Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Задание 8 - Построение остовного дерева минимальной стоимости

тема

|  |  |  |
| --- | --- | --- |
| Преподаватель |  | Р.Ю. Царев |
|  | подпись, дата | инициалы, фамилия |
| Студент КИ18-17/1б 031831229 |  | В.А. Прекель |
| номер группы, зачетной книжки | подпись, дата | инициалы, фамилия |

Красноярск 2020

# 1 Цель работы с постановкой задачи

## 1.1 Цель работы

Реализовать один из алгоритмов построения остовного дерева минимальной стоимости неориентированного графа: алгоритм Прима или алгоритм Краскала (по выбору студента).

Программа должна наглядно отображать работы алгоритма.

## 1.2 Задача работы

Требования к выполнению лабораторной работы:

1. Самостоятельные разработка, тестирование и отладка программы.
2. Строгое соответствие программы и результатов ее работы с полученным заданием.
3. Устойчивость работы программы при любых воздействиях, задаваемых пользователем через интерфейс программы.
4. Предоставление демонстрационных примеров и исходного текста программы для защиты.
5. Предоставление отчета по лабораторной работе, содержащего описание реализованного алгоритма, программы, результатов работы программы.

Условия сдачи лабораторной работы:

* Знание теории по сдаваемому алгоритму.
* Умение объяснить полученные результаты.
* Способность быстро продемонстрировать на компьютере владение предметной областью.

# 2 Описание реализованного алгоритма

Был реализован алгоритм Прима.

# 3 Описание программы (листинги кода)

Листинг 1 – Alg\_08/Alg\_08.Core/Prim.cs

using System;  
using System.Collections.Generic;  
using System.Diagnostics;  
using System.Linq;  
  
namespace Alg\_08.Core  
{  
 public class Prim<T>  
 where T : IComparable  
 {  
 public readonly Edges<T> Mst = new Edges<T>();  
  
 public double MstWeight => Mst.Select(e => e.Weight).Sum();  
  
 public Prim(Graph<T> g)  
 {  
 G = g;  
 Q = new SortedDictionary<T, Vertex<T>>(V);  
 }  
  
 public Graph<T> G { get; }  
 private Vertices<T> V => G.V;  
 private Edges<T> E => G.E;  
  
 private SortedDictionary<Vertex<T>, double> d { get; } = new SortedDictionary<Vertex<T>, double>();  
 private SortedDictionary<Vertex<T>, Vertex<T>?> p { get; } = new SortedDictionary<Vertex<T>, Vertex<T>?>();  
  
 private SortedDictionary<T, Vertex<T>> Q { get; }  
 private double w(Vertex<T> i, Vertex<T> j) => E.First(e => e.HasVertex(i) && e.HasVertex(j)).Weight;  
  
 public void Calc()  
 {  
 foreach (var i in V.Select(pair => pair.Value))  
 {  
 d[i] = Double.PositiveInfinity;  
 p[i] = null;  
 }  
  
 d[V.Select(y => y.Value).First()] = 0;  
  
 var v = Q.OrderBy(i => d[i.Value]).First().Value;  
 Q.Remove(v.Value);  
  
 while (Q.Count > 0)  
 {  
 foreach (var u in v.Select(e => e.OtherVertex(v)))  
 {  
 if (u == null)  
 {  
 Debugger.Break();  
 continue;  
 }  
  
 if (Q.ContainsValue(u) && w(v, u) < d[u])  
 {  
 d[u] = w(v, u);  
 p[u] = v;  
 }  
 }  
  
 v = Q.OrderBy(i => d[i.Value]).First().Value;  
 Q.Remove(v.Value);  
 Mst.Add(E.First(e => e.HasVertex(p[v] ?? v) && e.HasVertex(v)));  
 }  
 }  
 }  
}

Листинг 2 – Alg\_08/Alg\_08.Core/Edge.cs

using System;  
  
namespace Alg\_08.Core  
{  
 public class Edge<T> : Tuple<Vertex<T>, Vertex<T>>, IComparable, IEquatable<Edge<T>>  
 where T : IComparable  
 {  
 public Edge(Vertex<T> item1, Vertex<T> item2, double weight) : base(item1, item2) => Weight = weight;  
  
 public double Weight { get; }  
  
 public Vertex<T> LessVertex => Item1.CompareTo(Item2) > 0 ? Item2 : Item1;  
 public Vertex<T> GreatVertex => Item1.CompareTo(Item2) > 0 ? Item1 : Item2;  
  
 public int CompareTo(object obj)  
 {  
 var e = (Edge<T>) obj;  
  
 var c1 = LessVertex.CompareTo(e.LessVertex);  
 var c2 = GreatVertex.CompareTo(e.GreatVertex);  
  
 return c1 == 0 ? c2 : c1;  
 }  
  
 public bool Equals(Edge<T>? other) =>  
 other != null && LessVertex.Equals(other.LessVertex) && GreatVertex.Equals(other.GreatVertex);  
  
 public bool HasVertex(Vertex<T> v) => Item1 == v || Item2 == v;  
  
 public Vertex<T>? OtherVertex(Vertex<T> v) => v == Item1 ? Item2 : v == Item2 ? Item1 : null;  
  
 public override string ToString() => $"({Item1} <-{Weight}-> {Item2})";  
  
 public override bool Equals(object? obj)  
 {  
 if (ReferenceEquals(null, obj))  
 {  
 return false;  
 }  
  
 if (ReferenceEquals(this, obj))  
 {  
 return true;  
 }  
  
 if (obj.GetType() != GetType())  
 {  
 return false;  
 }  
  
 return Equals((Edge<T>) obj);  
 }  
  
 public override int GetHashCode() => base.GetHashCode();  
  
 public static bool operator ==(Edge<T>? left, Edge<T>? right) => Equals(left, right);  
  
 public static bool operator !=(Edge<T>? left, Edge<T>? right) => !Equals(left, right);  
 }  
}

Листинг 3 – Alg\_08/Alg\_08.Core/Edges.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_08.Core  
{  
 public class Edges<T> : SortedSet<Edge<T>>  
 where T : IComparable  
 {  
 public override string ToString() => String.Join("; ", this);  
 }  
}

Листинг 4 – Alg\_08/Alg\_08.Core/Vertex.cs

using System;  
  
namespace Alg\_08.Core  
{  
 public class Vertex<T> : Edges<T>, IComparable, IEquatable<Vertex<T>>  
 where T : IComparable  
 {  
 public Vertex(T value) => Value = value;  
  
 public T Value { get; }  
  
 public int CompareTo(object obj)  
 {  
 var v = (Vertex<T>) obj;  
  
 return Value.CompareTo(v.Value);  
 }  
  
 public bool Equals(Vertex<T>? other)  
 {  
 if (ReferenceEquals(null, other))  
 {  
 return false;  
 }  
  
 if (ReferenceEquals(this, other))  
 {  
 return true;  
 }  
  
 return Value.CompareTo(other.Value) == 0;  
 }  
  
 public override string ToString() => $"{Value}";  
  
 public override bool Equals(object? obj)  
 {  
 if (ReferenceEquals(null, obj))  
 {  
 return false;  
 }  
  
 if (ReferenceEquals(this, obj))  
 {  
 return true;  
 }  
  
 if (obj.GetType() != GetType())  
 {  
 return false;  
 }  
  
 return Equals((Vertex<T>) obj);  
 }  
  
 public override int GetHashCode() => Value.GetHashCode();  
  
 public static bool operator ==(Vertex<T>? left, Vertex<T>? right) => Equals(left, right);  
  
 public static bool operator !=(Vertex<T>? left, Vertex<T>? right) => !Equals(left, right);  
 }  
}

Листинг 5 – Alg\_08/Alg\_08.Core/Vertices.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_08.Core  
{  
 public class Vertices<T> : SortedDictionary<T, Vertex<T>>  
 where T : IComparable  
 {  
 public override string ToString() => String.Join("; ", Values);  
 }  
}

Листинг 6 – Alg\_08/Alg\_08.Core/Graph.cs

using System;  
  
namespace Alg\_08.Core  
{  
 public class Graph<T> : Tuple<Vertices<T>, Edges<T>>  
 where T : IComparable  
 {  
 public Graph() : base(new Vertices<T>(), new Edges<T>())  
 {  
 }  
  
 public Vertices<T> V => Item1;  
 public Edges<T> E => Item2;  
  
 public Vertex<T> AddVertex(T value)  
 {  
 V[value] = new Vertex<T>(value);  
 return V[value];  
 }  
  
 public void RemoveVertex(T value)  
 {  
 E.RemoveWhere(e => e.HasVertex(V[value]));  
 V.Remove(value);  
 }  
  
 public void RemoveVertex(Vertex<T> v)  
 {  
 E.RemoveWhere(e => e.HasVertex(v));  
 V.Remove(v.Value);  
 }  
  
 public void RemoveEdge(Edge<T> e)  
 {  
 V[e.Item1.Value].RemoveWhere(ed => ed.CompareTo(e) == 0);  
 V[e.Item2.Value].RemoveWhere(ed => ed.CompareTo(e) == 0);  
 E.RemoveWhere(ed => ed.CompareTo(e) == 0);  
 }  
  
 public void RemoveEdge(T value1, T value2)  
 {  
 var v1 = V[value1];  
 var v2 = V[value2];  
 v1.RemoveWhere(ed => ed.HasVertex(v1) && ed.HasVertex(v2));  
 v2.RemoveWhere(ed => ed.HasVertex(v1) && ed.HasVertex(v2));  
 E.RemoveWhere(ed => ed.HasVertex(v1) && ed.HasVertex(v2));  
 }  
  
 public Edge<T> AddEdge(T value1, T value2, double weigth) => AddEdge(V[value1], V[value2], weigth);  
  
 public Edge<T> AddEdge(Vertex<T> v1, Vertex<T> v2, double weigth)  
 {  
 var e = new Edge<T>(v1, v2, weigth);  
 E.Add(e);  
 v1.Add(e);  
 v2.Add(e);  
 return e;  
 }  
  
 public override string ToString() => $"V: {String.Join(", ", V.Values)}; E: {String.Join(", ", E)}";  
 }  
}

Листинг 7 – Alg\_08/Alg\_08.Core.Tests/PrimTests.cs

using NUnit.Framework;  
  
namespace Alg\_08.Core.Tests  
{  
 public class PrimTests  
 {  
 [Test]  
 public void TestFromWiki1()  
 {  
 var g = new Graph<int>();  
 g.AddVertex(1);  
 g.AddVertex(2);  
 g.AddVertex(3);  
 g.AddVertex(4);  
 g.AddVertex(5);  
 g.AddVertex(6);  
 g.AddVertex(7);  
 g.AddVertex(8);  
 g.AddVertex(9);  
 g.AddVertex(10);  
 g.AddEdge(1, 2, 9);  
 g.AddEdge(1, 3, 3);  
 g.AddEdge(1, 4, 6);  
 g.AddEdge(2, 3, 9);  
 g.AddEdge(2, 5, 8);  
 g.AddEdge(2, 9, 18);  
 g.AddEdge(3, 4, 4);  
 g.AddEdge(3, 10, 2);  
 g.AddEdge(3, 5, 9);  
 g.AddEdge(4, 10, 2);  
 g.AddEdge(4, 6, 9);  
 g.AddEdge(5, 10, 8);  
 g.AddEdge(5, 6, 7);  
 g.AddEdge(5, 8, 9);  
 g.AddEdge(5, 9, 10);  
 g.AddEdge(6, 10, 9);  
 g.AddEdge(6, 7, 4);  
 g.AddEdge(6, 8, 5);  
 g.AddEdge(7, 8, 1);  
 g.AddEdge(7, 9, 4);  
 g.AddEdge(8, 9, 3);  
  
 var p = new Prim<int>(g);  
 p.Calc();  
   
 Assert.That(p.MstWeight, Is.EqualTo(38));  
 }  
 }  
}

Листинг 8 – Alg\_08/Alg\_08.Console/Program.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
using Alg\_08.Core;  
  
namespace Alg\_08.Console  
{  
 public class Program  
 {  
 public static void Main(string[] args)  
 {  
 System.Console.InputEncoding = Encoding.UTF8;  
 System.Console.OutputEncoding = Encoding.UTF8;  
  
 var g = new Graph<int>();  
  
 while (true)  
 {  
 try  
 {  
 System.Console.WriteLine("Введите номера вершин через пробел: ");  
 var a = System.Console.ReadLine()  
 .Split(new[] {" "}, StringSplitOptions.RemoveEmptyEntries)  
 .Select(Int32.Parse)  
 .Distinct()  
 .Select(g.AddVertex)  
 .ToList();  
  
 break;  
 }  
 catch (Exception e)  
 {  
 System.Console.Error.WriteLine($"Ошибка: {e.Message}\n");  
 }  
 }  
  
 while (true)  
 {  
 try  
 {  
 System.Console.WriteLine(  
 "Введите через Enter 2 номера вершины и вес через пробел, обозначающих ребро: ");  
  
 while (true)  
 {  
 var es = System.Console.ReadLine();  
 if (es == "")  
 {  
 break;  
 }  
  
 var esp = es.Split(new[] {" "}, StringSplitOptions.RemoveEmptyEntries).ToList();  
  
 if (esp.Count == 3)  
 {  
 g.AddEdge(Int32.Parse(esp[0]), Int32.Parse(esp[1]), Double.Parse(esp[2]));  
 }  
 }  
  
 break;  
 }  
 catch (Exception e)  
 {  
 System.Console.Error.WriteLine($"Ошибка: {e.Message}\n");  
 }  
 }  
  
 while (true)  
 {  
 try  
 {  
 System.Console.WriteLine($"Вершины графа: {g.V}");  
 System.Console.WriteLine($"Рёбра графа: {g.E}");  
   
 System.Console.WriteLine();  
   
 var p = new Prim<int>(g);  
 p.Calc();  
   
 System.Console.WriteLine($"Вес минимального остовного дерева: {p.MstWeight}");  
 System.Console.WriteLine($"Рёбра минимального остовного дерева: {String.Join(", ", p.Mst)}");  
   
 break;  
 }  
 catch (Exception e)  
 {  
 System.Console.Error.WriteLine($"Ошибка: {e.Message}\n");  
 }  
 }  
  
 System.Console.ReadKey();  
 }  
 }  
}

# 4 Результаты работы программы

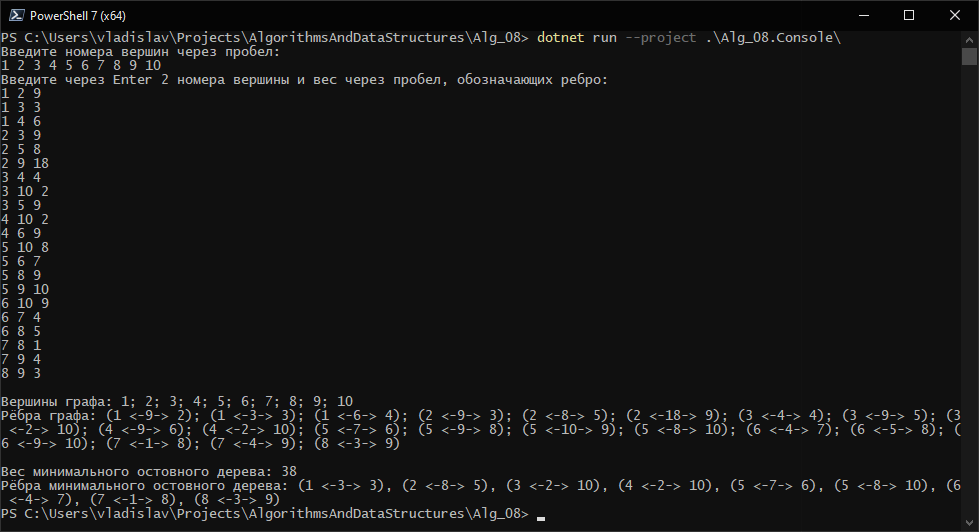


Рисунок 1 – Запуск программы, находящей минимальное остовное дерево для графа из следующего рисунка

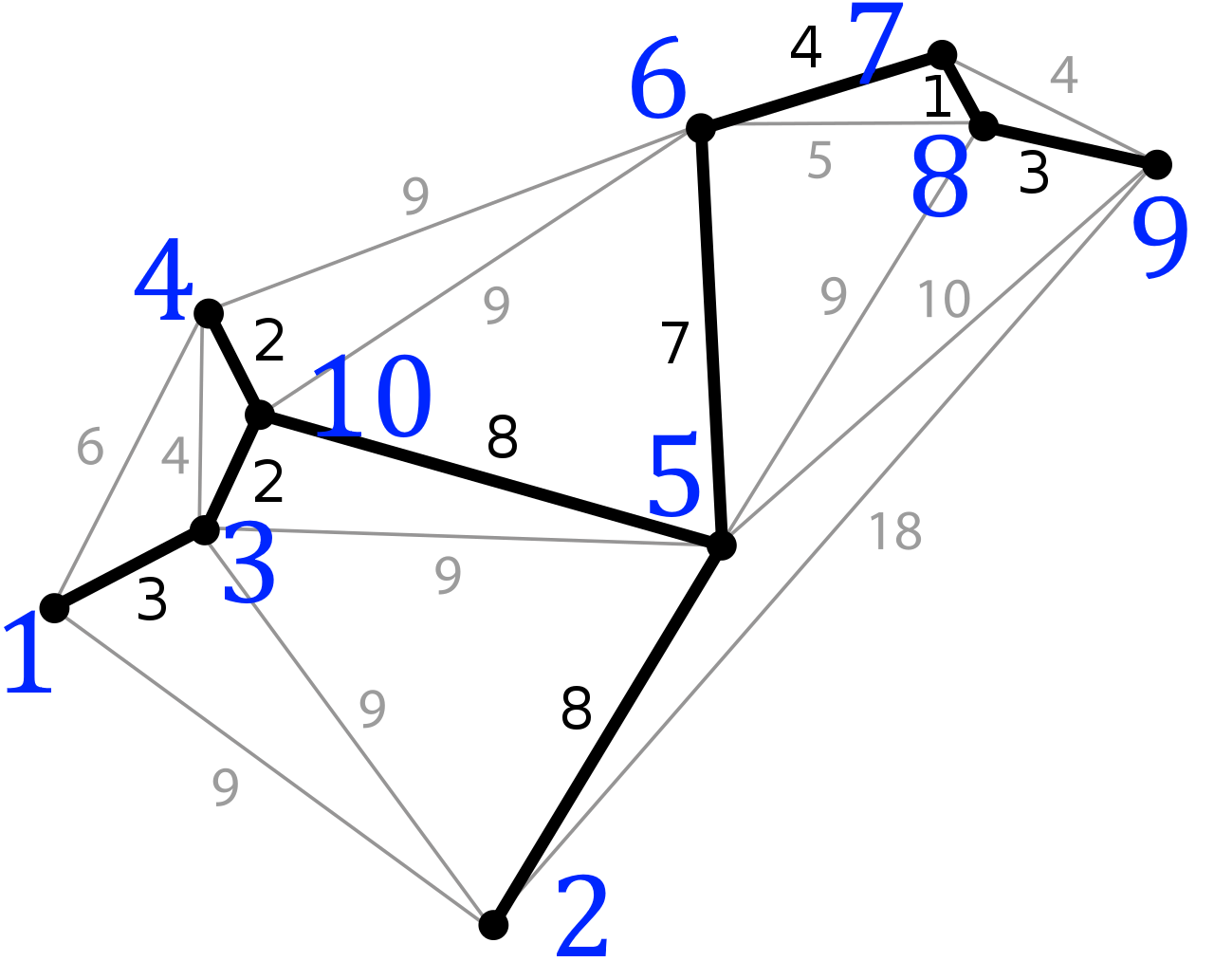


Рисунок 2 – Граф из Википедии [1]

Список использованных источников

1. Минимальное остовное дерево — Википедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Минимальное_остовное_дерево> (Дата обращения: 04.06.2020)