**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра ІПІ**

**Звіт**

з лабораторної роботи № 3 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

„ **Прикладні задачі з теорії графів**”

**Перевірив**

(прізвище, ім'я, по батькові)

*Соколовський Владислав Володимирович*

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

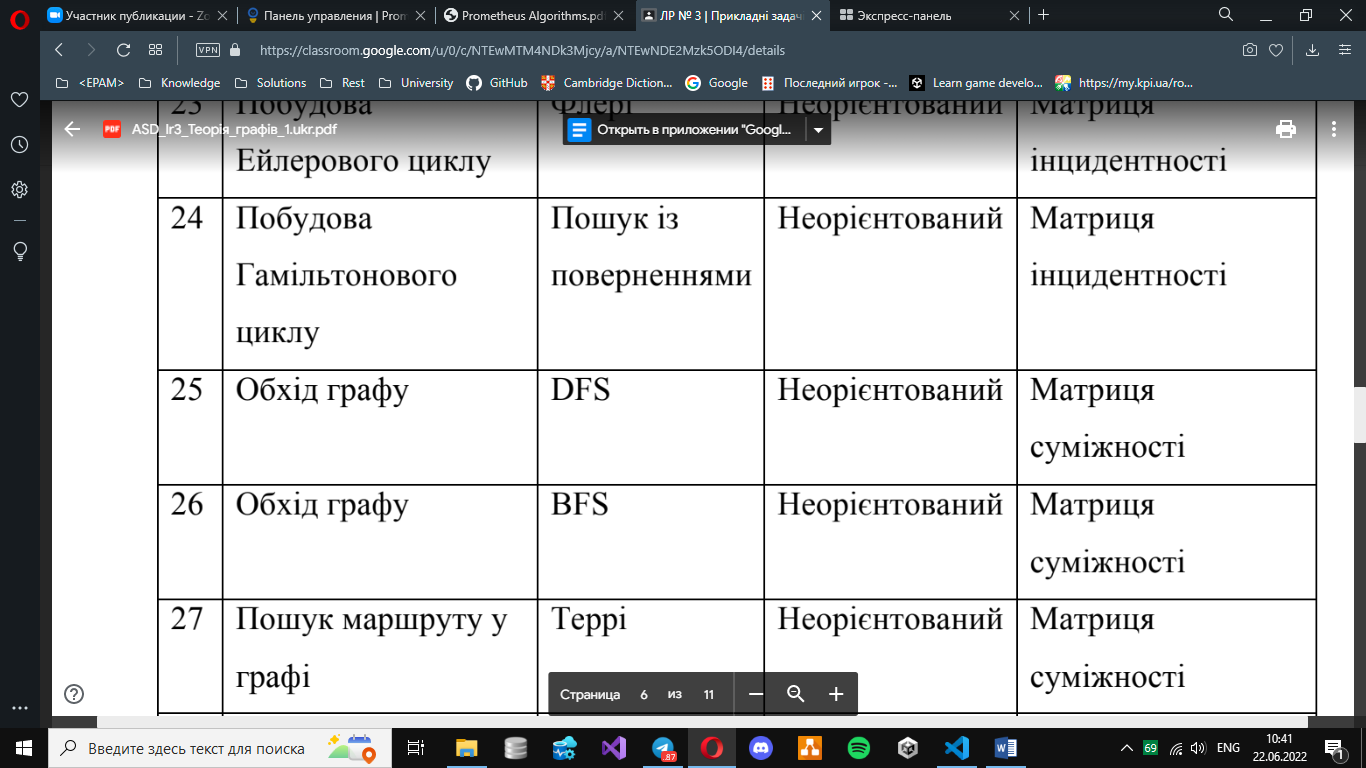
*ІП-15 Плугатирьов Дмитро Валерійович ійович*

Київ 2022

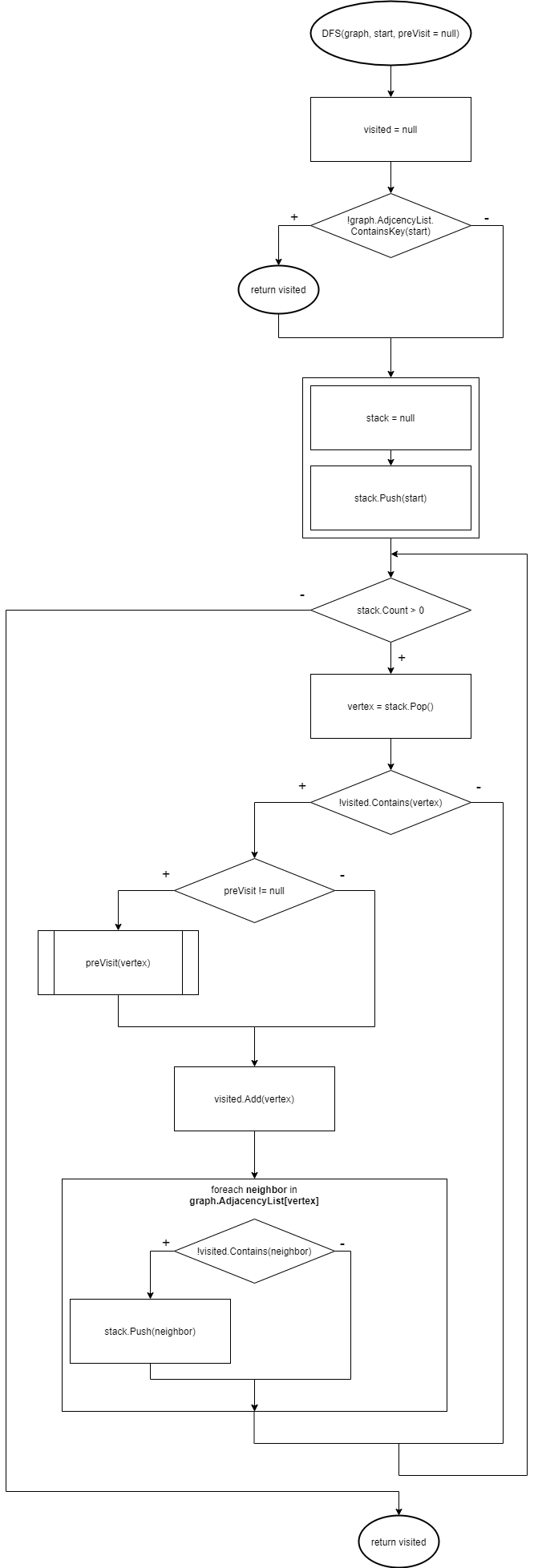
*Мета роботи* – вивчити основні прикладні алгоритми на графах та

способи їх імплементації.

**Варіант 25**



**Блок-схема алгоритму**

****

**Вихідний код**

public class DFSalgorithm

    {

        public static HashSet<T> DFS<T>(Graph<T> graph, T start, Action<T> preVisit = null)

        {

            HashSet<T> visited = new();

            if (!graph.AdjacencyList.ContainsKey(start))

            {

                return visited;

            }

            Stack<T> stack = new();

            stack.Push(start);

            while (stack.Count > 0)

            {

                T vertex = stack.Pop();

                if (!visited.Contains(vertex))

                {

                    if (preVisit != null)

                    {

                        preVisit(vertex);

                    }

                    visited.Add(vertex);

                    foreach(T neighbor in graph.AdjacencyList[vertex])

                    {

                        if (!visited.Contains(neighbor))

                        {

                            stack.Push(neighbor);

                        }

                    }

                }

            }

            return visited;

        }

    }

public class Graph<T>

    {

        public Dictionary<T, HashSet<T>> AdjacencyList { get; } = new Dictionary<T, HashSet<T>>();

        public Graph() {}

        public Graph(IEnumerable<T> vertices, IEnumerable<Tuple<T,T>> edges) {

            foreach(T vertex in vertices)

            {

                AddVertex(vertex);

            }

            foreach(Tuple<T, T> edge in edges)

            {

                AddEdge(edge);

            }

        }

        public void AddVertex(T vertex)

        {

            AdjacencyList[vertex] = new HashSet<T>();

        }

        public void AddEdge(Tuple<T,T> edge)

        {

            if (AdjacencyList.ContainsKey(edge.Item1) && AdjacencyList.ContainsKey(edge.Item2))

            {

                AdjacencyList[edge.Item1].Add(edge.Item2);

                AdjacencyList[edge.Item2].Add(edge.Item1);

            }

        }

    }

public class GraphValidator

    {

        public static void ValidateVertexExistence(List<int> graphVertices, int validatableVertex)

        {

            if (!graphVertices.Contains(validatableVertex))

            {

                throw new ArgumentException($"The vertex {validatableVertex} isn't inside the graph");

            }

        }

        public static void ValidateVertexAbsence(List<int> graphVertices, int validatableVertex)

        {

            if (graphVertices.Contains(validatableVertex))

            {

                throw new ArgumentException($"The vertex {validatableVertex} is inside the graph");

            }

        }

        public static void ValidateVertexExistence(int validatableVertex)

        {

            if (validatableVertex <= 0)

            {

                throw new ArgumentOutOfRangeException(nameof(validatableVertex),

                "the value of vertex should be bigger than 0");

            }

        }

        public static void ValidateEdge(List<Tuple<int, int>> graphEdges, Tuple<int, int> validatableEdge)

        {

            if (graphEdges.Any(tuple => tuple.Item1 == validatableEdge.Item1

                && tuple.Item2 == validatableEdge.Item2)

                || graphEdges.Any(tuple => tuple.Item1 == validatableEdge.Item2

                && tuple.Item2 == validatableEdge.Item1))

            {

                throw new ArgumentException("The edge is already inside the graph");

            }

        }

    }

public static class VertexCapturer

    {

        public static List<int> CaptureVertices()

        {

            System.Console.WriteLine("Enter <auto> to generate vertices automatically"

                + " from 1 up to the chosen number or <man> to enter each of them on you own: ");

            bool inputIsInvalid = true;

            List<int> result = new();

            while (inputIsInvalid)

            {

                inputIsInvalid = false;

                switch (Console.ReadLine().Trim())

                {

                    case "auto":

                        result = CaptureVerticesInRange();

                        break;

                    case "man":

                        result = CaptureCustomVertices();

                        break;

                    default:

                        System.Console.WriteLine("You entered wrong command");

                        System.Console.Write("Try again: ");

                        inputIsInvalid = true;

                        break;

                }

            }

            return result;

        }

        private static List<int> CaptureVerticesInRange()

        {

            List<int> result = new();

            bool exceptionIsCaught = true;

            while (exceptionIsCaught)

            {

                System.Console.Write("Enter the number whom the maximum vertex value should equal: ");

                exceptionIsCaught = false;

                try

                {

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

                    GraphValidator.ValidateVertexExistence(vertex);

                    result = Enumerable.Range(1,vertex).ToList();

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("The entered value isn't a number");

                    System.Console.WriteLine("Try again: ");

                    exceptionIsCaught = true;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    System.Console.WriteLine("Try again: ");

                    exceptionIsCaught = true;

                }

            }

            return result;

        }

        private static List<int> CaptureCustomVertices()

        {

            List<int> result = new();

            bool exceptionIsCaught = true;

            do

            {

                exceptionIsCaught = false;

                try

                {

                    System.Console.Write("Enter the value of vertex whom to add in list: ");

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

                    GraphValidator.ValidateVertexAbsence(result, nodeValue);

                    result.Add(nodeValue);

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("The entered value isn't a number");

                    System.Console.WriteLine("Try again");

                    exceptionIsCaught = true;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    System.Console.WriteLine("Try again");

                    exceptionIsCaught = true;

                }

                if (!exceptionIsCaught)

                {

                    System.Console.WriteLine("Enter <Backspace> to end typing or any key to continue");

                }

            } while (exceptionIsCaught || Console.ReadKey().Key != ConsoleKey.Backspace);

            return result;

        }

        public static int CaptureVertex(List<int> graphVertices)

        {

            System.Console.Write("Enter the value of vertex that equals at least 1: ");

            int vertex = default;

            bool exceptionIsCaught = true;

            while (exceptionIsCaught)

            {

                exceptionIsCaught = false;

                try

                {

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

                    GraphValidator.ValidateVertexExistence(graphVertices,vertex);

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("The entered value isn't a number");

                    System.Console.Write("Try again: ");

                    exceptionIsCaught = true;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    System.Console.Write("Try again: ");

                    exceptionIsCaught = true;

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    System.Console.Write("Try again: ");

                    exceptionIsCaught = true;

                }

            }

            return vertex;

        }

    }

public static class EdgeCapturer

    {

        public static List<Tuple<int, int>> CaptureEdges(List<int> graphVertices)

        {

            List<Tuple<int, int>> result = new();

            bool isCommandCaught = false;

            while (!isCommandCaught)

            {

                isCommandCaught = true;

                System.Console.Write("Enter <auto> to capture auto edges or <man> to do it manually: ");

                switch (Console.ReadLine().Trim())

                {

                    case "auto":

                        result = CaptureRandomEdges(graphVertices);

                        break;

                    case "man":

                        result = CaptureCustomEdges(graphVertices);

                        break;

                    default:

                        System.Console.WriteLine("Unknown command");

                        isCommandCaught = false;

                        break;

                }

            }

            return result;

        }

        private static List<Tuple<int, int>> CaptureRandomEdges(List<int> graphVertices)

        {

            List<Tuple<int, int>> result = new();

            for (var i = 0; i < graphVertices.Count; i++)

            {

                for (var y = i + 1; y < graphVertices.Count && y - i != 3; y++)

                {

                    result.Add(new Tuple<int, int>(graphVertices[i],

                        graphVertices[y]));

                }

            }

            return result;

        }

        private static List<Tuple<int, int>> CaptureCustomEdges(List<int> graphVertices)

        {

            List<Tuple<int, int>> result = new();

            bool exceptionIsCaught = true;

            do

            {

                exceptionIsCaught = false;

                try

                {

                    System.Console.WriteLine("Enter the first vertex: ");

                    int firstVertex = VertexCapturer.CaptureVertex(graphVertices);

                    System.Console.WriteLine("Enter the second vertex: ");

                    int secondVertex = VertexCapturer.CaptureVertex(graphVertices);

                    GraphValidator.ValidateEdge(result,

                        new Tuple<int, int>(firstVertex, secondVertex));

                    result.Add(new Tuple<int, int>(firstVertex,secondVertex));

                    ip15\_pluhatyrov\_03.PrintHorizontalRule();

                }

                catch (ArgumentException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    System.Console.WriteLine("Try again");

                    exceptionIsCaught = true;

                }

                if (!exceptionIsCaught)

                {

                    System.Console.WriteLine("Enter <Backspace> to end typing or any key to continue");

                }

            } while (exceptionIsCaught || Console.ReadKey().Key != ConsoleKey.Backspace);

            return result;

        }

    }

class ip15\_pluhatyrov\_03

    {

        static void Main(string[] args)

        {

            List<int> vertices = VertexCapturer.CaptureVertices();

            PrintVertices(vertices);

            List<Tuple<int, int>> edges = EdgeCapturer.CaptureEdges(vertices);

            Graph<int> graph = new Graph<int>(vertices, edges);

            PrintVertices(vertices);

            System.Console.WriteLine("Select the start vertex: ");

            int startVertex = VertexCapturer.CaptureVertex(vertices);

            Console.WriteLine(string.Join("-->", DFSalgorithm.

            DFS(graph, startVertex)));

        }

        public static void PrintHorizontalRule()

        {

            System.Console.WriteLine(new string('-', 40));

        }

        static void PrintVertices(List<int> vertices)

        {

            System.Console.WriteLine("The current graph has the next vertices:");

            foreach (int vertex in vertices)

            {

                System.Console.Write($"{vertex}  ");

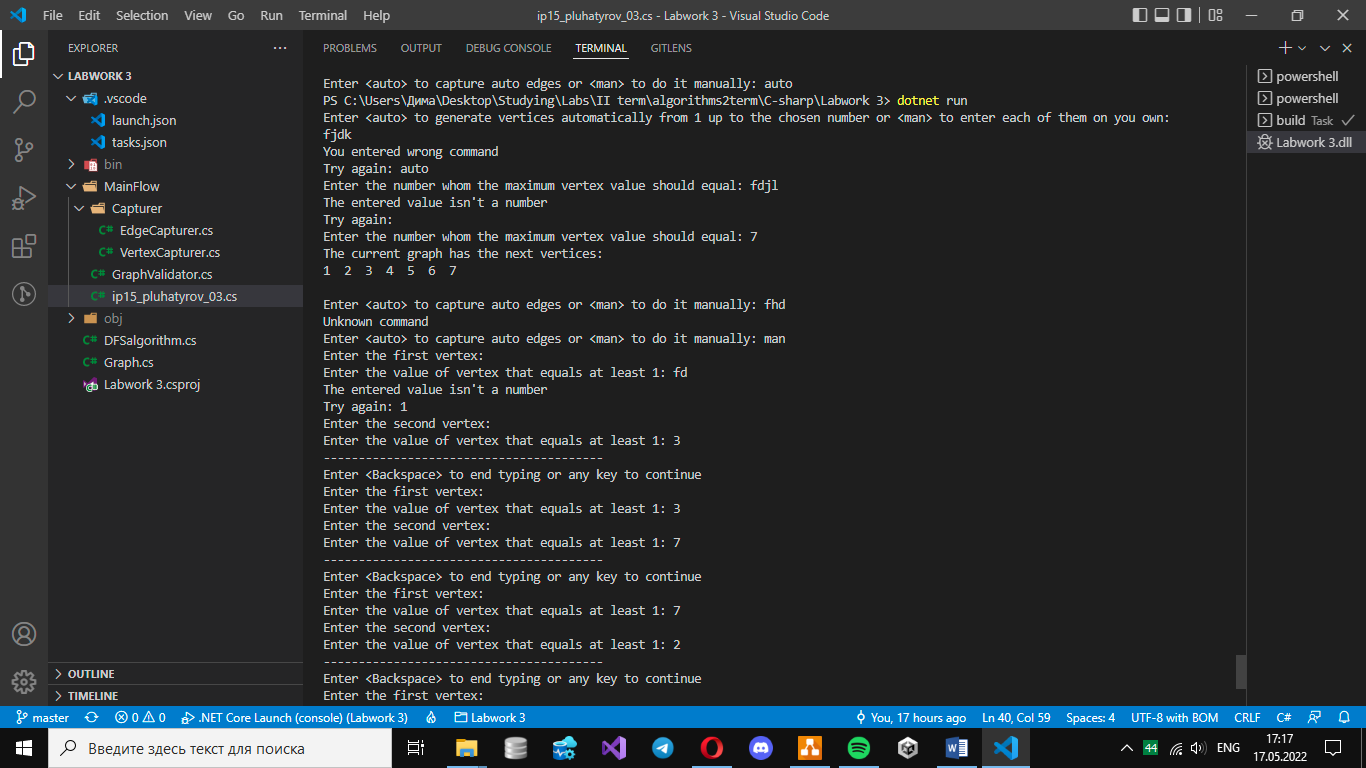
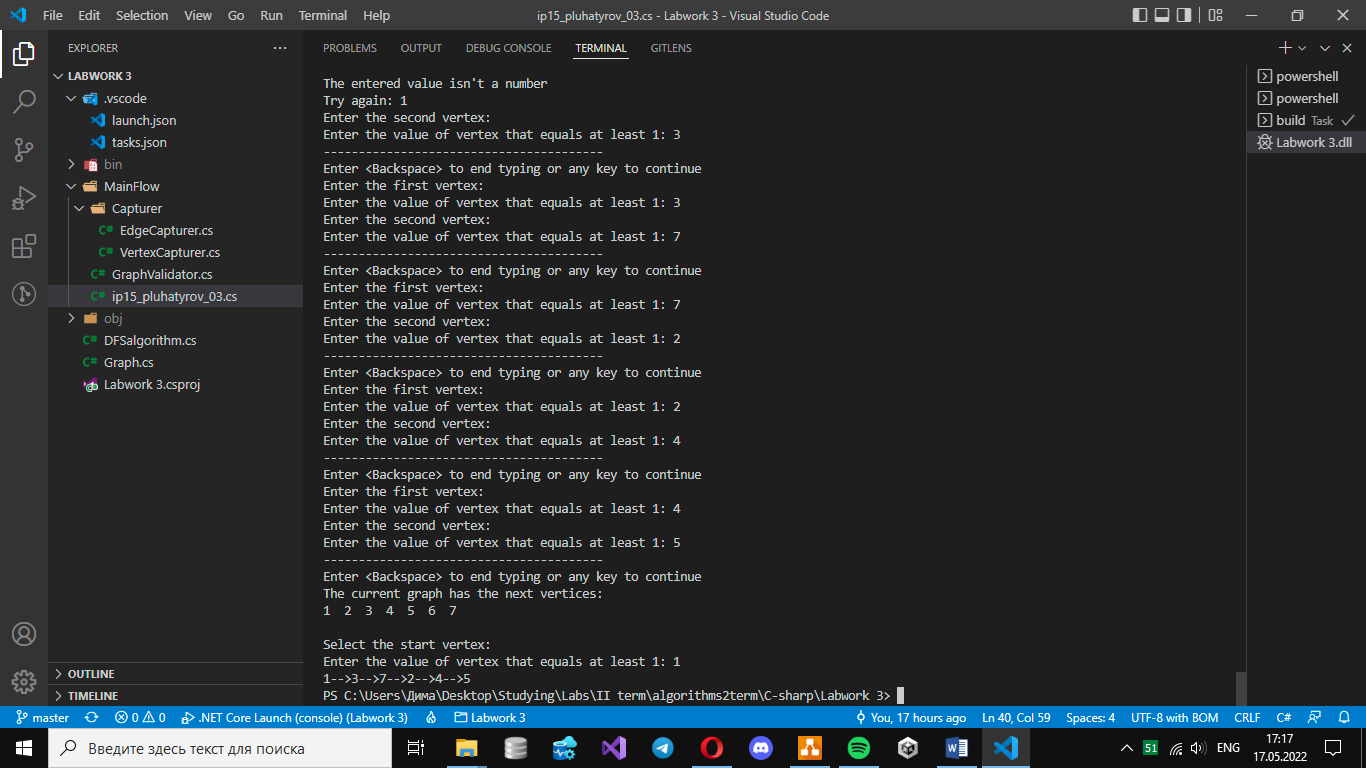
            }

            System.Console.WriteLine(Environment.NewLine);

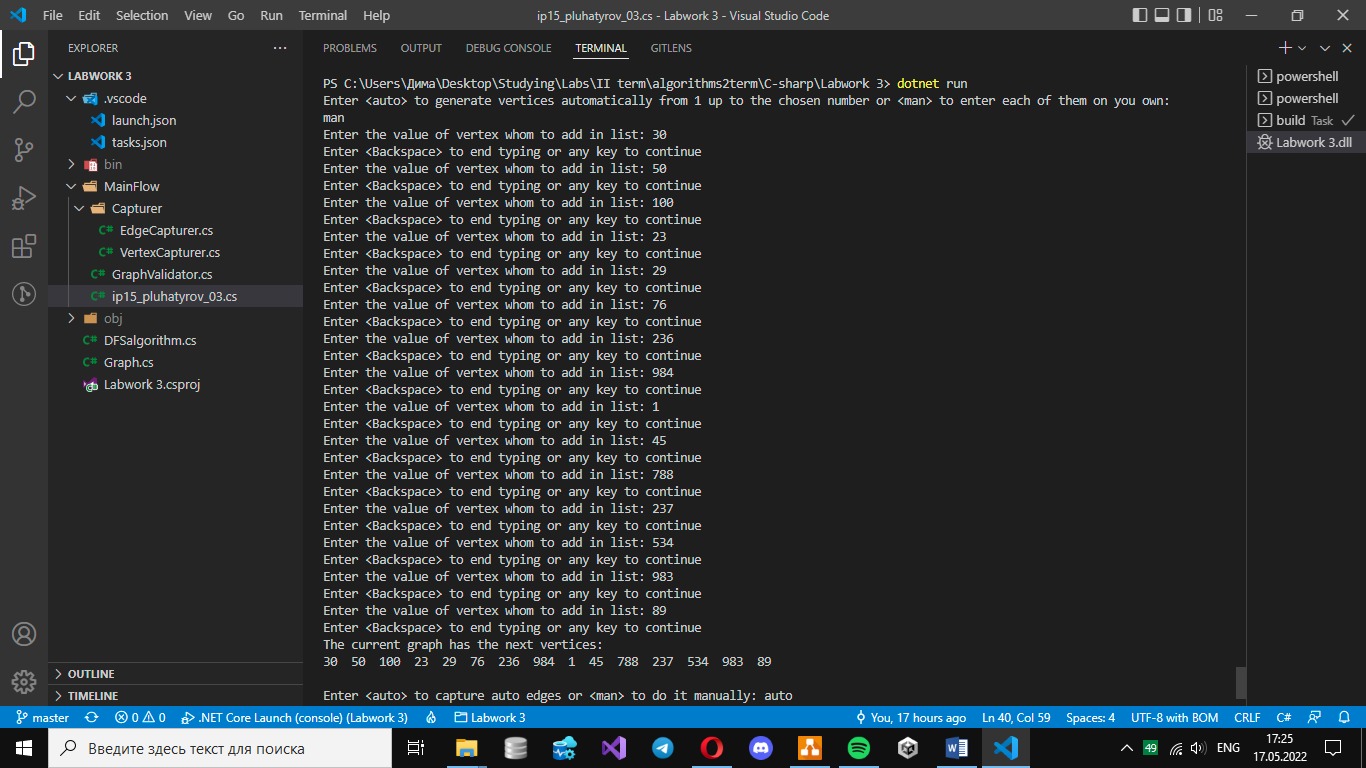
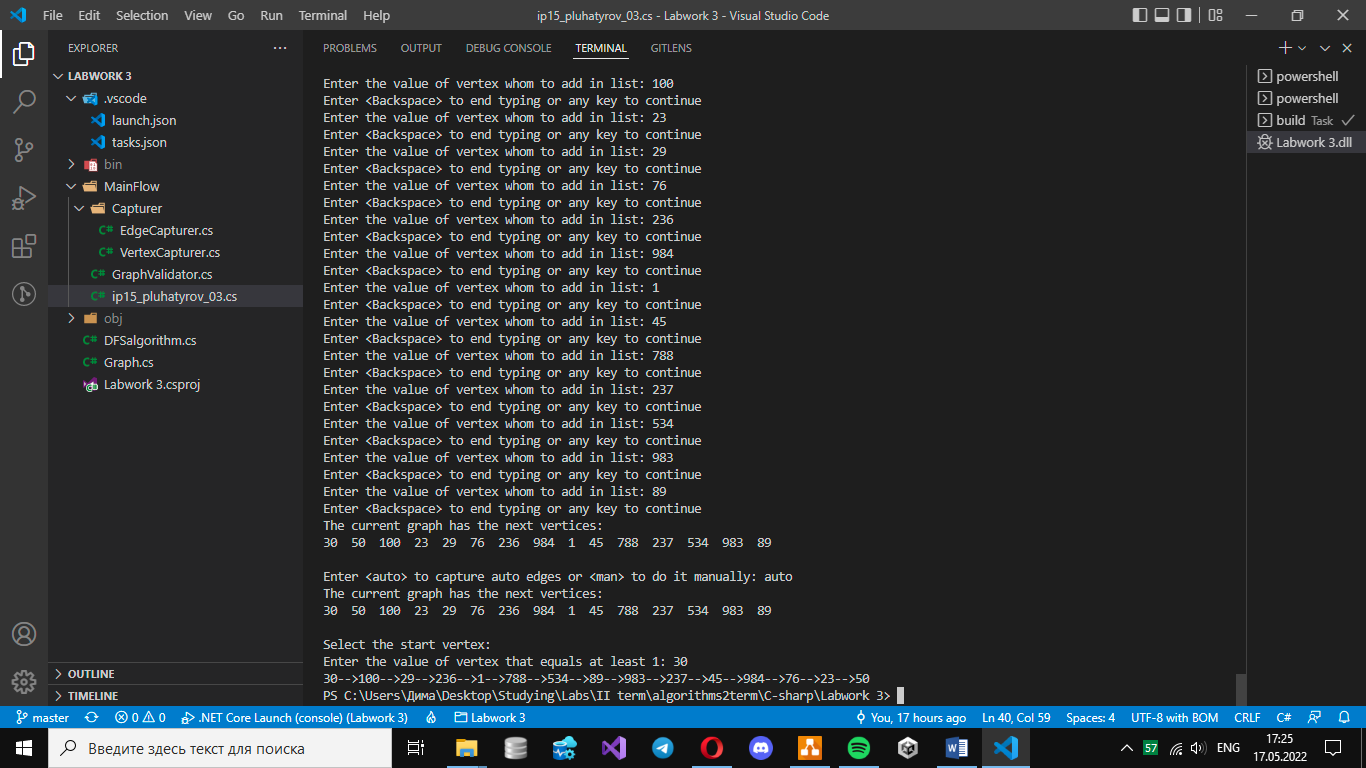
        }

    }

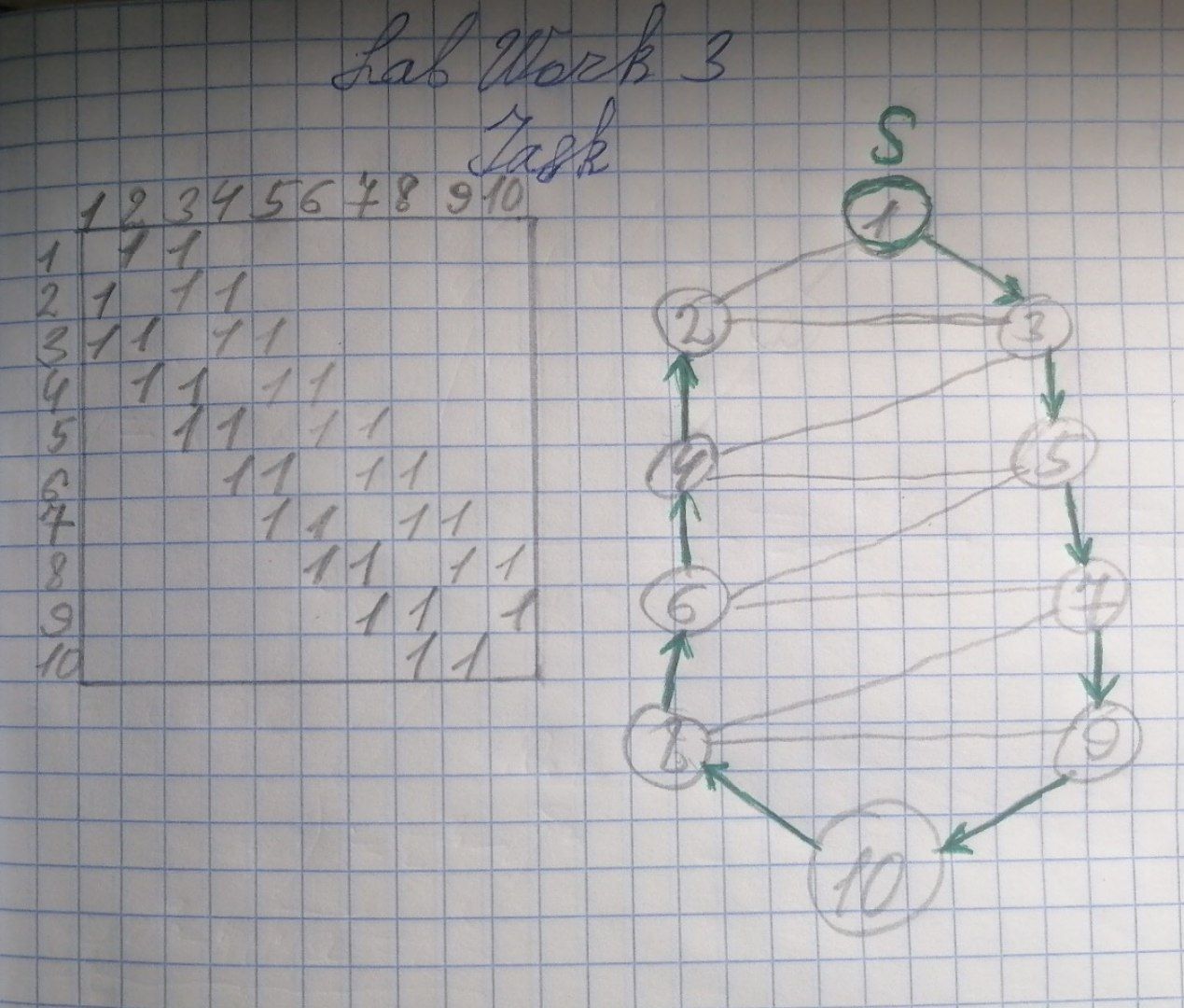
**Приклад роботи графу на 7 вершин**

**Приклад роботи графу на 15 вершин**

**Розв’язання задачі вручну**

****

**Висновок**

На цій лабораторній роботі я реалізував алгоритм обходу графу в глибину (DFS), задаючи вершини графу матрицею суміжості автоматично або з користувацького надання даних. Цей алгоритм особливий тим, що не знаходить найкоротший прохід, а один з можливих. Програмне розв'язання задачі подібне до ручного, але дозволяє проводити розрахунки набагато швидше.