***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №3**

**По дисциплине МОИС за III семестр**

**Тема: «Нахождение кратчайшего пути в графе»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Шуть В. Н.

Брест 2018

Цель: научиться находить кратчайший путь в графе.

***Задание.***

1. Алгоритмом Дейкстры вычислить кратчайшие пути от вершины  ко всем вершинам графа. Варианты графов указаны в таблице 1. Графы заданы списком ребер, в квадратных скобках указаны веса соответствующих ребер.
2. Алгоритмом Флойда-Уоршолла вычислить кратчайшие пути от вершины ко всем вершинам графа.

Количество вершин - 6

Количество рёбер – 9

Ребра - {1,2},{1,3},{1,4},{2,4},{2,6}, {3,4},{3,5}{4,5}{4,6}

Веса - [5,7,4,3,2,5,8,6,4]

Код программы:

**//Application.cpp**

#include <iostream>

#include <Graphs.h>

#include "ShortestPath.h"

int main()

{

const int numOfVer = 6;

const int numOfE = 9;

int e[numOfE][2] = {

{1,2},

{1,3},

{1,4},

{2,4},

{2,6},

{3,4},

{3,5},

{4,5},

{4,6}

};

int weights[numOfE] = { 5,7,4,3,2,5,8,6,4 };

Graph g;

g.addVer(numOfVer);

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

g.addE(e[i][0], e[i][1], weights[i]);

ShortestPathDijkstra shPathD(g);

ShortestPathFloydWarshall shPathFW(g);

int fVer;

std::cout << "First vertex: ";

fVer = std::cin.get() - 48;

std::cout << "Dijkstra: " << std::endl;

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

{

shPathD.getShortestPath(fVer, i).viewPath();

std::cout << "Length: " << shPathD.getShortestPath(fVer, i).length() << std::endl;

}

std::cout << "Floyd-Warshall: " << std::endl;

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

{

shPathFW.getShortestPath(fVer, i).viewPath();

std::cout << "Length: " << shPathFW.getShortestPath(fVer, i).length() << std::endl;

}

system("pause");

}

**//ShortestPath.h**

#pragma once

#include "graph.h"

#include "List\List.h"

#include "Path.h"

class ShortestPathDijkstra

{

Graph m\_Graph;

Path findShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const;

public:

//ShortestPath(const wGraph&);

ShortestPathDijkstra(const Graph&);

~ShortestPathDijkstra();

Path getShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const;

};

class ShortestPathFloydWarshall

{

Graph m\_Graph;

Path findShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const;

public:

//ShortestPath(const wGraph&);

ShortestPathFloydWarshall(const Graph&);

~ShortestPathFloydWarshall();

Path getShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const;

};

**// ShortestPath.cpp**

#include "ShortestPath.h"

#include <iostream>

#include "common.h"

ShortestPathDijkstra::ShortestPathDijkstra(const Graph& wg)

: m\_Graph(wg){}

ShortestPathDijkstra::~ShortestPathDijkstra()

{}

Path ShortestPathDijkstra::getShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const

{

try

{

if (fVertex < 1)

throw 1;

if (fVertex >= m\_Graph.getNumOfVer())

throw 2;

if (fVertex == secVertex)

throw 3;

return findShortestPath(fVertex, secVertex);

}

catch (int e)

{

return Path();

}

}

Path ShortestPathDijkstra::findShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const

{

const GraphNumberOfVertices numOfVer = m\_Graph.getNumOfVer();

// paths from first vertex to other vertices

Path \* paths = new Path[numOfVer];

// path from first vertex to another is initially equal to infinity (1000 000 by default)

for (unsigned int i = 1; i <= numOfVer; i++)

{

if (i == fVertex)

continue;

paths[i-1] = Edge(i, fVertex, m\_Graph.weight(i, fVertex));

}

double \* weights = new double[numOfVer];

for (unsigned int otherVertex = 1; otherVertex <= numOfVer; otherVertex++)

weights[otherVertex - 1] = m\_Graph.weight(fVertex, otherVertex);

// vertices, for which we've already found a shortest path, are marked

bool \* marked = new bool[numOfVer];

// initialy we have only first vertex marked

for (unsigned int i = 0; i < numOfVer; i++)

marked[i] = false;

marked[fVertex - 1] = true;

while (1)

{

// searching for vertex which is nearest to first vertex

GraphVertex cur = 0;

for (unsigned int i = 1; i <= numOfVer; i++)

{

if (marked[i - 1] == false)

{

cur = i;

break;

}

}

if (!cur)

return Path();

for (unsigned int i = 1; i <= numOfVer; i++)

{

if (fVertex == i)

continue;

if (weights[i - 1] < weights[cur - 1] && marked[i - 1] == false)

{

cur = i;

}

}

// if nearest vertex is that one we need then we return its path

if (cur == secVertex)

return paths[secVertex - 1];

// else we mark it

marked[cur - 1] = true;

// then we figure out minimal paths to the vertices using "cur" vertex

for (unsigned int consideredVertex = 1; consideredVertex <= numOfVer; consideredVertex++)

{

if (marked[consideredVertex - 1] == true)

continue;

// if path through "cur" vertex is shorter that considered one

// then we set new path

if ((m\_Graph.weight(cur, consideredVertex) + weights[cur - 1])

< weights[consideredVertex - 1])

{

weights[consideredVertex - 1] = m\_Graph.weight(cur, consideredVertex)

+ weights[cur - 1];

paths[consideredVertex - 1] = paths[cur - 1]

+ Edge(cur, consideredVertex, m\_Graph.weight(cur, consideredVertex));

}

}

}

return Path();

}

ShortestPathFloydWarshall::ShortestPathFloydWarshall(const Graph& wg)

: m\_Graph(wg) {}

ShortestPathFloydWarshall::~ShortestPathFloydWarshall()

{}

Path ShortestPathFloydWarshall::getShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const

{

try

{

if (fVertex < 1)

throw 1;

if (fVertex >= m\_Graph.getNumOfVer())

throw 2;

if (fVertex == secVertex)

throw 3;

return findShortestPath(fVertex, secVertex);

}

catch (int e)

{

return Path();

}

}

Path ShortestPathFloydWarshall::findShortestPath(const GraphVertex fVertex, const GraphVertex secVertex) const

{

Path \*\* paths = new Path\*[m\_Graph.getNumOfVer()];

for (int i = 0; i < m\_Graph.getNumOfVer(); i++)

{

paths[i] = new Path[m\_Graph.getNumOfVer()];

for (int j = 0; j < m\_Graph.getNumOfVer(); j++)

{

paths[i][j].add(Edge(i + 1, j + 1, m\_Graph.weight(i + 1, j + 1)));

}

}

for (int k = 0; k < m\_Graph.getNumOfVer(); k++)

{

for (int i = 0; i < m\_Graph.getNumOfVer(); i++)

{

for (int j = 0; j < m\_Graph.getNumOfVer(); j++)

{

paths[i][j] = (paths[i][j].length() > (paths[i][k].length() + paths[k][j].length()))

? paths[i][k] + paths[k][j] : paths[i][j];

}

}

}

Path res = paths[fVertex - 1][secVertex - 1];

for (int i = 0; i < m\_Graph.getNumOfVer(); i++)

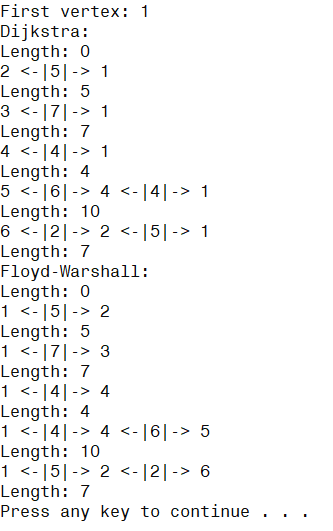
delete[] paths[i];

delete[] paths;

return res;

}

Результат выполнения:



Вывод: научился находить кратчайшие пути в графе.