Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №5**

Дисциплина: «Дискретная математика и математическая

логика»

Тема: Компоненты связности графа

Выполнил работу

студент группы ИВТ-22-2б

Казанцев А.В.

Проверила

Старший преподаватель кафедры ИТАС

Рустамханова Г.И.

Пермь 2024

**Цель работы**

1. Создание консольного приложения «Определение компонент связности»

2. Изучение теории дискретной математики об свойствах графов

3. Изучение и отработка навыков определения компонент связности, работы с матрицами достижимости и матрицей смежности.

**Постановка задачи**

1. Программа должен иметь возможность задавать систему функций через матрицу смежности, полученную из файла.
2. Программа должна выполнять следующие функции:
   1. Создание и вывод на экран матрицы достижимости графа.
   2. Показать количество компонент связности в графе.
   3. Вывод на экран вершины каждой компоненты связности.

**Ход работы**

Программа имеет возможность открыть файл с подходящий имеем.  
Программа считывает файл в класс MatrixGraf. Строки нормализируются для корректной работы. Файл закрывается.

Программа позволяет выводить Матрицы смежности и достижимости.   
Позволяет узнать количество компонент связности графа.  
Позволяет вывести на экран алгоритм вычисления компонент связности и вершины этих компонент.

Методы, используемые в классе:

ReadMatrix (string path)– метод открывает и считывает файл в статический двумерный массив.

WriteMatrix (int[,] matrix) выводит на квадратную матрицу в консоль.

int[,] SumMatrix(int[,] matrix1, int[,] matrix2) – возвращает двумерный массив суммирующий две матрицы по особому правилу «1+1=1».

int[,] DegreeMatrix(int[,] matrix0) – возвращает двумерный массив, умноженный на статическую матрицу класса.

bool EqualsMatrix(int[,] matrix1, int[,] matrix2) возвращает истину если матрицы равны.

int[,] CopyMatrix(int[,] matrix) ) – возвращает копию двумерного массива.

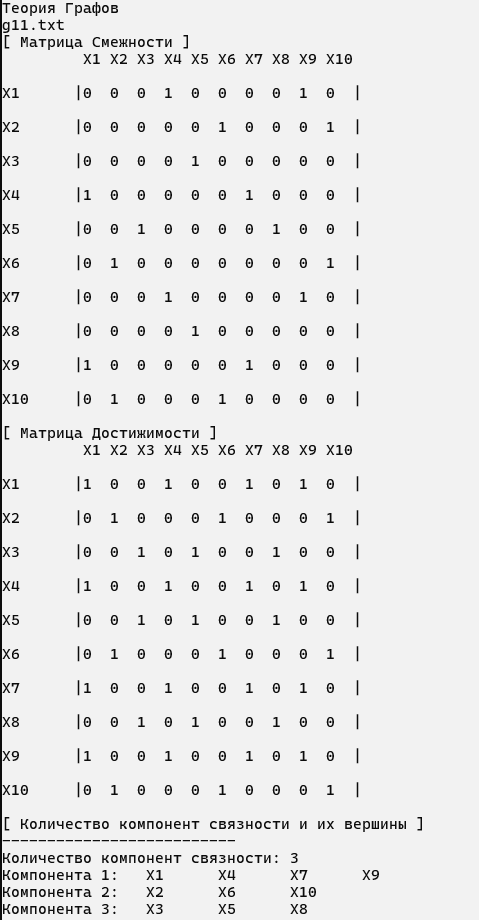
int[,] CreateReachabilityMatrix()) – возвращает матрицу достижимости от статической матрицы класса.

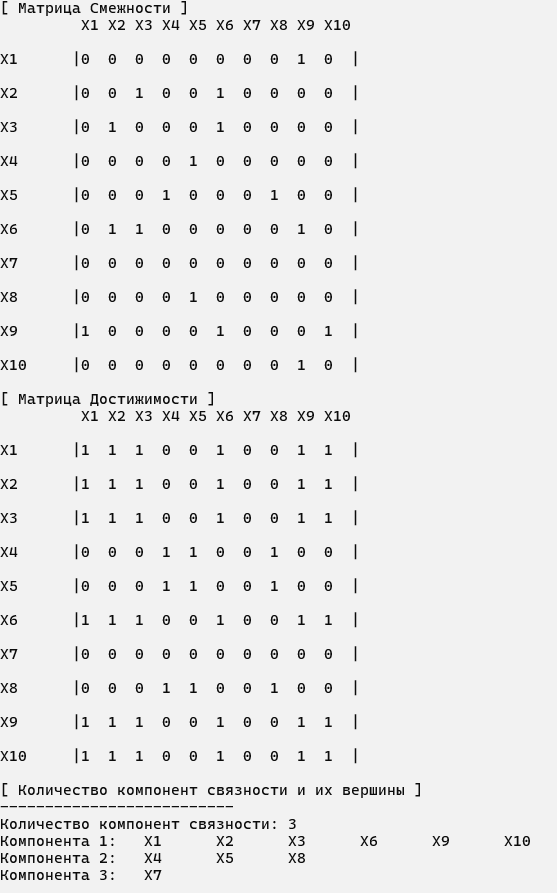
Start()- запускает демонстрационную программу.

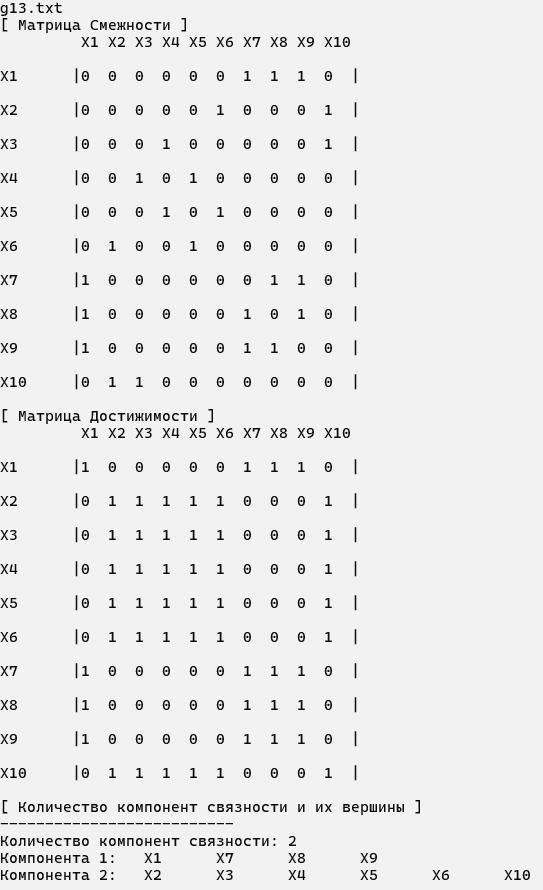
string[] IntoStr(int[,] matrix0) – возвращает массив переводит строки целочисленной матрицы в строковый массив.

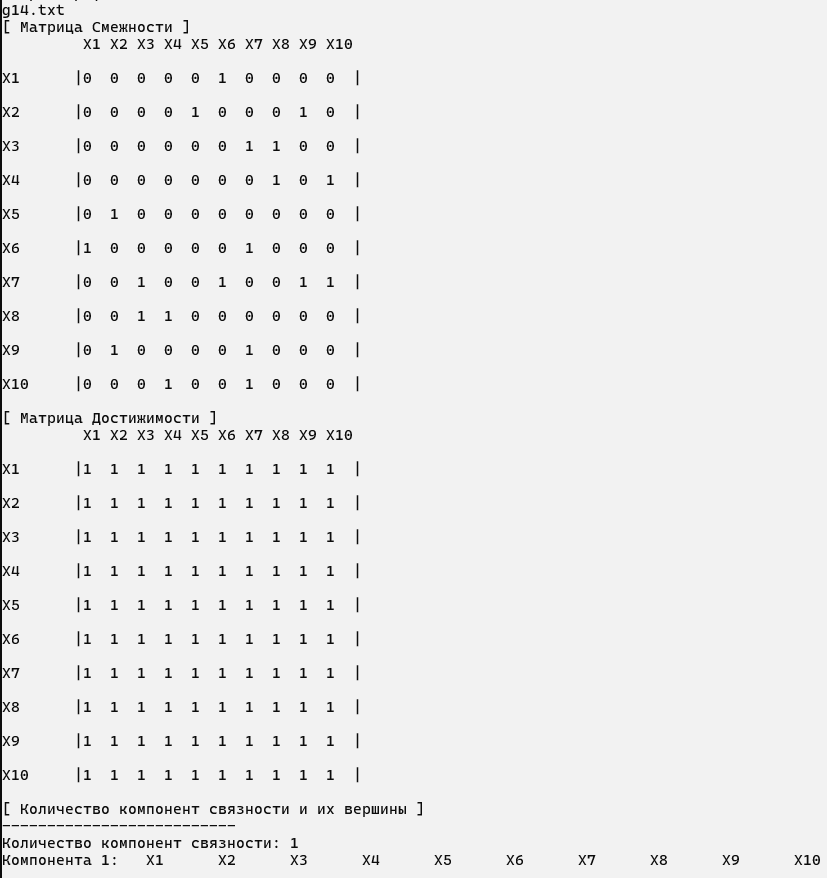
void FindComp(int[,] matrix)- вычисление компонент связности по матрице достижимости и вывод ответа на экран.

Тесты





****

****

**Код программы**

namespace AppGraphTheory

{

public class MatrixGraf

{

static int[,] matrix;

public int[,] Matrix { get { return matrix; } }

public MatrixGraf() { }

public MatrixGraf(string path)

{

string[] lines = File.ReadAllLines(path);

matrix = new int[lines.GetLength(0), (lines[0].Length + 1) / 2];

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

{

string[] buf = lines[i].Split(' ');

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

{

matrix[i, j] = int.Parse(buf[j]);

}

}

}

public static int[,] CopyMatrix(int[,] matrix)

{

int[,] res = new int[matrix.GetLength(0), matrix.GetLength(1)];

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

{

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

{

res[i, j] = matrix[i, j];

}

}

return res;

}

public static int[,] CreateReachabilityMatrix()

{

int[,] buf = new int[matrix.GetLength(0), matrix.GetLength(1)];

int[,] buf1 = new int[matrix.GetLength(0), matrix.GetLength(1)];

int[,] attainMat = new int[matrix.GetLength(0), matrix.GetLength(1)];

attainMat = SumMatrix(attainMat, matrix);

buf1 = CopyMatrix(matrix);

buf = CopyMatrix(attainMat);

int k = 0;

while (true)

{

buf1 = DegreeMatrix(buf1);

attainMat = SumMatrix(attainMat, buf1);

if (EqualsMatrix(buf, attainMat))

k++;

else

k = 0;

if (k == 2)

break;

buf = CopyMatrix(attainMat);

}

return attainMat;

}

public static int[,] DegreeMatrix(int[,] matrix0)

{

int[,] res = new int[matrix.GetLength(0), matrix.GetLength(1)];

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

{

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

{

for (int k = 0; k < matrix.GetLength(0); k++)

{

res[i, j] += matrix0[i, k] \* matrix[k, j];

}

if (res[i, j] > 0)

res[i, j] = 1;

}

}

return res;

}

public static bool EqualsMatrix(int[,] matrix1, int[,] matrix2)

{

bool isCheck = true;

isCheck = matrix1.GetLength(0) == matrix2.GetLength(0)

&& matrix1.GetLength(1) == matrix2.GetLength(1);

if (!isCheck)

return isCheck;

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

{

for (int j = 0; j < matrix2.GetLength(1) && isCheck; j++)

{

isCheck = matrix1[i, j] == matrix2[i, j];

}

}

return isCheck;

}

public static void Exit()

{

Console.WriteLine("\n[ Нажмите любую кнопку для продолжения... ]\n[ Для выхода нажмите esc ]");

if (Console.ReadKey(true).Key == ConsoleKey.Escape)

Environment.Exit(0);

}

public static void FindComp(int[,] matrix)

{

string[] matrixStr = IntoStr(matrix);

Dictionary<string, List<string>> pairs = new Dictionary<string, List<string>>();

for (int i = 0; i < matrixStr.Length; i++)

{

if (pairs.Keys.Contains(matrixStr[i]))

continue;

pairs[matrixStr[i]] = new List<string>() { "X" + (i + 1) };

for (int j = i + 1; j < matrixStr[i].Length; j++)

{

if (matrixStr[i] == matrixStr[j])

pairs[matrixStr[i]].Add("X" + (j + 1));

}

}

Console.WriteLine("Количество компонент связности: " + pairs.Keys.Count);

int r = 1;

foreach (var item in pairs)

{

Console.Write("Компонента " + r++ + ": \t");

foreach (var item1 in item.Value)

{

Console.Write(item1 + "\t");

}

Console.WriteLine();

}

}

public int CountComp()

{

int[,] matrix = CreateReachabilityMatrix();

string[] matrixStr = IntoStr(matrix);

Dictionary<string, List<string>> pairs = new Dictionary<string, List<string>>();

for (int i = 0; i < matrixStr.Length; i++)

{

if (pairs.Keys.Contains(matrixStr[i]))

continue;

pairs[matrixStr[i]] = new List<string>() { "X" + (i + 1) };

for (int j = i + 1; j < matrixStr[i].Length; j++)

{

if (matrixStr[i] == matrixStr[j])

pairs[matrixStr[i]].Add("X" + (j + 1));

}

}

return pairs.Keys.Count();

}

public static string[] IntoStr(int[,] matrix0)

{

string[] res = new string[matrix0.GetLength(1)];

for (int i = 0; i < matrix0.GetLength(0); i++)

{

string str = "";

for (int j = 0; j < matrix0.GetLength(1); j++)

{

str += matrix0[j, i];

}

res[i] = str;

}

return res;

}

public static void ReadMatrix(string path)

{

string[] lines = File.ReadAllLines(path);

matrix = new int[lines.GetLength(0), (lines[0].Length + 1) / 2];

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

{

string[] buf = lines[i].Split(' ');

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

{

matrix[i, j] = int.Parse(buf[j]);

}

}

}

public static void Run(string path)

{

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.Clear();

Console.WriteLine("Теория Графов");

ReadMatrix(path);

Console.WriteLine(path);

Console.WriteLine("[ Матрица Смежности ]");

WriteMatrix(matrix);

matrix = CreateReachabilityMatrix();

Console.WriteLine("[ Матрица Достижимости ]");

WriteMatrix(matrix);

Console.WriteLine("[ Количество компонент связности и их вершины ]");

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

FindComp(matrix);

}

public static int[,] SumMatrix(int[,] matrix1, int[,] matrix2)

{

int[,] res = new int[matrix.GetLength(0), matrix.GetLength(1)];

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

{

for (int j = 0; j < matrix2.GetLength(1); j++)

{

res[i, j] = matrix2[i, j] + matrix1[i, j];

if (res[i, j] > 0)

res[i, j] = 1;

}

}

return res;

}

public static void WriteMatrix(int[,] matrix)

{

if (matrix == null) { Console.WriteLine("Матрица пуста. "); return; }

Console.Write("\t ");

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

{

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

}

Console.WriteLine("\n");

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

{

Console.Write("X" + (i + 1) + "\t|");

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

{

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

}

Console.WriteLine("|\v");

}

}

public void Start()

{

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

Console.Clear();

Console.WriteLine("Теория Графов");

string path;

for (int i = 1; i < 5; i++)

{

path = "g1" + i + ".txt";

Run(path);

Exit();

}

Console.WriteLine("\n[ Матрицы закончились] ");

}

}

}