Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

Звіт

з лабораторної роботи № 3 з дисципліни
«Алгоритми та структури даних 3. Структури даних»

«Прикладні задачі теорії графів ч.1»

Виконав(ла) *ІП-13 Шиманська Ганна Артурівна*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович*
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні прикладні алгоритми на графах та способи їх імплементації.

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм задачі на графах за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування для довільного графа, передбачити введення розмірності графа та введення даних графа вручну чи випадковим чином.

Для самостійно обраного графа (розмірності не менше 9 вершин) розв’язати задану за варіантом задачу вручну.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти програмне та ручне розв’язання задачі.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача**	**Алгоритм**	**Тип графу**	**Спосіб задання графу**
1	Обхід графу	DFS	Неорієнтований	Матриця суміжності
2	Обхід графу	BFS	Неорієнтований	Матриця суміжності
3	Пошук маршруту у графі	Террі	Неорієнтований	Матриця суміжності
4	Пошук відстані між вершинами графа	Хвильовий	Неорієнтований	Матриця суміжності
5	Пошук найкоротшого шляху між парою вершин	Дейкстри	Орієнтований	Матриця вагів
6	Пошук найкоротшого шляху між парою вершин	Беллмана-Форда	Орієнтований	Матриця вагів

7	Побудова мінімальних покриваючих дерев	Прима	Неорієнтований	Матриця вагів
8	Побудова мінімальних покриваючих дерев	Крускала	Неорієнтований	Матриця вагів
9	Побудова мінімальних покриваючих дерев	Борувки	Неорієнтований	Матриця вагів
10	Побудова Ейлерового циклу	За означенням	Неорієнтований	Матриця суміжності
11	Побудова Ейлерового циклу	Флері	Неорієнтований	Матриця суміжності
12	Побудова Гамільтонового циклу	Пошук із поверненнями	Неорієнтований	Матриця суміжності
13	Обхід графу	DFS	Неорієнтований	Матриця інцидентності
14	Обхід графу	BFS	Неорієнтований	Матриця інцидентності
15	Пошук маршруту у графі	Террі	Неорієнтований	Матриця інцидентності
16	Пошук відстані між вершинами графа	Хвильовий	Неорієнтований	Матриця інцидентності
17	Пошук найкоротшого шляху між парою вершин	Дейкстри	Орієнтований	Матриця вагів
18	Пошук	Беллмана-	Орієнтований	Матриця вагів

	найкоротшого шляху між парою вершин	Форда		
19	Побудова мінімальних покриваючих дерев	Прима	Неорієнтований	Матриця вагів
20	Побудова мінімальних покриваючих дерев	Крускала	Неорієнтований	Матриця вагів
21	Побудова мінімальних покриваючих дерев	Борувки	Неорієнтований	Матриця вагів
22	Побудова Ейлерового циклу	За означенням	Неорієнтований	Матриця інцидентності
23	Побудова Ейлерового циклу	Флері	Неорієнтований	Матриця інцидентності
24	Побудова Гамільтонового циклу	Пошук із поверненнями	Неорієнтований	Матриця інцидентності
25	Обхід графу	DFS	Неорієнтований	Матриця суміжності
26	Обхід графу	BFS	Неорієнтований	Матриця суміжності
27	Пошук маршруту у графі	Террі	Неорієнтований	Матриця суміжності
28	Пошук відстані між вершинами графа	Хвильовий	Неорієнтований	Матриця суміжності
29	Пошук найкоротшого	Дейкстри	Орієнтований	Матриця вагів

	шляху між парою вершин			
30	Пошук найкоротшого шляху між парою вершин	Беллмана-Форда	Орієнтований	Матриця вагів
31	Побудова мінімальних покриваючих дерев	Прима	Неорієнтований	Матриця вагів
32	Побудова мінімальних покриваючих дерев	Крускала	Неорієнтований	Матриця вагів
33	Побудова мінімальних покриваючих дерев	Борувки	Неорієнтований	Матриця вагів
34	Побудова Ейлерового циклу	За означенням	Неорієнтований	Матриця суміжності
35	Побудова Ейлерового циклу	Флері	Неорієнтований	Матриця суміжності
36	Побудова Гамільтонового циклу	Пошук із поверненнями	Неорієнтований	Матриця суміжності

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

```
Функція FindRoute(startPointIndex, endPointIndex)
        queue = new()
        Vertices[startPointIndex].MinDistance = 0;
         queue.Push(startPointIndex, 0);
        поки (queue.Count > 0) повторити
                currentVertice = queue.Pop()
                якщо currentVertice == endPointIndex
                то
                        повернути true
                для i = 0; i<Vertices.Length; i++ повторити
                        якщо _distanceMatrix[currentVertice,i]==Int32.MaxValue ||
                        currentVertice==i || Vertices[i].Passed
                        то
                                продовжити
                        все якщо
                        якщо Vertices[i].MinDistance >
                        Vertices[currentVertice].MinDistance + _distanceMatrix[currentVertice,i]
                        то
                        Vertices[i].MinDistance = Vertices[currentVertice].MinDistance +
                        _distanceMatrix[currentVertice,i]
                        Vertices[i].Previous = currentVertice
                        все якщо
                        якщо !Vertices[i].Passed
                        то
                        queue.Push(i, Vertices[i].MinDistance)
                        все якщо
                все повторити
                Vertices[currentVertice].Passed = true
        все поки
повернути false

        Функція TraceRoute(finishIndex)
                route = new()
                current = finishIndex
                поки current > -1
                        route.Push(current)
                        current = Vertices[current].Previous
                все поки
        повернути route
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
namespace Lab3
{
```

```
    public class Node<T>
    {
        public T Value { get; }
        public Node<T>? Next { get; set; }
        public double Criteria { get; }


        public Node(T value, double criteria = 0)
        {
            Value = value;
            Next = null;
            Criteria = criteria;
        }
    }
}
```

```
using System;

namespace Lab3
{
    public class Queue<T>
    {

        public int Count { get; protected set; }
        public Node<T>? Head { get; set; }
        private Node<T>? _tail;

        public Queue()
        {
            Count = 0;
            Head = null;
            _tail = null;
        }

        public virtual void Push(T value)
        {
            Node<T> newNode = new Node<T>(value);
            if (Head == null || _tail == null)
            {
                Head = newNode;
                _tail = newNode;
            }
            else
            {
                _tail.Next = newNode;
                _tail = newNode;
            }
            Count++;
        }

        public T Pop()
        {
            if (Head == null) throw new IndexOutOfRangeException();
            Node<T> headNode = Head;
            Head = Head.Next;
            Count--;
            return headNode.Value;
        }
    }
}
```

```
namespace Lab3
{
```

```
    public class PriorityQueue : Queue<int>
    {
        public void Push(int value, double criteria)
        {
            Node<int> newNode = new Node<int>(value, criteria);
            if (Head == null)
            {
                Head = newNode;
            }
            else
            {
                Node<int> last = Head;
                while (last.Next != null && last.Next.Criteria <= newNode.Criteria)
                {
                    last = last.Next;
                    if (last.Value == newNode.Value) return;
                }
                if (last.Value == newNode.Value) return;
                newNode.Next = last.Next;
                last.Next = newNode;
                last = newNode;
                while (last != null && last.Next is not null)
                {
                    if (last.Next.Value == newNode.Value) last.Next =
last.Next.Next;
                    last = last.Next;
                }

            }

            Count++;
        }
    }
}
```

```
namespace Lab3
{
    public class Stack<T> : Queue<T>
    {
        public Stack() : base() { }

        public override void Push(T value)
        {
            Node<T> newNode = new Node<T>(value);

            newNode.Next = Head;
            Head = newNode;
            Count++;
        }
    }
}
```

```
using System;

namespace Lab3
{
    public class InputOperations
    {
        public static int GetDimensions()
        {
            while (true)
            {
                Console.WriteLine("Enter your graph size");
```

```
                string input = Console.ReadLine();
                if (int.TryParse(input, out int x))
                {
                    return x;
                }
            }
        }

        public static bool IsRandomized()
        {
            Console.WriteLine("Do you want to fill the matrix randomly (r) or
manually (m) ? ");
            while (true)
            {
                string answer = Console.ReadLine();
                if (answer == "r")
                {
                    return true;
                }

                if (answer == "m")
                {
                    return false;
                }

                Console.WriteLine("Incorrect input. Try again...");
            }
        }

        public static int[,] GetManualInput(int n)
        {
            int[,] matrix = new int[n, n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                     matrix[i, j] = int.MaxValue;
                }

                while (true)
                {
                    Console.WriteLine($"Enter adjacent vertice's number to vertice
{i + 1}:");
                    string input = Console.ReadLine();
                    if (string.IsNullOrEmpty(input)) break;
                    if (int.TryParse(input, out int adjacentVertice) &&
adjacentVertice <= n && adjacentVertice > 0)
                    {
                        if (adjacentVertice == i)
                        {
                            Console.WriteLine("Distance must be 0");
                            continue;
                        }
                        while (true)
                        {
                            Console.WriteLine($"Enter your distance between {i+1}
and {adjacentVertice} ");
                            string distanceInput = Console.ReadLine();
                            if (int.TryParse(distanceInput, out int distance))
                            {
                                matrix[i, adjacentVertice-1] = distance;
                                break;
                            }
```

```
                            Console.WriteLine("Incorrect input");
                        }
                    }
                    else Console.WriteLine("Incorrect input");
                }
            }

            return matrix;
        }

        public static (int, int) GetEntryPointsIndexes()
        {
            Console.WriteLine("Enter startpoint and endpoint indexes, separated by
','  : ");
            while (true)
            {
                string input = Console.ReadLine();
                string[] points = input.Split(",");
                if (points.Length == 2 && int.TryParse(points[0], out int
startPoint) && int.TryParse(points[1], out int endPoint))
                {
                    return (startPoint-1, endPoint-1);
                }
                Console.WriteLine("Incorrect input");
            }
        }

    }
}
```

```
using System;

namespace Lab3
{
    public class MatrixRandomizer
    {
        public static int[,] CreateRandomMatrix(int n)
        {
            Random rand = new Random();
            int[,] matrix = new int[n, n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    if (i == j) matrix[i, j] = 0;
                    else matrix[i, j] = rand.Next(2) == 0 ? rand.Next(10, 50) :
Int32.MaxValue;
                }
            }
            return matrix;
        }
    }
}
```

```
namespace Lab3
{
    public class Vertice
    {
        public int MinDistance = Int32.MaxValue;
        public bool Passed = false;
        public int Previous;

        public Vertice()
```

```
        {
            Previous = -1;
        }
    }
}
```

```
using System;


namespace Lab3
{
    public class DijkstrasAlgorithm
    {
        public readonly Vertice[] Vertices;
        private readonly int[,] _distanceMatrix;

        public DijkstrasAlgorithm(int[,] distanceMatrix)
        {
            Vertices = new Vertice[distanceMatrix.GetLength(0)];
            for (int i = 0; i < Vertices.Length; i++)
            {
                Vertices[i] = new();
            }
            _distanceMatrix = distanceMatrix;
        }
        public bool FindRoute(int startPointIndex, int endPointIndex)
        {
            PriorityQueue queue = new();
            Vertices[startPointIndex].MinDistance = 0;
            queue.Push(startPointIndex, 0);
            while (queue.Count > 0)
            {
                int currentVertice = queue.Pop();
                if (currentVertice == endPointIndex)
                {
                    return true;
                }
                for (int i = 0; i<Vertices.Length; i++)
                {
                    if (_distanceMatrix[currentVertice,i]==Int32.MaxValue ||
currentVertice==i || Vertices[i].Passed) continue;
                    if (Vertices[i].MinDistance >
                        Vertices[currentVertice].MinDistance +
_distanceMatrix[currentVertice,i])
                    {
                        Vertices[i].MinDistance =
                            Vertices[currentVertice].MinDistance +
_distanceMatrix[currentVertice,i];
                        Vertices[i].Previous = currentVertice;
                    }
                    if (!Vertices[i].Passed)
                    {
                        queue.Push(i, Vertices[i].MinDistance);
                    }
                }

                Vertices[currentVertice].Passed = true;

            }
            return false;
        }

        public Stack <int> TraceRoute(int finishIndex)
```

```
        {
            Stack <int> route = new();
            int current = finishIndex;
            while (current > -1)
            {
                route.Push(current);
                current = Vertices[current].Previous;
            }

            return route;
        }
    }
}
```

```
using System;

namespace Lab3
{
    public class DisplayOperations
    {
        public static void DisplayMatrix(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] ==
int.MaxValue?"inf":$"{matrix[i,j],-3}") + " ");
                }

                Console.WriteLine();
            }
        }
    }
}
```

```
using System;

namespace Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
           int n = InputOperations.GetDimensions();
           int[,] matrix;
           if (InputOperations.IsRandomized())
           {
               matrix = MatrixRandomizer.CreateRandomMatrix(n);
           }
           else
           {
               matrix = InputOperations.GetManualInput(n);
           }
           DisplayOperations.DisplayMatrix(matrix);
           (int startPoint, int endPoint) = InputOperations.GetEntryPointsIndexes();

           DijkstrasAlgorithm algorithm = new DijkstrasAlgorithm(matrix);
           bool IsFound = algorithm.FindRoute(startPoint, endPoint);
           if (IsFound)
           {
               Stack<int> stack = algorithm.TraceRoute(endPoint);
```

```
                Console.WriteLine("Path found: ");
                Console.WriteLine(GetPath(stack));
                Console.WriteLine("Path length is " +
algorithm.Vertices[endPoint].MinDistance);
            }
            else
            {
                Console.WriteLine("No path found");
            }

        }

        public static string GetPath(Stack<int> stack)
        {
            string path = "" + (stack.Pop()+1);
            while (stack.Count > 0)
            {
                path += " -> " + (stack.Pop()+1);
            }
            return path;
        }
    }
}
```

3.2.2 Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для графів на 7 і 15 вершин відповідно.

```
Enter your graph size
7
Do you want to fill the matrix randomly (r) or manually (m) ?
r
0   28  45  28  inf inf 28
inf 0   inf 30  inf 24  inf
33  39  0   inf 31  inf inf
inf inf 42  0   inf 41  inf
inf 46  inf inf 0   inf 49
35  20  inf inf inf 0   29
46  inf 43  inf 20  inf 0
Enter startpoint and endpoint indexes, separated by ',' :
1,6
Path found:
1 -> 2 -> 6
Path length is 52

Process finished with exit code 0.
```

Рисунок 3.1. Приклад роботи програми для 7 вершин

```
Enter your graph size
15
Do you want to fill the matrix randomly (r) or manually (m) ?
r
0   inf 21  inf 37  inf 28  inf 22  16  16  39  inf inf inf
inf 0   31  inf 34  inf inf 37  inf inf inf 18  inf 34  15
inf 35  0   14  inf 30  inf inf inf inf inf inf 19  inf inf
12  17  49  0   inf 25  49  36  34  15  inf 12  inf 26  18
13  10  inf 48  0   inf 24  12  inf inf inf inf inf 16  inf
15  48  inf inf 42  0   inf 11  inf inf inf inf inf inf 26
inf 29  40  21  inf inf 0   inf 10  inf 39  23  inf 25  39
21  22  inf 31  40  inf 11  0   31  inf inf inf inf 10  inf
17  20  11  inf 22  inf inf 18  0   inf 48  43  12  inf 39
inf inf 18  26  inf inf 23  13  15  0   17  35  inf 10  inf
inf inf inf 48  39  48  inf 27  inf inf 0   43  48  18  inf
inf 40  42  35  inf 42  inf inf 39  31  inf 0   32  inf inf
21  36  inf inf inf 15  inf inf inf inf 48  inf 0   15  inf
32  inf 30  inf inf inf 21  inf 46  14  39  inf 20  0   inf
inf 20  inf inf inf 27  35  inf 34  inf 44  31  inf 39  0
Enter startpoint and endpoint indexes, separated by ',' :
1,14
Path found:
1 -> 10 -> 14
Path length is 26
```

Рисунок 3.2 – Приклад роботи програми для 15 вершин

Розв’язання задачі вручну

На рисунку 3.3 наведено візуалізацію графа для розв’язання задачі вручну.

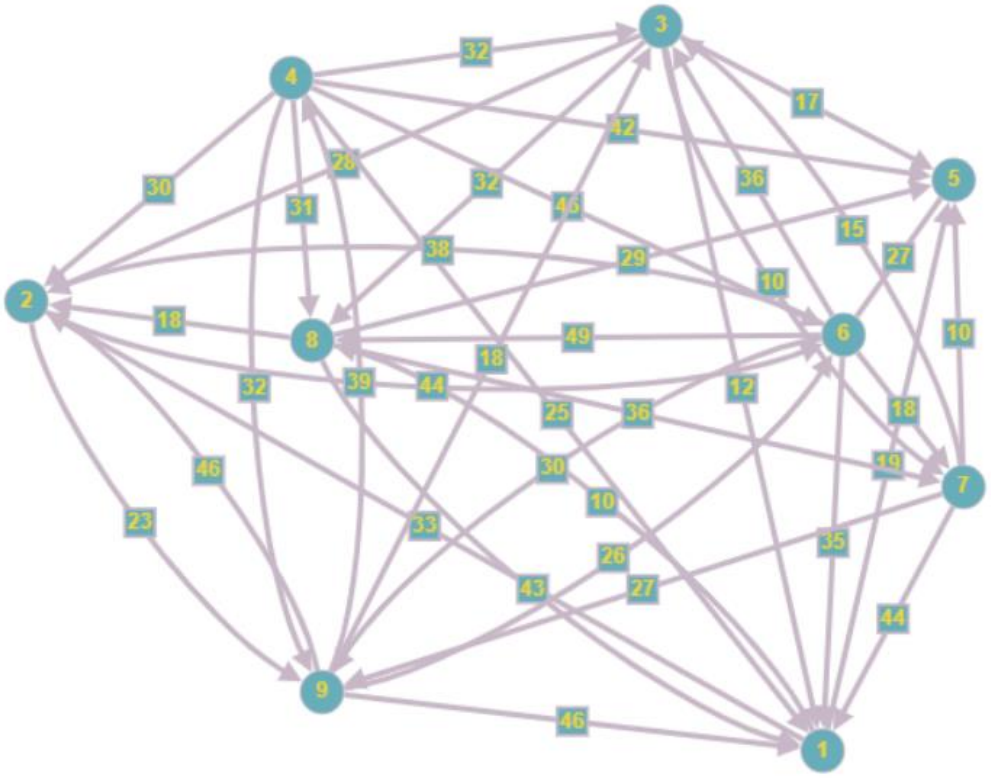

Рисунок 3.3 – Граф на 9 вершин для розв’язку задачі вручну.

Розглянемо початкову матрицю вагів.

0 33 inf inf 19 inf inf 10 inf
inf 0 inf inf inf 44 inf inf 23
12 28 0 inf 17 inf 10 32 inf
25 30 32 0 42 46 inf 31 32
inf inf inf inf 0 27 inf inf inf
35 38 36 inf 27 0 18 49 30
44 inf 15 inf 10 inf 0 inf 27
43 18 inf inf 29 inf 36 0 inf
46 46 18 39 inf 26 inf inf 0

Починаючи зі стартової точки (1), йдучи в кінцеву точку (8), проаналізуємо всі суміжні вершини та шляхи до них.

Точка 1

vertice 2

inf > 33
2.min = 33
2.Previous = 1
vertice 5
inf > 19
5.min = 19
5.Previous = 1
vertice 8
inf > 10
8.min = 10
8.Previous = 1
Стан черги на даний момент (з пріоритетами по відстані):
2(33) -> 8(10) -> 5(19)

Матриця вагів на даний момент:

0 33 inf inf 19 inf inf 10 inf
inf 0 inf inf inf 44 inf inf 23
12 28 0 inf 17 inf 10 32 inf
25 30 32 0 42 46 inf 31 32
inf inf inf inf 0 27 inf inf inf
35 38 36 inf 27 0 18 49 30
44 inf 15 inf 10 inf 0 inf 27
43 18 inf inf 29 inf 36 0 inf
46 46 18 39 inf 26 inf inf 0

Точка 2
vertice 6
inf > 77
6.min = 77
6.Previous = 2
vertice 9
inf > 56
9.min = 56
9.Previous = 2

Стан черги на даний момент:
8(10) -> 5(19) -> 9(56) -> 6(77)
Поточна матриця вагів:

0 33 inf inf 19 inf inf 10 inf
inf 0 inf inf inf 44 inf inf 23
12 28 0 inf 17 inf 10 32 inf
25 30 32 0 42 46 inf 31 32
inf inf inf inf 0 27 inf inf inf
35 38 36 inf 27 0 18 49 30
44 inf 15 inf 10 inf 0 inf 27
43 18 inf inf 29 inf 36 0 inf
46 46 18 39 inf 26 inf inf 0

Точка 8
0 < 53
vertice 2
33 > 28
2.min = 28
2.Previous = 8
19 < 39
vertice 7
inf > 46
7.min = 46
7.Previous = 8

Поточна черга:
5(19) -> 7(46) -> 9(56) -> 6(77)
Поточна матриця вагів:

0 33 inf inf 19 inf inf 10 inf

inf 0 inf inf inf 44 inf inf 23
12 28 0 inf 17 inf 10 32 inf
25 30 32 0 42 46 inf 31 32
inf inf inf inf 0 27 inf inf inf
35 38 36 inf 27 0 18 49 30
44 inf 15 inf 10 inf 0 inf 27
43 18 inf inf 29 inf 36 0 inf
46 46 18 39 inf 26 inf inf 0

Точка 5

vertice 6

77 > 46

6.min = 46

6.Previous = 5

Поточна черга:

7(46) -> 6(46) -> 9(56)

Поточна матриця вагів:

0 33 inf inf 19 inf inf 10 inf
inf 0 inf inf inf 44 inf inf 23
12 28 0 inf 17 inf 10 32 inf
25 30 32 0 42 46 inf 31 32
inf inf inf inf 0 27 inf inf inf
35 38 36 inf 27 0 18 49 30
44 inf 15 inf 10 inf 0 inf 27
43 18 inf inf 29 inf 36 0 inf
46 46 18 39 inf 26 inf inf 0

Точка 7.

7 – кінцева точка. Довжина шляху – 46.

Знайдений шлях:

1 -> 7 -> 8.

ВИСНОВОК

При виконанні даної лабораторної роботи я детальніше ознайомилася з прикладними задачами теорії графів, а особливо з алгоритмом Дейкстри, та придумала спосіб її імплементації. Також я розв'язала поставлену задачу вручну та проаналізувала правильність роботи написаного мною алгоритму.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 15.03.2022 включно максимальний бал дорівнює – 5. Після 15.03.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

– псевдокод алгоритму – 10%;

– програмна реалізація алгоритму – 50%;

– розв’язання задачі вручну – 20%;

– відповідь на 3 теоретичні питання по темі роботи 15%

– висновок – 5%.

НЕ ДІЮТЬ