Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Специальность ПОИТ

**Контрольная Работа**

По дисциплине «Методы и алгоритмы принятия решений»

Лабораторная работа 6

Выполнил: студент гр. 881072 Пискарёв К.А.

Проверил: Бакунов А.М.

Минск 2020

**ЛАБОРАТОРНАЯ РАБОТА №6**

**КЛАССИФИКАЦИЯ ОБЪЕКТОВ МЕТОДОМ ИЕРАРХИЧЕСКОГО ГРУППИРОВАНИЯ**

Цель работы: изучить правила построения иерархических группировок, а также метод классификации объектов на основе иерархических группировок.

Порядок выполнения работы

1. Ознакомление с теоретической частью лабораторной работы.

2. Реализация классификации объектов с помощью иерархий.

3. Оформление отчета по лабораторной работе.

Исходные данные:

1. n – количество объектов группирования.

2. Таблица расстояний между объектами. Таблица заполняется автоматически случайными значениями.

Выходные данные: иерархии, построенные по критериям минимума и максимума. Результаты работы программы должны представляться в графическом виде

**ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ**

После запуска программы отображается форма, представленная на рисунке 1.

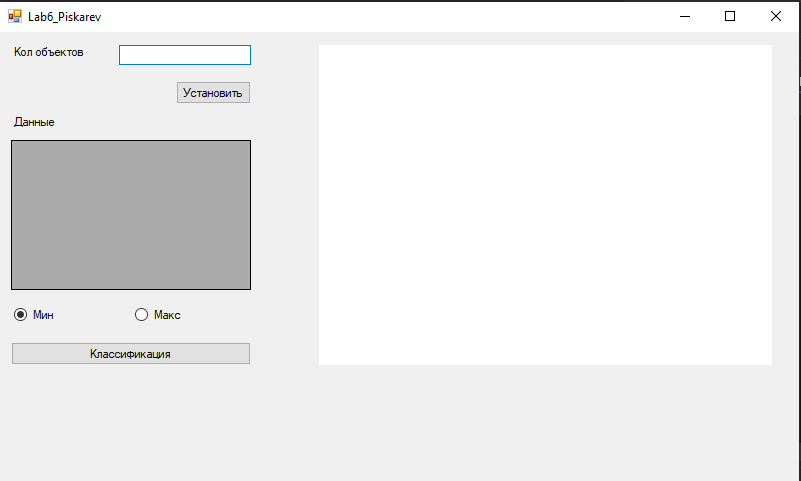


Рисунок 1

Требуется заполнить количество объектов установить, для первичной инициализации. Результат представлен на рисунке 2.

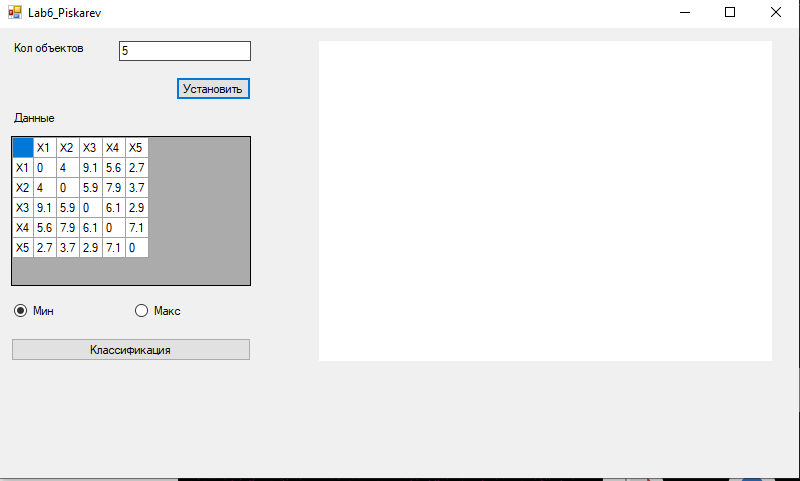


Рисунок 2

Для запуска алгоритма требуется выбрать режим классификации Ми или Макс после чего нажать на кнопку Классификация. Пример классификации на режиме Мин представлен на рисунке 3.

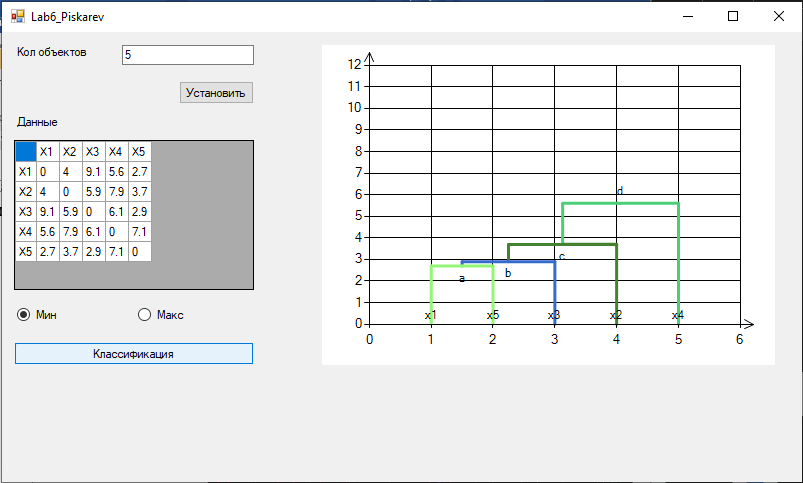


Рисунок 3

Пример классификации на режиме Макс с теми же исходными данными представлен на рисунке 4.

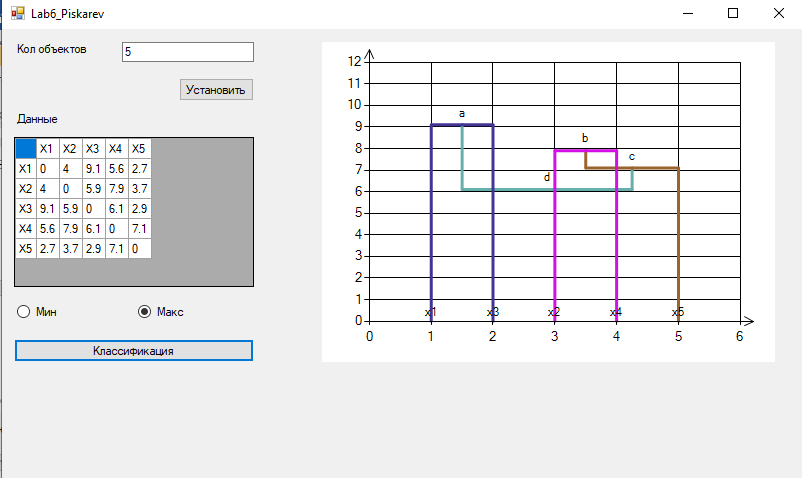


Рисунок 4

**ПРИЛОЖЕНИЕ**

Код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace lab6

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

public class HPoint

{

public double X { get; set; }

public double Y { get; set; }

public HPoint(double x, double y)

{

X = x;

Y = y;

}

public HPoint()

{

X = 0;

Y = 0;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

public class Group : HPoint

{

public List<Distance> Distances = new List<Distance>();

public List<Group> SubGroups = new List<Group>();

public string Name;

public Group(double x, double y): base(x,y) {

}

public Group() : base()

{

}

public Distance GetDistance(Group group)

{

return Distances.FirstOrDefault(distance => distance.Group.Equals(group));

}

public void DeleteDistances(List<Group> deleteList)

{

foreach (Group deleteGroup in deleteList)

{

var deleteDistances = (Distances.Where(distance => distance.Group.Equals(deleteGroup))).ToList();

foreach (Distance deleteDistance in deleteDistances)

Distances.Remove(deleteDistance);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab6

{

public class Distance

{

public double Value { get; set; }

public Group Group { get; set; }

public Distance(double value, Group group)

{

Value = value;

Group = group;

}

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.DataVisualization.Charting;

namespace lab6

{

public class Algorithm

{

private List<Group> groups;

private List<string> namesInUI;

private Random rand = new Random();

private int offsetX = 1;

private int charNext = 0;

public Algorithm()

{

}

public void SetData(int count, double[,] data)

{

offsetX = 1; charNext = 0;

groups = new List<Group>();

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

{

groups.Add(new Group());

groups[i].Name = "x" + (i + 1);

}

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

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

if (i != j)

groups[i].Distances.Add(new Distance(data[i, j], groups[j]));

}

public void Do(bool isMax)

{

var result = false;

do

{

result = false;

double minDistance = int.MaxValue;

var groupsWithMinDistance = new List<Group>();

foreach (Group group in groups)

{

foreach (Distance distance in group.Distances)

{

if (distance.Value < minDistance)

{

minDistance = distance.Value;

result = true;

groupsWithMinDistance.Clear();

groupsWithMinDistance.Add(group);

}

else if(distance.Value == minDistance)

{

groupsWithMinDistance.Add(group);

}

}

}

if (result && groupsWithMinDistance.Any())

SetNewGroups(groupsWithMinDistance, minDistance, isMax);

} while (result);

}

private char NextChar()

{

return (char)('a' + charNext++);

}

public void Draw(Chart chart)

{

namesInUI = new List<string> ();

SetDefaultChart(chart);

foreach (Group subGroup in groups)

DrawSubGroups(subGroup, chart);

}

private void SetDefaultChart(Chart chart)

{

chart.Series.Clear();

chart.ChartAreas[0].AxisX.ArrowStyle = chart.ChartAreas[0].AxisY.ArrowStyle = AxisArrowStyle.Lines;

chart.ChartAreas[0].AxisX.Crossing = chart.ChartAreas[0].AxisY.Crossing = 0;

chart.ChartAreas[0].AxisX.IsStartedFromZero = chart.ChartAreas[0].AxisY.IsStartedFromZero = true;

chart.ChartAreas[0].AxisX.Title = chart.ChartAreas[0].AxisY.Title = "";

chart.ChartAreas[0].AxisX.Interval = chart.ChartAreas[0].AxisY.Interval = 1;

chart.ChartAreas[0].AxisX.LineWidth= chart.ChartAreas[0].AxisY.LineWidth = 1;

chart.ChartAreas[0].AxisX.Minimum = chart.ChartAreas[0].AxisY.Minimum = 0;

chart.ChartAreas[0].AxisX.Maximum = offsetX;

chart.ChartAreas[0].AxisY.Maximum = 12;

}

private void DrawSubGroups(Group group, Chart chart)

{

bool res = true;

foreach (Series currSeries in chart.Series)

if (currSeries.Name == group.Name)

res = false;

if (res)

{

var pointsSeries = new Series { ChartType = SeriesChartType.Point, IsVisibleInLegend = false };

pointsSeries.Name = group.Name;

pointsSeries.MarkerSize = 1;

pointsSeries.MarkerColor = NewColor();

pointsSeries.Points.AddXY(group.X, group.Y);

if (chart.Series.IndexOf(pointsSeries) == -1)

chart.Series.Add(pointsSeries);

foreach (Group subGroup in group.SubGroups)

{

var lineSeries = new Series { ChartType = SeriesChartType.Line, IsVisibleInLegend = false };

lineSeries.BorderWidth = 3;

lineSeries.Name = group.Name + " " + subGroup.Name;

lineSeries.Color = pointsSeries.MarkerColor;

SetLabel(lineSeries, lineSeries.Points.AddXY(subGroup.X, subGroup.Y), subGroup.Name);

lineSeries.Points.AddXY(subGroup.X, group.Y);

SetLabel(lineSeries, lineSeries.Points.AddXY(group.X, group.Y), group.Name);

res = true;

foreach (Series currSeries in chart.Series)

if (currSeries.Name == lineSeries.Name)

res = false;

if (res)

chart.Series.Add(lineSeries);

DrawSubGroups(subGroup, chart);

}

}

}

private void SetLabel(Series series, int id, string label)

{

if (!namesInUI.Contains(label))

{

series.Points[id].Label = label;

namesInUI.Add(label);

}

}

private void SetNewGroups(List<Group> data, double minDistance, bool isMax)

{

var newGroup = new Group();

newGroup.Name = NextChar().ToString();

foreach (Group group in groups)

{

if (!data.Contains(group))

{

Distance minDist = group.GetDistance(data[0]);

foreach (Group currGroup in data)

if (group.GetDistance(currGroup).Value < minDist.Value)

minDist = group.GetDistance(currGroup);

group.DeleteDistances(data);

group.Distances.Add(new Distance(minDist.Value, newGroup));

newGroup.Distances.Add(new Distance(minDist.Value, group));

}

}

foreach (Group group in data)

if (group.X == 0)

{

group.X = offsetX;

offsetX++;

}

newGroup.SubGroups = data;

var subGroupsPoints = new List<HPoint>();

foreach (Group addedGroup in data)

{

subGroupsPoints.Add(new HPoint(addedGroup.X, addedGroup.Y));

groups.Remove(addedGroup);

}

double x = 0;

foreach (HPoint point in subGroupsPoints)

x += point.X;

newGroup.X = x / subGroupsPoints.Count;

newGroup.Y = isMax ? 1.0 / minDistance: minDistance;

groups.Add(newGroup);

}

private Color NewColor()

{

return Color.FromArgb(rand.Next(256), rand.Next(256), rand.Next(256));

}

}

}

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 lab6

{

public partial class Form1 : Form

{

private readonly Algorithm algorithm;

private readonly Random rand;

private double[,] data;

public Form1()

{

InitializeComponent();

algorithm = new Algorithm();

rand = new Random();

}

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

{

int count = int.Parse(textBox1.Text);

dataGridView.ColumnCount = count + 1;

dataGridView.RowCount = count + 1;

data = new double[count, count];

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

{

dataGridView[0, i].Value = dataGridView[i, 0].Value = $"X{i}";

}

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

{

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

{

if (i <= j)

{

data[i, j] = data[j,i] = i == j ? 0 : Math.Round(rand.NextDouble() \* 10 + 1, 1);

dataGridView[i + 1, j + 1].Value = dataGridView[j + 1, i + 1].Value = data[i, j];

}

}

}

}

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

{

int count = int.Parse(textBox1.Text);

var newData = new double[count, count];

if (radioButton2.Checked)

{

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

{

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

{

if (i <= j)

{

newData[i, j] = newData[j, i] =1.0/data[i,j];

}

}

algorithm.SetData(count, newData);

}

}

else

{

algorithm.SetData(count, data);

}

algorithm.Do(radioButton2.Checked);

algorithm.Draw(chart1);

}

private void chart1\_Click(object sender, EventArgs e)

{

}

}

}