;

temp->next = NULL;

temp->data = Value;

if (!this->IsEmpty())

{

temp->before = last;

last->next = temp;

last = temp;

}

else {

temp->before = NULL;

first = last = temp;

}

size++;

}

template<class Type>

void BiList<Type>::push\_front(Type Value)

{

Node\* temp = new Node;

temp->before = NULL;

temp->data = Value;

if (!this->IsEmpty())

{

temp->next = first;

first->before = temp;

first = temp;

}

else

{

temp->before = NULL;

temp->next = NULL;

first = last = temp;

}

size++;

}

template<class Type>

void BiList<Type>::pop\_back()

{

if (size == 0) {

return;

}

if (size == 1)

{

delete last;

last = first = NULL;

size--;

return;

}

delete last->next;

last = last->before;

last->next = NULL;

size--;

}

template<class Type>

void BiList<Type>::pop\_front()

{

if (size == 0) {

return;

}

if (size == 1)

{

delete first;

first = last = NULL;

size--;

return;

}

delete first->before;

first->before = NULL;

first = first->next;

size--;

}

template<class Type>

void BiList<Type>::insert(Type Value, size\_t index)

{

if ((size == 0 && index == 0) || index >= size)

throw "Error";

if (index == 0)

{

this->push\_front(Value);

return;

}

Node\* currentf = first;

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

currentf = currentf->next;

}

Node\* temp = new Node;

temp->data = Value;

temp->next = currentf;

temp->before = currentf->before;

currentf->before = temp;

temp->before->next = temp;

size++;

}

template<class Type>

Type BiList<Type>::at(size\_t index)

{

if (index >= size || index < 0) {

throw "Error";

}

Node\* currentf = first;

for (size\_t i = 0; i < index; i++)

currentf = currentf->next;

return currentf->data;

}

template<class Type>

void BiList<Type>::remove(size\_t index)

{

if (index >= size || index <= 0) {

throw "Error";

}

if (index == 0)

{

this->pop\_front();

return;

}

if (index == size - 1)

{

this->pop\_back();

return;

}

Node\* currentf = first;

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

currentf = currentf->next;

}

Node\* temp = currentf;

currentf->before->next = currentf->next;

currentf->next->before = currentf->before;

delete temp;

size--;

}

template<class Type>

size\_t BiList<Type>::get\_size()

{

return size;

}

template<class Type>

void BiList<Type>::print\_to\_console()

{

Node\* currentf = first;

if (first)

{

while (currentf->next)

{

std::cout << currentf->data << "<-->";

currentf = currentf->next;

}

std::cout << currentf->data;

}

else {

throw "List is empty";

}

}

template<class Type>

void BiList<Type>::clear()

{

while (first != NULL) {

this->pop\_front();

}

first = last = NULL;

}

template<class Type>

void BiList<Type>::set(size\_t index, Type Value)

{

if (index >= size || index < 0) {

throw "Error";

}

Node\* currentf = first;

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

currentf = currentf->next;

}

currentf->data = Value;

}

template<class Type>

bool BiList<Type>::IsEmpty()

{

if (first == NULL) {

return true;

}

return false;

}

template<class Type>

void BiList<Type>::reverse()

{

if (!first || !first->next) {

return;

}

last = first;

Node\* temp = NULL;

Node\* current = first;

while (current != NULL)

{

temp = current->before;

current->before = current->next;

current->next = temp;

current = current->before;

}

first = temp->before;

}

template<class Type>

void BiList<Type>::Swap(size\_t first, size\_t other)

{

Node\* temp\_f = first, \* temp\_s = first;

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

temp\_f = temp\_f->next;

}

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

temp\_s = temp\_s->next;

}

Type temp = temp\_f->data;

temp\_f->data = temp\_s->data;

temp\_s->data = temp;

}

template<class Type>

int BiList<Type>::find(Type info)

{

Node\* current = first;

int i = 0;

while (i < size)

{

if (current->data == info)

return i;

current = current->next;

i++;

}

return -1;

}

template<class Type>

BiList<Type>::~BiList()

{

while (first)

{

last = first->next;

delete first;

first = last;

}

}

**Floyd-Warshall.h**

#pragma once

#include <fstream>

#include "List.cpp"

class Graph

{

private:

BiList<string> Node, Information;

int\*\* Matrix;

size\_t size;

void Summarize();

public:

Graph();

void input(string filename = "in.txt");

void clear();

size\_t FloydWarshall(size\_t, size\_t);

~Graph();

};

**Floyd-Warshall.cpp**

#include "Floyd-Warshall.h"

#include <iostream>

#define inf INT\_MAX/2-1

using namespace std;

Graph::Graph()

{

Matrix = NULL;

size = 0;

}

Graph::~Graph()

{

clear();

}

void Graph::input(string filename)

{

Information.clear(); Node.clear();

if (filename == "") {

filename = "in.txt";

}

string temp;

ifstream File;

File.open(filename.c\_str(), ios::in);

if (File.bad()) {

throw "Error";

}

while (!File.eof())

{

getline(File, temp);

Information.push\_back(temp);

}

Summarize();

}

size\_t Graph::FloydWarshall(size\_t first, size\_t other)

{

if (first == other) {

return 0;

}

int\* Length = new int[Node.get\_size()];

int\* Way = new int[Node.get\_size()];

for (size\_t i = 0; i < Node.get\_size(); i++)

{

if (i != first) {

Length[i] = Matrix[first][i];

}

else {

Length[i] = 0;

}

Way[i] = -1;

}

for (size\_t k = 0; k < Node.get\_size(); k++)

{

bool A = false;

for (size\_t i = 0; i < Node.get\_size(); i++)

{

for (size\_t j = 0; j < Node.get\_size(); j++)

{

if (i != j && (Length[j] + Matrix[j][i]) < Length[i])

{

Length[i] = Length[j] + Matrix[j][i];

Way[k] = (int)j;

A = true;

}

}

}

if (!A) break;

}

if (Length[other] == inf) {

cout << "Пути нет" << endl;

}

else

{

cout << "Путь:" << Length[other] << endl;

cout << "Маршрут: " << Node.at(first);

for (size\_t i = 0; i < Node.get\_size(); i++)

{

if (Way[i] == -1 || Way[i] == other)

break;

cout << " -> " << Node.at(Way[i]);

}

cout << "->" << Node.at(other) << endl;

}

size\_t result = Length[other];

delete[] Length; delete[] Way;

return result;

}

void Graph::clear()

{

Node.clear();

Information.clear();

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

{

delete[] Matrix[i];

}

delete[] Matrix;

Matrix = NULL;

size = 0;

}

void Graph::Summarize()

{

size\_t Iterator = Information.get\_size();

int\*\* temp1 = new int\* [Iterator];

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

temp1[i] = new int[4];

}

for (size\_t i = 0; i < Iterator; i++)

{

string temp = Information.at(i), name = "";

string Way[4] = { "","","","" };

size\_t k = 0;

for (size\_t j = 0; j < temp.size(); j++)

{

if (temp[j] == ';') {

k++;

}

else {

Way[k].push\_back(temp[j]);

}

}

for (size\_t k = 0; k < 2; k++)

{

if (Node.find(Way[k]) == -1) {

Node.push\_back(Way[k]);

}

if (Way[k + 2][0] == 'N') {

temp1[i][k + 2] = inf;

}

else {

temp1[i][k + 2] = stoi(Way[k + 2]);

}

}

temp1[i][0] = Node.find(Way[0]);

temp1[i][1] = Node.find(Way[1]);

}

size = Node.get\_size();

Matrix = new int\* [size];

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

{

Matrix[i] = new int[size];

for (size\_t j = 0; j < Node.get\_size(); j++) {

Matrix[i][j] = inf;

}

}

for (size\_t i = 0; i < Iterator; i++)

{

Matrix[temp1[i][0]][temp1[i][1]] = temp1[i][2];

Matrix[temp1[i][1]][temp1[i][0]] = temp1[i][3];

delete[] temp1[i];

}

delete[] temp1;

}

**UnitTest1.cpp**

#include <iostream>

#include "CppUnitTest.h"

#include "../AlgStrD\_4sem\_lab3/Floyd-Warshall.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(test)

{

Assert::AreEqual(1, 1);

}

TEST\_METHOD(Graph1)

{

try {

Graph Test;

string s = "in2.txt";

Test.input(s);

size\_t res = 3, temp1 = Test.FloydWarshall(0, 3);

Assert::AreEqual(temp1, res);

Test.clear();

}

catch (char\*)

{

Assert::AreEqual(1, 0);

}

}

TEST\_METHOD(Graph2)

{

try {

Graph Test;

Test.input();

size\_t res = 68, temp1 = Test.FloydWarshall(3, 0);;

Assert::AreEqual(temp1, res);

Test.clear();

}

catch (char\*)

{

Assert::AreEqual(1, 0);

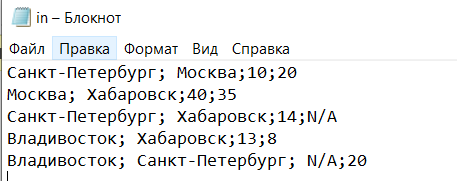
}

}

};

}

**Примеры работы программы**



Путь: 10

Маршрут: Санкт-Петербург->Москва

Путь: 14

Маршрут: Санкт-Петербург->Хабаровск

Путь: 20

Маршрут: Санкт-Петербург->Владивосток

Путь: 20

Маршрут: Москва->Санкт-Петербург

Путь: 34

Маршрут: Москва -> Санкт-Петербург->Хабаровск

Путь: 40

Маршрут: Москва -> Санкт-Петербург->Владивосток

Путь: 55

Маршрут: Хабаровск -> Москва->Санкт-Петербург

Путь: 35

Маршрут: Хабаровск->Москва

Путь: 8

Маршрут: Хабаровск -> Москва->Владивосток

Путь: 68

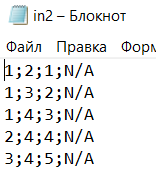
Маршрут: Владивосток -> Хабаровск -> Москва->Санкт-Петербург

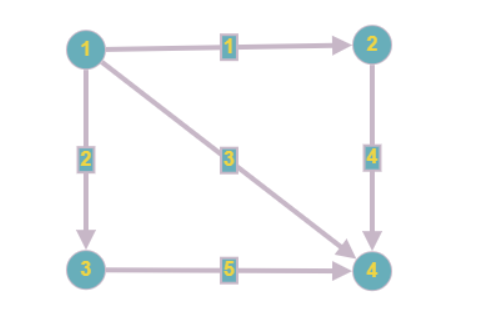
Путь: 48

Маршрут: Владивосток -> Хабаровск->Москва

Путь: 13

Маршрут: Владивосток->Хабаровск





Путь: 1

Маршрут: 1->2

Путь: 2

Маршрут: 1 ->3

Путь: 3

Маршрут: 1 ->4

Пути нет

Путь: 9

Маршрут: 2 -> 4->3

Путь: 4

Маршрут: 2 ->4

Пути нет

Пути нет

Путь: 5

Маршрут: 3->4

Пути нет

Пути нет

Пути нет

# Вывод

Я научилась реализовывать алгоритм Флойда-Уоршелла.