ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСП

Кафедра ПИ

Индивидуальное задание

по курсу: «Компьютерная дискретная математика»

на тему: «Поиск в глубину»

Выполнил:

ст. гр. ПИ-21в

Неснов А. А.

Проверил:

Дмитрюк Т. Г.

Донецк — 2022

РЕФЕРАТ

Отчет по индивидуальной работе содержит: 35 страниц, 5 рисунков, 2 приложения и 3 источника.

Цель работы – это закрепить практические навыки по самостоятельной постановке решению задач, связанных обработкой данных с помощью ЭВМ средствами объектно-ориентированного программирования (ООП).

Объект исследования — поиск в глубину.

Цель работы – реализация алгоритма поиска в глубину в виде программы с разработанной визуализацией ввода, входных данных и решения алгоритма.

Результат – программное обеспечение, реализующее решение алгоритма поиска в глубину, вывод результата и запись в файл.

ПОИСК В ГЛУБИНУ, АЛГОРИМТ, ГРАФ, ДИСКРЕТНАЯ МАТЕМАТИКА, ЦИКЛЫ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дискретная математика — часть математики, изучающая дискретные математические структуры, такие как графы и утверждения в логике.

В контексте математики в целом дискретная математика часто отождествляется с конечной математикой — направлением, изучающим конечные структуры — конечные графы, конечные группы, конечные автоматы.

Граф - математический объект, который изображает отношения между сущностями. Граф состоит из вершин (объектов) и рёбер (связей). С помощью графов можно представить разных ситуации: например, пользователей соцсети, которые находятся друг у друга в друзьях, клиентов банка, которые переводят друг другу денежные средства, географические объекты и пути между ними.

Обход графа — это переход от одной его вершины к другой в поисках свойств связей этих вершин. Связи (линии, соединяющие вершины) называются направлениями, путями, гранями или ребрами графа. Двумя основными алгоритмами обхода графа являются поиск в глубину (Depth-First Search, DFS) и поиск в ширину (Breadth-First Search, BFS).

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Постановка задачи

Поиск в глубину — один из методов обхода графа G = (V, E), суть которого состоит в том, чтобы идти “вглубь” пока это возможно.

Поиск начинается с некоторой фиксированной вершины *v*. Рассматривается вершина *u*, смежная с *v*. Она выбирается. Процесс повторяется с вершиной *u*. Если на очередном шаге мы работаем с вершиной *q* и нет вершин, смежных с *q* и не рассмотренных ранее (новых), то возвращаемся из вершины *q* к вершине, которая была до нее. В том случае, когда это вершина *v*, процесс просмотра закончен. Очевидно, что для фиксации признака, просмотрена вершина графа или нет, требуется структура данных типа: Nnew: Array[1..N] Of Boolean [1].

1.2 Метод решения

Метод решения на языке Pascal [2]:

```
Procedure Pg(v:integer); {*Массивы Nnew и A глобальные.*}
     Var j:Integer;
     Begin
          Nnew[v]:=False;Write(v:3);
          For j:=1 To N Do If (A[v, j] <> 0) And Nnew[j]
               Then Pg (j)
     End;
```

Метод решения задачи из языка C#:

```
public List<List<int>> DFS(int start)
   {
       var result = new List<int>();
       var pre = new List<int>();
       result.Add(start);
       pre.Add(-1);
```

```
        var arr = Enumerable.Range(0, adj_matrix.Count).Select(x
=> true).ToList();

        void DFSrecursion(int v)
        {
            arr[v] = false;
            for (int i = 0; i < arr.Count; i++)
            {
                if (adj_matrix[v][i] && arr[i])
                {
                    result.Add(i);
                    pre.Add(v);
                    DFSrecursion(i);
                }
            }
        }

        DFSrecursion(start);
        return new List<List<int>>() { result, pre };
    }
```

Для обеспечения корректного отображения результата поиска в глубину рекурсия была вынесена в функцию (void DFSrecursion(int v)), а глобальная переменная с пройденными вершинами (arr) сделана локальной для уменьшения потребления памяти. Также, функция DFS возвращает список [2] из двух списков: порядка обхода вершин (res) и их предшественников (pre). Второй массив нужен для того, чтобы при отрисовке графа корректно указывался путь, по которому происходит обход. Первый элемент этого списка всегда будет равен -1, т.к. вершина, предшествующая ему, не существует. Таким образом, при каждом шаге в результирующий массив будет записываться следующий элемент, а текущий — в предшествующий.

2 КОНТРОЛЬНЫЙ ПРИМЕР

2.1 Реализация контрольного примера

В качестве контрольного примера выбран следующий граф (см. рис. 2.1).

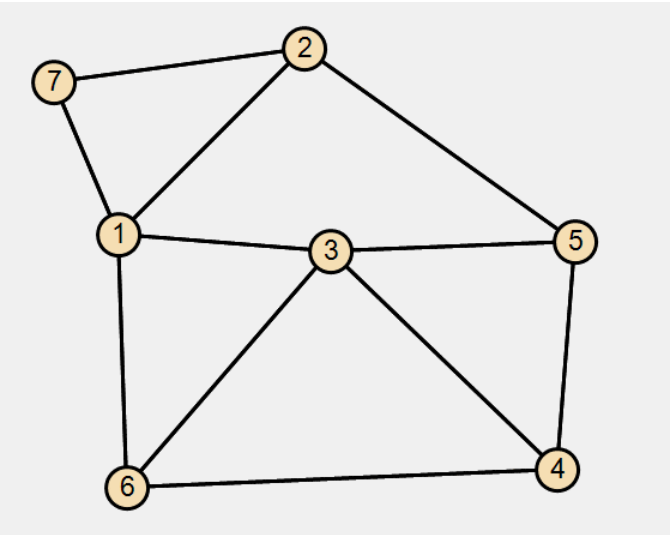

Рисунок 2.1 – Граф контрольного примера

2.2 Ручной просчет контрольного примера

A =

	1	2	3	4	5	6	7
1	0	1	1	0	0	1	1
2	1	0	0	0	1	0	1
3	1	0	0	1	1	1	0
4	0	0	1	0	1	1	0
5	0	1	1	1	0	0	0
6	1	0	1	1	0	0	0
7	1	1	0	0	0	0	0

Таблица 1. Матрица смежности

Начнем с вершины 1.

Посещенные вершины N: {1}

Порядок обхода: {1}

Следующая вершина – v2, v2 ∉ N.

Посещенные вершины N: {1, 2}

Порядок обхода: {1, 2}

Следующая вершина – v1, v1 ∈ N; v5 ∉ N.

Посещенные вершины N: {1, 2, 5}

Порядок обхода: {1, 2, 5}

Следующая вершина – v3, v3 ∉ N.

Посещенные вершины N: {1, 2, 3, 5}

Порядок обхода: {1, 2, 5, 3}

Следующая вершина – v1, v1 ∈ N; v4 ∉ N.

Посещенные вершины N: {1, 2, 3, 4, 5}

Порядок обхода: {1, 2, 5, 3, 4}

Следующая вершина – v3, v3 ∈ N, v5 ∈ N; v6 ∉ N.

Посещенные вершины N: {1, 2, 3, 4, 5, 6}

Порядок обхода: {1, 2, 5, 3, 4, 6}

Следующая вершина – v1, v1 ∈ N, v3 ∈ N, v4 ∈ N.

Нет допустимых вершин, возвращение назад.

Следующая вершина – v3, v3 ∈ N, v5 ∈ N, v6 ∈ N.

Нет допустимых вершин, возвращение назад.

Следующая вершина – v1, v1 ∈ N, v4 ∈ N, v5 ∈ N; v6 ∈ N.

Нет допустимых вершин, возвращение назад.

Следующая вершина – v2, v2 ∈ N, v3 ∈ N; v4 ∈ N.

Нет допустимых вершин, возвращение назад.

Следующая вершина – v1, v1 ∈ N, v5 ∈ N; v7 ∉ N.

Посещенные вершины N: {1, 2, 3, 4, 5, 6, 7}

Порядок обхода: {1, 2, 5, 3, 4, 6, 7}

Следующая вершина – v1, v1 ∈ N, v2 ∈ N.

Нет допустимых вершин, возвращение назад.

Следующая вершина – v1, v1 ∈ N, v5 ∈ N, v7 ∈ N.

Нет допустимых вершин, возвращение назад.

Возвращение в вершину 1, конец алгоритма.

В результате поиска в глубину получен следующий порядок обхода: {1, 2, 5, 3, 4, 6, 7}

2.2 Программный просчет контрольного примера

Результат программного расчета приведен на рисунке 2.2.

Листинг решенного контрольного примера приведен в соответствующем файле лога: «Порядок обхода: 1 2 5 3 4 6 7»

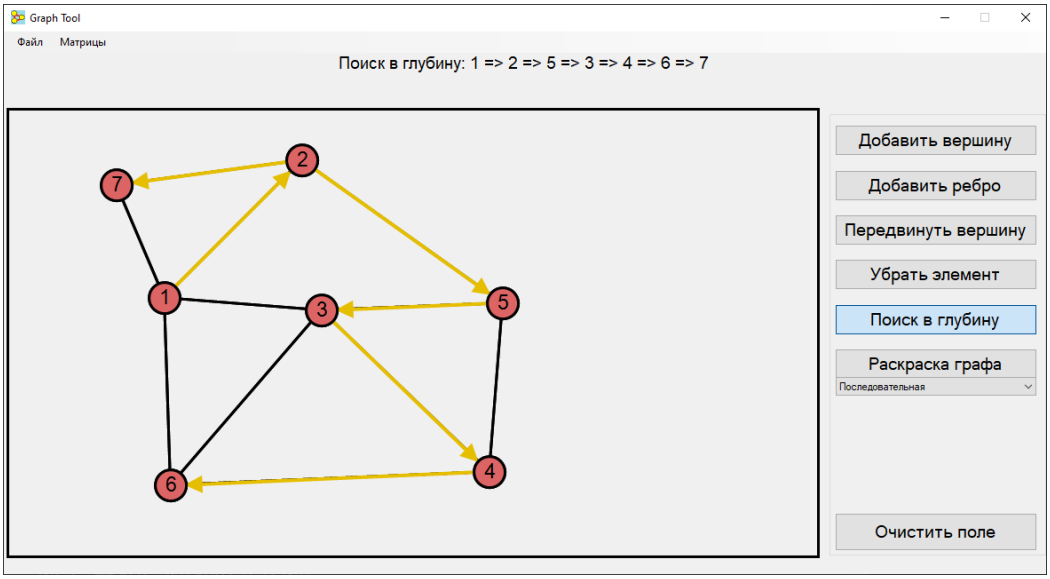

Рисунок 2.2 – Программное решение задачи

3 ДОПОЛНИТЕЛЬНЫЕ ПРИМЕРЫ

3.1. Решение пустого графа

Поиск в глубину при пустом графе невозможно запустить из-за отсутствия вершин и, соответственно, возможности выбрать начало.

3.2. Решение связного графа, полученного генерацией варианта

Решение графа G(13, {5, 6}) приведено ниже (см. рис 3.1). Листинг работы программы: «Порядок обхода: 1 5 7 3 6 2 4 11 9 12 13 8 10»

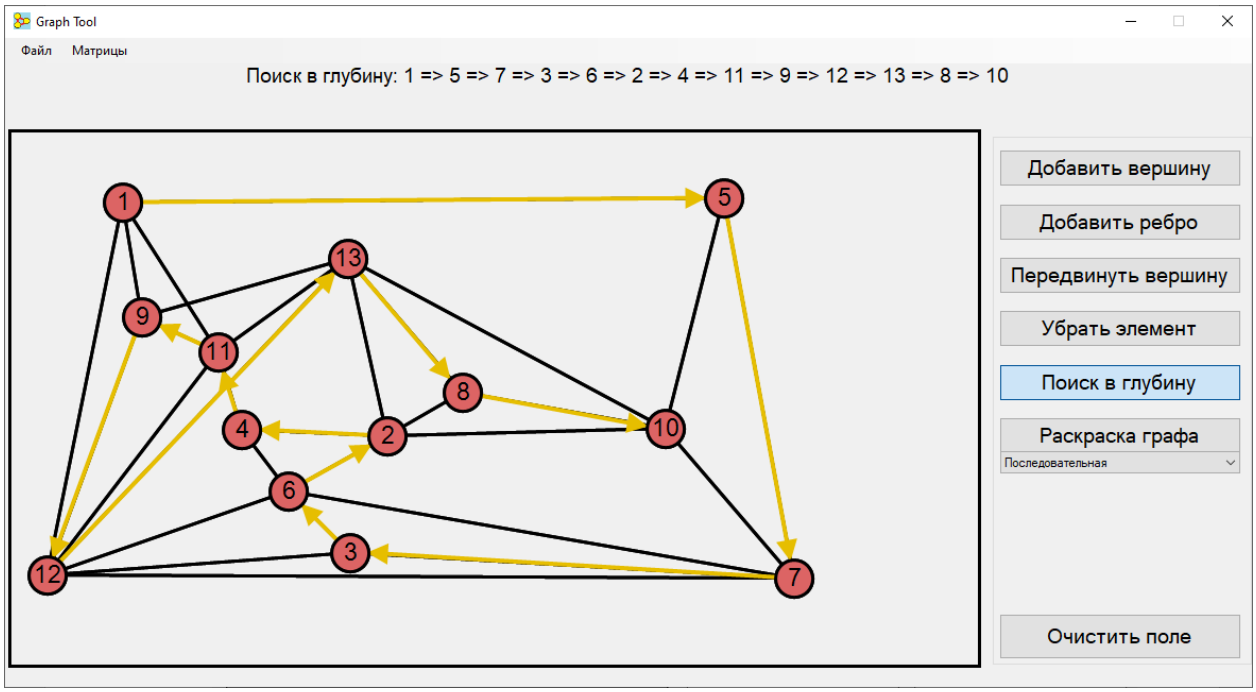

Рисунок 3.1 – Решение связного графа

3.3. Решение орграфа

Если граф ориентированный, то, находясь в узле x, необходимо выбирать ребро (x, y), только выходящее из x. Исследовав все ребра, выходящие из y, возвращаемся в x даже тогда, когда в y входят другие ребра, еще не рассмотренные [3]. Программа способна находить решения для орграфов (см. рис. 3.2). Результат работы указан ниже (см. рис. 3.3.). Листинг решения: «Порядок обхода: 1 3».

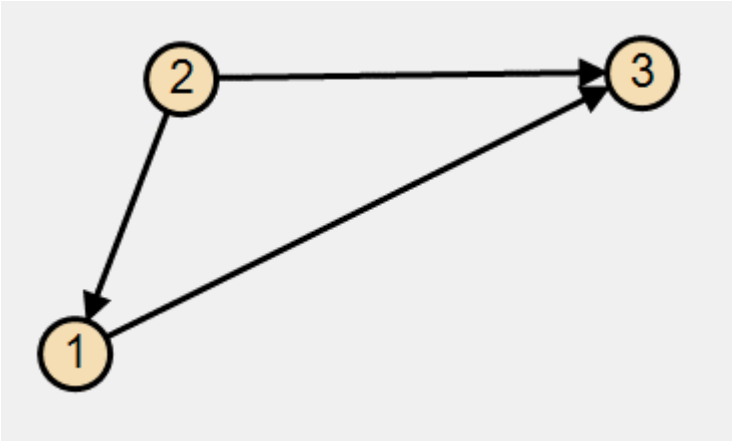

Рисунок 3.2 – Связный орграф

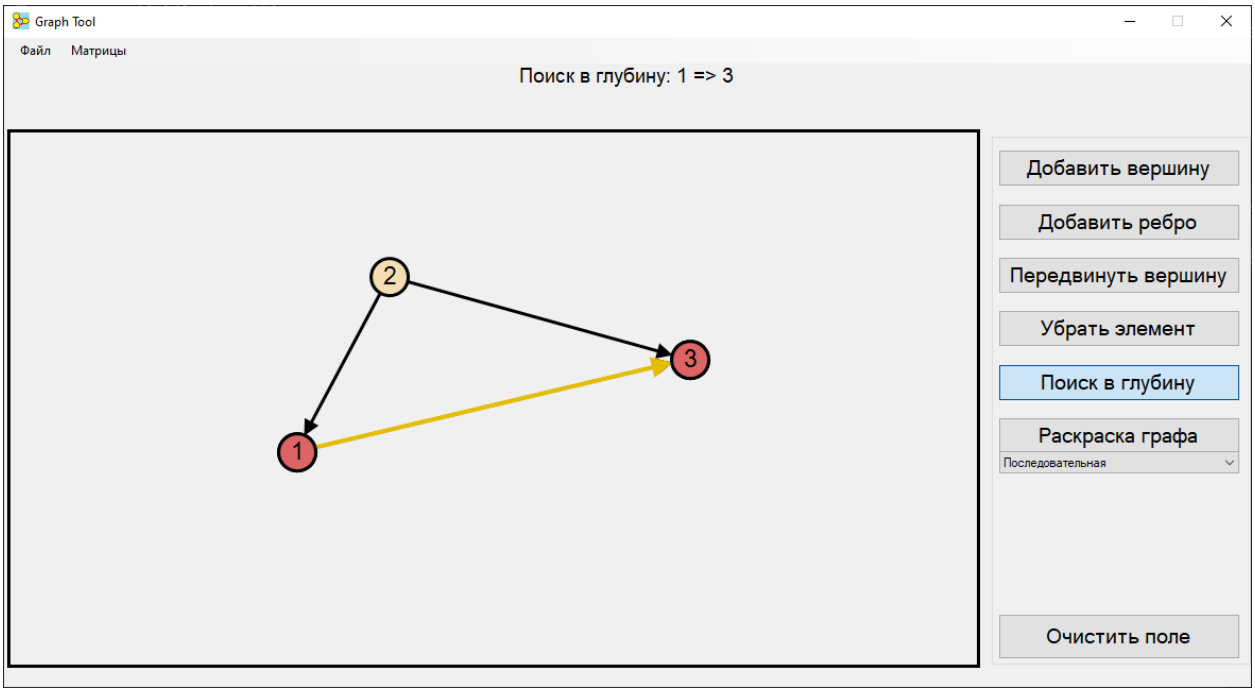

Рисунок 3.3 – Решение связного орграфа

4 ОПИСАНИЕ ПРОГРАММЫ

Программа «Graph.exe» графический интерфейс с возможностью решения поиска в глубину в графе.

Программа предназначена для использования в личных целях при решении задач поиска в глубину и в высших учебных заведениях с целью демонстрации и проверки решений алгоритма.

Для запуска программы требуется персональный компьютер с частотой процессора не менее 1.2 ГГц и объемом оперативной памяти не менее 500 Мб.

Вызов и загрузка программы осуществляется путем запуска исполняемого файла Graph.exe.

В качестве исходных данных программа использует данные, считываемые из внешних файлов, а также вводимые пользователем с клавиатуры или с помощью мыши.

Выходные данные выводятся на экран и в файл логов в папке программы.

ВЫВОД

Разработанная программа является демонстрацией решения алгоритма поиска фундаментальных циклов. Она предназначена для использования в личных целях при решении задач поиска фундаментальных циклов и в высших учебных заведениях с целью демонстрации и проверки решений алгоритма.

Разработанная программа соответствует поставленным требованиям, реализует поставленную задачу.

Функционал программы дополняется графическим представлением графа и табличным вариантом матрицы смежности.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Окулов, С.М. Программирование в алгоритмах — М.: БИНОМ. Лаборатория знаний, 2002. — 341 с.
2. Прайс, М. C# 8 и .NET Core. Разработка и оптимизация. / Прайс М., 2021. — 187 с.
3. Иванов, Б. Н. Дискретная математика. Алгоритмы и программы: Учеб. пособие. / Б. Н. Иванов. — М.: Лаборатория Базовых Знаний, 2003. — 118 с.

ПРИЛОЖЕНИЕ А
РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для начала работы требуется запустить файл Graph.exe.

После запуска пользователь может загрузить файл с расширением «.grf», содержащий в себе список координат вершин и матрицу смежности. Также у пользователя есть возможность использования указателя мыши для выбора режима работы программы, такого как добавление вершин и рёбер, передвижение и удаление элементов и выбор начала поиска в глубину.

В любой момент пользователь может открыть матрицу смежности через контекстное меню для изменения графа.

При выборе начальной вершины для поиска в глубину программа покажет порядок обхода вершин в текстовом поле, а пройденный путь отобразится на графе другим цветом.

Все действия пользователя будут записаны в лог с датой, а результат алгоритма будет сохранён в отдельный файл.

ПРИЛОЖЕНИЕ Б
ЛИСТИНИНГ ПРОГРАММЫ

```
//Graph.cs
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;


namespace Graph
{    class GGraph
    {
        public List<Point> vertexes
= new List<Point>();
        public List<List<bool>>
adj_matrix = new List<List<bool>>();



        public Pen np;
        public Brush nb;
        public Font drawFont;
        public Brush fontb;
        public Color v_color;
        public int size;




        public GGraph(List<Point>
vertexes, List<List<bool>> matrix)
        {
            this.vertexes =
vertexes;
            this.adj_matrix =
matrix;



            v_color = Color.Wheat;
            np = new
Pen(Color.Black, 3);
            nb = new
SolidBrush(v_color);
            fontb = new
SolidBrush(Color.Black);
            size = 35;
            drawFont = new
Font("Arial", size/2);
        }



        public int vertexClicked(int
x, int y)
        {
            foreach (var vertex in
vertexes)
            {
                if (distance(x, y,
vertex.X + size/2, vertex.Y +
size/2) < size/2)
                    return
vertexes.IndexOf(vertex);
            }
            return -1;
        }



        public Tuple<int, int>
edgeClicked(int x, int y)
        {
            for (int i = 0; i <
adj_matrix.Count; i++)
                for (int j = i; j <
adj_matrix.Count; j++)
                {
                    if (i == j)
                    {
                        double r =
distance(x, y, vertexes[i].X + size
/ 2, vertexes[i].Y - size / 2);
                        if (r <
size/2 )
                            return
new Tuple<int, int>(i, j);
                    }
```

```
                    else if
(adj_matrix[i][j] ||
adj_matrix[j][i])
                    {
                        double d1 =
Math.Sqrt(Math.Pow(x -
vertexes[i].X, 2) + Math.Pow(y -
vertexes[i].Y, 2));
                        double d2 =
Math.Sqrt(Math.Pow(x -
vertexes[j].X, 2) + Math.Pow(y -
vertexes[j].Y, 2));
                        double d =
Math.Sqrt(Math.Pow(vertexes[i].X -
vertexes[j].X, 2) +
Math.Pow(vertexes[i].Y -
vertexes[j].Y, 2));
                        if ((d - 3 <
d1 + d2) && (d1 + d2 < d + 3))
                            return
new Tuple<int, int>(i, j);
                    }
                }


            return null;
        }


        public List<List<int>>
DFS(int start)
        {
            var result = new
List<int>();
            var pre = new
List<int>();
            result.Add(start);
            pre.Add(-1);


            var arr =
Enumerable.Range(0,
adj_matrix.Count).Select(x =>
true).ToList();


            void DFSrecursion(int v)
            {
                arr[v] = false;
                for (int i = 0; i <
arr.Count; i++)
                {
                    if
(adj_matrix[v][i] && arr[i])
                    {

result.Add(i);
                        pre.Add(v);

DFSrecursion(i);
                    }
                }
            }


            DFSrecursion(start);
            return new
List<List<int>>() { result, pre };
        }


        public Tuple<List<int>, int>
coloring(List<int> order)
        {
            if (order == null) //
Если не задан порядок - взять
последовательный
                order =
Enumerable.Range(0,
adj_matrix.Count).ToList();
            List<int> colors =
Enumerable.Range(0,
adj_matrix.Count).Select(x =>
0).ToList();


            colors[order[0]] = 1;


            for (int i=1;
i<adj_matrix.Count; i++)
            {
                List<int> taken =
new List<int>();
                for (int j = 0; j <
adj_matrix.Count; j++)
```

```
                {
                    if
(adj_matrix[order[i]][j] &&
taken.IndexOf(colors[j]) == -1 &&
colors[j] > 0)

taken.Add(colors[j]);
                }
                taken.Sort();
                for (int j = 0; j <
taken.Count; j++)
                {
                    if(j+1 !=
taken[j])
                    {

colors[order[i]] = j + 1;
                        break;
                    }
                }
                if (colors[order[i]]
== 0)
                    colors[order[i]]
= taken.Count+1;
            }


            return new
Tuple<List<int>, int>(colors,
listMax(colors));
        }


        public Tuple<List<int>, int>
descColoring() // НП-упор.
        {
            List<Tuple<int, int>>
v_deg = new List<Tuple<int, int>>();
            for (int i = 0; i <
adj_matrix.Count; i++)
                v_deg.Add(new
Tuple<int, int>(i, degree(i)));
            v_deg = v_deg.OrderBy(x
=> x.Item2).ToList();
            v_deg.Reverse();
            var q = from x in v_deg
                    group x by
x.Item2 into g
                    let count =
g.Count()
                    orderby count
descending
                    select new {
Value = g.Key, Count = count };


            foreach (var v in q)
            {
                if (v.Count > 1)
                {
                    Tuple<int, int>
temp = null;
                    var index =
v_deg.FindIndex(t => t.Item2 ==
v.Value);
                    for (int i =
index; i < index + v.Count - 1; i++)
                    {
                        for (int j =
i; j < index + v.Count - 1; j++)
                        {
                            var
ndeg1 = ndegree(v_deg[i].Item1, 2);
                            var
ndeg2 = ndegree(v_deg[j].Item1, 2);
                            if
(ndeg1 < ndeg2)
                            {
                                temp
= v_deg[i];

v_deg[i] = v_deg[j];

v_deg[j] = temp;
                            }
                        }
                    }
                }
            }
```

```
            var result =
Enumerable.Range(0,
v_deg.Count).Select(x =>
v_deg[x].Item1).ToList();

            return coloring(result);

        }


        public Tuple<List<int>, int>
ascColoring() // ПН-упор.

        {

            var result = new
List<int>();

            var matrix = new
List<List<bool>>();

            for (int i = 0;
i<adj_matrix.Count;i++)

            {

                matrix.Add(new
List<bool>());

                for (int j = 0; j <
adj_matrix[i].Count; j++)


matrix[i].Add(adj_matrix[i][j]);

            }



            var count =
Enumerable.Range(0,
matrix.Count).Select(x =>
true).ToList();

            for (int v = 0; v <
matrix.Count-1; v++)

            {

                int min =
int.MaxValue;

                int min_i = 0;

                for (int i = 0; i <
matrix.Count; i++)

                {

                    int deg =
degree(i, matrix);

                    if (0 < deg &&
deg < min)

                    {

                        min_i = i;

                        min = deg;

                    }


                }

                matrix =
remove_v(min_i, matrix);

                result.Add(min_i);

                count[min_i] =
false;

            }



result.Add(count.IndexOf(true));

            result.Reverse();

            return coloring(result);

        }


        public
Tuple<List<List<int>>, int>
edgeColoring() // Рёберная раскраска

        {

            var result =
Enumerable.Range(0,
adj_matrix.Count).Select(

                x =>
Enumerable.Range(0,
adj_matrix.Count).Select(y =>
0).ToList()).ToList();



            for (int i = 0; i <
result.Count; i++)

            {

                for (int j = i; j <
result.Count; j++)

                {

                    if
(adj_matrix[i][j])

                    {

                        for (int cr
= 1; cr < result[i].Count; cr++)

                        {
```

```
                            if
(result[i].IndexOf(cr) == -1 &&
result[j].IndexOf(cr) == -1)
                            {

result[i][j] = result[j][i] = cr;

break;
                            }
                        }

                    }
                }
            }
            int max = 0;
            foreach (var v in
result)
                max = Math.Max(max,
listMax(v));


            return new
Tuple<List<List<int>>, int>(result,
max);
        }


        public double distance(int
x1, int y1, int x2, int y2)
        {
            return
Math.Sqrt(Math.Pow(x1 - x2, 2) +
Math.Pow(y1 - y2, 2));
        }


        public int degree(int v,
List<List<bool>> matrix = null)
        {
            int deg = 0;
            if (matrix == null)
            {
                for (int i = 0; i <
adj_matrix.Count; i++)
                {
                    if
(adj_matrix[i][v]) deg++;
                }
            }
            else
            {
                for (int i = 0; i <
matrix.Count; i++)
                {
                    if
(matrix[i][v]) deg++;
                }
            }
            return deg;
        }


        public int ndegree(int v,
int n)
        {
            if (n > 1)
            {
                int sum = 0;
                for (int i = 0; i <
adj_matrix.Count; i++)
                {
                    if
(adj_matrix[v][i])
                    {
                        sum +=
ndegree(i, n - 1);
                    }
                }
                return sum;
            }
            else return degree(v);
        }


        public List<List<bool>>
remove_v(int v, List<List<bool>>
matrix = null)
```

```
        {
            if (matrix == null)
            {
                vertexes.RemoveAt(v);
                adj_matrix.RemoveAt(v);
                for (int i = 0; i < adj_matrix.Count; i++)
                    adj_matrix[i].RemoveAt(v);
                return null;
            }
            else
            {
                for (int i=0; i<matrix.Count; i++)
                {
                    matrix[i][v] = matrix[v][i] = false;
                }
                return matrix;
            }
        }

        public int listMax(List<int> list)
        {
            int max = -1;
            foreach (var i in list)
            {
                if (i > max) max = i;
            }
            return max;
        }

        public int edgeCount()
        {
            int count = 0;
            for (int i = 0; i < adj_matrix.Count; i++)
            {
                for (int j = i; j < adj_matrix[i].Count; j++)
                {
                    if (adj_matrix[i][j] || adj_matrix[j][i])
                        count++;
                }
            }
            return count;
        }

    }
}
// MainForm.cs
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Windows.Forms;
using System.Threading;
using System.Globalization;
using Graph.Properties;
using System.IO;

using System.ComponentModel;
using System.Runtime.InteropServices.ComTypes;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Text;
```

```
using
System.Windows.Forms.VisualStyles;

using static
System.Windows.Forms.VisualStyles.Vi
sualStyleElement;

using static
System.Windows.Forms.VisualStyles.Vi
sualStyleElement.TreeView;

using System.Xml;

using System.Collections;

using static
System.Net.WebRequestMethods;



namespace Graph

{

    public partial class MainForm :
Form

    {



        GGraph graph;

        int clicked_v = -1;

        int current = -1;

        List<List<int>> dfs = null;

        Tuple<List<int>, int>
v_colors = null;

        Tuple<List<List<int>>, int>
e_colors = null;

        List<Color> colors = null;

        TextWriter log = null;



        public MainForm()

        {

            InitializeComponent();



            graph = new GGraph(new
List<Point>(), new
List<List<bool>>());

            colors = new
List<Color>();


ColoringMenu.SelectedItem =
"Последовательная";

SetStyle(ControlStyles.UserPaint,
true);


SetStyle(ControlStyles.AllPaintingIn
WmPaint, true);


SetStyle(ControlStyles.DoubleBuffer,
true);

        }

        private void
Form1_Load(object sender, EventArgs
e)

        {

            var t =
DateTime.Now.ToString("yy:MM:dd:H:mm
:ss tt").Replace(':', '_');

            log = new
StreamWriter(string.Format("GrapthTo
ol [{0:S}].log", t));

            AddV_Click(this, new
EventArgs());

        }



        private void
Form1_Paint(object sender,
PaintEventArgs e)

        {

            log.WriteLine("Начало
отрисовки.");

            Graphics g = e.Graphics;

            g.SmoothingMode =
SmoothingMode.AntiAlias;



            void draw_vertex(Font
font, Brush f_brush, Pen pen, Brush
v_brush, Point p)

            {

                Rectangle rect = new
Rectangle(p.X, p.Y, graph.size,
graph.size);

                StringFormat sf =
new StringFormat();

                sf.Alignment =
StringAlignment.Center;
```

```
                sf.LineAlignment =
StringAlignment.Center;



g.FillEllipse(v_brush, rect);
                g.DrawEllipse(pen,
rect);
                Point sp = new
Point(p.X + graph.size / 2 + 1, p.Y
+ graph.size / 2 + 1);

g.DrawString((graph.vertexes.IndexOf
(p) + 1).ToString(), font, f_brush,
sp, sf);
            }


            void draw_edge(Point
from, Point to, Pen pen, bool
directed=false)
            {
                if (from == to)
                {
                    Rectangle rect =
new Rectangle(from.X - graph.size/3,
from.Y - (int)(graph.size),
(int)(graph.size * 1.5),
(int)(graph.size * 1.5));

g.DrawEllipse(pen, rect);
                }


                else
                {
                    Point p1 = new
Point(from.X + graph.size / 2,
from.Y + graph.size / 2);
                    Point p2 = new
Point(to.X + graph.size / 2, to.Y +
graph.size / 2);
                    if (directed)
                    {
                        double sin()
                        {
                            return
(Math.Abs(from.Y - to.Y) /
graph.distance(from.X, from.Y, to.X,
to.Y));
                        }


                        Pen arrowpen
= new Pen(Color.Black, pen.Width);

arrowpen.CustomEndCap = new
AdjustableArrowCap(5, 5);
                        if (DFS !=
null)
                        {

arrowpen.Width = 4;

arrowpen.Color = Color.FromArgb(255,
230, 190, 0);
                        }


                        int x_diff =
(int)(graph.size / 2 * Math.Sqrt(1 -
Math.Pow(sin(), 2)));
                        int y_diff =
(int)(graph.size / 2 * sin());


                        Func<bool,
int, int> swap = (x, y) => x ? y * -
1 : y;
                        p2 = new
Point(to.X + graph.size / 2 +
swap(to.X > from.X, x_diff), to.Y +
graph.size / 2 + swap(to.Y > from.Y,
y_diff));

g.DrawLine(arrowpen, p1, p2);
                    }
                    else
g.DrawLine(pen, p1, p2);
                }
            }



g.DrawRectangle(graph.np,
panel1.Location.X,
panel1.Location.Y,
panel1.Width+panel1.Location.X,
panel1.Height);
```

```
            for (int i = 0; i <
graph.adj_matrix.Count; i++)
            {
                for (int j = 0; j <
graph.adj_matrix[i].Count; j++)
                {
                    if
(graph.adj_matrix[i][j] &&
graph.adj_matrix[j][i])
                    {

draw_edge(graph.vertexes[i],
graph.vertexes[j], graph.np, false);
                    }
                    else if
(graph.adj_matrix[i][j] &&
!graph.adj_matrix[j][i])
                    {

draw_edge(graph.vertexes[i],
graph.vertexes[j], graph.np, true);
                    }
                }
            }


            if (e_colors != null)
            {
                Random rnd = new
Random();
                label.Text =
"Количество цветов: " +
e_colors.Item2.ToString();
                colors.Clear();
                for (int i = 0; i <
e_colors.Item2; i++) // for (int i =
colors.Count; i < e_colors.Item2;
i++)
                {

colors.Add(Color.FromArgb(255,
rnd.Next(255), rnd.Next(255),
rnd.Next(255)));
                }
                for (int i = 0; i <
e_colors.Item1.Count; i++)
                {
                    for (int j = 0;
j < e_colors.Item1[i].Count; j++)
                    {
                        if
(e_colors.Item1[i][j] > 0)
                        {
                            Pen
colorpen = new
Pen(colors[e_colors.Item1[i][j]-1],
3);

draw_edge(graph.vertexes[i],
graph.vertexes[j], colorpen);
                        }
                    }
                }
            }


            if (graph.vertexes !=
null)
            {
                foreach (var v in
graph.vertexes)
                {

draw_vertex(graph.drawFont,
graph.fontb, graph.np, graph.nb, v);
                }
            }
            if (clicked_v >= 0)
            {

draw_vertex(graph.drawFont,
graph.fontb, graph.np, new
SolidBrush(Color.FromArgb(255, 230,
150, 150)),
graph.vertexes[clicked_v]);
            }


            if (dfs != null)
            {
```

```
                label.Text = "Поиск
в глубину: ";


                for (int i = 1; i <
dfs[0].Count; i++)
                {

draw_edge(graph.vertexes[dfs[1][i]],
graph.vertexes[dfs[0][i]], graph.np,
true);
                }




                var t =
string.Format("DFS [{0:S}].log",
DateTime.Now.ToString("yy:MM:dd:H:mm
:ss tt").Replace(':', '_'));
                TextWriter dfslog =
new StreamWriter(t);

log.WriteLine("Результат поиска в
глубину записан в файл [" + t +
']');



dfslog.Write("Порядок обхода: ");
                foreach (var v in
dfs[0])
                {

dfslog.Write((v+1).ToString() + "
");
                    label.Text =
label.Text + (v + 1).ToString() + "
=> ";

draw_vertex(graph.drawFont,
graph.fontb, graph.np, new
SolidBrush(Color.FromArgb(255, 220,
100, 100)), graph.vertexes[v]);
                }
                dfslog.Close();
                label.Text =
label.Text.ToString().Remove(label.T
ext.Length - 4, 4);
                label.Refresh();
            }
            if (v_colors != null)
            {
                Random rnd = new
Random();
                label.Text =
"Количество цветов: " +
v_colors.Item2.ToString();
                colors.Clear();
                for (int i = 0; i <
v_colors.Item2; i++) // for (int i =
colors.Count; i < v_colors.Item2;
i++)
                {

colors.Add(Color.FromArgb(255,
rnd.Next(255), rnd.Next(255),
rnd.Next(255)));
                }


                for (int i = 0; i <
v_colors.Item1.Count; i++)
                {


draw_vertex(graph.drawFont,
graph.fontb, graph.np, new
SolidBrush(colors[v_colors.Item1[i]-
1]), graph.vertexes[i]);
                    Point p = new
Point(graph.vertexes[i].X +
graph.size/2, graph.vertexes[i].Y -
graph.size/2);
                    StringFormat sf
= new StringFormat();
                    sf.Alignment =
StringAlignment.Center;
                    sf.LineAlignment
= StringAlignment.Center;

g.DrawString((v_colors.Item1[i]).ToS
tring(), graph.drawFont,
graph.fontb, p, sf);
                }
            }
            log.WriteLine("Конец
отрисовки.");
        }
```

```
        private void
Form1_MouseClick(object sender,
MouseEventArgs e)
        {

            if (AddE.Checked)
            {
                int vc =
graph.vertexClicked(e.X, e.Y);

                if (vc != -1)
                {
                    if (clicked_v ==
-1)
                    {
                        clicked_v =
vc;

log.WriteLine("Выбрана вершина " +
vc);
                    }
                    else
                    {
                        EdgeDialogue
choice = new EdgeDialogue(clicked_v,
vc);

                        if
((graph.adj_matrix[vc][clicked_v] ==
false) ||
(graph.adj_matrix[clicked_v][vc] ==
false))
                        {
                            if
(choice.ShowDialog() ==
DialogResult.OK)
                            {
                                if
(choice.isDirected)
                                {

graph.adj_matrix[clicked_v][vc] =
true;

log.WriteLine(string.Format("Добавле
на дуга ({0:S},{1:S})",
clicked_v.ToString(),
vc.ToString()));
                                }
                                else
                                {

graph.adj_matrix[vc][clicked_v] =
graph.adj_matrix[clicked_v][vc] =
true;

log.WriteLine(string.Format("Добавле
но ребро ({0:S},{1:S})",
clicked_v.ToString(),
vc.ToString()));
                                }
                            }
                        }
                        clicked_v =
-1;
                    }
                }
                else
                {
                    clicked_v = -1;

log.WriteLine("Вершина не была
выбрана.");
                }
                }


            if (AddV.Checked)
            {
                if (e.X <
panel1.Width + panel1.Location.X -
graph.size &&
                    e.Y <
panel1.Height + panel1.Location.Y -
graph.size &&
                    e.X >
panel1.Location.X && e.Y >
panel1.Location.Y)
                {
```

```
log.WriteLine("Добавлена вершина " +
graph.adj_matrix.Count);

graph.vertexes.Add(new Point(e.X,
e.Y));

graph.adj_matrix.Add(Enumerable.Rang
e(0,
graph.adj_matrix.Count+1).Select(x
=> false).ToList());
                    for (int i = 0;
i < graph.adj_matrix.Count-1; i++)
                    {

graph.adj_matrix[i].Add(false);
                    }
                }
            }

            if (remove.Checked)
            {
                var ec =
graph.edgeClicked(e.X, e.Y);
                int vc =
graph.vertexClicked(e.X, e.Y);

                if (vc != -1)
                {

graph.remove_v(vc);

log.WriteLine("Удалена вершина " +
vc);
                }

                else if (ec != null)
                {

graph.adj_matrix[ec.Item1][ec.Item2]
=
graph.adj_matrix[ec.Item2][ec.Item1]
= false;

log.WriteLine("Удалено ребро " +
ec);
                }
            }
            if (DFS.Checked)
            {
                int vc =
graph.vertexClicked(e.X, e.Y);
                if (vc != -1)
                {
                    dfs =
graph.DFS(vc);

log.WriteLine("Выполнен поиск в
глубину.");
                }
            }
            this.Refresh();
        }

        private void
Form1_MouseDown(object sender,
MouseEventArgs e)
        {
            if (move.Checked)
            {
                if (current == -1)
                {
                    current =
graph.vertexClicked(e.X, e.Y);

log.WriteLine("Перемещение вершины "
+ current);
                }
                this.Refresh();
            }
        }

        private void
Form1_MouseUp(object sender,
MouseEventArgs e)
```

```
        {
            if (move.Checked)
            {
                log.WriteLine("Конец
перемещения.");
                current = -1;
                this.Refresh();
            }
        }

        private void
Form1_MouseMove(object sender,
MouseEventArgs e)
        {
            if (move.Checked &&
current != -1)
            {
                int x =
graph.vertexes[current].X;
                if
(panel1.Location.X < e.X && e.X <
panel1.Width + panel1.Location.X -
graph.size)
                    x = e.X;

                int y =
graph.vertexes[current].Y;
                if
(panel1.Location.Y < e.Y && e.Y <
panel1.Height + panel1.Location.Y -
graph.size)
                    y = e.Y;

graph.vertexes[current] = new
Point(x, y);
                this.Refresh();
            }
        }

        private void
adjacencyToolStripMenuItem_Click(obj
ect sender, EventArgs e)
        {
            log.WriteLine("Открытие
матрицы смежности.");
            AdjMatrix dialoge = new
AdjMatrix(graph.adj_matrix);

            if (dialoge.ShowDialog()
== DialogResult.OK)
            {

log.WriteLine("Изменение матрицы
смежности.");
                Random rnd = new
Random();
                Func<int, bool>
reverse = x => x > 0;
                graph.adj_matrix =
new List<List<bool>>();
                for (int i = 0; i <
dialoge.dt.Rows.Count-1; i++)
                {

graph.adj_matrix.Add(new
List<bool>());
                    for (int j = 0;
j < dialoge.dt.Columns.Count; j++)
                    {
                        var cellvaue
= dialoge.dt.Rows[i][j];

graph.adj_matrix[i].Add(reverse(int.
Parse(cellvaue.ToString())));
                    }
                }

log.WriteLine("Обновление списка
точек вершин");
                if
(graph.adj_matrix.Count >
graph.vertexes.Count)
                {
                    for (int i =
graph.vertexes.Count; i <
graph.adj_matrix.Count; i++)

graph.vertexes.Add(new
```

```
Point(rnd.Next(panel1.Location.X,
panel1.Width - graph.size),
rnd.Next(panel1.Location.Y,
panel1.Height - graph.size)));
                }
                else
                {

graph.vertexes.RemoveRange(graph.adj
_matrix.Count, graph.vertexes.Count
- graph.adj_matrix.Count);
                }
            }

            this.Refresh();
        }

        private void
AddV_Click(object sender, EventArgs
e)
        {
            label.Text = "Щелкните
на поле для добавления вершины";
            this.Refresh();
        }

        private void
AddE_Click(object sender, EventArgs
e)
        {
            clicked_v = -1;
            label.Text = "Выберите
вершину";
            this.Refresh();
        }

        private void
move_CheckedChanged(object sender,
EventArgs e)
        {
            label.Text = "Зажмите и
перемещайте вершину";
            this.Refresh();
        }

        private void
remove_CheckedChanged(object sender,
EventArgs e)
        {
            label.Text = "Щелкните
по вершине или ребру для его
удаления";
            this.Refresh();
        }

        private void
DFS_CheckedChanged(object sender,
EventArgs e)
        {
            dfs = null;
            label.Text = "Выберите
вершину для поиска в глубину";
            this.Refresh();
        }

        private void
Coloring_MouseClick(object sender,
MouseEventArgs e)
        {
            if (Coloring.Checked)
            {

log.WriteLine("Раскраская графа.");
                switch
(ColoringMenu.SelectedItem.ToString(
))
                {
                    case
"Последовательная":
                        v_colors =
graph.coloring(null);
                        break;

                    case "НП -
нисходящая":
```

```
                        v_colors =
graph.descColoring();
                        break;


                    case "ПН -
восходящая":
                        v_colors =
graph.ascColoring();
                        break;


                    case "Рёберная":
                        e_colors =
graph.edgeColoring();
                        break;
                }
                this.Refresh();
            }
        }
        private void
Coloring_CheckedChanged(object
sender, EventArgs e)
        {
            if (Coloring.Checked ==
false)
            {
                v_colors = null;
                e_colors = null;
            }
            this.Refresh();
        }


        private void
saveToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            log.WriteLine("Попытка
сохранить в файл.");
            SaveFileDialog sfd = new
SaveFileDialog();
            sfd.ShowHelp = true;
            sfd.FileName =
"graph.grf";
            sfd.Filter =
"Graph|*.grf";
            sfd.InitialDirectory =
Directory.GetCurrentDirectory();
            sfd.RestoreDirectory =
true;


            if (sfd.ShowDialog() ==
DialogResult.OK)
            {
                try
                {


                    TextWriter file
= new StreamWriter(sfd.FileName);

file.WriteLine(graph.vertexes.Count.
ToString());
                    foreach (var v
in graph.vertexes)
                    {

file.WriteLine(v.ToString());
                    }
                    foreach (var v
in graph.adj_matrix)
                    {
                        foreach (var
j in v)

file.WriteLine(j.ToString());
                    }
                    file.Close();

log.WriteLine("Сохранено в файл [" +
sfd.FileName + ']');
                }


                catch (Exception ex)
                {

MessageBox.Show(ex.Message);
                }
```

```
            }
        }


        private void
loadToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            log.WriteLine("Попытка
загрузить из файла.");
            OpenFileDialog ofd = new
OpenFileDialog();
            ofd.ShowHelp = true;
            ofd.FileName =
"graph.grf";
            ofd.Filter =
"Graph|*.grf";
            ofd.InitialDirectory =
Directory.GetCurrentDirectory();
            ofd.RestoreDirectory =
true;


            if (ofd.ShowDialog() ==
DialogResult.OK)
            {
                try
                {
                    TextReader file
= new StreamReader(ofd.FileName);
                    int count =
int.Parse(file.ReadLine());


                    var list = new
List<Point>();
                    for (int i = 0;
i < count; i++)
                    {
                        var s =
file.ReadLine().Split(',');
                        var s1 =
s[0].Substring(3);
                        var s2 =
s[1].Substring(2);
                        int x =
int.Parse(s1);
                        int y =
int.Parse(s2.Remove(s2.Length - 1));
                        Point p =
new Point(x, y);
                        list.Add(p);
                    }


                    var result =
Enumerable.Range(0, count).Select(
                x =>
Enumerable.Range(0, count).Select(y
=> false).ToList()).ToList();


                    for (int i = 0;
i < count; i++)
                    {
                        for (int j =
0; j < count; j++)
                        {

result[i][j] =
bool.Parse(file.ReadLine());
                        }
                    }
                    file.Close();


graph.vertexes.Clear();
                    graph.vertexes =
list;

graph.adj_matrix.Clear();
                    graph.adj_matrix
= result;


                    dfs = null;
                    v_colors = null;
                    e_colors = null;
```

```
log.WriteLine("Граф был загружен из
файла [" + ofd.FileName + ']');
                    this.Refresh();
                }
                catch (Exception ex)
                {

MessageBox.Show(ex.Message);
                }
            }
        }


        private void
MainForm_FormClosing(object sender,
FormClosingEventArgs e)
        {

log.WriteLine("Завершение работы.");
            log.Close();
        }


        private void
exitToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            Close();
        }


        private void
clearField_Click(object sender,
EventArgs e)
        {
            graph.vertexes.Clear();

graph.adj_matrix.Clear();
            dfs = null;
            v_colors = null;
            e_colors= null;
            label.Text = "Поле
очищено.";
            this.Refresh();
        }
    }

}
```

```
// AdjMatrix.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.Common;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Graph
{
    public partial class AdjMatrix :
Form
    {
        public DataTable dt = new
DataTable();
        public bool changed = false;
        private int temp = -1;
        private List<List<bool>>
tmatrix;


        public
AdjMatrix(List<List<bool>> matrix)
        {
            InitializeComponent();
            tmatrix = matrix;
            dt = new DataTable();
            adj_matrix.DataError +=
new
DataGridViewDataErrorEventHandler(data
errorHandler);
        }

        private void
dataerrorHandler(object sender,
DataGridViewDataErrorEventArgs
anError)
        {
            adj_matrix.RefreshEdit();
            anError.ThrowException =
false;
        }

        private void
adj_matrix_CellValueChanged(object
sender, DataGridViewCellEventArgs e)
        {
            changed = true;
            confirm.Enabled = true;
        }
```

```
        private void
confirm_Click(object sender, EventArgs
e)
        {
            foreach
(DataGridViewColumn column in
adj_matrix.Columns)
            {
                dt.Columns.Add();
            }

            object[] cellValues = new
object[adj_matrix.Columns.Count];
            foreach (DataGridViewRow
row in adj_matrix.Rows)
            {
                for (int i = 0; i <
row.Cells.Count; i++)
                {
                    cellValues[i] =
row.Cells[i].Value;
                }

dt.Rows.Add(cellValues);
            }
            this.DialogResult =
DialogResult.OK;
            Close();
        }

        private void
cancel_Click(object sender, EventArgs
e)
        {
            changed = false;
            this.DialogResult =
DialogResult.Cancel;
            Close();
        }

        private void
adj_matrix_RowsAdded(object sender,
DataGridViewRowsAddedEventArgs e)
        {
            if (temp != -1 &&
adj_matrix.ColumnCount <
adj_matrix.RowCount-1)
            {

adj_matrix.Columns.Add('v' +
adj_matrix.ColumnCount.ToString(),
(adj_matrix.ColumnCount+1).ToString())
;

adj_matrix.Columns[adj_matrix.ColumnCo
unt-1].SortMode =
DataGridViewColumnSortMode.NotSortable
;

adj_matrix.Columns[adj_matrix.ColumnCo
unt - 1].ValueType = typeof(int);

adj_matrix.Columns[adj_matrix.ColumnCo
unt - 1].Width = 20;

adj_matrix.Rows[adj_matrix.ColumnCount
- 1].HeaderCell.Value =
adj_matrix.ColumnCount.ToString();
                for (int i = 0; i <
adj_matrix.ColumnCount; i++)
                {

adj_matrix.Rows[i].Cells[adj_matrix.Co
lumnCount - 1].Value =
adj_matrix.Rows[adj_matrix.ColumnCount
- 1].Cells[i].Value = 0;
                }
            }
        }

        private void
adj_matrix_CellEndEdit(object sender,
DataGridViewCellEventArgs e)
        {
            //if
(int.Parse(adj_matrix.Rows[e.RowIndex]
.Cells[e.ColumnIndex].Value.ToString()
) != 0 &&
int.Parse(adj_matrix.Rows[e.RowIndex].
Cells[e.ColumnIndex].Value.ToString())
!= 1)
            if
(adj_matrix.Rows[e.RowIndex].Cells[e.C
olumnIndex].Value == null)
            {

adj_matrix.Rows[e.RowIndex].Cells[e.Co
lumnIndex].Value = temp;
                return;
            }
            if
(adj_matrix.Rows[e.RowIndex].Cells[e.C
olumnIndex].Value.ToString() != "0" &&
adj_matrix.Rows[e.RowIndex].Cells[e.Co
lumnIndex].Value.ToString() != "1")
            {

adj_matrix.Rows[e.RowIndex].Cells[e.Co
lumnIndex].Value = temp;
            }
        }

        private void
adj_matrix_CellBeginEdit(object
sender,
DataGridViewCellCancelEventArgs e)
        {
            if (e.RowIndex <
adj_matrix.RowCount-1)
                temp =
int.Parse(adj_matrix.Rows[e.RowIndex].
Cells[e.ColumnIndex].Value.ToString())
;
            else temp = 0;
        }

        private void
adj_matrix_RowsRemoved(object sender,
DataGridViewRowsRemovedEventArgs e)
```

```
        {
            if (adj_matrix.ColumnCount
> 1 )
            {

adj_matrix.Columns.RemoveAt(e.RowIndex
);
            }
            else
            {
                adj_matrix.ColumnCount
= 1;
                adj_matrix.RowCount =
1;
            }
            changed = true;
            confirm.Enabled = true;
        }

        private void Form2_Load(object
sender, EventArgs e)
        {
            if (tmatrix.Count > 0)
            {
                for (int i = 0; i <
tmatrix.Count; i++)
                {

adj_matrix.Columns.Add("v" + (i +
1).ToString(), (i + 1).ToString());

adj_matrix.Rows.Add();

adj_matrix.Columns[i].SortMode =
DataGridViewColumnSortMode.NotSortable
;
                }
                Func<bool, int> func =
x => x ? 1 : 0;
                for (int i = 0; i <
tmatrix.Count; i++)
                {

adj_matrix.Columns[i].ValueType =
typeof(int);

adj_matrix.Rows[i].HeaderCell.Value =
(i+1).ToString();

adj_matrix.Columns[i].Width = 25;
                    for (int j = 0; j
< tmatrix.Count; j++)
                    {
                        try
                        {

adj_matrix.Rows[i].Cells[j].Value =
func(tmatrix[i][j]);
                        }
                        catch
(Exception) {
adj_matrix.Rows[i].Cells[j].Value = 0;
}
                    }
                }
            }
            else
            {

adj_matrix.Columns.Add("v1", "1");

adj_matrix.Columns[0].Width = 20;
                adj_matrix.Rows.Add();

adj_matrix.Rows[0].HeaderCell.Value =
"1";

adj_matrix.Columns[0].SortMode =
DataGridViewColumnSortMode.NotSortable
;

adj_matrix.Columns[0].ValueType =
typeof(int);

adj_matrix.Rows[0].Cells[0].Value = 0;

            }
            adj_matrix.RowHeadersWidth
= 51;
            this.Width =
adj_matrix.RowHeadersWidth +
adj_matrix.ColumnCount*25 + 40;
            this.Height =
(adj_matrix.RowCount + 2) * 22 + 40;
            confirm.Enabled = false;
            changed = false;
            temp = 0;
        }

        private void PasteClipboard()
        {
            try
            {
                string s =
Clipboard.GetText();
                MessageBox.Show(s);
            }

            catch (FormatException)
            {

            }
        }

    }
}


// EdgeDialogue.cs

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 Graph
{
    public partial class EdgeDialogue
: Form
    {
        public bool isDirected =
false;
        public EdgeDialogue(int v1,
int v2)
        {
            InitializeComponent();
            if (v1 != v2)
            {
                label.Text += "{" +
v1.ToString() + ", " + v2.ToString() +
"}";
                undirected.Checked =
true;
            }

            else
            {
                label.Text = "Adding
loop {" + v1.ToString() + ", " +
v2.ToString() + "}";
                EdgeChoice.Enabled =
false;
            }
        }

        private void
cancel_Click(object sender, EventArgs
e)
        {
            DialogResult =
DialogResult.Cancel;
            Close();
        }

        private void Ok_Click(object
sender, EventArgs e)
        {
            DialogResult =
DialogResult.OK;
            isDirected =
directed.Checked;
            Close();
        }
    }
}
```