Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Унарные и бинарные операции над графами»

Выполнили:

студенты группы 24ВВВ3

Агапов И.А.

Любченко В.К.

Приняли:

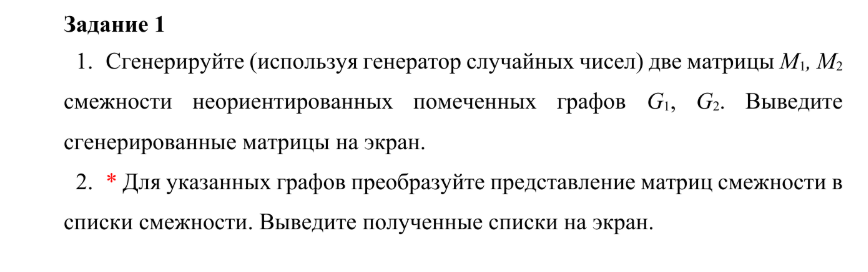
к.т.н., доцент Юрова О.В.

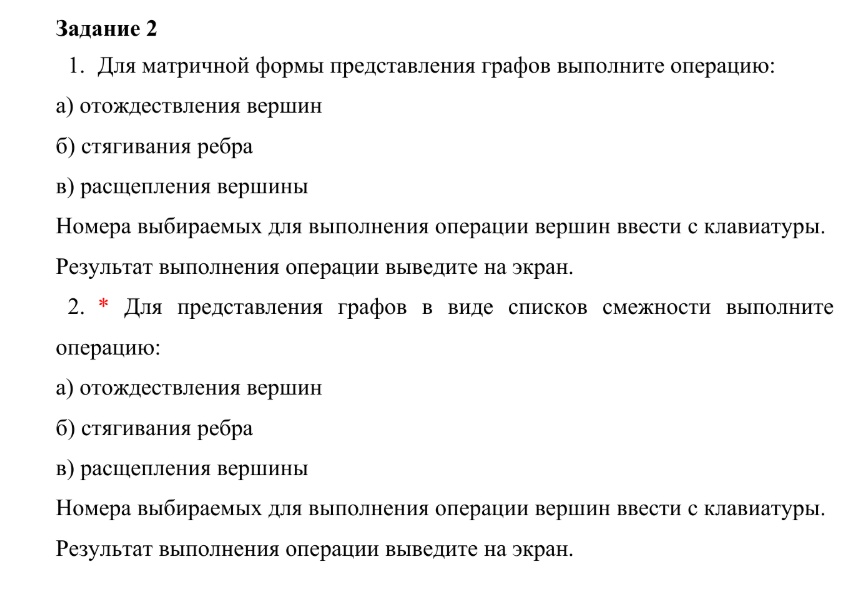
к.т.н., Деев М.В.

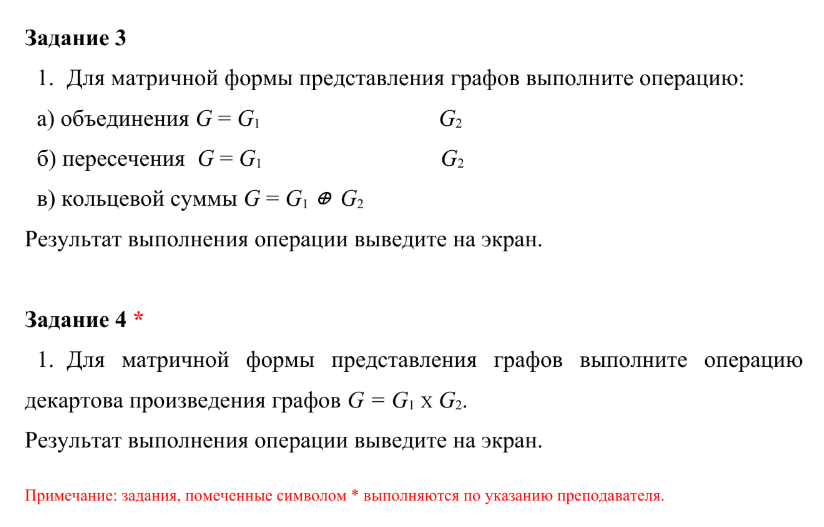
Пенза 2025

**Цель работы** – Изучение и практическое освоение унарных и бинарных операций над графами, включая операции, уменьшающие или увеличивающие число элементов графа, такие как удаление и добавление вершин и рёбер, отождествление и расщепление вершин, стягивание рёбер.

**Лабораторное задание:**







**Задание 1.**

**Код программы**

**C#**

using System;

public class proekt

{

public static int[,] CreateMatr()

{

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

int size = Convert.ToInt32(Console.ReadLine());

int[,] G = new int[size, size];

Random rnd = new Random();

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

{

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

{

if (i == j)

{

G[i, j] = 0;

}

else

{

int value = rnd.Next(0, 2);

G[i, j] = value;

G[j, i] = value;

}

}

}

PrintMatrx(G);

return G;

}

public static void PrintMatrx(int[,] matr)

{

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

{

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

{

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

}

Console.WriteLine();

}

}

public static int[,] First(int[,] matr, int first, int second)

{

Console.WriteLine("Отождествление вершин " + first + " и " + second + ":");

int size = matr.GetLength(0);

if (first < 0 || first >= size || second < 0 || second >= size || first == second)

{

Console.WriteLine("Ошибка: неверные номера вершин!");

return matr;

}

int newSize = size - 1;

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

int mergedVertex = Math.Min(first, second);

int removedVertex = Math.Max(first, second);

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

{

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

{

if (i == removedVertex || j == removedVertex)

continue;

int newI = i;

int newJ = j;

if (i > removedVertex) newI = i - 1;

if (j > removedVertex) newJ = j - 1;

if (i == mergedVertex || j == mergedVertex)

{

int value1 = matr[i, j];

int value2;

if (i == mergedVertex)

{

value2 = matr[removedVertex, j];

}

else

{

value2 = matr[i, removedVertex];

}

if (value1 == 1 || value2 == 1)

{

resMatr[newI, newJ] = 1;

}

else

{

resMatr[newI, newJ] = 0;

}

}

else

{

resMatr[newI, newJ] = matr[i, j];

}

}

}

PrintMatrx(resMatr);

return resMatr;

}

public static int[,] Second(int[,] matr, int first, int second)

{

Console.WriteLine("Стягивание вершин " + first + " и " + second + ":");

if (matr[first, second] == 0)

{

Console.WriteLine("Ошибка: между вершинами " + first + " и " + second + " нет ребра!");

return matr;

}

int[,] resMatr = new int[matr.GetLength(0) - 1, matr.GetLength(0) - 1];

int remains = Math.Min(first, second);

int goOut = Math.Max(first, second);

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

{

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

{

if (i == goOut || j == goOut)

continue;

int newI = i;

int newJ = j;

if (i > goOut) newI = i - 1;

if (j > goOut) newJ = j - 1;

if (i == remains || j == remains)

{

int value1 = matr[i, j];

int value2;

if (i == remains)

{

value2 = matr[goOut, j];

}

else

{

value2 = matr[i, goOut];

}

if ((i == remains && j == remains) ||

(i == remains && j == goOut) ||

(i == goOut && j == remains))

{

resMatr[newI, newJ] = 0;

}

else

{

resMatr[newI, newJ] = (value1 == 1 || value2 == 1) ? 1 : 0;

}

}

else

{

resMatr[newI, newJ] = matr[i, j];

}

}

}

PrintMatrx(resMatr);

return resMatr;

}

public static int[,] Third(int[,] matr, int first)

{

int newSize = matr.GetLength(0) + 1;

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

int newVertex = newSize - 1;

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

{

for (int j = 0; j < matr.GetLength(0); j++)

{

resMatr[i, j] = matr[i, j];

}

}

Random rnd = new Random();

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

for (int j = 0; j < matr.GetLength(0); j++)

{

if (matr[first, j] == 1 && j != first)

{

adjacentVertices.Add(j);

}

}

int edgesToMove = rnd.Next(1, adjacentVertices.Count);

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

while (movedEdges.Count < edgesToMove && adjacentVertices.Count > 0)

{

int randomIndex = rnd.Next(0, adjacentVertices.Count);

int vertex = adjacentVertices[randomIndex];

movedEdges.Add(vertex);

adjacentVertices.RemoveAt(randomIndex);

}

foreach (int vertex in movedEdges)

{

resMatr[first, vertex] = 0;

resMatr[vertex, first] = 0;

resMatr[newVertex, vertex] = 1;

resMatr[vertex, newVertex] = 1;

}

resMatr[first, newVertex] = 1;

resMatr[newVertex, first] = 1;

PrintMatrx(resMatr);

return resMatr;

}

public static void Main()

{

Console.WriteLine("Создание первого графа:");

int[,] G1 = CreateMatr();

Console.WriteLine("\nВведите номера вершин для отождествления (через Enter):");

int first = Convert.ToInt32(Console.ReadLine());

int second = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("\nРезультат отождествления:");

int[,] G3 = First(G1, first, second);

Console.WriteLine("Введите числа для стягивания: ");

first = Convert.ToInt32(Console.ReadLine());

second = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Результат стягивания:");

G3 = Second(G1, first, second);

Console.WriteLine("Расщепление вершины: ");

Console.WriteLine("Введите число для рассщепления: ");

first = Convert.ToInt32(Console.ReadLine());

G3 = Third(G1, first);

}

}

**Задание 2.**

**Код программы**

using System.Collections;

using System.Collections.Generic;

using System.Drawing;

using System.Net.Http.Headers;

public class proekt

{

public static int[,] CreateMatr()

{

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

int size = Convert.ToInt32(Console.ReadLine());

int[,] G = new int[size, size];

Random rnd = new Random();

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

{

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

{

if (i == j)

{

G[i, j] = 0;

}

else

{

int value = rnd.Next(0, 2);

G[i, j] = value;

G[j, i] = value;

}

}

}

PrintMatr(G);

return G;

}

public static void PrintMatr(int[,] matr)

{

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

{

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

{

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

}

Console.WriteLine();

}

}

public static void PrintSpisok(List<List<int>> spisok)

{

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

{

string str = "[";

List<int> nums = spisok[i];

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

{

if (j != nums.Count - 1) str += nums[j] + ",";

else str += nums[j];

}

str += "]";

Console.WriteLine($"Вершина {i}: {str}");

}

}

public static List<List<int>> Identification(List<List<int>> spisok, int first, int second)

{

int remains = Math.Min(first, second);

int deleted = Math.Max(first, second);

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

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

{

List<int> list = spisok[i];

if (i == first)

{

List<int> list1 = spisok[remains];

List<int> list2 = spisok[deleted];

list1.Remove(deleted);

list2.Remove(remains);

list1.AddRange(list2);

for (int j = 0; j < list1.Count; j++)

{

if (list1[j] > deleted) list1[j] -= 1;

}

list1 = new List<int>(new HashSet<int>(list1));

result.Add(list1);

}

else if (i == deleted) continue;

else

{

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

{

if (list[j] == deleted) list[j] = remains;

if (list[j] > deleted) list[j] -= 1;

}

list = new List<int>(new HashSet<int>(list));

result.Add(list);

}

}

PrintSpisok(result);

return result;

}

public static List<List<int>> Spisok(int[,] matr)

{

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

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

{

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

for (int j = 0; j < matr.GetLength(0); j++)

{

if (matr[i, j] == 1) nums.Add(j);

}

alllist.Add(nums);

}

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

{

string str = "[";

List<int> nums = alllist[i];

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

{

if (j != nums.Count - 1) str += nums[j] + ",";

else str += nums[j];

}

str += "]";

Console.WriteLine($"Вершина {i}: {str}");

}

return alllist;

}

public static List<List<int>> CheckConect(List<List<int>> spisok, int first, int second)

{

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

if (spisok[first].Contains(second) == false || spisok[second].Contains(first) == false)

{

Console.WriteLine("У вершин нет общего ребра!");

return result;

}

int remains = Math.Min(first, second);

int deleted = Math.Max(first, second);

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

{

List<int> list = spisok[i];

if (i == remains)

{

list.Remove(deleted);

List<int> addList = spisok[deleted];

addList.Remove(remains);

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

{

if (addList[j] > deleted) addList[j] -= 1;

}

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

{

if (list[j] > deleted) list[j] -= 1;

}

list.AddRange(addList);

list = new List<int>(new HashSet<int>(list));

result.Add(list);

}

else if (i == deleted) continue;

else

{

if (list.Contains(deleted))

{

int index = list.IndexOf(deleted);

list[index] = remains;

}

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

{

if (list[j] > deleted) list[j] -= 1;

}

result.Add(list);

}

}

PrintSpisok(result);

return result;

}

public static List<List<int>> Splitting(List<List<int>> spisok, int input)

{

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

if (spisok[input].Count < 2)

{

Console.WriteLine("Нельзя расщепить!");

return result;

}

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

{

if (i == input)

{

Random rnd = new Random();

int splitIndex = rnd.Next(1, spisok[input].Count);

List<int> list1 = spisok[input].Take(splitIndex).ToList();

List<int> list2 = spisok[input].Skip(splitIndex).ToList();

list2.Add(input);

//list2.Add(100);

result.Add(list1);

result.Add(list2);

}

else

{

result.Add(spisok[i]);

}

}

PrintSpisok(result);

return result;

}

public static void Main()

{

int first, second;

List<List<int>> G3;

// для теста матрица с которой можно все делать

//int[,] matrTest = {

// { 0, 1, 0, 1, 0, 1},

// { 1, 0, 1, 0, 1, 0},

// { 0, 1, 0, 1, 0, 1},

// { 1, 0, 1, 0, 1, 0},

// { 0, 1, 0, 1, 0, 1},

// { 1, 0, 1, 0, 1, 0}

//};

//PrintMatr(matrTest);

//List<List<int>> matr1Spisok = Spisok(matrTest);

int[,] matr1 = CreateMatr();

List<List<int>> matr1Spisok = Spisok(matr1);

Console.WriteLine("Введите объединяемые вершины: ");

first = Convert.ToInt32(Console.ReadLine());

second = Convert.ToInt32(Console.ReadLine());

G3 = Identification(matr1Spisok, first, second);

Console.WriteLine("\nВведите вершины для стягивания:");

first = Convert.ToInt32(Console.ReadLine());

second = Convert.ToInt32(Console.ReadLine());

G3 = CheckConect(matr1Spisok, first, second);

Console.WriteLine("\nВведите вершину для расщепления:");

first = Convert.ToInt32(Console.ReadLine());

G3 = Splitting(matr1Spisok, first);

}

}

**Задание 3.**

**Код программы**

using System.Drawing;

class Laba6

{

public static int[,] CreateMatr(int size)

{

int[,] G = new int[size, size];

Random rnd = new Random();

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

{

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

{

if (i == j)

{

G[i, j] = 0;

}

else

{

int value = rnd.Next(0, 2);

G[i, j] = value;

G[j, i] = value;

}

}

}

Print(G);

return G;

}

public static void Print(int[,] matr)

{

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

{

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

{

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

}

Console.WriteLine();

}

}

public static int[,] Matr1Matr2A(int[,] G1, int[,] G2)

{

Console.WriteLine("Объединение:");

int[,] resultMatrix = new int[G1.GetLength(0), G1.GetLength(0)];

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

{

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

{

if (G1[i, j] == 1 || G2[i, j] == 1)

{

resultMatrix[i, j] = 1;

}

else resultMatrix[i, j] = 0;

}

}

Print(resultMatrix);

return resultMatrix;

}

public static int[,] Matr1Matr2B(int[,] G1, int[,] G2)

{

Console.WriteLine("Пересечение:");

int[,] resultMatrix = new int[G1.GetLength(0), G1.GetLength(0)];

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

{

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

{

if (G1[i, j] == 1 && G2[i, j] == 1)

{

resultMatrix[i, j] = 1;

}

else resultMatrix[i, j] = 0;

}

}

Print(resultMatrix);

return resultMatrix;

}

public static int[,] Matr1Matr2C(int[,] G1, int[,] G2)

{

Console.WriteLine("Кольцевой суммы:");

int[,] resultMatrix = new int[G1.GetLength(0), G1.GetLength(0)];

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

{

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

{

if (G1[i, j] != G2[i, j]) resultMatrix[i, j] = 1;

else resultMatrix[i, j] = 0;

}

}

Print(resultMatrix);

return resultMatrix;

}

public static void Main()

{

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

int size = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Матрица 1:");

int[,] G1 = CreateMatr(size);

Console.WriteLine("\nМатрица 2: ");

int[,] G2 = CreateMatr(size);

Console.WriteLine("Что получилось: ");

int[,] G3 = Matr1Matr2A(G1, G2);

G3 = Matr1Matr2B(G1, G2);

G3 = Matr1Matr2C(G1, G2);

}

}

**Результаты работы программ**

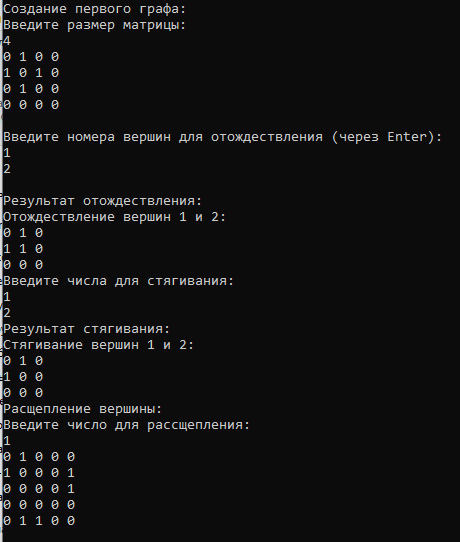


Рисунок 1 - Результат работы программы 1

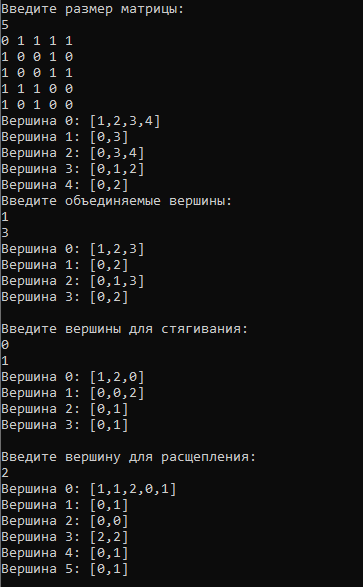


Рисунок 2 - Результат работы программы 2

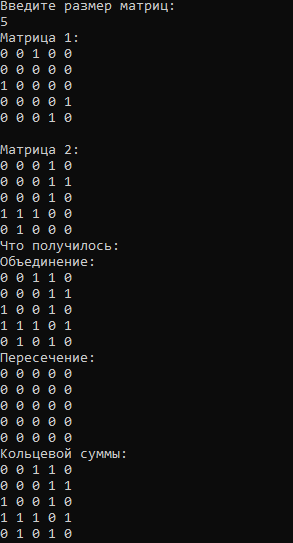


Рисунок 3 - Результат работы программы 3

**Вывод:** В ходе выполнения лабораторной работы была разработана программа на языке C#, реализующая основные унарные и бинарные операции над графами.