Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

ЗВІТ

про виконання лабораторної роботи № 5(2)

з дисципліни

“Дискретні структури”

Тема:

Дискретна математика. Теорія графів

АЛГОРИТМИ БЕЛМАНА-ФОРДА ТА ДЖОНСОНА

Виконав:

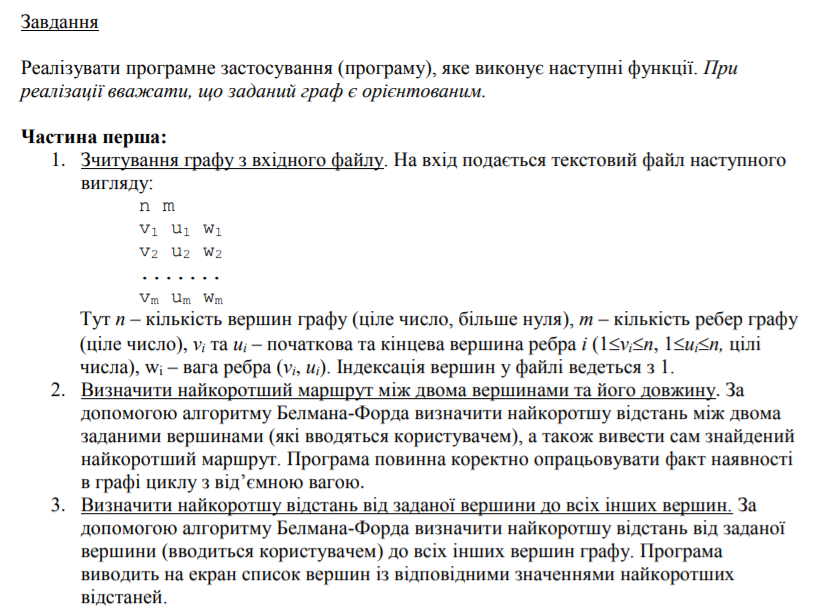
студент 1-го курсу

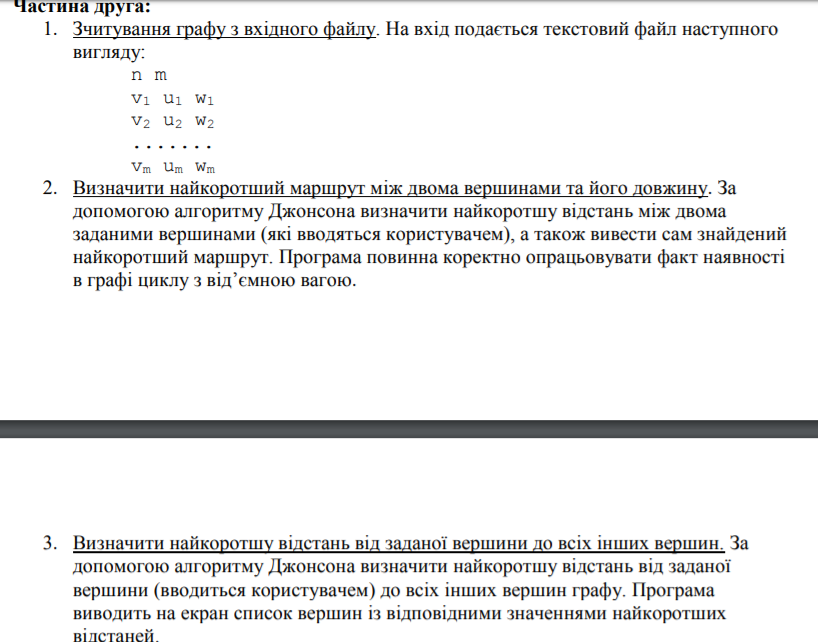
групи ІП-92 ФІОТ

Онищук Микола

Олександрович

Київ-2020





**Код:**

#include <iostream>

#include <iomanip>

#include <fstream>

#include <stack>

using namespace std;

void mainMenu(double\*\*, int, int);

void BellmanFordAlgorithm(double\*\*, int, int);

void BellmanFordAlgorithm1(double\*\*, int, int);

void BellmanFordAlgorithm2(double\*\*, int, int);

void BellmanFordAlgorithmJ(double\*\*, int, int, double\*&);

void JohnsonAlgorithm(double\*\*, int, int);

void JohnsonAlgorithm1(double\*\*, int, int);

void JohnsonAlgorithm2(double\*\*, int, int);

void DijkstraAlgorithm1(double\*\*, int, int, double\*&, int\*&);

void DijkstraAlgorithm2(double\*\*, int, int, double\*&);

double\*\* adjacencyMatrixGenOr(double\*\*, int, int);

void pathOutput(int\*, int, int);

int main()

{

int n = 0;

int m = 0;

ifstream in("E:\\input9.txt");

in >> n >> m;

double\*\* edges = new double\* [m];

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

{

edges[i] = new double[3];

}

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

{

for (int j = 0; j < 3; j++)

{

in >> edges[i][j];

}

}

mainMenu(edges, n, m);

system("pause");

}

void mainMenu(double\*\* edges, int n, int m)

{

bool option = true;

cout << "Press 0 to run Bellman-Ford algorithm\n";

cout << "Press 1 to run Johnson algorithm\n";

cin >> option;

system("cls");

if (option == 0)

{

BellmanFordAlgorithm(edges, n, m);

}

else {

JohnsonAlgorithm(edges, n, m);

}

}

void BellmanFordAlgorithm(double\*\* edges, int n, int m)

{

cout << "Press 1 to find the shortest way between two vertexes\n";

cout << "Press 2 to find the shortest ways from one vertex\n";

cout << endl << endl << "Press 9 to go back to the main menu\n";

int q;

cin >> q;

if (q == 1)

{

system("cls");

BellmanFordAlgorithm1(edges, n, m);

cout << endl << endl << "If you want to go back press 9\n";

cin >> q;

if (q == 9)

{

system("cls");

BellmanFordAlgorithm(edges, n, m);

}

}

else if (q == 2)

{

system("cls");

BellmanFordAlgorithm2(edges, n, m);

cout << endl << endl << "If you want to go back press 9\n";

cin >> q;

if (q == 9)

{

system("cls");

BellmanFordAlgorithm(edges, n, m);

}

}

else if (q == 9)

{

system("cls");

mainMenu(edges, n, m);

}

}

void BellmanFordAlgorithm1(double\*\* edges, int n, int m)

{

double\*\* adjacencyMatrix = new double\* [n];

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

{

adjacencyMatrix[i] = new double[n];

}

int\* path = new int[n];

adjacencyMatrix = adjacencyMatrixGenOr(edges, n, m);

int start = 0;

int finish = 0;

cout << "Enter the vertexes\n";

cin >> start;

cin >> finish;

cout << endl;

double\*\* BellmanFordMatrix = new double\* [n];

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

{

BellmanFordMatrix[i] = new double[n];

}

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

{

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

{

BellmanFordMatrix[i][j] = INFINITY;

}

}

BellmanFordMatrix[0][start - 1] = 0;

int i = 1;

bool end = true;

do {

end = true;

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

{

BellmanFordMatrix[i][j] = BellmanFordMatrix[i - 1][j];

}

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

{

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

{

if (adjacencyMatrix[j][k] != INFINITY && BellmanFordMatrix[i][k] > BellmanFordMatrix[i - 1][j] + adjacencyMatrix[j][k] && start - 1 != k)

{

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

path[k] = j + 1;

}

}

}

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

{

if (BellmanFordMatrix[i][j] != BellmanFordMatrix[i - 1][j])

{

end = false;

}

}

i++;

} while (i < n && end == false);

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

{

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

{

if (BellmanFordMatrix[i - 1][k] > BellmanFordMatrix[i - 1][j] + adjacencyMatrix[j][k])

{

cout << "Error! The graph contains a negative value cycle!\n";

return;

}

}

}

if (BellmanFordMatrix[i - 1][finish - 1] != INFINITY)

{

cout << "The shortest way from " << start << " to " << finish << " is " << BellmanFordMatrix[i - 1][finish - 1] << endl;

}

else {

cout << "There is no path from " << start << " to " << finish << endl;

}

if (BellmanFordMatrix[i - 1][finish - 1] != INFINITY)

{

pathOutput(path, start, finish);

}

}

void BellmanFordAlgorithm2(double\*\* edges, int n, int m)

{

double\*\* adjacencyMatrix = new double\* [n];

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

{

adjacencyMatrix[i] = new double[n];

}

adjacencyMatrix = adjacencyMatrixGenOr(edges, n, m);

int start = 0;

cout << "Enter the starting vertex\n";

cin >> start;

cout << endl;

double\*\* BellmanFordMatrix = new double\* [n];

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

{

BellmanFordMatrix[i] = new double[n];

}

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

{

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

{

BellmanFordMatrix[i][j] = INFINITY;

}

}

BellmanFordMatrix[0][start - 1] = 0;

int i = 1;

bool end = true;

do {

end = true;

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

{

BellmanFordMatrix[i][j] = BellmanFordMatrix[i - 1][j];

}

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

{

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

{

if (adjacencyMatrix[j][k] != INFINITY && (BellmanFordMatrix[i][k] > BellmanFordMatrix[i - 1][j] + adjacencyMatrix[j][k]) && start - 1 != k)

{

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

}

}

}

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

{

if (BellmanFordMatrix[i][j] != BellmanFordMatrix[i - 1][j])

{

end = false;

}

}

i++;

} while (i < n && end == false);

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

{

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

{

if (BellmanFordMatrix[i - 1][k] > BellmanFordMatrix[i - 1][j] + adjacencyMatrix[j][k])

{

cout << "Error! The graph contains a negative value cycle!\n";

return;

}

}

}

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

{

if (BellmanFordMatrix[i - 1][j] != INFINITY && j != start - 1)

{

cout << "The shortest way from " << start << " to " << j + 1 << " is " << BellmanFordMatrix[i - 1][j] << endl;

}

else if (j != start - 1) {

cout << "There is no path from " << start << " to " << j + 1 << endl;

}

}

}

void BellmanFordAlgorithmJ(double\*\* edges, int n, int m, double\*& sLength)

{

double\*\* adjacencyMatrix = new double\* [n];

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

{

adjacencyMatrix[i] = new double[n];

}

adjacencyMatrix = adjacencyMatrixGenOr(edges, n, m);

int start = n;

cout << endl;

double\*\* BellmanFordMatrix = new double\* [n];

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

{

BellmanFordMatrix[i] = new double[n];

}

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

{

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

{

BellmanFordMatrix[i][j] = INFINITY;

}

}

BellmanFordMatrix[0][start - 1] = 0;

int i = 1;

bool end = true;

do {

end = true;

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

{

BellmanFordMatrix[i][j] = BellmanFordMatrix[i - 1][j];

}

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

{

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

{

if (adjacencyMatrix[j][k] != INFINITY && (BellmanFordMatrix[i][k] > BellmanFordMatrix[i - 1][j] + adjacencyMatrix[j][k]) && start - 1 != k)

{

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

}

}

}

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

{

if (BellmanFordMatrix[i][j] != BellmanFordMatrix[i - 1][j])

{

end = false;

}

}

i++;

} while (i < n && end == false);

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

{

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

{

if (BellmanFordMatrix[i - 1][k] > BellmanFordMatrix[i - 1][j] + adjacencyMatrix[j][k])

{

cout << "Error! The graph contains a negative value cycle!\n";

return;

}

}

}

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

{

sLength[j] = BellmanFordMatrix[i - 1][j];

}

}

void JohnsonAlgorithm(double\*\* edges, int n, int m)

{

cout << "Press 1 to find the shortest way between two vertexes\n";

cout << "Press 2 to find the shortest ways from one vertex\n";

cout << endl << endl << "Press 9 to go back to the main menu\n";

int q;

cin >> q;

if (q == 1)

{

system("cls");

JohnsonAlgorithm1(edges, n, m);

cout << endl << endl << "If you want to go back press 9\n";

cin >> q;

if (q == 9)

{

system("cls");

JohnsonAlgorithm(edges, n, m);

}

}

else if (q == 2)

{

system("cls");

JohnsonAlgorithm2(edges, n, m);

cout << endl << endl << "If you want to go back press 9\n";

cin >> q;

if (q == 9)

{

system("cls");

JohnsonAlgorithm(edges, n, m);

}

}

else if (q == 9)

{

system("cls");

mainMenu(edges, n, m);

}

}

void JohnsonAlgorithm1(double\*\* edges, int n, int m)

{

int\* path = new int[n];

int start = 0;

int finish = 0;

cout << "Enter the vertexes\n";

cin >> start;

cin >> finish;

int n1 = n + 1;

int m1 = m + n;

double\*\* edges1 = new double\* [m1];

for (int i = 0; i < m1; i++)

{

edges1[i] = new double[3];

}

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

{

for (int j = 0; j < 3; j++)

{

edges1[i][j] = edges[i][j];

}

}

for (int i = m; i < m1; i++)

{

for (int j = 0; j < 3; j++)

{

edges1[i][0] = n1;

edges1[i][1] = i - m + 1;

edges1[i][2] = 0;

}

}

double\* sLength = new double[m1];

BellmanFordAlgorithmJ(edges1, n1, m1, sLength);

double\*\* adjacencyMatrix = new double\*;

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

{

adjacencyMatrix[n] = new double[n];

}

adjacencyMatrix = adjacencyMatrixGenOr(edges, n, m);

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

{

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

{

adjacencyMatrix[i][j] = adjacencyMatrix[i][j] + sLength[i] - sLength[j];

}

}

double\* distanceArray = new double[n];

DijkstraAlgorithm1(adjacencyMatrix, n, start, distanceArray, path);

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

{

distanceArray[i] = distanceArray[i] - sLength[start - 1] + sLength[i];

}

if (distanceArray[finish - 1] != INFINITY)

{

cout << "The shortest path from " << start << " to " << finish << " is " << distanceArray[finish - 1] << endl;

}

else

{

cout << "There is no path between " << start << " and " << finish << endl;

}

if (distanceArray[finish - 1] != INFINITY)

{

pathOutput(path, start, finish);

}

}

void JohnsonAlgorithm2(double\*\* edges, int n, int m)

{

int start = 0;

cout << "Enter the starting vertex\n";

cin >> start;

int n1 = n + 1;

int m1 = m + n;

double\*\* edges1 = new double\* [m1];

for (int i = 0; i < m1; i++)

{

edges1[i] = new double[3];

}

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

{

for (int j = 0; j < 3; j++)

{

edges1[i][j] = edges[i][j];

}

}

for (int i = m; i < m1; i++)

{

for (int j = 0; j < 3; j++)

{

edges1[i][0] = n1;

edges1[i][1] = i - m + 1;

edges1[i][2] = 0;

}

}

double\* sLength = new double[m1];

BellmanFordAlgorithmJ(edges1, n1, m1, sLength);

double\*\* adjacencyMatrix = new double\*;

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

{

adjacencyMatrix[n] = new double[n];

}

adjacencyMatrix = adjacencyMatrixGenOr(edges, n, m);

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

{

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

{

adjacencyMatrix[i][j] = adjacencyMatrix[i][j] + sLength[i] - sLength[j];

}

}

double\* distanceArray = new double[n];

DijkstraAlgorithm2(adjacencyMatrix, n, start, distanceArray);

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

{

distanceArray[i] = distanceArray[i] - sLength[start - 1] + sLength[i];

}

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

{

if (i != start - 1 && distanceArray[i] != INFINITY)

{

cout << "The shortest path from " << start << " to " << i + 1 << " is " << distanceArray[i] << endl;

}

else if (i != start - 1)

{

cout << "There is no path between " << start << " and " << i + 1 << endl;

}

}

}

void DijkstraAlgorithm1(double\*\* adjacencyMatrix, int n, int start, double\*& distanceArray, int\*& path)

{

bool\* constVert = new bool[n];

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

{

constVert[i] = false;

}

cout << endl;

double\*\* DijkstraMatrix = new double\* [n];

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

{

DijkstraMatrix[i] = new double[n];

}

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

{

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

{

DijkstraMatrix[i][j] = INFINITY;

}

}

DijkstraMatrix[0][start - 1] = 0;

constVert[start - 1] = true;

int prevVertex = start - 1;

double prevValue = 0;

double minValue = 0;

int minVertex = 0;

int count = 0;

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

{

count = 0;

minValue = DijkstraMatrix[i][0];

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

{

if (adjacencyMatrix[prevVertex][j] != INFINITY && adjacencyMatrix[prevVertex][j] + prevValue < DijkstraMatrix[i - 1][j] && constVert[j] == false)

{

DijkstraMatrix[i][j] = adjacencyMatrix[prevVertex][j] + prevValue;

path[j] = prevVertex + 1;

}

else if (constVert[j] == false)

{

DijkstraMatrix[i][j] = DijkstraMatrix[i - 1][j];

}

}

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

{

if (DijkstraMatrix[i][j] < minValue)

{

minValue = DijkstraMatrix[i][j];

minVertex = j;

count++;

}

}

if (count == 0)

{

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

{

if (constVert[j] == false)

{

minVertex = j;

break;

}

}

constVert[minVertex] = true;

distanceArray[minVertex] = INFINITY;

}

else

{

distanceArray[minVertex] = minValue;

}

prevValue = minValue;

prevVertex = minVertex;

constVert[minVertex] = true;

}

}

void DijkstraAlgorithm2(double\*\* adjacencyMatrix, int n, int start, double\*& distanceArray)

{

bool\* constVert = new bool[n];

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

{

constVert[i] = false;

}

cout << endl;

double\*\* DijkstraMatrix = new double\* [n];

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

{

DijkstraMatrix[i] = new double[n];

}

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

{

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

{

DijkstraMatrix[i][j] = INFINITY;

}

}

DijkstraMatrix[0][start - 1] = 0;

constVert[start - 1] = true;

int prevVertex = start - 1;

double prevValue = 0;

double minValue = 0;

int minVertex = 0;

int count = 0;

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

{

count = 0;

minValue = DijkstraMatrix[i][0];

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

{

if (adjacencyMatrix[prevVertex][j] != INFINITY && adjacencyMatrix[prevVertex][j] + prevValue < DijkstraMatrix[i - 1][j] && constVert[j] == false)

{

DijkstraMatrix[i][j] = adjacencyMatrix[prevVertex][j] + prevValue;

}

else if (constVert[j] == false)

{

DijkstraMatrix[i][j] = DijkstraMatrix[i - 1][j];

}

}

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

{

if (DijkstraMatrix[i][j] < minValue)

{

minValue = DijkstraMatrix[i][j];

minVertex = j;

count++;

}

}

if (count == 0)

{

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

{

if (constVert[j] == false)

{

minVertex = j;

break;

}

}

constVert[minVertex] = true;

distanceArray[minVertex] = INFINITY;

}

else

{

distanceArray[minVertex] = minValue;

}

prevValue = minValue;

prevVertex = minVertex;

constVert[minVertex] = true;

}

}

double\*\* adjacencyMatrixGenOr(double\*\* edges, int n, int m)

{

double\*\* adjacencyMatrix = new double\* [n];

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

{

adjacencyMatrix[i] = new double[n];

}

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

{

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

{

adjacencyMatrix[i][j] = INFINITY;

}

}

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

{

adjacencyMatrix[static\_cast<int>(edges[i][0] - 1)][static\_cast<int>(edges[i][1] - 1)] = edges[i][2];

}

return adjacencyMatrix;

}

void pathOutput(int\* path, int start, int finish)

{

stack<int> pathStack;

int prev = path[finish - 1];

while (prev != start)

{

pathStack.push(prev);

prev = path[prev - 1];

}

cout << "The path from " << start << " to " << finish << " is " << start << " ";

while (!pathStack.empty())

{

cout << pathStack.top() << " ";

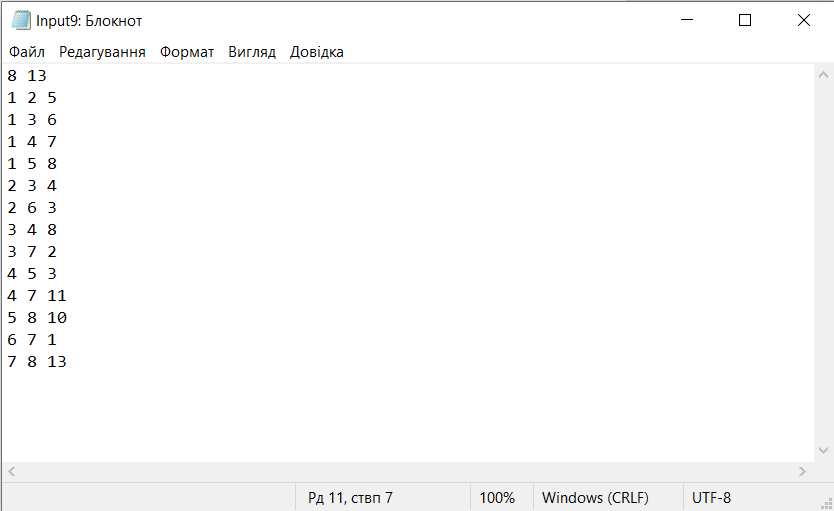
pathStack.pop();

}

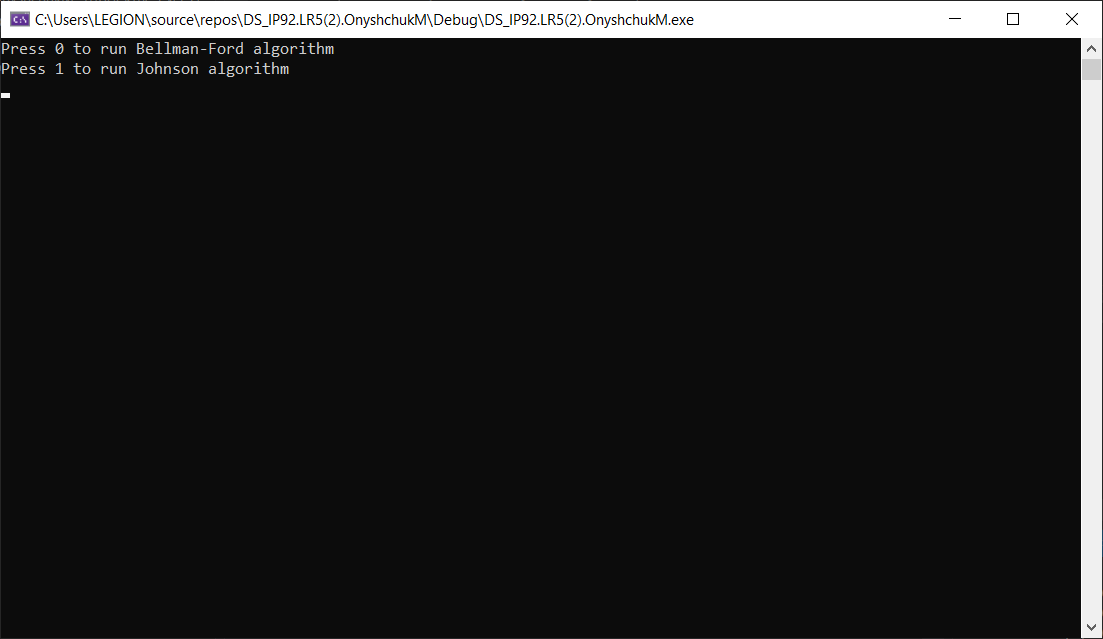
cout << finish << endl;

}

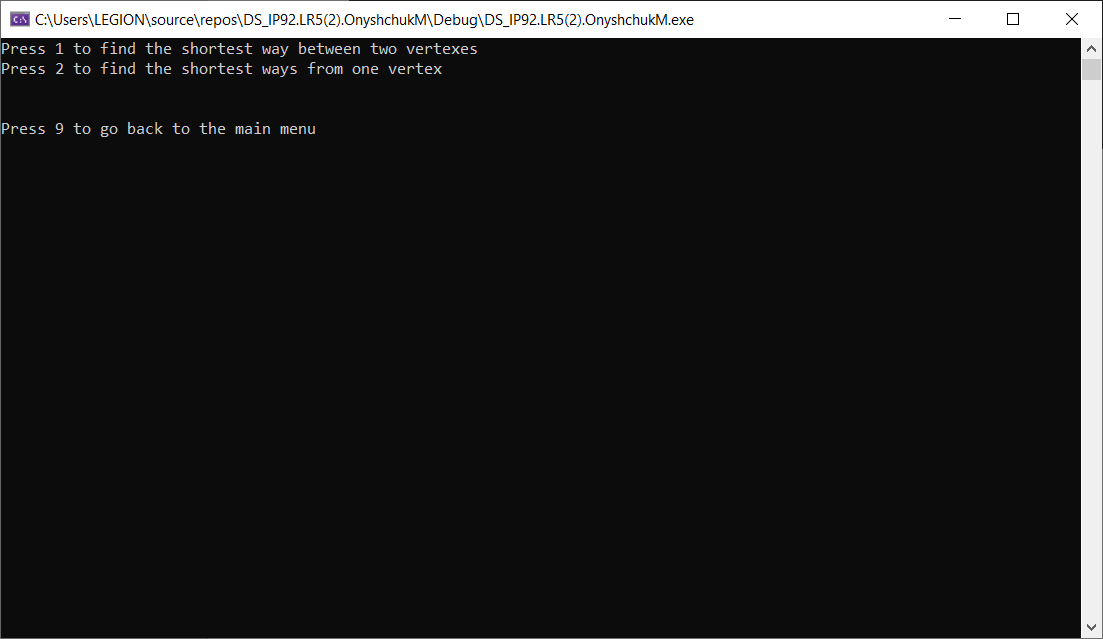
**Вхідний файл:**

****

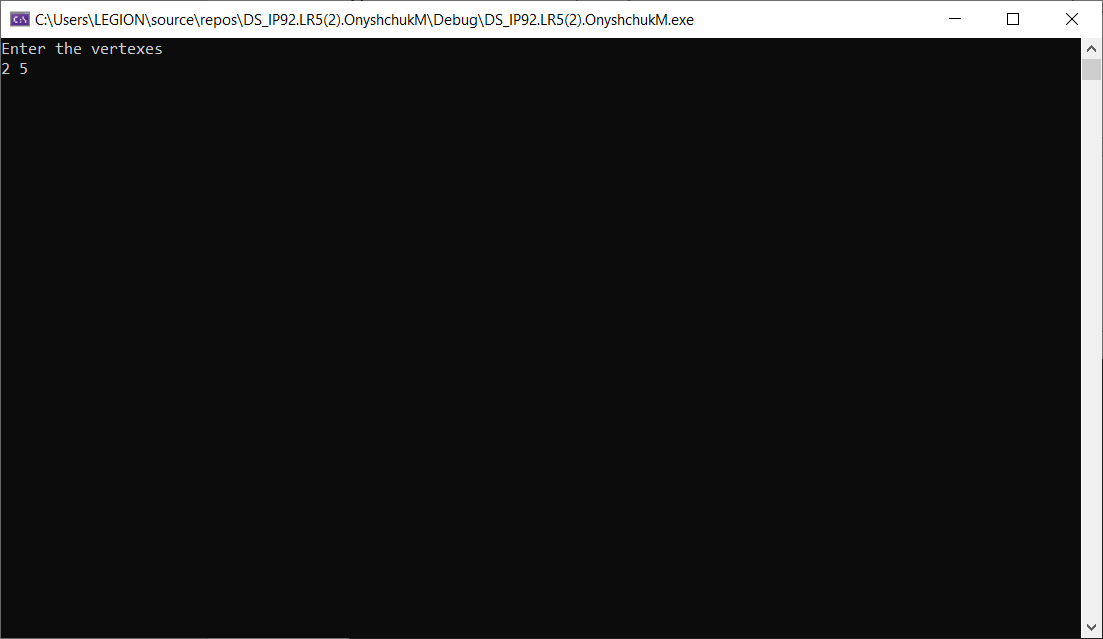
**Результат:**



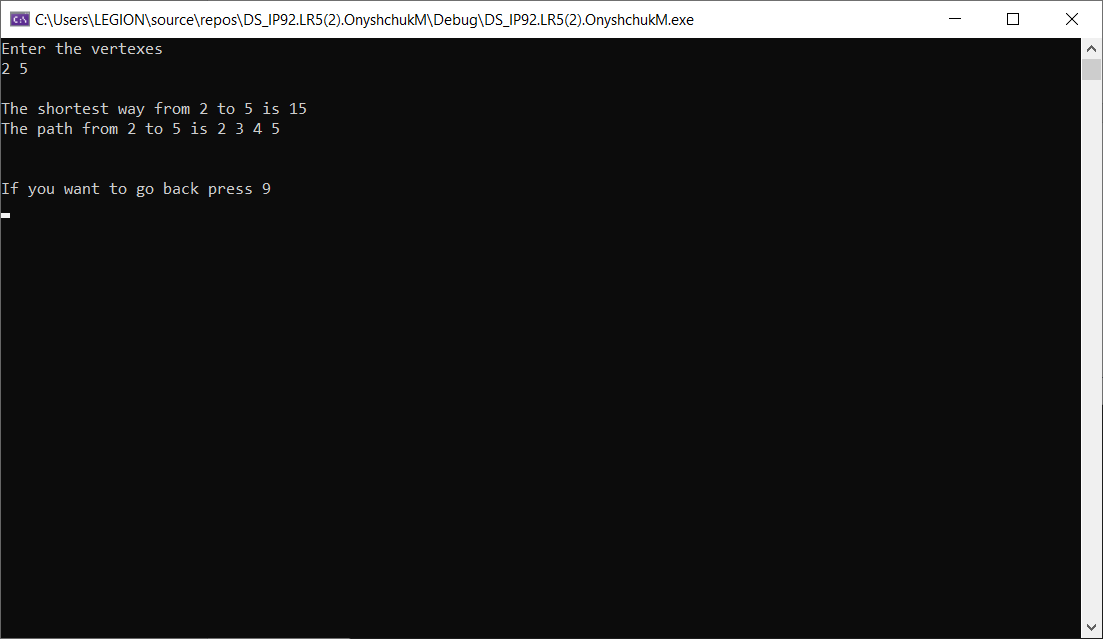
Спочатку в головному меню оберемо алгоритм Белмана-Форда.



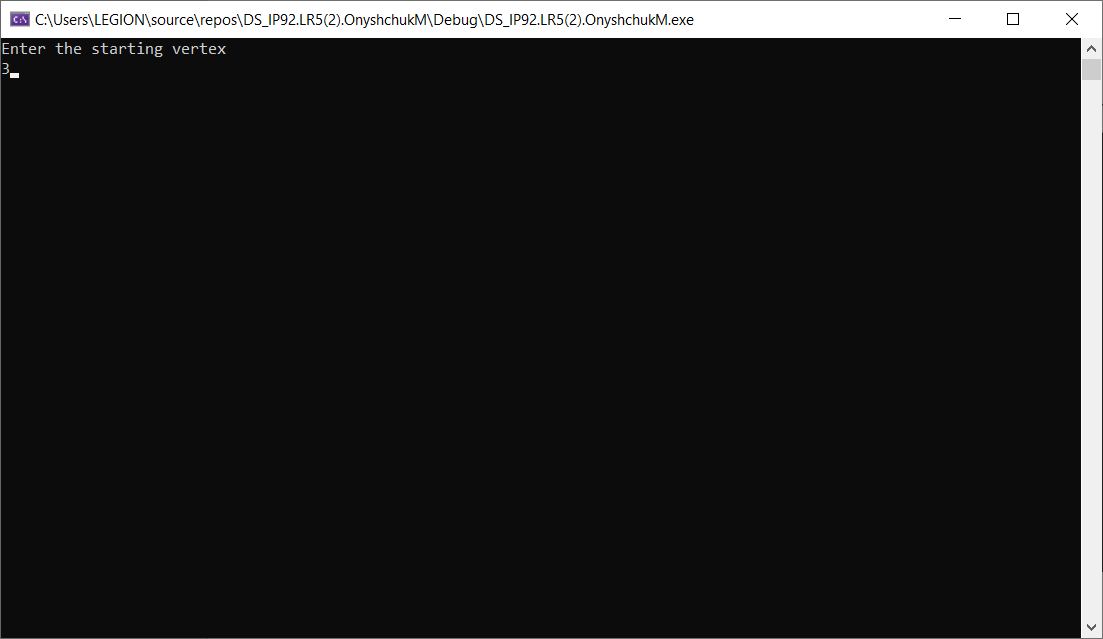
Натиснемо один, щоб побачити найкоротший шлях між двома обраними вершинами.



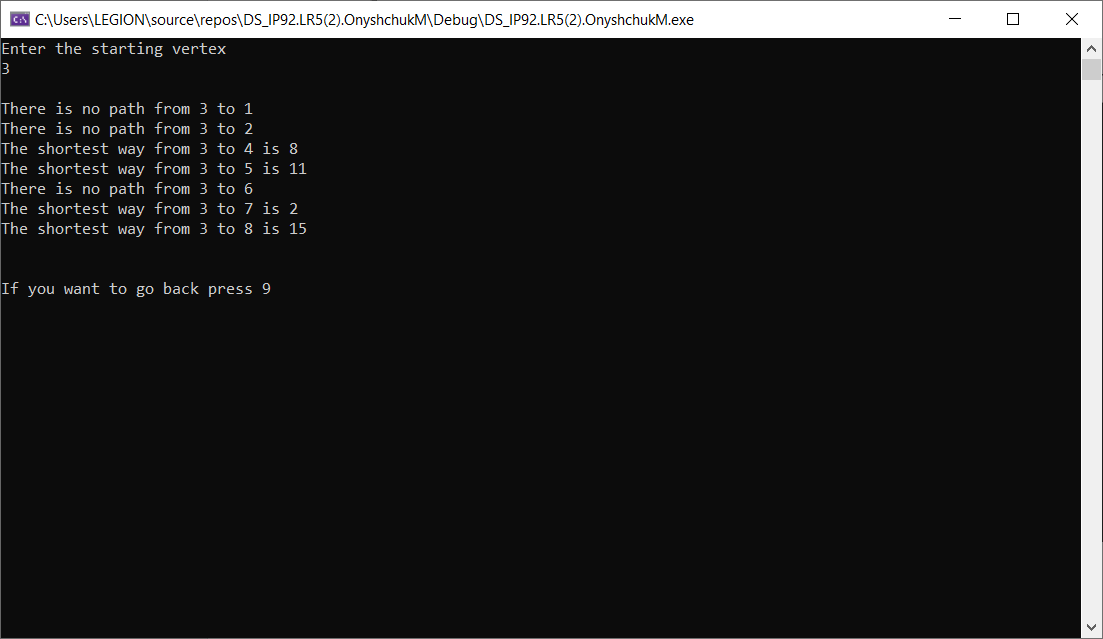
Введемо вершини 2 і 5.



Програма знайшла, що найкоротший шлях – 15 і відтворила маршрут. Повернемось назад і оберемо 2, щоб побачити список відстаней до різних вершин до обраної.

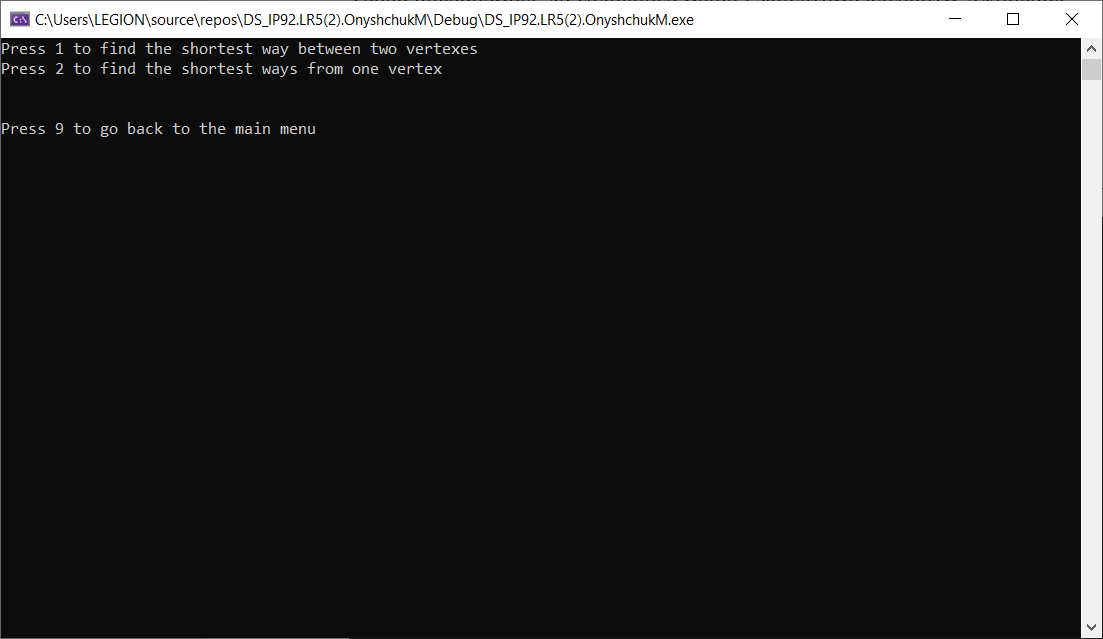


Оберемо вершину 3.

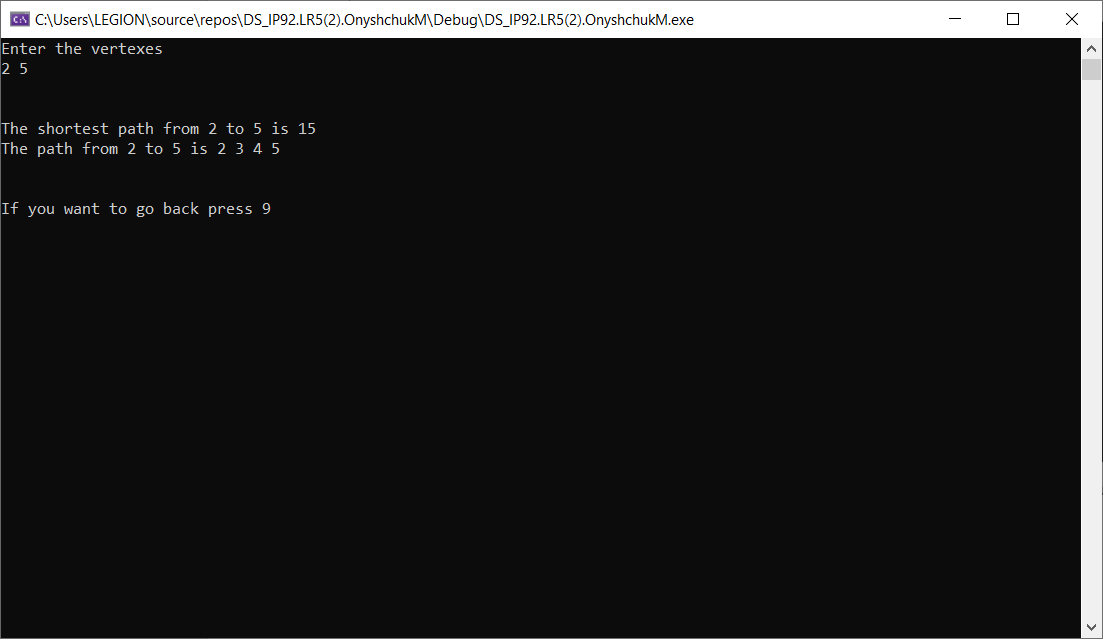


Програма висвітила список відстаней між вершинами в разі наявності між ними маршруту.

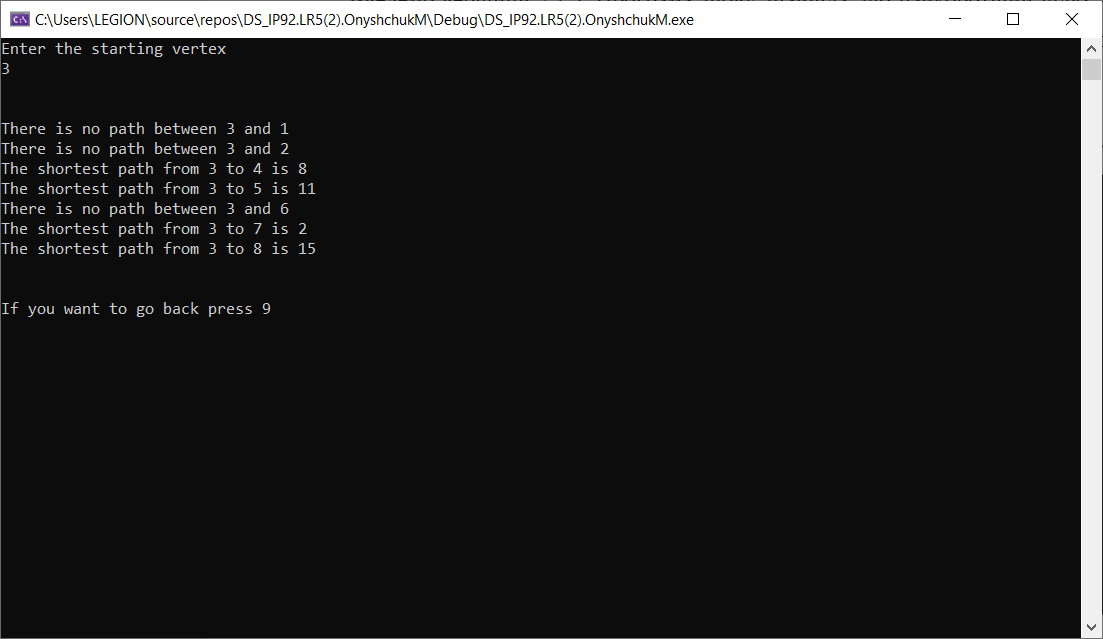
Тепер повернемось до головного меню і запустимо алгоритм Джонсона



Натиснемо один, щоб побачити найкоротший шлях між двома обраними вершинами.



Введемо вершини 2 і 5. Програма знову знайшла, що найкоротший шлях – 15 і відтворила маршрут. Повернемось назад і оберемо 2, щоб побачити список відстаней до різних вершин до обраної.



Оберемо вершину 3. Програма висвітила список відстаней між вершинами в разі наявності між ними маршруту.

**Висновок**

Ми розглянули задачу пошуку найкоротших шляхів в графі за допомогою алгоритмів Белмана-Форда та Джонсона. Результати роботи програми з однаковими вхідними даними для двох алгоритмів є ідентичними, а отже програма була розроблена коректно.