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 №2**

Дисциплина: «Дискретная математика и математическая логика»

Тема: Калькулятор отношений в матрице

Выполнил работу

студент группы ИВТ-22-2б

Мельников Г. В.

Проверила

Доцент кафедры ИТАС

Рустамханова Г. И.

Пермь, 2023

**Постановка задачи**

Разработать программу «Калькулятор отношений в матрице», которая будет считывать матрицу 6 на 6, содержащую 0 и 1, из файла. Между элементами стоит пробел. Проверить какими базовыми свойствами обладает матрица (рефлексивность, симметричность, транзитивность, связность).

**Методы решения задачи**

В главном методе Main последовательно считываются матрицы с помощью методаReadMatrix и выводятся на экран посредством метода WriteInfo.

ReadMatrix читает матрицу из файла.

WriteMatrix выводит матрицу на экран.

CalculateReflexivity проверяет матрицу на рефлективность, посредством подсчётам 0 и 1 на главной диагонали, затраченное время O(n).

CalculateSymmetry проверяет матрицу на симметричность. Используя полный проход по матрице, затраченное время O(n^2).

CalculateTransitivity проверяет матрицу на транзитивность. Время работы алгоритма оптимизированно с помощью флага isTransitivy и списка suitable. В худшем случае, алгоритм выполняется за O(n^3).

CalculateСonnectivity проверяет матрицу на полноту. Время работы алгоритма оптимизировано с помощью флага isConnectivity. В худшем случае, алгоритм выполняется за O(n^2).

WriteInfo выводит матрицу с помощью метода WriteMatrix и свойства матрицы, полученные из результатов других методов (кроме главного), поэтому для удобства методы возвращают строки, содержащие ответы.

**Результаты**

Матрица

1 0 1 0 0 1

0 1 0 0 0 0

0 0 0 0 0 1

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

Не рефлексивна и

не антирефлексивна

Антисимметрична

Транзитивна

Неполная

Матрица

1 0 1 0 0 1

0 1 0 0 0 0

1 0 1 0 0 1

0 0 0 1 0 0

0 0 0 0 1 0

1 0 1 0 0 1

Рефлексивна

Симметрична

Транзитивна

Неполная

Матрица

0 0 1 0 0 1

1 0 0 0 0 0

0 1 0 0 0 1

1 1 1 0 0 0

1 1 1 1 0 0

0 1 0 1 1 0

Антирефлексивна

Асимметрична

Не транзитивна

Полная

Матрица

0 1 1 1 1 0

0 1 1 1 1 0

0 1 1 1 1 0

0 1 1 1 1 0

0 1 1 1 1 0

0 1 1 1 1 0

Не рефлексивна и

не антирефлексивна

Не подходит под условие симметричности

Транзитивна

Неполная

Матрица

0 0 0 0 0 0

0 0 0 0 0 0

1 1 1 1 1 1

1 1 1 1 1 1

0 0 0 0 0 0

0 0 0 0 0 0

Не рефлексивна и

не антирефлексивна

Не подходит под условие симметричности

Транзитивна

Неполная

Матрица

1 1 1 1 1 1

0 0 1 1 1 1

0 0 0 1 1 1

0 0 0 0 1 1

0 0 0 0 0 1

0 0 0 0 0 1

Не рефлексивна и

не антирефлексивна

Антисимметрична

Транзитивна

Полная

Матрица

1 0 1 1 1 0

1 1 1 1 1 1

0 0 1 1 1 1

0 0 0 1 1 1

0 0 0 0 1 1

1 0 0 0 0 1

Рефлексивна

Антисимметрична

Не транзитивна

Полная

Матрица

0 1 1 1 1 1

1 0 1 1 1 1

1 1 0 1 1 1

1 1 1 0 1 1

1 1 1 1 0 1

1 1 1 1 1 0

Антирефлексивна

Симметрична

Не транзитивна

Полная

**Код программы**

namespace MatrixInfo

{

internal class Program

{

public static void Main()

{

int[,] matrix = new int[6, 6];

string rootProject = Directory.GetParent(Directory.GetCurrentDirectory())

.Parent.Parent.FullName;

string filePath;//Путь к файлу с матрицей

for (int i = 1; i <= 8; i++)

{

if (i != 1)

{

Console.WriteLine('\n');

}

filePath = rootProject + $@"\Tests\m{i}.txt";

matrix = ReadMatrix(filePath);

WriteInfo(matrix);

}

}

public static int[,] ReadMatrix(string filePath)

{

if (!File.Exists(filePath))

{

throw new FileNotFoundException("Файл не найден!");

}// Проверка наличия файла

string[] lines = File.ReadAllLines(filePath);//Чтение строк из файла

const int rowsCount = 6;//количество строк

const int columnsCount = 6;//Количество столбцов

int[,] matrix = new int[rowsCount, columnsCount];//Инициализация матрицы

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

{

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

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

{

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

}

}//Заполнение матрицы данными из файла

return matrix;

}//Прочесть матрицу из файла

public static void WriteMatrix(int[,] matrix)

{

int rowCount = matrix.GetLength(0);

int colCount = matrix.GetLength(1);

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

{

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

{

if (j != 0)

{

Console.Write(' ');

}

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

}

Console.WriteLine();

}

}//Вывести матрицу на экран

public static string CalculateReflexivity(int[,] matrix)

{

int[] countOfZeroAndOne = new int[2];

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

(countOfZeroAndOne[0] == 0 || countOfZeroAndOne[1] == 0);

countOfZeroAndOne[matrix[i, i]]++, i++);

var result = "Рефлексивна";

if (countOfZeroAndOne[0] == 6)

{

result = "Антирефлексивна";

}

else if (countOfZeroAndOne[1] < 6)

{

result = "Не рефлексивна и\n" +

"не антирефлексивна";

}

return result;

}//Вычислить рефлексивность

public static string CalculateSymmetry(int[,] matrix)

{

var result = "Симметрична";

int zeroCount = 0;

int symmetry = 0;

int asymmetry = 0;

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

{

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

{

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

{

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

{

symmetry++;

}

else

{

asymmetry++;

}

}

else if (i == j && matrix[i, j] == 0)

{

zeroCount++;

}

}

}

if (asymmetry != 0)

{

if (symmetry != 0)

{

result = "Не подходит под условие симметричности";

}

else if (zeroCount == 6)

{

result = "Асимметрична";

}

else

{

result = "Антисимметрична";

}

}

return result;

}//Вычислить симметричность

public static string CalculateTransitivity(int[,] matrix)

{

var isTransitivy = true;

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

{

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

{

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

{

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

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

{

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

{

suitable.Add(k);

}

}

for (int k = 0; k < suitable.Count && isTransitivy; k++)

{

isTransitivy = matrix[i, suitable[k]] == matrix[i, j];

}

}

}

}

var result = "Транзитивна";

if (!isTransitivy)

{

result = "Не транзитивна";

}

return result;

}//Вычислить транзитивность

public static string CalculateСonnectivity(int[,] matrix)

{

var isConnectivity = true;

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

{

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

{

if (i != j)

{

isConnectivity = matrix[i, j] == 1 || matrix[j, i] == 1;

}

}

}

var result = "Полная";

if (!isConnectivity)

{

result = "Неполная";

}

return result;

}//Вычислить связность

public static void WriteInfo(int[,] matrix)

{

Console.WriteLine("Матрица");

WriteMatrix(matrix);

Console.WriteLine();

Console.WriteLine(CalculateReflexivity(matrix) + '\n' +

CalculateSymmetry(matrix) + '\n' +

CalculateTransitivity(matrix) + '\n' +

CalculateСonnectivity(matrix));

}//Вывести информацию о матрице на экран

}

}