Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Обход графов
Тема

Преподаватель _____ Р. Ю. Царев
подпись, дата | инициалы, фамилия

Студент КИ19-17/1б 031939175 _____ А. Д. Непомнящий
номер группы, зачетной книжки | подпись, дата | инициалы, фамилия

Красноярск 2020

1 Цель работы

Изучение некоторых алгоритмов обхода графов.

2 Задачи

Написать программу, реализующую алгоритм обхода графа в глубину.

Предъявлены следующие требования к выполнению работы.

1. Строгое соответствие программы и результатов ее работы с полученным заданием.

2. Самостоятельные тестирование и отладка программы.

3. Устойчивость работы программы при любых воздействиях, задаваемых пользователем через интерфейс программы.

4. Предоставление демонстрационного примера и исходного текста программы для защиты.

5. Предоставление отчета по практическому заданию, содержащего описание реализованного алгоритма, программы, результатов работы программы (отчет необходимо загрузить на сайт курса).

3 Описание реализованного алгоритма

Реализован алгоритм обхода графа в глубину.

4 Описание программы

Для решения задачи была написана программа на языке C#. Было создано пять классов. CraphVertex – класс, необходимый для хранения информации о вершине графа (имя вершины), GraphEdge – класс, необходимый для хранения информации о ребре графа (вершины, соединяемые ребром, вес), Graph – хранит коллекции ребер, вершин, класс EdgeComparer, реализующий компаратор для вершин графа. В классе Graph описан метод Dfs(), реализующий алгоритм обхода в глубину. Класс Program реализует пользовательский интерфейс для внесения информации о графе и демонстрации обхода.

Листинг 1 – Код в файле GraphVertex.cs

```
namespace Lab6
{
    public class GraphVertex
    {
        public string Name { get; }

        public GraphVertex(string name)
        {
            Name = name;
        }
    }
}
```

Листинг 2 – Код в файле GraphEdge.cs

```
namespace Lab6
{
    public class GraphEdge
    {
        public GraphVertex Tail { get; }

        public GraphVertex Head { get; }

        public int Weight { get; set; }

        public GraphEdge(GraphVertex tail, GraphVertex head)
        {
            Tail = tail;
            Head = head;
            Weight = 0;
        }

        public GraphEdge(GraphVertex tail, GraphVertex head, int weight)
        {
            Tail = tail;
            Head = head;
            Weight = weight;

        }

    }
}
```

Листинг 3 – Код в файле Graph.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.Design;
using System.Linq;

namespace Lab6
{
    public class Graph
    {
        private readonly HashSet<GraphVertex> _vertices;
        private readonly SortedSet<GraphEdge> _edges;

        public Graph()
        {
            _vertices = new HashSet<GraphVertex>();
            _edges = new SortedSet<GraphEdge>(new EdgeComparer());
        }

        public int CountVertices()
        {
            return _vertices.Count;
        }

        public int CountEdges()
        {
            return _edges.Count;
        }

        private class EdgeComparer : IComparer<GraphEdge>
        {
            public int Compare(GraphEdge x, GraphEdge y)
            {
                return (string.Compare(x.Head.Name, y.Head.Name,
StringComparison.Ordinal));
            }
        }

        public bool AddVertex(GraphVertex vertex)
        {
            if (_vertices.Any(x => x.Name == vertex.Name))
            {
```

Продолжение листинга 3

```
                return false;
            }

            _vertices.Add(vertex);
            return true;
        }

        public bool AddEdge(GraphEdge edge)
        {
            if (!(_vertices.Contains(edge.Head) &&
_vertices.Contains(edge.Tail)))
            {
                return false;
            }

            _edges.Add(edge);
            return true;
        }

        public GraphVertex VertexByName(string name)
        {
            return _vertices.FirstOrDefault(x => x.Name == name);
        }

        public void InitDfs(GraphVertex current, Action<GraphVertex> body,
            Dictionary<GraphVertex, bool> compliance)
        {
            foreach (var i in _vertices)
            {
                compliance.Add(i, false);
            }

            Dfs(current, body, compliance);
        }

        private void Dfs(GraphVertex current, Action<GraphVertex> body,
            IDictionary<GraphVertex, bool> compliance)
        {
            compliance[current] = true;
            foreach (var i in _edges.Where(x => x.Tail == current &&
!compliance[x.Head]))
```

Окончание листинга 3

```
            {
                Dfs(i.Head, body, compliance);
            }

            body(current);
        }
    }
}
```

Листинг 4 – Код в файле Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Lab6
{
    class Program
    {
        static void Main(string[] args)
        {
            var graph = new Graph();
            string s;
            byte n;
            do
            {
                Console.WriteLine("Enter number of vertices");
                s = Console.ReadLine();
            } while (!byte.TryParse(s, out n));

            for (var i = 0; i < n; i++)
            {
                graph.AddVertex(new GraphVertex((i + 1).ToString()));
            }

            do
            {
                Console.WriteLine("Enter number of edges");
                s = Console.ReadLine();
            } while (!byte.TryParse(s, out n));

            if (graph.CountVertices() == 0)
            {
```

Продолжение листинга 4

```
            Console.WriteLine("No vertices in graph");
            return;
        }

        for (var i = 0; i < n; i++)
        {
            string from;
            string to;
            Console.WriteLine($"Edge {i + 1} of {n}. Edge from: ");
            from = Console.ReadLine();
            Console.WriteLine($"Edge {i + 1} of {n}. Edge from {from} to:
");
            to = Console.ReadLine();
            var vertexFrom = graph.VertexByName(from);
            var vertexTo = graph.VertexByName(to);
            if (graph.AddEdge(new GraphEdge(vertexFrom, vertexTo)))
            {
                Console.WriteLine($"Added edge from {from} to {to}!");
            }
            else
            {
                Console.WriteLine("Something went wrong!");
                i--;
            }
        }

        do
        {
            Console.WriteLine("Name of start vertex of the DFS");
            s = Console.ReadLine();
        } while (graph.VertexByName(s) == null);

        Console.WriteLine($"Starting DFS from {s}, order of visiting:");
        var compliance = new Dictionary<GraphVertex, bool>();

        graph.InitDfs(graph.VertexByName(s), x => Console.WriteLine(x.Name),
compliance);
        Console.WriteLine("Following vertices were never visited:");
        foreach (var i in compliance.Where(x => x.Value == false))
        {
            Console.WriteLine(i.Key.Name);
```

Окончание листинга 4

```
                }
            }
        }
}
```

5 Результаты работы программы

На рисунке 1 приведен скриншот с результатами работы программы. На рисунке 2 приведен граф, обход в глубину которого был выполнен программой.

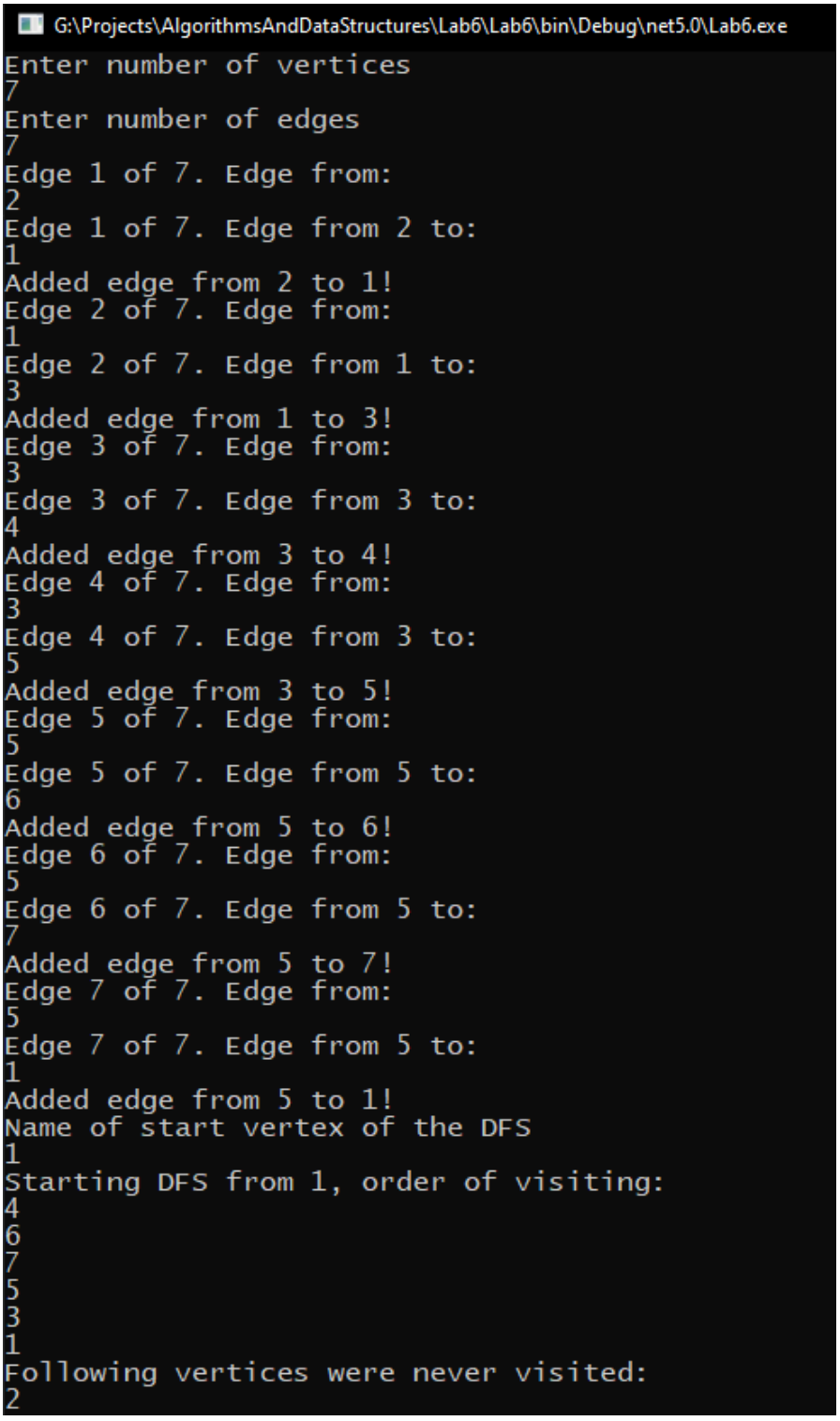

Рисунок 1 – Результаты работы программы

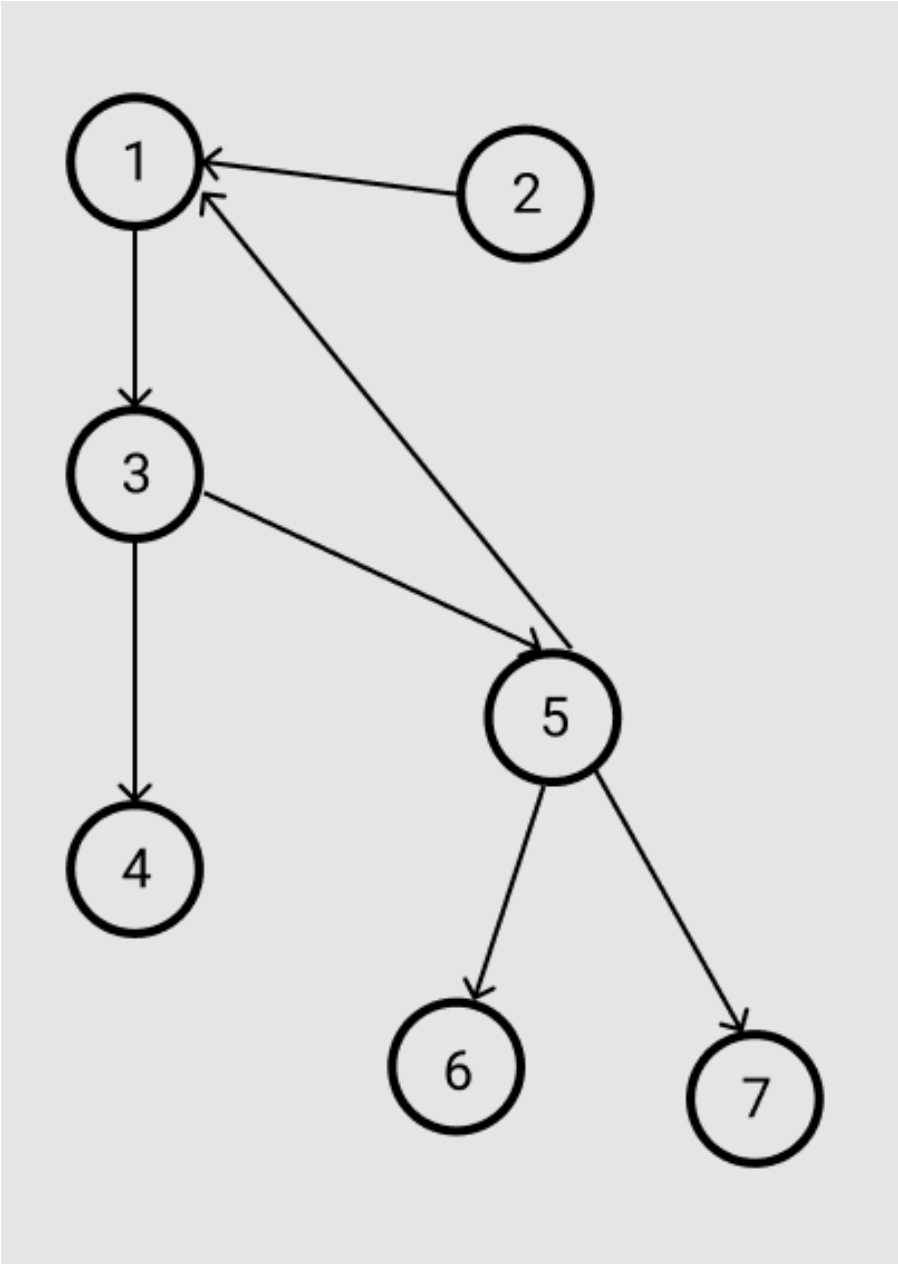

Рисунок 2 – Граф, обход которого был выполнен