ПОСТАНОВКА ЗАДАЧІ

Зрівняти два неорієнтовані графи, в яких не більше 30 вершин та не більше 45 ребер в представленні FO за такою характеристикою: зрівняти їх вектори кількості повних підграфів потужності від 2 до N-1, де N це кількість вершин. Передбачити візуалізацію графа.

КОД ГОЛОВНОГО ВІКНА

using GraphLibrary;

using System;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Runtime.Serialization.Json;

using System.Windows.Forms;

namespace UI

{

public partial class MainForm : Form

{

Graph Graph1; // Граф 1

Graph Graph2; // Граф 2

byte mode1 = 1, mode2 = 1; // 1 -> Установить вершину

// 2 -> Установить ребро

// 3 -> Удалить вершины и ближние рёбра

bool is\_chosen\_first\_rib1 = false; // Выбрана ли первая вершина ребра? (Граф 1)

bool is\_chosen\_first\_rib2 = false; // Выбрана ли первая вершина ребра? (Граф 2)

ResultForm LastResultWindow;

public MainForm()

{

InitializeComponent();

Graph1 = new Graph(ProgressEqual);

Graph2 = new Graph(ProgressEqual);

StatusSaveLoad.Text = "";

SaveFile.Filter = "Json Files(\*.json)|\*.json";

OpenFile.Filter = "Json Files(\*.json)|\*.json";

}

// Универсальные методы и кнопки

private void IsGraphsEqual(object sender, EventArgs e)

{

if (Graph1.N != 0 && Graph2.N != 0)

{

LastResultStrip.Enabled = false;

Graph1.MakeFO();

Graph2.MakeFO();

bool result; // Результат сравнения

Stopwatch sw = new Stopwatch(); // Секундомер времени на сравнение

sw.Start();

if (Graph1.IsEqualTo(Graph2))

{

sw.Stop();

ResultLabel.ForeColor = Color.Green;

ResultLabel.Text = "✔ Графы равны\n\n";

result = true;

}

else

{

sw.Stop();

ResultLabel.ForeColor = Color.Red;

ResultLabel.Text = "✖ Графы НЕ равны\n\n";

result = false;

}

if(Graph1.N == Graph2.N) // Если кол-во вершин не равно, то графы заведомо не равны, поэтому подробных результатов нет

{

LastResultWindow = new ResultForm(Graph1.VectorOfRibs, Graph2.VectorOfRibs, sw, result);

LastResultStrip.Enabled = true; // Включить вкладку "Last Result"

}

ResultStatusTimer.Enabled = true;

}

} // Проверка графов на эквивалентность

public Graph LoadAll(string path)

{

try

{

DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer(typeof(Graph));

FileStream buffer = File.OpenRead(path);

return jsonSerializer.ReadObject(buffer) as Graph;

}

catch

{

MessageBox.Show("Не удалось загрузить файл графа.\n" +

"Возможно он повреждён или удалён.\n" +

"Попробуйте снова, или создайте граф заново.", "Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

return null;

}

} // Десериализация графа из памяти

public void SaveAll(Graph graph, string path)

{

DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer(typeof(Graph));

FileStream buffer = File.Create(path);

jsonSerializer.WriteObject(buffer, graph);

buffer.Close();

} // Сериализация графа из памяти

private void LoadSatusOff(object sender, EventArgs e)

{

StatusSaveLoad.Text = "";

LoadInfoTimer.Enabled = false;

} // Скрыть сообщение об успешной загрузки графа из памяти

private void ResultStatusOff(object sender, EventArgs e)

{

ResultLabel.Text = "";

ResultStatusTimer.Enabled = false;

} // Скрыть результат о сравнении графов

private void ShowAboutWindow(object sender, EventArgs e)

{

AboutProgram ap = new AboutProgram();

ap.ShowDialog();

} // Показать окно "О программе"

private void ShowLastSuccessfulResult(object sender, EventArgs e)

{

LastResultWindow.ShowDialog();

} // Показать окно подробностей результата срвнения

// Кнопки и методы для поля 1

private void Choose\_Top1(object sender, EventArgs e)

{

mode1 = 1;

button\_top1.BackColor = Color.LawnGreen;

button\_rib1.BackColor = Color.White;

button\_delete1.BackColor = Color.White;

} // Выбрать режим "Установить новую вершину"

private void Choose\_Rib1(object sender, EventArgs e)

{

mode1 = 2;

button\_top1.BackColor = Color.White;

button\_rib1.BackColor = Color.LawnGreen;

button\_delete1.BackColor = Color.White;

} // Выбрать режим "Установить новое ребро"

private void Choose\_Del1(object sender, EventArgs e)

{

mode1 = 3;

button\_top1.BackColor = Color.White;

button\_rib1.BackColor = Color.White;

button\_delete1.BackColor = Color.Red;

} // Выбрать режим "Удалить вершину и прилежащие рёбра"

private void Clear\_Graph1(object sender, EventArgs e)

{

GroupOfGraph1.Text = "";

Graph1 = new Graph(ProgressEqual);

Graph\_Zone1.Refresh();

array1.Text = "";

} // Очистка поля и памяти от текущего графа 1

private void SaveGraph1(object sender, EventArgs e)

{

SaveFile.FileName = "Graph1.json";

if (SaveFile.ShowDialog() == DialogResult.Cancel)

return;

SaveAll(Graph1, SaveFile.FileName);

} // Сохранить граф 1

private void LoadGraph1(object sender, EventArgs e)

{

OpenFile.FileName = "Graph1.json";

if (OpenFile.ShowDialog() == DialogResult.Cancel)

return;

Graph1 = LoadAll(OpenFile.FileName);

if (Graph1 == null)

return;

Graph\_Zone1.Refresh();

array1.Text = Graph1.ToString();

GroupOfGraph1.Text = OpenFile.FileName;

StatusSaveLoad.Text = $"✔ Файл \"{OpenFile.FileName} \" загружен успешно";

LoadInfoTimer.Enabled = true;

Graph1.bar = ProgressEqual;

} // Загрузить граф 1

private void Refresh\_Graph1(object sender, PaintEventArgs e)

{

if (Graph1 == null)

return;

Graphics g = e.Graphics;

Pen pen = new Pen(Color.Black, 3);

for (int i = 0; i < Graph1.Ribs.Count; i++)

g.DrawLine(pen, Graph1.Ribs[i].X1, Graph1.Ribs[i].Y1, Graph1.Ribs[i].X2, Graph1.Ribs[i].Y2);

for (int i = 0; i < Graph1.Tops.Count; i++)

{

if (Graph1.Tops[i].chosen)

{

Pen help\_pen = new Pen(Color.Yellow, 3);

g.DrawEllipse(help\_pen, Graph1.Tops[i].X - 1, Graph1.Tops[i].Y - 1, 27, 27);

help\_pen = new Pen(Color.Black, 2);

g.DrawEllipse(help\_pen, Graph1.Tops[i].X - 3, Graph1.Tops[i].Y - 3, 30, 30);

}

g.FillEllipse(Brushes.Black, Graph1.Tops[i].X, Graph1.Tops[i].Y, 24, 24);

g.DrawString(Graph1.Tops[i].name.ToString(), button\_top1.Font, Brushes.White, Graph1.Tops[i].X + 4, Graph1.Tops[i].Y + 5);

}

} // Событие "Paint" для графа 1

private void DoAction1(object sender, MouseEventArgs e)

{

if (mode1 == 1 && Graph1.Tops.Count < 10) // Установить вершину (до 10 штук)

{

if (Graph1.IsThereTop(e.X, e.Y) == -1)

{

Graph1.AddTop(e.X, e.Y); // Добавить новую вершину

Graph\_Zone1.Refresh();

}

}

else if (mode1 == 2) // Установить ребро

{

if (!is\_chosen\_first\_rib1) // Если не выбрано превой вершины

{

int top\_name = Graph1.IsThereTop(e.X, e.Y);

if (top\_name != -1)

{

Graph1.AddFirstTopOfRib(top\_name);

is\_chosen\_first\_rib1 = true;

}

}

else

{

int top\_name = Graph1.IsThereTop(e.X, e.Y);

if (top\_name != -1)

{

Graph1.AddSecondTopOfRib(top\_name);

is\_chosen\_first\_rib1 = false;

}

}

Graph\_Zone1.Refresh();

}

else if (mode1 == 3) // Удалить вершину и ближние рёбра

{

int top\_name = Graph1.IsThereTop(e.X, e.Y);

if (top\_name != -1)

Graph1.DeleteTop(top\_name);

if (Graph1.Tops.Count == 0)

array1.Text = "";

Graph\_Zone1.Refresh();

}

MakeFO1(sender, e);

} // Совершить действие относительно выбранного режима (Граф 1)

private void MakeFO1(object sender, EventArgs e)

{

array1.Text = Graph1.ToString();

} // Преобразование графа 1 в вид FO

private void Random\_Graph1(object sender, EventArgs e)

{

GroupOfGraph1.Text = "";

Graph1 = new Graph(ProgressEqual);

Graph1.MakeRandomGraph((byte)numericrandom1.Value);

MakeFO1(sender, e);

Graph\_Zone1.Refresh();

} // Случайная генерация графа 1

// Кнопки и методы для поля 2

private void Choose\_Top2(object sender, EventArgs e)

{

mode2 = 1;

button\_top2.BackColor = Color.LawnGreen;

button\_rib2.BackColor = Color.White;

button\_delete2.BackColor = Color.White;

} // Выбрать режим "Установить новую вершину"

private void Choose\_Rib2(object sender, EventArgs e)

{

mode2 = 2;

button\_top2.BackColor = Color.White;

button\_rib2.BackColor = Color.LawnGreen;

button\_delete2.BackColor = Color.White;

} // Выбрать режим "Установить новое ребро"

private void Choose\_Del2(object sender, EventArgs e)

{

mode2 = 3;

button\_top2.BackColor = Color.White;

button\_rib2.BackColor = Color.White;

button\_delete2.BackColor = Color.Red;

} // Выбрать режим "Удалить вершину и прилежащие рёбра"

private void Clear\_Graph2(object sender, EventArgs e)

{

GroupOfGraph2.Text = "";

Graph2 = new Graph(ProgressEqual);

Graph\_Zone2.Refresh();

array2.Text = "";

} // Очистка поля и памяти от текущего графа 2

private void SaveGraph2(object sender, EventArgs e)

{

SaveFile.FileName = "Graph2.json";

if (SaveFile.ShowDialog() == DialogResult.Cancel)

return;

SaveAll(Graph2, SaveFile.FileName);

} // Сохранить граф 2

private void LoadGraph2(object sender, EventArgs e)

{

OpenFile.FileName = "Graph2.json";

if (OpenFile.ShowDialog() == DialogResult.Cancel)

return;

Graph2 = LoadAll(OpenFile.FileName);

if (Graph2 == null)

return;

Graph\_Zone2.Refresh();

array2.Text = Graph2.ToString();

GroupOfGraph2.Text = OpenFile.FileName;

StatusSaveLoad.Text = $"✔ Файл \"{OpenFile.FileName} \" загружен успешно";

LoadInfoTimer.Enabled = true;

Graph2.bar = ProgressEqual;

} // Загрузить граф 2

private void Refresh\_Graph2(object sender, PaintEventArgs e)

{

if (Graph2 == null)

return;

Graphics g = e.Graphics;

Pen pen = new Pen(Color.Black, 3);

for (int i = 0; i < Graph2.Ribs.Count; i++)

g.DrawLine(pen, Graph2.Ribs[i].X1, Graph2.Ribs[i].Y1, Graph2.Ribs[i].X2, Graph2.Ribs[i].Y2);

for (int i = 0; i < Graph2.Tops.Count; i++)

{

if (Graph2.Tops[i].chosen)

{

Pen help\_pen = new Pen(Color.Yellow, 3);

g.DrawEllipse(help\_pen, Graph2.Tops[i].X - 1, Graph2.Tops[i].Y - 1, 27, 27);

help\_pen = new Pen(Color.Black, 2);

g.DrawEllipse(help\_pen, Graph2.Tops[i].X - 3, Graph2.Tops[i].Y - 3, 30, 30);

}

g.FillEllipse(Brushes.Black, Graph2.Tops[i].X, Graph2.Tops[i].Y, 24, 24);

g.DrawString(Graph2.Tops[i].name.ToString(), button\_top2.Font, Brushes.White, Graph2.Tops[i].X + 4, Graph2.Tops[i].Y + 5);

}

} // Событие "Paint" для графа 2

private void DoAction2(object sender, MouseEventArgs e)

{

if (mode2 == 1 && Graph2.Tops.Count < 10) // Установить вершину (до 10 штук)

{

if (Graph2.IsThereTop(e.X, e.Y) == -1)

{

Graph2.AddTop(e.X, e.Y); // Добавить новую вершину

Graph\_Zone2.Refresh();

}

}

else if (mode2 == 2) // Установить ребро

{

if (!is\_chosen\_first\_rib2) // Если не выбрано превой вершины

{

int top\_name = Graph2.IsThereTop(e.X, e.Y);

if (top\_name != -1)

{

Graph2.AddFirstTopOfRib(top\_name);

is\_chosen\_first\_rib2 = true;

}

}

else

{

int top\_name = Graph2.IsThereTop(e.X, e.Y);

if (top\_name != -1)

{

Graph2.AddSecondTopOfRib(top\_name);

is\_chosen\_first\_rib2 = false;

}

}

Graph\_Zone2.Refresh();

}

else if (mode2 == 3) // Удалить вершину и ближние рёбра

{

int top\_name = Graph2.IsThereTop(e.X, e.Y);

if (top\_name != -1)

Graph2.DeleteTop(top\_name);

if (Graph2.Tops.Count == 0)

array2.Text = "";

Graph\_Zone2.Refresh();

}

MakeFO2(sender, e);

} // Совершить действие относительно выбранного режима (Граф 2)

private void MakeFO2(object sender, EventArgs e)

{

array2.Text = Graph2.ToString();

} // Преобразование графа 2 в вид FO

private void Random\_Graph2(object sender, EventArgs e)

{

GroupOfGraph2.Text = "";

Graph2 = new Graph(ProgressEqual);

Graph2.MakeRandomGraph((byte)numericrandom2.Value);

MakeFO2(sender, e);

Graph\_Zone2.Refresh();

} // Случайная генерация графа 2

}

}

КОД БІБЛІОТЕКИ КЛАСІВ

using System;

using System.Collections.Generic;

using System.Runtime.Serialization;

using System.Windows.Forms;

namespace GraphLibrary

{

[Serializable]

public class Graph

{

public List<int> GraphInFO; // Граф в представлении FO

public byte N; // Кол-во вершин

public List<Top> Tops; // Коллекция вершин

public List<Rib> Ribs; // Коллекция рёбер

byte top\_name; // Номер порядковый номер вершин

int X\_firstpointrib; // Кордината X начала ребра

int Y\_firstpointrib; // Кордината Y начала ребра

int name\_firsttopinrib; // Имя первой крайной вершины в ребре

int firsttop\_index; // Индекс первой вершины ребра

[NonSerialized]

public uint[] VectorOfRibs; // Вектор подграфов графа

[NonSerialized]

public ProgressBar bar; // Визуализация процесса сравнения графов

public Graph(ProgressBar progress)

{

bar = progress;

Tops = new List<Top>();

Ribs = new List<Rib>();

top\_name = 1;

}

// Методы по обработке графа

public void MakeFO()

{

GraphInFO = new List<int>();

if (Tops.Count > 1 && Ribs.Count > 0)

{

for (int a = 0; a < Tops.Count; a++)

{

for (int b = 0; b < Tops[a].Connected.Count; b++)

GraphInFO.Add(Tops[a].Connected[b]);

if (a != Tops.Count - 1)

GraphInFO.Add(0);

}

}

} // Делает граф в представлении FO

public void AddTop(int X, int Y)

{

Tops.Add(new Top(X - 12, Y - 12, top\_name++));

N++;

} // Добавляет новую вершину

public void AddFirstTopOfRib(int name)

{

name\_firsttopinrib = name;

firsttop\_index = IndexOfTop(name);

X\_firstpointrib = Tops[firsttop\_index].X + 12;

Y\_firstpointrib = Tops[firsttop\_index].Y + 12;

Tops[firsttop\_index].chosen = true;

} // Сохранить в памяти первую верину в ребре

public void AddSecondTopOfRib(int name)

{

if (name\_firsttopinrib != name)

{

int secondindex = IndexOfTop(name);

Rib newrib = new Rib(name\_firsttopinrib, name, X\_firstpointrib, Y\_firstpointrib, Tops[secondindex].X + 12, Tops[secondindex].Y + 12);

if (!IsExist(newrib))

{

Ribs.Add(newrib);

Tops[secondindex].Connected.Add(name\_firsttopinrib);

Tops[firsttop\_index].Connected.Add(name);

}

}

Tops[firsttop\_index].chosen = false;

} // Добавить ребро в память, на основе первой вершины в памяти и второй вершины (вторая вершина - аргумент)

public void DeleteTop(int name)

{

Tops.RemoveAt(IndexOfTop(name)); // Удалить вершину

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

if (Ribs[i].top\_from == name || Ribs[i].top\_to == name)

{

Ribs.RemoveAt(i);

i--;

}

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

if (Tops[i].Connected.Contains(name))

Tops[i].Connected.Remove(name);

if (Tops.Count > 0)

top\_name = (byte)(Tops[Tops.Count - 1].name + 1);

else

top\_name = 1;

N--;

} // Удаляет выбранную вершины и все подключенные рёбра

public bool IsEqualTo(Graph second\_graph)

{

bar.Value = 0; // Обнуление прогрессбара

bar.Maximum = (N \* (N - 3)) \* 2 + N + 1; // Нахождение максимального значение в прогресс баре

bar.Step = 1; // Шаг для прогрессбара

// Переопределение графа в виде FO в удобный для обработки вид

if (N != second\_graph.N) // Если кол-во вершин не совпадает, то графы точно не равны

{

bar.Value = bar.Maximum;

return false;

}

List<int>[] ListOfHightsCurr = new List<int>[N + 1]; // Массив коллекций где номеру вершини соответвуют вершини, с которыми она свзяна ребром

List<int>[] ListOfHightsArg = new List<int>[N + 1]; // Массив коллекций где номеру вершини соответвуют вершини, с которыми она свзяна ребром (аргумент)

byte index\_in\_FO1 = 0; // Индекс указателя в представлении FO

byte index\_in\_FO2 = 0; // Индекс указателя в представлении FO (аргумент)

for (byte name\_of\_top = 1; name\_of\_top <= N; name\_of\_top++, index\_in\_FO1++, index\_in\_FO2++)

{

ListOfHightsCurr[name\_of\_top] = new List<int>(); // Инициализация коллекций под номером вершины

ListOfHightsArg[name\_of\_top] = new List<int>(); // Инициализация коллекций под номером вершины (аргумент)

while (index\_in\_FO1 != GraphInFO.Count && GraphInFO[index\_in\_FO1] != 0)

{

ListOfHightsCurr[name\_of\_top].Add(GraphInFO[index\_in\_FO1]);

index\_in\_FO1++;

}

while (index\_in\_FO2 != second\_graph.GraphInFO.Count && second\_graph.GraphInFO[index\_in\_FO2] != 0)

{

ListOfHightsArg[name\_of\_top].Add(second\_graph.GraphInFO[index\_in\_FO2]);

index\_in\_FO2++;

}

bar.PerformStep();

}

List<byte> Used\_Tops = new List<byte>();

uint count = 0; // Кол-во подграфов

byte count\_of\_cycles = 0; // Количество рекурсивно вызваных на данный момент методов

VectorOfRibs = new uint[N]; // Вектор подграфов графа

second\_graph.VectorOfRibs = new uint[N]; // Вектор подграфов графа (аршумент)

for (uint max\_power = 2; max\_power < N - 1; max\_power++) // Проверяем мощность от 2 до количества вершин

for (byte last\_top = 1; last\_top <= N; last\_top++) // Начальная вершина для подграфа

{

SearchLastTop(ref VectorOfRibs, ref count\_of\_cycles, ref Used\_Tops, last\_top, ref ListOfHightsCurr, ref count, max\_power);

bar.PerformStep();

}

for (byte max\_power = 2; max\_power < N - 1; max\_power++) // Проверяем мощность от 2 до количества вершин

for (byte last\_top = 1; last\_top <= N; last\_top++) // Начальная вершина для подграфа

{

SearchLastTop(ref second\_graph.VectorOfRibs, ref count\_of\_cycles, ref Used\_Tops, last\_top, ref ListOfHightsArg, ref count, max\_power);

bar.PerformStep();

}

if (IsMatrixesAreEqual(VectorOfRibs, second\_graph.VectorOfRibs))

{

bar.PerformStep();

return true;

}

else

{

bar.PerformStep();

return false;

}

} // Сравнивает графы, возвращает 'true' если они равны, и 'false' если нет

public void MakeRandomGraph(byte hights)

{

Random rand = new Random();

for (int i = 0; i < hights; i++)

AddTop(rand.Next(500), rand.Next(500)); // Генерация вершин и их кординат

for (int i = 0; i < hights + hights \* 0.5; i++)

{

int random\_first = rand.Next(1, Tops.Count + 1); // Генерация первой вершины в ребре

AddFirstTopOfRib(random\_first);

int random\_second = -1;

while ((random\_first == random\_second) || random\_second == -1) // Генерация второй вершины в ребре

random\_second = rand.Next(1, Tops.Count + 1); //

AddSecondTopOfRib(random\_second);

}

} // Создание случайного графа

// Вспомогательные методы

public int IsThereTop(int X, int Y)

{

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

if (Math.Sqrt((X - Tops[i].X) \* (X - Tops[i].X) + (Y - Tops[i].Y) \* (Y - Tops[i].Y)) <= 24) // Если ты нажал на вершину, которая уже существует

return Tops[i].name;

return -1;

} // Возвращает индекс вершины, нажатой мышью, или -1, если в этом месте нет вершин

public override string ToString()

{

string str = "";

for (int a = 0; a < Tops.Count; a++)

{

Tops[a].Connected.Sort();

for (int b = 0; b < Tops[a].Connected.Count; b++)

str += Tops[a].Connected[b] + " ";

if (a != Tops.Count - 1)

str += ", ";

}

return str;

} // Перегрузка метода ToString() для вывода грава в виде FO на экран

// Служебные методы

private int IndexOfTop(int name)

{

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

if (Tops[i].name == name)

return i;

return -1;

} // Ищет индекс вершины в коллекции по её индексу

private bool IsExist(Rib newrib)

{

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

if (Ribs[i].top\_from == newrib.top\_from && Ribs[i].top\_to == newrib.top\_to && Ribs[i].X1 == newrib.X1 && Ribs[i].X2 == newrib.X2 && Ribs[i].Y1 == newrib.Y1 && Ribs[i].Y2 == newrib.Y2 && Ribs[i].lenght == newrib.lenght || Ribs[i].top\_from == newrib.top\_to && Ribs[i].top\_to == newrib.top\_from && Ribs[i].X1 == newrib.X2 && Ribs[i].X2 == newrib.X1 && Ribs[i].Y1 == newrib.Y2 && Ribs[i].Y2 == newrib.Y1 && Ribs[i].lenght == newrib.lenght)

return true;

return false;

} // Проверка на существование уже такого ребра

private void SearchLastTop(ref uint[] VectorOfRibs, ref byte count\_of\_cycles, ref List<byte> Used\_Tops, byte pre\_top, ref List<int>[] Tops\_In\_Fo, ref uint count, uint power)

{

count\_of\_cycles++; // Увеличение вызванного на данный момент метода "SearchLastTop"

Used\_Tops.Add(pre\_top); // "pre\_top" вершина -> использована

for (byte new\_top = 1; new\_top <= N; new\_top++)

{

if (Tops\_In\_Fo[new\_top].Contains(pre\_top) && !Used\_Tops.Contains(new\_top)) // Если вершини "new\_top" и "pre\_top" имеют соединяющее ребро, то...

{

if (count\_of\_cycles < power) // Рекурсивный вызов функции "SearchLastTop" пока их число меньше текущей требуемой мощности

{

pre\_top = new\_top; // Найденная вершина становится "pre\_top", и ищется следующая "new\_top" вершина

SearchLastTop(ref VectorOfRibs, ref count\_of\_cycles, ref Used\_Tops, pre\_top, ref Tops\_In\_Fo, ref count, power);

pre\_top = Used\_Tops[Used\_Tops.Count - 1];

}

else

{

Used\_Tops.Add(new\_top); // "new\_top" вершина -> использована

VectorOfRibs[power]++;

}

}

}

count\_of\_cycles--;

Used\_Tops = CopyCollection(Used\_Tops, count\_of\_cycles); // Из колеекции "Used\_Tops" оставляем первые "count\_of\_cycles" элементов

} // Рекурсивный цикл для поиска подграфов

private List<byte> CopyCollection(List<byte> Collection, uint length)

{

List<byte> BufferCollection = new List<byte>();

for (int i = 0; i < length; i++)

BufferCollection.Add(Collection[i]);

return BufferCollection;

} // Возвращает подстроку длиной "lenght" первых символов из коллекции "Collection"

private bool IsMatrixesAreEqual(uint[] Matrix1, uint[] Matrix2)

{

for (uint a = 0; a < N; a++)

if (Matrix1[a] != Matrix2[a])

return false;

return true;

} // Сравнивает две матрицы

// Служебные классы

[DataContract]

public class Rib

{

public Rib(int top1, int top2, int x1, int y1, int x2, int y2)

{

X1 = x1;

Y1 = y1;

X2 = x2;

Y2 = y2;

top\_from = top1;

top\_to = top2;

LengthBetweenDots();

}

[DataMember]

public int X1, X2, Y1, Y2, top\_from, top\_to;

[DataMember]

public double lenght;

private void LengthBetweenDots()

{

lenght = Math.Sqrt((X1 - X2) \* (X1 - X2) + (Y1 - Y2) \* (Y1 - Y2));

}

} // Обьект "Ребро"

[DataContract]

public class Top

{

public Top(int x, int y, int num)

{

X = x;

Y = y;

this.name = num;

}

[DataMember]

public int X, Y, name;

[DataMember]

public bool chosen = false;

[DataMember]

public List<int> Connected = new List<int>();

} // Обьект "Вершина"

}

}

КОД ВІКНА ДОПОМОГИ

using System.Drawing;

using System.Windows.Forms;

namespace GraphLibrary

{

public partial class AboutProgram : Form

{

public AboutProgram()

{

InitializeComponent();

infolabel.Text = "- Для создания новой вершины, нажмите на кнопку \"Top\", а затем на место, где хотите установить вершину (Нумерация автоматическая).\n\n" +

"- Для установки ребра, нажмите на кнопку \"Rib\", а затем поочередно нажмите на две вершины, которые хотите соеденить.\n\n" +

"- Для удаления вершины, нажмите кнопку \"DEL\", а затем на вершину, которую хотите удалить (ребра, которые соединяют эту вершину также удаляются).\n\n" +

"- Для полной очистки вашего графа, нажмите на кнопку \"CL\".\n\n" +

"- Для того, что бы сгенерировать граф, выберите его размер под клавишей \"RND\", а затем нажмите на клавишу \"RND\".\n\n" +

"- Для сохранения и загрузки вашего графа используются кнопки \"Save\" и \"Load\" соответственно.\n\n" +

"- Для сравнения графов используется кнопка \"Compare\".\n" +

" - Процесс сравнения будет показан в виде полосы загрузки под кнопкой \"Compare\";\n" +

" - По завершению сравнения результат будет показан под полосой прогресса;\n" +

" - Что-бы посмотреть подробности последнего \*успешного результата сравнения, \n" +

" нажмите на верхнее меню \"Last Successful Result\".\n\n" +

"- Граф в виде FO будет показан под рабочей областью.\n\n\n" +

"\* - Под успешным, подразумевается что кол-во вершин сравниваемых графов было равно";

} // Конструктор окна, который выводит информацию в окно

}

}

КОД ВІКНА ПОДРОБИЦЬ РЕЗУЛЬТАТУ

using System.Diagnostics;

using System.Drawing;

using System.Windows.Forms;

namespace UI

{

public partial class ResultForm : Form

{

public ResultForm(uint[] Vector1, uint[] Vector2, Stopwatch sw, bool result)

{

InitializeComponent();

for (int i = 2; i < Vector1.Length; i++) // Заполнение таблиц результатами

{

ResultView1.Rows.Add(i, Vector1[i]);

ResultView2.Rows.Add(i, Vector2[i]);

}

if (result)

{

ResultAndTimeLanel.ForeColor = Color.Green; // Если результат положительный

ResultAndTimeLanel.Text = "✔ Графы равны\n\n"; //

}

else

{

ResultAndTimeLanel.ForeColor = Color.Red; // Если результат отрицательный

ResultAndTimeLanel.Text = "✖ Графы НЕ равны\n\n"; //

}

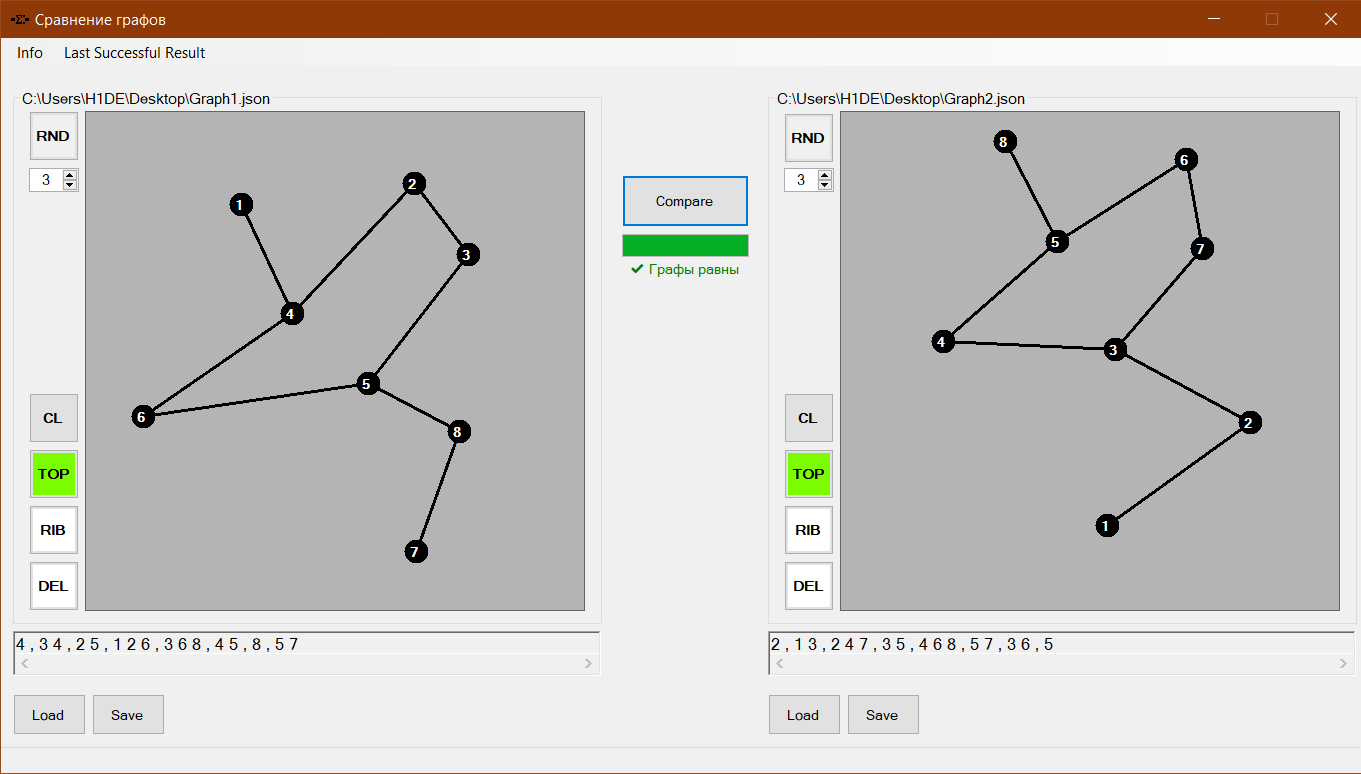
ResultAndTimeLanel.Text += "Потрачено " + sw.Elapsed.ToString(); // Вывод потраченного времени

}

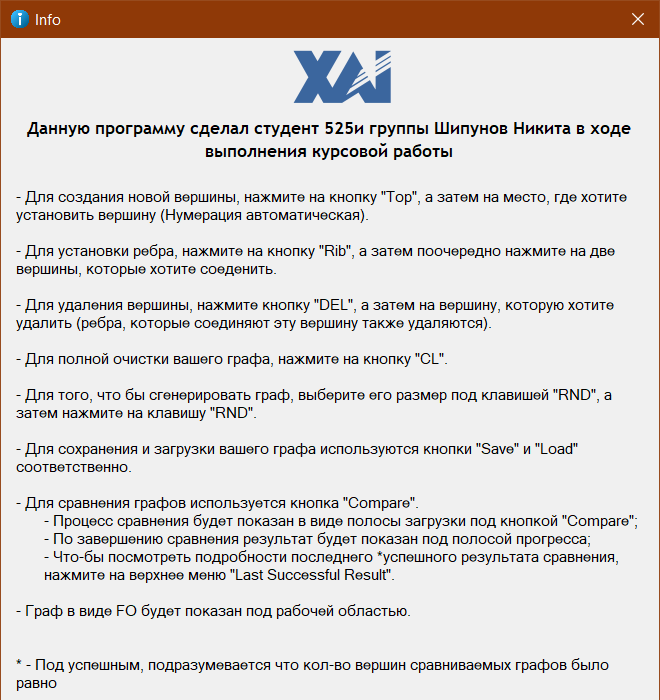
}

}

СРІНШОТ ГОЛОВНОГО ВІКНА



СРІНШОТ ВІКНА “ІНФО”



СРІНШОТ ВІКНА ПОДРОБИЦЬ РЕЗУЛЬТАТУ

