# 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**

**Алгоритм Флойда-Уоршелла и матрица смежности**

**Вариант 3**

Студент 0301 ————————— Сморжок В. Е.

Преподаватель ————————— Тутуева А.В.

Санкт-Петербург,

2022

Задание: Дан список возможных авиарейсов в текстовом файле в формате:

Город отправления 1;Город прибытия 1;цена прямого перелета 1;цена обратного перелета 1

Город отправления 2;Город прибытия 2;цена перелета 2;цена обратного перелета 1

…

Город отправления N;Город прибытия N;цена перелета N;цена обратного перелета N

 В случае, если нет прямого или обратного рейса, его цена будет указана как N/A (not available)

*Пример данных:*

Санкт-Петербург;Москва;10;20

Москва;Хабаровск;40;35

Санкт-Петербург;Хабаровск;14;N/A

Владивосток;Хабаровск;13;8

Владивосток;Санкт-Петербург;N/A;20

*Задание:* найти наиболее эффективный по стоимости перелет из города ***i*** в город ***j***.

|  |  |
| --- | --- |
| Вариант | Алгоритм и структура данных |
| 1 | алгоритм Дейкстры и списки смежности |
| 2 | алгоритм Беллмана-Форда и матрицу смежности |
| 3 | алгоритм Флойда-Уоршелла и матрицу смежности |

Текст программы:

#include "Header.h"

int main()

{

string text = read();

cout << text<< endl<< endl;

string townDep;

string townArr;

cout << "Enter city of departure:" << endl;

cin >> townDep;

cout << "Enter city of arrival" << endl;

cin >> townArr;

algorithmFloyd(text, townDep,townArr);

}

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

const int invalidPrice = INT\_MAX;

string read()

{

string text = "";

string newText;

ifstream creat;

char ch;

creat.open("Price.txt");

while (!creat.eof()) {

creat.get(ch);

text += ch;

}

creat.close();

newText = text.substr(0, text.length() - 1);

return newText;

}

class List

{

private:

struct Node

{

string name;

Node\* next;

};

Node\* head;

Node\* tail;

public:

List()

{

head = tail = NULL;

}

void add(string name)

{

if (head == NULL)

{

Node\* buffer = new Node;

buffer->name = name;

buffer->next = NULL;

head = tail = buffer;

}

else

{

Node\* buffer = head;

Node\* prev = head;

while (buffer != NULL)

{

prev = buffer;

buffer = buffer->next;

}

buffer = new Node;

buffer->name = name;

buffer->next = NULL;

prev->next = buffer;

tail = buffer;

}

}

bool check(string text)

{

Node\* buffer = head;

bool i = false;

while (buffer != NULL)

{

if (buffer->name == text)

{

i = true;

break;

}

buffer = buffer->next;

}

return i;

}

void print()

{

Node\* buffer = head;

while (buffer != NULL)

{

cout << buffer->name << endl;

buffer = buffer->next;

}

}

Node\* stringToList(string text)

{

int i = 0;

string stroka = "";

while (i != text.length())

{

while (text[i] != ';')

{

if (text[i] <= 90 && text[i] >= 60 || text[i] <= 122 && text[i] >= 97 || text[i] == '-')

{

stroka += text[i];

}

if (text[i] == '\n')

{

stroka = "";

}

i++;

if (i == text.length())

{

break;

}

}

if (i == text.length())

{

break;

}

i++;

if (stroka != "NA" && stroka != "")

{

if (!check(stroka))

add(stroka);

}

stroka = "";

}

Node\* buffer = head;

return buffer;

}

int\*\* doingMatrix(string text)

{

int i = 0;

int\*\* array = new int\* [0];

array = new int\* [maxIndex() + 1];

for (int count = 0; count <= maxIndex(); count++)

array[count] = new int[maxIndex()]; // и пять столбцов

// заполнение массива

for (int count\_row = 0; count\_row <= maxIndex(); count\_row++)

for (int count\_column = 0; count\_column <= maxIndex(); count\_column++)

{

if (count\_row == count\_column)

{

array[count\_row][count\_column] = 0;

}

else array[count\_row][count\_column] = invalidPrice;

}

i = 0;

string stroka = "";

string town1 = "";

int index1 = 0;

string town2 = "";

int index2 = 0;

int length1 = 0;

int length2 = 0;

while (1)

{

while (text[i] != ';')

{

if (text[i] <= 90 && text[i] >= 60 || text[i] <= 122 && text[i] >= 97 || text[i] == '-')

{

stroka += text[i];

}

if (text[i] >= 48 && text[i] <= 57)

{

stroka += text[i];

}

if (text[i] == '\n')

{

break;

}

i++;

if (i == text.length())

{

break;

}

}

if (town1 == "")

{

town1 = stroka;

index1 = index(town1);

stroka = "";

}

else if (town2 == "")

{

town2 = stroka;

index2 = index(town2);

stroka = "";

}

if (stroka != town1 && stroka != town2 && stroka != "NA" && stroka != "")

{

if (length1 == 0)

{

length1 = stoi(stroka);

}

else

{

length2 = stoi(stroka);

}

stroka = "";

}

if (stroka == "NA")

{

if (length1 == 0)

length1 = invalidPrice;

else

length2 = invalidPrice;

stroka = "";

}

if (length1 != 0 && length2 != 0 && index1 != -1 && index2 != -1)

{

array[index1][index2] = length1;

array[index2][index1] = length2;

length1 = length2 = 0;

index1 = index2 = -1;

town1 = town2 = "";

}

i++;

if (i >= text.length())

break;

}

return array;

}

string find(int index)

{

Node\* buffer = head;

int i = 0;

while (buffer != NULL)

{

if (i == index)

{

break;

}

else {

buffer = buffer->next;

i++;

}

}

if (buffer != NULL)

return buffer->name;

}

void printMatrix(int\*\* arrayNew)

{

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

{

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

{

if (arrayNew[i][j] != invalidPrice)

cout << arrayNew[i][j] << " ";

else cout << "N/A ";

}

cout << endl;

}

cout << endl;

}

int index(string text)

{

Node\* buffer = head;

int i = 0;

while (buffer != NULL)

{

if (buffer->name == text)

{

break;

}

buffer = buffer->next;

i++;

}

return i;

}

int maxIndex()

{

Node\* buffer = head;

int i = 0;

while (buffer != NULL)

{

buffer = buffer->next;

i++;

}

return i - 1;

}

};

string printWay(List list, int\*\* way, int\*\* arrayNew, string townDep, string townArr)

{

int index1 = list.index(townDep);

int index2 = list.index(townArr);

string Way = "";

if (index1 == index2)

{

throw invalid\_argument("Incorrect way. City ??names match");

}

if (way[index1][index2] != 0)

{

Way += list.find(index1);

Way += "->";

Way += list.find(way[index1][index2] - 1);

Way += "->";

Way += list.find(index2);

}

else

{

Way += list.find(index1);

Way += "->";

Way += list.find(index2);

}

cout << "The cost of the flight is " << arrayNew[index1][index2] << endl;

return Way;

}

int\*\* algorithmFloyd(string text, string townDep, string townArr)

{

List list;

list.stringToList(text);

int\*\* array = list.doingMatrix(text);

cout << "Adjacency matrix:" << endl;

list.printMatrix(array);

int\*\* arrayNew = array;

int\*\* way = new int\* [0];

way = new int\* [list.maxIndex() + 1];

for (int i = 0; i <= list.maxIndex(); i++)

way[i] = new int[list.maxIndex()];

for (int i = 0; i <= list.maxIndex(); i++)

for (int j = 0; j <= list.maxIndex(); j++)

way[i][j] = 0;

for (int k = 0; k <= list.maxIndex(); k++) { //Пробегаемся по всем вершинам и ищем более короткий путь через вершину k

for (int i = 0; i <= list.maxIndex(); i++) {

for (int j = 0; j <= list.maxIndex(); j++) {

if (arrayNew[i][k] != invalidPrice && arrayNew[k][j] != invalidPrice)

{

if (arrayNew[i][k] + arrayNew[k][j] < arrayNew[i][j])

{

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

way[i][j] = way[i][j] \* 10 + k + 1;

}

}

}

}

}

cout << "Matrix processed by floyd's algorithm" << endl;

list.printMatrix(arrayNew);

cout << "The way: " << printWay(list, way, arrayNew, townDep, townArr);

return arrayNew;

}

**Текст Unit-тестов:**

**Краткое описание реализуемого алгоритма и используемых структур данных:**

1. class List – касс, содержащий в себе односвязный список, который хранит названия городов. В нем содержатся различные методы, которые помогают, например, определить индекс элемента по названию для составления матрицы смежности. Другие методы описаны далее.

2. stringToList(string text) – функция, которая обрабатывает поданную на вход строку, выписывает все названия городов в односвязный список без повторений с помощью метода add(). Для исключения повторений названий городов была создана функция check().

3. print() – печатает односвязный список

4. maxIndex() – находит максимальный индекс, то есть количество элементов всего списка-1

5. find – находит элемент списка по индексу и возвращает его имя

6. doingMatrix – обрабатывает строку, проверяя ее на наличие в списке. Получая индексы, она записывает значения в матрицу. Это и является матрицей смежности

7. printMatrix – печатает матрицу, поданную на вход

8. algorithmFloyd – вызывает все вышеназванные функции для получения матрицы смежности, в результате обработки которой выполняется алгоритм Флойда и получается матрица кратчайших расстояний. Также параллельно заполняется матрица путей.

9. printWay - предыдущий алгоритм передает в printWay матрицу путей, которая выводится на экран и возвращает строку запрашиваемого пути. Оценки временной сложности реализуемых алгоритмов:

Описание реализуемых Unit-тестов:

1. TestMethodAlgorithmFloyd – вводит данные об искомом маршруте перелета, вызывает алгортим флойда, затем сравнивает введенную матрицу с матрицей после алгоритма флойда

2. TestMethodDoMatrix – вводится строка, которая обрабатывается, а затем вводится в матрицу, которая впоследствии сравнивается с заданной матрицей

3. TestMethodMaxIndex - обрабатывается строка, вызывается функция maxIndex, которая возвращает максимальный индекс городов. Он сравнивается с заданным индексом. Это и является размерностью матрицы+1

4. TestMethodIndex - обрабатывается строка, вызывается функция index, которая возвращает индекс искомого города. Он сравнивается с заданным индексом

5. TestMethodFind – обрабатывается строка, вызывается функция find, которая возвращает название искомого города. Он сравнивается с заданным названием

**Оценка временной сложности каждого метода:**

1. stringToList – O(n^2)
2. maxIndex() – O(n)
3. print() – O(n)
4. find – O(n)
5. printMatrix– O(n^2)
6. algorithmFloyd– O(n^3)
7. printWay – O(1)







