Пензенский государственный университет  
Кафедра «Вычислительной техники»

**Отчет**по лабораторной работе №3  
по курсу "ЛиОАвИЗ"

**Выполнили студент группы 20ВВ1:**

Вяльмисов М.О.

**Приняли:**

Юрова О.В.

Акифьев И.В.

Пенза 2021

**Лабораторные задания:**

### **Задание 1**

1. Сгенерируйте (используя генератор случайных чисел) две матрицы *M*1*, М*2 смежности неориентированных помеченных графов *G*1, *G*2. Выведите сгенерированные матрицы на экран.
2. \* Для указанных графов преобразуйте представление матриц смежности в списки смежности. Выведите полученные списки на экран.

### **Задание 2**

1. Для матричной формы представления графов выполните операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

1. \* Для представления графов в виде списков смежности выполните операцию:

а) отождествления вершин

б) стягивания ребра

в) расщепления вершины

Номера выбираемых для выполнения операции вершин ввести с клавиатуры.

Результат выполнения операции выведите на экран.

**Задание 3**

1. Для матричной формы представления графов выполните операцию:

а) объединения *G* = *G*1  *G*2

б) пересечения *G* = *G*1  *G*2

в) кольцевой суммы *G* = *G*1  *G*2

Результат выполнения операции выведите на экран.

**Листинг программы:**

Класс Programm:

using System;

using Laba3\_.Graphs;

namespace Laba3\_

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("\t \t \t Добро пожаловать!!!");

Console.WriteLine("\t \t \t 1- Создать граф \n \t \t \t 2- Отождествление вершин у графа " +

"\n \t \t \t 3- Расщепление вершины у графа \n \t \t \t 4- Oбъединение со случайным графом " +

"\n \t \t \t 5- Пересечение со случайным графом \n \t \t \t 6- Кольцевая сумма графов \n \t \t \t" +

"5 - Декартовое произведение \n \t \t \t");

Console.WriteLine();

Program program = new Program();

int operation;

do

{

Console.WriteLine();

Console.Write("\t \t \t Введите номер операции: ");

operation = int.Parse(Console.ReadLine());

switch (operation)

{

case 0:

break;

case 1:

program.CreateGrahs();

break;

case 2:

program.VertexContraction();

break;

case 3:

program.SplitVertex();

break;

case 4:

program.UnionGraphs();

break;

case 5:

program.CrossingGraphs();

break;

case 6:

program.AnnularSum();

break;

case 7:

program.DecartSum();

break;

default:

Console.WriteLine("Неизвестная операция ");

break;

}

} while (operation != 0);

Console.WriteLine();

Console.WriteLine("\t \t \t Работа завершена.");

}

MatrixGraph \_myMatrixGraph;

private ListGraph \_myListGraph;

public void CreateGrahs()

{

Console.WriteLine("Введите размер графа: ");

int size = int.Parse(Console.ReadLine());

\_myMatrixGraph = new MatrixGraph(size);

\_myListGraph = new ListGraph(\_myMatrixGraph, size);

Console.WriteLine();

Console.WriteLine("Граф в матричной форме: ");

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

Console.WriteLine("Граф в форме списка: ");

ListGraph.Display(\_myListGraph);

}

public void VertexContraction()

{

if (\_myMatrixGraph == null || \_myListGraph == null)

{

Console.WriteLine("Какой-либо граф отсутствует");

return;

}

Console.WriteLine();

Console.WriteLine("Введите вершины для отождествления");

Console.WriteLine("v1: ");

int v1 = int.Parse(Console.ReadLine());

Console.WriteLine("v2: ");

int v2 = int.Parse(Console.ReadLine());

\_myMatrixGraph.VertexContraction(v1, v2);

\_myListGraph.VertexContraction(v1, v2);

Console.WriteLine();

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

ListGraph.Display(\_myListGraph);

}

public void SplitVertex()

{

if (\_myMatrixGraph == null || \_myListGraph == null)

{

Console.WriteLine("Какой-либо граф отсутствует");

return;

}

Console.WriteLine();

Console.WriteLine("Введите вершинy для расщепления: ");

int v = int.Parse(Console.ReadLine());

\_myMatrixGraph.SplitVertex(v);

\_myListGraph.SplitVertex(v);

Console.WriteLine();

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

ListGraph.Display(\_myListGraph);

}

public void UnionGraphs()

{

if (\_myMatrixGraph == null || \_myListGraph == null)

{

Console.WriteLine("Какой-либо граф отсутствует");

return;

}

Console.WriteLine("Введите размер графа для объединения: ");

int size = int.Parse(Console.ReadLine());

MatrixGraph myNewMatrixGraph = new MatrixGraph(size);

Console.WriteLine();

Console.WriteLine("Граф для объединения в матричной форме: ");

MatrixGraph.Display(myNewMatrixGraph);

Console.WriteLine();

\_myMatrixGraph = MatrixGraph.Union(\_myMatrixGraph, myNewMatrixGraph);

\_myListGraph = new ListGraph(\_myMatrixGraph, size);

Console.WriteLine();

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

ListGraph.Display(\_myListGraph);

}

public void CrossingGraphs()

{

if (\_myMatrixGraph == null || \_myListGraph == null)

{

Console.WriteLine("Какой-либо граф отсутствует");

return;

}

Console.WriteLine("Введите размер графа для пересечения: ");

int size = int.Parse(Console.ReadLine());

MatrixGraph myNewMatrixGraph = new MatrixGraph(size);

Console.WriteLine();

Console.WriteLine("Граф для пересечения в матричной форме: ");

MatrixGraph.Display(myNewMatrixGraph);

Console.WriteLine();

\_myMatrixGraph = MatrixGraph.Crossing(\_myMatrixGraph, myNewMatrixGraph);

\_myListGraph = new ListGraph(\_myMatrixGraph, \_myMatrixGraph.Size);

Console.WriteLine();

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

ListGraph.Display(\_myListGraph);

}

public void AnnularSum()

{

if (\_myMatrixGraph == null || \_myListGraph == null)

{

Console.WriteLine("Какой-либо граф отсутствует");

return;

}

Console.WriteLine("Введите размер графа для кольцевой суммы: ");

int size = int.Parse(Console.ReadLine());

MatrixGraph myNewMatrixGraph = new MatrixGraph(size);

Console.WriteLine();

Console.WriteLine("Граф для кольцевой суммы в матричной форме: ");

MatrixGraph.Display(myNewMatrixGraph);

Console.WriteLine();

\_myMatrixGraph = MatrixGraph.AnnularSum(\_myMatrixGraph, myNewMatrixGraph);

\_myListGraph = new ListGraph(\_myMatrixGraph, size);

Console.WriteLine();

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

ListGraph.Display(\_myListGraph);

}

public void DecartSum()

{

if (\_myMatrixGraph == null || \_myListGraph == null)

{

Console.WriteLine("Какой-либо граф отсутствует");

return;

}

Console.WriteLine("Введите размер графа для Декартового произведения: ");

int size = int.Parse(Console.ReadLine());

MatrixGraph myNewMatrixGraph = new MatrixGraph(size);

Console.WriteLine();

Console.WriteLine("Граф для Декартового произведения в матричной форме: ");

MatrixGraph.Display(myNewMatrixGraph);

Console.WriteLine();

\_myMatrixGraph = MatrixGraph.DecartSumm(\_myMatrixGraph, myNewMatrixGraph);

\_myListGraph = new ListGraph(\_myMatrixGraph, size);

Console.WriteLine();

MatrixGraph.Display(\_myMatrixGraph);

Console.WriteLine();

ListGraph.Display(\_myListGraph);

}

}

}

Класс MatrixGraph:

using System;

namespace Laba3\_.Graphs

{

class MatrixGraph

{

private int[,] \_matrix;

private int \_size;

public int[,] Matrix

{

get { return \_matrix; }

set

{

\_matrix = value;

\_size = Convert.ToInt32(Math.Sqrt(\_matrix.Length));

}

}

public int Size

{

get { return \_size; }

}

public MatrixGraph(int size)

{

\_size = size;

\_matrix = new int[\_size, \_size];

Random random = new Random();

for (int i = 0; i < \_size; i++)

{

for (int j = i + 1; j < \_size; j++)

{

\_matrix[i, j] = random.Next(0, 2);

}

}

for (int i = 0; i < \_size; i++)

{

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

{

\_matrix[i, j] = \_matrix[j, i];

}

}

}

public void VertexContraction(int v1, int v2)

{

if (v1 == v2) return;

int v1Index = v1-1;

int v2Index = v2-1;

int[,] newMatrix = new int[\_size - 1, \_size - 1];

if (\_matrix[v1Index, v2Index] == 1) //создание петли

{

\_matrix[v1Index, v1Index] = 1;

}

for (int i = 0; i < \_size; i++) // прикрепили все связи к одной вершине

{

if (\_matrix[v1Index, i] == 0)

{

\_matrix[v1Index, i] += \_matrix[v2Index, i];

}

if (\_matrix[i, v1Index] == 0)

{

\_matrix[i, v1Index] = \_matrix[i, v2Index];

}

}

for (int i = 0; i < v2Index; i++) //удалили ребро и перезаписали матрицу

{

for (int j = 0; j < \_size; j++)

{

if (j < v2Index)

{

newMatrix[i, j] = \_matrix[i, j];

}

else if (j > v2Index)

{

newMatrix[i, j - 1] = \_matrix[i, j];

}

}

}

for (int i = v2Index + 1; i < \_size; i++)

{

for (int j = 0; j < \_size; j++)

{

if (j < v2Index)

{

newMatrix[i - 1, j] = \_matrix[i, j];

}

if (j > v2Index)

{

newMatrix[i - 1, j - 1] = \_matrix[i, j];

}

}

}

\_matrix = newMatrix;

\_size--;

}

public void SplitVertex(int v1)

{

v1--;

int[,] newMatrix = new int[\_size + 1, \_size + 1];

for (int i = 0; i <= v1; i++) //перезаписали матрицу с добавлением новой вершины

{

for (int j = 0; j < \_size; j++)

{

if (j <= v1)

{

newMatrix[i, j] = \_matrix[i, j];

}

if (j > v1)

{

newMatrix[i, j + 1] = \_matrix[i, j];

}

}

}

for (int i = v1 + 1; i < \_size; i++)

{

for (int j = 0; j < \_size; j++)

{

if (j <= v1)

{

newMatrix[i + 1, j] = \_matrix[i, j];

}

if (j > v1)

{

newMatrix[i + 1, j + 1] = \_matrix[i, j];

}

}

}

\_matrix = newMatrix;

\_size++;

for (int i = 0; i < \_size; i++) //скопировали в новую вершину связи

{

\_matrix[i, v1 + 1] = \_matrix[i, v1];

\_matrix[v1 + 1, i] = \_matrix[v1, i];

}

\_matrix[v1, v1 + 1] = 1;

\_matrix[v1 + 1, v1] = 1;

}

public static MatrixGraph Union(MatrixGraph matrix1, MatrixGraph matrix2)

{

int newSize = matrix1.Size > matrix2.Size ? matrix1.Size : matrix2.Size;

int[,] newMatrix = new int[newSize, newSize];

for (int i = 0; i < matrix1.Size; i++)

{

for (int j = 0; j < matrix1.Size; j++)

{

newMatrix[i, j] = matrix1.Matrix[i, j];

}

}

for (int i = 0; i < matrix2.Size; i++)

{

for (int j = 0; j < matrix2.Size; j++)

{

if (matrix2.Matrix[i, j] > newMatrix[i, j])

{

newMatrix[i, j] = matrix2.Matrix[i, j];

}

}

}

MatrixGraph newGraph = new MatrixGraph(newSize);

newGraph.Matrix = newMatrix;

return newGraph;

}

public static MatrixGraph Crossing(MatrixGraph matrix1, MatrixGraph matrix2)

{

int newSize = matrix1.Size < matrix2.Size ? matrix1.Size : matrix2.Size;

int virtualSize = matrix1.Size > matrix2.Size ? matrix1.Size : matrix2.Size;

int[,] newMatrix = new int[virtualSize, virtualSize];

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

{

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

{

newMatrix[i, j] = matrix1.Matrix[i, j];

}

}

for (int i = 0; i < newSize; i++) //

{

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

{

if (matrix2.Matrix[i, j] > newMatrix[i, j])

{

newMatrix[i, j] = matrix2.Matrix[i, j];

}

}

}

MatrixGraph newGraph = new MatrixGraph(newSize);

newGraph.Matrix = newMatrix;

return newGraph;

}

public static MatrixGraph AnnularSum(MatrixGraph matrix1, MatrixGraph matrix2)

{

int newSize = matrix1.Size > matrix2.Size ? matrix1.Size : matrix2.Size;

int[,] newMatrix = new int[newSize, newSize];

for (int i = 0; i < matrix1.Size; i++)

{

for (int j = 0; j < matrix1.Size; j++)

{

newMatrix[i, j] = matrix1.Matrix[i, j];

}

}

for (int i = 0; i < matrix2.Size; i++)

{

for (int j = 0; j < matrix2.Size; j++)

{

newMatrix[i, j] += matrix2.Matrix[i, j];

if (newMatrix[i, j] > 1)

{

newMatrix[i, j] = 0;

}

}

}

MatrixGraph newGraph = new MatrixGraph(newSize);

newGraph.Matrix = newMatrix;

return newGraph;

}

public static MatrixGraph DecartSumm(MatrixGraph matrix1, MatrixGraph matrix2)

{

return null;

}

public static void Display(MatrixGraph graph)

{

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

{

for (int j = 0; j < graph.Size; j++)

{

Console.Write(graph.Matrix[i, j] + " ");

}

Console.WriteLine();

}

}

}

}

Класс ListGraph:

using System;

using System.Collections.Generic;

namespace Laba3\_.Graphs

{

class ListGraph

{

private List<List<int>> \_list;

public List<List<int>> List

{

get { return \_list; }

}

public ListGraph(MatrixGraph matrixGraph, int size)

{

\_list = new List<List<int>>(size);

int[,] matrix = matrixGraph.Matrix;

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

{

\_list.Add(new List<int>());

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

{

if (matrix[i, j] == 1)

{

\_list[i].Add(j + 1);

}

}

}

}

public void VertexContraction(int v1, int v2)

{

int v1Inndex = v1 - 1;

int v2Inndex = v2 - 1;

for (int i = 0;

i < \_list[v2Inndex].Count;

i++) // перенос из одной вершины всех соединений в другую и сортировка

{

if (\_list[v2Inndex][i] != v2)

{

\_list[v1Inndex].Add(\_list[v2Inndex][i]);

}

else

{

\_list[v1Inndex].Add(v1);

}

}

List[v1Inndex].Sort();

for (int i = 0; i < \_list[v1Inndex].Count - 1; i++)

{

if (\_list[v1Inndex][i] == \_list[v1Inndex][i + 1])

{

\_list[v1Inndex].RemoveAt(i + 1);

}

}

List[v1Inndex].Sort();

List.RemoveAt(v2 - 1);

for (int i = 0; i < \_list.Count; i++) //запись новой вершины в другие

{

bool isHaveV1 = false;

for (int j = 0; j < \_list[i].Count; j++) // проверка на наличие связи с v1

{

if (List[i][j] == v1)

{

isHaveV1 = true;

}

}

for (int j = 0; j < \_list[i].Count; j++) // замена или удаление связей

{

if (\_list[i][j] == v2 && !isHaveV1)

{

\_list[i][j] = v1;

}

else if (\_list[i][j] == v2 && isHaveV1)

{

\_list[i].RemoveAt(j);

}

else if (\_list[i][j] > v2)

{

\_list[i][j]--;

}

}

}

}

public void SplitVertex(int v)

{

int vIndex = v - 1;

\_list.Insert(vIndex + 1, \_list[vIndex]);

foreach (List<int> l in \_list)

{

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

{

if (l[i] > v)

{

l[i]++;

}

}

int j = l.IndexOf(v);

if (j > -1)

{

l.Insert(j + 1, v + 1);

}

}

\_list[v].Add(v+1);

\_list[v].Sort();

\_list[v+1].Add(v);

\_list[v+1].Sort();

}

public static void Display(ListGraph graph)

{

List<List<int>> list = graph.List;

foreach (List<int> v in list)

{

if (v.Count == 0) Console.Write("-");

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

{

Console.Write(v[j] + " ");

}

Console.WriteLine();

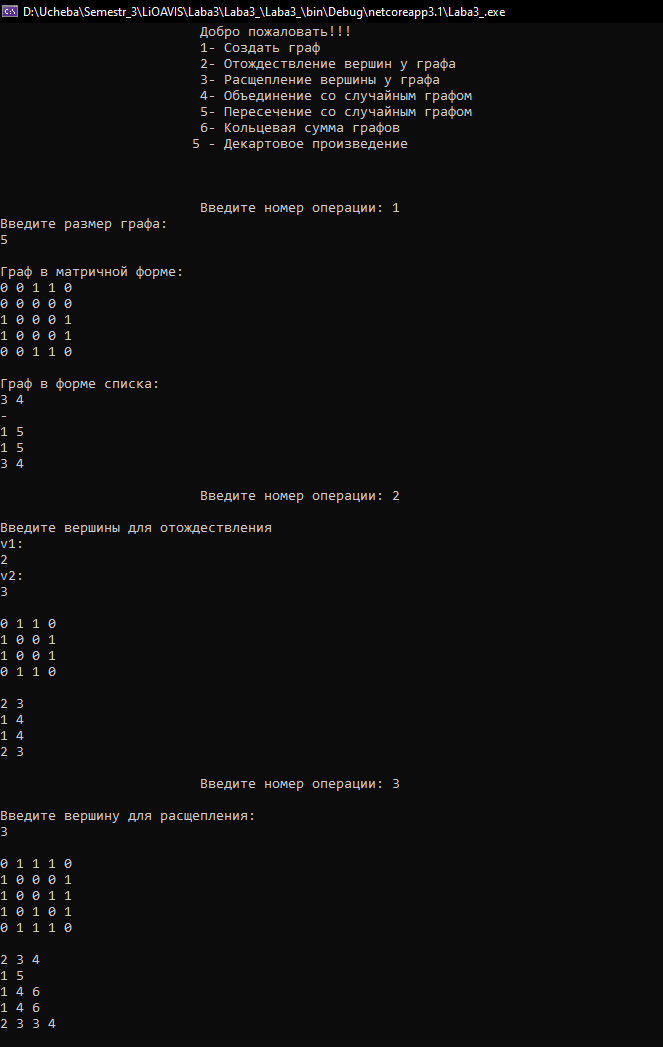
}

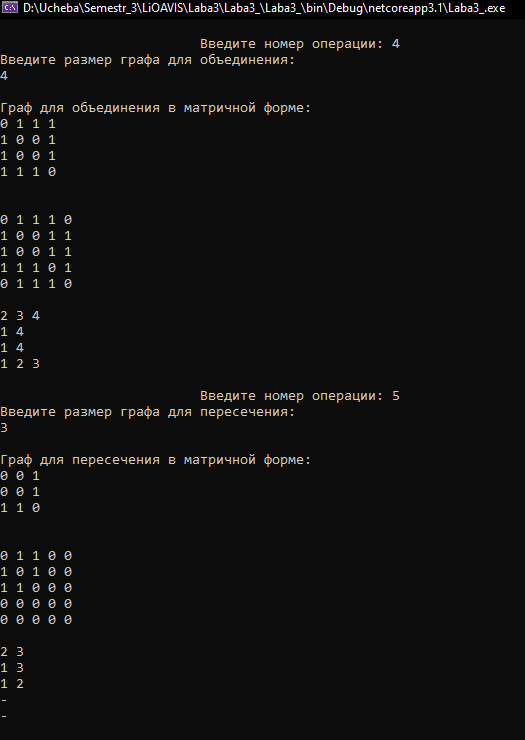
}

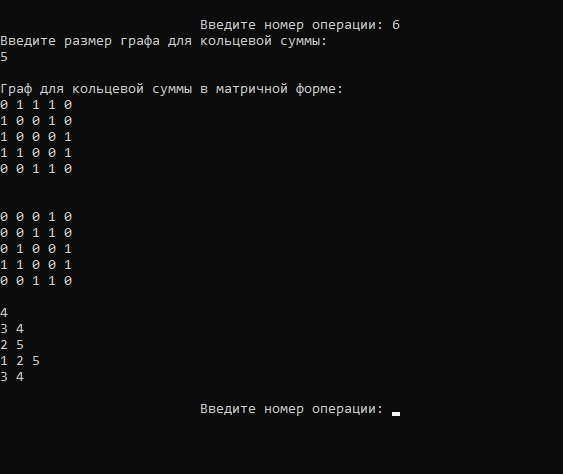
}

}

**Результаты выполнения операций:**







**Вывод:** Изучил основные операции над графом и научился работать с графом в матричной и в форме списка.