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:

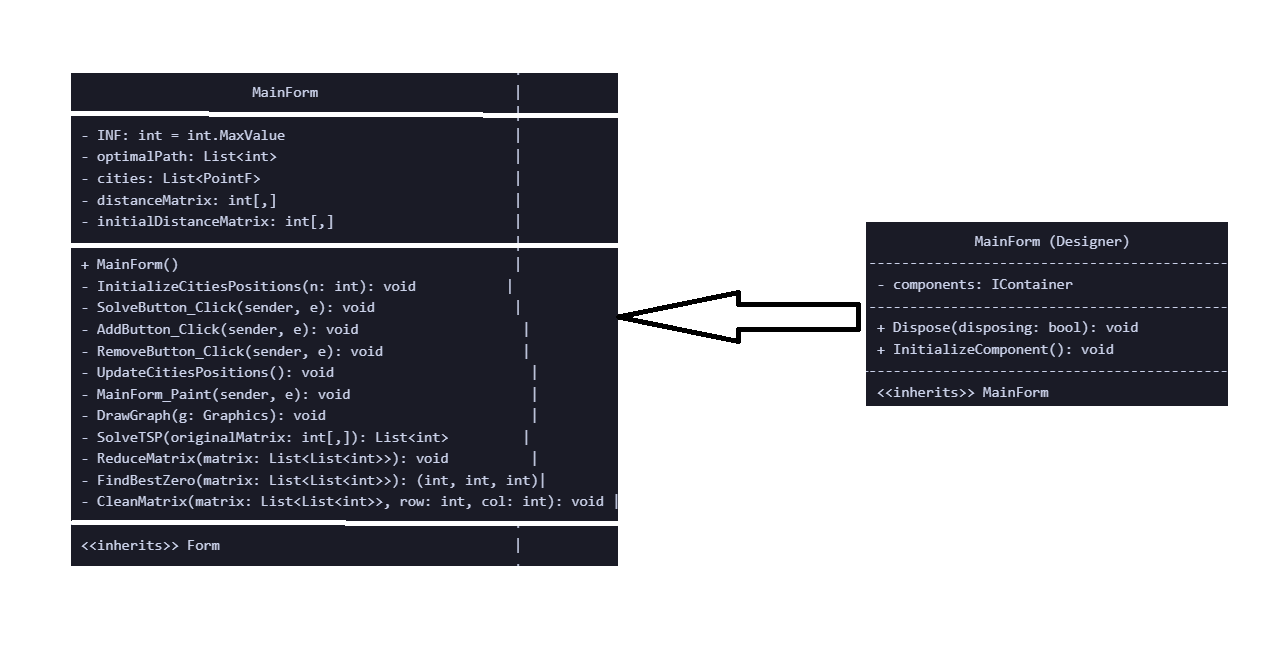
Имеется N городов. Выезжая из исходного города А1, коммивояжер должен побывать во всех городах по одному разу и вернуться в город А1. Задача заключается в определении последовательности объезда городов, при которой коммивояжеру требуется минимизировать некоторый критерий эффективности: стоимость проезда,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

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;

var solveButton = new Button { Text = "Решить", Dock = DockStyle.Bottom };

solveButton.Click += SolveButton\_Click;

this.Controls.Add(solveButton);

var addButton = new Button { Text = "Добавить город", Dock = DockStyle.Bottom };

addButton.Click += AddButton\_Click;

this.Controls.Add(addButton);

var removeButton = new Button { Text = "Удалить город", Dock = DockStyle.Bottom };

removeButton.Click += RemoveButton\_Click;

this.Controls.Add(removeButton);

InitializeCitiesPositions(initialDistanceMatrix.GetLength(0));

distanceMatrix = initialDistanceMatrix;

}

// Инициализация позиций городов

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));

}

}

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

{

optimalPath = SolveTSP(distanceMatrix);

this.Invalidate();

}

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

{

// Добавление нового города

int newCityIndex = cities.Count;

// Увеличиваем размер матрицы расстояний

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; // Обновляем расстояния

newDistanceMatrix[newCityIndex, i] = INF; // Обновляем расстояния

}

newDistanceMatrix[newCityIndex, newCityIndex] = int.MaxValue; // Расстояние до самого себя

distanceMatrix = newDistanceMatrix;

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

UpdateCitiesPositions();

this.Invalidate();

}

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

{

if (cities.Count > 0)

{

int cityIndexToRemove = cities.Count - 1; // Удаляем последний добавленный город

int newSize = cities.Count - 1;

// Уменьшаем размер матрицы расстояний

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

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

{

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

{

if (i < cityIndexToRemove && j < cityIndexToRemove)

{

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

}

else if (i < cityIndexToRemove && j >= cityIndexToRemove)

{

newDistanceMatrix[i, j] = distanceMatrix[i, j + 1];

}

else if (i >= cityIndexToRemove && j < cityIndexToRemove)

{

newDistanceMatrix[i, j] = distanceMatrix[i + 1, j];

}

}

}

distanceMatrix = newDistanceMatrix;

cities.RemoveAt(cityIndexToRemove);

UpdateCitiesPositions();

this.Invalidate();

}

}

private void UpdateCitiesPositions()

{

InitializeCitiesPositions(cities.Count);

}

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

{

DrawGraph(e.Graphics);

}

private void DrawGraph(Graphics g)

{

int n = cities.Count;

// Рисуем все ребра светло-серым

using (var pen = new Pen(Color.LightGray))

{

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

for (int j = i + 1; j < n; j++)

{

if (distanceMatrix[i, j] != INF)

{

g.DrawLine(pen, cities[i], cities[j]);

var mid = new PointF((cities[i].X + cities[j].X) / 2, (cities[i].Y + cities[j].Y) / 2);

g.DrawString(distanceMatrix[i, j].ToString(), this.Font, Brushes.Black, mid);

}

}

}

// Рисуем оптимальный путь красным

if (optimalPath != null && optimalPath.Count > 1)

{

using (var pen = new Pen(Color.Red, 3))

{

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

{

int from = optimalPath[i];

int to = optimalPath[i + 1];

g.DrawLine(pen, cities[from], cities[to]);

}

}

}

// Рисуем города и их номера

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

{

g.FillEllipse(Brushes.Blue, cities[i].X - 10, cities[i].Y - 10, 20, 20);

var text = $"A{i + 1}";

var size = g.MeasureString(text, this.Font);

g.DrawString(text, this.Font, Brushes.White, cities[i].X - size.Width / 2, cities[i].Y - size.Height / 2);

}

}

// Алгоритм решения задачи коммивояжёра

private List<int> SolveTSP(int[,] originalMatrix)

{

int n = originalMatrix.GetLength(0);

List<List<int>> matrix = new List<List<int>>();

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

{

var row = new List<int>();

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

row.Add(originalMatrix[i, j]);

matrix.Add(row);

}

List<(int from, int to)> edges = new List<(int, int)>();

while (true)

{

ReduceMatrix(matrix);

var bestZero = FindBestZero(matrix);

if (bestZero.zeroRow == -1) break;

edges.Add((bestZero.zeroRow, bestZero.zeroCol));

CleanMatrix(matrix, bestZero.zeroRow, bestZero.zeroCol);

}

Dictionary<int, int> nextCity = edges.ToDictionary(e => e.from, e => e.to);

List<int> path = new List<int>();

int current = 0;

path.Add(current);

while (nextCity.ContainsKey(current) && path.Count < n)

{

current = nextCity[current];

path.Add(current);

}

path.Add(path[0]);

return path;

}

private void ReduceMatrix(List<List<int>> matrix)

{

int n = matrix.Count;

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

{

int minRow = INF;

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

if (matrix[i][j] < minRow)

minRow = matrix[i][j];

if (minRow == INF || minRow == 0) continue;

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

if (matrix[i][j] != INF)

matrix[i][j] -= minRow;

}

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

{

int minCol = INF;

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

if (matrix[i][j] < minCol)

minCol = matrix[i][j];

if (minCol == INF || minCol == 0) continue;

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

if (matrix[i][j] != INF)

matrix[i][j] -= minCol;

}

}

private (int score, int zeroRow, int zeroCol) FindBestZero(List<List<int>> matrix)

{

int n = matrix.Count;

int maxScore = -1;

int bestRow = -1;

int bestCol = -1;

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

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

if (matrix[i][j] == 0)

{

int rowMin = INF;

int colMin = INF;

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

if (k != j && matrix[i][k] < rowMin)

rowMin = matrix[i][k];

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

if (k != i && matrix[k][j] < colMin)

colMin = matrix[k][j];

if (rowMin == INF) rowMin = 0;

if (colMin == INF) colMin = 0;

int score = rowMin + colMin;

if (score > maxScore)

{

maxScore = score;

bestRow = i;

bestCol = j;

}

}

return (maxScore, bestRow, bestCol);

}

private void CleanMatrix(List<List<int>> matrix, int row, int col)

{

int n = matrix.Count;

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

matrix[row][j] = INF;

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

matrix[i][col] = INF;

matrix[col][row] = INF;

}

}

}

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);

}

private void InitializeComponent()

{

this.SuspendLayout();

//

// MainForm

//

this.ClientSize = new System.Drawing.Size(800, 600);

this.Name = "MainForm";

this.Text = "Визуализация задачи коммивояжера";

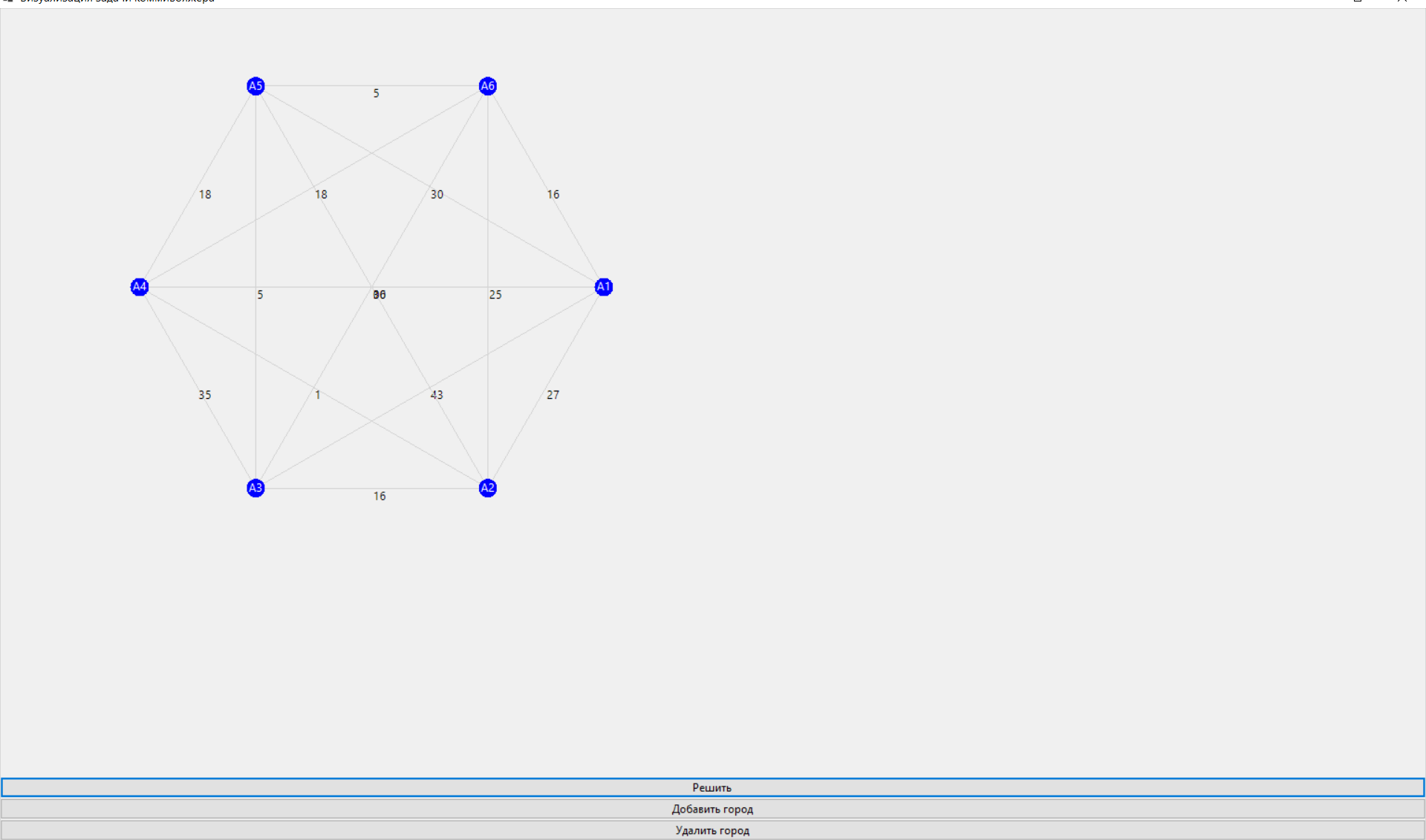
this.ResumeLayout(false);

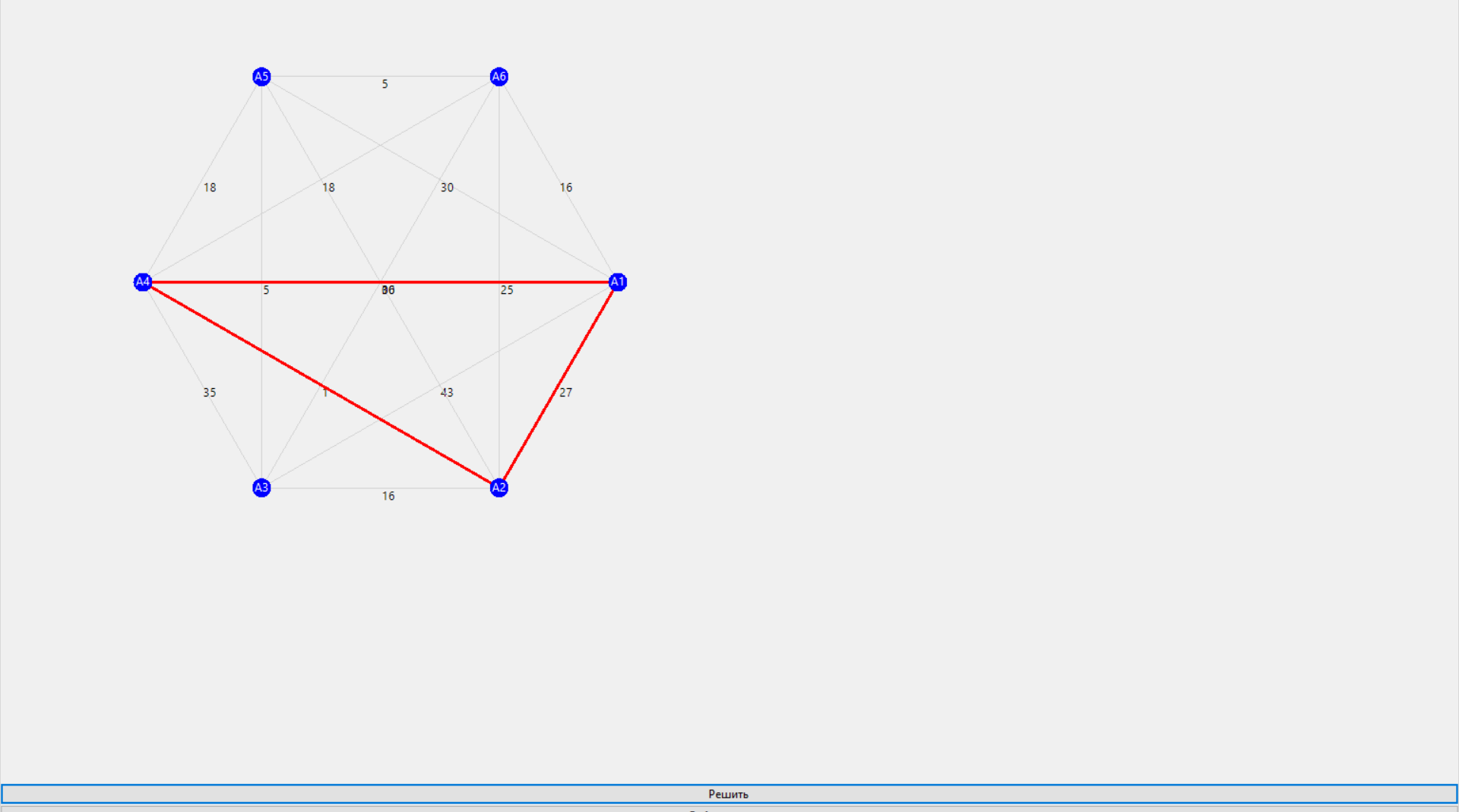
}

}

}

Результат работы:





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