Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР

Кафедра ПИн

ЛАБОРАТОРНАЯ РАБОТА №5

По Дискретной математике

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИн - 121
(группа)

Ермилов М.В.
(фамилия, инициалы)

(подпись) (дата)

Муром 2022

Лабораторная работа №5

Тема: ПОИСК КРАТЧАЙШХ ПУТЕМ МЕТОДОМ ФЛОЙДА

Цель работы: Изучить алгоритм поиска кратчайших путей в графе используя алгоритм Флойда.

Порядок выполнения работы

1. Составить программу, осуществляющую ввод матрицы смежности.
2. Используя алгоритм Флойда найти кратчайшие пути в заданном графе

Код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace lab3dis
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace lab3dis
{
    public partial class Form1 : Form
    {
        private Graph g;
        private int from { get => edges(TextBoxFrom); }
        private int to { get => edges(TextBoxTo); }
        public Form1()
        {
            g = new Graph();
            InitializeComponent();
        }
        private int edges(TextBox textBox)
        {
            int r = 0;
            int.TryParse(textBox.Text, out r);
            if (r < 0)
                r = 0;
            if (r >= g.numEdges)
                r = g.numEdges - 1;
            return r;
        }
        private void ButtonFile_Click(object sender, EventArgs e)
        {
            OpenFileDialog OPF = new OpenFileDialog();
            OPF.Filter = "Файлы json|*.json";
            if (OPF.ShowDialog() == DialogResult.OK)
            {
                g = new Graph(OPF.FileName);
            }
        }
```

```
        private void ButtonTracerDijkstraAlgm_Click(object sender, EventArgs e) =>
LabelTracer.Text = $"Путь из {from} в {to}: {g.DijkstraAlgm(from, to)}";

        private void ButtonTracerBFS_Click(object sender, EventArgs e) =>
LabelTracer.Text = $"Путь из {from} в {to}: {g.BFS(from, to)}";

        private void ButtonFloyd_Click(object sender, EventArgs e) =>
MessageBox.Show(g.Floyd(), "Метод флойда"); //LabelTracer.Text = g.Floyd();
    }
}
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3dis
{
    class Graph
    {
        private int[,] adjacency;
        public int numEdges;
        public Graph(string file)
        {
            StreamReader r = new StreamReader(file);
            string strjson = r.ReadToEnd();
            JObject json = JObject.Parse(strjson);
            numEdges = (int)json["Edges"];
            adjacency = new int[numEdges, numEdges];
            for (int i = 0; i < numEdges; i++)
                for (int j = 0; j < numEdges; j++)
                    adjacency[i, j] = (int)json["Adjacency"][i][j];
        }
        public Graph() => numEdges = 0;
        public string BFS(int from, int to)
        {
            if (numEdges <= 0)
                return "нет пути";
            int[] mark = new int[numEdges];
            int[] parent = new int[numEdges];
            for (int i = 0; i < numEdges; i++)
            {
                mark[i] = 0;
                parent[i] = from;
            }
            Queue<int> Q = new Queue<int>();
            int v = from;
            mark[v] = 1;
            Q.Enqueue(v);
            while (Q.Count != 0)
            {
                v = Q.Dequeue();
                for (int i = 0; i < numEdges; i++)
                    if ((adjacency[v, i] != 0) && (mark[i] == 0))
                    {
                        mark[i] = 1;
                        Q.Enqueue(i);
                        parent[i] = v;
                    }
                mark[v] = 2;
            }
            return tracer(parent, from, to);
        }
        public string DijkstraAlgm(int vBegin, int vEnd)
```

```
        {
            int[] distance = new int[numEdges];
            int[] parent = new int[numEdges];
            int[,] matr = new int[numEdges, numEdges];
            // инициализация
            for (int i = 0; i < adjacency.GetLength(0); i++)
                for (int j = 0; j < numEdges; j++)
                {
                    matr[i, j] = adjacency[i, j];
                    if (adjacency[i, j] == 0)
                        matr[i, j] = int.MaxValue;
                }
            HashSet<int> edges = new HashSet<int>();
            for (int i = 0; i < numEdges; i++)
            {
                edges.Add(i);
                distance[i] = matr[vBegin, i];
                if (distance[i] < int.MaxValue)
                    parent[i] = vBegin;
            }
            distance[vBegin] = 0;
            parent[vBegin] = -1;
            edges.Remove(vBegin);
            while (edges.Count != 0)
            {
                int minDistance = int.MaxValue;
                int minEdge = -1;
                foreach (int u in edges)
                {
                    if (distance[u] < minDistance)
                    {
                        minDistance = distance[u];
                        minEdge = u;
                    }
                }
                if (minEdge != -1)
                    edges.Remove(minEdge);
                foreach (int u in edges)
                {
                    if (matr[minEdge, u] < int.MaxValue)
                    {
                        distance[u] = Math.Min(
                            distance[u],
                            distance[minEdge] + matr[minEdge, u]
                       );
                        if (distance[u] == (distance[minEdge] + matr[minEdge, u]))
                        {
                            parent[u] = minEdge;
                        }
                    }
                }
            }

            return tracer(parent, vBegin, vEnd);
        }
        public string Floyd()
        {
            string ret = "";
            int[]  distance = new int[numEdges];
            int[]  parent   = new int[numEdges];
            int[,] matrdist = new int[numEdges, numEdges];
            int[,] matr     = new int[numEdges, numEdges];
            // инициализация
            for (int i = 0; i < numEdges; i++)
                for (int j = 0; j < numEdges; j++)
                {
```

```
                matr[i, j] = adjacency[i, j];
                if (adjacency[i, j] == 0)
                    matr[i, j] = int.MaxValue;
            }
        int c = 0;
        //writebox.Text += c + " ";
        for (int v_Begin = 0; v_Begin < numEdges; v_Begin++)
        {
            ret += "\n";
            HashSet<int> edges = new HashSet<int>();
            for (int i = 0; i < numEdges; i++)
            {
                edges.Add(i);
                distance[i] = matr[v_Begin, i];
                if (distance[i] < int.MaxValue)
                    parent[i] = v_Begin;
            }
            distance[v_Begin] = 0;
            parent[v_Begin] = -1;
            edges.Remove(v_Begin);
            while (edges.Count != 0)
            {
                int minDistance = int.MaxValue;
                int minEdge = -1;
                foreach (int u in edges)
                {
                    if (distance[u] < minDistance)
                    {
                        minDistance = distance[u];
                        minEdge = u;
                    }
                }
                if (minEdge != -1)
                    edges.Remove(minEdge);
                foreach (int u in edges)
                {
                    if (matr[minEdge, u] < int.MaxValue)
                    {
                        distance[u] = Math.Min(
                        distance[u],
                       distance[minEdge] + matr[minEdge, u]
                       );
                        if (distance[u] ==
                        (distance[minEdge] + matr[minEdge, u])
                       )
                        {
                            parent[u] = minEdge;
                        }
                    }
                }
            }
            ret += $"Расстояния от вершины {v_Begin} до ";
            for (int i = 0; i < numEdges; i++)
            {
                matrdist[i, v_Begin] = distance[i];
                ret += "\n";
                ret += $"{i} = {distance[i]}";


            }
        }
        ret += "\n";
        return ret;
    }
    private string tracer(int[] parent, int from, int to)
    {
```

```
            string ret = to.ToString();
            int check = to;
            while (true)
            {
                check = parent[check];
                ret = check + "-" + ret;
                if (check == from)
                    break;
            }
            return ret;
        }
    }
}
```

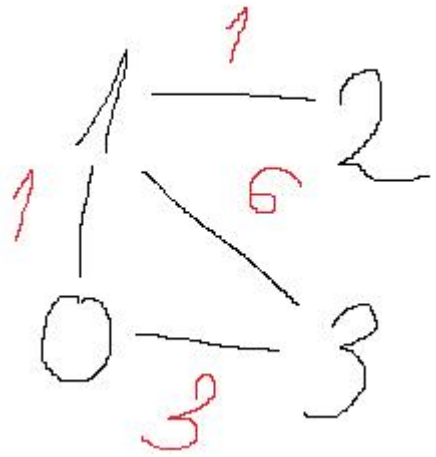

Рис 1 - Граф в тестовом json файле

Содержание тестового json файла:

```
{
  "Edges": 4,
  "Adjacency": [
    [0, 1, 0, 3],
    [1, 0, 1, 6],
    [0, 1, 0, 0],
    [3, 6, 0, 0]
  ],
}
```

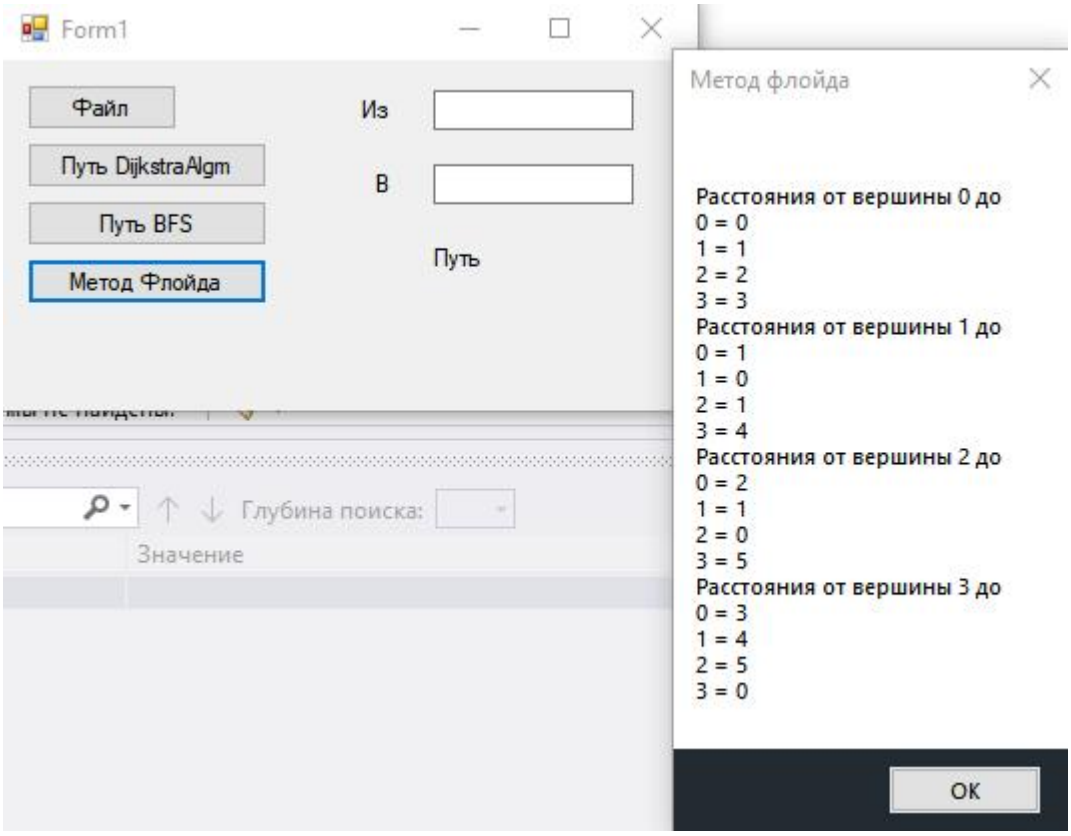

Рис 2 - пример работы программы