**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ  
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук  
Департамент программной инженерии

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Старший преподаватель департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. В. Пантюхин «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2017 г. | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. В. Шилов «\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2017 г. |

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** | RU.17701729.503200-01 12 01-1-ЛУ |

**ПРОГРАММА КЛАСТЕРИЗАЦИИ АЛГОРИТМОМ FASTDBSCAN**

**Текст программы**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU.17701729.503200-01 12 01-1-ЛУ**

Исполнитель  
студент группы БПИ165  
\_\_\_\_\_\_\_\_\_\_ / Соколов Д.В. /  
«\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2017 г.

**Москва 2017**

УТВЕРЖДЕНRU.17701729.503200-01 12 01-1-ЛУ

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. № подл | RU.17701729.503200-01 12 01-1 |

**ПРОГРАММА КЛАСТЕРИЗАЦИИ АЛГОРИТМОМ FASTDBSCAN**

**Текст программы**

**RU.17701729.503200-01 12 01-1**

**Листов 51**

**Москва 2017**

**Содержание**

[**1. Текст программы 3**](#_Toc483254894)

[**1.1. Класс Form1.cs 3**](#_Toc483254895)

[**1.2. Класс Graphic.cs 13**](#_Toc483254896)

[**1.3. Класс Dbscanwithrtree.cs 19**](#_Toc483254897)

[**1.4. Класс Rstartree.cs 21**](#_Toc483254898)

[**1.5. Класс TRNode.cs 22**](#_Toc483254899)

[**1.6. Класс TRtree.cs 28**](#_Toc483254900)

[**1.7. Класс Dbscan.cs 44**](#_Toc483254901)

[**1.8. Класс Kmeans.cs 45**](#_Toc483254902)

[**1.9. Класс Point.cs 50**](#_Toc483254903)

[**1.10. Класс WorkWithFilePoint.cs 51**](#_Toc483254904)

# **1. Текст программы**

Программа написана на языке C# (использована среда разработки Microsoft Visual Studio 2015). Программа состоит из 6 классов (учтены только вручную написанные классы, полностью автоматически сгенерированные классы не учтены).

В данном документе содержится только вручную написанный исходный код программы, код из полностью автоматически сгенерированных исходных файлов в данном документе не представлен.

## **1.1. Класс Form1.cs**

**using** **System**;

**using** **System.Windows.Forms**;

**using** **MyLib**;

**using** **System.Collections.Generic**;

**using** **System.Diagnostics**;

**using** **Lib4rtree**;

**namespace** **Start\_page**

{

**public** **partial** **class** **Form1** : Form

{

**bool** OnlyDBscan = **false**;//использовать ли только DBscan

**bool** UseRTree = **false**;//Использовать ли дерево в алгоритмах

/// <summary>

/// Конструктор создания стартового окна

/// </summary>

**public** **Form1**()

{

InitializeComponent();

comboBox1.Items.Add("FastDBScan");

comboBox1.Items.Add("Только DBscan");

comboBox1.SelectedItem = comboBox1.Items[**0**];

//всплывающие надписи над лэйблами

ToolTip t = **new** ToolTip();

t.SetToolTip(label1, "Нажмите кнопку справа и выберите файл в проводнике(.txt)");

t.SetToolTip(label2, "Количество кластеров для K-means(целое положительное >1)");

t.SetToolTip(label3, "Минимальное расстояние для DBScan(вещественное положительное)");

t.SetToolTip(label4, "Минимальное количество точек в кластере для DBScan(целое положительное)");

t.SetToolTip(label5, "Количество точек каждого кластера после K-means которые берутся для dbscan\*в процентах(0-100)");

t.SetToolTip(label6, "Максимальное количество точек в узле R\*-дерева(>4)");

textBox6.ReadOnly = **true**;

progressBar1.Maximum = **100**;

progressBar1.Minimum = **0**;

progressBar1.Value = **0**;

}

/// <summary>

/// По нажатию на кнопку "Открыть файл"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

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

{

OpenFileDialog a = **new** OpenFileDialog();

a.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*"; //только txt файлы

a.ShowDialog();

textBox1.Text = a.FileName;

}

/// <summary>

/// По нажатию на кнопку "Запустить кластеризацию"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

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

{

**bool** check = **true**;//успешно ли заполнены все поля

**int** k, minPTS;

**double** eps;

**int** t;

**int** maxm;

//проверка поля с адресом файла

**if** (textBox1.Text.Length < **1** && check) { MessageBox.Show("Не выбран файл!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); check = **false**; }

//проверка поля с К

**if** (textBox2.Text == "" && check && OnlyDBscan == **false**) { textBox2.Focus(); check = **false**; }

**if** ((!**int**.TryParse(textBox2.Text, **out** k) || k <= **1**) && check && OnlyDBscan==**false**) { MessageBox.Show("Введено неверное значение k(k >1)", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); textBox2.Clear(); textBox2.Focus(); check = **false**; }

//проверка поля с eps

**if** (textBox3.Text == "" && check) { textBox3.Focus(); check = **false**; }

**if** ((!**double**.TryParse(textBox3.Text.Replace('.', ','), **out** eps) || eps < **0.0**) && check) { MessageBox.Show("Введено неверное значение eps(eps>0)", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); textBox3.Clear(); textBox3.Focus(); check = **false**; }

//проверка поля с minPTS

**if** (textBox4.Text == "" && check) { textBox4.Focus(); check = **false**; }

**if** ((!**int**.TryParse(textBox4.Text, **out** minPTS) || minPTS <= **0**) && check) { MessageBox.Show("Введено неверное значение minPTS(minPTS>0)", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); textBox4.Clear(); textBox4.Focus(); check = **false**; }

//проверка поля с процентом

**if** (textBox5.Text =="" && check && OnlyDBscan==**false**) { textBox5.Focus(); check = **false**; }

**if** ((!**int**.TryParse(textBox5.Text,**out** t) || t<**0** || t>**100**) && check && OnlyDBscan==**false**) { MessageBox.Show("Введено неверное значение t(0 <= t <= 100", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); textBox5.Clear(); textBox5.Focus(); check = **false**; }

**if** (textBox6.Text == "" && check && UseRTree == **true**) { textBox6.Focus(); check = **false**; }

**if** ((!**int**.TryParse(textBox6.Text, **out** maxm) || maxm <= **4**) && check && UseRTree == **true**) { MessageBox.Show("Введено неверное значение MAXM (MAXM > 4)", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); textBox6.Clear(); textBox6.Focus(); check = **false**; }

//если все поля заданы верно, то запускаем форму с графиком

**if** (check) { ForRstartree.MAX\_M = maxm; ForRstartree.MIN\_M = (**int**)(maxm \* **0.4**); DrawGraph(textBox1.Text, k, eps, minPTS, t); }

}

/// <summary>

/// Закрытие формы по нажатию на кнопку

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

**private** **void** **closeToolStripMenuItem\_Click**(**object** sender, EventArgs e)

{

**this**.Close();

}

/// <summary>

/// Метод для выбора варианта кластеризации

/// </summary>

/// <param name="path"></param>

/// <param name="k"></param>

/// <param name="eps"></param>

/// <param name="minPTS"></param>

/// <param name="t"></param>

**void** **DrawGraph**(**string** path, **int** k, **double** eps, **int** minPTS, **int** t)

{

**try**

{

**if** (UseRTree == **true**)

{

**if** (OnlyDBscan == **true**)

{

UseOnlyDBScanwithRtree(path, eps, minPTS);

}

**else**

{

UseFastDBscanwithRtree(path, k, eps, minPTS, t);

}

}

**else**

{

**if** (OnlyDBscan == **true**)

{

UseOnlyDBScan(path, eps, minPTS);

}

**else**

{

UseOnlyFastDBScan(path, k, eps, minPTS, t);

}

}

}

**catch**

{

MessageBox.Show("Произошла непредвиденная ошибка. Попробуйте еще раз!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Метод кластеризации Dbscan с использованием R\*-дерева

/// </summary>

/// <param name="path">Путь до файла с точками</param>

/// <param name="eps"></param>

/// <param name="minPTS"></param>

**void** **UseOnlyDBScanwithRtree**(**string** path, **double** eps, **int** minPTS)

{

**bool** check = **true**;

List<Point> points = **new** List<Point>(**0**);

**try**

{

points = WorkWithFile.GetPointFromFile(path);

progressBar1.Value = **30**;

}

**catch** (ArgumentException ex)

{

check = **false**;

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

progressBar1.Value = **0**;

}

**if** (check)

{

Stopwatch dbscantime = **new** Stopwatch();

**int** count\_of\_db;

//if (ForRstartree.MAX\_M )

dbscantime.Start();

Dbscanwithrtree.Clustering(points, eps, minPTS, **out** count\_of\_db);

dbscantime.Stop();

progressBar1.Value = **70**;

MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения DBScan с R\*-деревом: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Скорость выполнения: " + $"{ ((dbscantime.Elapsed.Seconds > 0) ? (points.Count / dbscantime.Elapsed.Seconds) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information);

Graph.Graphic f2 = **new** Graph.Graphic(points, count\_of\_db,dbscantime,"Алгоритм кластеризации DBscan с использованием R\*-дерева");

progressBar1.Value = **100**;

f2.Show();

}

progressBar1.Value = **100**;

}

/// <summary>

/// Кластеризация FastDBSCan с R\*-деревом

/// </summary>

/// <param name="path"></param>

/// <param name="k"></param>

/// <param name="eps"></param>

/// <param name="minPTS"></param>

/// <param name="t"></param>

**void** **UseFastDBscanwithRtree**(**string** path, **int** k, **double** eps, **int** minPTS,**int** t)

{

**bool** check = **true**;

List<Point> points = **new** List<Point>(**0**);

**try**

{

points = WorkWithFile.GetPointFromFile(path);

progressBar1.Value = **30**;

}

**catch** (ArgumentException ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

check = **false**;

progressBar1.Value = **0**;

}

**if** (k>points.Count) { MessageBox.Show("Введенный параметр K больше чем количество точек в выбранном файле. Уменьшите К или выберите другой файл!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); check = **false**; progressBar1.Value = **0**; }

**if** (check)

{

Stopwatch kmeantime = **new** Stopwatch();

Stopwatch dbscantime = **new** Stopwatch();

kmeantime.Start();

**int**[] clustering = kmeans.Cluster(points, k-**1**, **0**);

progressBar1.Value = **50**;

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

points[i].cluster = clustering[i];

progressBar1.Value = **30**;

kmeantime.Stop();

List<Point> fordbscan = **new** List<Point>(**0**);

**int** ct = **0**;

**for** (**int** i = **0**; i < k; i++)

{

**int** count = **0**;

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

**if** (points[j].cluster == i) count++;

**int** finalcount = (**int**)Math.Ceiling((**double**)count \* t / **100**);

**int** l = **0**;

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

**if** (points[m].cluster == i && l < finalcount)

{

fordbscan.Add(points[m]);

ct++;

l++;

}

}

progressBar1.Value = **70**;

**int** count\_db;

dbscantime.Start();

**try**

{

Lib4rtree.Dbscanwithrtree.Clustering(fordbscan, eps, minPTS, **out** count\_db);

}

**catch**

{

count\_db = **0**;

MessageBox.Show("Превышено максимально допустимое количество итераций для Dbscan. Попробуйте другой значение MAXM.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

GetFullPicture(fordbscan, k, count\_db);

progressBar1.Value = **90**;

dbscantime.Stop();

MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения K-means: " + kmeantime.Elapsed.Seconds + " сек." + "\n" + "Время выполнения DBScan с R\*-деревом: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Общее время выполнения: " + (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds) + " сек." + "\n" + "Скорость выполнения: " + $"{((kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds > 0) ? (points.Count / (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds)) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information);

Graph.Graphic f1 = **new** Graph.Graphic(points, k, kmeantime,dbscantime,"Алгоритм кластеризации FastDBscan с использованием R-дерева");

progressBar1.Value = **100**;

f1.Show();

}

}

/// <summary>

/// Кластеризаця DBSCan

/// </summary>

/// <param name="path"></param>

/// <param name="eps"></param>

/// <param name="minPTS"></param>

**void** **UseOnlyDBScan**(**string** path, **double** eps,**int** minPTS)

{

**bool** check = **true**;

List<Point> points = **new** List<Point>(**0**);

**try**

{

points = MyLib.WorkWithFile.GetPointFromFile(path);

progressBar1.Value = **30**;

}

**catch** (ArgumentException ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

check = **false**;

progressBar1.Value = **0**;

}

**if** (check)

{

Stopwatch dbscantime = **new** Stopwatch();

dbscantime.Start();

**int** count\_of\_db = dbscan.Clustering(points, eps, minPTS);

progressBar1.Value = **80**;

dbscantime.Stop();

MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения DBScan: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Время выполнения DBScan: " + dbscantime.Elapsed.Seconds + "\n" + "Скорость выполнения: " + $"{ ((dbscantime.Elapsed.Seconds > 0) ? (points.Count / dbscantime.Elapsed.Seconds) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information);

Graph.Graphic f2 = **new** Graph.Graphic(points, count\_of\_db, dbscantime, "Алгоритм кластеризации DBscan без использования R\*-дерева");

f2.Show();

progressBar1.Value = **100**;

}

}

/// <summary>

/// Кластеризация FastDBScan

/// </summary>

/// <param name="path"></param>

/// <param name="k"></param>

/// <param name="eps"></param>

/// <param name="minPTS"></param>

/// <param name="t"></param>

**void** **UseOnlyFastDBScan**(**string** path, **int** k, **double** eps, **int** minPTS, **int** t)

{

**bool** check = **true**;

Stopwatch dbscantime = **new** Stopwatch();

Stopwatch kmeantime = **new** Stopwatch();

List<Point> points = **new** List<Point>(**0**);

**try**

{

points = WorkWithFile.GetPointFromFile(path);

progressBar1.Value = **30**;

}

**catch** (ArgumentException ex)

{

MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

check = **false**;

progressBar1.Value = **0**;

}

**if** (k > points.Count) { MessageBox.Show("Введенный параметр K больше чем количество точек в выбранном файле. Уменьшите К или выберите другой файл!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error); check = **false**; progressBar1.Value = **0**; }

**if** (check)

{

kmeantime.Start();

**int**[] clustering = kmeans.Cluster(points, k-**1**, **0**);

progressBar1.Value = **50**;

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

points[i].cluster = clustering[i];

progressBar1.Value = **30**;

kmeantime.Stop();

List<MyLib.Point> fordbscan = **new** List<MyLib.Point>(**0**);

**for** (**int** i = **0**; i < k; i++)

{

**int** count = **0**;

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

**if** (points[j].cluster == i) count++;

**int** finalcount = (**int**)Math.Ceiling((**double**)count \* t / **100**);

**int** l = **0**;

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

**if** (points[m].cluster == i && l < finalcount) { fordbscan.Add(points[m]); l++; }

}

progressBar1.Value = **70**;

dbscantime.Start();

**int** count\_db = dbscan.Clustering(fordbscan, eps, minPTS);

GetFullPicture(fordbscan, k, count\_db);

progressBar1.Value = **80**;

dbscantime.Stop();

MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения K-means: " + kmeantime.Elapsed.Seconds + " сек." + "\n" + "Время выполнения DBScan: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Общее время выполнения: " + (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds) + " сек." + "\n" + "Скорость выполнения: " + $"{((kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds > 0) ? (points.Count / (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds)) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information);

Graph.Graphic f1 = **new** Graph.Graphic(points, k,kmeantime,dbscantime, "Алгоритм кластеризации FastDbscan без ипользования R\*-дерева");

progressBar1.Value = **100**;

f1.Show();

}

}

/// <summary>

/// Обрабатываем каждый кластер K-means

/// </summary>

/// <param name="points">лист точек</param>

/// <param name="k">количество кластеров k-means</param>

/// <param name="count\_db">количество кластеров в dbscan</param>

**void** **GetFullPicture**(List<Point> points, **int** k, **int** count\_db)

{

List<Point> cluster;

**for** (**int** i = **0**; i < k; i++)

{

cluster = **new** List<Point>(**0**);

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

**if** (points[j].cluster == i) cluster.Add(points[j]);

WorkWithDb(cluster, i, count\_db);

}

progressBar1.Value = **50**;

}

/// <summary>

/// В листе точек, принадлежащих одному кластеру K-means, ищем самый большой кластер dbscan - его принимаем за истинный, остальные как шум

/// </summary>

/// <param name="points">Лист точек, принадлежащих одному кластеру в K-means</param>

/// <param name="clusterkmean">Количество кластеров K-means</param>

/// <param name="count\_db">Количество кластеров в dbscan</param>

**void** **WorkWithDb**(List<Point> points, **int** clusterkmean, **int** count\_db)

{

**int** count = **0**;

List<**int**> maxdb = **new** List<**int**>(**0**);

**for**(**int** i=**0**;i<count\_db;i++)

{

**int** counter = **0**;

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

**if** (points[j].db == i) counter++;

**if** (counter > count) { count = counter; maxdb.Clear(); maxdb.Add(i); }

**else** **if** (counter == count) maxdb.Add(i);

}

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

**if** (!maxdb.Contains(points[i].db)) points[i].cluster = -**1**;

progressBar1.Value = **70**;

}

/// <summary>

/// Открываем файл readme.txt блокнотом

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

**private** **void** **readmetxtToolStripMenuItem\_Click**(**object** sender, EventArgs e)

{

**string** tmp = @"readme.txt";

Process.Start("C:\\Windows\\System32\\notepad.exe", tmp.Trim());

}

/// <summary>

/// Меняем булевскую переменную в зависимости от выбранного элемента из комбобокса

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

**private** **void** **comboBox1\_SelectionChangeCommitted**(**object** sender, EventArgs e)

{

**if** (comboBox1.SelectedItem == comboBox1.Items[**0**])

{

comboBox1.SelectedItem = comboBox1.Items[**0**];

OnlyDBscan = **false**;

textBox2.ReadOnly = **false**;

textBox5.ReadOnly = **false**;

}

**else**

{

comboBox1.SelectedItem = comboBox1.Items[**1**];

OnlyDBscan = **true**;

textBox2.ReadOnly = **true**;

textBox5.ReadOnly = **true**;

}

}

/// <summary>

/// Флажок для "Использовать R\*-tree"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

**private** **void** **checkBox1\_Click**(**object** sender, EventArgs e)

{

**if** (checkBox1.Checked == **true**)

{

checkBox1.CheckState = CheckState.Checked;

textBox6.ReadOnly = **false**;

UseRTree = **true**;

}

**else**

{

checkBox1.CheckState = CheckState.Unchecked;

textBox6.ReadOnly = **true**;

UseRTree = **false**;

}

}

}

}

## **1.2. Класс Graphic.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Drawing**;

**using** **System.Windows.Forms**;

**using** **ZedGraph**;

**using** **System.Diagnostics**;

**namespace** **Graph**

{

**public** **partial** **class** **Graphic** : Form

{

Stopwatch dbscantime = **null**;

Stopwatch kmeantime = **null**;

List<MyLib.Point> points;

Color[] array = { Color.Blue, Color.Red, Color.Orange, Color.Violet, Color.Pink, Color.Purple, Color.Green, Color.Brown, Color.SeaGreen, Color.SkyBlue, Color.Tomato, Color.SteelBlue, Color.Aqua, Color.Bisque, Color.BlanchedAlmond, Color.BlueViolet, Color.BurlyWood, Color.CadetBlue, Color.Chocolate, Color.Crimson, Color.Fuchsia, Color.Indigo, Color.Khaki, Color.Lavender, Color.LimeGreen, Color.Silver, Color.YellowGreen , Color.Azure, Color.CornflowerBlue, Color.DarkOrange, Color.DarkViolet, Color.ForestGreen, Color.Tan, Color.Teal};

**public** **Graphic**()

{

InitializeComponent();

}

/// <summary>

/// Конструктор для графика FastDBscan

/// </summary>

/// <param name="points">Лист точек</param>

/// <param name="numClusters">Количество кластеров</param>

**public** **Graphic**(List<MyLib.Point> points, **int** numClusters, Stopwatch kmeantime, Stopwatch dbscantime, **string** name)

{

InitializeComponent();

zedGraph.ContextMenuBuilder += **new** ZedGraphControl.ContextMenuBuilderEventHandler(zedGraph\_ContextMenuBuilder);

**this**.kmeantime = kmeantime;

**this**.dbscantime = dbscantime;

**this**.points = points;

DrawGraph(points, numClusters, name);

}

/// <summary>

/// Конструктор для графика DBscan

/// </summary>

/// <param name="points">Лист точек</param>

/// <param name="numClusters">Количество кластеров</param>

**public** **Graphic**(List<MyLib.Point> points, **int** numClusters, Stopwatch dbscantime, **string** name)

{

InitializeComponent();

zedGraph.ContextMenuBuilder += **new** ZedGraphControl.ContextMenuBuilderEventHandler(zedGraph\_ContextMenuBuilder);

**this**.dbscantime = dbscantime;

**this**.points = points;

DrawGraph(points, numClusters, name, " ");

}

/// <summary>

/// Метод для отрисовки графика FastDBScan'a

/// </summary>

/// <param name="points">Лист точек</param>

/// <param name="numClusters">Количество кластеров</param>

**void** **DrawGraph**(List<MyLib.Point> points, **int** numClusters, **string** name)

{

GraphPane pane = zedGraph.GraphPane;

pane.Title.Text = name;

pane.CurveList.Clear();

PointPairList list = **new** PointPairList();

**for** (**int** j = -**1**; j < numClusters; j++)

{

**int** count = **0**;

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

**if** (points[i].cluster == j) { list.Add(points[i].x, points[i].y); list[count].Tag = j.ToString(); count++; }

LineItem myCurve;

**if** (j==-**1**) myCurve = pane.AddCurve("", list, Color.Black, SymbolType.Circle); **else** myCurve = pane.AddCurve("", list, array[j % array.Length], SymbolType.Circle);

myCurve.Line.IsVisible = **false**;

myCurve.Symbol.Fill.Type = FillType.Solid;

myCurve.Symbol.Size = **3**;

list = **new** PointPairList();

}

pane.XAxis.Cross = **0.0**;

pane.YAxis.Cross = **0.0**;

pane.XAxis.Scale.Min = MyLib.WorkWithFile.xmin;

pane.XAxis.Scale.Max = MyLib.WorkWithFile.xmax;

pane.YAxis.Scale.Min = MyLib.WorkWithFile.ymin;

pane.YAxis.Scale.Max = MyLib.WorkWithFile.ymax;

zedGraph.AxisChange();

zedGraph.Invalidate();

pane.XAxis.Title.IsVisible = **false**;

pane.YAxis.Title.IsVisible = **false**;

}

/// <summary>

/// Метод для отрисовки графика DBScan'a

/// </summary>

/// <param name="points">Лист точек</param>

/// <param name="numClusters">Количество кластеров</param>

/// <param name="name"></param>

**void** **DrawGraph**(List<MyLib.Point> points, **int** numClusters, **string** name, **string** test)

{

GraphPane pane = zedGraph.GraphPane;

pane.Title.Text = name;

pane.CurveList.Clear();

PointPairList list = **new** PointPairList();

**for** (**int** j = -**1**; j < numClusters; j++)

{

**int** count = **0**;

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

**if** (points[i].db == j) { list.Add(points[i].x, points[i].y); list[count].Tag = j.ToString(); count++; }

LineItem myCurve;

**if** (j == -**1**) myCurve = pane.AddCurve("", list, Color.Black, SymbolType.Circle); **else** myCurve = pane.AddCurve("", list, array[j%array.Length], SymbolType.Circle);

myCurve.Line.IsVisible = **false**;

myCurve.Symbol.Fill.Type = FillType.Solid;

myCurve.Symbol.Size = **3**;

list = **new** PointPairList();

}

pane.XAxis.Cross = **0.0**;

pane.YAxis.Cross = **0.0**;

pane.XAxis.Scale.Min = MyLib.WorkWithFile.xmin;

pane.XAxis.Scale.Max = MyLib.WorkWithFile.xmax;

pane.YAxis.Scale.Min = MyLib.WorkWithFile.ymin;

pane.YAxis.Scale.Max = MyLib.WorkWithFile.ymax;

zedGraph.AxisChange();

zedGraph.Invalidate();

pane.XAxis.Title.IsVisible = **false**;

pane.YAxis.Title.IsVisible = **false**;

}

/// <summary>

/// Меняет надписи по правой кнопке мыши на русский язык и убирает лишнее

/// </summary>

/// <param name="sender"></param>

/// <param name="menuStrip"></param>

/// <param name="mousePt"></param>

/// <param name="objState"></param>

**void** **zedGraph\_ContextMenuBuilder**(ZedGraphControl sender, ContextMenuStrip menuStrip, System.Drawing.Point mousePt, ZedGraphControl.ContextMenuObjectState objState)

{

menuStrip.Items[**0**].Text = "Копировать";

menuStrip.Items[**1**].Text = "Сохранить как картинку…";

menuStrip.Items[**2**].Text = "Параметры страницы…";

menuStrip.Items[**3**].Text = "Печать…";

menuStrip.Items[**4**].Text = "Показывать значения в точках…";

menuStrip.Items.RemoveAt(**4**);

menuStrip.Items.RemoveAt(**4**);

menuStrip.Items.RemoveAt(**4**);

menuStrip.Items.RemoveAt(**4**);

ToolStripItem clusteritem = **new** ToolStripMenuItem("Показывать номер кластера…");

menuStrip.Items.Add(clusteritem);

clusteritem.Click += **new** EventHandler(newMenuItem\_Click);

ToolStripItem pointitem = **new** ToolStripMenuItem("Показывать координаты…");

menuStrip.Items.Add(pointitem);

pointitem.Click += **new** EventHandler(pointitem\_Click);

}

**void** **pointitem\_Click**(**object** sender, EventArgs e)

{

zedGraph.IsShowPointValues = **true**;

zedGraph.PointValueEvent += **new** ZedGraphControl.PointValueHandler(zedGraph\_Point\_show);

}

**string** **zedGraph\_Point\_show**(ZedGraphControl sender,

GraphPane pane,

CurveItem curve,

**int** iPt)

{

// Получим точку, около которой находимся

PointPair point = curve[iPt];

// Сформируем строку

**string** result = $"X: {point.X:F3} Y: {point.Y}";

**return** result;

}

**void** **newMenuItem\_Click**(**object** sender, EventArgs e)

{

zedGraph.IsShowPointValues = **true**;

zedGraph.PointValueEvent += **new** ZedGraphControl.PointValueHandler(zedGraph\_Cluster\_show);

}

**string** **zedGraph\_Cluster\_show**(ZedGraphControl sender,

GraphPane pane,

CurveItem curve,

**int** iPt)

{

// Получим точку, около которой находимся

PointPair point = curve[iPt];

// Сформируем строку

**string** result = point.Tag.ToString();

**return** result;

}

**private** **void** **toolStripLabel1\_Click**(**object** sender, EventArgs e)

{

**if** (kmeantime != **null**)

**if** (points != **null**) MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения K-means: " + kmeantime.Elapsed.Seconds + " сек." + "\n" + "Время выполнения DBScan: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Общее время выполнения: " + (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds) + " сек." + "\n" + "Скорость выполнения: " + $"{((kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds > 0) ? (points.Count / (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds)) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information); **else** MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения K-means: " + kmeantime.Elapsed.Seconds + " сек." + "\n" + "Время выполнения DBScan с R\*-деревом: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Общее время выполнения: " + (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds) + " сек." + "\n" + "Скорость выполнения: " + $"{((kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds > 0) ? (points.Count / (kmeantime.Elapsed.Seconds + dbscantime.Elapsed.Seconds)) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information);

**else**

**if** (points != **null**) MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения DBScan: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Общее время выполнения: " + (dbscantime.Elapsed.Seconds) + " сек." + "\n" + "Скорость выполнения: " + $"{((dbscantime.Elapsed.Seconds > 0) ? (points.Count / (dbscantime.Elapsed.Seconds)) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information); **else** MessageBox.Show("Количество точек: " + points.Count + "\n" + "Время выполнения DBScan с R\*-деревом: " + dbscantime.Elapsed.Seconds + " сек." + "\n" + "Общее время выполнения: " + (dbscantime.Elapsed.Seconds) + " сек." + "\n" + "Скорость выполнения: " + $"{((dbscantime.Elapsed.Seconds > 0) ? (points.Count / dbscantime.Elapsed.Seconds) : points.Count):F2} точки в сек.", "Время выполнения", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

## **1.3. Класс Dbscanwithrtree.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**using** **MyLib**;

**namespace** **Lib4rtree**

{

**public** **static** **class** **Dbscanwithrtree**

{

**public** **static** **double** eps;//расстояние для dbscan'a

**static** **int** minPTS;//минимальное количество точек в заданном радиусе для dbscan'a

**public** **static** **void** **Clustering**(List<MyLib.Point> list, **double** e, **int** mpts, **out** **int** cluster\_number)

{

//сохраним переменные в статические чтобы не передавать их лишний раз в методы

eps = e;

minPTS = mpts;

TRtree tree = **new** TRtree();

Console.WriteLine("count " + list.Count);

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

{ list[i].idx = i; tree.insertObject(list[i]); }

cluster\_number = **0**;//начинаем счет кластеров с нуля

CheckAllPoints(tree);

**for** (**int** i = **0**; i < tree.FNodeArr.Length; i++)//пройдем по всем узлам дерева

{

**for** (**int** j = **0**; j < tree.FNodeArr[i].FObject.Length; j++)//проходим по всем точкам в узле дерева

{

**if** (tree.FNodeArr[i].FObject[j].IsVisited == **false** && tree.FNodeArr[i].FObject[j].minPTScheck == **true**)//если точка не была посещена и выполнено условие на количество точек в eps-окрестности

{

tree.FNodeArr[i].FObject[j].IsVisited = **true**;//помечаем как посещенная

tree.FNodeArr[i].FObject[j].db = cluster\_number;//помечаем текущую точку как текущий кластер

Startclust(tree, tree.FNodeArr[i].FObject[j].findpoint, cluster\_number);//запускаем рекурсивную функцию для заполнения кластера

cluster\_number++;//прибавляем номер кластера

}

**else**

{

tree.FNodeArr[i].FObject[j].IsVisited = **true**;//иначе помечаем как посещенную(точка не прошла проверку на плотность в eps-окрестности)

}

}

}

**for** (**int** i = **0**; i < tree.FNodeArr.Length; i++)

**for** (**int** j = **0**; j < tree.FNodeArr[i].FObject.Length; j++)

list[tree.FNodeArr[i].FObject[j].idx].db = tree.FNodeArr[i].FObject[j].db;

}

**static** **void** **CheckAllPoints**(TRtree tree)

{

**for** (**int** i = **0**; i < tree.FNodeArr.Length; i++)

**for** (**int** j = **0**; j < tree.FNodeArr[i].FObject.Length; j++)

**if** (Check(tree, tree.FNodeArr[i].FObject[j])) tree.FNodeArr[i].FObject[j].minPTScheck = **true**; **else** tree.FNodeArr[i].FObject[j].minPTScheck = **false**;

}

**static** **bool** **Check**(TRtree tree, MyLib.Point point)

{

ForRstartree.Point lefttop = **new** ForRstartree.Point(point.x - eps, point.y + eps);

ForRstartree.Point rightdown = **new** ForRstartree.Point(point.x + eps, point.y - eps);

ForRstartree.TMBR findzone = **new** ForRstartree.TMBR(lefttop, rightdown);//Получили квадрат описывающий круг с центром в текущей точке

List<MyLib.Point> list = **new** List<MyLib.Point>(**0**);

List<MyLib.Point> find = **new** List<MyLib.Point>(**0**);

find = tree.findobjectinarea(findzone, tree.FRoot, list, point.idx);//получили лист точек, которые находятся в eps-окрестности текущей(ее не берем)

**for** (**int** i = **0**; i < tree.FNodeArr.Length; i++)

tree.FNodeArr[i].IsVisited = **false**;

point.findpoint = find;

**if** (find.Count >= minPTS) **return** **true**; **else** **return** **false**;

}

/// <summary>

/// Рекурсивная функция для расширеия кластера

/// </summary>

/// <param name="tree">Дерево</param>

/// <param name="findpoint">Лист найденных точек</param>

/// <param name="cluster">Номер кластера</param>

**static** **void** **Startclust**(TRtree tree, List<MyLib.Point> findpoint, **int** cluster)

{

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

{

**if** (findpoint[i].IsVisited == **false** && findpoint[i].minPTScheck==**true**) { findpoint[i].db = cluster; findpoint[i].IsVisited = **true**; Startclust(tree, findpoint[i].findpoint, cluster); }

}

}

}

}

## **1.4. Класс Rstartree.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **Lib4rtree**

{

/// <summary>

/// Класс с основными компонентами в виде структур

/// </summary>

**public** **class** **ForRstartree**

{

**public** **static** **int** MAX\_M = **100**; // Максимальное количество объектов в узле

**public** **static** **int** MIN\_M = (**int**)(MAX\_M \* **0.4**); // Минимальное количество объектов в узле

**public** **enum** TAxis { X, Y };// для chooseSplitAxis - по какой оси будет разделение

**protected** **enum** TBound { Left, Right };// граница по какой будет идти сортировка (левая\правая)

**public** **struct** **Point**// структура описывающая точку

{

**public** **Point**(**double** x, **double** y)

{

X = x;

Y = y;

}

**public** **double** X, Y;

}

**public** **struct** **TMBR**// структура описывающая область покрытия узла MBR = Minimum Bounding Rectangle

{

**public** **TMBR**(Point l, Point r)

{

Left = l;

Right = r;

}

**public** Point Left, Right;// left - координаты верхнего левого угла right - координаты нижнего правого угла

}

}

}

## **1.5. Класс TRNode.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **Lib4rtree**

{

**public** **class** **TRNode** : ForRstartree

{

**public** TMBR fmbr;// ограничивающий прямоугольник узла

**public** **int** FParent;// индекс в массиве узлов дерева, указывающий на узел-родитель

**public** **int**[] FChildren = **new** **int**[**0**]; // список индексов дочерних детей в массиве узлов дерева

**public** MyLib.Point[] FObject = **new** MyLib.Point[**0**];// массив с обьектами

**public** **bool** FisLeaf;// свойство показывающее является ли этот узел конечным(листом)

**public** **int** FLevel;// уровень узла в дереве (0=лист)

**public** **bool** IsVisited = **false**;

/// <summary>

/// Есть ли дети у узла

/// </summary>

/// <returns></returns>

**public** **bool** **getIsLeaf**()

{

**if** (FObject.Length > **0**) **return** **true**; **else** **return** **false**;

}

/// <summary>

/// метод доступа дочерним узлам

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

**public** **int** **getChild**(**int** index)

{

**if** (FChildren.Length > index) **return** FChildren[index]; **else** **throw** **new** ArgumentOutOfRangeException("Попытка получения несуществующего объекта");

}

/// <summary>

/// метод доступа к обьектам в узле

/// </summary>

/// <param name="index"></param>

/// <returns></returns>

**public** MyLib.Point getObject(**int** index)

{

**if** (index < FObject.Length && index >= **0**) **return** FObject[index]; **else** **throw** **new** ArgumentOutOfRangeException("Попытка получения несуществующего объекта");

}

/// <summary>

/// метод присваивания дочернего узла

/// </summary>

/// <param name="index"></param>

/// <param name="node\_id"></param>

**public** **void** **setChild**(**int** index, **int** node\_id)

{

**if** (FChildren.Length > index && index >= **0**)

{

// Console.WriteLine("index "+index +" node\_id " + node\_id);

FChildren[index] = node\_id;

FisLeaf = **false**;

}

**else**

{

**if** (index <= (MAX\_M - **1**) && index >= **0**)

{

// Console.WriteLine("index " + index + " node\_id " + node\_id);

Array.Resize(**ref** FChildren, index + **1**);

FChildren[index] = node\_id;

FisLeaf = **false**;

}

}

}

/// <summary>

/// метод присваивания обьекта узлу

/// </summary>

/// <param name="index"></param>

/// <param name="obj"></param>

**public** **void** **setObject**(**int** index, MyLib.Point obj)

{

**if** (FObject.Length > index && index >= **0**)

{

FObject[index] = obj;

FisLeaf = **true**;

}

**else**

{

**if** (index <= MAX\_M - **1** && index >= **0**)

{

Array.Resize(**ref** FObject, index + **1**);

FObject[index] = obj;

FisLeaf = **true**;

}

}

}

/// <summary>

/// метод присваивания узла-родителя

/// </summary>

/// <param name="parent\_id"></param>

**protected** **void** **setParent**(**int** parent\_id)

{

**if** (parent\_id >= **0**) FParent = parent\_id;

}

/// <summary>

/// Копирование

/// </summary>

/// <param name="node"></param>

**public** **void** **copy**(TRNode node)// метод копирования узла

{

FObject = **new** MyLib.Point[node.FObject.Length];

**if** (FObject.Length > **0**)

{

**for** (**int** i = **0**; i < node.FObject.Length; i++)

{

FObject[i] = **new** MyLib.Point(node.FObject[i].x, node.FObject[i].y, node.FObject[i].idx);

FObject[i].cluster = node.FObject[i].cluster;

}

FisLeaf = **true**;

}

**else**

{

FChildren = **new** **int**[node.FChildren.Length];

**for** (**int** i = **0**; i < node.FChildren.Length; i++)

FChildren[i] = node.FChildren[i];

FisLeaf = **false**;

}

fmbr = **new** TMBR(node.fmbr.Left, node.fmbr.Right);

FParent = node.FParent;

FLevel = node.FLevel;

}

/// <summary>

/// Очишает массив объектов

/// </summary>

**public** **void** **clearObjects**()

{

FisLeaf = **false**;

FObject = **new** MyLib.Point[**0**];

}

**public** **void** **clearChildren**()

{

FChildren = **new** **int**[**0**];

}

/// <summary>

/// Конструктор умолчания

/// </summary>

**public** **TRNode**()

{

FParent = -**10**;

}

/// <summary>

/// Конструктор копирования

/// </summary>

/// <param name="node"></param>

**public** **TRNode**(TRNode node)

{

FParent = -**10**;

copy(node);

}

/// <summary>

/// свойство предоставляющее доступ к полям через соответствующие методы

/// </summary>

**public** TMBR mbr

{

**get**

{

**return** fmbr;

}

**set**

{

fmbr = **value**;

}

}

/// <summary>

/// Является ли узел листом

/// </summary>

**public** **bool** isLeaf

{

**get**

{

**return** FisLeaf;

}

}

**public** **int** Parent

{

**get**

{

**return** FParent;

}

**set**

{

setParent(**value**);

}

}

**public** **int** Level

{

**get**

{

**return** FLevel;

}

**set**

{

FLevel = **value**;

}

}

/// <summary>

/// метод определяющий пересекаются ли две области mbr1-объект и mbr2 - область поиска

/// </summary>

/// <param name="mbr1">Объект</param>

/// <param name="mbr2">Область</param>

/// <returns></returns>

**public** **static** **bool** **isIntersected**(TMBR mbr1, TMBR mbr2)

{

**double** d1, d2, d3, d4;

d1 = Math.Min(mbr1.Left.X, mbr1.Right.X);

d2 = Math.Max(mbr1.Left.Y, mbr1.Right.Y);

d3 = Math.Max(mbr1.Left.X, mbr1.Right.X);

d4 = Math.Min(mbr1.Left.Y, mbr1.Right.Y);

mbr1.Left.X = d1;

mbr1.Left.Y = d2;

mbr1.Right.X = d3;

mbr1.Right.Y = d4;

d1 = Math.Min(mbr2.Left.X, mbr2.Right.X);

d2 = Math.Max(mbr2.Left.Y, mbr2.Right.Y);

d3 = Math.Max(mbr2.Left.X, mbr2.Right.X);

d4 = Math.Min(mbr2.Left.Y, mbr2.Right.Y);

mbr2.Left.X = d1;

mbr2.Left.Y = d2;

mbr2.Right.X = d3;

mbr2.Right.Y = d4;

**if** (mbr1.Left.X >= mbr2.Left.X && mbr1.Right.X <= mbr2.Right.X && mbr1.Left.Y <= mbr2.Left.Y && mbr1.Right.Y >= mbr2.Right.Y) **return** **true**; **else** **return** **false**;

}

/// <summary>

/// метод определяющий пересекается ли два прямоугольника

/// </summary>

/// <param name="mbr"></param>

/// <returns></returns>

**public** **bool** **IsIntersected**(TMBR mbr)

{

**double** d1, d2, d3, d4;

d1 = Math.Min(fmbr.Left.X, fmbr.Right.X);

d2 = Math.Max(fmbr.Left.Y, fmbr.Right.Y);

d3 = Math.Max(fmbr.Left.X, fmbr.Right.X);

d4 = Math.Min(fmbr.Left.Y, fmbr.Right.Y);

fmbr.Left.X = d1;

fmbr.Left.Y = d2;

fmbr.Right.X = d3;

fmbr.Right.Y = d4;

d1 = Math.Min(mbr.Left.X, mbr.Right.X);

d2 = Math.Max(mbr.Left.Y, mbr.Right.Y);

d3 = Math.Max(mbr.Left.X, mbr.Right.X);

d4 = Math.Min(mbr.Left.Y, mbr.Right.Y);

mbr.Left.X = d1;

mbr.Left.Y = d2;

mbr.Right.X = d3;

mbr.Right.Y = d4;

**return** !(mbr.Left.Y < fmbr.Right.Y || mbr.Right.Y > fmbr.Left.Y || mbr.Right.X < fmbr.Left.X || mbr.Left.X > fmbr.Right.X);

}

/// <summary>

/// Возвращает площадь перекрытия MBR узла и заданной области

/// </summary>

/// <param name="mbr\_ovrl"></param>

/// <returns></returns>

**public** **double** **Overlap**(TMBR mbr\_ovrl)

{

**double** x, y;

x = Math.Min(mbr\_ovrl.Right.X, fmbr.Right.X) - Math.Max(mbr\_ovrl.Left.X, fmbr.Left.X);

**if** (x <= **0**) **return** **0**;

y = Math.Min(mbr\_ovrl.Left.Y, fmbr.Left.Y) - Math.Max(mbr\_ovrl.Right.Y, fmbr.Right.Y);

**if** (y <= **0**) **return** **0**;

**return** x \* y;

}

/// <summary>

/// Возвращает площадь MBR узла

/// </summary>

/// <returns></returns>

**public** **double** **Area**()

{

**return** (fmbr.Right.X - fmbr.Left.X) \* (fmbr.Left.Y - fmbr.Right.Y);

}

/// <summary>

/// Возвращает площадь MBR узла

/// </summary>

/// <returns></returns>

**public** **double** **Area**(TMBR mbr)

{

**return** (mbr.Right.X - mbr.Left.X) \* (mbr.Left.Y - mbr.Right.Y);

}

/// <summary>

/// Возвращает периметр MBR

/// </summary>

/// <returns></returns>

**public** **double** **margin**()

{

**return** **2** \* (fmbr.Right.X - fmbr.Left.X + fmbr.Left.Y - fmbr.Right.Y);

}

}

}

## **1.6. Класс TRtree.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **MyLib**;

**namespace** **Lib4rtree**

{

**public** **class** **TRtree** : ForRstartree

{

**public** TRNode[] FNodeArr = **new** TRNode[**0**]; //массив узлов дерева

**public** **int** FRoot;// ссылка на положение корневого узла в массиве узлов

**private** **int** FHeight; // высота дерева

/// <summary>

/// Бытсрая сортировка массива точек по заданной оси

/// </summary>

/// <param name="a">Массив точек</param>

/// <param name="l">Левая граница</param>

/// <param name="r">Правая граница</param>

/// <param name="axe">Ось по которой происходит сортировка</param>

**static** **void** **QuickSort**(MyLib.Point[] a, **int** l, **int** r, TAxis axe)

{

MyLib.Point temp;

**double** mid = **0**;

**switch** (axe)

{

**case** TAxis.X: mid = a[l + (r - l) / **2**].x; **break**;

**case** TAxis.Y: mid = a[l + (r - l) / **2**].y; **break**;

}

//запись эквивалентна (l+r)/2,

//но не вызввает переполнения на больших данных

**int** i = l;

**int** j = r;

//код в while обычно выносят в процедуру particle

**while** (i <= j)

{

**switch** (axe)

{

**case** TAxis.X:

{

**while** (a[i].x < mid) i++;

**while** (a[j].x > mid) j--;

}

**break**;

**case** TAxis.Y:

{

**while** (a[i].y < mid) i++;

**while** (a[j].y > mid) j--;

}

**break**;

}

**if** (i <= j)

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

i++;

j--;

}

}

**if** (i < r)

QuickSort(a, i, r, axe);

**if** (l < j)

QuickSort(a, l, j, axe);

}

/// <summary>

/// быстрая сортировка для узлов по их MBR. axe - ось по которой происходит сортировка, bound - граница по которой происходит сортировка (левая/правая)

/// </summary>

/// <param name="List"></param>

/// <param name="iLo"></param>

/// <param name="iHi"></param>

/// <param name="axe"></param>

/// <param name="bound"></param>

#region

**private** **void** **QuickSort**(**int**[] List, **int** iLo, **int** iHi, TAxis axe, TBound bound)

{

//Console.WriteLine("QuickSort2 work");

**int** Lo = iLo, Hi = iHi;

**int** T;

**double** Mid = **0.0**;

**switch** (bound)

{

**case** TBound.Left:

{

**switch** (axe)

{

**case** TAxis.X: Mid = FNodeArr[List[(Lo + Hi) / **2**]].mbr.Left.X; **break**;

**case** TAxis.Y: Mid = FNodeArr[List[(Lo + Hi) / **2**]].mbr.Left.Y; **break**;

}

}

**break**;

**case** TBound.Right:

{

**switch** (axe)

{

**case** TAxis.X: Mid = FNodeArr[List[(Lo + Hi) / **2**]].mbr.Right.X; **break**;

**case** TAxis.Y: Mid = FNodeArr[List[(Lo + Hi) / **2**]].mbr.Right.Y; **break**;

}

}

**break**;

}

**do**

{

**switch** (bound)

{

**case** TBound.Left:

{

**switch** (axe)

{

**case** TAxis.X:

{

**while** (FNodeArr[List[Lo]].mbr.Left.X < Mid)

Lo++;

**while** (FNodeArr[List[Hi]].mbr.Left.X > Mid)

Hi--;

}

**break**;

**case** TAxis.Y:

{

**while** (FNodeArr[List[Lo]].mbr.Left.Y < Mid)

Lo++;

**while** (FNodeArr[List[Hi]].mbr.Left.Y > Mid)

Hi--;

}

**break**;

}

}

**break**;

**case** TBound.Right:

{

**switch** (axe)

{

**case** TAxis.X:

{

**while** (FNodeArr[List[Lo]].mbr.Right.X < Mid)

Lo++;

**while** (FNodeArr[List[Hi]].mbr.Right.X > Mid)

Hi--;

}

**break**;

**case** TAxis.Y:

{

**while** (FNodeArr[List[Lo]].mbr.Right.Y < Mid)

Lo++;

**while** (FNodeArr[List[Hi]].mbr.Right.Y > Mid)

Hi--;

}

**break**;

}

}

**break**;

}

**if** (Lo <= Hi)

{

T = List[Lo];

List[Lo] = List[Hi];

List[Hi] = T;

Lo++;

Hi--;

}

} **while** (Lo <= Hi);

**if** (Hi > iLo) QuickSort(List, iLo, Hi, axe, bound);

**if** (Lo < iHi) QuickSort(List, Lo, iHi, axe, bound);

}

#endregion

/// <summary>

/// разделяет узел на 2 в соответствии с алгоритмами R\*-tree (node\_id = ссылка на узел для разделения obj = обьект для вставки)

/// </summary>

#region

**private** **void** **splitNodeRStar**(**int** node\_id, MyLib.Point obj)

{

//Console.WriteLine("splitNodeRStar work1 " + FHeight);

TAxis axe;

**int** parent\_id, new\_child\_id;

TRNode node\_1, node\_2, node\_1\_min, node\_2\_min;

**int** i, j, k;

MyLib.Point[] arr\_obj;

**double** area\_min, Area;

**if** (!FNodeArr[node\_id].isLeaf) **return**;

**if** (isRoot(node\_id))

{

//Console.WriteLine("it works");

parent\_id = newNode();

FNodeArr[FRoot].Parent = parent\_id;

FNodeArr[parent\_id].setChild(**0**, FRoot);

FNodeArr[parent\_id].Level = FNodeArr[FNodeArr[parent\_id].getChild(**0**)].Level + **1**;

FRoot = parent\_id;

FHeight++;

}

**else**

{

parent\_id = newNode();

FNodeArr[parent\_id].Parent = FNodeArr[node\_id].Parent;

FNodeArr[FNodeArr[parent\_id].Parent].setChild(FNodeArr[FNodeArr[parent\_id].Parent].FChildren.Length, parent\_id);

FNodeArr[node\_id].Parent = parent\_id;

FNodeArr[parent\_id].setChild(**0**, node\_id);

/\*

parent\_id = FNodeArr[node\_id].Parent;

Console.WriteLine("fuck); " + parent\_id);

FNodeArr[FNodeArr[node\_id].Parent].FisLeaf = false;

\*/

}

arr\_obj = **new** MyLib.Point[MAX\_M + **1**];

**for** (i = **0**; i < arr\_obj.Length - **1**; i++)

arr\_obj[i] = FNodeArr[node\_id].getObject(i);

arr\_obj[arr\_obj.Length - **1**] = obj;

node\_1\_min = **new** TRNode();

node\_2\_min = **new** TRNode();

node\_1 = **new** TRNode();

node\_2 = **new** TRNode();

axe = chooseSplitAxis(obj, node\_id);

area\_min = **double**.MaxValue;

**for** (i = **0**; i <= **1**; i++)

{

QuickSort(arr\_obj, **0**, arr\_obj.Length - **1**, axe);

**for** (k = MIN\_M - **1**; k <= MAX\_M - MIN\_M; k++)

{

node\_1.clearObjects();

node\_2.clearObjects();

j = **0**;

**while** (j <= k)

{

node\_1.setObject(j, arr\_obj[j]);

j++;

}

**for** (j = k; j < arr\_obj.Length - **1**; j++)

node\_2.setObject(j - k, arr\_obj[j + **1**]);

updateMBR(node\_1);

updateMBR(node\_2);

Area = node\_1.Area() + node\_2.Area();

**if** (Area < area\_min)

{

node\_1\_min.copy(node\_1);

node\_2\_min.copy(node\_2);

area\_min = Area;

}

}

}

node\_1\_min.Level = **0**;

node\_2\_min.Level = **0**;

FNodeArr[node\_id].copy(node\_1\_min);

FNodeArr[node\_id].Parent = parent\_id;

updateMBR(node\_id);

new\_child\_id = newNode();

FNodeArr[new\_child\_id].copy(node\_2\_min);

FNodeArr[new\_child\_id].Parent = parent\_id;

updateMBR(new\_child\_id);

**if** (FNodeArr[parent\_id].FChildren.Length < MAX\_M)

{

FNodeArr[parent\_id].setChild(FNodeArr[parent\_id].FChildren.Length, new\_child\_id);

updateMBR(parent\_id);

FNodeArr[parent\_id].FisLeaf = **false**;

}

**else**

**splitNodeRStar**(parent\_id, new\_child\_id);

}

#endregion

/// <summary>

/// разделяет узел на 2 в соответствии с алгоритмами R\*-tree splited\_Node\_Id = ссылка на узел для разделения, inserted\_Node\_Id = узел для вставки

/// </summary>

/// <param name="splited\_Node\_Id"></param>

/// <param name="inserted\_Node\_Id"></param>

#region

**private** **void** **splitNodeRStar**(**int** splited\_Node\_Id, **int** inserted\_Node\_Id)

{

//Console.WriteLine("splitNodeRSter2 work");

TAxis axe;

**int** parent\_id, new\_child\_id;

TRNode node\_1, node\_2, node\_1\_min, node\_2\_min;

**int** i, j, k;

**int**[] arr\_node;

**double** area\_min, Area;

**if** (FNodeArr[splited\_Node\_Id].isLeaf) **return**;

**if** (isRoot(splited\_Node\_Id))

{

parent\_id = newNode();

FNodeArr[FRoot].Parent = parent\_id;

FNodeArr[parent\_id].setChild(**0**, FRoot);

FNodeArr[parent\_id].Level = FNodeArr[FNodeArr[parent\_id].getChild(**0**)].Level + **1**;

FRoot = parent\_id;

FNodeArr[parent\_id].FisLeaf = **false**;

FHeight = FHeight + **1**;

}

**else**

{

parent\_id = newNode();

FNodeArr[parent\_id].Parent = FNodeArr[splited\_Node\_Id].Parent;

FNodeArr[FNodeArr[parent\_id].Parent].setChild(FNodeArr[FNodeArr[parent\_id].Parent].FChildren.Length, parent\_id);

FNodeArr[splited\_Node\_Id].Parent = parent\_id;

FNodeArr[parent\_id].setChild(**0**, splited\_Node\_Id);

//parent\_id = FNodeArr[splited\_Node\_Id].Parent;

}

arr\_node = **new** **int**[MAX\_M + **1**];

**for** (i = **0**; i < arr\_node.Length - **1**; i++)

arr\_node[i] = FNodeArr[splited\_Node\_Id].getChild(i);

arr\_node[arr\_node.Length - **1**] = inserted\_Node\_Id;

node\_1\_min = **new** TRNode();

node\_2\_min = **new** TRNode();

node\_1 = **new** TRNode();

node\_2 = **new** TRNode();

axe = chooseSplitAxis(splited\_Node\_Id, inserted\_Node\_Id);

area\_min = **double**.MaxValue;

**for** (i = **0**; i <= **1**; i++)

{

QuickSort(arr\_node, **0**, arr\_node.Length - **1**, axe, (TBound)i);

**for** (k = MIN\_M - **1**; k <= MAX\_M - MIN\_M; k++)

{

node\_1.clearChildren();

node\_2.clearChildren();

j = **0**;

**while** (j <= k)

{

node\_1.setChild(j, arr\_node[j]);

j++;

}

**for** (j = k; j < arr\_node.Length - **1**; j++)

node\_2.setChild(j - k, arr\_node[j + **1**]);

updateMBR(node\_1);

updateMBR(node\_2);

Area = node\_1.Area() + node\_2.Area();

**if** (Area < area\_min)

{

node\_1\_min.copy(node\_1);

node\_2\_min.copy(node\_2);

area\_min = Area;

}

}

}

node\_1\_min.Level = FNodeArr[splited\_Node\_Id].Level;

node\_2\_min.Level = FNodeArr[splited\_Node\_Id].Level;

FNodeArr[splited\_Node\_Id].copy(node\_1\_min);

FNodeArr[splited\_Node\_Id].Parent = parent\_id;

new\_child\_id = newNode();

FNodeArr[new\_child\_id].copy(node\_2\_min);

FNodeArr[new\_child\_id].Parent = parent\_id;

node\_1 = **null**;

node\_2 = **null**;

node\_1\_min = **null**;

node\_2\_min = **null**;

**for** (i = **0**; i < FNodeArr[new\_child\_id].FChildren.Length; i++)

FNodeArr[FNodeArr[new\_child\_id].getChild(i)].Parent = new\_child\_id;

**if** (FNodeArr[parent\_id].FChildren.Length < MAX\_M)

{

FNodeArr[parent\_id].setChild(FNodeArr[parent\_id].FChildren.Length, new\_child\_id);

FNodeArr[parent\_id].FisLeaf = **false**;

updateMBR(parent\_id);

}

**else**

**splitNodeRStar**(parent\_id, new\_child\_id);

}

#endregion

/// <summary>

/// метод создания нового узла. Возвращает индекс только что созданого узла

/// </summary>

/// <returns></returns>

**private** **int** **newNode**()

{

//Console.WriteLine("NewNode work");

Array.Resize(**ref** FNodeArr, FNodeArr.Length + **1**);

FNodeArr[FNodeArr.Length - **1**] = **new** TRNode();

**return** FNodeArr.Length - **1**;

}

/// <summary>

/// метод определяющий является ли корнем узел с индексом node\_id

/// </summary>

/// <param name="node\_id"></param>

/// <returns></returns>

**private** **bool** **isRoot**(**int** node\_id)

{

**if** (node\_id == FRoot) **return** **true**; **else** **return** **false**;

}

/// <summary>

/// обновляет MBR узла

/// </summary>

/// <param name="node"></param>

#region

**public** **void** **updateMBR**(TRNode node)

{

//Console.WriteLine("updateMBR1 work");

**int** i, idx;

**bool** changed = **false**;

node.fmbr.Left.X = **double**.MaxValue;

node.fmbr.Left.Y = **double**.MinValue;

node.fmbr.Right.X = **double**.MinValue;

node.fmbr.Right.Y = **double**.MaxValue;

**if** (node.getIsLeaf())

{

**for** (i = **0**; i < node.FObject.Length; i++)

{

**if** (node.FObject[i].x < node.fmbr.Left.X)

{

node.fmbr.Left.X = node.FObject[i].x;

changed = **true**;

}

**if** (node.FObject[i].y > node.fmbr.Left.Y)

{

node.fmbr.Left.Y = node.FObject[i].y;

changed = **true**;

}

**if** (node.FObject[i].x > node.fmbr.Right.X)

{

node.fmbr.Right.X = node.FObject[i].x;

changed = **true**;

}

**if** (node.FObject[i].y < node.fmbr.Right.Y)

{

node.fmbr.Right.Y = node.FObject[i].y;

changed = **true**;

}

}

}

**else**

{

**for** (i = **0**; i < node.FChildren.Length; i++)

{

idx = node.FChildren[i];

**if** (FNodeArr[idx].fmbr.Left.X < node.fmbr.Left.X)

{

node.fmbr.Left.X = FNodeArr[idx].fmbr.Left.X;

changed = **true**;

}

**if** (FNodeArr[idx].fmbr.Left.Y > node.fmbr.Left.Y)

{

node.fmbr.Left.Y = FNodeArr[idx].fmbr.Left.Y;

changed = **true**;

}

**if** (FNodeArr[idx].fmbr.Right.X > node.fmbr.Right.X)

{

node.fmbr.Right.X = FNodeArr[idx].fmbr.Right.X;

changed = **true**;

}

**if** (FNodeArr[idx].fmbr.Right.Y < node.fmbr.Right.Y)

{

node.fmbr.Right.Y = FNodeArr[idx].fmbr.Right.Y;

changed = **true**;

}

}

}

**if** (changed)

**if** (node.Parent >= **0**) updateMBR(node.Parent);

}

**private** **void** **updateMBR**(**int** node\_id)

{

//Console.WriteLine("updateMBR2 work");

**int** i, idx;

**bool** changed = **false**;

FNodeArr[node\_id].fmbr.Left.X = **double**.MaxValue;

FNodeArr[node\_id].fmbr.Left.Y = **double**.MinValue;

FNodeArr[node\_id].fmbr.Right.X = **double**.MinValue;

FNodeArr[node\_id].fmbr.Right.Y = **double**.MaxValue;

**if** (FNodeArr[node\_id].getIsLeaf())

{

**for** (i = **0**; i < FNodeArr[node\_id].FObject.Length; i++)

{

**if** (FNodeArr[node\_id].getObject(i).x < FNodeArr[node\_id].fmbr.Left.X)

{

FNodeArr[node\_id].fmbr.Left.X = FNodeArr[node\_id].getObject(i).x;

changed = **true**;

}

**if** (FNodeArr[node\_id].getObject(i).y > FNodeArr[node\_id].fmbr.Left.Y)

{

FNodeArr[node\_id].fmbr.Left.Y = FNodeArr[node\_id].getObject(i).y;

changed = **true**;

}

**if** (FNodeArr[node\_id].getObject(i).x > FNodeArr[node\_id].fmbr.Right.X)

{

FNodeArr[node\_id].fmbr.Right.X = FNodeArr[node\_id].getObject(i).x;

changed = **true**;

}

**if** (FNodeArr[node\_id].getObject(i).y < FNodeArr[node\_id].fmbr.Right.Y)

{

FNodeArr[node\_id].fmbr.Right.Y = FNodeArr[node\_id].getObject(i).y;

changed = **true**;

}

}

}

**else**

{

**for** (i = **0**; i < FNodeArr[node\_id].FChildren.Length; i++)

{

idx = FNodeArr[node\_id].FChildren[i];

**if** (FNodeArr[idx].fmbr.Left.X < FNodeArr[node\_id].fmbr.Left.X)

{

FNodeArr[node\_id].fmbr.Left.X = FNodeArr[idx].fmbr.Left.X;

changed = **true**;

}

**if** (FNodeArr[idx].fmbr.Left.Y > FNodeArr[node\_id].fmbr.Left.Y)

{

FNodeArr[node\_id].fmbr.Left.Y = FNodeArr[idx].fmbr.Left.Y;

changed = **true**;

}

**if** (FNodeArr[idx].fmbr.Right.X > FNodeArr[node\_id].fmbr.Right.X)

{

FNodeArr[node\_id].fmbr.Right.X = FNodeArr[idx].fmbr.Right.X;

changed = **true**;

}

**if** (FNodeArr[idx].fmbr.Right.Y < FNodeArr[node\_id].fmbr.Right.Y)

{

FNodeArr[node\_id].fmbr.Right.Y = FNodeArr[idx].fmbr.Right.Y;

changed = **true**;

}

}

}

**if** (changed)

**if** (FNodeArr[node\_id].Parent >= **0**) updateMBR(FNodeArr[node\_id].Parent);

}

#endregion

/// <summary>

/// TAXis

/// </summary>

/// <param name="obj"></param>

/// <param name="node\_id"></param>

/// <returns></returns>

#region

**private** TAxis **chooseSplitAxis**(MyLib.Point obj, **int** node\_id)

{

//Console.WriteLine("chooseSplitAxis1 work");

MyLib.Point[] arr\_obj;

**int** i, j, k, idx;

TRNode node\_1, node\_2;

**double** perimetr\_min, perimetr;

TAxis result = **new** TAxis();

arr\_obj = **new** MyLib.Point[MAX\_M + **1**];

**if** (!FNodeArr[node\_id].isLeaf) **return** **0**;

**for** (i = **0**; i < FNodeArr[node\_id].FObject.Length; i++)

arr\_obj[i] = FNodeArr[node\_id].FObject[i];

arr\_obj[arr\_obj.Length - **1**] = obj;

node\_1 = **new** TRNode();

node\_2 = **new** TRNode();

perimetr\_min = **double**.MaxValue;

**for** (i = **0**; i <= **1**; i++)

{

perimetr = **0**;

**for** (j = **0**; j <= **1**; j++)

{

node\_1.clearObjects();

node\_2.clearObjects();

QuickSort(arr\_obj, **0**, arr\_obj.Length - **1**, (TAxis)i);

**for** (k = **1**; k <= MAX\_M - MIN\_M \* **2** + **2**; k++) // высчитваем периметры

{

idx = **0**;

**while** (idx < ((MIN\_M - **1**) + k))

{

node\_1.setObject(idx, arr\_obj[idx]);

idx++;

}

**for** (; idx < arr\_obj.Length; idx++)

node\_2.setObject(idx - ((MIN\_M - **1**) + k), arr\_obj[idx]);

updateMBR(node\_1);

updateMBR(node\_2);

perimetr = perimetr + ((node\_1.mbr.Right.X - node\_1.mbr.Left.X) \* **2** + (node\_2.mbr.Left.Y - node\_2.mbr.Right.Y) \* **2**);

}

}

**if** (perimetr <= perimetr\_min)

{

result = (TAxis)i;

perimetr\_min = perimetr;

}

}

**return** result;

}

**private** TAxis **chooseSplitAxis**(**int** nodeFather, **int** nodeChild)

{

//Console.WriteLine("chooseSplitAxis work");

**int**[] arr\_node;

**int** i, j, k, idx;

TRNode node\_1, node\_2;

**double** perimetr, perimetr\_min;

TAxis result = **new** TAxis();

arr\_node = **new** **int**[MAX\_M + **1**];

**for** (i = **0**; i < FNodeArr[nodeFather].FChildren.Length; i++)

arr\_node[i] = FNodeArr[nodeFather].FChildren[i];

arr\_node[arr\_node.Length - **1**] = nodeChild;

perimetr\_min = **double**.MaxValue;

node\_1 = **new** TRNode();

node\_2 = **new** TRNode();

**for** (i = **0**; i <= **1**; i++)

{

perimetr = **0**;

**for** (j = **0**; j <= **1**; j++)

{

node\_1.clearChildren();

node\_2.clearChildren();

QuickSort(arr\_node, **0**, arr\_node.Length - **1**, (TAxis)i, (TBound)j);

**for** (k = **1**; k <= MAX\_M - MIN\_M \* **2** + **2**; k++)

{

idx = **0**;

**while** (idx < ((MIN\_M - **1**) + k))

{

node\_1.setChild(idx, arr\_node[idx]);

idx++;

}

**for** (; idx < arr\_node.Length; idx++)

node\_2.setChild(idx - ((MIN\_M - **1**) + k), arr\_node[idx]);

updateMBR(node\_1);

updateMBR(node\_2);

perimetr = perimetr + node\_1.margin() + node\_2.margin();

}

}

**if** (perimetr <= perimetr\_min)

{

result = (TAxis)i;

perimetr\_min = perimetr;

}

perimetr = **0**;

}

node\_1 = **null**;

node\_2 = **null**;

arr\_node = **new** **int**[**0**];

**return** result;

}

#endregion

**private** **int** **chooseSubtree**(MyLib.Point obj, **int** node\_id)

{

//Console.WriteLine("chooseSubtree work");

**int** i, id\_child;

**int**[] idChild\_area = **new** **int**[**0**];

**int** id\_zero;

**double** dspace;

**if** (FNodeArr[node\_id].isLeaf) **return** node\_id;

Array.Resize(**ref** idChild\_area, FNodeArr[node\_id].FChildren.Length);

dspace = **double**.MaxValue;

id\_zero = **0**;

**for** (i = **0**; i < FNodeArr[node\_id].FChildren.Length; i++)

idChild\_area[i] = FNodeArr[node\_id].FChildren[i];

dspace = **double**.MaxValue;

**for** (i = **0**; i < idChild\_area.Length; i++)

{

id\_child = idChild\_area[i];

**if** (FNodeArr[id\_child].Area() < dspace)

{

id\_zero = idChild\_area[i];

dspace = FNodeArr[id\_child].Area();

}

}

node\_id = id\_zero;

chooseSubtree(obj, node\_id);

**return** node\_id;

}

/// <summary>

/// Конструктор

/// </summary>

**public** **TRtree**()

{

FNodeArr = **new** TRNode[**1**];

FNodeArr[**0**] = **new** TRNode();

FRoot = **0**;

FNodeArr[FRoot].FisLeaf = **true**;

}

**static** **int** ct=**0**;

/// <summary>

/// метод для вставки обьекта в дерево

/// </summary>

/// <param name="obj"></param>

**public** **void** **insertObject**(MyLib.Point obj)

{

**int** node\_id = FRoot;

Console.WriteLine(ct++);

node\_id = chooseSubtree(obj, node\_id);

**if** (FNodeArr[node\_id].FObject.Length < MAX\_M) //если количество объектов в узле меньше максимального

{

FNodeArr[node\_id].setObject(FNodeArr[node\_id].FObject.Length, obj);

updateMBR(node\_id);

}

**else**//если количество объектов достигло максимально допустимого

{

splitNodeRStar(node\_id, obj); //делим узел

}

}

**public** List<MyLib.Point> findobjectinarea(TMBR mbr, **int** node\_id, List<MyLib.Point> array, **int** idx)

{

**if** (FNodeArr[node\_id].FChildren.Length > **0**)

{

**for** (**int** i = **0**; i < FNodeArr[node\_id].FChildren.Length; i++)

{

**if** (FNodeArr[FNodeArr[node\_id].FChildren[i]].IsIntersected(mbr) && !FNodeArr[FNodeArr[node\_id].FChildren[i]].IsVisited) { findobjectinarea(mbr, FNodeArr[node\_id].FChildren[i], array, idx); }

}

}

**else**

{

array = findObjectinarea(mbr, node\_id, array, idx);

FNodeArr[node\_id].IsVisited = **true**;

}

//Console.WriteLine("count " + array.Count);

**return** array;

}

**public** List<MyLib.Point> findObjectinarea(TMBR mbr, **int** node\_id, List<MyLib.Point> array, **int** idx)

{

**double** eps = Dbscanwithrtree.eps;

MyLib.Point f = **new** MyLib.Point(mbr.Left.X + eps, mbr.Left.Y - eps, -**1**);

**for** (**int** i = **0**; i < FNodeArr[node\_id].FObject.Length; i++)

**if** (FNodeArr[node\_id].FObject[i].idx != idx && f.Distance(FNodeArr[node\_id].FObject[i]) <= eps) array.Add(FNodeArr[node\_id].FObject[i]);

**return** array;

}

/// <summary>

/// Свойство, возвращающее высоту дерева

/// </summary>

**public** **int** Height

{

**get**

{

**return** FHeight;

}

}

}

}

## **1.7. Класс Dbscan.cs**

**using** **System.Collections.Generic**;

**namespace** **MyLib**

{

**public** **class** **dbscan**

{

**static** **double** eps;

**static** **int** minPTS;

**public** **static** **int** **Clustering**(List<Point> points, **double** e, **int** mpts)

{

eps = e;

minPTS = mpts;

**int** cluster\_number = **0**;

CheckAllPoint(points);

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

{

//points[i].dbscan\_result = true;

**if** (points[i].IsVisited == **false** && points[i].minPTScheck)

{

points[i].IsVisited = **true**;

points[i].db = cluster\_number;

StartClusterization(points, points[i],cluster\_number);

cluster\_number++;

}

}

**return** cluster\_number;

}

**static** **void** **CheckAllPoint**(List<Point> points)

{

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

{

**int** count = -**1**;

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

**if** (points[i].Distance(points[j]) <= eps) count++;

**if** (count >= minPTS) points[i].minPTScheck = **true**;

}

}

**static** **void** **StartClusterization**(List<Point> points, Point p, **int** cluster)

{

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

{

**if** (points[i].IsVisited == **false** && p.Distance(points[i]) <= eps && points[i].minPTScheck) { points[i].db = cluster; points[i].IsVisited = **true**; StartClusterization(points, points[i],cluster); }

}

}

}

}

## **1.8. Класс Kmeans.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **MyLib**

{

**public** **class** **kmeans**

{

**public** **static** **int**[] **Cluster**(List<Point> rawData, **int** numClusters, **int** seed)

{

//массив для итоговых значений кластеров

**int**[] clustering = **new** **int**[rawData.Count];

//копируем точки в новый лист

List<Point> data = **new** List<Point>(**0**);

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

data.Add(**new** Point(rawData[i].x, rawData[i].y));

Normalized(data);

**bool** changed = **true**;

**bool** success = **true**;

List<Point> means = InitMeans(numClusters, data, seed);

Console.WriteLine("count" + means.Count);

**int** maxCount = rawData.Count \* **10**;

**int** ct = **0**;

**while** (changed == **true** && success == **true** && ct < maxCount)

{

changed = UpdateClustering(data, means);

success = UpdateMeans(data, means);

++ct;

}

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

clustering[i] = data[i].cluster;

**return** clustering;

}

/// <summary>

/// Инициализация центроидов для кластеризации

/// </summary>

/// <param name="numClusters">Количество кластеров</param>

/// <param name="data">Точки</param>

/// <param name="seed"></param>

/// <returns>Вовзвращает лист центров</returns>

**static** List<Point> InitMeans(**int** numClusters, List<Point> data, **int** seed)

{

List<Point> means = **new** List<Point>(numClusters);

**for** (**int** i = **0**; i < numClusters; i++)

means.Add(**new** Point(**0**, **0**));

List<**int**> used = **new** List<**int**>();

Random rnd = **new** Random(seed);

**int** idx = rnd.Next(**0**, data.Count);

means.Add(data[idx]);

used.Add(idx);

**for** (**int** k = **1**; k < numClusters; k++)

{

**double**[] dSquared = **new** **double**[data.Count];

**int** newMean = -**1**;

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

{

**if** (used.Contains(i) == **true**) **continue**;

**double**[] distances = **new** **double**[k];

**for** (**int** j = **0**; j < k; ++j)

distances[j] = data[i].Distance(means[k]);

**int** m = MinIndex(distances);

dSquared[i] = distances[m] \* distances[m];

}

**double** p = rnd.NextDouble();

**double** sum = **0.0**;

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

sum += dSquared[i];

**double** cumulative = **0.0**;

**int** ii = **0**;

**int** sanity = **0**;

**while** (sanity < data.Count \* **2**) // 'stochastic acceptance'

{

cumulative += dSquared[ii] / sum;

**if** (cumulative >= p && used.Contains(ii) == **false**)

{

newMean = ii; // the chosen index

used.Add(newMean); // don't pick again

**break**;

}

++ii; // next candidate

**if** (ii >= dSquared.Length) ii = **0**; // back to first item

++sanity;

}

// check if newMean is still -1 . . .

// save the data of the chosen index

means[k] = data[newMean];

}

**return** means;

} // InitMeans

/// <summary>

/// Нормализация

/// </summary>

/// <param name="rawData">Лист с точками</param>

**static** **void** **Normalized**(List<Point> rawData)

{

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

{

**double** colSum = **0.0**;//сумма всех точек по одной из координат (j==0)?x:y;

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

**if** (j == **0**) colSum += rawData[i].x; **else** colSum += rawData[i].y;

**double** mean = colSum / rawData.Count;//среднее всех точек по одной из координат

**double** sum = **0.0**;

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

**if** (j==**0**) sum += (rawData[i].x - mean) \* (rawData[i].x - mean); **else** sum += (rawData[i].y - mean) \* (rawData[i].y - mean);

**double** sd = sum / rawData.Count;

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

**if** (j == **1**) rawData[i].xnorm = (rawData[i].x - mean) / sd; **else** rawData[i].ynorm = (rawData[i].y - mean) / sd;

}

}

**private** **static** **bool** **UpdateMeans**(List<Point> data, List<Point> means)

{

**int** numClusters = means.Count;

**int**[] clusterCounts = **new** **int**[numClusters];

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

{

**int** cluster = data[i].cluster;

++clusterCounts[cluster];

}

**for** (**int** k = **0**; k < numClusters; k++)

**if** (clusterCounts[k] == **0**)

**return** **false**; // bad clustering. no change to means[][]

// update, zero-out means so it can be used as scratch matrix

**for** (**int** k = **0**; k < means.Count; k++)

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

**if** (j == **0**) means[k].xnorm = **0.0**; **else** means[k].ynorm = **0.0**;

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

{

**int** cluster = data[i].cluster;

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

**if** (j == **0**) means[cluster].xnorm += data[i].xnorm; **else** means[cluster].ynorm += data[i].ynorm;

}

**for** (**int** k = **0**; k < means.Count; k++)

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

**if** (j==**0**) means[k].xnorm /= clusterCounts[k]; **else** means[k].ynorm /= clusterCounts[k];

**return** **true**;

}

**private** **static** **bool** **UpdateClustering**(List<Point> data, List<Point> means)

{

**int** numClusters = means.Count;

**bool** changed = **false**;

**int**[] newClustering = **new** **int**[data.Count]; // proposed result

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

newClustering[i] = data[i].cluster;

**double**[] distances = **new** **double**[numClusters]; // from curr tuple to each mean

**for** (**int** i = **0**; i < data.Count; i++) // walk thru each tuple

{

**for** (**int** k = **0**; k < numClusters; k++)

distances[k] = data[i].Distance(means[k]); // usually Euclidean

**int** newClusterID = MinIndex(distances); // find closest mean ID

//Console.WriteLine("new cluster Id = " + newClusterID);

//Console.ReadLine();

**if** (newClusterID != newClustering[i])

{

changed = **true**;

newClustering[i] = newClusterID; // update

}

}

**if** (changed == **false**)

**return** **false**; // no change so bail and don't update clustering[][]

// check proposed clustering[] cluster counts

**int**[] clusterCounts = **new** **int**[numClusters];

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

{

**int** cluster = newClustering[i];

++clusterCounts[cluster];

}

**for** (**int** k = **0**; k < numClusters; k++)

**if** (clusterCounts[k] == **0**)

**return** **false**; // bad clustering. no change to clustering[][]

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

data[i].cluster = newClustering[i];

**return** **true**;

}

**private** **static** **int** **MinIndex**(**double**[] distances)

{

// index of smallest value in array

// helper for UpdateClustering()

**int** indexOfMin = **0**;

**double** smallDist = distances[**0**];

**for** (**int** k = **0**; k < distances.Length; k++)

{

**if** (distances[k] < smallDist)

{

smallDist = distances[k];

indexOfMin = k;

}

}

**return** indexOfMin;

}

}

}

## **1.9. Класс Point.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **MyLib**

{

/// <summary>

/// Класс для точки на плоскости

/// </summary>

**public** **class** **Point**

{

**public** **double** x, y; //абцисса и ордината

**public** **int** cluster; //номер кластера

**public** **int** db = -**1**;

**public** **bool** IsVisited = **false**;

**public** **bool** minPTScheck = **false**;

**public** **double** xnorm, ynorm; //нормализованные координаты

**public** **int** idx;

**public** List<Point> findpoint;

/// <summary>

/// Конструктор с параметрами

/// </summary>

/// <param name="x">Абцисса</param>

/// <param name="y">Ордината</param>

**public** **Point**(**double** x, **double** y)

{

**this**.x = x;

**this**.y = y;

}

**public** **Point**(**double** x, **double** y, **int** idx)

{

**this**.x = x;

**this**.y = y;

**this**.idx = idx;

}

/// <summary>

/// Метод, расчитывающий расстояние между двумя точками

/// </summary>

/// <param name="first">Первая точка</param>

/// <returns>Возвращает вещественное расстояние между точкой и заданной</returns>

**public** **double** **Distance**(Point first)

{

**return** Math.Sqrt(Math.Pow(first.x - **this**.x, **2**) + Math.Pow(first.y - **this**.y, **2**));

}

}

}

## **1.10. Класс WorkWithFilePoint.cs**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов | | | | Всего листов (страниц) в документе | № документа | Входя- щий № сопро- водит. докум. и дата | Под- пись | Дата |
| изменен- ных | заменен- ных | новых | аннулиро- ванных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |