**Министерство образования и науки Российской Федерации**

*Федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования*   
**«АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И.И. ПОЛЗУНОВА» (АлтГТУ)**

Факультет информационных технологий

Кафедра прикладной математики

Отчёт по дисциплине «Алгоритмы и структуры данных»

Лабораторная работа №8

«Вычислительная геометрия»

Выполнил:

Студент гр. ПИ-32

Глушков Г.Г.

Проверил:

ст. преподаватель Н.Д. Бубнова

Барнаул 2015

* Формулировка задачи:

Ввести число точек и сгенерировать их координаты. Найти все выпуклые слои.

* Алгоритм решения:
* Введем нужное нам количество точек и рандомно сгенерируем их координаты х и у;
* Далее добавляем все вершины в список вершин;
* И выводим координаты точек в ListBox1;
* Далее используем так называемый алгоритм быстрой оболочки:
* На начальном этапе мы имеем множество точек на плоскости;
* Находим самую левую(Л) и самую правую(П) точки из представленных. Если существует несколько самых левых или самых правых точек – то берем любую из них;
* Формируем списки точек S1, лежащий выше прямой ЛП, и список точек S2, находящихся ниже ЛП.   
  Теперь наша задача построить выпуклую оболочку для верхней полуплоскости(S1) и нижней(S2), а затем грамотно их объединить в одну;
* функция принимает на вход два параметра: набор точек(**vertex**) и контейнер(**convexHull**), который должен содержать индексы точек, образующих выпуклую оболочку, по завершению работы функции. Эти индексы соответствуют индексам точек из массива **vertex**.   
  В конце функции есть два однотипных вызова рекурсивной функции, которая строит выпуклую оболочку сначала для набора точек S1(строчка 29), затем для набора точек S2;
* Рассмотрим, как ведет себя функция для набора точек **S1**.   
  Сначала находим точку **TOP**, входящую в набор **S1**, такую, что площадь треугольника **(Л-TOP-П) максимальна.** Если таких точек несколько нужно взять самую **левую**. Как можно догадаться, точка **TOP** будет максимально удаленной от прямой **ЛП.** Точка **TOP** точно входит в выпуклую оболочку. Затем получаем набор точек **S11**, лежащий левее прямой **Л-TOP** и набора точек **S12**, находящихся левей прямой **TOP-П;**
* Далее мы повторяем все что проделали для набора точек S1 относительно прямой ЛП для S11 и S12. Не привожу в посте полный код рекурсивной функции, т.к. он довольно большой;
* Терминальным условием для рассматриваемой рекурсивной функции будет тот факт, что передаваемое множество точек, которое оказалось левее прямой на предыдущем шаге, пусто;
* Для нижней половины алгоритм работает идентично;
* Сложность данного алгоритма в среднем случае O(N\*log N), но в худшем случае, когда набор точек образует выпуклую оболочку сложность становится квадратичной.
* Но так как нам надо найти не оболочку, а слой, то после формирования одной оболочки, удаляем все точки, которые в ней содержаться и рекурсивно вызываем алгоритм для оставшихся точек;
* Отрисовываем систему координат точки и все выпуклые слои на экран.
* Текст программы с комментариями:

ConvexHull.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace Лаба\_8

{

class ConvexHull

{

private List<Point> vertex = null;//вершина

private List<List<Point>> listHulls = null; //список оболочек

private const double eps = 1e-8;

private class line

{

public int a, b, c;

public line() { }

public line(Point f, Point s)

{

a = s.Y - f.Y;

b = f.X - s.X;

c = -(a \* f.X + b \* f.Y);

}

public double dist(Point p)//расстояние

{

return Math.Abs(a \* p.X + b \* p.Y + c) / Math.Sqrt((double)a \* a + b \* b);

}

public bool isLeft(Point p)//левая точка

{

return a \* p.X + b \* p.Y + c < 0;

}

public bool isRight(Point p)//правая

{

return a \* p.X + b \* p.Y + c > 0;

}

}

public ConvexHull(List<Point> list)

{

vertex = list; //выпуклая оболочка

listHulls = new List<List<Point>>();

while(vertex.Count>2)//пока количество вершин больше двух

{

List<int> newList = solve();//решение

List<Point> newVertex = new List<Point>();

listHulls.Add(new List<Point>());//список оболочек

foreach (int num in newList)

{//добавляем вершину в список

listHulls[listHulls.Count - 1].Add(vertex[num]);

}

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

{

bool flag = false;

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

if (newList[j] == i) //если вернина уже есть в слое мы ее не добавляем в следующий слой

flag = true;

if (!flag)

newVertex.Add(vertex[i]);

}

vertex = newVertex;

}

}

//проверка, лежат ли точки на одной прямой

private bool SameLine(List<Point> points)

{

//менее 3 точек

if (points.Count < 3)

return true;

//более 2 точек

int k = (points[0].Y - points[1].Y) / (points[0].X - points[1].X), b = points[0].Y - points[0].X \* k;

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

if (points[i].Y != k \* points[i].X + b)

return false;

return true;

}

bool Equal(double a, double b)//равен

{

return Math.Abs(a - b) <= eps;

}

bool Less(double a, double b)//меньше

{

return !Equal(a, b) && a < b;

}

void GetPointsLeftByLine(List<Point> vertex, List<int> setPoints, line LINE, List<int> leftSetPoints)

{//получаем точки на одной линии

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

{

if (LINE.isLeft(vertex[setPoints[i]]))

leftSetPoints.Add(setPoints[i]);

}

}

void QuickHull(List<Point> vertex, List<int> convexHull, int leftPos, int rightPos, List<int> setPoints)

{//"быстрая оболочка"

if (setPoints.Count == 0)

{

convexHull.Add(rightPos);

return;

}

line LR = new line(vertex[leftPos], vertex[rightPos]);

// Находим точку, наиболее удаленную от прямой LR

int topPos = setPoints[0];

line topLine = new line(vertex[leftPos], vertex[topPos]);

double maxDist = LR.dist(vertex[topPos]);

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

{

if (setPoints[i] != leftPos && setPoints[i] != rightPos)

{

double curDist = LR.dist(vertex[setPoints[i]]);

// равноудаленные точки

if (Equal(maxDist, curDist))

{

// но угол у новой точки больше

if (topLine.isLeft(vertex[setPoints[i]]))

{

topPos = setPoints[i];

topLine = new line(vertex[leftPos], vertex[topPos]);

}

}

if (Less(maxDist, curDist))

{

maxDist = curDist;

topPos = setPoints[i];

topLine = new line(vertex[leftPos], vertex[topPos]);

}

}

}

List<int> S11 = new List<int>(); ;

line LT = new line(vertex[leftPos], vertex[topPos]);

// формируем множество точек, находящихся слева от прямой LT

GetPointsLeftByLine(vertex, setPoints, LT, S11);

QuickHull(vertex, convexHull, leftPos, topPos, S11);

List<int> S12 = new List<int>();

line TR = new line(vertex[topPos], vertex[rightPos]);

// формируем множество точек, находящихся слева от прямой TR

GetPointsLeftByLine(vertex, setPoints, TR, S12);

QuickHull(vertex, convexHull, topPos, rightPos, S12);

}

void QuickHull(List<Point> vertex, List<int> convexHull)

{

// нельзя построить выпуклую оболочку

if (vertex.Count < 3)

return;

// поиск самой левой и самой правой точки

int leftPos = 0, rightPos = 0;

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

{

if (Less(vertex[i].X, vertex[leftPos].X))

leftPos = i;

else if (Less(vertex[rightPos].X, vertex[i].X))

rightPos = i;

}

line LR = new line(vertex[leftPos], vertex[rightPos]);

List<int> S1 = new List<int>(); // точки выше прямой LR

List<int> S2 = new List<int>(); // точки ниже прямой LR

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

{

if (i != leftPos && i != rightPos)

{

if (LR.isLeft(vertex[i]))

S1.Add(i);

else if (LR.isRight(vertex[i]))

S2.Add(i);

}

}

QuickHull(vertex, convexHull, leftPos, rightPos, S1);

QuickHull(vertex, convexHull, rightPos, leftPos, S2);

}

double dist(Point a, Point b)

{

return Math.Sqrt((double)(a.X - b.X) \* (a.X - b.X) + (a.Y - b.Y) \* (a.Y - b.Y));

}

double findP(List<Point> vetex, List<int> convexHull)

{

double res = 0;

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

res += dist(vertex[convexHull[i]],

vertex[convexHull[(i + 1) % convexHull.Count]]);

return res;

}

List<int> solve()

{

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

QuickHull(vertex, convexHull);

return convexHull;

}

public List<List<Point>> ListHulls

{

get { return this.listHulls; }

}

}

}

Form1.cs:

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 Лаба\_8

{

public partial class Form1 : Form

{

SystemXY form = null;

public Form1()

{

InitializeComponent();

}

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

{

MessageBox.Show("Ввести число точек и сгенерировать их координаты. Найти все выпуклые слои.");

}

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

{

try

{

int n = Convert.ToInt32(textBox1.Text);

List<Point> list = new List<Point>();

Random random = new Random();

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

{

int x = random.Next(201) - 100, y = random.Next(201) - 100;

//генерация точек и вывод их на экран в лист бокс

list.Add(new Point(x, y));

listBox1.Items.Add("(" + list[i].X.ToString() + "; " + list[i].Y.ToString() + ")");

}

if (form != null)

{

form.Close();

form = null;

}

form = new SystemXY(list);

form.Show();

}

catch { MessageBox.Show("Ошибка при вводе количества точек!"); }

}

}

}

* Тесты:







