**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](file:///C:\Users\я\Desktop\МОП%20ОТЧЕТ%20ЛБ3.docx#_Toc103071248)

[**2. Описание алгоритма решения задачи** 4](file:///C:\Users\я\Desktop\МОП%20ОТЧЕТ%20ЛБ3.docx#_Toc103071249)

[**3. Ручной просчёт** 5](file:///C:\Users\я\Desktop\МОП%20ОТЧЕТ%20ЛБ3.docx#_Toc103071250)

[**4. Описание программы** 7](file:///C:\Users\я\Desktop\МОП%20ОТЧЕТ%20ЛБ3.docx#_Toc103071251)

[**5.Тестирование программы** 9](file:///C:\Users\я\Desktop\МОП%20ОТЧЕТ%20ЛБ3.docx#_Toc103071252)

[**6. Текст программы**. 12](file:///C:\Users\я\Desktop\МОП%20ОТЧЕТ%20ЛБ3.docx#_Toc103071253)

# **1.Постановка задачи**

Написать программу, реализующую нахождение минимального остова графа, используя алгоритм Краскала.

# **2. Описание алгоритма решения задачи**

1) Узнаем и записываем количество вершин графа.

2) Выбираем вид графа (ориентированный или неориентированный).

3) Узнаем и записываем веса ребер графа.

4) Сортируем одновременно массив ребер и массив их весов по не убыванию весов ребер.

5) Создаем двухмерный массив первая строчка которого отвечает за отдельные вершины, а вторая за метку этой вершины.

6) Узнаем вершину из которой выходит ребро.

7) Узнаем вершину в которую входит ребро.

8) Получаем метки этих вершин.

9) Проверим не создает ли ребро цикл.

10) Узнаем минимальную и максимальную метки вершин ребра.

11) Заменяем максимальные метки на минимальные.

12) Добавляем ребро в массив.

13) Подсчитываем сумму остового дерева.

14) Повторяем пункты 6-12 до тех пор, пока не пройдем по всем ребрам.

# **3. Ручной просчёт**

Допустим нагруженный граф имеет следующие ребра:

V1V1; V1V2; V1V3; V1V4; V1V5; V1V6; V1V7; V2V1; V2V2; V2V3; V2V4; V2V5; V2V6; V2V7; V3V1; V3V2; V3V3; V3V4; V3V5; V3V6; V3V7; V4V1; V4V2; V4V3; V4V4; V4V5; V4V6; V4V7; V5V1; V5V2; V5V3; V5V4; V5V5; V6V6; V6V7; V7V1; V7V2; V7V3; V7V4; V7V5; V7V6; V7V7;

И эти ребра имеют следующие веса:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
| V1 | 0 | 3 | 4 | 19 | 9 | 16 | 10 |
| V2 | 3 | 0 | 16 | 6 | 8 | 7 | 7 |
| V3 | 4 | 16 | 0 | 5 | 25 | 8 | 1 |
| V4 | 19 | 6 | 5 | 0 | 15 | 2 | 14 |
| V5 | 9 | 8 | 25 | 15 | 0 | 7 | 9 |
| V6 | 16 | 7 | 8 | 2 | 7 | 0 | 23 |
| V7 | 10 | 7 | 1 | 14 | 9 | 23 | 0 |

После сортировки по весу ребер получим:

|  |  |
| --- | --- |
| Вес | Ребро |
| 1 | V3V7 |
| 2 | V4V6 |
| 3 | V1V2 |
| 4 | V1V3 |
| 5 | V3V4 |
| 6 | V2V4 |
| 7 | V2V6 |
| 7 | V2V7 |
| 7 | V5V6 |
| 8 | V2V5 |
| 8 | V3V5 |
| 9 | V1V5 |
| 9 | V5V7 |
| 10 | V1V7 |
| 14 | V4V7 |
| 15 | V4V3 |
| 16 | V1V6 |
| 16 | V2V3 |
| 19 | V1V4 |
| 23 | V6V7 |
| 25 | V3V5 |

Алгоритм

Изначально множество вершин выглядит так:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| V1 | V2 | V3 | V4 | V5 | V6 | V7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

|  |  |  |  |
| --- | --- | --- | --- |
| Вес | Ребро | Множество вершин | Вес дерева |
| 1 | V3V7 | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | | 1 | 2 | 3 | 4 | 5 | 6 | 3 | | 1 |
| 2 | V4V6 | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | | 1 | 2 | 3 | 4 | 5 | 4 | 3 | | 3 |
| 3 | V1V2 | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | | 1 | 1 | 3 | 4 | 5 | 4 | 3 | | 6 |
| 4 | V1V3 | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | | 1 | 1 | 1 | 4 | 5 | 4 | 1 | | 10 |
| 5 | V3V4 | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | | 1 | 1 | 1 | 1 | 5 | 1 | 1 | | 15 |
| 6 | V2V4 | Ребро образует цикл | - |
| 7 | V2V6 | Ребро образует цикл | - |
| 7 | V2V7 | Ребро образует цикл | - |
| 7 | V5V6 | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 22 |
| 8 | V2V5 | Работа алгоритма завершена |  |
| 8 | V3V5 | Работа алгоритма завершена | - |
| 9 | V1V5 | Работа алгоритма завершена | - |
| 9 | V5V7 | Работа алгоритма завершена | - |
| 10 | V1V7 | Работа алгоритма завершена | - |
| 14 | V4V7 | Работа алгоритма завершена | - |
| 15 | V4V3 | Работа алгоритма завершена | - |
| 16 | V1V6 | Работа алгоритма завершена | - |
| 16 | V2V3 | Работа алгоритма завершена | - |
| 19 | V1V4 | Работа алгоритма завершена | - |
| 23 | V6V7 | Работа алгоритма завершена | - |
| 25 | V3V5 | Работа алгоритма завершена | - |

# **4. Описание программы**

Спрашиваем у пользователя количество вершин в графе.

Спрашиваем у пользователя вид графа (а или б).

Записываем результат в переменную NumberOfVertices.

Используя цикл for и переменную NumberOfVertices, заполняем динамические массивы. Первый (Ribs) – массив названия ребра. Второй (Weight) – массив веса этого ребра. Если вес ребра равен 0, то такое ребро пропускается.

Сортируем одновременно 2 массива по не убыванию веса ребер.

Создаем двухмерный массив первая строка которого отвечает за вершины, вторая за метки этих вершин.

Создаем динамический массив, в который будем помещать ребра, которые подходят по алгоритму Краскала.

Создаем цикл.

При помощи функции IdentificationRib меняем значение переменных, отвечающих за индекс ребра, вершину из которой выходит ребра и вершину в которое входит ребро.

При помощи функции findLableVertex меняем значение переменных, отвечающих за метку вершины из которой выходит ребро и метку вершины в которую это ребро входит.

При помощи функции CheckingForCycle узнаем создает ли ребро цикл. Если ребро создает цикл, то программа переходит к следующему ребру. Если ребро не создает цикл, то программа работает с ним дальше.

При помощи функции findMinLable меняем значение перемееной MinLable на значение минимальной метки.

При помощи функции findMaxLable меняем значение переменной MaxLable на значение максимальной метки.

При помощи функции unificationLable меням в двухмерном массиве меняем во второй строчку значения, совпадающие с максимальной меткой, на значение минимальной метки.

Добавляем ребро в динамический массив.

При помощи функции sumCounter подсчитываем сумму минимального остового дерева.

Повторяем цикл до тех пор, пока не пройдем по всем ребрам.

# **5.Тестирование программы**

\*Жирным курсивом выделено то что вводит пользователь

Введите количество вершин

***7***

Выберите какой граф

а) ориентированный б) неориентированный

***а***

Введите веса всех ребер соединяемых вершин (Пример: V1V2 - соединяет вершины V1 и V2), если ребра нет введите 0

Введите V1V1 ***0***

Введите V1V2 ***3***

Введите V1V3 ***4***

Введите V1V4 ***16***

Введите V1V5 ***9***

Введите V1V6 ***16***

Введите V1V7 ***10***

Введите V2V1 ***3***

Введите V2V2 ***0***

Введите V2V3 ***16***

Введите V2V4 ***6***

Введите V2V5 ***8***

Введите V2V6 ***7***

Введите V2V7 ***7***

Введите V3V1 ***4***

Введите V3V2 ***16***

Введите V3V3 ***0***

Введите V3V4 ***5***

Введите V3V5 ***25***

Введите V3V6 ***8***

Введите V3V7 ***1***

Введите V4V1 ***19***

Введите V4V2 ***6***

Введите V4V3 ***5***

Введите V4V4***0***

Введите V4V5 ***15***

Введите V4V6 ***2***

Введите V4V7 ***14***

Введите V5V1 ***9***

Введите V5V2 ***8***

Введите V5V3 ***25***

Введите V5V4 ***15***

Введите V5V5 ***0***

Введите V5V6 ***7***

Введите V5V7 ***9***

Введите V6V1 ***16***

Введите V6V2 ***7***

Введите V6V3 ***8***

Введите V6V4 ***2***

Введите V6V5 ***7***

Введите V6V6 ***0***

Введите V6V7 ***23***

Введите V7V1 ***10***

Введите V7V2 ***7***

Введите V7V3 ***1***

Введите V7V4 ***14***

Введите V7V5 ***9***

Введите V7V6 ***23***

Введите V7V7 ***0***

Для получения минимального остовного дерева следует оставить следующие ребра

V7V3 V6V4 V2V1 V3V1 V4V3 V6V5

Вес минимального остовного дерева равен 22

# **6. Текст программы**.

*Приложение 1* – код программы.

using System;

using System.Collections.Generic;

namespace LB\_MOP\_3\_BETA

{

    internal class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Введите количество вершин");

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

Console.WriteLine("Выберите какой граф");

Console.WriteLine("а) ориентированный б) неориентированный");

char Graf = Convert.ToChar(Console.ReadLine());

bool graf;

if (Graf == 'а' || Graf == 'a')

{

graf = true;

}

else

{

graf = false;

}

            Console.WriteLine("Введите веса всех ребер (Пример: V1V2 - соединяет вершины V1 и V2) ");

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

            List<string> Ribs = new List<string>();

            if (graf)

{

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

{

for (int j = 1; j <= NumberOfVertices; j++)

{

string z = "V" + i + "V" + j;

Console.Write($"Введите {z} ");

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

if (c == 0)

{

continue;

}

else

{

Ribs.Add(z);

Weight.Add(c);

}

}

}

}

else

{

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

{

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

{

string z = "V" + i + "V" + j;

Console.Write($"Введите {z} ");

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

if (c == 0)

{

continue;

}

else

{

Ribs.Add(z);

Weight.Add(c);

}

}

}

}

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

            {

                for (int j = i; j < Ribs.Count; j++)

                {

                    if (Weight[i] >= Weight[j])

                    {

                        int swapInt = Weight[i];

                        Weight[i] = Weight[j];

                        Weight[j] = swapInt;

                        string swapString = Ribs[i];

                        Ribs[i] = Ribs[j];

                        Ribs[j] = swapString;

                    }

                }

            }

            int[,] arrayArrays = new int[2, NumberOfVertices];

            for (int i = 0, j = 0; j < arrayArrays.GetLength(1); j++)

            {

                arrayArrays[i, j] = j+1;

            }

            for (int i = 1, j = 0; j < arrayArrays.GetLength(1); j++)

            {

                arrayArrays[i, j] = j+1;

            }

            int TheVertexOfEdgeExit=0;

            int TheVertexOfEdgeEntry=0;

            int LableTheVertexOfEdgeExit = 0;

            int LableTheVertexOfEdgeEntry = 0;

            int MinLabel;

            int MaxLabel;

            int SumGrafs = 0;

            int indexRib = -1;

            List<string> ResultRibs = new List<string>();

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

            {

                IdentificationRib(Ribs[i], ref TheVertexOfEdgeExit, ref TheVertexOfEdgeEntry, ref indexRib);

                findLableVertex(arrayArrays, ref TheVertexOfEdgeExit, ref TheVertexOfEdgeEntry, ref LableTheVertexOfEdgeExit, ref LableTheVertexOfEdgeEntry);

                if (!CheckingForCycle(LableTheVertexOfEdgeExit, LableTheVertexOfEdgeEntry))

                {

                    MinLabel = findMinLable(arrayArrays, TheVertexOfEdgeExit, TheVertexOfEdgeEntry);

                    MaxLabel = findMaxLable(arrayArrays, TheVertexOfEdgeExit, TheVertexOfEdgeEntry);

                    unificationLable(ref arrayArrays, TheVertexOfEdgeExit, TheVertexOfEdgeEntry, MinLabel, MaxLabel);

                    ResultRibs.Add(Ribs[i]);

                    sumCounter(Weight, indexRib, ref SumGrafs);

                }

            }

            Console.WriteLine("Для получения минимального остовного дерева следует оставить следующие ребра");

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

            {

                Console.Write($"{ResultRibs[i]} ");

            }

            Console.WriteLine();

            Console.WriteLine($"Вес минимального остовного дерева равен {SumGrafs}");

            Console.ReadKey();

        }

        static void IdentificationRib(string Rib, ref int TheVertexOfEdgeExit, ref int TheVertexOfEdgeEntry, ref int indexRib)

        {

            indexRib++;

            string VertexExit = "", VertexEntry = "";

            int indexV = 0;

            for (int i = 1; i < Rib.Length; i++)

            {

                if (Rib[i] == 'V')

                {

                    indexV = i;

                    break;

                }

                VertexExit = VertexExit + Rib[i];

            }

            for (int i = indexV + 1; i < Rib.Length; i++)

            {

                VertexEntry = VertexEntry + Rib[i];

            }

            TheVertexOfEdgeExit = Convert.ToInt32(VertexExit);

            TheVertexOfEdgeEntry = Convert.ToInt32(VertexEntry);

        }

        static void findLableVertex(int[,] arrayArrays, ref int TheVertexOfEdgeExit, ref int TheVertexOfEdgeEntry, ref int LableTheVertexOfEdgeExit, ref int LableTheVertexOfEdgeEntry)

        {

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

            {

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

                {

                    if (arrayArrays[i, j] == TheVertexOfEdgeExit)

                    {

                        LableTheVertexOfEdgeExit = arrayArrays[1, j];

                        break;

                    }

                }

            }

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

            {

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

                {

                    if (arrayArrays[i, j] == TheVertexOfEdgeEntry)

                    {

                        LableTheVertexOfEdgeEntry = arrayArrays[1, j];

                        break;

                    }

                }

            }

        }

        static bool CheckingForCycle(int LableTheVertexOfEdgeExit, int LableTheVertexOfEdgeEntry)

        {

            if (LableTheVertexOfEdgeExit == LableTheVertexOfEdgeEntry)

            {

                return true;

            }

            else

            {

                return false;

            }

        }

        static int findMinLable(int[,] array, int FirstVertex, int SecondVertex)

        {

            int MinLable = array[1, FirstVertex-1];

            if (MinLable > array[1, SecondVertex - 1])

            {

                MinLable = array[1, SecondVertex - 1];

            }

            return MinLable;

        }

        static int findMaxLable(int[,] array, int FirstVertex, int SecondVertex)

        {

            int MaxLable = array[1, FirstVertex - 1];

            if (MaxLable < array[1, SecondVertex - 1])

            {

                MaxLable = array[1, SecondVertex - 1];

            }

            return MaxLable;

        }

        static void unificationLable(ref int[,] arrayArrays, int TheVertexOfEdgeExit, int TheVertexOfEdgeEntry, int MinLabel, int MaxLabel)

        {

            for (int i = 1; i < 2; i++)

            {

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

                {

                    if (arrayArrays[1, j] == MaxLabel)

                    {

                        arrayArrays[1, j] = MinLabel;

                    }

                }

            }

        }

        static void sumCounter(List<int> Weight, int indexRib, ref int SumGrafs)

        {

            SumGrafs += Weight[indexRib];

        }

    }

}