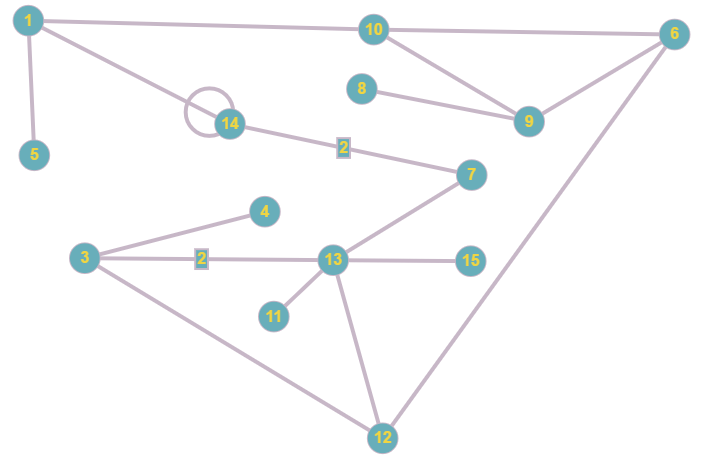
**Лабораторная работа №1**

**Вариант 25**

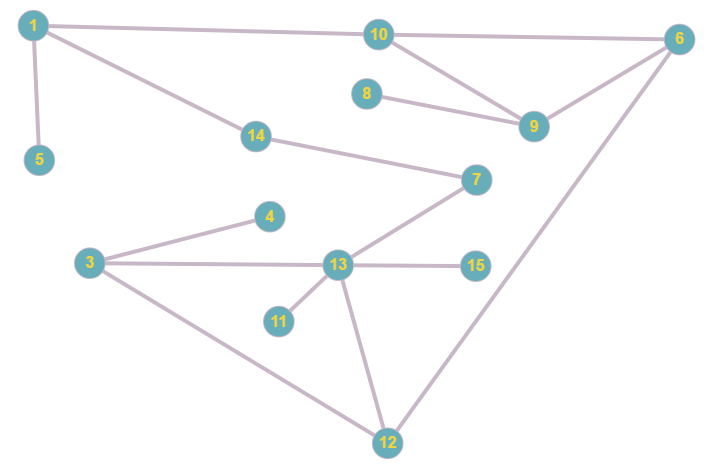
**Задание:** Написать программу, которая для неориентированного графа, заданного множеством вершин и ребер (данные находятся во входном файле):

1. Строит и выводит на экран списки смежности и матрицу смежности
2. Определяет:
   * степени всех вершин и выводит список вершин с их степенями;
   * изолированные вершины и выводит их список;
   * висячие вершины и выводит их список;
   * висячие ребра и выводит их список;
   * вершины, в которых имеются петли, и выводит список таких вершин с кратностями петель;
   * кратные ребра и выводит их список с кратностями
3. Приводит исходный граф к форме простого графа: удаляет петли и для кратных ребер оставляет только одно ребро. Для преобразованного графа выводит списки смежности
4. Определяет количество компонент связности преобразованного графа и выводит на экран состав каждой компоненты в форме списка входящих в компоненту вершин
5. Для каждой не тривиальной компоненты связности строит остовное дерево двумя способами: с помощью алгоритма поиска в глубину (DFS) и с помощью алгоритма поиска в ширину (BFS)

**Иллюстрации к лабораторной работе:**

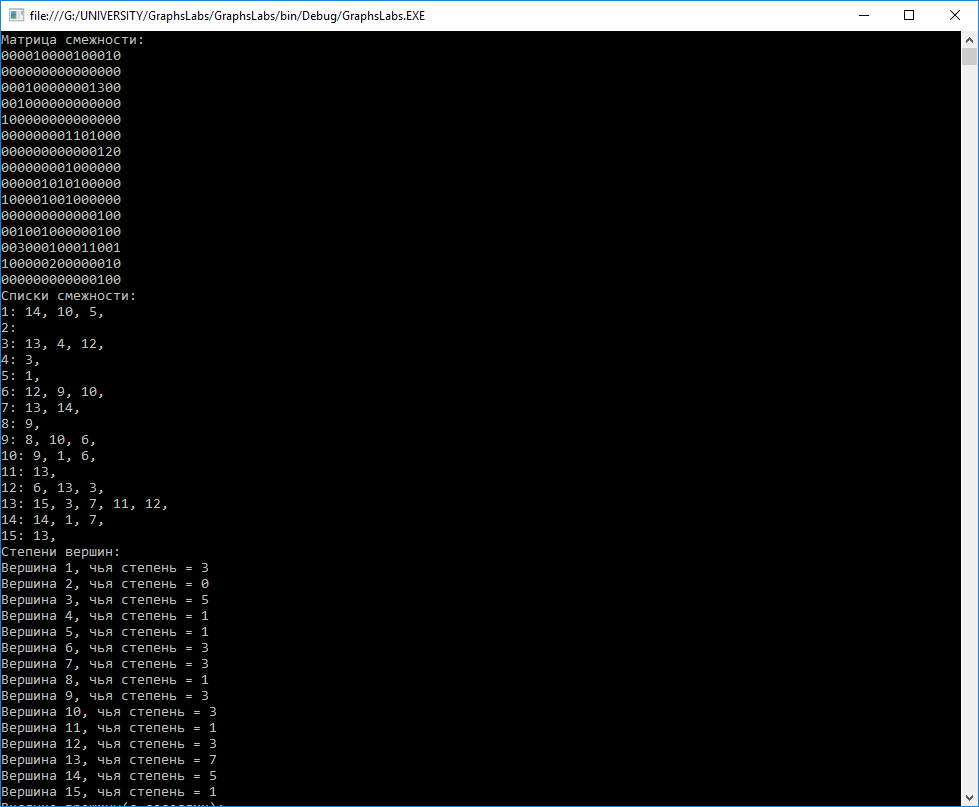


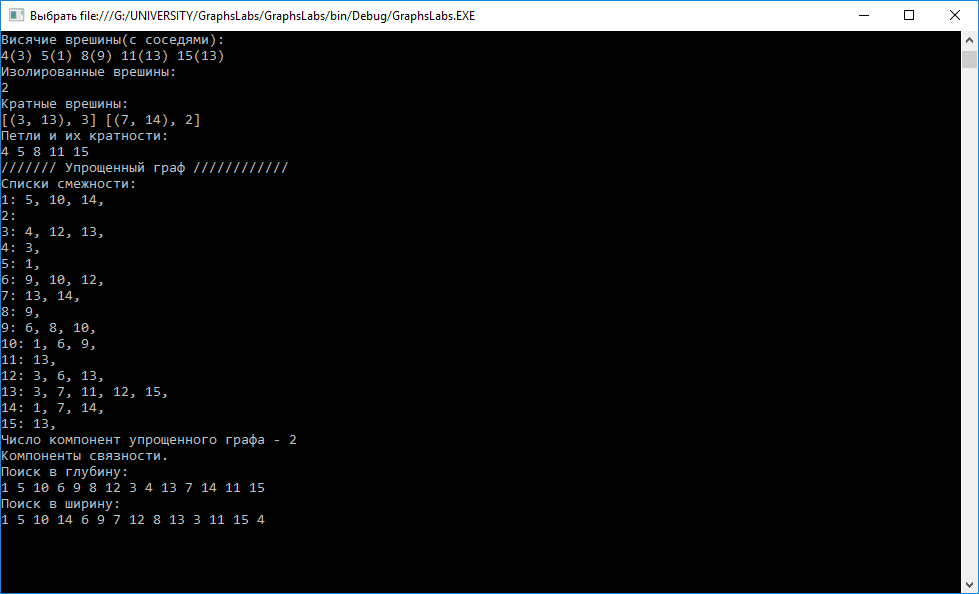
**Рисунок 1.1** – Исходный граф



**Рисунок 1.2** – Упрощенный граф

**Результаты работы:**





**Рисунок 1.3** – Результаты работы программы

**Лабораторная работа №2**

**Вариант 25**

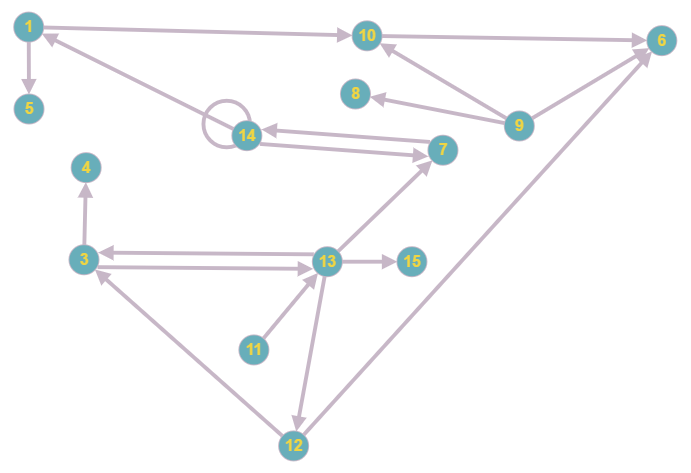
**Задание:** 1. С использованием исходных данных лаб. работы № 1 и разработанного программного обеспечения построить орграф, соответствующий максимальной связной компоненте заданного графа (максимальной считается компонента связности с максимальным количеством вершин).

2. Для построенного орграфа выполнить следующее:

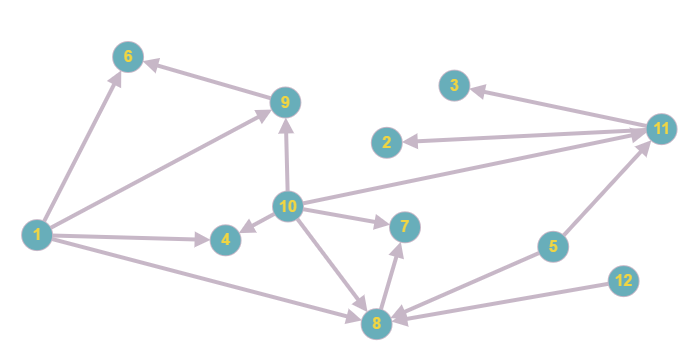
* + для каждой вершины вывести значения полустепени исхода и полустепени захода;
  + вывести список вершин-источников и список вершин-стоков;
  + для каждой вершины вывести список достижимых из нее вершин;
  + на основе анализа списков достижимости определить компоненты сильной связности (или доказать их отсутствие);
  + проверить орграф на ацикличность.

3. Выполнить топологическую сортировку ациклического орграфа, заданного в исходном файле к лаб. работе № 2. Изобразить орграф в линейном виде с учетом топологического упорядочения.

**Иллюстрации к лабораторной работе:**

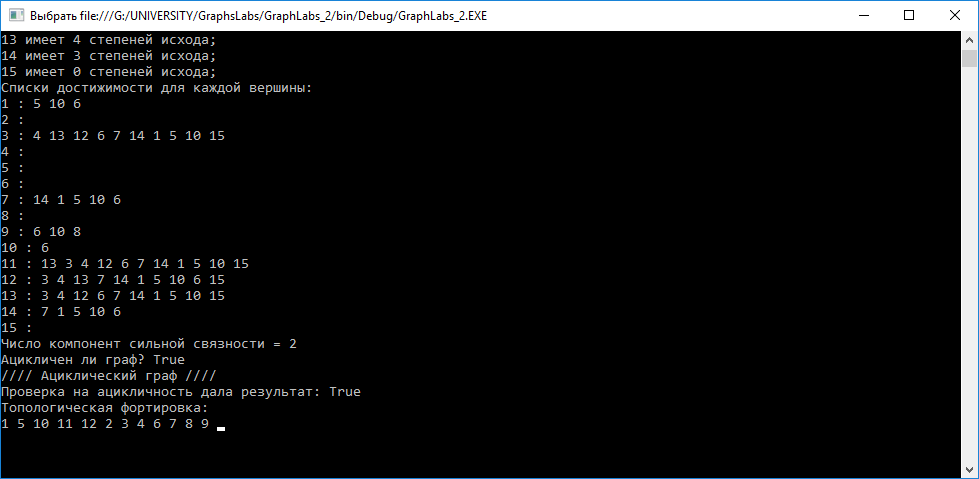
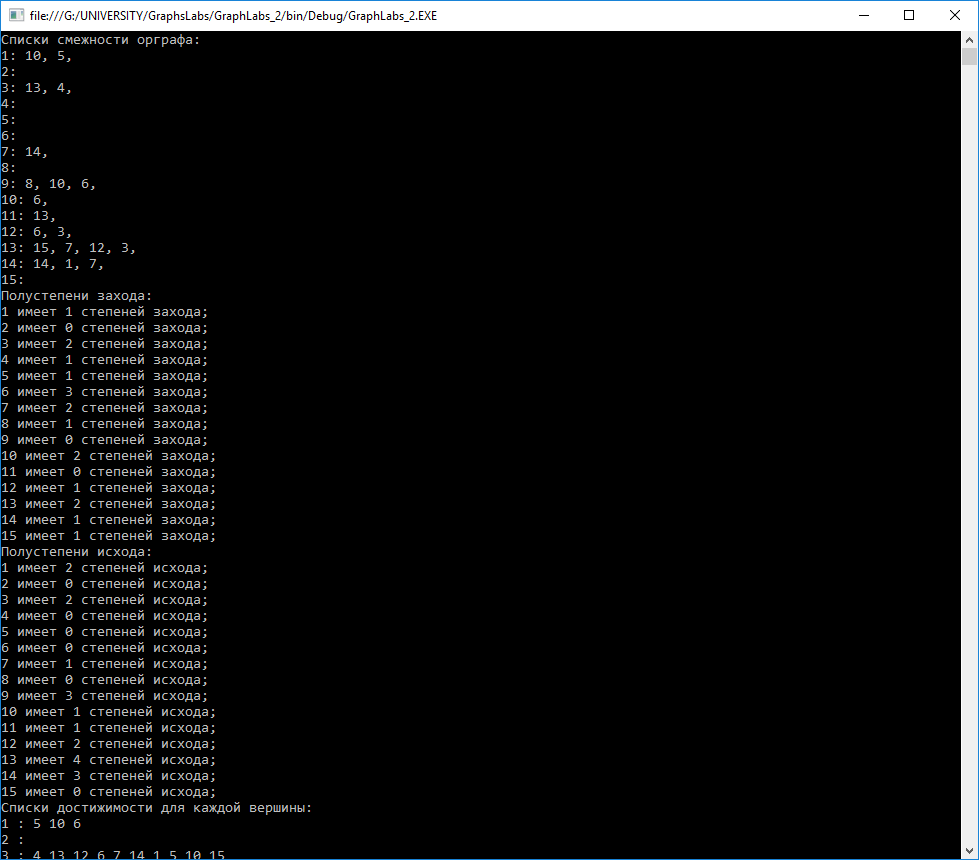


**Рисунок 2.1** – Исходный граф для первого задания



**Рисунок 2.2** – Исходный граф для второго и третьего заданий

**Результаты работы:**



**Рисунок 2.3** – Результаты работы программы

**Лабораторная работа №3**

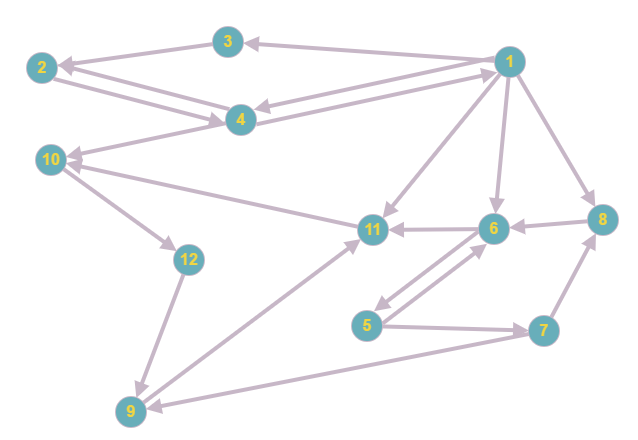
**Вариант 25**

**Задание:** 1. Разработать программу, приводящую заданный ациклический орграф (DAG) к ярусно-параллельной форме. Использовать исходные данные к лаб. работе № 2.

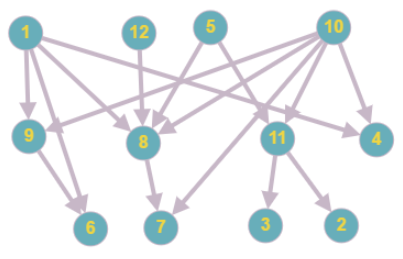
2. Разработать программу, отыскивающую компоненты сильной связности в заданном орграфе (см.исходные данные к лаб. работе № 3). Программа должна быть построена на основе алгоритма Kosaraju-Sharir.

3. Модифицировать программу из п.2 таким образом, чтобы она позволяла для заданного орграфа получать его метаграф. Показать, что метаграф является DAG.

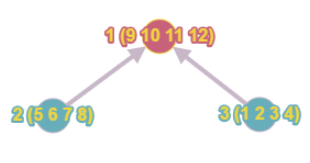
**Иллюстрации к лабораторной работе:**



**Рисунок 3.1** – Исходный граф к лаб. 3

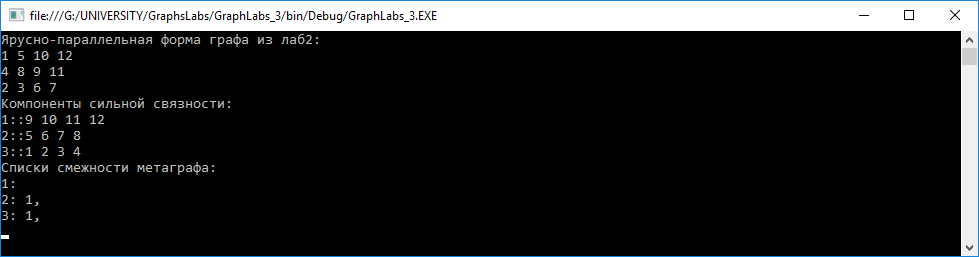


**Рисунок 3.2** – Ярусно-параллельная форма



**Рисунок 3.3** – Метаграф

**Результаты работы:**



**Рисунок 3.4** – Результаты работы

**Лабораторная работа №4**

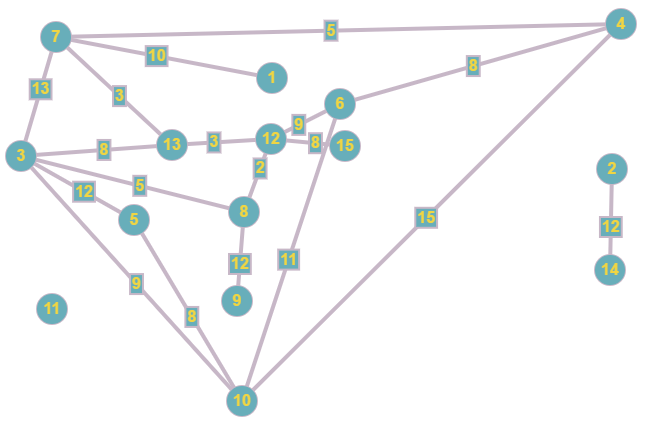
**Вариант 25**

**Задание:** Построить минимальное остовное дерево для заданного неориентированного взвешенного графа.

Формат входного файла: первая строка – количество вершин графа, вторая и последующие строки – ребра графа в формате «вершина-вершина-вес».

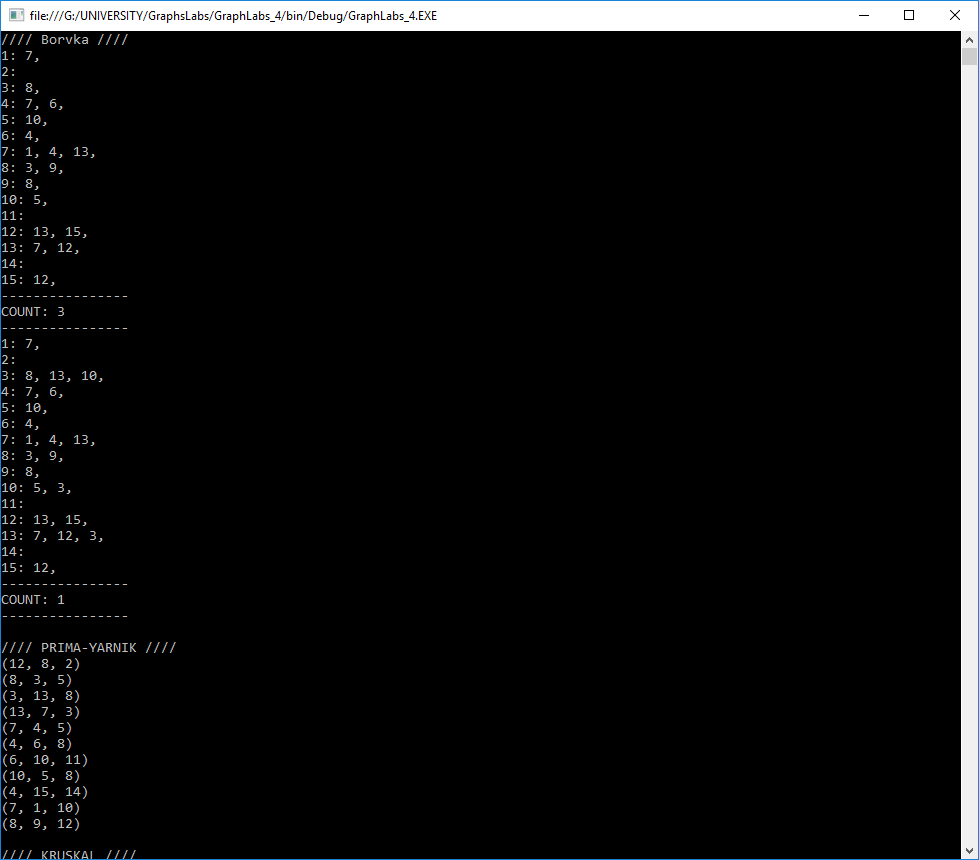
1. Разработать и реализовать 3 программы потроения минимального остовного дерева по алгоритмам: Борувки, Ярника (Прима) и Крускала.

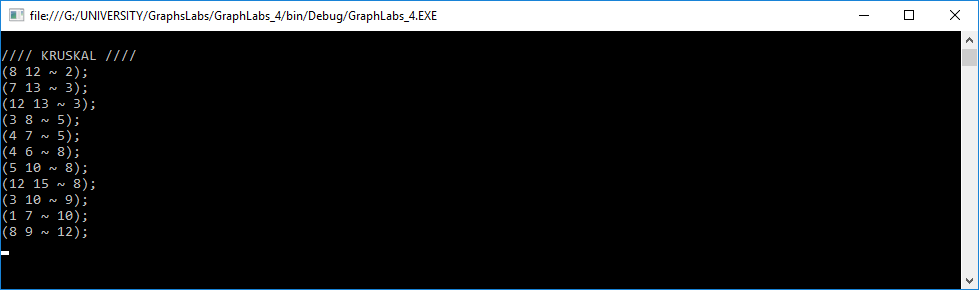
**Иллюстрации к лабораторной работе:**



**Рисунок 4.1** – Исходный граф

**Результаты работы:**





**Рисунок 4.2** – Результаты работы

**Лабораторная работа №5**

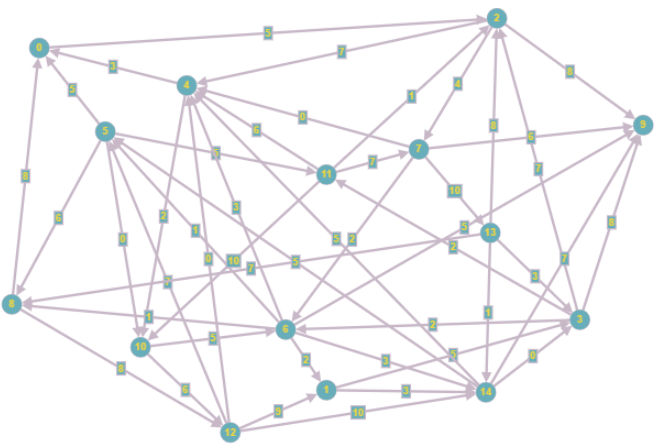
**Вариант 25**

**Задание:** Построить дерево кратчайших путей для заданного ориентированного взвешенного графа.

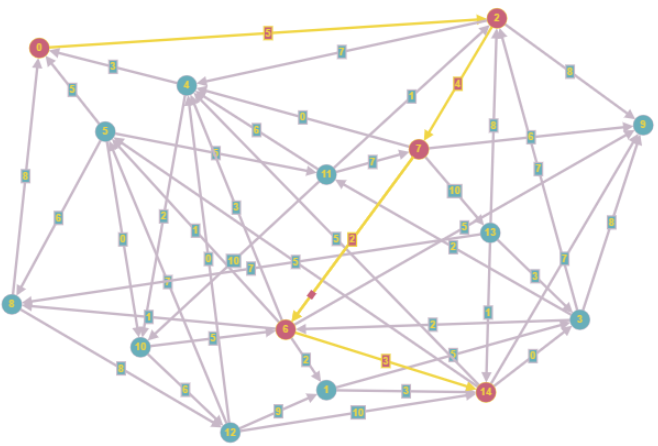
Формат входного файла: первая строка – количество вершин графа, вторая и последующие строки – дуги графа в формате «вершина-вершина-вес».

Для построения дерева кратчайших путей разработать и реализовать программу на основе унивесального релаксационного алгоритма Форда с 3 вариантами для «мешка»: стек, очередь и приоритетная очередь.

**Иллюстрации к лабораторной работе:**

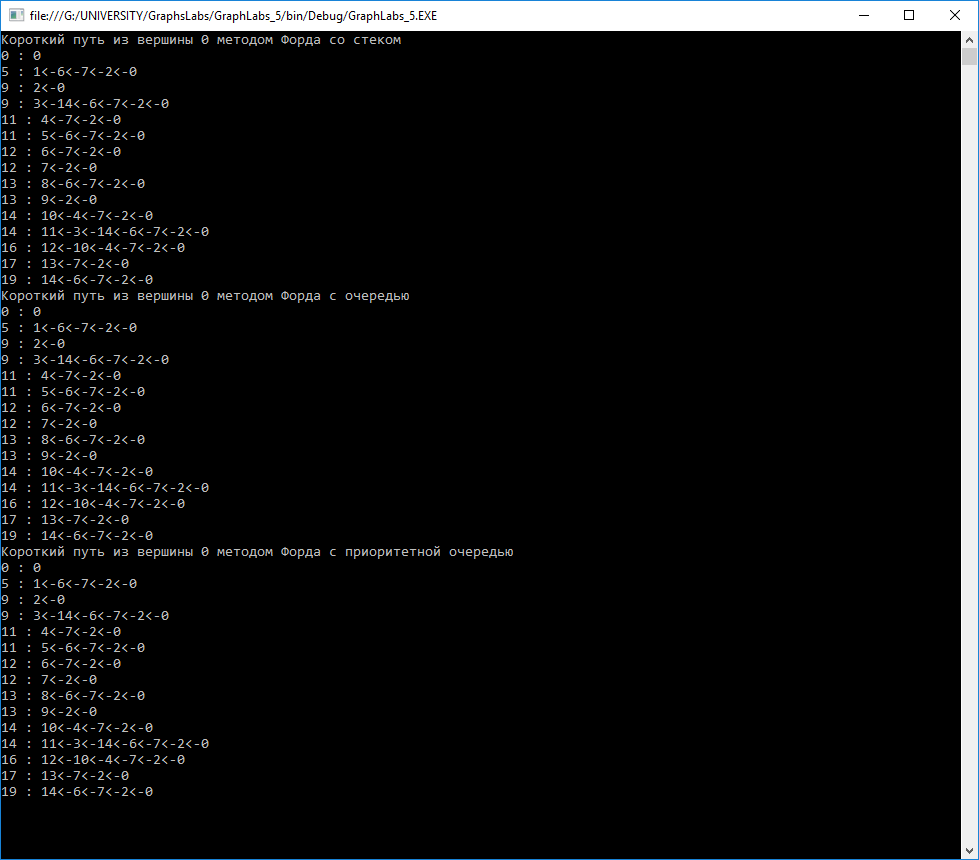


**Рисунок 5.1** – Исходный граф



**Рисунок 5.2** – Кратчайший путь из вершины 0 в вершину 14

**Результаты работы программы:**



**Рисунок 5.3** – Результаты работы

**Исходный код лабораторных работ**

*Лабораторная работа №1*

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

namespace GraphsLabs

{

public class UndirectedGraph

{

public int[,] adjMatrix { get; private set; }

public Dictionary<int, HashSet<int>> adjList { get; private set; }

Dictionary<int, int> degrees;

public int vertecesCount { get; private set; }

public UndirectedGraph(int vertecesCount = 0)

{

this.adjList = new Dictionary<int, HashSet<int>>();

this.vertecesCount = vertecesCount;

this.adjMatrix = new int[vertecesCount, vertecesCount];

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

this.adjList.Add(i + 1, new HashSet<int>());

}

public UndirectedGraph(string path) { this.InitGraph(path); }

public UndirectedGraph(int[,] adjMatrix)

{

vertecesCount = adjMatrix.GetLength(0);

this.adjMatrix = new int[vertecesCount, vertecesCount];

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

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

this.adjMatrix[i, j] = adjMatrix[i, j];

this.adjList = new Dictionary<int, HashSet<int>>();

}

private void AddEdge(int i, int j)

{

if (i > 0 && i <= vertecesCount && j > 0 && j <= vertecesCount)

{

adjMatrix[i - 1, j - 1]++;

if (i != j)

adjMatrix[j - 1, i - 1]++;

}

}

public void AddArc(int i, int j)

{

if (!adjList.ContainsKey(i))

{

adjList.Add(i, new HashSet<int>());

adjList[i].Add(j);

}

else

adjList[i].Add(j);

if (!adjList.ContainsKey(j))

{

adjList.Add(j, new HashSet<int>());

adjList[j].Add(i);

}

else

adjList[j].Add(i);

}

public Dictionary<int, int> GetDegrees()

{

degrees = new Dictionary<int, int>();

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

{

if (!degrees.ContainsKey(i + 1))

degrees.Add(i + 1, 0);

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

{

degrees[i + 1] += adjMatrix[i, j];

if (i == j && adjMatrix[i, j] != 0)

degrees[i + 1]++;

}

}

Console.WriteLine();

return degrees;

}

public int[] GetHangingVertexies()

{

int[] hanging = adjList.Where(v => v.Value.Where(n => n != v.Key).Count() == 1).Select(v => v.Key).ToArray();

return hanging;

}

public Dictionary<int, int> GetLoops()

{

var loops = new Dictionary<int, int>();

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

if (adjMatrix[i, i] > 0)

loops.Add(i + 1, adjMatrix[i, i]);

return loops;

}

public Dictionary<Tuple<int, int>, int> GetMultipleEdges()

{

var multE = new Dictionary<Tuple<int, int>, int>();

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

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

{

if (adjMatrix[i, j] > 1 && !multE.Keys.Contains(new Tuple<int, int>(j + 1, i + 1)))

multE.Add(new Tuple<int, int>(i + 1, j + 1), adjMatrix[i, j]);

}

return multE;

}

private void InitGraph(string path)

{

var str = File.ReadAllText(path);

var strA = str.Split(new char[] { ' ', '\n' }).Where(s => s != string.Empty).ToArray();

int[] verts = Array.ConvertAll(strA, s => int.Parse(s));

vertecesCount = verts[0];

var step = (verts.Length - 1) % 3 == 0 ? 3 : 2;

adjMatrix = new int[vertecesCount, vertecesCount];

adjList = new Dictionary<int, HashSet<int>>();

for (int v = 1; v <= vertecesCount; v++)

adjList.Add(v, new HashSet<int>());

for (int v = 1; v < verts.Length; v += step)

{

AddEdge(verts[v], verts[v + 1]);

AddArc(verts[v], verts[v + 1]);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphsLabs

{

public class SimplifiedGraph

{

//private UndirectedGraph simpleGraph;

public int[,] adjMatrix { get; private set; }

public Dictionary<int, HashSet<int>> adjList { get; private set; }

public bool[] markers { get; private set; }

public int[] label { get; private set; }

public int vertecesCount { get; private set; }

public SimplifiedGraph(int n = 0)

{

this.vertecesCount = n;

this.adjMatrix = new int[n, n];

this.adjList = new Dictionary<int, HashSet<int>>();

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

this.adjList.Add(i + 1, new HashSet<int>());

label = new int[vertecesCount];

this.markers = new bool[vertecesCount];

//this.simpleGraph = new UndirectedGraph();

}

public SimplifiedGraph(UndirectedGraph graph)

{

if (graph.vertecesCount != 0)

{

this.MakeGraphSimple(graph);

label = new int[vertecesCount];

this.markers = new bool[vertecesCount];

}

}

public void AddEdge(int v, int n)

{

if ((v < 1) || (v > vertecesCount)) return;

if ((n < 1) || (n > vertecesCount)) return;

foreach (var x in adjList[v])

if (x == n) return;

adjList[v].Add(n);

adjList[n].Add(v);

}

private void MakeGraphSimple(UndirectedGraph graph)

{

//this.simpleGraph = new UndirectedGraph(graph.adjMatrix);

this.vertecesCount = graph.vertecesCount;

this.adjList = new Dictionary<int, HashSet<int>>();

this.adjMatrix = new int[this.vertecesCount, this.vertecesCount];

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

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

{

if (i == j && graph.adjMatrix[i, j] > 0)

adjMatrix[i, j] = 0;

if (i != j && graph.adjMatrix[i, j] > 1)

adjMatrix[i, j] = 1;

}

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

{

if (!adjList.ContainsKey(i + 1))

adjList.Add(i + 1, new HashSet<int>());

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

if (graph.adjMatrix[i, j] != 0)

adjList[i + 1].Add(j + 1);

}

}

public void ClearMarks()

{

this.markers = new bool[vertecesCount];

}

public int CountComponents()

{

this.ClearMarks();

int count = 0;

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

{

if (markers[i])

continue;

count++;

this.FindBindingComponentsInDepth(i + 1);

}

return count;

}

public void FindBindingComponentsInDepth(int cur, List<int> verteces = null)

{

if (markers[cur - 1])

return;

markers[cur - 1] = true;

if (verteces != null)

verteces.Add(cur);

foreach (var n in adjList[cur])

FindBindingComponentsInDepth(n, verteces);

}

public void FindBindingComponentsInWidth(int s)

{

Console.Write(s + " ");

Queue<int> q = new Queue<int>();

q.Enqueue(s);

//int[] d = new int[simpleGraph.vertecesCount];

//int[] p = new int[simpleGraph.vertecesCount];

markers[s - 1] = true;

//p[s] = -1;

var adjList = this.adjList;

while (q.Count() != 0)

{

int v = q.Dequeue();

for (int i = 0; i < adjList[v].Count(); i++)

{

int to = adjList[v].ElementAt(i);

if (!markers[to - 1])

{

markers[to - 1] = true;

q.Enqueue(to);

//d[to] = d[v - 1] + 1;

//p[to] = v;

Console.Write(to + " ");

}

}

}

}

public List<int> GetMaxComponent()

{

List<List<int>> components = new List<List<int>>();

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

{

components.Add(new List<int>());

FindBindingComponentsInDepth(i, components[i - 1]);

}

return components.OrderByDescending(c => c.Count).FirstOrDefault();

}

private void LabelComponent(int v, int count)

{

markers[v - 1] = true;

label[v - 1] = count;

foreach (var n in adjList[v])

if (!markers[n - 1])

LabelComponent(n, count);

}

public int CountAndLabel(List<int> maxC)

{

int count = 0;

//label = new int[simpleGraph.vertecesCount];

ClearMarks();

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

if (!markers[i - 1] && maxC.Contains(i))

LabelComponent(i, count++);

return count;

}

public void PrintAdjLists()

{

foreach (var v in this.adjList.OrderBy(v => v.Key))

{

Console.Write(v.Key + ": ");

foreach (var n in v.Value)

Console.Write(n + ", ");

Console.WriteLine();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphsLabs

{

class Program

{

static void Main(string[] args)

{

string path = "../../../Lab01-gr5-25.dat";

var graph = new UndirectedGraph(path);

Console.WriteLine("Матрица смежности: ");

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

{

for (int j = 0; j < graph.adjMatrix.GetLength(1); j++)

{

Console.Write(graph.adjMatrix[i, j]);

}

Console.WriteLine();

}

Console.WriteLine("Списки смежности: ");

foreach (var v in graph.adjList.OrderBy(v => v.Key))

{

Console.Write(v.Key + ": ");

foreach (var n in v.Value)

Console.Write(n + ", ");

Console.WriteLine();

}

Console.Write("Степени вершин: ");

var d = graph.GetDegrees();

foreach (var v in d)

Console.WriteLine("Вершина " + v.Key + ", чья степень = " + v.Value);

Console.WriteLine("Висячие врешины(c соседями): ");

var h = graph.GetHangingVertexies();

foreach (var hv in h)

Console.Write(hv + "(" + graph.adjList[hv].First() + ") ");

Console.WriteLine();

Console.WriteLine("Изолированные врешины: ");

foreach (var i in graph.adjList.Where(v => v.Value.Count() == 0))

Console.Write(i.Key + " ");

Console.WriteLine();

Console.WriteLine("Кратные врешины: ");

var multipleEdges = graph.GetMultipleEdges();

foreach (var me in multipleEdges)

Console.Write(me + " ");

Console.WriteLine();

Console.WriteLine("Петли и их кратности: ");

var loops = graph.GetLoops();

foreach (var hv in h)

Console.Write(hv + " ");

Console.WriteLine();

Console.WriteLine("/////// Упрощенный граф ////////////");

var simplifiedGraph = new SimplifiedGraph(graph);

Console.WriteLine("Списки смежности: ");

foreach (var v in simplifiedGraph.adjList.OrderBy(v => v.Key))

{

Console.Write(v.Key + ": ");

foreach (var n in v.Value)

Console.Write(n + ", ");

Console.WriteLine();

}

Console.WriteLine("Число компонент упрощенного графа - " + simplifiedGraph.CountComponents());

Console.WriteLine("Компоненты связности. ");

Console.WriteLine("Поиск в глубину: ");

var depthTree = new List<int>();

simplifiedGraph.ClearMarks();

simplifiedGraph.FindBindingComponentsInDepth(1, depthTree);

foreach (var v in depthTree)

Console.Write(v + " ");

Console.WriteLine();

Console.WriteLine("Поиск в ширину: ");

simplifiedGraph.ClearMarks();

simplifiedGraph.FindBindingComponentsInWidth(1);

Console.ReadKey();

}

}

}

*Лабораторная работа №2*

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphLabs\_2

{

public enum States { New, Active, Done }

public class DirectedGraph

{

public int[,] adjMatrix { get; private set; }

public Dictionary<int, HashSet<int>> adjList { get; private set; }

public Dictionary<int, HashSet<int>> revAdjList { get; private set; }

public int vertecesCount { get; private set; }

private bool[] markers;

private Stack<int> vertStack;

private int[] label;

private States[] states;

public bool[,] adjMatrixIns { get; private set; }

private Dictionary<int, HashSet<int>> asycTempAdjList;

public Dictionary<int, int> HalfDegreesOut { get; private set; }

public Dictionary<int, int> HalfDegreesIn { get; private set; }

public DirectedGraph() { }

public DirectedGraph(int[,] adjMatrix,

Dictionary<int, HashSet<int>> adjList)

{

this.adjMatrix = adjMatrix;

this.adjList = adjList;

this.vertecesCount = adjList.Count;

}

private void AddEdgeToMatrix(int i, int j)

{

if (i > 0 && i <= vertecesCount && j > 0 && j <= vertecesCount)

{

adjMatrix[i - 1, j - 1] += 1;

adjMatrixIns[j - 1, i - 1] = true;

}

}

private void AddArcToList(int v, int n)

{

if (v > 0 && n <= vertecesCount && v > 0 && n <= vertecesCount)

{

adjList[v].Add(n);

revAdjList[n].Add(v);

}

}

public void InitGraph(string path)

{

var str = File.ReadAllText(path);

var strA = str.Split(new char[] { ' ', '\n' }).Where(s => s != string.Empty).ToArray();

int[] verts = Array.ConvertAll(strA, s => int.Parse(s));

vertecesCount = verts[0];

adjMatrix = new int[vertecesCount, vertecesCount];

adjMatrixIns = new bool[vertecesCount, vertecesCount];

adjList = new Dictionary<int, HashSet<int>>();

revAdjList = new Dictionary<int, HashSet<int>>();

for (int v = 1; v <= vertecesCount; v++)

{

adjList.Add(v, new HashSet<int>());

revAdjList.Add(v, new HashSet<int>());

}

for (int v = 1; v < verts.Length; v += 2)

{

AddEdgeToMatrix(verts[v], verts[v + 1]);

AddArcToList(verts[v], verts[v + 1]);

}

GetHalfDegreesIn();

GetHalfDegreesOut();

}

public void GetHalfDegreesOut()

{

HalfDegreesOut = new Dictionary<int, int>();

foreach (var v in adjList)

HalfDegreesOut.Add(v.Key, v.Value.Count);

}

public void GetHalfDegreesIn()

{

HalfDegreesIn = new Dictionary<int, int>();

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

{

HalfDegreesIn.Add(i + 1, 0);

var t = adjList.Where(v => v.Key != (i + 1));

foreach (var n in t)

if (n.Value.Contains(i + 1))

HalfDegreesIn[i + 1]++;

}

}

public List<int> GetSources()

{

GetHalfDegreesIn();

GetHalfDegreesOut();

return HalfDegreesOut.Where(v => !HalfDegreesIn.ContainsKey(v.Key)).Select(v => v.Key).ToList();

}

public List<int> GetStokes()

{

return HalfDegreesIn.Where(v => !HalfDegreesOut.ContainsKey(v.Key)).Select(v => v.Key).ToList();

}

public void ClearMarks()

{

this.markers = new bool[vertecesCount];

}

public void RevDFS(int cur)

{

markers[cur - 1] = true;

foreach (var n in revAdjList[cur])

if (!markers[n - 1])

RevDFS(n);

vertStack.Push(cur);

}

public void LabelDFS(int cur, int count)

{

markers[cur - 1] = true;

label[cur - 1] = count;

foreach (var n in adjList[cur])

if (!markers[n - 1])

LabelDFS(n, count);

}

public List<int> Traverse(int v, bool r = true)

{

ClearMarks();

var bag = new Stack<Tuple<int, int>>();

bag.Push(new Tuple<int, int>(0, v));

var parent = new int[vertecesCount];

var reach = new List<int>();

var adj = r ? adjList : revAdjList;

Tuple<int, int> p = new Tuple<int, int>(0, 0);

while (bag.Count > 0)

{

p = bag.Pop();

if (markers[p.Item2 - 1])

continue;

markers[p.Item2 - 1] = true;

parent[p.Item2 - 1] = p.Item1;

reach.Add(p.Item2);

foreach (var n in adj[p.Item2])

{

if (markers[n - 1])

continue;

bag.Push(new Tuple<int, int>(p.Item2, n));

}

}

return reach;

}

public List<int> GetComponent(int v)

{

return Traverse(v).Intersect(Traverse(v, false)).ToList();

}

public List<int> Kasaraju()

{

vertStack = new Stack<int>();

label = new int[vertecesCount];

ClearMarks();

for (int v = 1; v <= vertecesCount; v++)

if (!markers[v - 1])

RevDFS(v);

ClearMarks();

for (int count = 0; vertStack.Count > 0;)

{

int v = vertStack.Pop();

if (!markers[v - 1])

LabelDFS(v, ++count);

}

return label.ToList();

}

private bool IsAcyclicDFS(int v)

{

states[v - 1] = States.Active;

foreach(var n in asycTempAdjList[v])

{

if(states[n - 1] == States.Active)

return false;

if (states[n - 1] == States.New)

if (IsAcyclicDFS(n) == false)

return false;

}

states[v - 1] = States.Done;

return true;

}

public bool IsAcyclic()

{

asycTempAdjList = new Dictionary<int, HashSet<int>>(adjList);

asycTempAdjList.Add(asycTempAdjList.Last().Key + 1, new HashSet<int>());

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

adjList.Last().Value.Add(i);

states = new States[vertecesCount + 1];

return IsAcyclicDFS(asycTempAdjList.Last().Key);

}

public List<int> TopologySort()

{

if (IsAcyclic())

{

var topologyList = new List<int>();

states = new States[vertecesCount];

while(true)

{

bool flag = true;

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

{

if (states[i - 1] == States.Done)

continue;

if (HalfDegreesIn[i] > 0)

continue;

flag = false;

topologyList.Add(i);

states[i - 1] = States.Done;

foreach (var n in adjList[i])

if (states[n - 1] == States.New)

HalfDegreesIn[n]--;

}

if (flag)

break;

}

return topologyList;

}

return null;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphLabs\_2

{

class Program

{

static void Main(string[] args)

{

var graph = new DirectedGraph();

string path = "../../../Lab01-gr5-25.dat";

graph.InitGraph(path);

//Console.WriteLine("Матрица смежности: ");

//for (int i = 0; i < graph.adjMatrix.GetLength(0); i++)

//{

// for (int j = 0; j < graph.adjMatrix.GetLength(1); j++)

// {

// Console.Write(graph.adjMatrix[i, j]);

// }

// Console.WriteLine();

//}

Console.WriteLine("Списки смежности орграфа: ");

foreach (var v in graph.adjList.OrderBy(v => v.Key))

{

Console.Write(v.Key + ": ");

foreach (var n in v.Value)

Console.Write(n + ", ");

Console.WriteLine();

}

Console.WriteLine("Полустепени захода: ");

graph.GetHalfDegreesIn();

foreach (var v in graph.HalfDegreesIn)

{

Console.Write(v.Key + " имеет " + v.Value + " степеней захода;");

Console.WriteLine();

}

Console.WriteLine("Полустепени исхода: ");

graph.GetHalfDegreesOut();

foreach (var v in graph.HalfDegreesOut)

{

Console.Write(v.Key + " имеет " + v.Value + " степеней исхода;");

Console.WriteLine();

}

//Console.WriteLine("Инверсированные списки смежности: ");

//foreach (var v in graph.revAdjList.OrderBy(v => v.Key))

//{

// Console.Write(v.Key + ": ");

// foreach (var n in v.Value)

// Console.Write(n + ", ");

// Console.WriteLine();

//}

Console.WriteLine("Cписки достижимости для каждой вершины: ");

for (int v = 1; v <= graph.vertecesCount; v++)

{

var t = graph.Traverse(v);

Console.Write(v + " : ");

foreach (var n in t)

if (n != v)

Console.Write(n + " ");

Console.WriteLine();

}

Console.Write("Число компонент сильной связности = ");

var c = graph.Kasaraju();

Console.Write(c.Select((v, i) => new { v,i }).GroupBy(v => v.v).Where(e => e.Count() > 1).Count());

Console.WriteLine();

Console.WriteLine("Ацикличен ли граф? " + graph.IsAcyclic());

Console.WriteLine("//// Ациклический граф ////");

var asGraph = new DirectedGraph();

asGraph.InitGraph(@"../../../Lab02-gr5-25.dat");

Console.WriteLine("Проверка на ацикличность дала результат: " + asGraph.IsAcyclic());

Console.WriteLine("Топологическая фортировка: ");

var ts = asGraph.TopologySort();

foreach (var v in ts)

Console.Write(v + " ");

Console.ReadKey();

}

}

}

*Лабораторная работа №3*

using System;

using System.Collections.Generic;

using System.Linq;

using GraphLabs\_2;

namespace GraphLabs\_3

{

static class SomeGraphFuncs

{

public static List<List<int>> ToTiersForms(DirectedGraph graph)

{

var bagOfVisitedVertices = new List<int>(graph.HalfDegreesIn.Where(v => v.Value == 0).Select(v => v.Key).ToList());

var bagOfRemainVertices = new List<int>(graph.HalfDegreesIn.Where(v => v.Value > 0).Select(v => v.Key).ToList());

var tiers = new List<List<int>>();

tiers.Add(graph.HalfDegreesIn.Where(v => v.Value == 0).Select(v => v.Key).ToList());

while (bagOfVisitedVertices.Count < graph.vertecesCount)

{

var lastTier = tiers.Last();

tiers.Add(new List<int>());

//foreach (var v in bagOfRemainVertices)

int rv = 0;

while (rv < bagOfRemainVertices.Count)

{

bool flag = true;

var v = bagOfRemainVertices[rv];

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

if (graph.adjMatrixIns[v - 1, i] && (!bagOfVisitedVertices.Contains(i + 1)

|| tiers.Last().Contains(i + 1)))

flag = false;

if (flag)

{

tiers.Last().Add(v);

bagOfVisitedVertices.Add(v);

bagOfRemainVertices.Remove(v);

rv = bagOfRemainVertices.Count > 0 ? -1 : 0;

}

rv++;

}

}

return tiers;

}

public static dynamic GetMaxComponents(DirectedGraph graph)

{

var k = graph.Kasaraju();

var comps = k.Select((v, i) => new { v, i })

.GroupBy(e => e.v)

.Where(e => e.Count() > 1)

.OrderBy(e => e.Key)

.ToList();

foreach (var v in comps)

{

Console.Write(v.Key + "::");

foreach (var vv in v)

Console.Write((vv.i + 1) + " ");

Console.WriteLine();

}

return comps;

}

public static dynamic GetMetaGraph(DirectedGraph graph)

{

var k = graph.Kasaraju();

var comps = k.Select((v, i) => new { v, i })

.GroupBy(e => e.v)

.Where(e => e.Count() > 1)

.OrderBy(e => e.Key)

.ToList();

var metaV = comps.SelectMany(v => v.Select(e => e.i)).ToList();

var metaAdjList = new Dictionary<int, HashSet<int>>();

var metaAdjMatrix = new bool[comps.Count, comps.Count];

var tempAdjL = graph.adjList.Where(v => metaV.Contains(v.Key));

foreach (var c in comps)

{

metaAdjList.Add(c.Key, new HashSet<int>());

foreach (var x in tempAdjL)

if (x.Value.Any(e => c.Any(j => j.i + 1 == e)))

{

var t = comps.Where(co => co.Any(coo => x.Value.Any(vvv => vvv == coo.i + 1))).FirstOrDefault();

metaAdjList[c.Key].Add(t.Key);

}

}

foreach (var v in metaAdjList)

if (v.Value.Contains(v.Key))

v.Value.Remove(v.Key);

return metaAdjList;

}

}

}

using System;

using GraphLabs\_2;

namespace GraphLabs\_3

{

class Program

{

static void Main(string[] args)

{

var asGraph2 = new DirectedGraph();

asGraph2.InitGraph("../../../Lab02-gr5-25.dat");

Console.WriteLine("Ярусно-параллельная форма графа из лаб2: ");

var tf =SomeGraphFuncs.ToTiersForms(asGraph2);

for (int i = 0; i < tf.Count; i++)

{

foreach (var v in tf[i])

Console.Write(v + " ");

Console.WriteLine();

}

Console.WriteLine("Компоненты сильной связности: ");

var graph3 = new DirectedGraph();

graph3.InitGraph("../../../Lab03-gr5-25.dat");

SomeGraphFuncs.GetMaxComponents(graph3);

Console.WriteLine("Списки смежности метаграфа: ");

var some = SomeGraphFuncs.GetMetaGraph(graph3);

foreach (var v in some)

{

Console.Write(v.Key + ": ");

foreach (var n in v.Value)

Console.Write(n + ", ");

Console.WriteLine();

}

Console.ReadKey();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using GraphLabs\_2;

namespace GraphLabs\_3

{

static class SomeGraphFuncs

{

public static List<List<int>> ToTiersForms(DirectedGraph graph)

{

var bagOfVisitedVertices = new List<int>(graph.HalfDegreesIn.Where(v => v.Value == 0).Select(v => v.Key).ToList());

var bagOfRemainVertices = new List<int>(graph.HalfDegreesIn.Where(v => v.Value > 0).Select(v => v.Key).ToList());

var tiers = new List<List<int>>();

tiers.Add(graph.HalfDegreesIn.Where(v => v.Value == 0).Select(v => v.Key).ToList());

while (bagOfVisitedVertices.Count < graph.vertecesCount)

{

var lastTier = tiers.Last();

tiers.Add(new List<int>());

//foreach (var v in bagOfRemainVertices)

int rv = 0;

while (rv < bagOfRemainVertices.Count)

{

bool flag = true;

var v = bagOfRemainVertices[rv];

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

if (graph.adjMatrixIns[v - 1, i] && (!bagOfVisitedVertices.Contains(i + 1)

|| tiers.Last().Contains(i + 1)))

flag = false;

if (flag)

{

tiers.Last().Add(v);

bagOfVisitedVertices.Add(v);

bagOfRemainVertices.Remove(v);

rv = bagOfRemainVertices.Count > 0 ? -1 : 0;

}

rv++;

}

}

return tiers;

}

public static dynamic GetMaxComponents(DirectedGraph graph)

{

var k = graph.Kasaraju();

var comps = k.Select((v, i) => new { v, i })

.GroupBy(e => e.v)

.Where(e => e.Count() > 1)

.OrderBy(e => e.Key)

.ToList();

foreach (var v in comps)

{

Console.Write(v.Key + "::");

foreach (var vv in v)

Console.Write((vv.i + 1) + " ");

Console.WriteLine();

}

return comps;

}

public static dynamic GetMetaGraph(DirectedGraph graph)

{

var k = graph.Kasaraju();

var comps = k.Select((v, i) => new { v, i })

.GroupBy(e => e.v)

.Where(e => e.Count() > 1)

.OrderBy(e => e.Key)

.ToList();

var metaV = comps.SelectMany(v => v.Select(e => e.i)).ToList();

var metaAdjList = new Dictionary<int, HashSet<int>>();

var metaAdjMatrix = new bool[comps.Count, comps.Count];

var tempAdjL = graph.adjList.Where(v => metaV.Contains(v.Key));

foreach (var c in comps)

{

metaAdjList.Add(c.Key, new HashSet<int>());

foreach (var x in tempAdjL)

if (x.Value.Any(e => c.Any(j => j.i + 1 == e)))

{

var t = comps.Where(co => co.Any(coo => x.Value.Any(vvv => vvv == coo.i + 1))).FirstOrDefault();

metaAdjList[c.Key].Add(t.Key);

}

}

foreach (var v in metaAdjList)

if (v.Value.Contains(v.Key))

v.Value.Remove(v.Key);

return metaAdjList;

}

}

}

*Лабораторная работа №4*

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using GraphsLabs;

namespace GraphLabs\_4

{

class WeightedGraph

{ // взвешенный граф

public Dictionary<int, HashSet<Tuple<int, int>>> AdjLists { get; private set; } // списки смежности

private bool[] markers; // маркеры вершин

public int[] label { get; private set; } // принадлежность вершины компоненте связности

SimplifiedGraph F; // промежуточный лес

int n; // кол-во вершин

int m; // кол-во ребер

//string path;

List<int> maxComponent;

private void AddEdge(int i, int j, int w)

{

if (i > 0 && i <= n && j > 0 && j <= n)

{

AdjLists[i].Add(new Tuple<int, int>(j, w));

AdjLists[j].Add(new Tuple<int, int>(i, w));

}

}

public WeightedGraph(string path)

{

//this.path = path;

F = new SimplifiedGraph(new UndirectedGraph(path));

maxComponent = F.GetMaxComponent();

//this.n = maxComponent.Count;

var str = File.ReadAllText(path);

var strA = str.Split(new char[] { ' ', '\n' }).Where(s => s != string.Empty).ToArray();

int[] verts = Array.ConvertAll(strA, s => int.Parse(s));

n = verts[0];

//F = new SimplifiedGraph(n);

//adjMatrix = new int[vertecesCount, vertecesCount];

//adjMatrixIns = new bool[vertecesCount, vertecesCount];

AdjLists = new Dictionary<int, HashSet<Tuple<int, int>>>();

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

AdjLists.Add(i, new HashSet<Tuple<int, int>>());

for (int v = 1; v < verts.Length; v += 3)

{

//if (maxComponent.Contains(verts[v]) && maxComponent.Contains(verts[v + 1]))

this.AddEdge(verts[v], verts[v + 1], verts[v + 2]);

}

F = new SimplifiedGraph(this.n);

}

void AddAllSafeEdges(int count)

{

List<Tuple<int, int, int>> S = new List<Tuple<int, int, int>>();

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

{

S.Add(new Tuple<int, int, int>(0, 0, 9999));

}

//for (int u = 1; u <= n; u++)

foreach (var u in AdjLists.Keys)

foreach (var v in AdjLists[u])

if (maxComponent.Contains(u) && maxComponent.Contains(v.Item1) && maxComponent.Contains(v.Item2))

{

if (label[u - 1] != label[v.Item1 - 1])

{

if (v.Item2 < (S[label[u - 1]]).Item3)

S[label[u - 1]] = new Tuple<int, int, int>(u, v.Item1, v.Item2);

if (v.Item2 < (S[label[v.Item1 - 1]].Item3))

S[label[v.Item1 - 1]] = new Tuple<int, int, int>(u, v.Item1, v.Item2);

}

}

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

if (S[i].Item3 != 1000)

F.AddEdge(S[i].Item1, S[i].Item2);

F.PrintAdjLists();

Console.WriteLine("----------------");

}

public void Borvka()

{

this.label = new int[n];

int count = F.CountAndLabel(maxComponent);

while (count > 1)

{

label = F.label;

AddAllSafeEdges(count);

count = F.CountAndLabel(maxComponent);

Console.WriteLine("COUNT: " + count);

Console.WriteLine("----------------");

}

}

public void PrimaYarnik()

{

List<Tuple<int, int, int>> MST = new List<Tuple<int, int, int>>();

//неиспользованные ребра

List<Tuple<int, int, int>> notUsedE = new List<Tuple<int, int, int>>();

foreach (var v in this.AdjLists)

if (maxComponent.Contains(v.Key))

foreach (var n in v.Value)

if (maxComponent.Contains(n.Item1))

notUsedE.Add(new Tuple<int, int, int>(v.Key, n.Item1, n.Item2));

//использованные вершины

List<int> usedV = new List<int>();

//неиспользованные вершины

List<int> notUsedV = new List<int>(maxComponent);

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

// notUsedV.Add(i);

//выбираем случайную начальную вершину

Random rand = new Random();

usedV.Add(maxComponent[rand.Next(0, maxComponent.Count - 1)]);

notUsedV.Remove(usedV[0]);

while (notUsedV.Count > 0)

{

Tuple<int, int, int> minE = new Tuple<int, int, int> (usedV.Last(), 0 , 1000); //наименьшее ребро

//поиск наименьшего ребра

//for (int i = 0; i < notUsedE.Count; i++)

//{

var t = this.AdjLists[minE.Item1].Where(e => !usedV.Contains(e.Item1)).OrderBy(e => e.Item2).FirstOrDefault();

if (t == null)

{

int k = 0;

while (t == null)

{

k++;

var prevV = usedV[usedV.Count - k];

var elements = AdjLists[prevV]

.Where(e => !usedV.Contains(e.Item1))

.OrderBy(e => e.Item2)

.ToList();

t = elements.Count() > 0 ? elements.First() : null;

//Console.WriteLine("--" + usedV[usedV.Count - k]);

}

//foreach (var v in AdjLists[usedV[usedV.Count - k]].Where(e => !usedV.Contains(e.Item1)).OrderBy(e => e.Item2))

// Console.WriteLine(v + "; ");

//Console.WriteLine(usedV[usedV.Count - k] + " " + t.Item1 + " " + t.Item2);

minE = new Tuple<int, int, int>(usedV[usedV.Count - k], 0, 0);

}

minE = new Tuple<int, int, int>(minE.Item1, t.Item1, t.Item2);

//}

//заносим новую вершину в список использованных и удаляем ее из списка неиспользованных

//if (usedV.IndexOf(notUsedE[minE].Item1) != -1)

//{

usedV.Add(minE.Item2);

notUsedV.Remove(minE.Item2);

//}

//else

//{

// usedV.Add(notUsedE[minE].Item1);

// notUsedV.Remove(notUsedE[minE].Item1);

//}

//заносим новое ребро в дерево и удаляем его из списка неиспользованных

MST.Add(minE);

//notUsedE.RemoveAt(minE);

}

foreach (var m in MST)

{

Console.Write(m + " ");

Console.WriteLine();

}

}

public void Kraskal()

{

List<Tuple<int, int, int>> result = new List<Tuple<int, int, int>>();

List<Tuple<int, int, int>> edges = new List<Tuple<int, int, int>>();

foreach (var v in this.AdjLists)

if (maxComponent.Contains(v.Key))

foreach (var n in v.Value)

if (maxComponent.Contains(n.Item1))

edges.Add(new Tuple<int, int, int>(v.Key, n.Item1, n.Item2));

edges = edges.OrderBy(e => e.Item3).ToList();

int m = edges.Count();

List<int> treeID = new List<int>();

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

treeID.Add(i);

int cost = 0;

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

{

int a = edges[i].Item1;

int b = edges[i].Item2;

int l = edges[i].Item3;

if (treeID[a - 1] != treeID[b - 1])

{

cost += l;

result.Add(new Tuple<int, int, int>(a, b, l));

int oldID = treeID[a - 1];

int newID = treeID[b - 1];

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

if (treeID[j] == oldID)

treeID[j] = newID;

}

}

foreach (var v in result)

{

Console.Write("(" + v.Item1 + " " + v.Item2 + " ~ " + v.Item3 + "); ");

Console.WriteLine();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GraphsLabs;

namespace GraphLabs\_4

{

class Program

{

static void Main(string[] args)

{

WeightedGraph wg = new WeightedGraph("../../../Lab04-gr5-25.dat");

Console.WriteLine("//// Borvka ////");

wg.Borvka();

Console.WriteLine();

Console.WriteLine("//// PRIMA-YARNIK ////");

wg.PrimaYarnik();

Console.WriteLine();

Console.WriteLine("//// KRUSKAL ////");

wg.Kraskal();

Console.ReadKey();

}

}

}

*Лабораторная работа №5*

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphLabs\_5

{

class DirectedWeightedGraph

{

public int[,] adjMatrix { get; private set; }

public Dictionary<int, HashSet<Tuple<int, int>>> adjList { get; private set; }

public int vertecesCount { get; private set; }

private bool[] markers;

private Stack<int> vertStack;

private int[] label;

private int[] dist;

private int[] pred;

public DirectedWeightedGraph(string path)

{

this.InitGraph(path);

}

private void AddEdgeToMatrix(int i, int j, int w)

{

if (i > 0 && i <= vertecesCount && j > 0 && j <= vertecesCount)

{

adjMatrix[i - 1, j - 1] += w;

}

}

private void AddArcToList(int v, int n, int w)

{

if (v >= 0 && n <= vertecesCount && v >= 0 && n <= vertecesCount)

{

adjList[v].Add(new Tuple<int, int>(n, w));

}

}

public void InitGraph(string path)

{

var str = File.ReadAllText(path);

var strA = str.Split(new char[] { ' ', '\n' }).Where(s => s != string.Empty).ToArray();

int[] verts = Array.ConvertAll(strA, s => int.Parse(s));

vertecesCount = verts[0];

adjMatrix = new int[vertecesCount, vertecesCount];

adjList = new Dictionary<int, HashSet<Tuple<int, int>>>();

for (int v = 0; v < vertecesCount; v++)

{

adjList.Add(v, new HashSet<Tuple<int, int>>());

}

for (int v = 1; v < verts.Length; v += 3)

{

AddEdgeToMatrix(verts[v], verts[v + 1], verts[v + 2]);

AddArcToList(verts[v], verts[v + 1], verts[v + 2]);

}

}

public void FordWithQueue(int s)

{

dist = new int[vertecesCount]; // оценки длин путей

pred = new int[vertecesCount]; // предшествующая вершина

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

{

dist[i] = int.MaxValue;

pred[i] = -1;

}

dist[s] = 0;

//stack<int> bag;

Queue<int> bag = new Queue<int>();

bag.Enqueue(s);

//var T = new Tuple<int, int>();

//priority\_queue<T, vector<T>, greater<T> > bag;

//bag.push(make\_pair(0, s));

while (bag.Count > 0)

{

//int u = bag.top(); // for stack

int u = bag.Dequeue(); // for queue

//T p = bag.top(); // for priority\_queue

//int u = p.second;

//bag.pop();

foreach (var x in adjList[u])

{

int v = x.Item1, w = x.Item2;

if (dist[u] + w < dist[v])

{

dist[v] = dist[u] + w;

pred[v] = u;

bag.Enqueue(v);

//bag.push(make\_pair(dist[v], v));

}

}

}

}

public void FordWithStack(int s)

{

dist = new int[vertecesCount]; // оценки длин путей

pred = new int[vertecesCount]; // предшествующая вершина

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

{

dist[i] = int.MaxValue;

pred[i] = -1;

}

dist[s] = 0;

Stack<int> bag = new Stack<int>();

bag.Push(s);

while (bag.Count > 0)

{

int u = bag.Pop();

foreach (var x in adjList[u])

{

int v = x.Item1, w = x.Item2;

if (dist[u] + w < dist[v])

{

dist[v] = dist[u] + w;

pred[v] = u;

bag.Push(v);

}

}

}

}

public void FordWithPriorityQueue(int s)

{

dist = new int[vertecesCount]; // оценки длин путей

pred = new int[vertecesCount]; // предшествующая вершина

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

{

dist[i] = int.MaxValue;

pred[i] = -1;

}

dist[s] = 0;

PriorityQueue<int, int> bag = new PriorityQueue<int, int>();

bag.Enqueue(0, s);

while (bag.Count > 0)

{

var p = bag.Dequeue();

int u = p.Value;

foreach (var x in adjList[u])

{

int v = x.Item1, w = x.Item2;

if (dist[u] + w < dist[v])

{

dist[v] = dist[u] + w;

pred[v] = u;

bag.Enqueue(dist[v], v);

}

}

}

}

public void PrintShortWay()

{

dist = dist.OrderBy(e => e).ToArray();

for (int v = 0; v < vertecesCount; v++)

{

if (dist[v] < int.MaxValue)

{

Console.Write(dist[v] + " : " + v);

for (int x = v; pred[x] >= 0; x = pred[x])

Console.Write("<-" + pred[x]);

Console.WriteLine();

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GraphLabs\_5

{

class Program

{

static void Main(string[] args)

{

DirectedWeightedGraph dwg = new DirectedWeightedGraph("../../../Lab05-gr5-25.dat");

Console.WriteLine("Короткий путь из вершины 0 методом Форда со стеком");

dwg.FordWithStack(0);

dwg.PrintShortWay();

Console.WriteLine("Короткий путь из вершины 0 методом Форда с очередью");

dwg.FordWithQueue(0);

dwg.PrintShortWay();

Console.WriteLine("Короткий путь из вершины 0 методом Форда с приоритетной очередью");

dwg.FordWithPriorityQueue(0);

dwg.PrintShortWay();

Console.ReadLine();

}

}

}