Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Дискретная математика»

Тема: «Связность графов»

Семестр 4

Выполнил работу

Студент группы ИВТ-23-2б

Меновщиков Глеб Николаевич

Проверил

старший преподаватель кафедры ИТАС

Рустамханова Гульшат Ильдаровна

Пермь 2025

**Постановка задачи**

Разработать программу, которая по заданной матрице смежности ориентированного графа:

* Строит матрицу достижимости.
* Проверяет граф на сильную связность.
* Находит компоненты сильной связности.

**Входные данные**

Граф задаётся матрицей смежности размером 10×10, хранящейся в текстовом файле. В файле числа записаны построчно через пробел (0 означает отсутствие ребра).

**Выходные данные**

Матрица смежности (исходная).

Матрица достижимости (показывает, какие вершины достижимы из каждой).

Матрица сильной связности (показывает, между какими вершинами есть двусторонние пути).

Поиск компонент сильной связности (вывод вершин, входящих в каждую компоненту).

**Код программы**

using System;

using System.Collections.Generic;

using System.IO;

namespace ReachabilityMatrix

{

class Program

{

const int N = 10; // размер матрицы

static void Main(string[] args)

{

int[,] matrix = new int[N, N];

Console.WriteLine("Выберите способ ввода матрицы:");

Console.WriteLine("1 - Считать из файла");

Console.WriteLine("2 - Ручной ввод");

string choice = Console.ReadLine();

if (choice == "1")

{

try

{

Console.WriteLine("Введите название файла: ");

string matrCh = Console.ReadLine();

matrix = ReadMatrixFromFile(matrCh);

}

catch (Exception ex)

{

Console.WriteLine("Ошибка при чтении файла: " + ex.Message);

return;

}

}

else

{

matrix = ReadMatrixFromConsole();

}

Console.WriteLine("Исходная матрица смежности:");

PrintMatrix(matrix);

if (IsDirected(matrix))

{

Console.WriteLine("Обнаружен ориентированный граф. Преобразуем в неориентированный...");

matrix = MakeUndirected(matrix);

Console.WriteLine("Матрица после преобразования:");

PrintMatrix(matrix);

}

else

{

Console.WriteLine("Граф не является ориентированным.");

}

// Вычисляем матрицу достижимости – алгоритм Флойда-Уоршелла для булевой логики

int[,] reachability = ComputeReachability(matrix);

Console.WriteLine("Матрица достижимости:");

PrintMatrix(reachability);

// Нахождение компонент связности с помощью DFS

List<List<int>> components = GetConnectedComponents(reachability);

Console.WriteLine($"Количество компонент связности: {components.Count}");

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

{

Console.Write($"Компонента {i + 1}: ");

foreach (int vertex in components[i])

{

Console.Write($"{vertex} ");

}

Console.WriteLine();

}

Console.WriteLine("Нажмите любую клавишу для выхода.");

Console.ReadKey();

}

// Метод для проверки, является ли граф ориентированным.

// Если найдется пара (i, j), для которой mat[i, j] != mat[j, i], то граф ориентированный.

static bool IsDirected(int[,] mat)

{

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

{

for (int j = i + 1; j < N; j++)

{

if (mat[i, j] != mat[j, i])

return true;

}

}

return false;

}

// Чтение матрицы из файла (ожидается, что в файле 10 строк, в каждой 10 чисел, разделённых пробелами)

static int[,] ReadMatrixFromFile(string fileName)

{

int[,] mat = new int[N, N];

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

if (lines.Length < N)

throw new Exception("Недостаточно строк в файле!");

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

{

string[] parts = lines[i].Split(new char[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length < N)

throw new Exception($"Недостаточно элементов в строке {i + 1}!");

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

{

mat[i, j] = int.Parse(parts[j]);

}

}

return mat;

}

// Ручной ввод матрицы

static int[,] ReadMatrixFromConsole()

{

int[,] mat = new int[N, N];

Console.WriteLine($"Введите матрицу {N}x{N} (числа через пробел):");

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

{

Console.WriteLine($"Строка {i + 1}:");

string[] parts = Console.ReadLine().Split(new char[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length < N)

{

Console.WriteLine("Недостаточно элементов, попробуйте снова.");

i--;

continue;

}

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

{

mat[i, j] = int.Parse(parts[j]);

}

}

return mat;

}

// Приведение матрицы к неориентированному виду (симметричная матрица)

static int[,] MakeUndirected(int[,] mat)

{

int[,] undirected = new int[N, N];

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

{

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

{

// Если есть ребро в любом направлении, то ставим 1

undirected[i, j] = (mat[i, j] == 1 || mat[j, i] == 1) ? 1 : 0;

}

}

return undirected;

}

// Вычисление матрицы достижимости через алгоритм Флойда-Уоршелла для булевой матрицы

static int[,] ComputeReachability(int[,] mat)

{

// Начинаем с матрицы смежности, добавляя петли (вершина достижима сама по себе)

int[,] reach = new int[N, N];

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

{

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

{

reach[i, j] = (i == j || mat[i, j] == 1) ? 1 : 0;

}

}

// Алгоритм Флойда-Уоршелла: если через k можно добраться из i в j, то отмечаем 1

for (int k = 0; k < N; k++)

{

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

{

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

{

reach[i, j] = (reach[i, j] == 1 || (reach[i, k] == 1 && reach[k, j] == 1)) ? 1 : 0;

}

}

}

return reach;

}

// Вывод матрицы в консоль

static void PrintMatrix(int[,] mat)

{

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

{

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

{

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

}

Console.WriteLine();

}

}

// Нахождение компонент связности с помощью DFS по матрице достижимости

static List<List<int>> GetConnectedComponents(int[,] reach)

{

bool[] visited = new bool[N];

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

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

{

if (!visited[i])

{

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

DFS(i, reach, visited, comp);

components.Add(comp);

}

}

return components;

}

static void DFS(int current, int[,] reach, bool[] visited, List<int> comp)

{

visited[current] = true;

comp.Add(current);

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

{

if (reach[current, j] == 1 && !visited[j])

{

DFS(j, reach, visited, comp);

}

}

}

}

}

**Результаты работы программы**

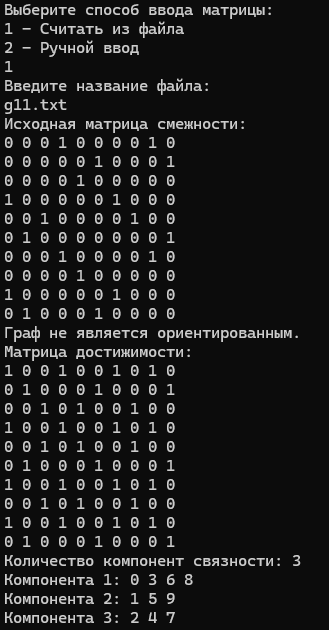
****

Рисунок 1 - результаты работы программы, файл g11

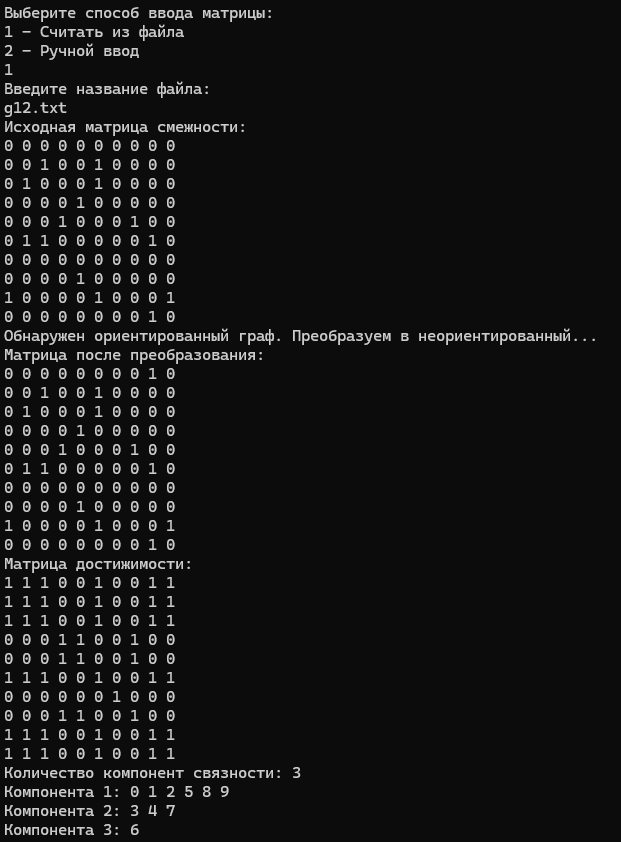


Рисунок 2 - результаты работы программы, файл g12