**Задание:**

Разработать программу, работающую в среде Visual Studio на основе Windows Forms и реализующую конкретное задание.

Программа должна обеспечивать ввод описания графа из файла и с клавиатуры.

Программа должна обеспечивать редактирование и сохранение описания графа.

Программа сначала должна найти и запомнить все пути и циклы, а потом показывать их по запросу.

Найти все циклы в графе, проходящие не более чем через две вершины центра графа(стек\_масс,граф\_масс)

**Описание работы программы:**

private void button1\_Click(object sender, EventArgs e) - считывает данные введенные пользователем и вызывает Gr.FindMinCycle(A,B); для поиска циклов в графе, а потом вызывает функцию для отрисовки путей и циклов.

private void button2\_Click(object sender, EventArgs e) - после вывода всех циклов графа, отслеживает куда нажал пользователь в DataGriedView и вызывает функцию для отрисовки цикла с заданным номером.

public void FindMinCycle(int[] A, int[] B) - ищет цикл в графе, путем обхода его вершин, вызывая CheckCycle(Trace, A) для проверки наличия цикла и не добавляя в массив циклов одинаковые циклы, с помощью использования массива из 0 и 1 для каждого цикла(вершина есть - 1, вершины нет - )

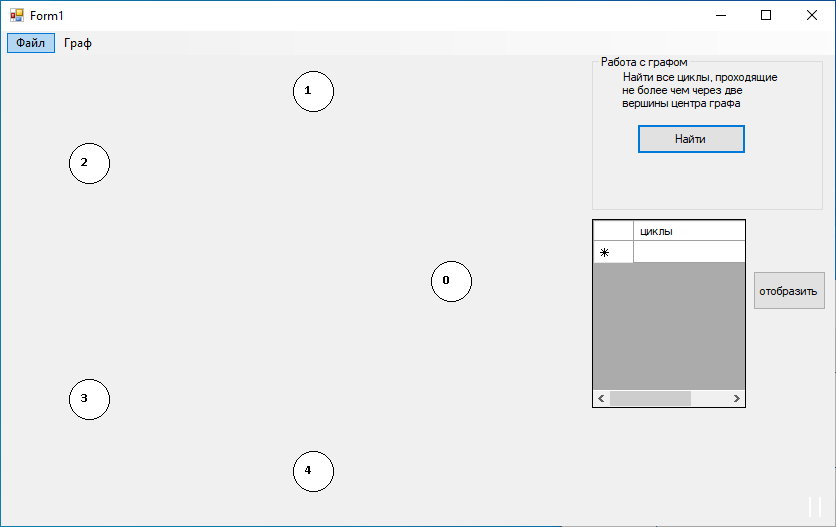
bool CheckCycle(int[] C, int[] A) - вызывается из FindMinCycle и проверяет является ли данный массив массивом вершин цикла графа

void countEccentricities() – вызывается из FindMinCycle(int[] A, int[] B) для поиска эксцентриситетов графа

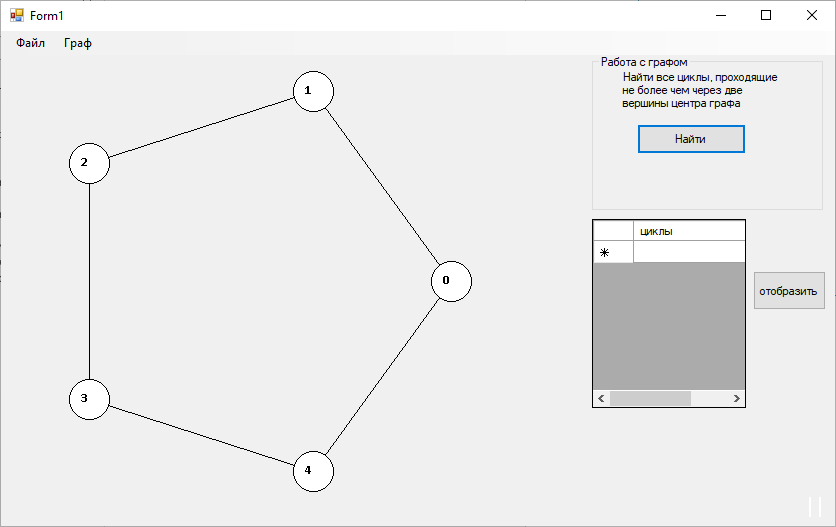
int existsArc(int a, int b) – проверяет смежность вершин a и b графа

**Тесты:**

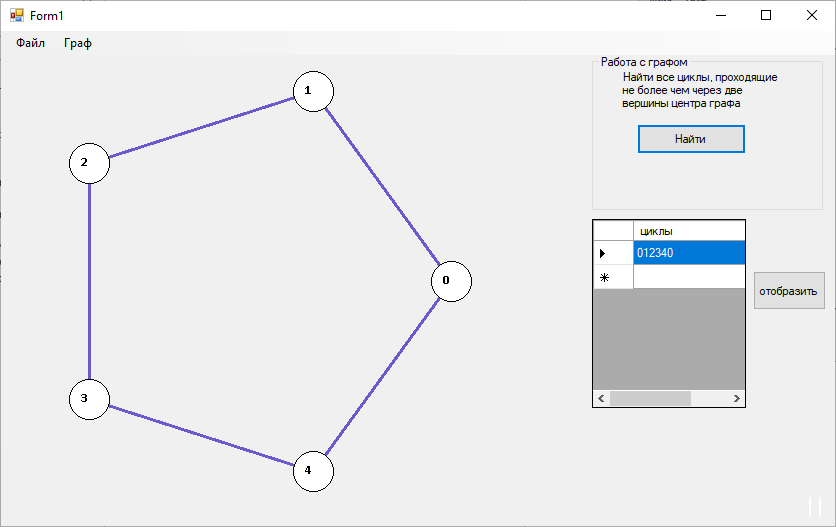
Ввод графа



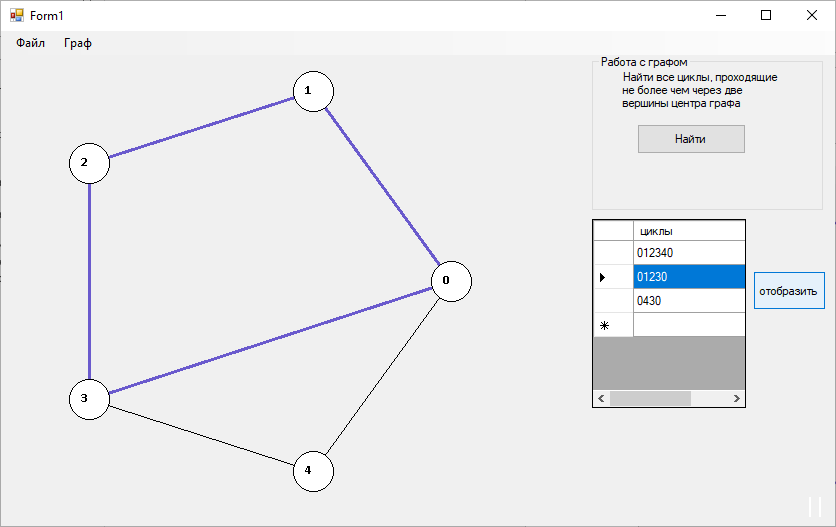
Соединение вершин

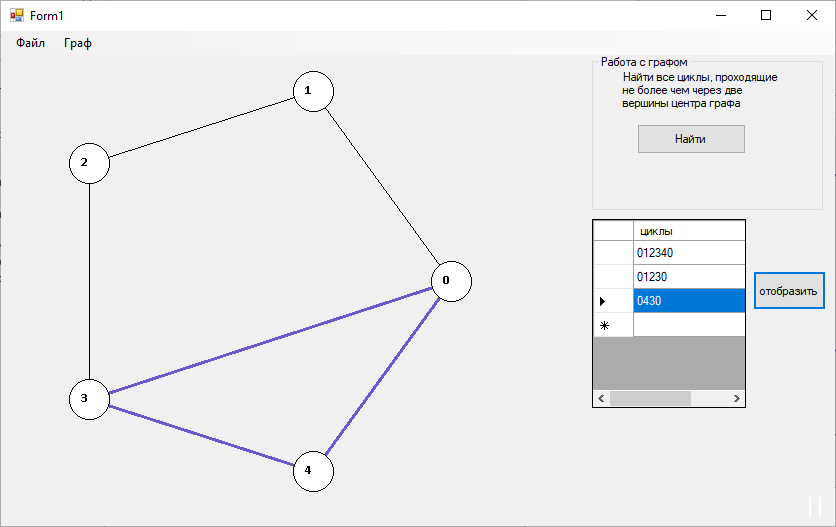


Вывод циклов графа(Тест 1). Центральных вершин нет

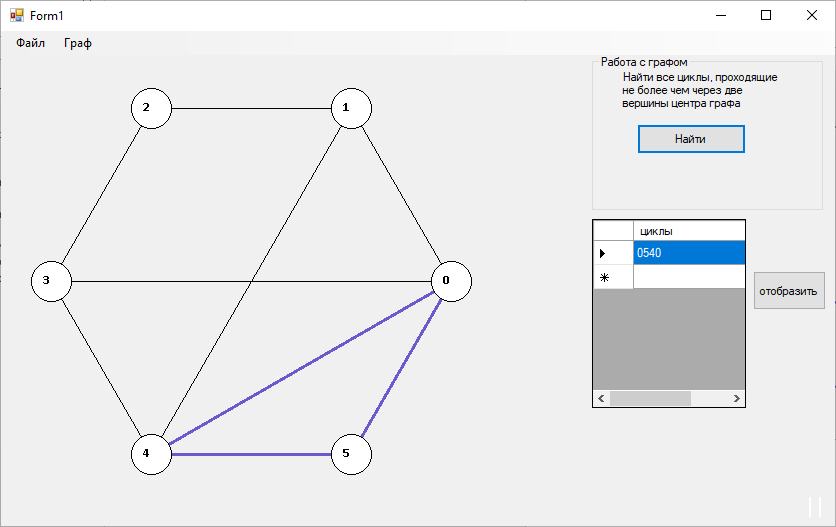


Тест 2(центральных вершин нет)





Тест 3( центральные вершины: 0, 1, 3, 4)



**Листинг:**

Graph.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows.Forms;

//using System.Windows.Forms;

namespace Lab3\_SppO

{

class node

{

public int vert;

public node next;

};

class Stack

{

node t; // node t

int n;

public Stack()

{

t = null;

n = 0;

}

public void push(int element)

{

node temp = new node();

temp.vert = element;

temp.next = t;

t = temp;

n++;

}

public int pop()

{

int temp;

node x;

x = t;

temp = x.vert;

t = t.next;

n--;

return temp;

}

public int top()

{

return t.vert;

}

public bool IsEmpty()

{

if (n == 0)

return true;

else

return false;

}

public int[] StackToVector()

{

node temp = t;

int[] v = new int[n];

int i = n - 1;

while (temp != null)

{

v[i] = temp.vert;

temp = temp.next;

i--;

}

return v;

}

};

abstract class Graph

{

protected int s;

protected Stack stack;

public int[] MinTrace;

public int MinSize;

public int[] MinCycle;

public int MinCycleSize;

public int[,] Trace22 = new int[30, 20]; //\*

public int ske = 0;

public int[] eccentricity = new int[20]; // 20 = N

public int diameter; // диаметр графа

public int center; // радиус графа

public int arcCount; //

public Graph()

{

MinSize = -1;

s = 0;

stack = new Stack();

MinCycleSize = -1;

}

public Graph(int size)

{

MinSize = -1;

s = size;

stack = new Stack();

MinCycleSize = -1;

}

public int size()

{

return s;

}

public int[] FindMinTrace(int vFrom, int vTo)

{

string s4;

MinSize = -1;

MinCycleSize = -1;

int[] M = new int[s];

int[] Trace;

int vert = vFrom;

while (!stack.IsEmpty()) // возможно то же сделать для функции цикла

{

stack.pop();

}

stack.push(vFrom);

int start = -1;

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

{

M[i] = 0;

}

M[vFrom] = 1;

while (!stack.IsEmpty())

{

vert = NextVertex(stack.top(), start, M);

if (vert == -1)

{

start = stack.pop();

M[start] = 0;

continue;

}

if (vert == vTo)

{

Trace = stack.StackToVector();

if ((MinSize == -1) || (Trace.Length < MinSize))

if (CheckTrace(Trace))

{

if (MinSize == -1)

{

MinSize = Trace.Length + 1;

MinTrace = new int[MinSize];

Trace.CopyTo(MinTrace, 0);

MinTrace[MinSize - 1] = vTo;

}

if ((MinSize > 0) & (Trace.Length < MinSize))

{

MinSize = Trace.Length + 1;

MinTrace = new int[MinSize];

Trace.CopyTo(MinTrace, 0);

MinTrace[MinSize - 1] = vTo;

}

}

start = vTo;

continue;

}

stack.push(vert);

start = -1;

M[vert] = 1;

}

return MinTrace;

}

abstract public int NextVertex(int v, int start, int[] M);

abstract public bool CheckTrace(int[] T);

};

class Graph\_ext : Graph

{

node[] List;

public Graph\_ext() : base() { }

public Graph\_ext(int n)

: base(n)

{

List = new node[n];

}

public void FindMinCycle(int[] A, int[] B, int[] Ex)

{

string s2;

bool flag2 = false;

int vert;

int[] M;

string Bool = "";

string[] Bool2 = new string[100];

bool flag = true; // если false то не добавляем элемент в массив массивов

M = new int[s];

int inde = 0; // индекс Trace2 для добавления в него Trace

int[] Trace;

string s4 = "";

string s5 = "";

countEccentricities();

MessageBox.Show("диаметр графа: " + diameter);

MessageBox.Show("радиус графа: " + center);

int xx = 0;

while (xx < s)

{

//MessageBox.Show("эксентриситет " + xx + ": " + eccentricity[xx]);

xx += 1;

}

int ij = 0;

int f = 0;

if (center == diameter)

{

Ex[0] = -1;

}

else

{

while (ij < s)

{

if (eccentricity[ij] == center) // if(eccentricity[C[i]] == center)

{

Ex[f] = ij;

f += 1;

}

ij += 1;

}

}

Stack st = new Stack();

for (int k = 0; k < s; k++)

{

st.push(k);

int start = -1;

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

{

M[i] = 0;

}

M[k] = s + 1;

while (!st.IsEmpty())

{

vert = NextVertex2(st.top(), start, M);

if (vert == -1)

{

start = st.pop();

M[start] = 0;

continue;

}

if (vert == k)

{

Trace = st.StackToVector(); // цикл

if (((MinCycleSize == -1 || Trace.Length < 10)) && CheckCycle2(Trace))

{

// добавить цикл Trace в массив циклов

MinCycleSize = Trace.Length + 1;

MinCycle = new int[MinCycleSize];

Trace.CopyTo(MinCycle, 0);

MinCycle[MinCycleSize - 1] = k;

// функция которая не добавляет в массив массивов одинаковые циклы

Bool = "";

flag = true;

//MessageBox.Show("gg");

for (int g = 0; g < 10; g++)

{

flag2 = false;

foreach (int gg in MinCycle) // Trace

{

if (gg == g)

{

Bool = Bool + "1";

flag2 = true;

break;

}

}

if (flag2 == false)

{

Bool = Bool + "0";

}

}

s5 = "";

foreach (int kk in MinCycle)

{

s4 = kk.ToString();

s5 = s5 + s4;

}

if (inde > 0)

{

foreach (string bb in Bool2)

{

if (bb == Bool)

{

flag = false;

break;

}

}

}

if (flag == true)

{

Bool2[inde] = Bool;

for (int h = 0; h < MinCycleSize; h++)

{

Trace22[inde, h] = MinCycle[h];

}

inde += 1;

s2 = inde.ToString();

}

}

if (MinCycleSize != -1)

{

Trace.CopyTo(B, 0); //

B[MinCycleSize - 1] = k; //

}

start = k;

continue;

}

M[vert] = st.top();

st.push(vert);

start = -1;

}

}

s2 = inde.ToString();

}

void countEccentricities()

{

diameter = 0;

center = 20;

for (int i = 0; i < s; i++) // s - число вершин(вместо nodeCount)

{

eccentricity[i] = countEccentricity(i);

if (eccentricity[i] > diameter)

{

diameter = eccentricity[i];

}

if (eccentricity[i] < center)

{

center = eccentricity[i];

}

}

}

int countEccentricity(int in2)

{

int v, j, f, r;

int[] m = new int[s];

int[] q = new int[s];

for (int jj = 0; jj < s; jj++)

{

m[jj] = 0;

}

f = -1;

r = 0;

q[r] = in2;

m[in2] = 1;

while (f != r)

{

f += 1;

v = q[f];

for (j = 0; j < s; j++)

{

if (existsArc(v, j) != 0)

{

if (m[j] == 0)

{

r += 1;

q[r] = j;

m[j] = m[v] + 1;

}

}

}

}

v = q[r];

return m[v] - 1;

}

int existsArc(int a, int b)

{

//MessageBox.Show("existsArc ");

node[] List2 = new node[s];

Array.Copy(List, List2, List.Length); // копирование из массива List в массив List2

while (List2[a] != null)

{

if (List2[a].vert == b)

return 1;

else

List2[a] = List2[a].next;

}

return 0;

}

int NextVertex2(int v, int start, int[] M)

{

node temp;

temp = List[v];

if (temp == null)

return -1;

if (start != -1)

while (temp != null)

{

if (temp.vert == start)

{

temp = temp.next;

break;

}

else

temp = temp.next;

}

while (temp != null)

{

if ((M[temp.vert] == 0 || M[temp.vert] == s + 1) && M[v] != temp.vert)

return temp.vert;

else

temp = temp.next;

}

return -1;

}

public void Edge(int i, int j)

{

node temp = new node();

temp.vert = j;

temp.next = null;

node t = List[i];

if (t == null)

List[i] = temp;

else

{

if (t.vert == j)

return;

while (t.next != null)

{

if (t.vert == j)

return;

t = t.next;

}

if (t.vert == j)

return;

t.next = temp;

}

temp = new node();

temp.vert = i;

temp.next = null;

t = List[j];

if (t == null)

List[j] = temp;

else

{

while (t.next != null)

t = t.next;

t.next = temp;

}

}

public void DelEdge(int i, int j)

{

node t = List[i];

node temp;

if (t == null)

return;

if (t.vert == j)

{

List[i] = t.next;

}

else

{

while (t.next != null)

{

if (t.next.vert == j)

{

temp = t.next;

t.next = t.next.next;

break;

}

t = t.next;

}

}

t = List[j];

if (t == null)

return;

if (t.vert == i)

{

List[j] = t.next;

}

else

{

while (t.next != null)

{

if (t.next.vert == i)

{

temp = t.next;

t.next = t.next.next;

break;

}

t = t.next;

}

}

}

public override int NextVertex(int v, int start, int[] M)

{

node temp;

temp = List[v];

if (temp == null)

return -1;

if (start != -1)

while (temp != null)

{

if (temp.vert == start)

{

temp = temp.next;

break;

}

else

temp = temp.next;

}

while (temp != null)

{

if (M[temp.vert] == 0)

return temp.vert;

else

temp = temp.next;

}

return -1;

}

bool CheckCycle2(int[] C)

{

int i = 0;

int f = 0;

int n = C.Length; // длина цикла(число вершин)

if (center == diameter)

{

return true;

}

while (i < n) // (i < n)

{

if (eccentricity[C[i]] == center)

{

if (f < 2)

{

f++;

}

else

{

return false;

}

}

i += 1;

}

return true;

}

public override bool CheckTrace(int[] T)

{

string s3;

int j2 = 0;

foreach (int j3 in T)

{

j2 += 1;

}

s3 = j2.ToString();

return true;

}

public int[] ListToVector(int i)

{

node t;

int[] v = new int[0];

int c = 0;

t = List[i];

while (t != null)

{

c++;

t = t.next;

}

if (c == 0)

return v;

v = new int[c];

t = List[i];

c = 0;

while (t != null)

{

v[c] = t.vert;

t = t.next;

c++;

}

return v;

}

}

}

Drawing.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Windows.Forms;

namespace Lab3\_SppO

{

struct Vertex

{

public int x;

public int y;

public int select;

public int num;

public int[] smej;

public void Draw(System.Drawing.Graphics g)

{

SolidBrush hBrushVertex, hBrushSelect1, hBrushSelect2, textBrush;

Pen pen = new Pen(Color.Black);

hBrushSelect1 = new SolidBrush(Color.FromArgb(0, 0, 255));

hBrushSelect2 = new SolidBrush(Color.FromArgb(255, 0, 0));

hBrushVertex = new SolidBrush(Color.White);

textBrush = new SolidBrush(Color.Black);

if (select == 1)

{

g.FillEllipse(hBrushSelect1, x - 20, y - 20, 40, 40);

g.DrawEllipse(pen, x - 20, y - 20, 40, 40);

}

if (select == 2)

{

g.FillEllipse(hBrushSelect2, x - 20, y - 20, 40, 40);

g.DrawEllipse(pen, x - 20, y - 20, 40, 40);

}

if (select == 0)

{

g.FillEllipse(hBrushVertex, x - 20, y - 20, 40, 40);

g.DrawEllipse(pen, x - 20, y - 20, 40, 40);

}

Font font = new Font("Calibri", 10, System.Drawing.FontStyle.Bold);

g.DrawString(num.ToString(), font, textBrush, x - 10, y - 10);

}

}

class Drawing

{

const int G\_LENGTH = 200;

static public Vertex [] CreateImageGraph(Graph\_ext gr)

{

int N = gr.size();

Vertex [] v = new Vertex [N];

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

{

v[i].x = Convert.ToInt32(250+G\_LENGTH\*Math.Cos((double)i\*(2\*Math.PI/N)));

v[i].y = Convert.ToInt32(250-G\_LENGTH\*Math.Sin((double)i\*(2\*Math.PI/N)));

v[i].select = 0;

v[i].smej = gr.ListToVector(i);

v[i].num = i;

}

return v;

}

static public void DrawGraph(System.Drawing.Graphics g,Vertex [] gr,Graph\_ext graph)

{

int N = gr.Length;

int n;

Pen pen = new Pen(Color.Black);

Point p1 = new Point();

Point p2 = new Point();

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

{

n = gr[i].smej.Length;

for(int j = 0;j < n;j++)

{

p1.X = gr[i].x;

p1.Y = gr[i].y;

p2.X = gr[gr[i].smej[j]].x;

p2.Y = gr[gr[i].smej[j]].y;

g.DrawLine(pen,p1,p2);

}

}

DrawPath(g, gr, graph);

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

gr[i].Draw(g);

}

static public bool Selection(int x,int y,Vertex [] igr,out int sel1,out int sel2,int type)

{

int N = igr.Length;

sel1 = -1;

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

if(igr[i].select > 0)

sel1 = i;

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

{

if( (x-igr[i].x)\*(x-igr[i].x)+(y-igr[i].y)\*(y-igr[i].y) < 1600 )

{

sel2 =i;

if(sel1 == sel2)

{

igr[i].select = 0;

return false;

}

if(sel1 != -1)

{

return true;

}

else

{

igr[i].select = type;

return false;

}

}

}

sel2 = 0;

return false;

}

static void DrawPath(Graphics g, Vertex[] gr, Graph\_ext graph)

{

string s1;

string s2;

int i = 0;

int fin = graph.Trace22[graph.ske,0];

int poi = -1;

int N2; // максимальное количество вершин

Point p1 = new Point();

Point p2 = new Point();

// отрисовывает циклы

Pen pen = new Pen(Color.SlateBlue, 3); // (Color.SlateBlue, 3);

while(fin != poi)

{

i += 1;

poi = graph.Trace22[graph.ske,i];

}

N2 = i+1;

if (N2 > 0)

{

p1.X = gr[graph.Trace22[graph.ske,0]].x;

p1.Y = gr[graph.Trace22[graph.ske,0]].y;

for (int k = 1; k < N2; k++)

{

p2.X = gr[graph.Trace22[graph.ske,k]].x;

p2.Y = gr[graph.Trace22[graph.ske,k]].y;

g.DrawLine(pen, p1, p2);

p1 = p2;

}

}

}

}

}

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Lab3\_SppO

{

public partial class Form1 : Form

{

Graph\_ext Gr;

Vertex[] GrI;

int N = 0;

public Form1()

{

InitializeComponent();

}

private void menuStrip1\_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

{

}

private void Form1\_Paint(object sender, PaintEventArgs e)

{

if(N > 0)

Drawing.DrawGraph(e.Graphics, GrI, Gr);

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void Form1\_MouseDown(object sender, MouseEventArgs e)

{

int sel1, sel2;

if (e.Button == System.Windows.Forms.MouseButtons.Left)

{

if (Drawing.Selection(e.X, e.Y, GrI,out sel1,out sel2,1))

{

Gr.Edge(sel1, sel2);

GrI = Drawing.CreateImageGraph(Gr);

}

}

else

if (Drawing.Selection(e.X, e.Y, GrI, out sel1, out sel2, 2))

{

Gr.DelEdge(sel1, sel2);

GrI = Drawing.CreateImageGraph(Gr);

}

Invalidate();

}

private void создатьГрафToolStripMenuItem\_Click(object sender, EventArgs e)

{

Form2 f = new Form2();

f.ShowDialog();

N = f.n;

Gr = new Graph\_ext(N);

GrI = Drawing.CreateImageGraph(Gr);

Invalidate();

}

private void textBox5\_TextChanged(object sender, EventArgs e)

{

}

private void label5\_Click(object sender, EventArgs e)

{

}

private void textBox4\_TextChanged(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

this.dataGridView1.Rows.Clear();

int Len;

int[] A;

int[] B = new int[10];

int[,] Trace2 = new int[30,20]; // \*

string s;

string s2;

string s3 = "";

int vv = 0;

int j = 1;

int inda = 0;

int a,b;

textBox1.Text = "1";

textBox2.Text = "2";

int[] Ex = new int[20];

try

{

a = Convert.ToInt32(textBox1.Text);

b = Convert.ToInt32(textBox2.Text);

}

catch

{

return;

}

A = Gr.FindMinTrace(a, b);

foreach (int i in A)

{

s = i.ToString();

}

Gr.FindMinCycle(A,B,Ex);

int povt = 0;

int ff = 0;

if (Ex[0] == -1)

MessageBox.Show("центральных вершин нет");

else

{

while (ff < Ex.Length)

{

if (Ex[ff] == 0)

povt += 1;

if (povt >= 2)

break;

MessageBox.Show("центральная вершина: " + Ex[ff]);

ff += 1;

}

}

Trace2 = Gr.Trace22;

foreach (int k2 in Trace2)

{

vv += 1;

s2 = k2.ToString();

}

Len = Gr.MinCycleSize;

while ((Trace2[inda, 0].ToString() != "0") | (Trace2[inda, 1].ToString() != "0"))

{

s3 = "";

j = 1;

s2 = Trace2[inda,0].ToString();

s3 = s3 + s2;

while (Trace2[inda,j].ToString() != "0")

{

s2 = Trace2[inda,j].ToString();

s3 = s3 + s2;

j += 1;

}

if ((Trace2[inda,j].ToString() == "0") & (Trace2[inda,0].ToString() == "0"))

{

s2 = Trace2[inda,j].ToString();

s3 = s3 + s2;

j += 1;

}

this.dataGridView1.Rows.Add(s3);

inda += 1;

}

GrI = Drawing.CreateImageGraph(Gr);

Invalidate();

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void label3\_Click(object sender, EventArgs e)

{

}

private void groupBox1\_Enter(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

Gr.ske = dataGridView1.SelectedCells[0].RowIndex;

GrI = Drawing.CreateImageGraph(Gr);

Invalidate();

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

}

}