Національний технічний університет України

«Київський політехнічний інститут»

Кафедра автоматизованих систем обробки інформації та управління

Звіт про виконання лабораторної роботи №8 з дисципліни

«Алгоритми та структури даних»

Перевірено: Ковалюк Т.В. Виконав ст. 1 курсу ФОІТ

Проскура С.Л. гр. ІС-52

Дорошенко А.В.

Київ 2016

## Лабораторна робота 8

# **Алгоритми на графах.**

Варіант № 9

*Мета:*

1. Вивчити особливості використання  структур типу графів.

2. Навчитися застосовувати структури типу графів у практичних задачах.

3. Опанувати алгоритми:

* Дейкстри для орієнтованого графа
* Форда-Беллмана
* Флойда-Уоршалла
* Данцига

### Завдання

|  |  |
| --- | --- |
| Задані N міст c номерами від 1 до N і мережа з M доріг з одностороннім рухом між ними. Кожна дорога задається трійкою (i, j, k), де i - номер міста, в якому дорога починається, j - номер міста, в якому дорога закінчується, k - її довжина (число k - натуральне). Дороги один з одним можуть перетинатися тільки в кінцевих містах. Усі шляхи між двома вказаними містами A і B можна упорядкувати в список за зростанням їх довжин (якщо є декілька шляхів однакової довжини, то вибираємо один з них). Необхідно знайти один з шляхів, який може бути другим в списку. Вивести його довжину і міста, через які він проходить. |  |

### Блок-схема алгоритму



Рис.1 (Блок-схема головного алгоритму)



Рис.2 (Блок-схема алгоритму зчитування з файлу)



Рис.3 (Блок-схема алгоритму побудови матриці суміжності)



Рис.4 (Блок-схема алгоритму Дейкстри)



Рис.5 (Блок-схема алгоритму знаходження всіх найкоротших шляхів)

### Код програми

//Laboratory work

//written by student of the first curse

//of the group IC-52

//Anton Doroshenko

//2016.05.04

//==========================================================================

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <iostream>

#include <iomanip>

#include <vector>

#include <algorithm>

using namespace std;

const int inf = 1000;

struct graph

{

int v = 0; //вершини графа

int u = 0; //ребра графу

int MatrixSecond[100][100]; //матриця суміжності

int MatrixDis[100][100]; //матриця відстаней

int MatrixAtt[100][100]; //матриця досяжності

};

graph Graph;

//== Для зчитування ==

int Arr[2];

int Array[10];

int weight = 0;

FILE \*f;

char FileName[100];

int Matrix[100][100];

int N = 0;

int\* PtrN = &N;

//== Для Дейкстри ==

int color[100];

int flags[100];

int way[100];

int t = 0;

int startV = 0;

int endV = 0;

bool negative = false;

int tempWay[100];

//==зчитування кількості вершин і ребер

void ReadNodesEdges()

{

for (int i = 0; i < 2; i++)

{

fscanf(f, "%d", &Arr[i]);

}

Graph.v = Arr[0];

Graph.u = Arr[1];

}

//==побудова матрикі суміжності

void MatrixSum()

{

for (int j = 0; j < Graph.v; j++)

{

for (int k = 0; k < Graph.v; k++)

{

if (j == k)

{

Graph.MatrixSecond[j][k] = 0;

}

else

{

Graph.MatrixSecond[j][k] = inf;

}

}

}

for (int i = 1; i <= Graph.u \* 2; i++)

{

fscanf(f, "%d", &Array[(i - 1) % 2]);

if ((i % 2) == 0)

{

fscanf(f, "%d", &weight);

if (weight < 0)

{

negative = true;

}

Graph.MatrixSecond[Array[0] - 1][Array[1] - 1] = weight;

}

}

}

//==виведення матриці суміжності

void OutputMatrixS()

{

for (int i = 0; i < Graph.v; i++)

{

for (int j = 0; j < Graph.v; j++)

{

if (Graph.MatrixSecond[i][j] == inf)

{

cout << setw(3) << "-";

}

else

{

cout << setw(3) << Graph.MatrixSecond[i][j];

}

}

cout << endl;

}

}

//==зчитування файла

void ReadFile()

{

//cout << "Enter file name" << endl;

//cin >> FileName;

//f = fopen(FileName, "r");

f = fopen("g2.txt", "r");

/\*f = fopen("graph\_03\_1.txt", "r");\*/

if (f != NULL)

{

ReadNodesEdges();

cout << Graph.v << " " << Graph.u << endl;

MatrixSum();

cout << "Second matrix" << endl;

OutputMatrixS();

system("pause");

}

else

{

cout << "File not open" << endl;

}

fclose(f);

}

void remResult()

{

int arr[100];

int i = endV;

int j = 0;

int index = 1;

while (i != startV)

{

arr[j] = i;

i = way[i];

j++;

}

//cout << startV + 1;

tempWay[0] = flags[endV];

tempWay[index] = startV;

index++;

for (int k = (j - 1); k >= 0; k--)

{

//cout << "->" << arr[k] + 1;

tempWay[index] = arr[k];

index++;

}

//cout << endl;

}

bool diferent()

{

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

{

if (flags[endV] == Matrix[i][0])

{

return false;

}

}

return true;

}

void dijkstra()

{

bool flag = true;

int x = 0;

int min;

for (int i = 0; i < Graph.v; i++)

{

flags[i] = inf;

color[i] = 0;

}

flags[startV] = 0;

color[startV] = 1;

x = startV;

while (color[endV] == 0)

{

for (int j = 0; j < Graph.v; j++)

{

if ((color[j] == 0) && ((flags[x] + Graph.MatrixSecond[x][j]) < flags[j]))

{

flags[j] = flags[x] + Graph.MatrixSecond[x][j];

way[j] = x;

}

}

min = inf;

for (int k = 0; k < Graph.v; k++)

{

if ((color[k] == 0) && (flags[k] < min))

{

min = flags[k];

x = k;

}

}

if (min == inf)

{

flag = false;

break;

}

else

{

color[x] = 1;

}

}

if (flag && diferent())

{

remResult();

N++;

}

}

void searchAllWays(int deapth)

{

deapth++;

int temp = 0;

for (int i = 0; i <= Graph.v; i++)

{

if (deapth == 1)

{

for (int j = 0; j < 100; j++)

{

Matrix[j][i] = inf;

}

}

tempWay[i] = inf;

}

dijkstra();

for (int k = 0; k <= Graph.v; k++)

{

if (tempWay[k] == inf)

{

break;

}

else

{

Matrix[N - 1][k] = tempWay[k];

}

}

for (int l = 1; l <= Graph.v; l++)

{

if (Matrix[deapth - 1][l] == inf)

{

break;

}

else

{

if (Matrix[deapth - 1][l + 1] == inf || (l + 1) == Graph.v)

{

break;

}

else

{

temp = Graph.MatrixSecond[Matrix[deapth - 1][l]][Matrix[deapth - 1][l + 1]];

Graph.MatrixSecond[Matrix[deapth - 1][l]][Matrix[deapth - 1][l + 1]] = inf;

searchAllWays(deapth);

Graph.MatrixSecond[Matrix[deapth - 1][l]][Matrix[deapth - 1][l + 1]] = temp;

}

}

}

}

void outputResult()

{

int ind = 0;

int index = 0;

int min = Matrix[0][0];

int second = Matrix[0][0];

for (int i = 1; i < inf; i++)

{

if (Matrix[i][0] == inf)

{

break;

}

else

{

if (min > Matrix[i][0])

{

min = Matrix[i][0];

}

}

}

if (Matrix[0][0] != min)

{

second = Matrix[0][0];

}

else

{

second = Matrix[1][0];

index = 1;

}

for (int j = 1; j < inf; j++)

{

if (Matrix[j][0] == inf)

{

break;

}

else

{

if (second > Matrix[j][0] && Matrix[j][0] != min)

{

second = Matrix[j][0];

index = j;

}

}

}

if (second == min)

{

cout << "There are not second way" << endl;

}

else

{

cout << "Legth of way: " << Matrix[index][0] << endl;

cout << "Way:" << endl;

if (Matrix[index][1] != inf)

{

cout << Matrix[index][1] + 1;

}

for (int k = 2; k < Graph.v; k++)

{

if (Matrix[index][k] == inf)

{

break;

}

else

{

cout << "->" << Matrix[index][k] + 1;

}

}

cout << endl;

}

}

int main()

{

cout << "laboratory work number 8 made by Anton Doroshenko, IS-52" << endl;

cout << endl;

ReadFile();

system("cls");

cout << "Enter start vertex: " << endl;

cin >> startV;

startV--;

cout << "Enter end vertex: " << endl;

cin >> endV;

endV--;

searchAllWays(0);

outputResult();

system("pause");

}

1. **Screen Shot результатів**

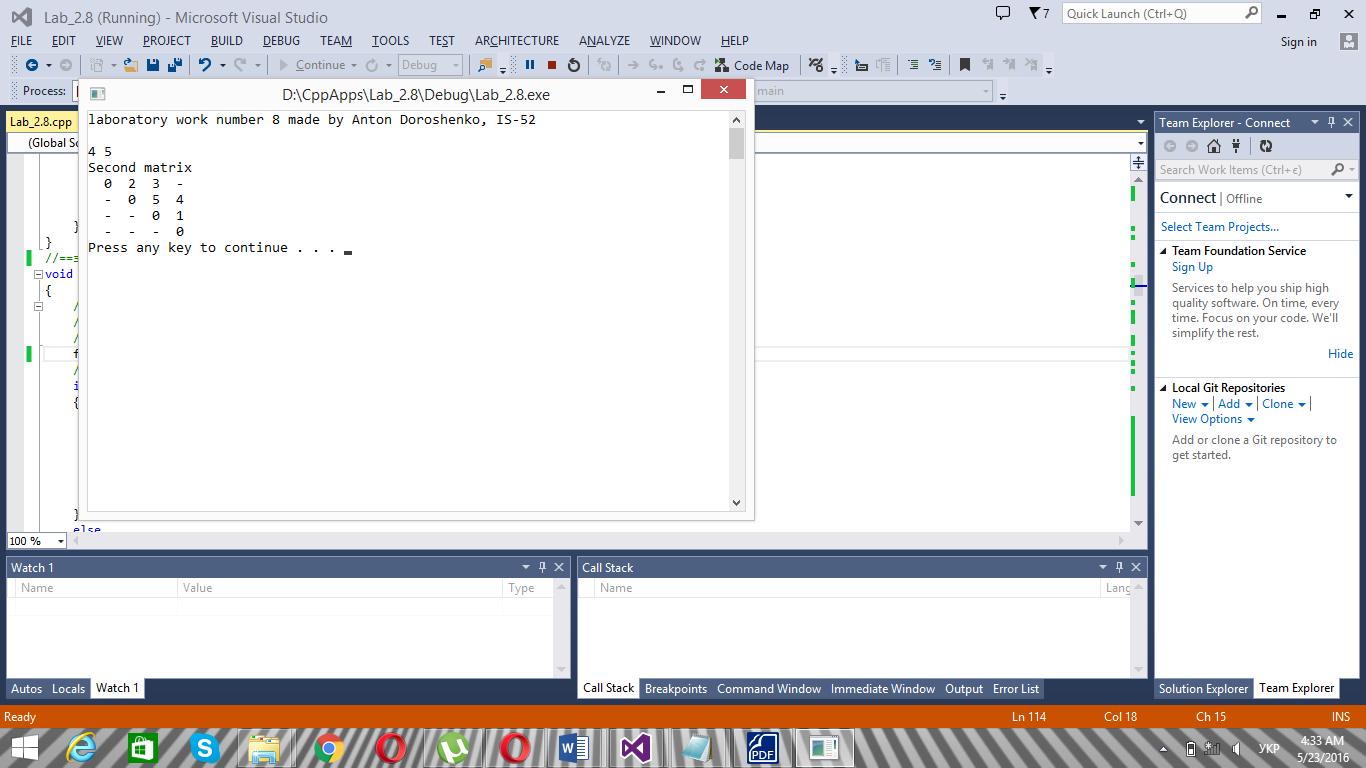


Рис.9 (Screen Shot результатів 1)

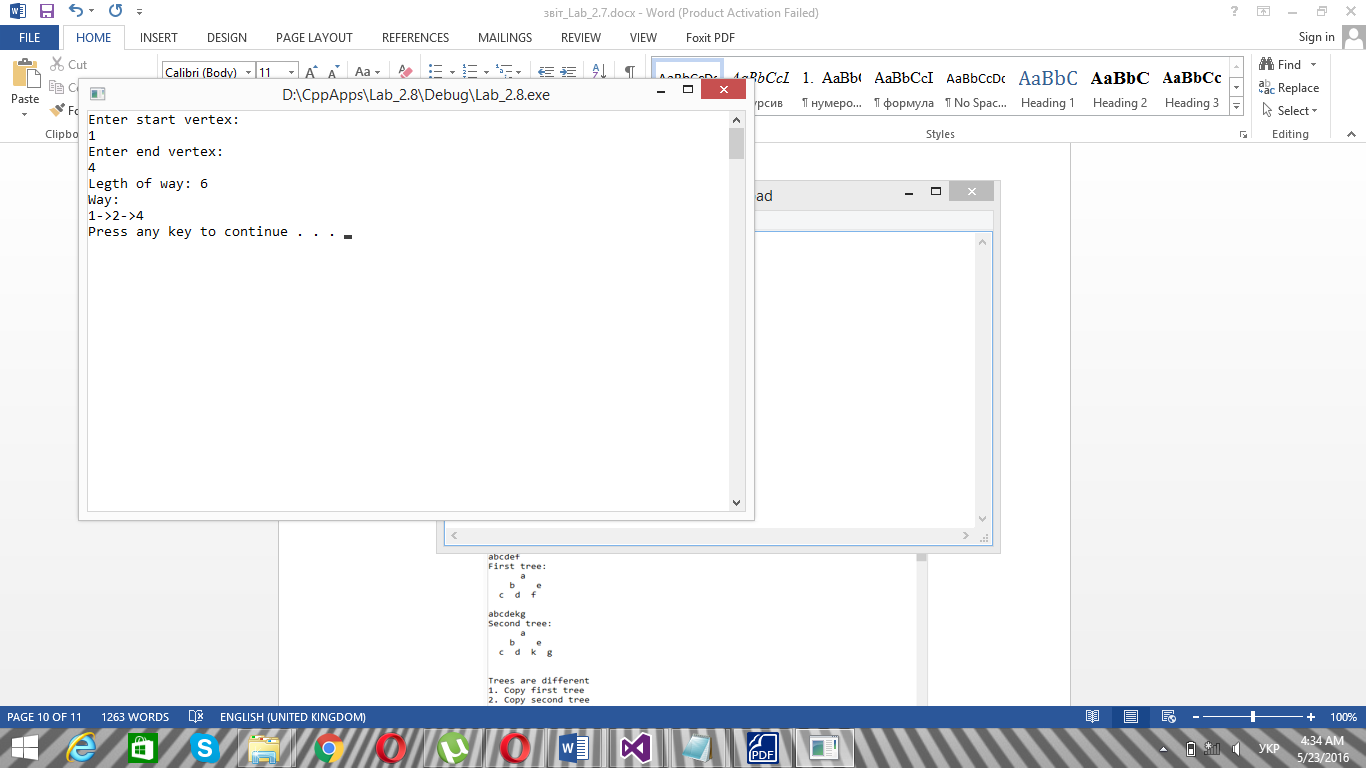


Рис.6 (Screen Shot результатів 2)

1. **Аналіз результатів**

Як ми бачимо зі ScreenShot’ів

Програма зчитує дані про міста та дороги що їх сполучають та виводить матрицю суміжності. Програма виводить другий по довжині шлях та його довжину.

1. **Висновок**

Програма працює правильно, про що свідчить аналіз результатів та ScreenShot’и. Програма коректно виводить результати. Програма застосовує алгоритм Дейкстри.