

**Ministerul Educației și Cercetării al Republicii Moldova**  
**Universitatea Tehnică a Moldovei**  
**Facultatea Calculatoare, Informatică și Microelectronică**



# Raport

Lucrarea de laborator nr. 3

Disciplina: **Analiza si proiectarea algoritmilor**

Tema: **Graf de acoperire. Algoritmii Prim si Kruskal.**

A efectuat:

Student grupa TI-231 FR

Apareci Aurica

A verificat:

Asistent universitar

Andrievschi-Bagrin Veronica

**Chișinău 2025**

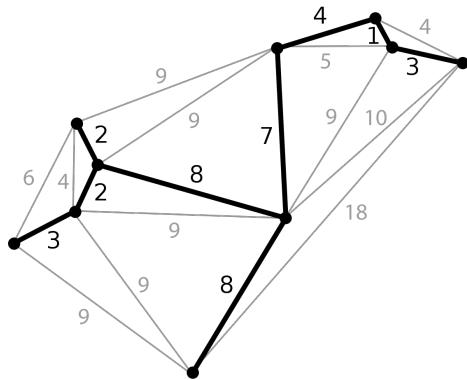
## **Cuprins**

1.Cadru teoretic.....	3
2. Listingul programului.....	4
3. Cazuri de testare.....	8
4. Concluzii.....	12

# 1. Cadru teoretic

**Tema:** Algoritmi lacomi

**Sarcina (conform variantei):** Analiza și implementarea algoritmilor lacomi de căutare a grafului de acoperire de cost minim (Algoritmul lui Prim și algoritmul lui Kruskal)



## 2. Listingul programului

```
//se utilizeaza urmatoarea biblioteca: GitHub - dotnet/BenchmarkDotNet: Powerful .NET library for benchmarking versiunea: 0.13.2

using BenchmarkDotNet.Running;
using Lab_3_utils;

namespace Laboratorul_3
{
    class Program
    {
        static void Main(string[] args)
        {
            //run this in Release mode
            BenchmarkRunner.Run<Benchmark>();
        }
    }
}
```

```
namespace Lab_3_utils
{
    public class Algs
    {
        public static void Kruskal(Graf graph)
        {
            //sort edges by cost
            graph.Muchii.Sort((x, y) => x.Cost.CompareTo(y.Cost));
            //create disjoint sets
            List<DisjointSet> sets = new List<DisjointSet>();
            foreach (int node in graph.Noduri)
            {
                sets.Add(new DisjointSet(node));
            }
            //create MST
            List<Muchie> MST = new List<Muchie>();
            foreach (Muchie edge in graph.Muchii)
            {
                //find sets containing the nodes of the edge
                DisjointSet set1 = null;
                DisjointSet set2 = null;
                foreach (DisjointSet set in sets)
                {
                    if (set.Contains(edge.Start))
                        set1 = set;
                    if (set.Contains(edge.End))
                        set2 = set;
                }
                //if the sets are different, merge them and add the
                //edge to the MST
                if (set1 != set2)
                {
                    set1.Merge(set2);
                    MST.Add(edge);
                }
            }
        }
    }
}
```

```

        sets.Remove(set2);
        MST.Add(edge);
    }
}
//print MST
PrintMST(MST);
}
public static void Prim(Graf graph)
{
    //create a list of edges in the MST
    List<Muchie> MST = new List<Muchie>();
    //create a list of visited nodes
    List<int> visited = new List<int>();
    //add the first node to the visited list
    visited.Add(graph.Noduri[0]);
    //while there are still nodes to visit
    while (visited.Count < graph.Noduri.Count)
    {
        //create a list of edges that connect to visited
nodes
        List<Muchie> edges = new List<Muchie>();
        foreach (var edge in graph.Muchii)
        {
            if (visited.Contains(edge.Start) && !vis-
ited.Contains(edge.End))
            {
                edges.Add(edge);
            }
        }
        //sort the edges by cost
        edges.Sort((x, y) => x.Cost.CompareTo(y.Cost));
        //add the cheapest edge to the MST
        MST.Add(edges[0]);
        //add the new node to the visited list
        visited.Add(edges[0].End);
    }
    PrintMST(MST);
}
private static void PrintMST(List<Muchie> MST)
{
    Console.WriteLine("MST:");
    foreach (var edge in MST)
    {
        Console.WriteLine(edge);
    }
}
}
}

```

```

using BenchmarkDotNet.Attributes;

namespace Lab_3_utils
{
    [MemoryDiagnoser]
    public class Benchmark
    {
        Graf Graph { get; set; }

        [GlobalSetup]
        public void Setup()
        {
            //Graph = GraphGenerator.Defav(500);
            //Graph = GraphGenerator.Fav(500);
            Graph = GraphGenerator.Med(500);
        }

        [Benchmark]
        public void Prims()
        {
            Algs.Prim(Graph);
        }

        [Benchmark]
        public void Kruskal()
        {
            Algs.Kruskal(Graph);
        }
    }
}

```

```

namespace Lab_3_utils
{
    public class DisjointSet
    {
        public List<int> Nodes { get; set; }
        public DisjointSet(int node)
        {
            Nodes = new List<int>();
            Nodes.Add(node);
        }
        public bool Contains(int node)
        {
            return Nodes.Contains(node);
        }
        public void Merge(DisjointSet set)
        {
            Nodes.AddRange(set.Nodes);
        }
    }
}

```

```

namespace Lab_3_utils
{
    public class Graf
    {
        public List<int> Noduri { get; set; }
        public List<Muchie> Muchii { get; set; }
        public Graf()
        {
            Noduri = new List<int>();
            Muchii = new List<Muchie>();
        }
        public void LoadGraph(int[,] arr)
        {
            for (int i = 0; i < arr.GetLength(0); i++)
            {
                for (int j = 0; j < arr.GetLength(1); j++)
                {
                    if (arr[i, j] != 0)
                    {
                        if (!Noduri.Contains(i))
                            Noduri.Add(i);
                        if (!Noduri.Contains(j))
                            Noduri.Add(j);
                        Muchii.Add(new Muchie(i, j, arr[i, j]));
                    }
                }
            }
        }
        public int[,] ToMatrix()
        {
            int[,] arr = new int[Noduri.Count, Noduri.Count];
            foreach (var muchie in Muchii)
            {
                arr[muchie.Start, muchie.End] = muchie.Cost;
            }
            return arr;
        }
        public static int[,] ReadFromFile(string path)
        {
            StreamReader rd = new StreamReader(path);
            string line = rd.ReadLine();
            int n = int.Parse(line.Split(' ')[0]);
            int v = int.Parse(line.Split(' ')[1]);
            int[,] arr = new int[n+1,n+1];
            for (int i = 0; i < v; i++)
            {
                line = rd.ReadLine();
                int start = int.Parse(line.Split(' ')[0]);
                int end = int.Parse(line.Split(' ')[1]);
                int cost = int.Parse(line.Split(' ')[2]);
                arr[start, end] = cost;
            }
            return arr;
        }
    }
}

```

```

namespace Lab_3_utils
{
    public class GraphGenerator
    {
        //mode:
        //fav - 1 - graph with n-1 edges
        //med - 2 - graph with n(n-1)/4
        //defav - 3 - graph with n(n-1)/2
        public static Graf Fav(int nodes)
        {
            //generates a graph with n nodes and n-1 edges of random
cost
            Graf graph = new Graf();
            Random rnd = new Random();
            for (int i = 0; i < nodes; i++)
            {
                graph.Noduri.Add(i);
            }
            for (int i = 0; i < nodes - 1; i++)
            {
                graph.Muchii.Add(new Muchie(i, i + 1, rnd.Next(1,
100)));
            }
            return graph;
        }
        public static Graf Med(int nodes)
        {
            //generates a graph with n nodes and n(n-1)/4 edges of
random cost
            Graf graph = new Graf();
            Random rnd = new Random();
            for (int i = 0; i < nodes; i++)
            {
                graph.Noduri.Add(i);
            }
            for (int i = 0; i < nodes; i++)
            {
                for (int j = i + 1; j < nodes; j++)
                {
                    if (rnd.Next(0, 2) == 1)// 50% chance of adding
an edge
                        graph.Muchii.Add(new Muchie(i, j,
rnd.Next(1, 100)));
                }
            }
            return graph;
        }
        public static Graf Defav(int nodes)
        {
            //generates a graph with n nodes and n(n-1)/2 edges of
random cost
        }
    }
}

```

```

        Graf graph = new Graf();
        Random rnd = new Random();
        for (int i = 0; i < nodes; i++)
        {
            graph.Noduri.Add(i);
        }
        for (int i = 0; i < nodes; i++)
        {
            for (int j = i + 1; j < nodes; j++)
            {
                graph.Muchii.Add(new Muchie(i, j, rnd.Next(1,
100)));
            }
        }
        return graph;
    }
}
}

```

```

namespace Lab_3_utils
{
    public class Muchie:IComparable<Muchie>
    {
        public int Start { get; set; }
        public int End { get; set; }
        public int Cost { get; set; }
        public Muchie(int start, int end, int cost)
        {
            Start = start;
            End = end;
            Cost = cost;
        }
        public override string ToString()
        {
            return $"({Start},{End})\t{Cost}";
        }
        public int CompareTo(Muchie? other)
        {
            if (other == null)
                return 1;
            else
                return this.Cost.CompareTo(other.Cost);
        }
    }
}

```

### 3. Cazuri de testare

N – Numărul de noduri în graf

V – Numărul de muchii din care este alcătuit graful

1 – cazul favorabil – graful generat unde  $V = n - 1$

2 – cazul mediu – graful generat unde  $V = n(n-1)/4$

3 – cazul defavorabil – graful generat unde  $V = n(n-1)/2$

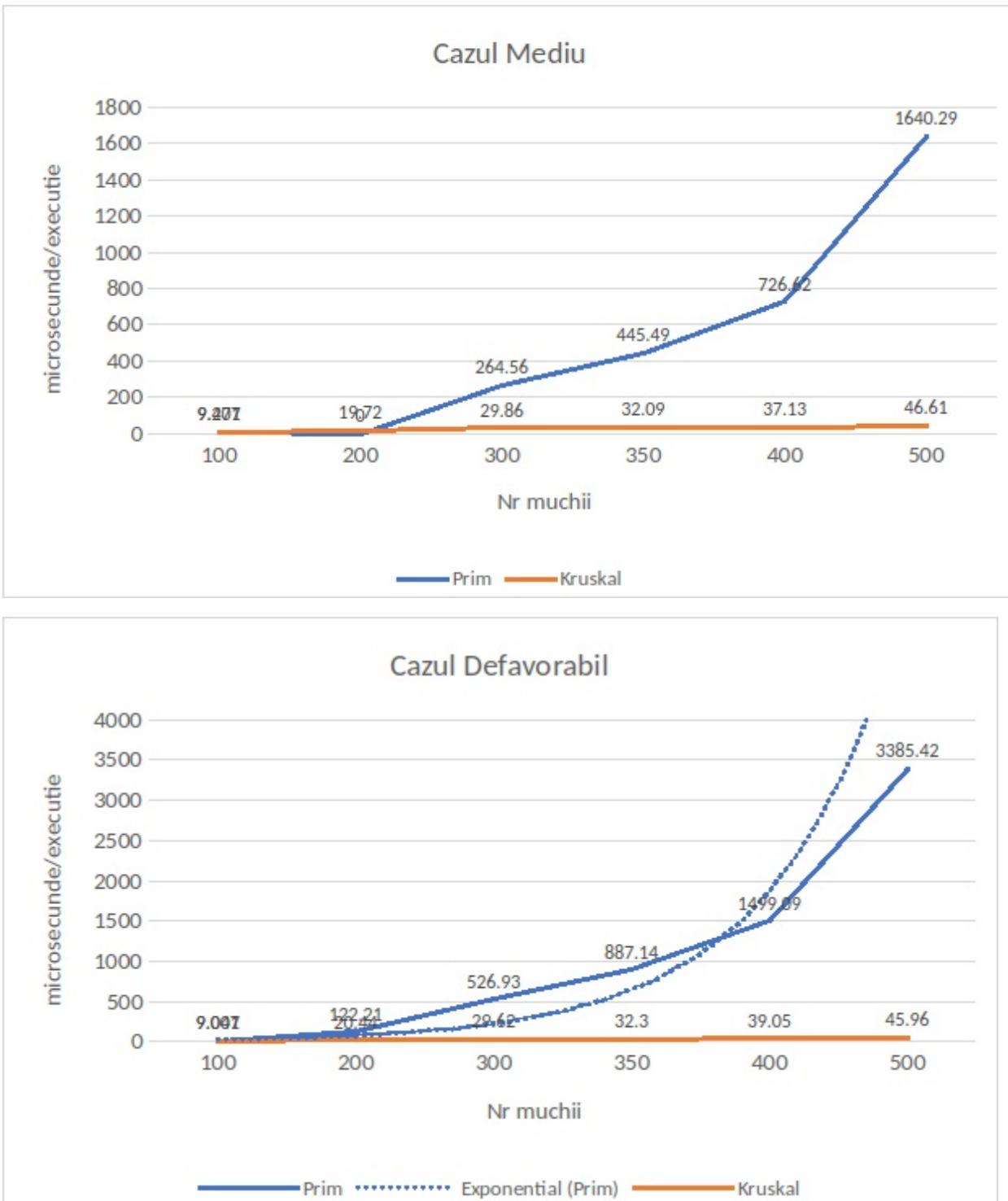
#### Tabelul valorilor obținute

Favorabil						
	100	200	300	350	400	500
Prim	9.007	20.07	29.62	31.43	36.95	45.97
Kruskal	9.041	19.87	29.42	31.8	36.42	45.27

Mediu						
	100	200	300	350	400	500
Prim	9.477	NAN	264.56	445.49	726.62	1,640.29
Kruskal	9.201	19.72	29.86	32.09	37.13	46.61

Defavorabil						
	100	200	300	350	400	500
Prim	9.007	122.21	526.93	887.14	1,499.09	3,385.42
Kruskal	9.041	20.44	29.62	32.3	39.05	45.96

## Analiza valorilor obținute



## **4. Concluzii**

In cadrul acestei lucrări de laborator am efectuat analiza eficientei de determinare a grafului de acoperire de cost minim pentru algoritmul lui Prim si algoritmul lui Kruskal. Aceste doua algoritme prezinta o abordare lacoma asupra generării soluției problemei și urmează sa prezinte un timp de execuție strict dependent de natura setului de date de intrare generat pentru testare. Tind sa menționez ca generatorul de grafuri care urmează sa creeze seturile de intrare trebuie bine formulat, iar algoritmul care sta la baza acestuia trebuie sa asigure condiții egale de generare pentru ambele algoritme testate. Deși in aceasta lucrare am preferat sa fac o analiza asupra timpului de execuție a algoritmului in loc de numărul de iterații/operări efectuate de fiecare implementare a algoritmilor, consider ca m-am descurcat cu proiectarea generatorului de grafuri cat si cu efectuarea unei analize obiective pentru fiecare din algoritmii propuși. Un accent foarte important trebuie pus pe generatorul de grafuri deoarece acesta este cheia către o analiza corecta. In urma analizei câtorva seturi de rezultate dar si a codurilor sursa ce aparțin colegilor de grupa, am fost capabil sa găsesc anumite erori in codul colegilor, dar si sa-mi confirm corectitudinea analizei efectuate pe baza timpului de execuție in comparație cu analiza colegilor efectuata pe numărul de iterații. Tot o data doresc sa menționez neclaritatea condiției acestui laborator si anume metodele de stocare a grafurilor in memoria calculatorului, lucru ce urmează sa influențeze direct la efectuarea analizei obiective. In final consider ca am efectuat o analiza obiectiva a celor doi algoritmi propuși si mi-am făcut concluziile personale ce urmează sa fie aplicate in viitoarele proiecte cu care o sa lucrez.