Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Задание 7 - Поиск кратчайшего пути в графе

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 нахождения кратчайшего пути между вершинами графа: алгоритм Дейкстры или алгоритм Флойда (по выбору студента).

Программа должна наглядно отображать результат 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

Реализован алгоритм Дейкстры.

# 3 Описание программы (листинги кода)

Листинг 1 – Alg\_07/Alg\_07.Core/Dijkstra.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace Alg\_07.Core  
{  
 public class Dijkstra<T>  
 where T : IComparable  
 {  
 public Dijkstra(Graph<T> g, Vertex<T> a)  
 {  
 G = g;  
 A = a;  
 }  
  
 public Graph<T> G { get; }  
 public Vertex<T> A { get; }  
 public SortedDictionary<Vertex<T>, double> Distances { get; } = new SortedDictionary<Vertex<T>, double>();  
  
 public SortedDictionary<Vertex<T>, List<Edge<T>>> Paths { get; } =  
 new SortedDictionary<Vertex<T>, List<Edge<T>>>();  
  
 private Vertices<T> V => G.V;  
 private Edges<T> E => G.E;  
 private SortedSet<Vertex<T>> U { get; } = new SortedSet<Vertex<T>>();  
  
 public void Calc()  
 {  
 Distances[A] = 0;  
 Paths[A] = new List<Edge<T>>();  
 foreach (var u in V.Where(y => y.Value != A).Select(y => y.Value))  
 {  
 Distances[u] = Double.PositiveInfinity;  
 }  
  
 while (U.Count < V.Count)  
 {  
 var v = V.Select(y => y.Value)  
 .Except(U)  
 .OrderBy(y => Distances[y])  
 .First();  
 U.Add(v);  
 foreach (var u in V.Select(y => y.Value)  
 .Except(U)  
 .Where(u => v.Contains(E.FirstOrDefault(e => e.Item1 == v && e.Item2 == u))))  
 {  
 var vu = E.First(e => e.Item1 == v && e.Item2 == u);  
 if (Distances[u] > Distances[v] + vu.Weight)  
 {  
 Distances[u] = Distances[v] + vu.Weight;  
 Paths[u] = new List<Edge<T>>(Paths[v]) {vu};  
 }  
 }  
 }  
 }  
 }  
}

Листинг 2 – Alg\_07/Alg\_07.Core/Edge.cs

using System;  
  
namespace Alg\_07.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 int CompareTo(object obj)  
 {  
 var (item1, item2) = (Edge<T>) obj;  
  
 var c1 = Item1.CompareTo(item1);  
 var c2 = Item2.CompareTo(item2);  
  
 return c1 == 0 ? c2 : c1;  
 }  
  
 public bool Equals(Edge<T>? other) =>  
 other != null && Item1.Equals(other.Item1) && Item2.Equals(other.Item2);  
  
 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\_07/Alg\_07.Core/Edges.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_07.Core  
{  
 public class Edges<T> : SortedSet<Edge<T>>  
 where T : IComparable  
 {  
 public override string ToString() => String.Join("; ", this);  
 }  
}

Листинг 4 – Alg\_07/Alg\_07.Core/Vertex.cs

using System;  
  
namespace Alg\_07.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\_07/Alg\_07.Core/Vertices.cs

using System;  
using System.Collections.Generic;  
  
namespace Alg\_07.Core  
{  
 public class Vertices<T> : SortedDictionary<T, Vertex<T>>  
 where T : IComparable  
 {  
 public override string ToString() => String.Join("; ", Values);  
 }  
}

Листинг 6 – Alg\_07/Alg\_07.Core/Graph.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace Alg\_07.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\_07/Alg\_07.Core.Tests/DijkstraTests.cs

using System;  
using System.Linq;  
  
using NUnit.Framework;  
  
namespace Alg\_07.Core.Tests  
{  
 public class DijkstraTests  
 {  
 [Test]  
 public void TestFromWiki()  
 {  
 var g = new Graph<int>();  
 g.AddVertex(1);  
 g.AddVertex(2);  
 g.AddVertex(3);  
 g.AddVertex(4);  
 g.AddVertex(5);  
 g.AddVertex(6);  
 g.AddEdge(1, 2, 7);  
 g.AddEdge(1, 3, 9);  
 g.AddEdge(1, 6, 14);  
 g.AddEdge(2, 3, 10);  
 g.AddEdge(2, 4, 15);  
 g.AddEdge(3, 6, 2);  
 g.AddEdge(3, 4, 11);  
 g.AddEdge(6, 5, 9);  
 g.AddEdge(5, 4, 6);  
 g.AddEdge(2, 1, 7);  
 g.AddEdge(3, 1, 9);  
 g.AddEdge(6, 1, 14);  
 g.AddEdge(3, 2, 10);  
 g.AddEdge(4, 2, 15);  
 g.AddEdge(6, 3, 2);  
 g.AddEdge(4, 3, 11);  
 g.AddEdge(5, 6, 9);  
 g.AddEdge(4, 5, 6);  
  
 var d = new Dijkstra<int>(g, g.V[1]);  
 d.Calc();  
  
 Assert.That(d.Distances.Count, Is.EqualTo(6));  
 Assert.That(d.Distances.Select(p => p.Value), Is.EquivalentTo(new[] {0, 7, 9, 20, 20, 11}));  
 Assert.That(d.Paths[g.V[5]],  
 Is.EquivalentTo(new[]  
 {  
 g.E.First(e => e.Item1.Value == 1 && e.Item2.Value == 3),  
 g.E.First(e => e.Item1.Value == 3 && e.Item2.Value == 6),  
 g.E.First(e => e.Item1.Value == 6 && e.Item2.Value == 5)  
 }));  
 }  
  
 [Test]  
 public void TestFromWikiWithNonConnectivity()  
 {  
 var g = new Graph<int>();  
 g.AddVertex(1);  
 g.AddVertex(2);  
 g.AddVertex(3);  
 g.AddVertex(4);  
 g.AddVertex(5);  
 g.AddVertex(6);  
 g.AddEdge(1, 2, 7);  
 g.AddEdge(1, 3, 9);  
 g.AddEdge(1, 6, 14);  
 g.AddEdge(2, 3, 10);  
 g.AddEdge(2, 4, 15);  
 g.AddEdge(3, 6, 2);  
 g.AddEdge(3, 4, 11);  
 g.AddEdge(6, 5, 9);  
 g.AddEdge(5, 4, 6);  
 g.AddEdge(2, 1, 7);  
 g.AddEdge(3, 1, 9);  
 g.AddEdge(6, 1, 14);  
 g.AddEdge(3, 2, 10);  
 g.AddEdge(4, 2, 15);  
 g.AddEdge(6, 3, 2);  
 g.AddEdge(4, 3, 11);  
 g.AddEdge(5, 6, 9);  
 g.AddEdge(4, 5, 6);  
  
 g.AddVertex(7);  
 g.AddVertex(8);  
 g.AddEdge(7, 8, 1);  
 g.AddEdge(8, 7, 1);  
  
 var d = new Dijkstra<int>(g, g.V[1]);  
 d.Calc();  
  
 Assert.That(d.Distances.Count, Is.EqualTo(8));  
 Assert.That(d.Distances.Select(p => p.Value),  
 Is.EquivalentTo(new[] {0, 7, 9, 20, 20, 11, Double.PositiveInfinity, Double.PositiveInfinity}));  
 Assert.That(d.Paths.ContainsKey(g.V[7]), Is.False);  
 Assert.That(d.Paths.ContainsKey(g.V[8]), Is.False);  
 }  
 }  
}

Листинг 8 – Alg\_07/Alg\_07.Console/Program.cs

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
using Alg\_07.Core;  
  
namespace Alg\_07.Console  
{  
 public class Program  
 {  
 private static string SearchAndOut(Action<int, Action<Vertex<int>>> search, int start, Graph<int> g)  
 {  
 var res = new List<int>();  
 search(start, v => res.Add(v.Value));  
 return String.Join(" ", res);  
 }  
  
 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(  
 "Вводите через Enter 2 номера вершины и вес через пробел, обозначающих дугу: ");  
  
 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("Введите из какой вершины искать пути: ");  
 var a = Int32.Parse(System.Console.ReadLine());  
  
 var d = new Dijkstra<int>(g, g.V[a]);  
 d.Calc();  
  
 System.Console.WriteLine();  
 System.Console.WriteLine("от ->расстояние-> до: список дуг");  
  
 foreach (var (vertex, path) in d.Paths)  
 {  
 System.Console.WriteLine(  
 $"{a} ->{d.Distances[vertex]}-> {vertex.Value}: {String.Join(", ", path)}");  
 }  
  
 break;  
 }  
 catch (Exception e)  
 {  
 System.Console.Error.WriteLine($"Ошибка: {e.Message}\n");  
 }  
 }  
  
 System.Console.ReadKey();  
 }  
 }  
}

# 4 Результаты работы программы

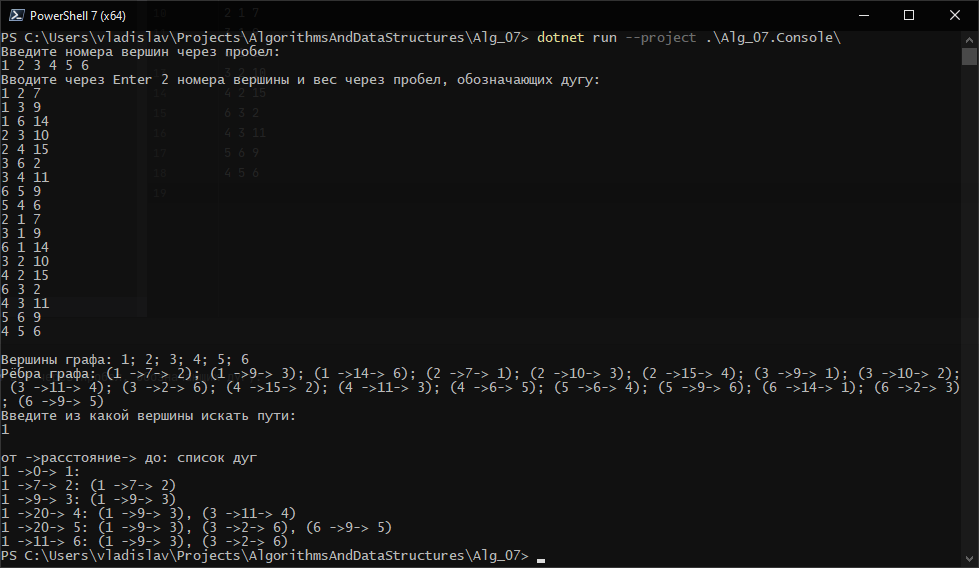


Рисунок 1 – Запуск программы, которая ищет пути из первой вершины в графе из следующего рисунка

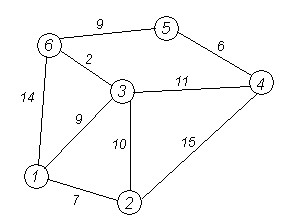


Рисунок 2 – Граф из Википедии [1]

Список использованных источников

1. Алгоритм Дейкстры — Википедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Алгоритм_Дейкстры> (Дата обращения: 04.06.2020)