Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

# Лабораторная работа " Задача коммивояжера."

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь, 2024

Вариант 8.

Постановка задачи:

Общая:

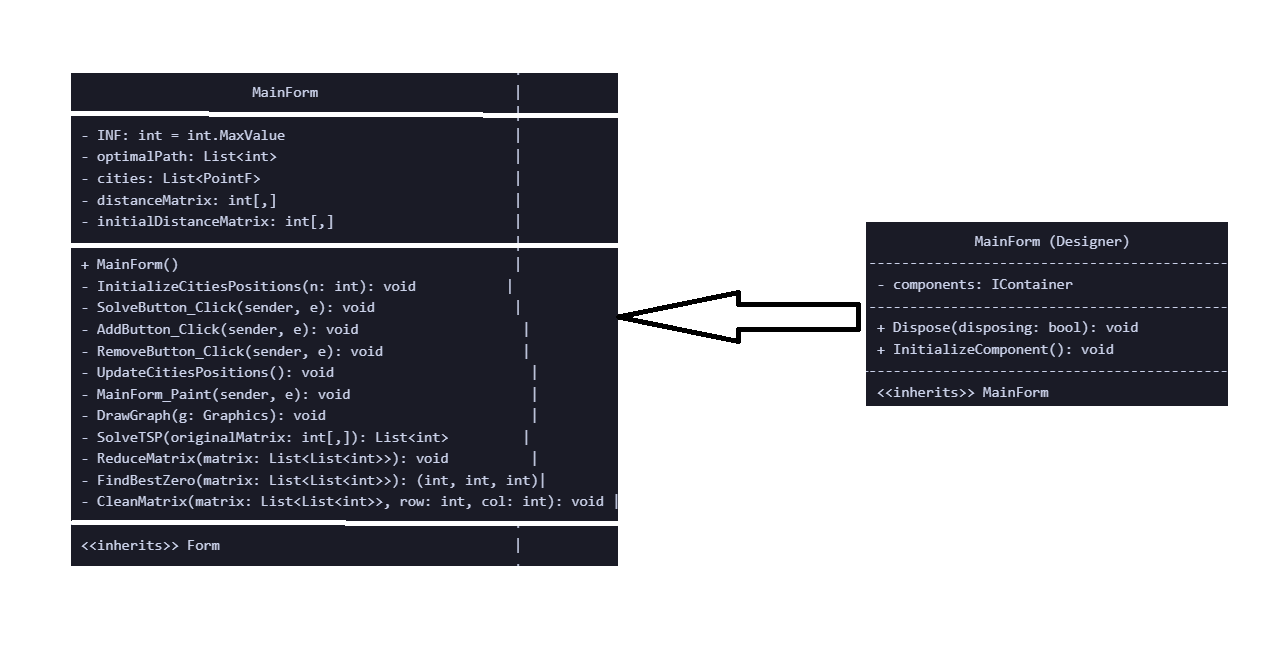
Имеется N городов. Выезжая из исходного города А1, коммивояжер должен побывать во всех городах по одному разу и вернуться в город А1. Задача заключается в определении последовательности объезда городов, при которой коммивояжеру требуется минимизировать некоторый критерий эффективности: стоимость проезда, время пути, суммарное расстояние.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого. Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.

Персональная:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| нн | 27 | 43 | 16 | 30 | 16 |
| 7 | нн | 16 | 1 | 30 | 25 |
| 20 | 13 | нн | 35 | 5 | 0 |
| 21 | 16 | 25 | нн | 18 | 18 |
| 12 | 46 | 27 | 48 | нн | 5 |
| 23 | 5 | 5 | 9 | 5 | нн |

UML – диаграмма:



Программная реализация:

Program.cs:

using System;

using System.Windows.Forms;

namespace TSPVisualization

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

MainForm.cs:

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using System.Drawing.Drawing2D;

namespace TSPVisualization

{

public partial class MainForm : Form

{

private int INF = int.MaxValue;

private List<int> optimalPath;

private List<PointF> cities = new List<PointF>();

private int[,] distanceMatrix;

// Начальная матрица расстояний (6 городов)

private int[,] initialDistanceMatrix = {

{ int.MaxValue, 27, 43, 16, 30, 16 },

{ 7, int.MaxValue, 16, 1, 30, 25 },

{ 20, 13, int.MaxValue, 35, 5, 0 },

{ 21, 16, 25, int.MaxValue, 18, 18 },

{ 12, 46, 27, 48, int.MaxValue, 5 },

{ 23, 5, 5, 9, 5, int.MaxValue }

};

public MainForm()

{

InitializeComponent();

this.Paint += MainForm\_Paint;

btnSolve.Click += SolveButton\_Click;

btnAddVertex.Click += AddButton\_Click;

btnRemoveVertex.Click += RemoveButton\_Click;

InitializeCitiesPositions(initialDistanceMatrix.GetLength(0));

distanceMatrix = initialDistanceMatrix;

}

// Инициализация позиций городов по кругу

private void InitializeCitiesPositions(int n)

{

cities.Clear();

float centerX = this.ClientSize.Width / 2f;

float centerY = this.ClientSize.Height / 2f;

float radius = Math.Min(centerX, centerY) - 50;

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

{

double angle = 2 \* Math.PI \* i / n;

float x = centerX + radius \* (float)Math.Cos(angle);

float y = centerY + radius \* (float)Math.Sin(angle);

cities.Add(new PointF(x, y));

}

}

// Обновление позиций городов при добавлении/удалении

private void UpdateCitiesPositions()

{

int n = cities.Count;

float centerX = this.ClientSize.Width / 2f;

float centerY = this.ClientSize.Height / 2f;

float radius = Math.Min(centerX, centerY) - 50;

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

{

double angle = 2 \* Math.PI \* i / n;

cities[i] = new PointF(centerX + radius \* (float)Math.Cos(angle),

centerY + radius \* (float)Math.Sin(angle));

}

}

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

{

if (int.TryParse(txtStartVertex.Text, out int startIndex) && startIndex >= 1 && startIndex <= cities.Count)

{

startIndex -= 1; // Преобразуем в 0-индекс

(optimalPath, int totalDistance) = SolveTSP(distanceMatrix, startIndex);

txtOptimalDistance.Text = $"Оптимальное расстояние: {totalDistance}"; // Отображаем расстояние

pictureBox.Invalidate(); // Перерисовать pictureBox

}

else

{

MessageBox.Show("Пожалуйста, введите корректный номер стартовой вершины (от 1 до " + cities.Count + ").");

}

}

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

{

int newCityIndex = cities.Count;

// Создаем новую матрицу расстояний с размером +1

int[,] newDistanceMatrix = new int[newCityIndex + 1, newCityIndex + 1];

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

{

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

{

newDistanceMatrix[i, j] = distanceMatrix[i, j];

}

newDistanceMatrix[i, newCityIndex] = INF;

newDistanceMatrix[newCityIndex, i] = INF; // Здесь можно добавить логику для вычисления расстояния

}

newDistanceMatrix[newCityIndex, newCityIndex] = int.MaxValue;

distanceMatrix = newDistanceMatrix;

cities.Add(new PointF(0, 0)); // Временная позиция, обновим ниже

UpdateCitiesPositions();

pictureBox.Invalidate(); // Перерисовать pictureBox

}

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

{

if (cities.Count == 0)

return;

int cityIndexToRemove = cities.Count - 1;

int newSize = cities.Count - 1;

if (newSize == 0)

{

cities.Clear();

distanceMatrix = new int[0, 0];

optimalPath = null;

txtOptimalDistance.Text = string.Empty; // Очищаем текстовое поле

pictureBox.Invalidate(); // Перерисовать pictureBox

return;

}

int[,] newDistanceMatrix = new int[newSize, newSize];

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

{

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

{

int sourceI = i < cityIndexToRemove ? i : i + 1;

int sourceJ = j < cityIndexToRemove ? j : j + 1;

newDistanceMatrix[i, j] = distanceMatrix[sourceI, sourceJ];

}

}

distanceMatrix = newDistanceMatrix;

cities.RemoveAt(cityIndexToRemove);

UpdateCitiesPositions();

txtOptimalDistance.Text = string.Empty; // Очищаем текстовое поле

pictureBox.Invalidate(); // Перерисовать pictureBox

}

// Метод решения задачи коммивояжера с заданной стартовой вершиной (полный перебор, т.к. мало городов)

private (List<int>, int) SolveTSP(int[,] dist, int start)

{

int n = dist.GetLength(0);

List<int> vertices = Enumerable.Range(0, n).Where(v => v != start).ToList();

List<int> bestPath = null;

int bestCost = int.MaxValue;

Permute(vertices, 0, vertices.Count - 1, (perm) =>

{

int cost = 0;

int prev = start;

foreach (int v in perm)

{

if (dist[prev, v] == INF)

{

cost = INF;

break;

}

cost += dist[prev, v];

prev = v;

}

// Возврат в стартовую вершину

if (cost != INF && dist[prev, start] != INF)

cost += dist[prev, start];

else

cost = INF;

if (cost < bestCost)

{

bestCost = cost;

bestPath = new List<int> { start };

bestPath.AddRange(perm);

bestPath.Add(start);

}

});

return (bestPath, bestCost);

}

// Рекурсивный перебор перестановок

private void Permute(List<int> list, int l, int r, Action<List<int>> action)

{

if (l == r)

{

action(new List<int>(list));

}

else

{

for (int i = l; i <= r; i++)

{

Swap(list, l, i);

Permute(list, l + 1, r, action);

Swap(list, l, i);

}

}

}

private void Swap(List<int> list, int i, int j)

{

int temp = list[i];

list[i] = list[j];

list[j] = temp;

}

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

{

// Этот метод можно удалить, если вы перенесли логику рисования в pictureBox\_Paint

}

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

{

Graphics g = e.Graphics;

g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

g.Clear(Color.White);

if (cities.Count == 0)

return;

Pen edgePen = new Pen(Color.LightGray, 1);

Pen pathPen = new Pen(Color.Red, 3);

Brush cityBrush = Brushes.Blue;

Brush textBrush = Brushes.Black;

Font font = new Font("Arial", 10);

AdjustableArrowCap bigArrow = new AdjustableArrowCap(5, 5);

edgePen.CustomEndCap = bigArrow;

pathPen.CustomEndCap = bigArrow;

// Чтобы избежать наложения подписей, запомним уже занятые позиции подписей

List<RectangleF> labelRects = new List<RectangleF>();

// Рисуем все ребра (серые) с расстояниями и стрелками

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

{

for (int j = 0; j < cities.Count; j++)

{

if (i != j && distanceMatrix[i, j] != INF && distanceMatrix[i, j] != int.MaxValue)

{

PointF p1 = cities[i];

PointF p2 = cities[j];

// Рисуем ребро со стрелкой

g.DrawLine(edgePen, p1, p2);

// Рассчитаем позицию для текста — немного смещаем от середины ребра по нормали

float dx = p2.X - p1.X;

float dy = p2.Y - p1.Y;

float length = (float)Math.Sqrt(dx \* dx + dy \* dy);

if (length == 0) continue;

// Нормализованные векторы

float nx = -dy / length;

float ny = dx / length;

// Позиция текста — середина ребра со смещением по нормали

float textX = (p1.X + p2.X) / 2 + nx \* 15;

float textY = (p1.Y + p2.Y) / 2 + ny \* 15;

string distStr = distanceMatrix[i, j].ToString();

SizeF textSize = g.MeasureString(distStr, font);

RectangleF textRect = new RectangleF(textX - textSize.Width / 2, textY - textSize.Height / 2, textSize.Width, textSize.Height);

// Проверяем пересечения с уже нарисованными подписями, если есть — сдвигаем вниз

bool intersects;

int offset = 0;

do

{

intersects = false;

foreach (var rect in labelRects)

{

if (rect.IntersectsWith(textRect))

{

intersects = true;

offset += (int)textSize.Height + 2;

textRect.Y += textSize.Height + 2;

break;

}

}

} while (intersects);

labelRects.Add(textRect);

g.DrawString(distStr, font, textBrush, textRect.Location);

}

}

}

// Рисуем оптимальный путь красным, если есть, со стрелками

if (optimalPath != null)

{

for (int i = 0; i < optimalPath.Count - 1; i++)

{

int from = optimalPath[i];

int to = optimalPath[i + 1];

PointF p1 = cities[from];

PointF p2 = cities[to];

g.DrawLine(pathPen, p1, p2);

}

}

// Рисуем города

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

{

PointF city = cities[i];

float radius = 8;

g.FillEllipse(cityBrush, city.X - radius, city.Y - radius, radius \* 2, radius \* 2);

g.DrawString((i + 1).ToString(), font, textBrush, city.X + radius, city.Y - radius);

}

}

}

}

MainFormDesigner.cs:

namespace TSPVisualization

{

partial class MainForm

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

private void InitializeComponent()

{

pictureBox = new PictureBox();

btnSolve = new Button();

btnAddVertex = new Button();

btnRemoveVertex = new Button();

txtStartVertex = new TextBox();

label7 = new Label();

txtOptimalDistance = new TextBox();

label8 = new Label();

((System.ComponentModel.ISupportInitialize)pictureBox).BeginInit();

SuspendLayout();

//

// pictureBox

//

pictureBox.BackColor = Color.White;

pictureBox.Location = new Point(13, 18);

pictureBox.Margin = new Padding(4, 5, 4, 5);

pictureBox.Name = "pictureBox";

pictureBox.Size = new Size(1073, 671);

pictureBox.TabIndex = 0;

pictureBox.TabStop = false;

pictureBox.Paint += pictureBox\_Paint;

//

// btnSolve

//

btnSolve.Location = new Point(49, 714);

btnSolve.Margin = new Padding(4, 5, 4, 5);

btnSolve.Name = "btnSolve";

btnSolve.Size = new Size(133, 35);

btnSolve.TabIndex = 1;

btnSolve.Text = "Решить";

btnSolve.UseVisualStyleBackColor = true;

//

// btnAddVertex

//

btnAddVertex.Location = new Point(243, 714);

btnAddVertex.Margin = new Padding(4, 5, 4, 5);

btnAddVertex.Name = "btnAddVertex";

btnAddVertex.Size = new Size(133, 35);

btnAddVertex.TabIndex = 2;

btnAddVertex.Text = "Добавить вершину";

btnAddVertex.UseVisualStyleBackColor = true;

//

// btnRemoveVertex

//

btnRemoveVertex.Location = new Point(426, 714);

btnRemoveVertex.Margin = new Padding(4, 5, 4, 5);

btnRemoveVertex.Name = "btnRemoveVertex";

btnRemoveVertex.Size = new Size(133, 35);

btnRemoveVertex.TabIndex = 3;

btnRemoveVertex.Text = "Удалить вершину";

btnRemoveVertex.UseVisualStyleBackColor = true;

//

// txtStartVertex

//

txtStartVertex.Location = new Point(953, 722);

txtStartVertex.Margin = new Padding(4, 5, 4, 5);

txtStartVertex.Name = "txtStartVertex";

txtStartVertex.Size = new Size(65, 27);

txtStartVertex.TabIndex = 13;

//

// label7

//

label7.AutoSize = true;

label7.Location = new Point(782, 722);

label7.Margin = new Padding(4, 0, 4, 0);

label7.Name = "label7";

label7.Size = new Size(147, 20);

label7.TabIndex = 20;

label7.Text = "Стартовая вершина";

//

// txtOptimalDistance

//

txtOptimalDistance.Location = new Point(897, 691);

txtOptimalDistance.Margin = new Padding(4, 5, 4, 5);

txtOptimalDistance.Name = "txtOptimalDistance";

txtOptimalDistance.ReadOnly = true;

txtOptimalDistance.Size = new Size(200, 27);

txtOptimalDistance.TabIndex = 21;

//

// label8

//

label8.AutoSize = true;

label8.Location = new Point(748, 694);

label8.Margin = new Padding(4, 0, 4, 0);

label8.Name = "label8";

label8.Size = new Size(141, 20);

label8.TabIndex = 22;

label8.Text = "Общая длина пути:";

//

// MainForm

//

AutoScaleDimensions = new SizeF(8F, 20F);

AutoScaleMode = AutoScaleMode.Font;

ClientSize = new Size(1110, 763);

Controls.Add(label8);

Controls.Add(txtOptimalDistance);

Controls.Add(label7);

Controls.Add(txtStartVertex);

Controls.Add(btnRemoveVertex);

Controls.Add(btnAddVertex);

Controls.Add(btnSolve);

Controls.Add(pictureBox);

Margin = new Padding(4, 5, 4, 5);

Name = "MainForm";

Text = "Graph Visualization";

((System.ComponentModel.ISupportInitialize)pictureBox).EndInit();

ResumeLayout(false);

PerformLayout();

}

#endregion

private System.Windows.Forms.PictureBox pictureBox;

private System.Windows.Forms.Button btnSolve;

private System.Windows.Forms.Button btnAddVertex;

private System.Windows.Forms.Button btnRemoveVertex;

private System.Windows.Forms.TextBox txtStartVertex;

private System.Windows.Forms.Label label7;

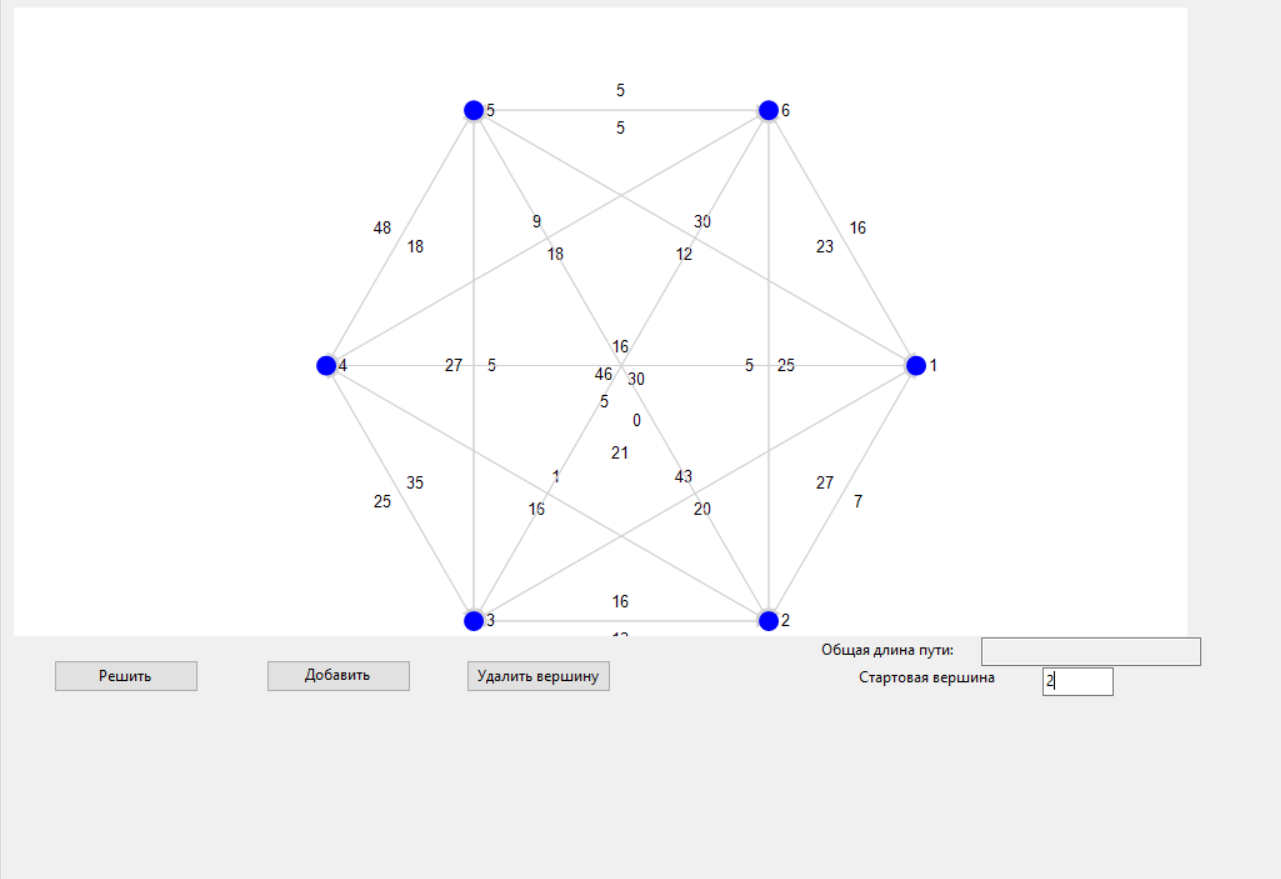
private System.Windows.Forms.TextBox txtOptimalDistance; // Новое текстовое поле

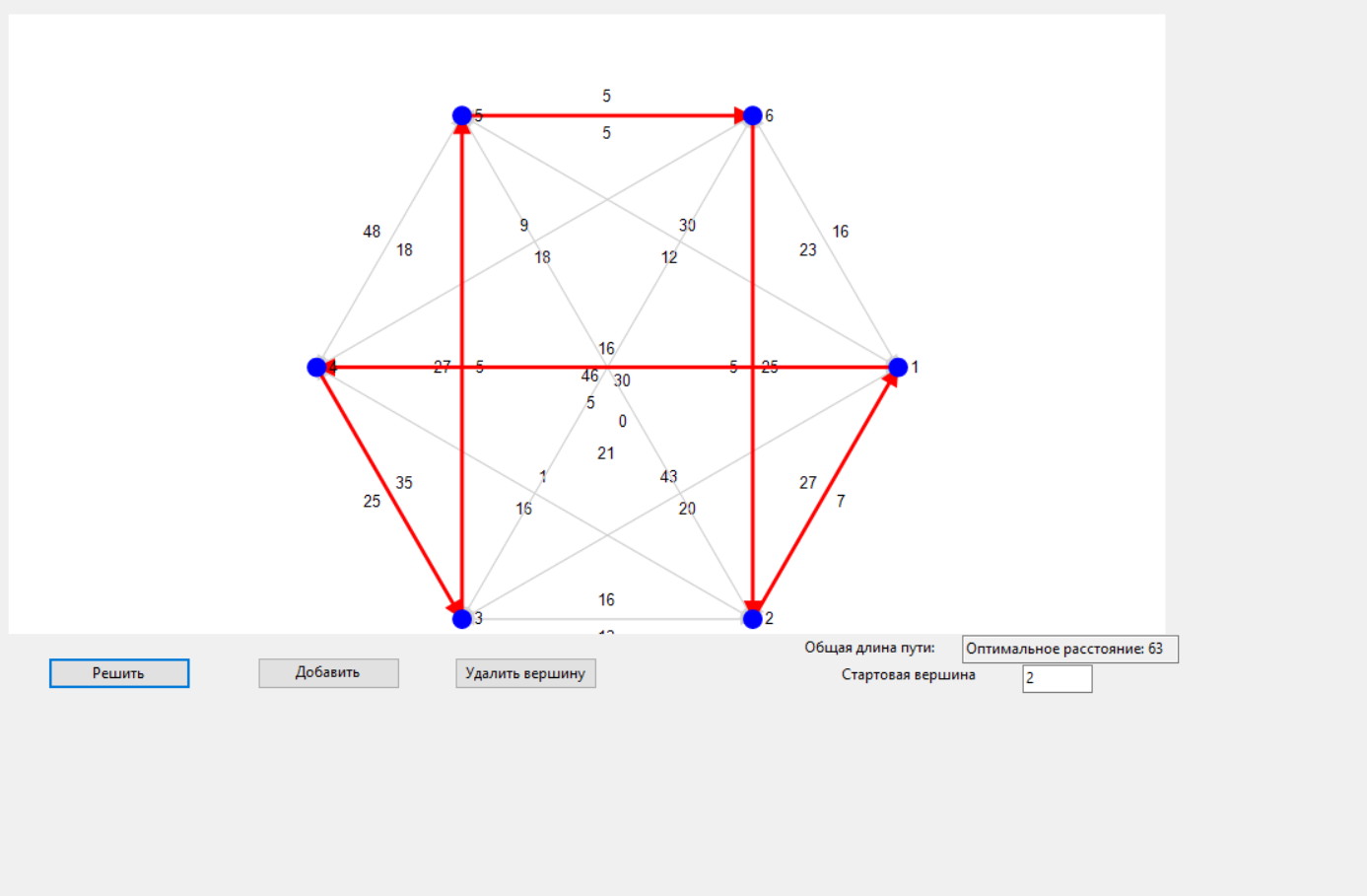
private System.Windows.Forms.Label label8; // Новая метка для текста расстояния

}

}

Результат работы:





Ссылка на GitHub:  
https://github.com/LeonidZhir/-