Звіт

Програмування інтелектуальних інформаційних систем

Лабораторна робота №4

“Дослідження алгоритмів пошуку підрядочків у рядку та найкоротшого шляху та графі”

Студентки групи ІП-01

Галько Міли Вячеславівни

Лабораторна робота №4

"Дослідження алгоритмів пошуку підрядочків у рядку та найкоротшого шляху та графі"

Завдання:

1. Реалізувати алгоритм ***Карпа-Рабіна***, обравши хеш-функцією результат ділення за модулем на номер Вашого варіанту. Протестувати роботу алгоритму над шаблонами у 6-8 символів.

2. Реалізувати алгоритм ***Дейкстри*** згідно заданого варіанту.

3. Реалізувати алгоритм ***Пріма*** згідно заданого варіанту.

Варіант №6:

1.1. Задано орієнтований граф та час переходу від однієї вершини до іншої: 25=4, 28=4, 23=3, 24=1, 34=3, 35=2, 42=3, 47=3, 56=5, 57=2, 62=1, 63=3, 65=2, 67=1, 68=3, 76=2, 72=1, 83=2, 84=2, 87=4. Необхідно знайти найкоротші відстані від заданої вершини до інших.

1.2. Задано неорієнтований граф та час переходу від однієї вершини до іншої: 21=4, 28=4, 23=3, 24=1, 31=3, 35=2, 42=3, 47=3, 56=5, 57=2, 62=1, 63=3, 65=2, 67=1, 68=3, 76=2, 72=1, 83=2, 84=2, 87=4. Необхідно побудувати мінімальне остовне дерево за допомогою алгоритму Пріма.

Реалізація завдання 1-3. Підготовка вхідних даних:

Маємо створити 3 класи для відтворення роботи усіх алгоритмів: RabinKarpAlgo, DijkstraAlgo та PrimaAlgo. На вході повинні передавати вхідні дані.

Для RabinKarpAlgo передаємо шаблон та текст.

Для DijkstraAlgo та PrimaAlgo вхідні дані для створення матриці задаються стрічкою. Отже додатково створюємо метод Parser, що приймає її у якості аргументу та булеву змінну (true, якщо матрицю має бути симетричною, як для алгоритму Пріма). Далі працюємо з готовими матрицями для Дейкстри та Пріми.

Для DijkstraAlgo додатково можемо вказати стартову вершину (за замовчуванням перша).

Реалізація завдання 1-3. Підготовка вхідних даних. Код:

```
// DATA
string template1 = "owners";
string text1 = "When Mila's dog barks like 'doGS and ownersOwneRs go out',
oWners are not responding";
string variant6_1 = "25=4, 28=4, 23=3, 24=1, 34=3, 35=2, 42=3, 47=3, 56=5,
57=2, 62=1, 63=3, 65=2, 67=1, 68=3, 76=2, 72=1, 83=2, 84=2, 87=4";
string variant6_2 = "21=4, 28=4, 23=3, 24=1, 31=3, 35=2, 42=3, 47=3, 56=5,
57=2, 62=1, 63=3, 65=2, 67=1, 68=3, 76=2, 72=1, 83=2, 84=2, 87=4";


// CALLING
int[,] task1 = Parser(variant6_1, 9, false);
int[,] task2 = Parser(variant6_2, 8, true);

RabinKarp(template1, text1);
Dijkstra(task1, 1);
Prima(task2);

// RABIN-KARP-ALGORITHM
void RabinKarp(string template, string text)
{
    var rabinKarp = new RabinKarpAlgo();
    int entries = rabinKarp.CountEntries(template, text);

    Console.WriteLine("RABIN-KARP-ALGO:");
    Console.WriteLine($"Text: {text1} \nTemplate: {template1}");
    Console.WriteLine($"Entries: {entries}");
    Console.WriteLine();
}

// DIJKSTRA-ALGORITHM
void Dijkstra(int[,] matrix, int start = 0)
{
    var dijkstra = new DijkstraAlgo();
    var results = dijkstra.Solve(matrix, start);

    Console.WriteLine("DIJKSTRA-ALGO:");
    PrintGraph(matrix);
    Console.WriteLine(string.Join(", ", results));
    Console.WriteLine();
}

// PRIMA-ALGORITHM
void Prima(int[,] matrix)
{
    var prima = new PrimaAlgo();
    var primaResult = prima.Solve(matrix);

    Console.WriteLine("PRIMA-ALGO:");
    PrintGraph(matrix);
    PrintGraph(primaResult);
    Console.WriteLine($"OctTree weight: {prima.Weight}");
    Console.WriteLine();
}
```

Реалізація завдання 1:

Нехай маємо шаблон «owners» та основний текст
«When Mila's dog barks like 'doGS and ownersOwneRs go out', oWners are not responding».
Видно, що даний шаблон зустрічається 3 рази, при не звертанні уваги на регістр букв.

Код класу RabinKarpAlgo:

```
public class RabinKarpAlgo
{
    private int _templateSize;
    private int _templateHash;
    private string _text;
    private string _template;

    private int GetHash(string str)
    {
        int hash = 0;
        foreach (var symbol in str)
        {
            hash += symbol;
        }
        return hash / 6;
    }

    public int CountEntries(string template, string text)
    {
        _text = text.ToLower();
        _template = template.ToLower();
        _templateSize = _template.Length;
        _templateHash = GetHash(_template);
        int entries = 0;

        for (int i = 0; i <= _text.Length - _templateSize; i++)
        {
            string subString = _text.Substring(i, _templateSize);
            int subStringHash = GetHash(subString);
            if (subStringHash == _templateHash && subString == _template)
            {
                entries++;
                i += _templateSize - 1;
            }
        }

        return entries;
    }
}
```

Результат:

```
RABIN-KARP-ALGO:
Text: When Mila's dog barks like 'doGS and ownersOwneRs go out', oWners are not responding
Template: owners
Entries: 3
```

Реалізація завдання 2:

Для більшої достовірності відтворимо даний варіант ручним та програмним способом.

Також за старт оберемо вершину №2 оскільки з №1 немає зв'язки.

На перших ітераціях формуємо шляхи без особливих проблем (рис.1)

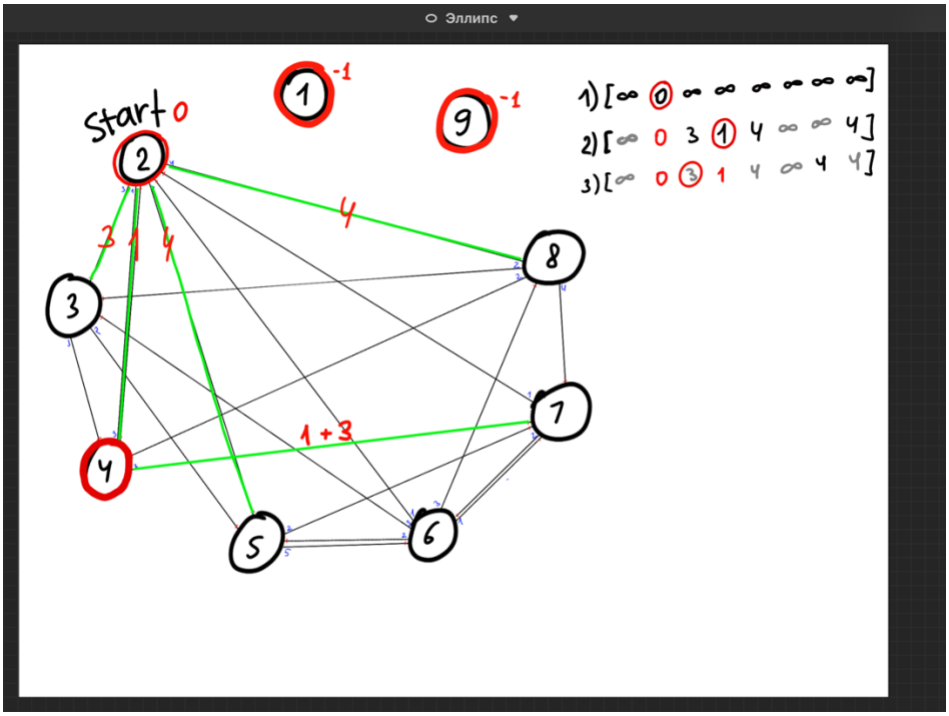

Рис. 1 – Алгоритм Дейкстри вручну

На 6 ітерації маємо спрощення (рис.2)

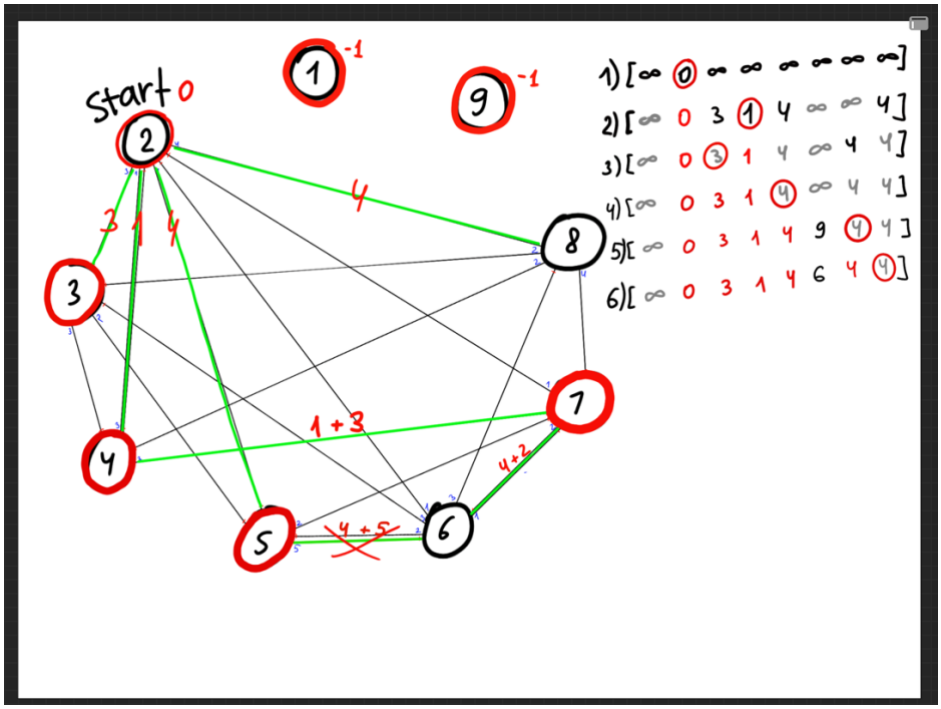

Рис.2 – Алгоритм Дейкстри 6-а ітерація

На 8 ітерації усе завершується (рис.3) і отримуємо фінальний вигляд шляхів (рис.4).

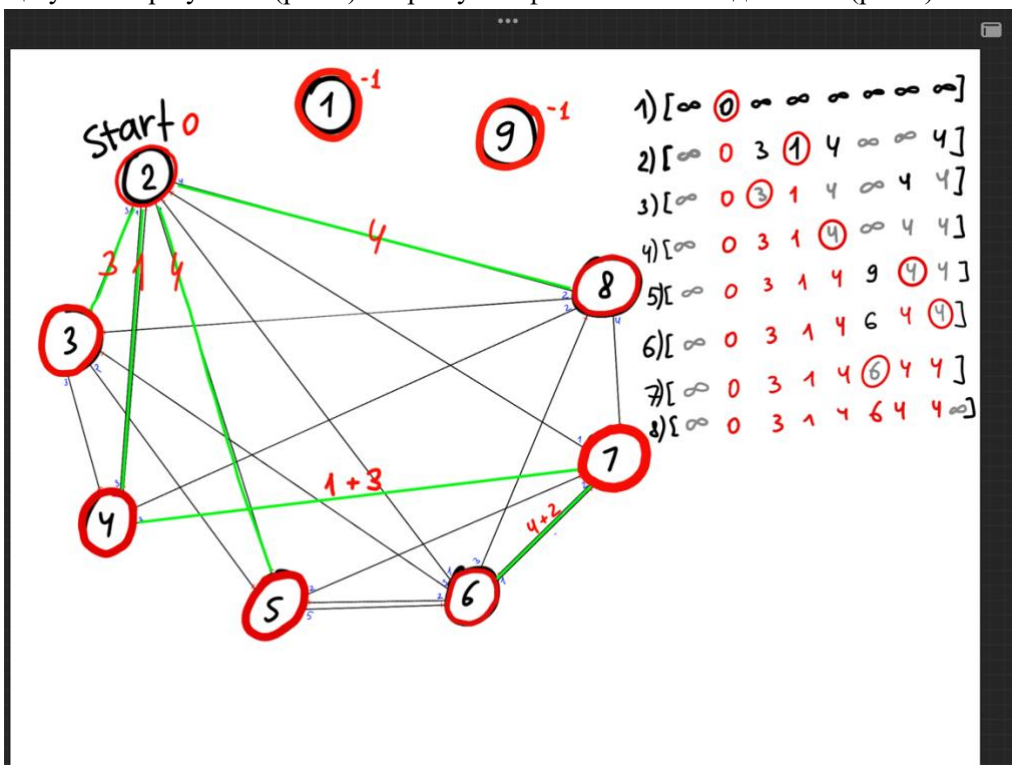

Рис.3 – Усі ітерації алгоритму Дейкстри

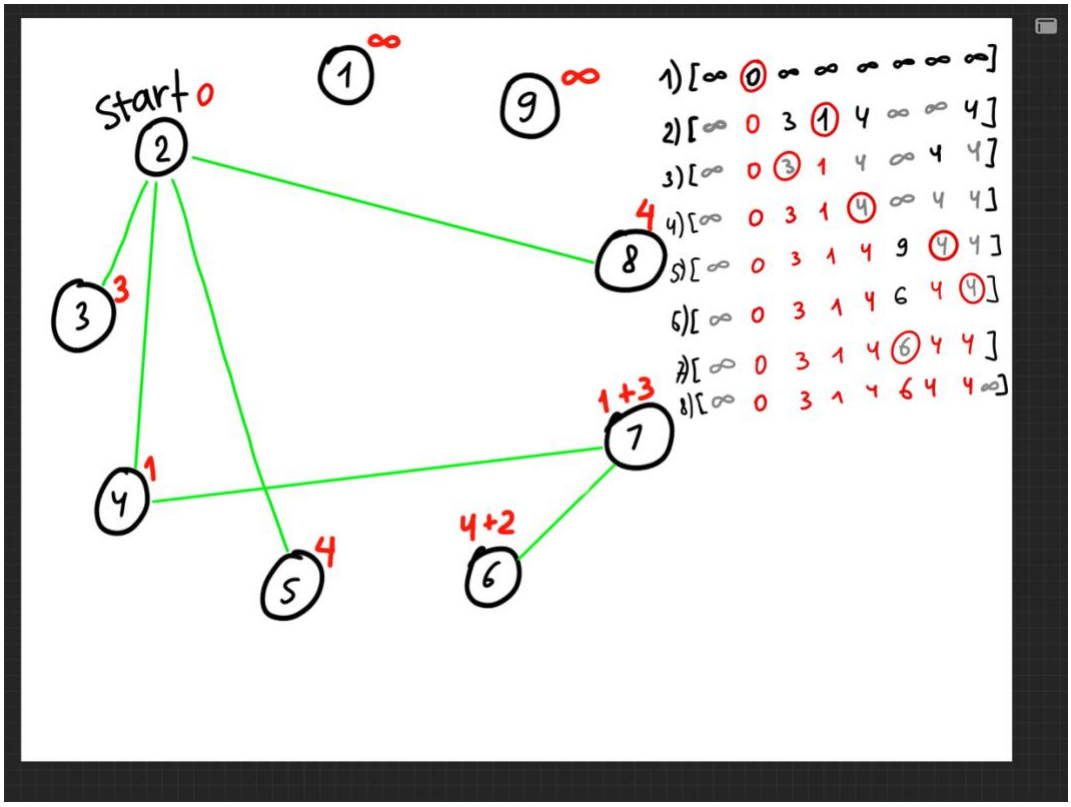

Рис.4 – Фінальний результат алгоритму Дейкстри

Код класу DijkstraAlgo:

```
public class DijkstraAlgo
{
    private int _size;
    private int[] _shortestPathes;
    private bool[] _visitedNodes;
    private int[,] _matrix;

    public int[] Solve(int[,] matrix, int start)
    {
        _size = matrix.GetLength(0);
        _visitedNodes = new bool[_size];
        _shortestPathes = Enumerable.Repeat(int.MaxValue, _size).ToArray();
        _shortestPathes[start] = 0;
        _matrix = matrix;

        int node;
        while ((node = NearestNodeIndex()) != -1)
        {
            UpdatePathesLength(node);
            _visitedNodes[node] = true;
        }
        return _shortestPathes.Select(path => path == int.MaxValue ? -1 :
path).ToArray();
    }

    private int NearestNodeIndex()
    {
        int pathLength = int.MaxValue;
        int nearestNode = -1;
        for (int node = 0; node < _size; node++)
        {
            if (!_visitedNodes[node] && _shortestPathes[node] < pathLength)
            {
                pathLength = _shortestPathes[node];
                nearestNode = node;
            }
        }
        return nearestNode;
    }

    private void UpdatePathesLength(int node)
    {
        for (int dest = 0; dest < _size; dest++)
        {
            int length = _matrix[node, dest];
            if (_matrix[node, dest] != 0 &&
               !_visitedNodes[dest] &&
                length + _shortestPathes[node] < _shortestPathes[dest])
            {
                _shortestPathes[dest] = length + _shortestPathes[node];
            }
        }
    }
}
```

Результат:

```
DIJKSTRA-ALGO:
0  0  0  0  0  0  0  0  0
0  0  3  1  4  0  0  4  0
0  0  0  3  2  0  0  0  0
0  3  0  0  0  0  3  0  0
0  0  0  0  0  5  2  0  0
0  1  3  0  2  0  1  3  0
0  1  0  0  0  2  0  0  0
0  0  2  2  0  0  4  0  0
0  0  0  0  0  0  0  0  0


-1, 0, 3, 1, 4, 6, 4, 4, -1
```

Реалізація завдання 3:

Підемо тим же шляхом, що й при виконанні 2 завдання і віднайдемо рішення 2-а способами.

При вирішенні завдання вручну (рис.5) отримали мінімальне остовне дерево (рис.6) із вагою 13.

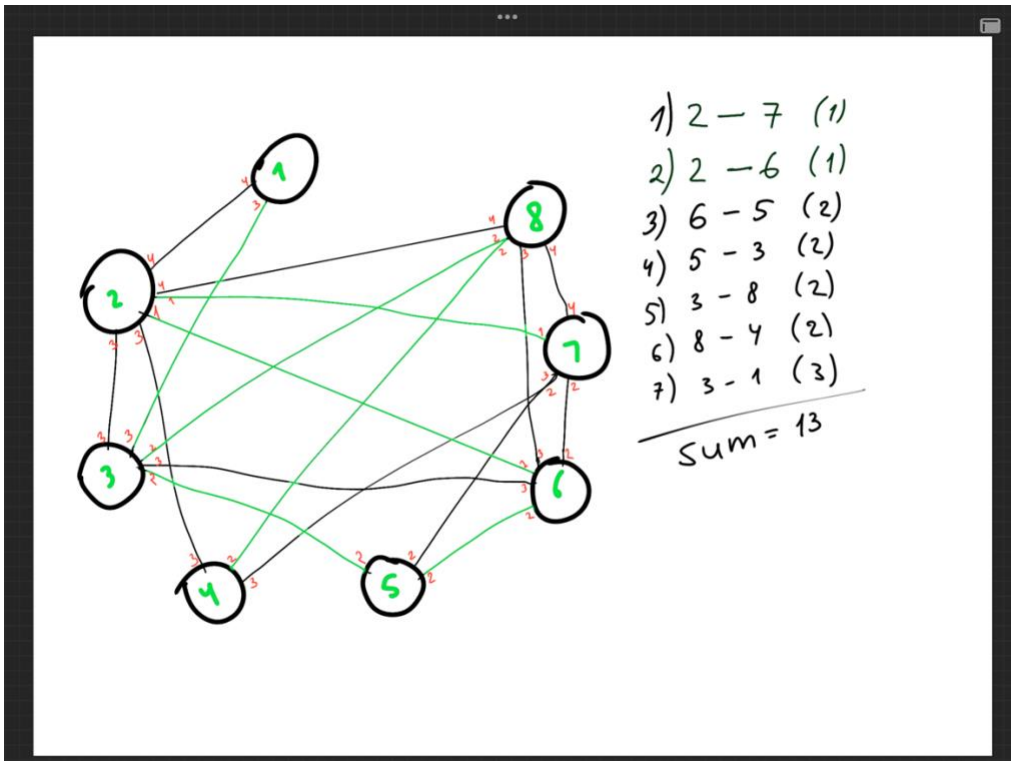

Рис.5 – Хід розв’язку завдання алгоритмом Прима

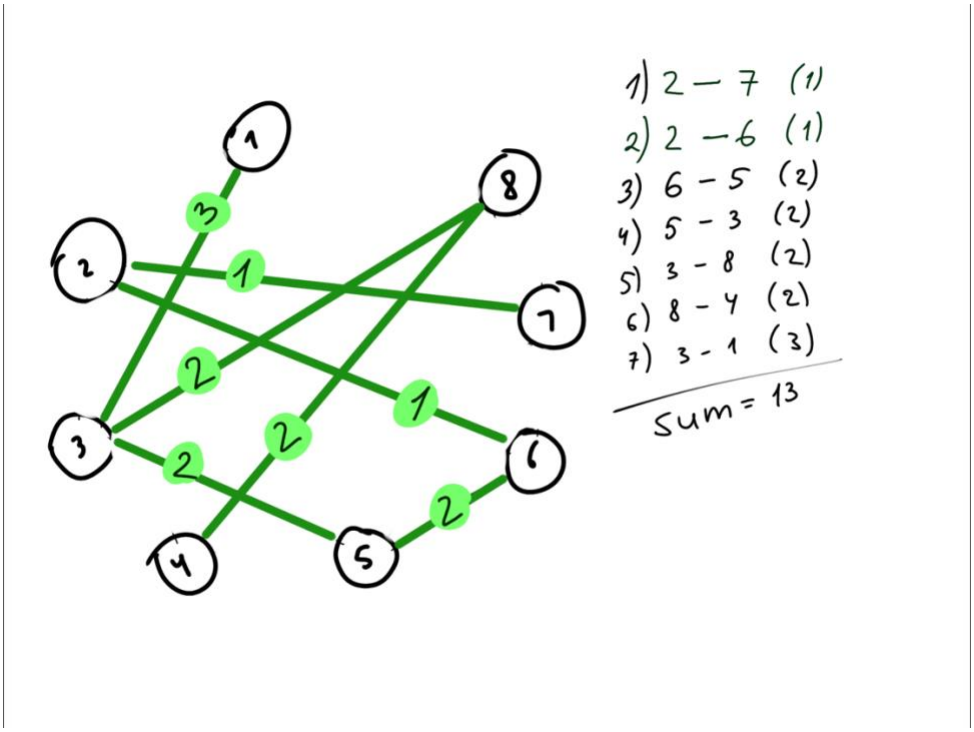

Рис.6 – Мінімальне остове дерево

Код класу PrimaAlgo:

```
public class PrimaAlgo
{
    public int Weight { get; private set; } = 0;
    private int[,] _matrix;
    private int matrixSize;
    private int[,] _octTree;
    private bool[] _addedVertices;

    public int[,] Solve(int[,] matrix)
    {
        _matrix = matrix;
        matrixSize = matrix.GetLength(0);
        _octTree = new int[matrixSize, matrixSize];
        _addedVertices = new bool[matrixSize];

        for (int i = 0; i < matrixSize - 1; i++)
        {
            var edge = GetMinEdge(_addedVertices);
            SetEdge(edge);
            Weight += edge.weight;
        }

        return _octTree;
    }

    private (int ver1, int ver2, int weight) GetMinEdge(bool[] vertices)
    {
        bool firstIteration = vertices.All(ver => !ver);
        (int ver1, int ver2, int weight) edge = (-1, -1, int.MaxValue);
        for (var ver1 = 0; ver1 < vertices.Length; ver1++)
        {
            if (vertices[ver1] || firstIteration)
            {
                for (int ver2 = 0; ver2 < matrixSize; ver2++)
                {
                    if (!vertices[ver2] && _matrix[ver1, ver2] > 0 &&
_matrix[ver1, ver2] < edge.weight)
                    {
                        edge = (ver1, ver2, _matrix[ver1, ver2]);
                    }
                }
            }
        }

        return edge;
    }

    private void SetEdge((int ver1, int ver2, int weight) edge)
    {
        _addedVertices[edge.ver1] = true;
        _addedVertices[edge.ver2] = true;
        _octTree[edge.ver1, edge.ver2] = edge.weight;
        _octTree[edge.ver2, edge.ver1] = edge.weight;
    }
}
```

Результат:

```
PRIMA-ALGO:
0  4  3  0  0  0  0  0
4  0  3  3  0  1  1  4
3  3  0  0  2  3  0  2
0  3  0  0  0  0  3  2
0  0  2  0  0  2  2  0
0  1  3  0  2  0  2  3
0  1  0  3  2  2  0  4
0  4  2  2  0  3  4  0

0  0  3  0  0  0  0  0
0  0  0  0  0  1  1  0
3  0  0  0  2  0  0  2
0  0  0  0  0  0  0  2
0  0  2  0  0  2  0  0
0  1  0  0  2  0  0  0
0  1  0  0  0  0  0  0
0  0  2  2  0  0  0  0

OctTree weight: 13
```

Висновок

В ході виконання лабораторної роботи мною були досліджені такі алгоритми: Карпа-Рабіна, Дейкстри та Пріми, що допомагають у роботі віднайдення під рядочків у тексті, пошуку найкоротших шляхів від певної вершини графу та формуванням мінімального остового дерева відповідно.

Моя реалізація алгоритмів була відтворена програмним забезпеченням на мові C#. Для достовірності у результативності програми була проведена реалізація ручним та програмним способами. Фінальна збіжність результатів свідчить про правильність роботи ПО.