**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное   
образовательное учреждение высшего образования  
«**ТЮМЕНСКИЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ**»

**ИНСТИТУТ ГЕОЛОГИИ И НЕФТЕГАЗОДОБЫЧИ**

Кафедра кибернетических систем

ОТЧЕТ

по лабораторной работе № **3** по дисциплине «**Математические основы программирования**»   
на тему «**Представление множеств в ЭВМ**»

РУКОВОДИТЕЛЬ:   
должность, ученая степень

Николаева Дарья Романовна

РАЗРАБОТЧИК:   
студент группы АСОиУб-21-1  
Зайцев Владимир Иванович

Буломбаев Алимжан Бийсембеевич

Быльцев Сергей Алексеевич

Ефименко Евгений Валерьевич

Тюмень, 2022

Оглавление

[**1.Постановка задачи** 3](#_Toc104722050)

[**2. Описание алгоритма решения задачи** 4](#_Toc104722051)

[**3. Ручной просчёт** 5](#_Toc104722052)

[**4. Описание программы** 7](#_Toc104722053)

[**5.Тестирование программы** 8](#_Toc104722054)

[**6. Текст программы**. 11](#_Toc104722055)

# **1.Постановка задачи**

Написать программу, реализующую нахождение компонентов сильной связности. орграфа

# **2. Описание алгоритма решения задачи**

1. Узнаем количество вершин графа.
2. Заполняем матрицу смежности орграфа.
3. Находим прямое транзитивное замыкание.
4. Находим обратное транзитивное замыкание.
5. Находим пересечения прямого и обратного транзитивных замыканий.
6. Создаем матрицу компонентов сильной связности
7. Удаляем вершины из орграфа
8. Повторяем шаги 3-7 до тех пор, пока в орграфе остаются вершины.

# **3. Ручной просчёт**

Допустим орграф имеет следующую матрице смежности рёбер:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
| V1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| V2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| V3 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| V4 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| V5 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| V6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| V7 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

P = 1

Прямое транзитивное замыкание равно:

Обратное транзитивное замыкание равно:

Пересечение транзитивных замыканий равно:

Матрица сильной связности равна:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | V1 | V3 | V5 | V7 |
| V1 | 0 | 1 | 1 | 0 |
| V3 | 1 | 0 | 1 | 0 |
| V5 | 0 | 1 | 0 | 1 |
| V7 | 1 | 0 | 1 | 1 |

P = 2

Прямое транзитивное замыкание равно:

Обратное транзитивное замыкание равно:

Пересечение транзитивных замыканий равно:

Матрица сильной связности равна:

|  |  |
| --- | --- |
|  | V2 |
| V2 | 1 |

P = 3

Прямое транзитивное замыкание равно:

Обратное транзитивное замыкание равно:

Пересечение транзитивных замыканий равно:

Матрица сильной связности равна:

|  |  |
| --- | --- |
|  | V4 |
| V4 | 1 |

P = 4

Прямое транзитивное замыкание равно:

Обратное транзитивное замыкание равно:

Пересечение транзитивных замыканий равно:

Матрица сильной связности равна:

|  |  |
| --- | --- |
|  | V6 |
| V6 | 0 |

# **4. Описание программы**

Узнаем количество вершин орграфа и записываем число в переменную

NumberOfVertices.

Создаем двухмерный массив MatrixAdjacencyDigraph размерностью

[NumberOfVertices+1 х NumberOfVertices+1]

Заполняем нулевую строку и нулевой столбец номерами вершин.

Узнаем у пользователя смежность двух вершин и записываем введенное число в созданный массив.

Создаем динамический массив VerticesOfTheInverseTransitiveClosure

который будет содержать вершины обратного транзитивного замыкания.

Создаем динамический массив VerticesOfDirectTransitiveClosure который будет содержать вершины прямого транзитивного замыкания.

Создаем динамический массив IntersectionList

который будет содержать вершины пересечения прямого и обратного транзитивных замыканий.

Создаем цикл

При помощи функции checkingForVertexes проверяем остались ли в нашем орграфе вершины.

В переменную Vertex при помощи функции findVertex записываем вершину орграфа.

При помощи функции directTransitiveClosure находим вершины прямого транзитивного замыкания.

При помощи функции inverseTransitiveClosure находим вершины обратного транзитивного замыкания.

При помощи функции intersectionOfListsOfTransitiveClosures находим пересечение прямого и обратного транзитивного замыканий.

Создаем двухмерный массив theMatrixOfStrongConnectivity в который будем записывать матрицу сильной связности орграфа

При помощи функции theMatrixOfStrongConnectivity заполняем наш двухмерный массив.

При помощи функции DeletingRows удаляем вершины орграфа.

При помощи функции Write выводим матрицу сильной связности на консоль.

При помощи функции ClearArray очищаем динамические массивы.

Повторяем цикл до тех пор пока в орграфе имеются вершины.

# **5.Тестирование программы**

\*Жирным курсивом выделено то что вводит пользователь

Введите количество вершин

***7***

Введите кoмпонент смежности V1V1 ***0***

Введите кoмпонент смежности V1V2 ***0***

Введите кoмпонент смежности V1V3 ***1***

Введите кoмпонент смежности V1V4 ***1***

Введите кoмпонент смежности V1V5 ***1***

Введите кoмпонент смежности V1V6 ***0***

Введите кoмпонент смежности V1V7 ***0***

Введите кoмпонент смежности V2V1 ***0***

Введите кoмпонент смежности V2V2 ***1***

Введите кoмпонент смежности V2V3 ***0***

Введите кoмпонент смежности V2V4 ***0***

Введите кoмпонент смежности V2V5 ***0***

Введите кoмпонент смежности V2V6 ***0***

Введите кoмпонент смежности V2V7 ***0***

Введите кoмпонент смежности V3V1 ***1***

Введите кoмпонент смежности V3V2 ***1***

Введите кoмпонент смежности V3V3 ***0***

Введите кoмпонент смежности V3V4 ***1***

Введите кoмпонент смежности V3V5 ***1***

Введите кoмпонент смежности V3V6 ***0***

Введите кoмпонент смежности V3V7 ***0***

Введите кoмпонент смежности V4V1 ***0***

Введите кoмпонент смежности V4V2 ***1***

Введите кoмпонент смежности V4V3 ***0***

Введите кoмпонент смежности V4V4 ***1***

Введите кoмпонент смежности V4V5 ***0***

Введите кoмпонент смежности V4V6 ***1***

Введите кoмпонент смежности V4V7 ***0***

Введите кoмпонент смежности V5V1 ***0***

Введите кoмпонент смежности V5V2 ***1***

Введите кoмпонент смежности V5V3 ***1***

Введите кoмпонент смежности V5V4 ***1***

Введите кoмпонент смежности V5V5 ***0***

Введите кoмпонент смежности V5V6 ***1***

Введите кoмпонент смежности V5V7 ***1***

Введите кoмпонент смежности V6V1 ***0***

Введите кoмпонент смежности V6V2 ***0***

Введите кoмпонент смежности V6V3 ***0***

Введите кoмпонент смежности V6V4 ***0***

Введите кoмпонент смежности V6V5 ***0***

Введите кoмпонент смежности V6V6 ***0***

Введите кoмпонент смежности V6V7 ***0***

Введите кoмпонент смежности V7V1 ***1***

Введите кoмпонент смежности V7V2 ***1***

Введите кoмпонент смежности V7V3 ***0***

Введите кoмпонент смежности V7V4 ***1***

Введите кoмпонент смежности V7V5 ***1***

Введите кoмпонент смежности V7V6 ***0***

Введите кoмпонент смежности V7V7 ***1***

V1 V3 V5 V7

V1 0 1 1 0

V3 1 0 1 0

V5 0 1 0 1

V7 1 0 1 1

V2

V2 1

V4

V4 1

V6

V6 0

# **6. Текст программы**.

*Приложение 1* – код программы.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LB\_MOP\_3\_NEW

{

    internal class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Введите количество вершин ");

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

            int[,] MatrixAdjacencyDigraph = new int[NumberOfVertices+1, NumberOfVertices+1];

            MatrixAdjacencyDigraph[0, 0] = -1;

            for (int i = 1; i < MatrixAdjacencyDigraph.GetLength(0); i++)

            {

                MatrixAdjacencyDigraph[i, 0] = i;

            }

            for (int i = 1; i < MatrixAdjacencyDigraph.GetLength(1); i++)

            {

                MatrixAdjacencyDigraph[0, i] = i;

            }

            for (int i = 1; i < MatrixAdjacencyDigraph.GetLength(0); i++)

            {

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

                {

                    Console.Write($"Введите кoмпонент смежности V{i}V{j} ");

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

                    MatrixAdjacencyDigraph[i, j] = x;

                }

            }

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

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

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

            while (!checkingForVertexes(MatrixAdjacencyDigraph))

            {

                int Vertex = findVertex(MatrixAdjacencyDigraph);

                directTransitiveClosure(MatrixAdjacencyDigraph,ref VerticesOfDirectTransitiveClosure, Vertex);

                inverseTransitiveClosure(MatrixAdjacencyDigraph,ref VerticesOfTheInverseTransitiveClosure, Vertex);

                intersectionOfListsOfTransitiveClosures(VerticesOfDirectTransitiveClosure, VerticesOfTheInverseTransitiveClosure, ref IntersectionList);

                int[,] theMatrixOfStrongConnectivity = new int[IntersectionList.Count, IntersectionList.Count];

                Program.theMatrixOfStrongConnectivity(MatrixAdjacencyDigraph, IntersectionList, ref theMatrixOfStrongConnectivity);

                DeletingRows(ref MatrixAdjacencyDigraph, IntersectionList);

                Program.Write(theMatrixOfStrongConnectivity, IntersectionList);

                ClearArray(ref VerticesOfDirectTransitiveClosure, ref VerticesOfTheInverseTransitiveClosure, ref IntersectionList);

                Console.WriteLine();

            }

            Console.ReadKey();

        }

        static void directTransitiveClosure (int[,] MatrixAdjacencyDigraph, ref List<int> VerticesOfDirectTransitiveClosure, int Vertex)

        {

            VerticesOfDirectTransitiveClosure.Add(Vertex);

            for (int count = 0; count < VerticesOfDirectTransitiveClosure.Count; count++)

            {

                for (int i = VerticesOfDirectTransitiveClosure[count]; i == VerticesOfDirectTransitiveClosure[count]; i++)

                {

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

                    {

                        if ((MatrixAdjacencyDigraph[i, j] == 1) && (!checkingForElementRepetition(VerticesOfDirectTransitiveClosure, MatrixAdjacencyDigraph[0, j])))

                        {

                                VerticesOfDirectTransitiveClosure.Add(MatrixAdjacencyDigraph[0, j]);

                        }

                    }

                }

            }

        }

        static void inverseTransitiveClosure(int[,] MatrixAdjacencyDigraph, ref List<int> VerticesOfTheInverseTransitiveClosure, int Vertex)

        {

            VerticesOfTheInverseTransitiveClosure.Add(Vertex);

            for (int count = 0; count < VerticesOfTheInverseTransitiveClosure.Count; count++)

            {

                for (int j = VerticesOfTheInverseTransitiveClosure[count]; j == VerticesOfTheInverseTransitiveClosure[count]; j++)

                {

                    for (int i = 1; i < MatrixAdjacencyDigraph.GetLength(0); i++)

                    {

                        if (((MatrixAdjacencyDigraph[i, j] == 1) && (!checkingForElementRepetition(VerticesOfTheInverseTransitiveClosure, MatrixAdjacencyDigraph[i, 0]))))

                        {

                            VerticesOfTheInverseTransitiveClosure.Add(MatrixAdjacencyDigraph[i, 0]);

                        }

                    }

                }

            }

        }

        static void intersectionOfListsOfTransitiveClosures(List<int> VerticesOfDirectTransitiveClosure, List<int> VerticesOfTheInverseTransitiveClosure, ref List<int> IntersectionList)

        {

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

            {

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

                {

                    if (VerticesOfTheInverseTransitiveClosure[j] == VerticesOfDirectTransitiveClosure[i])

                    {

                        IntersectionList.Add(VerticesOfDirectTransitiveClosure[i]);

                    }

                }

            }

        }

        static bool checkingForVertexes(int[,] MatrixAdjacencyDigraph)

        {

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

            {

                if (MatrixAdjacencyDigraph[0, i] != -1)

                {

                    return false;

                }

            }

            return true;

        }

        static void theMatrixOfStrongConnectivity(int[,] MatrixAdjacencyDigraphCopy, List<int> IntersectionList, ref int[,] theMatrixOfStrongConnectivity)

        {

            IntersectionList.Sort();

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

            {

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

                {

                    theMatrixOfStrongConnectivity[i, j] = MatrixAdjacencyDigraphCopy[IntersectionList[i], IntersectionList[j]];

                }

            }

        }

        static int findVertex(int[,] MatrixAdjacencyDigraphCopy)

        {

            int Vertex = 0;

            for (int i = 0; i < 1; i++)

            {

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

                {

                    if ((MatrixAdjacencyDigraphCopy[i, j] != -1) && (MatrixAdjacencyDigraphCopy[i, j] != 0))

                    {

                       return  Vertex = MatrixAdjacencyDigraphCopy[i, j];

                    }

                }

            }

            return 0;

        }

        static void ClearArray(ref List<int> VerticesOfDirectTransitiveClosure,ref List<int> VerticesOfTheInverseTransitiveClosure, ref List<int> IntersectionList)

        {

            VerticesOfDirectTransitiveClosure.Clear();

            VerticesOfTheInverseTransitiveClosure.Clear();

            IntersectionList.Clear();

        }

        static bool checkingForElementRepetition(List<int> VerticesTransitiveClosure, int Vertex)

        {

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

            {

                if (VerticesTransitiveClosure[i] == Vertex)

                {

                    {

                        return true;

                    }

                }

            }

            return false;

        }

        static void DeletingRows(ref int[,] MatrixAdjacencyDigraph, List<int> IntersectionList)

        {

            for (int count = 0; count < IntersectionList.Count; count++)

            {

                for (int i = IntersectionList[count]; i == IntersectionList[count]; i++)

                {

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

                    {

                        MatrixAdjacencyDigraph[i, j] = -1;

                    }

                }

                for (int i = IntersectionList[count]; i == IntersectionList[count]; i++)

                {

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

                    {

                        MatrixAdjacencyDigraph[j, i] = -1;

                    }

                }

            }

        }

        static void Write(int[,] theMatrixOfStrongConnectivity, List<int> IntersectionList)

        {

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

            {

                    Console.Write($"\t V{IntersectionList[i]}");

            }

            Console.WriteLine();

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

            {

                Console.Write("V" + IntersectionList[i] + "\t");

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

                {

                    Console.Write($" {theMatrixOfStrongConnectivity[i, j]} \t");

                }

                Console.WriteLine();

            }

        }

    }

}