namespace GraphMethod

{

public partial class SolvingGraphM : Form

{

Chart chart;

ComboBox maxOrMin;

ModelForPlotViev model1, model2, model3;

bool IsM1 = true, IsM2 = false, IsM3 = false;

string text1, text2, text3;

Font font = new Font("Times New Roman", 14);

public SolvingGraphM()

{

InitializeComponent();

MinimumSize = new Size(700, 600);

Size = MinimumSize;

plotView1.Dock = DockStyle.Fill;

panel1.Size = new Size(Width, (int)(Height \* 0.8));

panel1.SendToBack();

panel1.BorderStyle = BorderStyle.None;

label6.Size = new Size((int)(panel1.Width \* 0.9), (int)(panel1.Height \* 0.4));

label6.Font = font;

label6.TextAlign = ContentAlignment.TopLeft;

label6.AutoSize = true;

label7.AutoSize = true;

label7.Text = "Целевая функция стремится к";

label7.Font = font;

label7.Location = new Point((int)(Width \* 0.05), label7.Height);

label8.Text = "F = ";

label8.Font = font;

label8.Location = new Point(label7.Location.X, label7.Location.Y + label7.Height + (int)(Height \* 0.02));

TextBox x1 = new TextBox() { Name = "x1", Size = new Size((int)(Width \* 0.04), (int)(Height \* 0.04)), Location = new Point(label8.Location.X + label8.Width, label8.Location.Y) };

Label x1d = new Label() { Name = "x1d", Text = "x1 + ", AutoSize = true, Font = font, Location = new Point(x1.Location.X + x1.Width, x1.Location.Y), };

TextBox x2 = new TextBox() { Name = "x2", Size = new Size((int)(Width \* 0.04), (int)(Height \* 0.04)), Location = new Point(x1d.Location.X + x1.Width + (int)(Width \* 0.02), label8.Location.Y) };

Label x2d = new Label() { Name = "x2d", Text = "x2", AutoSize = true, Font = font, Location = new Point(x2.Location.X + x2.Width, x1.Location.Y), };

maxOrMin = new ComboBox() { Size = new Size((int)(Width \* 0.1), (int)(Height \* 0.04)), Location = new Point(label7.Location.X + label7.Width + (int)(Width \* 0.02), label7.Location.Y) };

maxOrMin.Items.AddRange(new string[] { "Max", "Min" });

label1.Font = font;

label1.Location = new Point(label8.Location.X, label8.Location.Y + label8.Height + (int)(Height \* 0.02));

textBox1.Size = maxOrMin.Size;

textBox1.Location = new Point(label1.Location.X + label1.Width + (int)(Width \* 0.02), label1.Location.Y);

label2.Font = font;

label2.Location = new Point(label1.Location.X, label1.Location.Y + label1.Height + (int)(Height \* 0.05));

comboBox1.Location = new Point(label2.Location.X + label2.Width, label2.Location.Y);

label3.Font = font;

label3.Location = new Point(comboBox1.Location.X + comboBox1.Width, comboBox1.Location.Y);

label4.Font = font;

label4.Location = new Point(label3.Location.X + label3.Width \* 3, label3.Location.Y);

comboBox2.Location = new Point(label4.Location.X + label4.Width, label4.Location.Y);

label5.Font = font;

label5.Location = new Point(comboBox2.Location.X + comboBox2.Width, label4.Location.Y);

button1.Text = "Создать";

button1.Font = font;

button1.AutoSize = true;

button1.Location = new Point(label5.Location.X + label5.Width \* 3, label5.Location.Y);

button3.Visible = false;

dataGridView2.Size = new Size((int)(Width \* 0.9), (int)(Height \* 0.4));

dataGridView2.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

dataGridView2.Font = font;

dataGridView2.Location = new Point((Width - dataGridView2.Width) / 2, button1.Location.Y + (int)(Width \* 0.1));

button2.Font = font;

button2.Text = "Решить";

button2.AutoSize = true;

button2.Location = new Point(dataGridView2.Location.X, dataGridView2.Location.Y + dataGridView2.Height + (int)(Height \* 0.05));

chart = new Chart(plotView1);

comboBox1.Items.AddRange(new string[] { ">=", "<=" });

comboBox2.Items.AddRange(new string[] { ">=", "<=" });

Controls.Add(maxOrMin);

Controls.Add(x1);

Controls.Add(x2);

Controls.Add(x1d);

Controls.Add(x2d);

maxOrMin.BringToFront();

}

private void CreateChart()

{

model1 = chart.CreateModel("Графическое решение ЗЛП");

model1.addAxisX(Minimum: -20, Maximum: 20);

model1.addAxisY(Minimum: -20, Maximum: 20);

model2 = chart.CreateModel("Графическое решение ЗЛП");

model2.addAxisX(Minimum: -20, Maximum: 20);

model2.addAxisY(Minimum: -20, Maximum: 20);

model3 = chart.CreateModel("Графическое решение ЗЛП");

model3.addAxisX(Minimum: -20, Maximum: 20);

model3.addAxisY(Minimum: -20, Maximum: 20);

string minOrMax = maxOrMin?.SelectedItem?.ToString() ?? "Min";

string x1Condition = comboBox1?.SelectedItem?.ToString() ?? ">=";

string x2Condition = comboBox2?.SelectedItem?.ToString() ?? ">=";

var constraints = new List<Tuple<float, float, float, string>>();

foreach (DataGridViewRow row in dataGridView2.Rows)

{

if (row.IsNewRow) continue;

float x1 = float.TryParse(row.Cells[0].Value?.ToString(), out x1) ? x1 : 0;

float x2 = float.TryParse(row.Cells[1].Value?.ToString(), out x2) ? x2 : 0;

string sign = row.Cells[2].Value?.ToString() ?? ">=";

float num = float.TryParse(row.Cells[3].Value?.ToString(), out num) ? num : 0;

constraints.Add(new Tuple<float, float, float, string>(x1, x2, num, sign));

}

var X1 = Controls.Find("x1", false)[0].Text;

var X2 = Controls.Find("x2", false)[0].Text;

float x = 0, y = 0;

x = float.TryParse(X1, out x) ? x : 1;

y = float.TryParse(X2, out y) ? x : 1;

model1.DrawLines(constraints);

model2.DrawLines(constraints);

model3.DrawLines(constraints);

if (!model2.IsSystemOfConstraintsConsistent(constraints, x1Condition, x2Condition))

{

text1 = $"Система ограничений задачи несовместима";

text2 = $"Система ограничений задачи несовместима";

text3 = $"Система ограничений задачи несовместима";

}

else if (model2.HasMultipleSolutions(constraints, x, y, minOrMax, x1Condition, x2Condition, out var point))

{

model2.DrawRangeOfAcceptableValues(constraints, x1Condition, x2Condition);

model3.DrawRangeOfAcceptableValues(constraints, x1Condition, x2Condition);

text1 = $"Необходимо найти {((minOrMax == "Min") ? "минимальное" : "максимальное")} значение функции F = {(x == 0 ? "" : x.ToString() + "X1")} " +

$"{(y == 0 ? "" : (y < 0 ? "- " + y.ToString() + "X2" : "+" + y.ToString() + "X2"))}" +

$" -> {minOrMax} при системе ограничений:\n";

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

{

text1 += $"{(constraints[i].Item1 == 0 ? "" : constraints[i].Item1 + "X1")} {(constraints[i].Item2 == 0 ? "" : (constraints[i].Item2 < 0 ? "- " + constraints[i].Item2 + "X2" : "+ " + constraints[i].Item2.ToString() + "X2"))} {constraints[i].Item4} {constraints[i].Item3} \t\t({i + 1})\n";

}

text1 += $"x1 {x1Condition} 0 \tx2 {x2Condition} 0\n";

text1 += "Шаг N1. Построим область допустимых решений, т.е. решим графически систему неравенств. Для этого построим каждую прямую.";

text2 = "Шаг N2. Границы области допустимых решений.\r\nПересечением полуплоскостей будет являться область, координаты точек которого удовлетворяют условию неравенствам системы ограничений задачи.\r\nОбозначим границы области многоугольника решений.";

model3.DrawLineGradient(x, y);

model3.DrawLevelLines(x, y, constraints, minOrMax, x1Condition, x2Condition, out double x1, out double x2, point);

text3 = $"Функция принимает {((minOrMax == "Min") ? "минимальное" : "максимальное")} значение в любой точке на отрезка AB\n" +

$"F(x) = {x} \* {x1:f3} + {y:f3} \* {x2:f3} = {x \* x1 + y \* x2:f3}";

}

//else if (model2.IsObjectiveFunctionUnbounded(constraints, x, y, minOrMax, x1Condition, x2Condition))

else if (!model2.HasMultipleSolutions(constraints, x, y, minOrMax, x1Condition, x2Condition, out point))

{

model2.DrawRangeOfAcceptableValues(constraints, x1Condition, x2Condition);

model3.DrawRangeOfAcceptableValues(constraints, x1Condition, x2Condition);

model3.DrawLineGradient(x, y);

model3.DrawLevelLines(x, y, constraints, minOrMax, x1Condition, x2Condition, out double x1, out double x2, point);

text1 = $"Необходимо найти {((minOrMax == "Min") ? "минимальное" : "максимальное")} значение функции F = {(x == 0 ? "" : x.ToString() + "X1")} " +

$"{(y == 0 ? "" : (y < 0 ? "- " + y.ToString() + "X2" : "+" + y.ToString() + "X2"))}" +

$" -> {minOrMax} при системе ограничений:\n";

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

{

text1 += $"{(constraints[i].Item1 == 0 ? "" : constraints[i].Item1 + "X1")} {(constraints[i].Item2 == 0 ? "" : (constraints[i].Item2 < 0 ? "- " + constraints[i].Item2 + "X2" : "+ " + constraints[i].Item2.ToString() + "X2"))} {constraints[i].Item4} {constraints[i].Item3} \t\t({i + 1})\n";

}

text1 += $"x1 {x1Condition} 0 \tx2 {x2Condition} 0\n";

text1 += "Шаг N1. Построим область допустимых решений, т.е. решим графически систему неравенств. Для этого построим каждую прямую.";

text2 = "Шаг N2. Границы области допустимых решений.\r\nПересечением полуплоскостей будет являться область, координаты точек которого удовлетворяют условию неравенствам системы ограничений задачи.\r\nОбозначим границы области многоугольника решений.";

text3 = $"Шаг N3. Рассмотрим целевую функцию задачи F = {(x == 0 ? "" : x.ToString() + "X1")} {(y == 0 ? "" : (y < 0 ? "- " + y.ToString() + "X2" : "+" + y.ToString() + "X2"))} → {minOrMax.ToLower()}.\n" +

$"Вектор-градиент, составленный из коэффициентов целевой функции, указывает направление максимизации F(X). Начало вектора – точка (0; 0), конец – точка ({x};{y}).\n" +

$"Прямая F(x) = const пересекает область в точке A, ее координатами являются ({x1:f3};{x2:f3})" +

$"Откуда найдем {(minOrMax == "Min" ? "минимальное" : "максимальное")} значение целевой функции\n" +

$"F(x) = {x} \* {x1:f3} + {y:f3} \* {x2:f3} = {x \* x1 + y \* x2:f3}";

}

else

{

model2.DrawRangeOfAcceptableValues(constraints, x1Condition, x2Condition);

model3.DrawRangeOfAcceptableValues(constraints, x1Condition, x2Condition);

text1 = $"Необходимо найти {((minOrMax == "Min") ? "минимальное" : "максимальное")} значение функции F = {(x == 0 ? "" : x.ToString() + "X1")} " +

$"{(y == 0 ? "" : (y < 0 ? "- " + y.ToString() + "X2" : "+" + y.ToString() + "X2"))}" +

$" -> {minOrMax} при системе ограничений:\n";

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

{

text1 += $"{(constraints[i].Item1 == 0 ? "" : constraints[i].Item1 + "X1")} {(constraints[i].Item2 == 0 ? "" : (constraints[i].Item2 < 0 ? "- " + constraints[i].Item2 + "X2" : "+ " + constraints[i].Item2.ToString() + "X2"))} {constraints[i].Item4} {constraints[i].Item3} \t\t({i + 1})\n";

}

text1 += $"x1 {x1Condition} 0 \tx2 {x2Condition} 0\n";

text1 += "Шаг N1. Построим область допустимых решений, т.е. решим графически систему неравенств. Для этого построим каждую прямую.";

text2 = "Шаг N2. Границы области допустимых решений.\r\nПересечением полуплоскостей будет являться область, координаты точек которого удовлетворяют условию неравенствам системы ограничений задачи.\r\nОбозначим границы области многоугольника решений.";

text3 = $"Функция неограничена сверху";

}

chart.DrawModel(model1);

plotView1.Size = new Size((int)(panel1.Width \* 0.5), (int)(panel1.Height \* 0.5));

plotView1.Location = new Point((panel1.Width - plotView1.Width) / 2, (Height - label6.Height - 20 - plotView1.Height) / 2);

}

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

{

dataGridView2.Columns.Clear();

for (int i = 0; i < 2; i++)

{

dataGridView2.Columns.Add($"X{i + 1}", $"X{i + 1}");

}

dataGridView2.Columns.Add("Sign", "Sign");

dataGridView2.Columns.Add("Num", "Num");

if (int.TryParse(textBox1.Text, out int rowCount))

{

dataGridView2.Rows.Clear();

dataGridView2.Rows.Add(rowCount);

}

}

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

{

Width = Screen.PrimaryScreen.Bounds.Width;

Height = Screen.PrimaryScreen.Bounds.Height;

Location = new Point(0, 0);

panel1.Size = new Size(Width, (int)(Height \* 0.7));

InitializeAdditionComponents();

CreateChart();

label6.Text = text1;

var control = Controls.Find("0", false)[0];

label6.Location = new Point((Width - label6.Width) / 4, control.Height + control.Location.Y + label6.Height / 2); ;

panel1.Location = new Point((Width - panel1.Width) / 2, label6.Location.Y + label6.Height + 10);

hideAndShowComponents(false);

}

private void button3\_Click(object sender, EventArgs e)

{

Random rnd = new Random();

foreach (DataGridViewRow row in dataGridView2.Rows)

{

if (row.IsNewRow) continue;

for (int i = 0; i < dataGridView2.ColumnCount - 2; i++)

row.Cells[i].Value = rnd.Next(-20, 20);

row.Cells[dataGridView2.ColumnCount - 2].Value = rnd.Next(1, 3) == 2 ? "<=" : ">=";

row.Cells[dataGridView2.ColumnCount - 1].Value = rnd.Next(-20, 20);

}

}

private void InitializeAdditionComponents()

{

Control[] buttons = new Control[3];

for (int i = 0; i < 3; i++)

{

Button button = new Button() { Text = $"Шаг {i + 1}", Name = i.ToString() };

button.Size = new Size((int)(Width \* 0.1), (int)(Height \* 0.04));

buttons[i] = button;

buttons[i].Click += ReplaceModel;

buttons[i].Visible = false;

}

Button ResetButton = new Button()

{

Size = new Size((int)(Width \* 0.03), (int)(Width \* 0.03)),

Visible = false,

Name = "ResetButton",

BackgroundImageLayout = ImageLayout.Stretch,

ImageAlign = ContentAlignment.MiddleCenter,

BackgroundImage = Image.FromFile(AppDomain.CurrentDomain.BaseDirectory + @"Assets\reset.png"),

};

ResetButton.Click += Reset;

int spacing = 10;

int ButtonsWidts = buttons.Sum(b => b.Width) + (buttons.Length - 1) \* spacing;

int StartPos = (Width - ButtonsWidts) / 2;

for (int i = 0; i < buttons.Length; i++)

{

buttons[i].Name = i.ToString();

buttons[i].Location = new Point(StartPos + buttons[i].Width \* i + spacing \* i, (int)(Height \* 0.01));

};

ResetButton.Location = new Point(Width - ResetButton.Width - spacing \* 3, spacing);

Controls.AddRange(buttons);

Controls.Add(ResetButton);

for (int i = 0; i < buttons.Length; i++)

buttons[i].BringToFront();

ResetButton.BringToFront();

}

private void Reset(object? sender, EventArgs e)

{

Size = MinimumSize;

hideAndShowComponents(true);

dataGridView2.Columns.Clear();

var X1 = Controls.Find("x1", false)[0].Text = "";

var X2 = Controls.Find("x2", false)[0].Text = "";

}

private void ReplaceModel(object? sender, EventArgs e)

{

var control = Controls.Find("0", false)[0];

if (sender == null) return;

if (((Button)sender).Name == "0" && !IsM1)

{

chart.DrawModel(model1);

label6.Text = text1;

IsM1 = true;

IsM2 = false;

IsM3 = false;

}

else if (((Button)sender).Name == "1" && !IsM2)

{

chart.DrawModel(model2);

label6.Text = text2;

IsM1 = false;

IsM2 = true;

IsM3 = false;

}

else if (((Button)sender).Name == "2" && !IsM3)

{

chart.DrawModel(model3);

label6.Text = text3;

IsM1 = false;

IsM2 = false;

IsM3 = true;

}

label6.Location = new Point((Width - label6.Width) / 4, control.Height + control.Location.Y + label6.Height / 2); ;

panel1.Location = new Point((Width - panel1.Width) / 2, label6.Location.Y + label6.Height + 10);

}

private void hideAndShowComponents(bool hide)

{

if (hide)

{

Controls.Find("x1", false)[0].Visible = true;

Controls.Find("x2", false)[0].Visible = true;

Controls.Find("x1d", false)[0].Visible = true;

Controls.Find("x2d", false)[0].Visible = true;

label7.Visible = true;

label8.Visible = true;

dataGridView2.Visible = true;

comboBox1.Visible = true;

comboBox2.Visible = true;

button1.Visible = true;

button2.Visible = true;

button3.Visible = true;

label1.Visible = true;

label2.Visible = true;

label3.Visible = true;

label4.Visible = true;

label5.Visible = true;

textBox1.Visible = true;

maxOrMin.Visible = true;

plotView1.Visible = false;

Controls.Find("ResetButton", false)[0].Visible = false;

label6.Visible = false;

foreach (var control in Controls)

{

try

{

var name = ((Button)control).Name;

if ((name == "0") || (name == "1") || (name == "2"))

((Button)control).Visible = false;

}

catch (Exception ex) { }

}

}

else

{

Controls.Find("x1", false)[0].Visible = false;

Controls.Find("x2", false)[0].Visible = false;

Controls.Find("x1d", false)[0].Visible = false;

Controls.Find("x2d", false)[0].Visible = false;

label7.Visible = false;

label8.Visible = false;

dataGridView2.Visible = false;

comboBox1.Visible = false;

comboBox2.Visible = false;

button1.Visible = false;

button2.Visible = false;

button3.Visible = false;

label1.Visible = false;

label2.Visible = false;

label3.Visible = false;

label4.Visible = false;

label5.Visible = false;

textBox1.Visible = false;

maxOrMin.Visible = false;

plotView1.Visible = true;

Controls.Find("ResetButton", false)[0].Visible = true;

foreach (var control in Controls)

{

try

{

var name = ((Button)control).Name;

if ((name == "0") || (name == "1") || (name == "2"))

((Button)control).Visible = true;

}

catch (Exception ex) { }

}

}

}

}

}

using OxyPlot;

using OxyPlot.Annotations;

using OxyPlot.Axes;

using OxyPlot.Series;

using OxyPlot.WindowsForms;

using System.Data;

namespace GraphMethod

{

class Chart

{

private PlotView plotView;

private ModelForPlotViev plotModel;

public Chart() { }

public Chart(PlotView PlotView) => this.plotView = PlotView;

public ModelForPlotViev CreateModel(string ModelsTitle) => this.plotModel = new ModelForPlotViev() { Title = ModelsTitle };

public void DrawModel(ModelForPlotViev model) => plotView.Model = model.plotModel;

}

class ModelForPlotViev : PlotModel

{

public PlotModel plotModel { get; }

int XMin, XMax, yMin, yMax;

public ModelForPlotViev() { plotModel = new PlotModel(); }

public void addAxisX(int Minimum = -30, int Maximum = 30, AxisPosition Position = AxisPosition.Bottom, bool PositionAtZeroCrossing = true, bool IsZoomEmable = true, bool IsPanEnabled = true)

{

XMin = Minimum;

XMax = Maximum;

plotModel.Axes.Add(new LinearAxis { Position = Position, Minimum = Minimum, Maximum = Maximum, PositionAtZeroCrossing = PositionAtZeroCrossing, IsZoomEnabled = IsZoomEmable, IsPanEnabled = IsPanEnabled});

var xAxisLine = new LineSeries { Color = OxyColors.Black, StrokeThickness = 2 };

xAxisLine.Points.Add(new OxyPlot.DataPoint(Minimum, 0));

xAxisLine.Points.Add(new OxyPlot.DataPoint(Maximum, 0));

plotModel.Series.Add(xAxisLine);

}

public void addAxisY(int Minimum = -30, int Maximum = 30, AxisPosition Position = AxisPosition.Left, bool PositionAtZeroCrossing = true, bool IsZoomEmable = true, bool IsPanEnabled = true)

{

yMin = Minimum;

yMax = Maximum;

plotModel.Axes.Add(new LinearAxis { Position = Position, Minimum = Minimum, Maximum = Maximum, PositionAtZeroCrossing = PositionAtZeroCrossing, IsZoomEnabled = IsZoomEmable, IsPanEnabled = IsPanEnabled});

var yAxisLine = new LineSeries { Color = OxyColors.Black, StrokeThickness = 2 };

yAxisLine.Points.Add(new OxyPlot.DataPoint(0, Minimum));

yAxisLine.Points.Add(new OxyPlot.DataPoint(0, Maximum));

plotModel.Series.Add(yAxisLine);

}

public void DrawLines(List<Tuple<float, float, float, string>> constraints)

{

foreach (var constraint in constraints)

{

Random rnd = new Random();

OxyColor color = OxyColor.FromArgb(255, (byte)rnd.Next(1, 255), (byte)rnd.Next(1, 255), (byte)rnd.Next(1, 255));

LineSeries series = new LineSeries

{

Color = color,

};

float x1 = constraint.Item1;

float x2 = constraint.Item2;

float num = constraint.Item3;

string sign = constraint.Item4;

DataPoint startPoint, endPoint;

if (x1 == 0)

{

startPoint = new DataPoint(XMin, num / x2);

endPoint = new DataPoint(XMax, num / x2);

}

else if (x2 == 0)

{

startPoint = new DataPoint(num / x1, yMin);

endPoint = new DataPoint(num / x1, yMax);

}

else

{

double yAtXMin = (num - x1 \* XMin) / x2;

double yAtXMax = (num - x1 \* XMax) / x2;

double xAtYMin = (num - x2 \* yMin) / x1;

double xAtYMax = (num - x2 \* yMax) / x1;

List<DataPoint> points = new List<DataPoint>();

if (yAtXMin >= yMin && yAtXMin <= yMax)

points.Add(new DataPoint(XMin, yAtXMin));

if (yAtXMax >= yMin && yAtXMax <= yMax)

points.Add(new DataPoint(XMax, yAtXMax));

if (xAtYMin >= XMin && xAtYMin <= XMax)

points.Add(new DataPoint(xAtYMin, yMin));

if (xAtYMax >= XMin && xAtYMax <= XMax)

points.Add(new DataPoint(xAtYMax, yMax));

if (points.Count >= 2)

{

startPoint = points[0];

endPoint = points[^1];

}

else

{

continue; // Если точки не найдены, пропускаем ограничение

}

}

series.Points.Add(startPoint);

series.Points.Add(endPoint);

//AddPerpendicularTicks(series, startPoint, endPoint, x1, x2, num, sign);

// Добавление подписи к линии

string equation = $"{x1}x1 + {x2}x2 {sign} {num}";

var textAnnotation = new TextAnnotation

{

Text = equation,

TextColor = color,

TextPosition = startPoint,

FontSize = 12,

TextHorizontalAlignment = OxyPlot.HorizontalAlignment.Center,

TextVerticalAlignment = VerticalAlignment.Top,

StrokeThickness = 0

};

plotModel.Annotations.Add(textAnnotation);

plotModel.Series.Add(series);

}

}

public void DrawLineGradient(double x, double y)

{

var lineGradient = new LineSeries()

{

LineStyle = LineStyle.Solid,

Color = OxyColors.Black,

StrokeThickness = 2

};

lineGradient.Points.Add(new DataPoint(0, 0));

lineGradient.Points.Add(new DataPoint(x, y));

plotModel.Series.Add(lineGradient);

// Добавление треугольника для указания направления

var arrowHead = new PolygonAnnotation()

{

Fill = OxyColors.Black,

Stroke = OxyColors.Black,

StrokeThickness = 0.1

};

double arrowSize = 0.5; // Размер стрелки

double angle = Math.Atan2(y, x);

double arrowX1 = x - arrowSize \* Math.Cos(angle - Math.PI / 6);

double arrowY1 = y - arrowSize \* Math.Sin(angle - Math.PI / 6);

double arrowX2 = x - arrowSize \* Math.Cos(angle + Math.PI / 6);

double arrowY2 = y - arrowSize \* Math.Sin(angle + Math.PI / 6);

arrowHead.Points.Add(new DataPoint(x, y));

arrowHead.Points.Add(new DataPoint(arrowX1, arrowY1));

arrowHead.Points.Add(new DataPoint(arrowX2, arrowY2));

arrowHead.Points.Add(new DataPoint(x, y)); // Закрываем треугольник

plotModel.Annotations.Add(arrowHead);

}

public void DrawRangeOfAcceptableValues(List<Tuple<float, float, float, string>>? constraints, string condition1, string condition2, bool DrawPoints = false)

{

List<Tuple<float, float>> intersections = FindIntersections(constraints ?? throw new ArgumentNullException());

foreach (var constraint in constraints)

{

var points = FindIntersectionWithAxes(constraint.Item1, constraint.Item2, constraint.Item3, constraint.Item4);

intersections.AddRange(points);

}

intersections.Add(new Tuple<float, float>(0, 0));

//foreach (var intersection in intersections) { ScatterSeries scatterSeries = new ScatterSeries { MarkerType = MarkerType.Circle, MarkerStrokeThickness = 2 }; scatterSeries.Points.Add(new ScatterPoint(intersection.Item1, intersection.Item2)); model.Series.Add(scatterSeries); }

var filteredIntersections = FilteringIntersection(constraints, intersections, condition1, condition2);

//FillIntersectionArea(model, filteredIntersections);

if (filteredIntersections.Count > 2)

{

if (DrawPoints)

foreach (var intersection in filteredIntersections)

{

ScatterSeries scatterSeries = new ScatterSeries

{

MarkerType = MarkerType.Circle,

MarkerStrokeThickness = 1

};

scatterSeries.Points.Add(new ScatterPoint(intersection.Item1, intersection.Item2));

plotModel.Series.Add(scatterSeries);

}

var center = new DataPoint(filteredIntersections.Average(p => p.Item1), filteredIntersections.Average(p => p.Item2));

var sortedIntersections = filteredIntersections.OrderBy(p => Math.Atan2(p.Item2 - center.Y, p.Item1 - center.X)).ToList();

var polygon = new PolygonAnnotation

{

Fill = OxyColor.FromAColor(50, OxyColors.Green),

StrokeThickness = 0

};

foreach (var intersection in sortedIntersections)

polygon.Points.Add(new DataPoint(intersection.Item1, intersection.Item2));

if (sortedIntersections.Count > 0)

polygon.Points.Add(new DataPoint(sortedIntersections[0].Item1, sortedIntersections[0].Item2));

plotModel.Annotations.Add(polygon);

}

}

public void DrawLevelLines(double gradientX, double gradientY, List<Tuple<float, float, float, string>> constraints, string MaxOrMin, string condition1, string condition2, out double x1, out double x2, (float, float)point)

{

x1 = 0; x2 = 0;

List<Tuple<float, float>> intersections = FindIntersections(constraints);

foreach (var constraint in constraints)

{

var AddPoints = FindIntersectionWithAxes(constraint.Item1, constraint.Item2, constraint.Item3, constraint.Item4);

intersections.AddRange(AddPoints);

}

intersections.Add(new Tuple<float, float>(0, 0));

var filteredIntersections = FilteringIntersection(constraints, intersections, condition1, condition2);

FindFirstAndLastIntersection(filteredIntersections, gradientX, gradientY, out var firstPoint, out var lastPoint);

var points = new List<DataPoint>()

{

new DataPoint(0,0),

(MaxOrMin == "Max") ?

new DataPoint(lastPoint.Item1, lastPoint.Item2) :

new DataPoint(firstPoint.Item1, firstPoint.Item2),

new DataPoint(gradientX, gradientY),

};

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

{

LineSeries levelLine = new LineSeries

{

LineStyle = LineStyle.LongDash,

StrokeThickness = 2,

Color = OxyColors.Black

};

double normalX = -gradientY;

double normalY = gradientX;

double offsetX = points[i].X + normalX;

double offsetY = points[i].Y + normalY;

levelLine.Points.Add(new DataPoint(offsetX - 5 \* normalX, offsetY - 5 \* normalY));

levelLine.Points.Add(new DataPoint(offsetX + 5 \* normalX, offsetY + 5 \* normalY));

if (i == 1)

{

x1 = points[i].X;

x2 = points[i].Y;

var textAnnotation = new TextAnnotation

{

Text = "A",

TextColor = OxyColors.Purple,

TextPosition = new DataPoint(points[i].X, points[i].Y),

FontSize = 12,

TextHorizontalAlignment = OxyPlot.HorizontalAlignment.Center,

TextVerticalAlignment = VerticalAlignment.Top,

StrokeThickness = 0

};

plotModel.Annotations.Add(textAnnotation);

}

if(point.Item1 != 0 || point.Item2 != 0)

{

var textAnnotation = new TextAnnotation

{

Text = "B",

TextColor = OxyColors.Purple,

TextPosition = new DataPoint(point.Item1, point.Item2),

FontSize = 12,

TextHorizontalAlignment = OxyPlot.HorizontalAlignment.Center,

TextVerticalAlignment = VerticalAlignment.Top,

StrokeThickness = 0

};

plotModel.Annotations.Add(textAnnotation);

}

plotModel.Series.Add(levelLine);

}

}

public bool IsSystemOfConstraintsConsistent(List<Tuple<float, float, float, string>> constraints, string condition1, string condition2)

{

// Найти все пересечения линий ограничений

List<Tuple<float, float>> intersections = FindIntersections(constraints);

foreach (var constraint in constraints)

{

var addPoints = FindIntersectionWithAxes(constraint.Item1, constraint.Item2, constraint.Item3, constraint.Item4);

intersections.AddRange(addPoints);

}

// Проверить, удовлетворяет ли хотя бы одна точка всем ограничениям

int count = 0;

foreach (var point in intersections)

{

if ((condition1 == ">=" ? point.Item1 >= 0 : point.Item1 <= 0) &&

(condition2 == ">=" ? point.Item2 >= 0 : point.Item2 <= 0) && SatisfiesConstraints(point, constraints))

{

count++; // Найдена хотя бы одна точка, удовлетворяющая всем ограничениям

}

}

if (count >= 2)

return true;

return false; // Не найдено ни одной точки, удовлетворяющей всем ограничениям

}

public bool HasMultipleSolutions(List<Tuple<float, float, float, string>> constraints, double gradientX, double gradientY, string MaxOrMin, string condition1, string condition2, out (float, float) point)

{

List<Tuple<float, float>> intersections = FindIntersections(constraints);

foreach (var constraint in constraints)

{

var addPoints = FindIntersectionWithAxes(constraint.Item1, constraint.Item2, constraint.Item3, constraint.Item4);

intersections.AddRange(addPoints);

}

intersections.Add(new Tuple<float, float>(0, 0));

var filteredIntersections = FilteringIntersection(constraints, intersections, condition1, condition2);

FindFirstAndLastIntersection(filteredIntersections, gradientX, gradientY, out var firstPoint, out var lastPoint);

var targetPoint = (MaxOrMin == "Max") ? lastPoint : firstPoint;

double levelSlope = -gradientX / gradientY;

foreach (var constraint in constraints)

{

if (IsPointOnConstraintLine(constraint, targetPoint))

{ double constraintSlope;

if (constraint.Item2 != 0)

{

constraintSlope = -constraint.Item1 / constraint.Item2;

} else

{

constraintSlope = double.PositiveInfinity;

}

if (Math.Abs(levelSlope - constraintSlope) < 1e-6)

{

foreach (var pointInSet in filteredIntersections)

{

if (pointInSet != targetPoint && IsPointOnConstraintLine(constraint, pointInSet))

{

point = (pointInSet.Item1, pointInSet.Item2);

return true;

}

}

}

}

}

point = (0, 0);

return false;

}

private bool IsPointOnConstraintLine(Tuple<float, float, float, string> constraint, Tuple<float, float> point)

{

// Проверка, лежит ли точка на линии ограничения

double leftSide = constraint.Item1 \* point.Item1 + constraint.Item2 \* point.Item2;

double rightSide = constraint.Item3;

return Math.Abs(leftSide - rightSide) < 1e-6;

}

private List<Tuple<float, float>> FilteringIntersection(List<Tuple<float, float, float, string>> constraints, List<Tuple<float, float>> intersections, string condition1, string condition2) =>

intersections.Where

(

pt =>

(condition1 == ">=" ? pt.Item1 >= 0 : pt.Item1 <= 0) &&

(condition2 == ">=" ? pt.Item2 >= 0 : pt.Item2 <= 0) &&

SatisfiesConstraints(pt, constraints)

).ToList();

private void FindFirstAndLastIntersection(List<Tuple<float, float>> intersections, double gradientX, double gradientY, out Tuple<float, float> firstIntersection, out Tuple<float, float> lastIntersection)

{

try

{

// Сортируем точки пересечения по проекции на направление градиента

intersections.Sort((a, b) =>

{

double projectionA = a.Item1 \* gradientX + a.Item2 \* gradientY;

double projectionB = b.Item1 \* gradientX + b.Item2 \* gradientY;

return projectionA.CompareTo(projectionB);

}

);

// Первая и последняя точки в отсортированном списке

firstIntersection = intersections.First();

lastIntersection = intersections.Last();

}

catch (Exception ex)

{

firstIntersection = new Tuple<float, float>(0f, 0f);

lastIntersection = new Tuple<float, float>(0f, 0f);

}

}

private List<Tuple<float, float>> FindIntersections(List<Tuple<float, float, float, string>> constraints)

{

var intersections = new List<Tuple<float, float>>(); for (int i = 0; i < constraints.Count; i++)

{

for (int j = i + 1; j < constraints.Count; j++)

{

try

{

var intersection = SolveEquations(constraints[i].Item1, constraints[i].Item2, constraints[i].Item3, constraints[j].Item1, constraints[j].Item2, constraints[j].Item3);

if (intersection.Item1 >= -1 && intersection.Item2 >= -1)

intersections.Add(intersection);

}

catch (InvalidOperationException)

{

}

}

}

return intersections;

}

private Tuple<float, float> SolveEquations(float a1, float b1, float c1, float a2, float b2, float c2)

{

float determinant = a1 \* b2 - a2 \* b1;

if (determinant == 0)

{

throw new InvalidOperationException("Прямые параллельны и не пересекаются.");

}

float x = (c1 \* b2 - c2 \* b1) / determinant;

float y = (a1 \* c2 - a2 \* c1) / determinant;

return new Tuple<float, float>(x, y);

}

private List<Tuple<float, float>> FindIntersectionWithAxes(float x1, float x2, float num, string sign)

{

var points = new List<Tuple<float, float>>();

// Проверка пересечения с осью X (Y = 0)

if (x1 != 0)

{

float x = num / x1;

if (x >= 0)

{

bool isSatisfied = (sign == "<=" && x <= 20) || (sign == ">=" && x >= 0);

if (isSatisfied)

{

points.Add(new Tuple<float, float>(x, 0));

}

}

}

// Проверка пересечения с осью Y (X = 0)

if (x2 != 0)

{

float y = num / x2;

if (y >= 0)

{

bool isSatisfied = (sign == "<=" && y <= 10) || (sign == ">=" && y >= 0);

if (isSatisfied)

{

points.Add(new Tuple<float, float>(0, y));

}

}

}

return points;

}

private bool SatisfiesConstraints(Tuple<float, float> point, List<Tuple<float, float, float, string>> constraints)

{

float x = point.Item1;

float y = point.Item2;

foreach (var constraint in constraints)

{

float a1 = constraint.Item1;

float a2 = constraint.Item2;

float b = constraint.Item3;

string sign = constraint.Item4;

if (!IsPointInRegion(x, y, a1, a2, b, sign))

return false;

}

return true;

}

private bool IsPointInRegion(float x, float y, float a1, float a2, float b, string sign, float epsilon = 2e-6f)

{

float lhs = a1 \* x + a2 \* y;

if (sign == "<=")

{

return (lhs <= b + epsilon);

}

else if (sign == ">=")

{

return (lhs >= b - epsilon);

}

return false; // На случай, если знак неравенства не "<=" и не ">="

}

}

}

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 SimplexMethod

{

public partial class MainScreen : System.Windows.Forms.Form

{

public MainScreen()

{

InitializeComponent();

}

int constraintsCount = 0;

int variablesCount = 0;

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

{

try

{

constraintsCount = Convert.ToInt32(nOfContraintsTextBox.Text);

variablesCount = Convert.ToInt32(nOfVariablesTextBox.Text);

fillConstraintsGrid();

fillFunctionGrid();

}

catch (Exception err)

{

MessageBox.Show(err.Message);

}

}

void fillConstraintsGrid()

{

constraintsGridView.Rows.Clear();

constraintsGridView.ColumnCount = variablesCount + 2;

constraintsGridView.RowHeadersVisible = false;

for (int i = 0; i < variablesCount + 2; i++)

{

constraintsGridView.Columns[i].Width = 50;

constraintsGridView.Columns[i].DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

if (i < variablesCount)

{

constraintsGridView.Columns[i].Name = "x" + (i + 1).ToString();

} else if (i == variablesCount)

{

constraintsGridView.Columns[i].Name = "Sign";

}

}

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

{

string[] row = new string[variablesCount + 2];

constraintsGridView.Rows.Add(row);

constraintsGridView.Rows[i].Height = 20;

}

}

void fillFunctionGrid()

{

functionGridView.Rows.Clear();

functionGridView.ColumnCount = variablesCount + 1;

functionGridView.RowHeadersVisible = false;

for (int i = 0; i < variablesCount + 1; i++)

{

functionGridView.Columns[i].Width = 50;

functionGridView.Columns[i].DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

if (i < variablesCount)

{

functionGridView.Columns[i].Name = "x" + (i + 1).ToString();

}

else

{

functionGridView.Columns[i].Name = "C";

}

}

string[] row = new string[variablesCount + 2];

constraintsGridView.Rows.Add(row);

constraintsGridView.Rows[0].Height = 20;

}

void Proceed()

{

Constraint[] constraints = new Constraint[constraintsCount];

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

{

double[] variables = new double[variablesCount];

double b = Convert.ToDouble(constraintsGridView.Rows[i].Cells[variablesCount + 1].Value);

string sign = Convert.ToString(constraintsGridView.Rows[i].Cells[variablesCount].Value);

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

{

variables[j] = Convert.ToDouble(constraintsGridView.Rows[i].Cells[j].Value);

}

constraints[i] = new Constraint(variables, b, sign);

}

double[] functionVariables = new double[variablesCount];

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

{

functionVariables[i] = Convert.ToDouble(functionGridView.Rows[0].Cells[i].Value);

}

double c = Convert.ToDouble(functionGridView.Rows[0].Cells[variablesCount].Value);

bool isExtrMax = extrComboBox.SelectedIndex == 0;

Function function = new Function(functionVariables, c, isExtrMax);

Simplex simplex = new Simplex(function, constraints);

Tuple<List<SimplexSnap>, SimplexResult> result = simplex.GetResult();

switch (result.Item2)

{

case SimplexResult.Found:

string extrStr = isExtrMax ? "max" : "min";

resultsLbl.Text = "The optimal solution is found: F" + extrStr + $" = {result.Item1.Last().fValue}";

break;

case SimplexResult.Unbounded:

resultsLbl.Text = "The domain of admissible solutions is unbounded";

break;

case SimplexResult.NotYetFound:

resultsLbl.Text = "Algorithm has made 100 cycles and hasn't found any optimal solution.";

break;

}

ShowResultsGrid(result.Item1);

}

void ShowResultsGrid(List<SimplexSnap> snaps)

{

resultsGridView.Rows.Clear();

resultsGridView.ColumnCount = snaps.First().matrix.Length + 4;

resultsGridView.RowHeadersVisible = false;

resultsGridView.ColumnHeadersVisible = false;

for (int i = 0; i < snaps.First().matrix.Length + 4; i++)

{

resultsGridView.Columns[i].Width = 50;

resultsGridView.Columns[i].DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

}

foreach (SimplexSnap snap in snaps)

{

string[] firstRow = new string[snaps.First().matrix.Length + 4];

firstRow[0] = "i";

firstRow[1] = "Basis";

firstRow[2] = "C";

firstRow[3] = "B";

for (int i = 4; i < snaps.First().matrix.Length + 4; i++)

{

firstRow[i] = $"A{i - 3}";

}

resultsGridView.Rows.Add(firstRow);

for (int i = 0; i < snaps.First().C.Length; i++)

{

string[] row = new string[snaps.First().matrix.Length + 4];

for (int j = 0; j < snaps.First().matrix.Length + 4; j++)

{

if (j == 0)

{

row[j] = (i + 1).ToString();

}

else if (j == 1)

{

row[j] = $"A{snap.C[i] + 1}";

}

else if (j == 2)

{

row[j] = snap.m[snap.C[i]] ? "-M" : $"{snap.fVars[snap.C[i]]}";

}

else if (j == 3)

{

row[j] = Round(snap.b[i]).ToString();

}

else

{

row[j] = Round(snap.matrix[j - 4][i]).ToString();

}

}

resultsGridView.Rows.Add(row);

}

string[] fRow = new string[snaps.First().matrix.Length + 4];

fRow[0] = "m+1";

fRow[1] = "F";

fRow[2] = "Δj";

fRow[3] = Round(snap.fValue).ToString();

for (int i = 4; i < snaps.First().matrix.Length + 4; i++)

{

fRow[i] = Round(snap.F[i - 4]).ToString();

}

resultsGridView.Rows.Add(fRow);

if (!snap.isMDone)

{

string[] mRow = new string[snaps.First().matrix.Length + 4];

mRow[0] = "m+2";

mRow[1] = "M";

mRow[2] = "Δj";

mRow[3] = "";

for (int i = 4; i < snaps.First().matrix.Length + 4; i++)

{

mRow[i] = Round(snap.M[i - 4]).ToString();

}

resultsGridView.Rows.Add(mRow);

}

string[] emptyRow = new string[snaps.First().matrix.Length + 4];

resultsGridView.Rows.Add(emptyRow);

}

}

double Round(double a)

{

return Math.Round(a, 2);

}

void fillDefaultsConstraints(double[][] consMatrx, string[] signs, double[] freeVars)

{

constraintsCount = signs.Length;

nOfContraintsTextBox.Text = constraintsCount.ToString();

variablesCount = consMatrx.First().Length;

nOfVariablesTextBox.Text = variablesCount.ToString();

fillConstraintsGrid();

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

{

for (int j = 0; j < variablesCount + 2; j++)

{

if (j < variablesCount)

{

constraintsGridView.Rows[i].Cells[j].Value = consMatrx[i][j];

}

else if (j < variablesCount + 1)

{

constraintsGridView.Rows[i].Cells[j].Value = signs[i];

}

else if (j < variablesCount + 2)

{

constraintsGridView.Rows[i].Cells[j].Value = freeVars[i];

}

}

}

}

void fillDefaultsFunction(double[] funcVars, double c, bool isExtrMax)

{

fillFunctionGrid();

for (int i = 0; i < variablesCount + 1; i++)

{

if (i < variablesCount)

{

functionGridView.Rows[0].Cells[i].Value = funcVars[i];

}

else

{

functionGridView.Rows[0].Cells[i].Value = c;

}

}

extrComboBox.SelectedIndex = isExtrMax ? 0 : 1;

}

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

{

Proceed();

}

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

{

Problem p = ProblemsService.shared.GetNext();

fillDefaultsConstraints(p.consMatrx, p.signs, p.freeVars);

fillDefaultsFunction(p.funcVars, p.c, p.isExtrMax);

Proceed();

}

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

{

nOfContraintsTextBox.Clear();

nOfVariablesTextBox.Clear();

resultsGridView.Columns.Clear();

functionGridView.Columns.Clear();

constraintsGridView.Columns.Clear();

extrComboBox.SelectedIndex = -1;

resultsLbl.ResetText();

}

}

} namespace TransportTask

{

public partial class TransportProblemSolving : Form

{

Tuple<Control[,], Control[], Control[]> textBoxes { get; set; }

Tuple<Control[,], Control[], Control[]> step2 { get; set; }

Tuple<Control[,], Control[], Control[]> step3 { get; set; }

Tuple<Control[,], Control[], Control[]> step4 { get; set; }

Tuple<Control[,], Control[], Control[]> step5 { get; set; }

bool Step1, Step2, Step3, Step4;

int Rows, Columns;

int ControlWidth;

int[,] costs;

int[] stocks, needs;

int CountOfFilledCells;

static bool check = true, IsOptimal = false, firstClick = true;

int[,] allocation;

int[,] costsControls;

int minDeltaRow = -1, minDeltaCol = -1;

public TransportProblemSolving()

{

InitializeComponent();

StartPosition = FormStartPosition.CenterScreen;

int w = Screen.PrimaryScreen.Bounds.Width;

int h = Screen.PrimaryScreen.Bounds.Height;

MinimumSize = new Size((int)(w / 1.5), (int)(h / 1.5));

ControlWidth = (int)(Width \* 0.05);

Resize += Form1\_Resize;

InitializeTabControl();

}

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

{

firstClick = true;

for (int i = 0; i < 5; i++)

DeleteBoxesFromPage(i + 1);

Step1 = false;

Step2 = false;

Step3 = false;

Step4 = false;

textBoxes = InitializeAddComponents();

var control = Controls.Find("tabControl", false)[0].Controls.Find("Page1", false)[0].Controls.Find("ResultOfCheck", false)[0];

control.BackColor = BackColor;

control.Text = "";

Controls.Find("tabControl", false)[0].Controls.Find("Page2", false)[0].Controls.Find("Continue", false)[0].Text = "";

}

private void DeleteBoxesFromPage(int page)

{

var Boxes = Controls.Find("tabControl", false)[0].Controls.Find($"Page{page}", false)[0].Controls.Find("TransportTable", false);

if (Boxes != null)

{

foreach (var box in Boxes)

{

Controls.Find("tabControl", false)[0].Controls.Find($"Page{page}", false)[0].Controls.Remove(box);

}

}

}

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

{

ControlWidth = (int)(Width \* 0.05);

var control = Controls.Find("tabControl", false)[0];

control.Size = new Size((int)(Width \* 0.9), (int)(Height \* 0.85));

control.Location = new Point((Width - control.Width) / 2, (Height - control.Height) / 2);

}

private void ResizeInnerComponents(object? sender, EventArgs e)

{

try

{

UpdateLocation(textBoxes, StartY);

if (Step1)

UpdateLocation(step2, step2.Item1[0, 0].Location.Y);

if (Step2)

UpdateLocation(step3, step3.Item1[0, 0].Location.Y);

if (Step3)

UpdateLocation(step4, step4.Item1[0, 0].Location.Y);

if (Step4)

UpdateLocation(step5, step5.Item1[0, 0].Location.Y);

}

catch (NullReferenceException) { }

}

private void FillTable(object? sender, EventArgs e)

{

try

{

Random random = new Random();

// Заполнение массивов случайными значениями

for (int i = 0; i < textBoxes.Item3.Length; i++)

{

textBoxes.Item3[i].Text = random.Next(10, 100).ToString(); // Случайные значения для запасов (от 10 до 100)

}

for (int j = 0; j < textBoxes.Item2.Length; j++)

{

textBoxes.Item2[j].Text = random.Next(10, a100).ToString(); // Случайные значения для потребностей (от 10 до 100)

}

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

{

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

{

textBoxes.Item1[i, j].Text = random.Next(1, 50).ToString(); // Случайные значения для тарифов (от 1 до 50)

}

}

}

catch (NullReferenceException)

{

MessageBox.Show("Создай таблицу");

}

}

private void CheckForBalance(object? sender, EventArgs e)

{

for (int i = 1; i < 5; i++)

DeleteBoxesFromPage(i + 1);

Step2 = false;

Step3 = false;

Step4 = false;

var control = Controls.Find("tabControl", false)[0].Controls.Find("Page1", false)[0].Controls.Find("ResultOfCheck", false)[0];

try

{

int NeedsSum = Sum(textBoxes.Item2);

int StocksSum = Sum(textBoxes.Item3);

if (NeedsSum == StocksSum)

{

control.Location = new Point(textBoxes.Item2[0].Location.X, textBoxes.Item2[0].Location.Y + textBoxes.Item2[0].Height + (int)(Height \* 0.02));

control.Text = "Go to the next step solving of problem";

control.BackColor = Color.LightGreen;

needs = new int[textBoxes.Item2.Length];

for (int i = 0; i < needs.Length; i++)

needs[i] = int.Parse(textBoxes.Item2[i].Text);

stocks = new int[textBoxes.Item3.Length];

for (int i = 0; i < stocks.Length; i++)

stocks[i] = int.Parse(textBoxes.Item3[i].Text);

costs = new int[textBoxes.Item1.GetLength(0), textBoxes.Item1.GetLength(1)];

for (int i = 0; i < costs.GetLength(0); i++)

for (int j = 0; j < costs.GetLength(1); j++)

costs[i, j] = int.Parse(textBoxes.Item1[i, j].Text);

Step1 = true;

InitTwoStep();

}

else if (NeedsSum > StocksSum)

{

AddRow(textBoxes.Item1, textBoxes.Item3, NeedsSum - StocksSum);

control.Location = new Point(textBoxes.Item2[0].Location.X, textBoxes.Item2[0].Location.Y + textBoxes.Item2[0].Height + (int)(Height \* 0.02));

control.Text = "Check the balance again";

}

else

{

AddCol(textBoxes.Item1, textBoxes.Item2, StocksSum - NeedsSum);

control.Location = new Point(textBoxes.Item2[0].Location.X, textBoxes.Item2[0].Location.Y + textBoxes.Item2[0].Height + (int)(Height \* 0.02));

control.Text = "Check the balance again";

}

}catch(Exception)

{

Step2 = false;

control.Text = "";

control.BackColor = BackColor;

MessageBox.Show("Заполните все поля!!!");

}

}

private void InitTwoStep()

{

Control[] stocks2 = new Control[stocks.Length];

Control[] needs2 = new Control[needs.Length];

Control[,] costs2 = new Control[costs.GetLength(0), costs.GetLength(1)];

for (int i = 0; i < stocks2.Length; i++)

{

var textBox = CreateTextBox(ControlWidth, Color.Yellow, stocks[i].ToString());

textBox.ReadOnly = true;

stocks2[i] = textBox;

Controls.Find("tabControl", false)[0].Controls.Find($"Page2", false)[0].Controls.Add(stocks2[i]);

}

for (int i = 0; i < needs2.Length; i++)

{

var textBox = CreateTextBox(ControlWidth, Color.Yellow, needs[i].ToString());

textBox.ReadOnly = true;

needs2[i] = textBox;

Controls.Find("tabControl", false)[0].Controls.Find($"Page2", false)[0].Controls.Add(needs2[i]);

}

for (int i = 0; i < costs2.GetLength(0); i++)

{

for (int j = 0; j < costs2.GetLength(1); j++)

{

var textBox = CreateTextBox(ControlWidth, BackColor, "");

textBox.ReadOnly = true;

costs2[i, j] = textBox;

Controls.Find("tabControl", false)[0].Controls.Find($"Page2", false)[0].Controls.Add(costs2[i, j]);

}

}

step2 = new(costs2, needs2, stocks2);

UpdateLocation(step2, textBoxes.Item1[0, 0].Location.Y);

}

private void InitThreeStep()

{

Control[] stocks3 = new Control[stocks.Length];

Control[] needs3 = new Control[needs.Length];

Control[,] costs3 = new Control[costs.GetLength(0), costs.GetLength(1)];

for (int i = 0; i < stocks3.Length; i++)

{

var textBox = CreateTextBox(ControlWidth, Color.Yellow, step2.Item3[i].Text);

textBox.ReadOnly = true;

stocks3[i] = textBox;

Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Add(stocks3[i]);

}

for (int i = 0; i < needs3.Length; i++)

{

var textBox = CreateTextBox(ControlWidth, Color.Yellow, step2.Item2[i].Text);

textBox.ReadOnly = true;

needs3[i] = textBox;

Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Add(needs3[i]);

}

for (int i = 0; i < costs3.GetLength(0); i++)

{

for (int j = 0; j < costs3.GetLength(1); j++)

{

var textBox = CreateTextBox(ControlWidth, BackColor, step2.Item1[i, j].Text);

textBox.ReadOnly = true;

costs3[i, j] = textBox;

Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Add(costs3[i, j]);

}

}

step3 = new(costs3, needs3, stocks3);

UpdateLocation(step3, (int)(Height \* 0.02));

var c = Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("Description3", false)[0];

c.Location = new Point

(

step3.Item3[0].Location.X + step3.Item3[0].Width + (int)(Width \* 0.02), step3.Item3[0].Location.Y

);

var c2 = Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("checkForDegenetacy", false)[0];

c2.Location = new Point

(

c.Location.X + c.Width / 4, c.Location.Y + c.Height + (int)(Height \* 0.02)

);

}

private void InitFourStep()

{

int page = 4;

Control[,] costs4 = CreateTransportTable(step3.Item1.GetLength(0), step3.Item1.GetLength(1), ControlWidth, page);

Control[] stocks4 = CreateStocks(stocks.Length, ControlWidth, page);

Control[] needs4 = CreateNeeds(needs.Length, ControlWidth, page);

for (int i = 0; i < costs4.GetLength(0); i++)

{

for (int j = 0; j < costs4.GetLength(1); j++)

{

if (step3.Item1[i, j].Text != "0" && step3.Item1[i, j].Text != "")

costs4[i, j].Text = step3.Item1[i, j].Text + "\*" + costs[i, j].ToString();

else

costs4[i, j].Text = "";

(costs4[i, j] as TextBox).ReadOnly = true;

}

}

for (int i = 0; i < stocks4.Length; i++)

(stocks4[i] as TextBox).ReadOnly = true;

for (int i = 0; i < needs4.Length; i++)

(needs4[i] as TextBox).ReadOnly = true;

step4 = new(costs4, needs4, stocks4);

UpdateLocation(step4, step3.Item1[0, 0].Location.Y);

var c = Controls.Find("tabControl", false)[0].Controls.Find($"Page4", false)[0].Controls.Find("Description4", false)[0];

c.Location = new Point

(

step4.Item3[0].Location.X + step4.Item3[0].Width + (int)(Width \* 0.02), step4.Item3[0].Location.Y

);

var c2 = Controls.Find("tabControl", false)[0].Controls.Find($"Page4", false)[0].Controls.Find("Calculate", false)[0];

c2.Location = new Point

(

c.Location.X + c.Width / 4, c.Location.Y + c.Height + (int)(Height \* 0.02)

);

}

private void InitFiveStep()

{

int page = 5;

Control[,] costs5 = CreateTransportTable(step3.Item1.GetLength(0), step3.Item1.GetLength(1), ControlWidth, page);

Control[] stocks5 = CreateStocks(stocks.Length, ControlWidth, page);

Control[] needs5 = CreateNeeds(needs.Length, ControlWidth, page);

for (int i = 0; i < costs5.GetLength(0); i++)

for (int j = 0; j < costs5.GetLength(1); j++)

(costs5[i, j] as TextBox).ReadOnly = true;

for (int i = 0; i < stocks5.Length; i++)

(stocks5[i] as TextBox).ReadOnly = true;

for (int i = 0; i < needs5.Length; i++)

(needs5[i] as TextBox).ReadOnly = true;

step5 = new(costs5, needs5, stocks5);

UpdateLocation(step5, step4.Item1[0, 0].Location.Y);

var c = Controls.Find("tabControl", false)[0].Controls.Find($"Page5", false)[0].Controls.Find("Description5", false)[0];

c.Location = new Point

(

step5.Item3[0].Location.X + step5.Item3[0].Width + (int)(Width \* 0.02), step5.Item3[0].Location.Y

);

var c2 = Controls.Find("tabControl", false)[0].Controls.Find($"Page5", false)[0].Controls.Find("MakePlan", false)[0];

c2.Location = new Point

(

c.Location.X + c.Width / 4, c.Location.Y + c.Height + (int)(Height \* 0.02)

);

}

private void SolveProblem(object? sender, EventArgs e)

{

Step3 = false;

Step4 = false;

if (Step1)

{

Controls.Find("tabControl", false)[0].Controls.Find($"Page4", false)[0].Controls.Find("CostFirstPlan", false)[0].Text = "";

Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("Degeneracy", false)[0].Text = "";

var button = Controls.Find("tabControl", false)[0].Controls.Find("Page2", false)[0].Controls.Find("FindPlan", false)[0];

var comboBox = Controls.Find("tabControl", false)[0].Controls.Find($"Page2", false)[0].Controls.Find("NWAOrMinEl", false)[0];

var labelContinue = Controls.Find("tabControl", false)[0].Controls.Find("Page2", false)[0].Controls.Find("Continue", false)[0];

labelContinue.Text = "Первоначальный план найден.\n\nПереходи к следующему шагу.";

comboBox.Location = new Point(textBoxes.Item3[0].Location.X + textBoxes.Item3[0].Width + (int)(Width \* 0.02), textBoxes.Item3[0].Location.Y);

button.Location = new Point(comboBox.Location.X + comboBox.Width + (int)(Width \* 0.02), comboBox.Location.Y);

labelContinue.Location = new Point(comboBox.Location.X, comboBox.Location.Y + comboBox.Height + (int)(Height \* 0.02));

for (int i = 0; i < step2.Item1.GetLength(0); i++)

for (int j = 0; j < step2.Item1.GetLength(1); j++)

step2.Item1[i, j].Text = "";

if ((comboBox as ComboBox).SelectedItem == null)

(comboBox as ComboBox).SelectedItem = "Минимального элемента";

DeleteBoxesFromPage(4);

if ((comboBox as ComboBox)?.SelectedItem == "Минимального элемента")

{

CountOfFilledCells = SolveByMinCost();

Step2 = true;

}

else

{

CountOfFilledCells = SolveProblemNWA();

Step2 = true;

}

try

{

var controls = Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("TransportTable", false);

foreach (var c in controls)

Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Remove(c);

}

catch (Exception ex)

{

}

InitThreeStep();

}

}

private int SolveProblemNWA()

{

Control[,] costControls = step2.Item1;

Control[] needsControls = step2.Item2;

Control[] stocksControls = step2.Item3;

int rows = stocksControls.Length;

int columns = needsControls.Length;

int[,] allocation = new int[rows, columns];

int[] remainingStocks = (int[])(stocks.Clone());

int[] remainingNeeds = (int[])(needs.Clone());

int[,] costs2 = (int[,])costs.Clone();

int row = 0, col = 0;

while (row < rows && col < columns)

{

int allocationAmount = Math.Min(remainingStocks[row], remainingNeeds[col]);

allocation[row, col] = allocationAmount;

remainingStocks[row] -= allocationAmount;

remainingNeeds[col] -= allocationAmount;

if (remainingStocks[row] == 0)

{

row++;

}

else

{

col++;

}

}

// Запись результатов обратно в контролы

int count = 0;

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

{

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

{

if (allocation[i, j] == 0)

costControls[i, j].Text = "";

else

{

costControls[i, j].Text = allocation[i, j].ToString();

count++;

}

}

}

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

step2.Item3[i].Text = remainingStocks[i].ToString();

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

step2.Item2[j].Text = remainingNeeds[j].ToString();

return count;

}

private int SolveByMinCost()

{

Control[,] costControls = step2.Item1;

Control[] needsControls = step2.Item2;

Control[] stocksControls = step2.Item3;

int rows = stocksControls.Length;

int columns = needsControls.Length;

int[,] allocation = new int[rows, columns];

int[] remainingStocks = (int[])(stocks.Clone());

int[] remainingNeeds = (int[])(needs.Clone());

int[,] costs2 = (int[,])costs.Clone();

bool[,] allocated = new bool[rows, columns];

while (true)

{

int minCost = int.MaxValue; int minRow = -1, minCol = -1;

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

{

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

{

if (!allocated[i, j] && remainingStocks[i] > 0 && remainingNeeds[j] > 0)

{

int cost = costs2[i, j];

if (cost < minCost)

{

minCost = cost;

minRow = i;

minCol = j;

}

}

}

}

if (minRow == -1 || minCol == -1)

break;

int allocationAmount = Math.Min(remainingStocks[minRow], remainingNeeds[minCol]);

allocation[minRow, minCol] = allocationAmount;

remainingStocks[minRow] -= allocationAmount;

remainingNeeds[minCol] -= allocationAmount;

allocated[minRow, minCol] = true;

}

int count = 0;

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

{

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

{

if (allocation[i, j] > 0){

count++;

costControls[i, j].Text = allocation[i, j].ToString();

}

}

}

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

step2.Item3[i].Text = remainingStocks[i].ToString();

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

step2.Item2[j].Text = remainingNeeds[j].ToString();

return count;

}

private void checkForDegenetacy(object? sender, EventArgs e)

{

if (Step2)

{

var label = Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("Degeneracy", false)[0];

var c1 = Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("Description3", false)[0];

var c2 = Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Find("checkForDegenetacy", false)[0];

if (CountOfFilledCells == Rows + Columns - 1)

{

(label as Label).Text = $"{CountOfFilledCells} = {Rows} + {Columns} - 1";

(label as Label).BackColor = Color.LightGreen;

(label as Label).ForeColor = Color.Black;

label.Location = new Point(c1.Location.X + label.Width / 4, c2.Location.Y + c2.Height + (int)(Height \* 0.02));

Step3 = true;

try

{

var controls = Controls.Find("tabControl", false)[0].Controls.Find($"Page4", false)[0].Controls.Find("TransportTable", false);

foreach (var c in controls)

Controls.Find("tabControl", false)[0].Controls.Find($"Page3", false)[0].Controls.Remove(c);

}

catch (Exception ex) { }

InitFourStep();

}

else

{

(label as Label).Text = $"{CountOfFilledCells} are not equal {Rows} + {Columns} - 1";

(label as Label).BackColor = Color.Red;

(label as Label).ForeColor = Color.White;

}

}

}

private void CalcCost(object? sender, EventArgs e)

{

firstClick = true;

if (Step3)

{

var c = Controls.Find("tabControl", false)[0].Controls.Find($"Page4", false)[0].Controls.Find("Calculate", false)[0];

var label = Controls.Find("tabControl", false)[0].Controls.Find($"Page4", false)[0].Controls.Find("CostFirstPlan", false)[0];

int cost = 0;

for (int i = 0; i < step4.Item1.GetLength(0); i++)

{

for (int j = 0; j < step4.Item1.GetLength(1); j++)

{

if (step4.Item1[i, j].Text != "")

{

cost += Convert.ToInt32(new DataTable().Compute(step4.Item1[i, j].Text, null));

label.BackColor = Color.LightGreen;

}

}

}

label.Text = $"Total cost is {cost}";

label.Location = new Point(c.Location.X, c.Location.Y + c.Height + (int)(Height \* 0.02));

Step4 = true;

InitFiveStep();

}

}

private async void MakePlan(Object sender, EventArgs e)

{

{

if (Step4)

{

if (firstClick)

{

Control[,] costControls = step3.Item1;

Control[] needsControls = step3.Item2;

Control[] stocksControls = step3.Item3;

Rows = stocksControls.Length;

Columns = needsControls.Length;

allocation = new int[Rows, Columns];

costsControls = new int[costControls.GetLength(0), costControls.GetLength(1)];

for (int i = 0; i < step3.Item1.GetLength(0); i++)

{

for (int j = 0; j < step3.Item1.GetLength(1); j++)

{

if (costControls[i, j].Text != "")

{

allocation[i, j] = int.Parse(step3.Item1[i, j].Text);

}

else allocation[i, j] = 0;

}

}

firstClick = false;

}

for (int i = 0; i < allocation.GetLength(0); i++)

for (int j = 0; j < allocation.GetLength(1); j++)

if (allocation[i, j] > 0)

costsControls[i, j] = int.Parse(textBoxes.Item1[i, j].Text);

// Вычисляем потенциалы

double[] u = new double[Rows];

double[] v = new double[Columns];

CalculatePotentials(Rows, Columns, costsControls, u, v);

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

step5.Item3[i].Text = u[i].ToString();

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

step5.Item2[j].Text = v[j].ToString();

// Оценка небазисных переменных

double minDelta = double.MaxValue;

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

{

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

{

if (allocation[i, j] == 0)

{

double delta = costs[i, j] - u[i] - v[j];

step5.Item1[i, j].Text = delta.ToString();

if (delta < minDelta)

{

minDelta = delta;

minDeltaRow = i;

minDeltaCol = j;

}

}

else

{

step5.Item1[i, j].Text = "0";

}

}

}

if (minDelta >= 0)

IsOptimal = true;

var label = Controls.Find("tabControl", false)[0].Controls.Find($"Page5", false)[0].Controls.Find("CostSecondPlan", false)[0];

if (IsOptimal)

{

int cost = 0;

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

{

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

{

step5.Item1[i, j].Text = allocation[i, j].ToString();

cost += allocation[i, j] \* costs[i, j];

}

}

label.Text = $"Total cost is {cost}";

}

else

{

try

{

ImprovePlan(allocation, minDeltaRow, minDeltaCol);

}

catch (NullReferenceException)

{

MessageBox.Show("Не удалось улучшить план. :)");

}

}

}

};

}

private void CalculatePotentials(int rows, int columns, int[,] costs, double[] u, double[] v)

{

bool[] uDetermined = new bool[rows];

bool[] vDetermined = new bool[columns];

// начальное значение потенциала

u[0] = 0;

uDetermined[0] = true;

bool changed;

do

{

changed = false;

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

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

{

if (costs[i, j] != 0)

{

if (uDetermined[i] && !vDetermined[j])

{

v[j] = costs[i, j] - u[i];

vDetermined[j] = true;

changed = true;

step5.Item2[j].Text = v[j].ToString();

}

else if (!uDetermined[i] && vDetermined[j])

{

u[i] = costs[i, j] - v[j];

uDetermined[i] = true;

changed = true;

step5.Item3[i].Text = u[i].ToString();

}

}

}

//} while (((count < (columns + rows - (rows + columns - 1) \* 2)) && (tmp < columns \* rows)));

} while (changed);

}

private static void ImprovePlan(int[,] allocation, int startRow, int startCol)

{

int rows = allocation.GetLength(0);

int columns = allocation.GetLength(1);

// Найти цикл

var cycle = FindCycle(allocation, startRow, startCol);

// Перераспределить запасы и потребности вдоль цикла

int minAllocation = int.MaxValue;

for (int i = 1; i < cycle.Count; i += 2)

{

int row = cycle[i].Item1;

int col = cycle[i].Item2;

minAllocation = Math.Min(minAllocation, allocation[row, col]);

}

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

{

int row = cycle[i].Item1;

int col = cycle[i].Item2;

if (i % 2 == 0)

{

allocation[row, col] += minAllocation;

}

else

{

allocation[row, col] -= minAllocation;

}

}

}

private static List<(int, int)> FindCycle(int[,] allocation, int startRow, int startCol)

{

int rows = allocation.GetLength(0);

int columns = allocation.GetLength(1);

var path = new List<(int, int)> { (startRow, startCol) };

var visited = new HashSet<(int, int)>();

var columnCount = new Dictionary<int, int>();

var rowCount = new Dictionary<int, int>();

if (!columnCount.ContainsKey(startCol)) { columnCount[startCol] = 0; }

if (!rowCount.ContainsKey(startRow)) { rowCount[startRow] = 0; }

columnCount[startCol]++;

rowCount[startRow]++;

//visited.Add((startRow, startCol));

if (FindCycleHelper(allocation, startRow, startCol, path, visited, columnCount, rowCount))

{

return path;

}

return null;

}

private static bool FindCycleHelper(int[,] allocation, int row, int col, List<(int, int)> path, HashSet<(int, int)> visited, Dictionary<int, int> columnCount, Dictionary<int, int> rowCount)

{

int rows = allocation.GetLength(0);

int columns = allocation.GetLength(1);

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

{

if (check && i != row && (allocation[i, col] != 0 || (i, col) == path[0]) && (!visited.Contains((i, col)) /\*|| (i, col) == path[0]\*/))

{

//if (!columnCount.ContainsKey(col)) { columnCount[col] = 0; }

//if (!rowCount.ContainsKey(i)) { rowCount[i] = 0; }

//if (columnCount[col] >= 2 || (rowCount.ContainsKey(i) && rowCount[i] >= 2)) { continue; }

visited.Add((i, col));

path.Add((i, col));

//columnCount[col]--;

//rowCount[i]++;

check = !check;

if ((i, col) == path[0] || FindCycleHelper(allocation, i, col, path, visited, columnCount, rowCount))

{

return true;

}

path.RemoveAt(path.Count - 1);

visited.Remove((i, col));

//rowCount[i]--;

//columnCount[col]--;

check = !check;

}

}

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

{

if (!check && j != col && (allocation[row, j] != 0 || (row, j) == path[0]) && (!visited.Contains((row, j)) /\*|| (row, j) == path[0]\*/))

{

//if (!rowCount.ContainsKey(row)) { rowCount[row] = 0; }

//if (!columnCount.ContainsKey(j)) { columnCount[j] = 0; }

//if (rowCount[row] >= 2 || (columnCount.ContainsKey(j) && columnCount[j] >= 2)) { continue; }

visited.Add((row, j));

path.Add((row, j));

//rowCount[row]++;

//columnCount[j]++;

check = !check;

if ((row, j) == path[0] || FindCycleHelper(allocation, row, j, path, visited, columnCount, rowCount)){ return true;}

path.RemoveAt(path.Count - 1);

visited.Remove((row, j));

//rowCount[row]--;

//columnCount[j]--;

check = !check;

} } return false;}}}