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

[РЕФЕРАТ 2](#_Toc121618251)

[ВВЕДЕНИЕ 4](#_Toc121618252)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc121618253)

[1.1 Постановка задачи 5](#_Toc121618254)

[1.2 Метод решения 5](#_Toc121618257)

[2 КОНТРОЛЬНЫЙ ПРИМЕР 7](#_Toc121618258)

[2.1 Реализация контрольного примера 7](#_Toc121618259)

[2.2 Ручной просчет контрольного примера 7](#_Toc121618260)

[3 ДОПОЛНИТЕЛЬНЫЕ ПРИМЕРЫ 10](#_Toc121618261)

[3.1. Решение пустого графа 10](#_Toc121618262)

[3.2. Решение несвязного графа, полученного генерацией варианта 10](#_Toc121618264)

[3.3. Решение орграфа 10](#_Toc121618265)

[4 ОПИСАНИЕ ПРОГРАММЫ 12](#_Toc121618266)

[ВЫВОД 12](#_Toc121618267)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 14](#_Toc121618268)

[ПРИЛОЖЕНИЕ А РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 15](#_Toc121618269)

[ПРИЛОЖЕНИЕ Б ЛИСТИНИНГ ПРОГРАММЫ 16](#_Toc121618270)

# 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 фиксирован-  
ной 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, ког-  
да 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 };

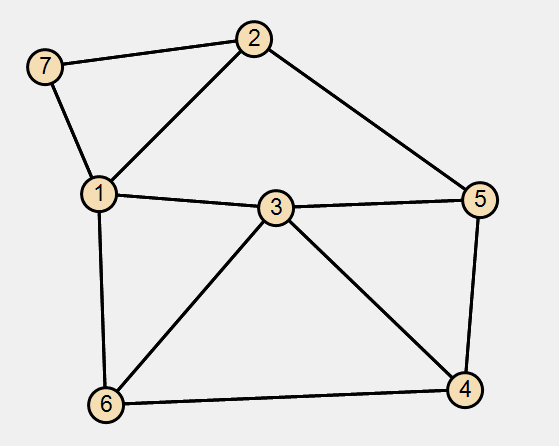
}

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»

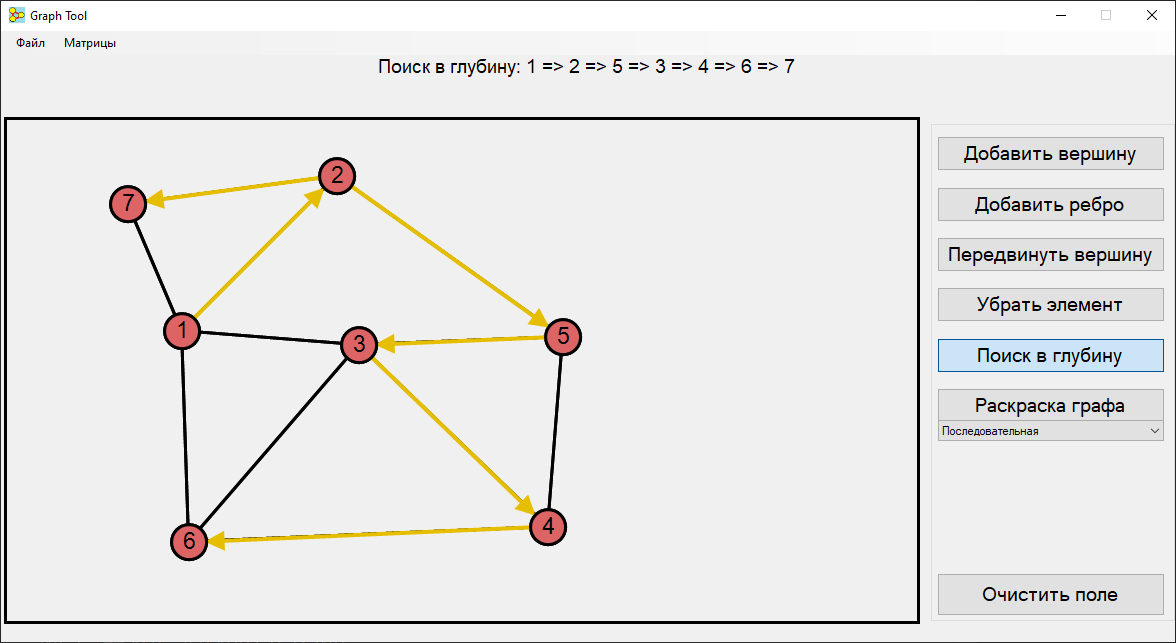


Рисунок 2.2 – Программное решение задачи

# 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»

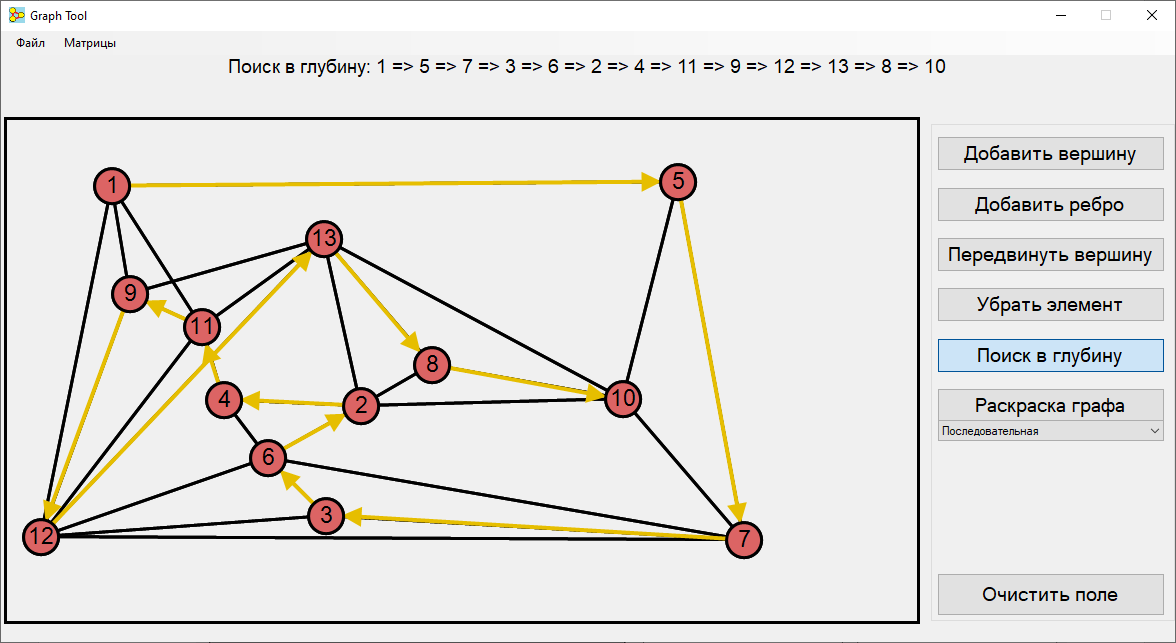


Рисунок 3.1 – Решение связного графа

## 3.3. Решение орграфа

Если граф ориентированный, то, находясь в узле x, необходимо выбирать ребро (х, у), только выходящее из х. Исследовав все ребра, выходящие из у, возвращаемся в х 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».

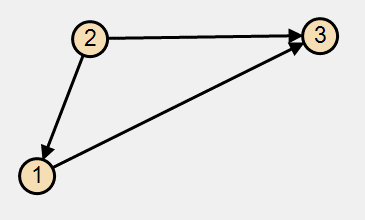


Рисунок 3.2 – Связный орграф

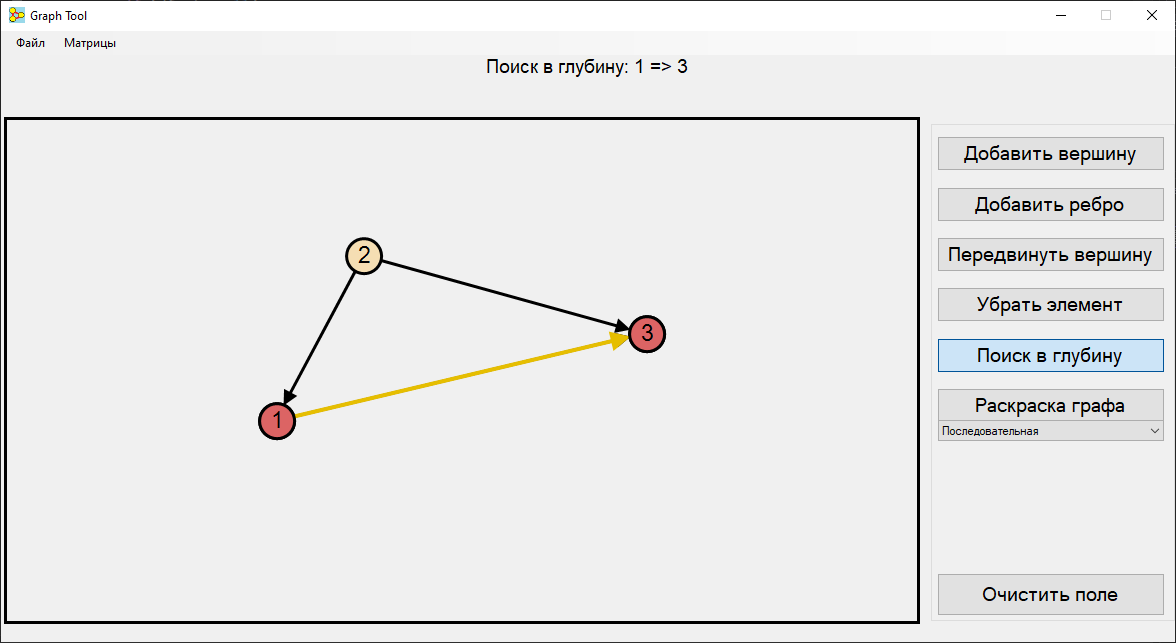


Рисунок 3.3 – Решение связного орграфа

# 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 c.
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.VisualStyleElement;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.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.AllPaintingInWmPaint, 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("GrapthTool [{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.Text.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]).ToString(), 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("Добавлена 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("Добавлено 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.Range(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(object 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(dataerrorHandler);

}

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.ColumnCount-1].SortMode = DataGridViewColumnSortMode.NotSortable;

adj\_matrix.Columns[adj\_matrix.ColumnCount - 1].ValueType = typeof(int);

adj\_matrix.Columns[adj\_matrix.ColumnCount - 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.ColumnCount - 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.ColumnIndex].Value == null)

{

adj\_matrix.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = temp;

return;

}

if (adj\_matrix.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString() != "0" && adj\_matrix.Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString() != "1")

{

adj\_matrix.Rows[e.RowIndex].Cells[e.ColumnIndex].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();

}

}

}