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

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Построение остовного дерева минимальной стоимости
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 алгоритма Прима.

2 Задачи

Написать программу, реализующую алгоритм Прима.

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 Прима.

4 Описание программы

Для решения задачи была написана программа на языке C#. Было создано пять классов. GraphVertex – 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 PrimsGraph(), реализующий алгоритм Прима и возвращающий экземпляр класса Graph. Класс Program реализует пользовательский интерфейс для внесения информации о графе и демонстрацию работы алгоритма.

Листинг 1 – Код в файле GraphVertex.cs

```
namespace Lab8
{
    public class GraphVertex
    {
        public string Name { get; }

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

        public override string ToString()
        {
            return "Vertex " + Name + ";";
        }
    }
}
```

Листинг 2 – Код в файле GraphEdge.cs

```
namespace Lab8
{
    public class GraphEdge
    {
        public GraphVertex Tail { get; }

        public GraphVertex Head { get; }

        public double Weight { get; set; }

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

        public override string ToString()
        {
```

Окончание листинга 2

```
            return $"{{{Tail.Name}, {Head.Name}}}, ({Weight});";
        }
    }
}
```

Листинг 3 – Код в файле Graph.cs

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

namespace Lab8
{
    public class Graph
    {
        private readonly SortedSet<GraphEdge> _edges;

        public HashSet<GraphVertex> Vertices { get; }

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

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

        private class EdgeComparer : IComparer<GraphEdge>
        {
            public int Compare(GraphEdge x, GraphEdge y)
            {
                var result = string.Compare(x.Head.Name, y.Head.Name,
StringComparison.Ordinal);
                if (result == 0)
                {
                    result = string.Compare(x.Tail.Name, y.Tail.Name,
StringComparison.Ordinal);
                }
```

Продолжение листинга 3

```
                if (result != 0 && string.Compare(x.Head.Name, y.Tail.Name,
StringComparison.Ordinal) == 0)
                {
                    result = string.Compare(x.Tail.Name, y.Head.Name,
StringComparison.Ordinal);
                }

                if (result == 0)
                {
                    result = x.Weight.CompareTo(y.Weight);
                }

                return result;
            }
        }

        public override string ToString()
        {
            var result = Vertices.Aggregate("GRAPH\nVertices:\n",
                (current, vertex) => current + (vertex.ToString() + ' '));
            result += "\nEdges:\n";
            result = _edges.Aggregate(result, (current, edge) => current + edge
+ "  ");
            return result;
        }

        public bool AddVertex(GraphVertex vertex)
        {
            if (Vertices.Any(x => x.Name == vertex.Name))
            {
                return false;
            }

            Vertices.Add(vertex);
            return true;
        }

        public bool AddEdge(GraphEdge edge)
        {
            if (!(Vertices.Contains(edge.Head) && Vertices.Contains(edge.Tail)))
            {
```

Продолжение листинга 3

```
            return false;
        }

        _edges.Add(edge);
        return true;
    }

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

    public Graph PrimsGraph()
    {
        if (CountVertices() <= 1)
        {
            return this;
        }

        var result = new Graph();

        result.AddVertex(Vertices.First());

        while (result.CountVertices() != CountVertices())
        {
            GraphVertex nextVertex = null;
            GraphEdge nextEdge = null;
            foreach (var vertex in result.Vertices)
            {
                foreach (var edge in _edges.Where(x =>
                    x.Tail == vertex && !result.Vertices.Contains(x.Head) ||
                    x.Head == vertex && !result.Vertices.Contains(x.Tail)))
                {
                    if (nextEdge == null || nextEdge.Weight > edge.Weight)
                    {
                        nextVertex = vertex;
                        nextEdge = edge;
                    }
                }
            }
```

Окончание листинга 3

```
                if (nextVertex == null)
                {
                    throw new Exception("Graph connectivity is in doubt");
                }
                result.Vertices.Add(nextVertex);
                result.Vertices.Add(result.Vertices.Contains(nextEdge.Head) ?
nextEdge.Tail : nextEdge.Head);
                result._edges.Add(nextEdge);
            }

            return result;
        }
    }
}
```

Листинг 4 – Код в файле Program.cs

```
using System;
using System.Drawing;

namespace Lab8
{
    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");
```

Продолжение листинга 4

```
            s = Console.ReadLine();
        } while (!byte.TryParse(s, out n));

        if (graph.CountVertices() == 0)
        {
            Console.WriteLine("No vertices in graph");
            return;
        }

        for (var i = 0; i < n; i++)
        {
            string from;
            string to;
            double weight = -1;
            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();
            do
            {
                Console.WriteLine($"Edge from {from} to {to}. Weight: ");
                s = Console.ReadLine();
            } while (!(double.TryParse(s, out weight) && weight >= 0));

            var vertexFrom = graph.VertexByName(from);
            var vertexTo = graph.VertexByName(to);
            if (graph.AddEdge(new GraphEdge(vertexFrom, vertexTo, weight)))
            {
                Console.WriteLine($"Added edge from {from} to {to}!");
            }
            else
            {
                Console.WriteLine("Something went wrong!");
                i--;
            }
        }

        Console.WriteLine(graph.ToString());

        var ostov = graph.PrimsGraph();
```

Окончание листинга 4

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

5 Результаты работы программы

На рисунке 1 приведен скриншот с результатами работы программы. Ввод данных опущен. На рисунке 2 приведен граф, к которому был применен алгоритм Прима, и выделено его остовное дерево минимальной стоимости.

```
GRAPH
Vertices:
Vertex 1; Vertex 2; Vertex 3; Vertex 4; Vertex 5; Vertex 6; Vertex 7;
Edges:
{2, 1}, (7);  {4, 1}, (5);  {2, 3}, (8);  {4, 2}, (9);  {5, 3}, (5);  {2, 5}, (7);  {4, 5}, (15);  {4, 6}, (
6);  {5, 6}, (8);  {5, 7}, (9);  {6, 7}, (11);
GRAPH
Vertices:
Vertex 1; Vertex 4; Vertex 6; Vertex 2; Vertex 5; Vertex 3; Vertex 7;
Edges:
{2, 1}, (7);  {4, 1}, (5);  {5, 3}, (5);  {2, 5}, (7);  {4, 6}, (6);  {5, 7}, (9);
```

Рисунок 1 – Результаты работы программы

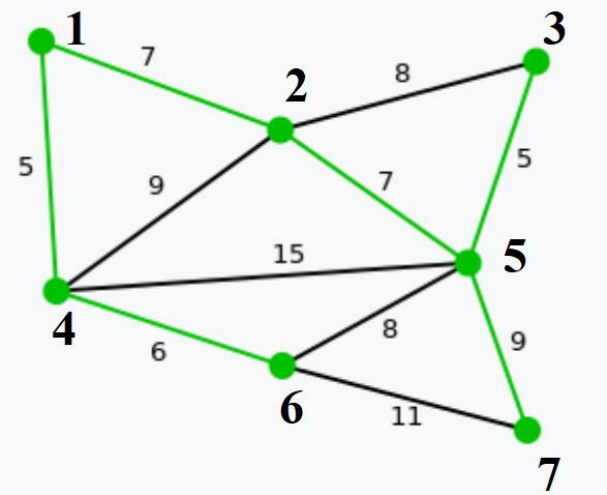

Рисунок 2 – Граф, к которому был применен алгоритм Прима