| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |
| --- |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-2 «Прикладные информационные технологии»

**ОТЧЕТ   
о выполнении лабораторной работы №2**

**«Реализация абстрактного типа данных «Граф»»**

**по дисциплине   
«Технологии и методы программирования»**

**Вариант № 77**

Выполнил: студент 2 курса

группы БИСО-03-20

шифр 20Б0877

Рудзик Александр Романович  
*(фио студента)*

Проверил:

*Филатов В.В.*

Москва 2021 г.

**Задание**

Реализовать в виде программы абстрактный тип данных «Граф» согласно варианту (***Номер варианта*** – **две последние цифры шифра студента, номера зачетной книжки**) с учетом заданного представления графа. Операторы (операции) АТД «Граф» *функционально* должны выполнять следующие операции (названия операций – *примерные*) ***(1 балл из 5)***:

1. FIRST(*v*) - возвращает ***индекс*** первой вершины, смежной с вершиной *v.* Если вершина *v* не имеет смежных вершин, то возвращается "нулевая" вершина Λ.
2. NEXT(*v*, *i*)- возвращает ***индекс*** вершины, смежной с вершиной *v*, следующий за индексом *i*. Если *i —* это индекс последней вершины, смежной с вершиной *v*, то возвращается Λ.
3. VERTEX(*v*, *i*) - возвращает ***вершину*** с индексом *i* из множества вершин, смежных с *v.*
4. ADD\_V(<имя>,<метка, mark>) - добавить УЗЕЛ
5. ADD\_Е(v, w, c) - добавить ДУГУ (здесь c — вес, цена дуги (v,w))
6. DEL\_V(<имя>) - удалить УЗЕЛ
7. DEL\_Е(v, w) – удалить ДУГУ
8. EDIT\_V(<имя>, <новое значение метки или маркировки>) - изменить метку (маркировку) УЗЛА

EDIT\_Е(v, w, <новый вес дуги>) - изменить вес ДУГИ

Алгоритм: Мостом графа G называется каждое ребро, удаление которого приводит к увеличению числа связных компонент графа. Представить алгоритм нахождения всех мостов графа

Способ представления графа: список смежности

**Теоретическое введение**

**Графом** называется система объектов произвольной природы (вершин) и связок (ребер), соединяющих некоторые пары этих объектов.

**Мостом** **в** неориентированном **графе** называется ребро, при удалении которого количество компонент связности **графа** увеличивается.

**Список** **смежности** — один из способов представления **графа** в виде коллекции **списков** вершин. Каждой вершине **графа** соответствует **список**, состоящий из «соседей» этой вершины.

**Листинг программы с расчетами**

using System;

using System.Collections.Generic;

namespace Project

{

class Project

{

public class Line

{

public Dot connected1 { get; }

public Dot connected2 { get; }

public List<Dot> connects { get; }

public int weight { get; set; }

public Line(Dot connect1, Dot connect2, int wieght)

{

connected1 = connect1;

connected2 = connect2;

connects = new List<Dot>();

connects.Add(connect1);

connects.Add(connect2);

weight = wieght;

}

public override string ToString()

{

return (connected1.name + "-" + connected2.name);

}

};

public class Dot

{

public string name { get; set; }

public List<Line> Lines { get; }

public Dot(string Name)

{

name = Name;

Lines = new List<Line>();

}

public Dot() { }

public void AddLine(Line newLine)

{

Lines.Add(newLine);

}

public void AddLine(Dot newDot, int weight)

{

Lines.Add(new Line(this, newDot, weight));

}

public void delDotLine(string v, string w)

{

if (this.name == v || this.name == w)

{

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

{

if (this.Lines[i].connects[0].name == w || this.Lines[i].connects[1].name == w)

{

this.Lines.RemoveAt(i);

}

}

}

}

};

public class Graph

{

public List<Dot> Dots { get; }

public List<Line> Lines { get; }

public Dot B { get; set; }

public Graph()

{

Dots = new List<Dot>();

Lines = new List<Line>();

B = new Dot();

}

public void AddDot(string name)

{

Dots.Add(new Dot(name));

}

public void setEnd()

{

B = this.Dots[this.Dots.Count - 1];

}

public Dot FindDot(string name)

{

foreach (var d in Dots)

{

if (d.name.Equals(name))

{

return d;

}

}

return null;

}

public void AddLine(string Dot1, string Dot2, int weight)

{

var d1 = FindDot(Dot1);

var d2 = FindDot(Dot2);

if (d1 != null && d2 != null)

{

d1.AddLine(d2, weight);

d2.AddLine(d1, weight);

Lines.Add(new Line(d1, d2, weight));

}

}

public int FirstMatch(string matchName)

{

foreach (var line in Lines)

{

if (line.connects[0].name == matchName)

{

return Dots.IndexOf(line.connects[1]);

}

}

return 0;

}

public int NextMatch(string matchName, int IndexNeeded)

{

int count = -1;

foreach (var line in Lines)

{

if (line.connects[0].name == matchName)

{

count++;

if (count > IndexNeeded)

{

return Dots.IndexOf(line.connects[1]);

}

}

}

return 0;

}

public Dot Vertex(string matchName, int indexMatch)

{

int count = -1;

foreach (var line in Lines)

{

if (line.connects[0].name == matchName)

{

count++;

if (count == indexMatch)

{

return line.connects[1];

}

}

}

Dot oshibka = new Dot("Error");

return oshibka;

}

public void delDot(string matchName)

{

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

{

if (Lines[i].connects[0].name == matchName || Lines[i].connects[1].name == matchName)

{

Lines.Remove(Lines[i]);

break;

}

}

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

{

if (Dots[i].name == matchName)

{

Dots.Remove(Dots[i]);

}

}

}

public void delLine(string v, string w)

{

for (int j = 0; j < this.Dots.Count; j++)

{

if (this.Dots[j].name == v || this.Dots[j].name == w)

{

for (int i = 0; i < Dots[j].Lines.Count; i++)

{

if (this.Dots[j].Lines[i].connects[0].name == v && this.Dots[j].Lines[i].connects[1].name == w)

{

this.Dots[j].Lines.RemoveAt(i);

}

}

}

}

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

{

if (Lines[i].connects[0].name == v && Lines[i].connects[1].name == w)

{

Lines.Remove(Lines[i]);

break;

}

}

}

public void editLine(string v, string w, int newWeight)

{

foreach (var line in Lines)

{

if (line.connects[0].name == v && line.connects[1].name == w)

{

line.weight = newWeight;

}

}

}

public void PrintGrapgh()

{

List<List<Dot>> spisok = new List<List<Dot>>();

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

{

List<Dot> podspisok = new List<Dot>();

for(int j = 0; j<Dots[i].Lines.Count; j++)

{

podspisok.Add(Dots[i].Lines[j].connected2);

}

spisok.Add(podspisok);

}

Console.WriteLine("Список смежности: ");

Console.WriteLine(new string('-', 20));

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

{

Console.Write(Dots[i].name + " --> ");

for (int j = 0; j < Dots[i].Lines.Count; j++)

{

Console.Write(spisok[i][j].name + " ");

}

Console.WriteLine();

}

Console.WriteLine(new string('-', 20));

}

public void keyFunc()

{

List<Line> most = new List<Line>();

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

{

Graph temp = new Graph();

foreach (var dot in this.Dots)

{

temp.Dots.Add(new Dot(dot.name));

}

foreach (var line in this.Lines)

{

string con1 = line.connects[0].name;

string con2 = line.connects[1].name;

int weight = line.weight;

temp.AddLine(con1, con2, weight);

}

temp.delLine(Lines[i].connected1.name, Lines[i].connected2.name);

bool[] used = new bool[Dots.Count];

used = temp.existingWay(temp.Dots[0]);

if (!used[Dots.Count - 1])

{

most.Add(Lines[i]);

}

}

if (most.Count > 0)

{

Console.WriteLine("Все мосты графа:");

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

{

Console.Write(most[i] + " ");

}

}

else

{

Console.WriteLine("Мостов в графе нет!");

}

}

public bool[] existingWay(Dot u, bool[] used = null)

{

if (used == null)

{

used = new bool[this.Dots.Count];

}

used[this.Dots.IndexOf(u)] = true;

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

q.Enqueue(u);

while (q.Count > 0)

{

u = q.Peek();

q.Dequeue();

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

{

Dot v = u.Lines[i].connects[1];

if (!used[Dots.IndexOf(v)])

{

used[Dots.IndexOf(v)] = true;

q.Enqueue(v);

}

}

}

return used;

}

};

static void Main(string[] args)

{

Graph graph = new Graph();

graph.AddDot("A");

graph.AddDot("B");

graph.AddDot("C");

graph.AddDot("D");

graph.AddDot("E");

graph.AddDot("F");

graph.AddDot("G");

graph.AddLine("A", "B", 10);

graph.AddLine("A", "C", 10);

graph.AddLine("B", "C", 10);

graph.AddLine("B", "D", 10);

graph.AddLine("D", "E", 10);

graph.AddLine("D", "F", 10);

graph.AddLine("E", "F", 10);

graph.AddLine("F", "G", 10);

graph.PrintGrapgh();

graph.keyFunc();

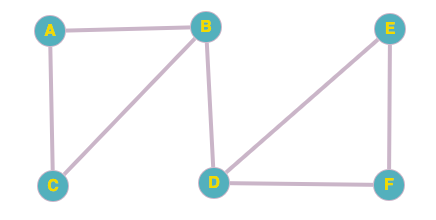
}

}

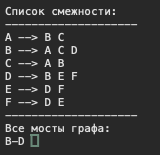
}

**Контрольные примеры**

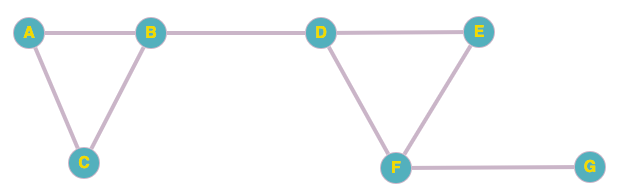
1)



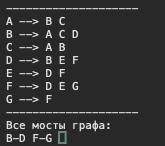
Вывод программы:



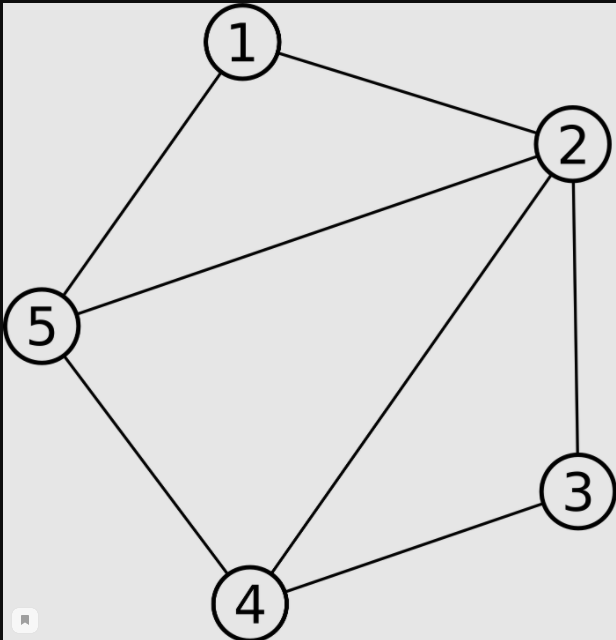
2)



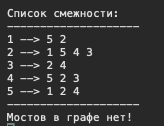
Вывод программы:



3)



Вывод программы:



**Выводы**

Изучил абстрактный тип данных «Граф», понятие «Мост графа» и реализовал алгоритм нахождения всех мостов графа на языке С# с выводом через список смежности. Проверил алгоритм на различных примерах, что значит о правильности выполнения алгоритма.