Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Дискретная математика

Лабораторная работа № 2

Тема: «Минимальный остов»

Выполнил: студентка группы ИВТ-23-1б

Хорошилова К. П.

Проверил: ст. преподаватель

Рустамханова Г. И.

г. Пермь – 2025**Цель работы**

Изучить алгоритмы поиска минимального остовного дерева (МОД) во взвешенном неориентированном графе.

**Задачи работы**

# Изучить понятие остовного дерева.

# Реализовать алгоритм поиска МОД, используя алгоритм Прима или Крускала.

1. Реализовать алгоритм.
2. Написать отчёт.

# Код программы

using System.Drawing;

using System.Runtime.ExceptionServices;

using System.Text.RegularExpressions;

using System.Xml.Linq;

namespace Дискретная\_математика\_\_\_минимальный\_остов

{

internal class Program

{

/\*

static void EnteringMatrix(int[,] matrix)

{

Console.WriteLine("Введите размерность матрицы");

int size = 0;

Int32.TryParse(Console.ReadLine(), out size);

matrix = new int[size, size];

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

{

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

{

Console.WriteLine("Введите элемент в ечейку (" + (i + 1) + ", " + (j + 1) + ") ");

Int32.TryParse(Console.ReadLine(), out matrix[i, j]);

}

}

}\*/

static string Assembling(string str)

{

string vector = "";

Regex regex = new Regex(@"\w+");

MatchCollection matches = regex.Matches(str);

foreach (Match match in matches)

{

vector += match.Value;

}

return vector;

}

static void EnteringMatrixFile(ref int[,] matrix)

{

string path = "C:\\Users\\Asus\\source\\repos\\disc2\_ostov\\disc2\_ostov\\Matrix.txt";

if (!File.Exists(path))

{

Console.WriteLine("Файл не найден.");

return;

}

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

if (lines.Length == 0)

{

Console.WriteLine("Файл пуст.");

return;

}

// Определяем размерность матрицы по первой строке

string[] firstLine = Regex.Split(lines[0].Trim(), @"\s+");

int size = firstLine.Length;

matrix = new int[size, size];

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

{

string[] elements = Regex.Split(lines[i].Trim(), @"\s+"); // Разделяем по пробелам или табам

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

{

if (!int.TryParse(elements[j], out matrix[i, j]))

{

Console.WriteLine($"Ошибка чтения числа в строке {i + 1}, столбце {j + 1}");

return;

}

}

}

}

/\*

static void InputMatrix(ref int[,] matrix)

{

bool isEnd = false;

string[] mas = new string[3];

mas[0] = "1) Ввод в ручную";

mas[1] = "2) Ввод из файла";

foreach (string str in mas) { Console.WriteLine(str); }

Console.WriteLine("Ввидите номер действия");

int number = 0;

Int32.TryParse(Console.ReadLine(), out number);

if (number == 1)

{

EnteringMatrix(matrix);

}

else

{

EnteringMatrixFile(ref matrix);

}

}\*/

public static void Print(int[,] matrix)

{

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

{

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

{

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

}

Console.WriteLine();

}

}

//Алгоритм Крускала

public static void ConstructionSkeleton(int[,] matrix)

{

int[,] ribs = new int[0, 0];

CheckingRibs(matrix, ref ribs);

SortRibs(ribs);

int[] chekRibs = new int[matrix.GetLength(0)];

int size = 0;

int temp = 0;

temp = size = ribs[0, 2];

chekRibs[ribs[0, 0]] = 1;

chekRibs[ribs[0, 1]] = 1;

Console.WriteLine((ribs[0, 0] + 1) + "----" + (ribs[0, 1] + 1) + "-------" + ribs[0, 2]);

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

matrixSkeleton[ribs[0, 0], ribs[0, 1]] = 1;

bool notRibs = true;

bool isEnd = false;

bool lastElement = true;

while (!isEnd && notRibs)

{

int i = 0;

while (chekRibs[i] != 0 && lastElement)

{

i++;

if (i == chekRibs.Length)

{

lastElement = false;

i = 0;

}

}

if (!lastElement)

{

isEnd = true;

}

else

{

size += ChekMinSize(chekRibs, ribs, matrixSkeleton);

if (temp == size)

{

notRibs = false;

}

else

{

temp = size;

}

}

}

if (!notRibs)

{

Console.WriteLine("Не все ребра графа соединены");

}

Console.WriteLine();

Print(matrixSkeleton);

Console.WriteLine();

Console.WriteLine("Вес остова равен " + size);

}

public static void CheckingRibs(int[,] matrix, ref int[,] ribs)

{

int size = 0;

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

{

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

{

if (matrix[i, j] != 0)

{

size++;

}

}

}

ribs = new int[size, 3];

int count = 0;

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

{

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

{

if (matrix[i, j] != 0)

{

ribs[count, 0] = i;

ribs[count, 1] = j;

ribs[count, 2] = matrix[i, j];

count++;

}

}

}

}

public static void SortRibs(int[,] matrix)

{

int min = 0;

int minJ = 0;

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

{

min = matrix[i, 2];

minJ = i;

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

{

if (matrix[j, 2] < min)

{

min = matrix[j, 2];

minJ = j;

}

}

if (minJ != i)

{

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

{

int temp = (int)matrix[i, j];

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

matrix[minJ, j] = temp;

}

}

}

}

static public int ChekMinSize(int[] chekRibs, int[,] ribs, int[,] matrixSkeleton)

{

int size = 0;

int element1 = 0, element2 = 0;

bool isFirst = true;

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

{

if (chekRibs[i] != 0)

{

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

{

if (ribs[j, 0] == i)

{

if (chekRibs[ribs[j, 1]] == 0)

{

if (isFirst)

{

element1 = i;

element2 = ribs[j, 1];

size = ribs[j, 2];

isFirst = false;

}

else

{

if (size > ribs[j, 2])

{

size = ribs[j, 2];

element1 = i;

element2 = ribs[j, 1];

}

}

}

}

}

}

}

matrixSkeleton[element1, element2] = 1;

matrixSkeleton[element2, element1] = 1;

chekRibs[element2] = 1;

Console.WriteLine((element1 + 1) + "----" + (element2 + 1) + "-------" + size);

return size;

}

//Алгоритм Крускала Конец

//Алгоритм Прима

public static bool Cheking(int[] chek)

{

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

{

if (chek[i] == 0)

{

return false;

}

}

return true;

}

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

{

int[,] newMatrix = 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++)

{

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

}

}

return newMatrix;

}

static void AlgoritmPrima(int[,] matrix)

{

Random random = new Random();

int[] listPoints = new int[matrix.GetLength(0)];

int[] pointsSkeleton = new int[matrix.GetLength(0)];

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

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

int minSize = 0;

int tempSize = 0;

bool firstElement = true;

while (!Cheking(listPoints))

{

int element = 0;

do

{

element = random.Next(matrix.GetLength(0));

}

while (listPoints[element] != 0);

listPoints[element] = 1;

pointsSkeleton[element] = 1;

if (firstElement)

{

CalculationSkeleton(matrix, pointsSkeleton, temp, ref minSize);

matrixAdjance = Overwritng(temp);

firstElement = false;

}

else

{

CalculationSkeleton(matrix, pointsSkeleton, temp, ref tempSize);

if (tempSize < minSize)

{

minSize = tempSize;

matrixAdjance = Overwritng(temp);

}

}

tempSize = 0;

pointsSkeleton = new int[matrix.GetLength(0)];

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

}

//Print(matrixAdjance);

//Console.WriteLine("Вес остова " + minSize);

}

static void CalculationSkeleton(int[,] matrix, int[] points, int[,] matrixAdjacency, ref int size)

{

int sizePoint = 0;

int numberPoint1 = 0;

int numberPoint2 = 0;

bool firsElement = true;

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

{

if (points[i] == 1)

{

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

{

if (matrix[i, j] != 0)

{

if (points[j] == 0)

{

if (firsElement)

{

sizePoint = matrix[i, j];

numberPoint1 = i;

numberPoint2 = j;

firsElement = false;

}

else

{

if (sizePoint > matrix[i, j])

{

sizePoint = matrix[i, j];

numberPoint1 = i;

numberPoint2 = j;

}

}

}

}

}

}

}

size += sizePoint;

points[numberPoint2] = 1;

matrixAdjacency[numberPoint1, numberPoint2] = 1;

matrixAdjacency[numberPoint2, numberPoint1] = 1;

if (!Cheking(points))

{

CalculationSkeleton(matrix, points, matrixAdjacency, ref size);

}

}

//Алгоритм Прима Конец

static void Main()

{

int[,] matrix = new int[0, 0];

string[] menu = {

"1) Ввод матрицы из файла",

"2) Построение минимального остова",

"3) Закончить работу"

};

bool isEnd = false;

do

{

foreach (string option in menu)

{

Console.WriteLine(option);

}

Console.Write("Выберите действие: ");

if (!int.TryParse(Console.ReadLine(), out int choice)) continue;

switch (choice)

{

case 1:

EnteringMatrixFile(ref matrix);

Print(matrix);

Console.WriteLine();

break;

case 2:

if (matrix.Length == 0)

{

Console.WriteLine("Матрица не загружена.");

break;

}

ConstructionSkeleton(matrix);

AlgoritmPrima(matrix);

Console.WriteLine();

break;

case 3:

isEnd = true;

break;

}

} while (!isEnd);

}

}

}

**Анализ кода**

Программа принимает на вход матрицу расстояний 10×10, где каждое значение обозначает вес ребра между вершинами, а отсутствие связи может обозначаться, например, бесконечностью. Для нахождения минимального остовного дерева используются два основных алгоритма: Прима и Крускала. Алгоритм Прима начинает с произвольной вершины и пошагово добавляет минимальное возможное ребро, пока все вершины не окажутся в дереве. Его сложность составляет O(n²) при использовании матрицы. Алгоритм Крускала сначала сортирует рёбра по весу, а затем добавляет их в дерево, избегая циклов, используя структуру данных для поиска и объединения. Его сложность составляет O(E log E), где E – количество рёбер.

Метод Assembling выполняет обработку входной строки, извлекая из неё все слова с помощью регулярных выражений и объединяя их в одну строку.

Метод EnteringMatrixFile загружает матрицу смежности из файла, проверяя его существование и корректность данных. Он использует регулярные выражения для разбиения строк по пробелам и табуляциям, затем преобразует их в числовые значения, формируя двумерный массив.

Метод Print выводит матрицу на экран, проходя по всем её элементам и форматируя вывод в виде таблицы.

Метод ConstructionSkeleton реализует алгоритм Краскала. Он сначала извлекает список рёбер графа с помощью CheckingRibs, сортирует их по весу через SortRibs, а затем добавляет рёбра в остовное дерево, избегая циклов. Проверка минимального возможного добавляемого ребра выполняется через ChekMinSize, который выбирает минимальное рёбра, соединяющее новые вершины с уже добавленными в остов. В конце алгоритм выводит матрицу остовного дерева и его вес.

Метод CheckingRibs проходит по матрице смежности и формирует массив рёбер, содержащий начальную и конечную вершины, а также вес ребра.

Метод SortRibs выполняет сортировку рёбер по весу, используя алгоритм выбора, переставляя элементы массива.

Метод ChekMinSize ищет минимальное по весу ребро, соединяющее вершины, уже входящие в остов, с оставшимися. Он обновляет матрицу остовного дерева, добавляет вершину в список посещённых и выводит выбранное ребро.

Метод AlgoritmPrima реализует алгоритм Прима. Он случайным образом выбирает стартовую вершину и последовательно добавляет минимальное ребро, соединяющее уже выбранные вершины с оставшимися.

Метод Cheking проверяет, все ли вершины уже включены в остовное дерево, чтобы завершить работу алгоритма.

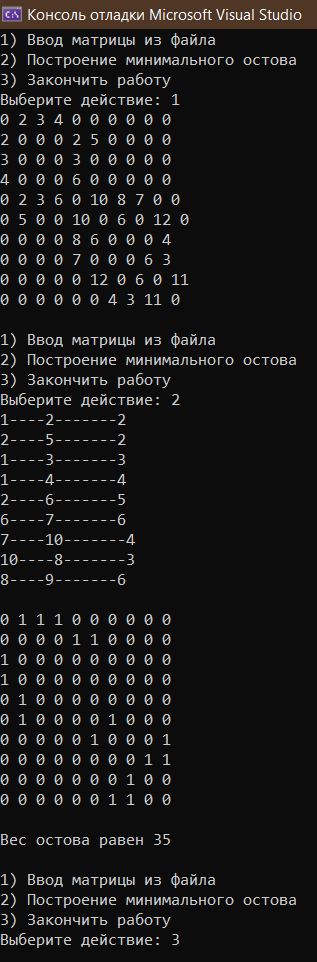
Метод Overwritng создаёт копию матрицы, чтобы избежать изменения исходных данных.

Метод CalculationSkeleton выполняет основную логику алгоритма Прима. Он ищет минимальное ребро, соединяющее уже добавленные вершины с остальными, обновляет список посещённых и рекурсивно продолжает поиск, пока не будут соединены все вершины.

Метод main организует консольное меню, позволяя пользователю загружать матрицу, запускать алгоритмы поиска минимального остовного дерева и завершать выполнение программы.

Ссылка на Github: <https://github.com/ksinph/disc_math>

**Тесты**



# Заключение

# В ходе лабораторной работы я реализовала построение минимального остова и определили его вес.