

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

Институт космических и информационных технологий
институт

Кафедра «Информатика»
кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Поиск кратчайшего пути в графе
Тема

Преподаватель

подпись, дата

Р. Ю. Царев

инициалы, фамилия

Студент

КИ19-17/16 031939175

номер группы, зачетной
книжки

подпись, дата

А. Д. Непомнящий

инициалы, фамилия

Красноярск 2020

1 Цель работы

Изучение некоторых алгоритмов поиска пути в графе.

2 Задачи

Написать программу, реализующую алгоритм Флойда.

Предъявлены следующие требования к выполнению работы.

1. Строгое соответствие программы и результатов ее работы с полученным заданием.
2. Самостоятельное 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 Флойда с запоминанием кратчайших путей.

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 `Floyd()`, реализующий алгоритм Флойда. Класс `Program` реализует пользовательский интерфейс для внесения информации о графе и демонстрацию работы алгоритма.

Листинг 1 – Код в файле GraphVertex.cs

```
namespace Lab7
{
    public class GraphVertex
    {
        public string Name { get; }

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

Листинг 2 – Код в файле GraphEdge.cs

```
namespace Lab7
{
    public class GraphEdge
    {
        public GraphVertex Tail { get; }

        public GraphVertex Head { get; }

        public double Weight { get; set; }

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

        public GraphEdge(GraphVertex tail, GraphVertex head, double 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 Lab7
{
    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;
        }

        public int CountEdges()
        {
            return _edges.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)
        {
            result = x.Weight.CompareTo(y.Weight);
        }

        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)))
    {
        return false;
    }

    _edges.Add(edge);
    return true;
}

private double ShortestEdgeWeight(GraphVertex a, GraphVertex b)
{
    if (!(Vertices.Contains(a) && Vertices.Contains(b)))
    {
        throw new Exception("Wrong vertices");
    }

    if (a == b)
    {
        return 0;
    }
}
```

Продолжение листинга 3

```
    }

    var result = double.PositiveInfinity;
    foreach (var i in _edges.Where(x => x.Tail == a && x.Head == b))
    {
        if (i.Weight < result)
        {
            result = i.Weight;
        }
    }

    return result;
}

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

public (Dictionary<GraphVertex, GraphVertex>, double>,
Dictionary<GraphVertex, GraphVertex>, GraphVertex>)
Floyd()
{
    var floydMatrix = new Dictionary<GraphVertex, GraphVertex>,
double>();
    var routeMatrix = new Dictionary<GraphVertex, GraphVertex>,
GraphVertex>();
    foreach (var i in Vertices)
    {
        foreach (var j in Vertices)
        {
            floydMatrix.Add((i, j), ShortestEdgeWeight(i, j));
            routeMatrix.Add((i, j),
double.IsPositiveInfinity(floydMatrix[(i, j)]) ? null : j);
        }
    }

    foreach (var k in Vertices)
    {
        foreach (var i in Vertices)
        {
```

Окончание листинга 3

```
        foreach (var j in Vertices)
        {
            if (floydMatrix[(i, k)] + floydMatrix[(k, j)] <
floydMatrix[(i, j)])
            {
                floydMatrix[(i, j)] = floydMatrix[(i, k)] +
floydMatrix[(k, j)];
                routeMatrix[(i, j)] = k;
            }
        }
    }

    return (floydMatrix, routeMatrix);
}
}
```

Листинг 4 – Код в файле Program.cs

```
using System;
using System.Linq;

namespace Lab7
{
    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()));
            }
        }
    }
}
```

Продолжение листинга 4

```
do
{
    Console.WriteLine("Enter number of edges");
    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--;
    }
}
```


Окончание листинга 4

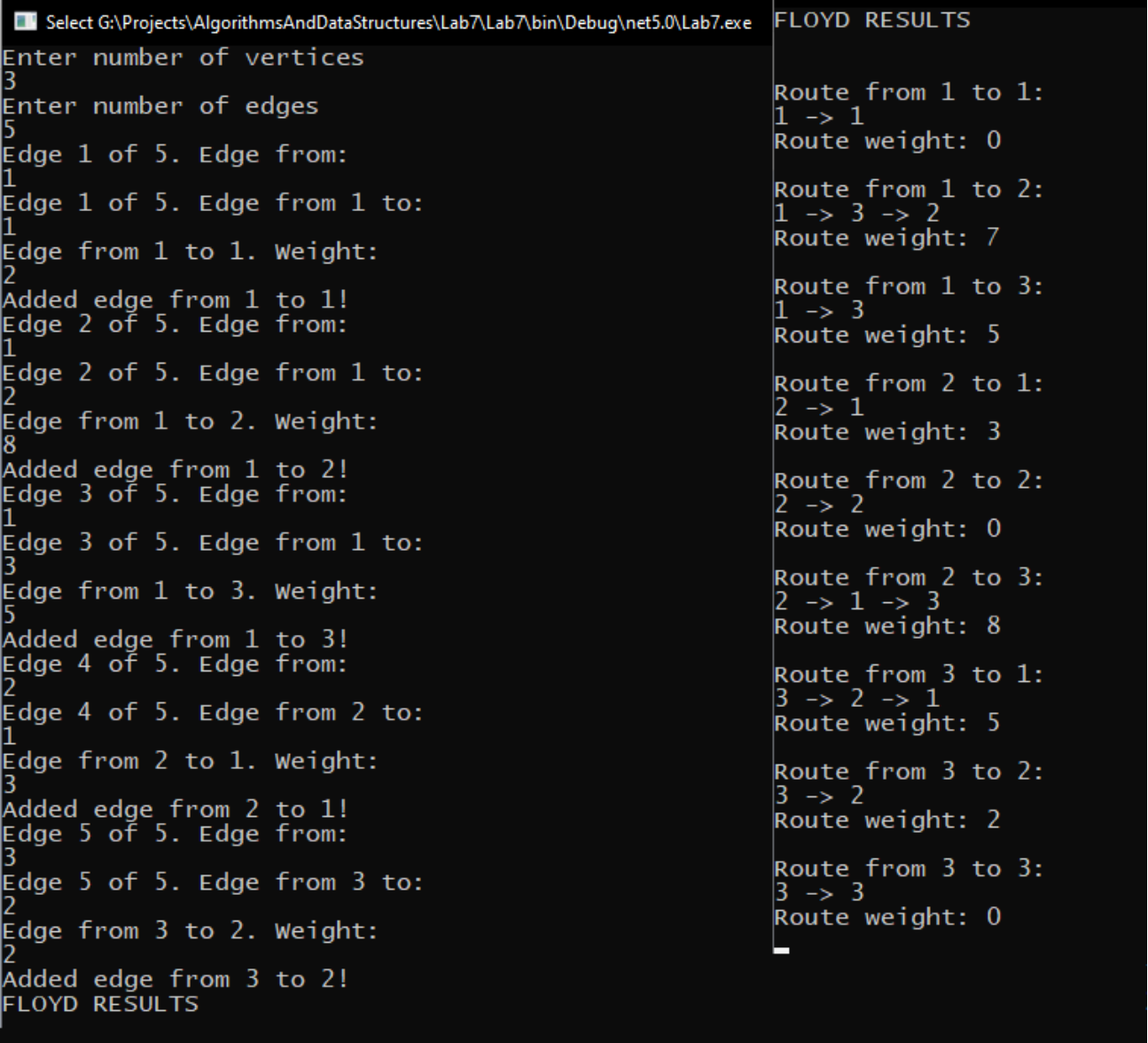
```
var floydResult = graph.Floyd();
Console.WriteLine("FLOYD RESULTS\n");
foreach (var i in graph.Vertices)
{
    foreach (var j in graph.Vertices)
    {
        Console.WriteLine($"Route from {i.Name} to {j.Name}:");
        GraphVertex x = i;
        if (floydResult.Item2[(i, j)] == null)
        {
            Console.WriteLine("No route");
            continue;
        }

        do
        {
            Console.Write($"{x.Name} -> ");
            x = floydResult.Item2[(x, j)];
        } while (x != j);

        Console.WriteLine($"{j.Name}\nRoute weight:
{floydResult.Item1[(i, j)]}");
    }
}
}
```

5 Результаты работы программы

На рисунке 1 приведен скриншот с результатами работы программы. На рисунке 2 приведен граф, к которому был применен алгоритм Флойда.



```
Select G:\Projects\AlgorithmsAndDataStructures\Lab7\Lab7\bin\Debug\net5.0\Lab7.exe
Enter number of vertices
3
Enter number of edges
5
Edge 1 of 5. Edge from:
1
Edge 1 of 5. Edge from 1 to:
1
Edge from 1 to 1. Weight:
2
Added edge from 1 to 1!
Edge 2 of 5. Edge from:
1
Edge 2 of 5. Edge from 1 to:
2
Edge from 1 to 2. Weight:
8
Added edge from 1 to 2!
Edge 3 of 5. Edge from:
1
Edge 3 of 5. Edge from 1 to:
3
Edge from 1 to 3. Weight:
5
Added edge from 1 to 3!
Edge 4 of 5. Edge from:
2
Edge 4 of 5. Edge from 2 to:
1
Edge from 2 to 1. Weight:
3
Added edge from 2 to 1!
Edge 5 of 5. Edge from:
3
Edge 5 of 5. Edge from 3 to:
2
Edge from 3 to 2. Weight:
2
Added edge from 3 to 2!
FLOYD RESULTS

FLOYD RESULTS

Route from 1 to 1:
1 -> 1
Route weight: 0

Route from 1 to 2:
1 -> 3 -> 2
Route weight: 7

Route from 1 to 3:
1 -> 3
Route weight: 5

Route from 2 to 1:
2 -> 1
Route weight: 3

Route from 2 to 2:
2 -> 2
Route weight: 0

Route from 2 to 3:
2 -> 1 -> 3
Route weight: 8

Route from 3 to 1:
3 -> 2 -> 1
Route weight: 5

Route from 3 to 2:
3 -> 2
Route weight: 2

Route from 3 to 3:
3 -> 3
Route weight: 0
```

Рисунок 1 – Результаты работы программы

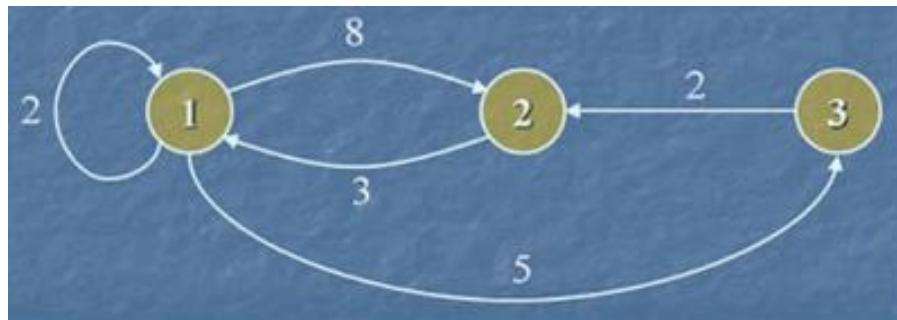


Рисунок 2 – Граф, к которому был применен алгоритм Флойда