***Министерство образования Республики Беларусь***

***Учреждение образования***

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

***Кафедра ИИТ***

**Лабораторная работа №4**

**По дисциплине МОИС за III семестр**

**Тема: «Нахождение минимального остового дерева связанного неориентированного графа»**

**Выполнил:**

Студент группы ИИ-15 (1)

2-го курса

Волк И. А.

**Проверил:**

Шуть В. Н.

Брест 2018

Цель: научиться находить минимальное остовое дерево в графе.

***Задание.***

1. Найти минимальное остовое дерево для заданного графа *G* алгоритмом Прима и Крускаля. Варианты графов указаны в таблице 1. Граф задан списком ребер.
2. Ответить на поставленные вопросы.
3. Графически изобразить граф и его минимальное остовое дерево.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7. | 9 | 12 | (a,b),(a,c), (b,c),(c,d),(c,g),(d,g),  (d,e),(d,f),(g,h),(g,i),(f,e),(h,i) | 2,3,4,5,7,6  3,5,1,1,1,2 |

Код программы:

**// Application.cpp**

#include <iostream>

#include "Graphs.h"

int main()

{

const int numOfVer = 9;

const int numOfE = 12;

int e[numOfE][2] = {

{1,2},

{1,3},

{2,3},

{3,4},

{3,7},

{4,7},

{4,5},

{4,6},

{7,8},

{7,9},

{6,5},

{8,9}

};

int weights[numOfE] = { 2,3,4,5,7,6,3,5,1,1,1,2 };

Graph graph;

graph.addVer(numOfVer);

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

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

Graph tree;

// graph

matrix::outMatrix(matrix::adjMatrix(graph), graph.getNumOfVer(), graph.getNumOfVer());

// MST

mintreeUndefined(tree, graph);

std::cout << std::endl;

matrix::outMatrix(matrix::adjMatrix(tree), tree.getNumOfVer(), tree.getNumOfVer());

// MST Prim's

mintreePrim(tree, graph);

std::cout << std::endl;

matrix::outMatrix(matrix::adjMatrix(tree), tree.getNumOfVer(), tree.getNumOfVer());

// MST Kruskal's

mintreeKruskal(tree, graph);

std::cout << std::endl;

matrix::outMatrix(matrix::adjMatrix(tree), tree.getNumOfVer(), tree.getNumOfVer());

std::cin.get();

}

**//mintree.h**

#pragma once

#include "graph.h"

void mintreeUndefined(Graph& result, Graph graph);

void mintreePrim(Graph& result, Graph graph);

void mintreeKruskal(Graph& result, Graph graph);

**//mintree.cpp**

#include "stdafx.h"

#include "mintree.h"

#include <stack>

bool isCircuit(const Graph& graph, const int firstEdge = 0)

{

bool \* marked = new bool[graph.getNumOfE()];

for (int i = 0; i < graph.getNumOfE(); i++)

marked[i] = false;

marked[firstEdge] = true;

std::stack<int> path;

path.push(graph.getEdge(firstEdge).fVertex);

bool isCir = false;

while (true)

{

if (path.empty())

break;

int isEdge = false;

for (int i = 1; i <= graph.getNumOfVer(); i++)

{

int id = graph.edgeId(path.top(), i);

if (id == -1)

continue;

if (marked[id])

continue;

isEdge = true;

marked[id] = true;

path.push(i);

break;

}

if (path.top() == graph.getEdge(firstEdge).secVertex)

{

isCir = true;

break;

}

if(!isEdge)

path.pop();

}

return isCir;

}

void mintreeUndefined(Graph& result, Graph graph)

{

int id = 0;

for (int i = 1; i < graph.getNumOfE(); i++)

{

if (graph[i].weight < graph[id].weight)

id = i;

}

result = Graph();

result.addVer(graph.getNumOfVer());

result.addE(graph[id]);

graph.remE(id);

while (graph.getNumOfE() != 0)

{

id = 0;

for (int i = 1; i < graph.getNumOfE(); i++)

{

if (graph[i].weight < graph[id].weight)

id = i;

}

result.addE(graph[id]);

graph.remE(id);

for (int i = 0; i < graph.getNumOfE(); i++)

{

result.addE(graph[i]);

if (isCircuit(result, result.getNumOfE() - 1))

{

graph.remE(i);

i = -1; // restart the loop

}

result.remE(result.getNumOfE() - 1);

}

}

}

void mintreePrim(Graph& result, Graph graph)

{

result = Graph();

result.addVer(graph.getNumOfVer());

bool \*marked = new bool[graph.getNumOfVer()];

for (int i = 0; i < graph.getNumOfVer(); i++)

marked[i] = false;

marked[0] = true; // initial vertex

while (1)

{

// choosing an appropriate edge

bool isEdge = false;

int e\_id;

for (int i = 0; i < graph.getNumOfE(); i++)

{

if ((marked[graph[i].fVertex - 1] && !marked[graph[i].secVertex - 1])

|| (!marked[graph[i].fVertex - 1] && marked[graph[i].secVertex - 1]))

{

e\_id = i;

isEdge = true;

break;

}

}

// is there is no appropriate edge

if (!isEdge)

break;

// choosing lightest of appropriate edges

for (int i = 0; i < graph.getNumOfE(); i++)

{

if ((marked[graph[i].fVertex - 1] && !marked[graph[i].secVertex - 1])

|| (!marked[graph[i].fVertex - 1] && marked[graph[i].secVertex - 1]))

{

if (graph[i].weight < graph[e\_id].weight)

e\_id = i;

isEdge = true;

}

}

// adding chosen edge to the result graph

result.addE(graph[e\_id]);

if (marked[graph[e\_id].fVertex - 1])

marked[graph[e\_id].secVertex - 1] = true;

else

marked[graph[e\_id].fVertex - 1] = true;

}

}

void mintreeKruskal(Graph& result, Graph graph)

{

result = Graph();

result.addVer(graph.getNumOfVer());

int \* treeMark = new int[graph.getNumOfVer()];

for (int i = 0; i < graph.getNumOfVer(); i++)

{

treeMark[i] = i + 1;

}

while (graph.getNumOfE())

{

// we find the lightest edge in the graph

int e\_id = 0;

for (int i = 0; i < graph.getNumOfE(); i++)

{

if (graph[i].weight < graph[e\_id].weight)

{

e\_id = i;

}

}

if (treeMark[graph[e\_id].fVertex - 1] == treeMark[graph[e\_id].secVertex - 1])

{

graph.remE(e\_id);

continue;

}

result.addE(graph[e\_id]);

for (int i = 0; i < graph.getNumOfVer(); i++)

{

if (i == (graph[e\_id].fVertex - 1))

continue;

if (treeMark[graph[e\_id].fVertex - 1] == treeMark[i])

treeMark[i] = treeMark[graph[e\_id].secVertex - 1];

}

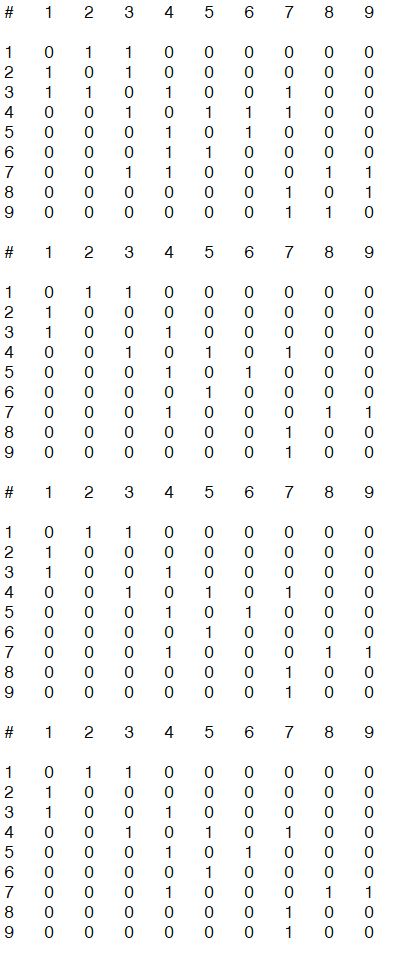
treeMark[graph[e\_id].fVertex - 1] = treeMark[graph[e\_id].secVertex - 1];

graph.remE(e\_id);

}

}

Результат выполнения:



Вывод: научился находить минимальное остовое дерево в графе.