Joshua Van Steeter

CS 312 Section 2

Convex Hull Report

[1]  Functioning Source Code

using System;

using System.Collections.Generic;

using System.Text;

using System.Drawing;

namespace \_2\_convex\_hull

{

   class ConvexHullSolver

   {

       System.Drawing.Graphics g;

       System.Windows.Forms.PictureBox pictureBoxView;

       public ConvexHullSolver(System.Drawing.Graphics g, System.Windows.Forms.PictureBox pictureBoxView)

       {

           this.g = g;

           this.pictureBoxView = pictureBoxView;

       }

       public void Refresh()

       {

           // Use this especially for debugging and whenever you want to see what you have drawn so far

           pictureBoxView.Refresh();

       }

       public void Pause(int milliseconds)

       {

           // Use this especially for debugging and to animate your algorithm slowly

           pictureBoxView.Refresh();

           System.Threading.Thread.Sleep(milliseconds);

       }

       public void Solve(List<System.Drawing.PointF> pointList)

       {

           // Sort the given list

           pointList.Sort(new Comparor());

           // O(nlogn)

           ConvexHull convexHull = computeConvexHull(pointList);

           // Draw the result

           List<PointF> points = convexHull.getPoints();

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

           {

               g.DrawLine(new Pen(Color.Red, 2), points[i - 1], points[i]);

           }

           g.DrawLine(new Pen(Color.Red, 2), points[points.Count - 1], points[0]);

       }

       private ConvexHull computeConvexHull(List<PointF> pointList)

       {

           // Base case O(1)

           if(pointList.Count == 1)

           {

               return new ConvexHull(pointList[0]);

           }

           // Begin the dividing and conquering.

           // Divide into two branches of size n/2

           int halfWay = (pointList.Count) / 2;

           List<PointF> leftHalf = pointList.GetRange(0, halfWay);

           List<PointF> rightHalf = pointList.GetRange(halfWay, pointList.Count - halfWay);

           ConvexHull leftHull = computeConvexHull(leftHalf);

           ConvexHull rightHull = computeConvexHull(rightHalf);

           //combine the left and right hulls

           // O(n)

           findUT(leftHull, rightHull);

           findBT(leftHull, rightHull);

           // add points into one hull

           // O(n)

           ConvexHull newHull = new ConvexHull(leftHull.getUpperTangent());

           PointF nextPoint = leftHull.getUpperTangent();

           while(nextPoint != leftHull.getBottomTangent())

           {

               nextPoint = leftHull.getCounterWise(nextPoint);

               newHull.add(nextPoint);

           }

           nextPoint = rightHull.getBottomTangent();

           newHull.add(nextPoint);

           while(nextPoint != rightHull.getUpperTangent())

           {

               nextPoint = rightHull.getCounterWise(nextPoint);

               newHull.add(nextPoint);

           }

           return newHull;

       }

       private void findUT(ConvexHull leftHull, ConvexHull rightHull)

       {

           // O(n)

           PointF leftTangent = leftHull.getRightMost();

           PointF rightTangent = rightHull.getLeftMost();

           double slope = (leftTangent.Y - rightTangent.Y) / (leftTangent.X - rightTangent.X);

           bool done = false;

           while (!done)

           {

               done = true;

               double nextSlope;

               do

               {

                   PointF rightNext = rightHull.getClockWise(rightTangent);

                   nextSlope = (leftTangent.Y - rightNext.Y) / (leftTangent.X - rightNext.X);

                   if (nextSlope < slope)

                   {

                       slope = nextSlope;

                       rightTangent = rightNext;

                       done = false;

                   }

                   if(rightNext == rightHull.getClockWise(rightTangent))

                   {

                       break;

                   }

               } while (slope == nextSlope);

               do

               {

                   PointF leftNext = leftHull.getCounterWise(leftTangent);

                   nextSlope = (rightTangent.Y - leftNext.Y) / (rightTangent.X - leftNext.X);

                   if (nextSlope > slope)

                   {

                       slope = nextSlope;

                       leftTangent = leftNext;

                       done = false;

                   }

                   if (leftNext == leftHull.getCounterWise(leftTangent))

                   {

                       break;

                   }

               } while (slope == nextSlope);

           }

           leftHull.setUpperTangent(leftTangent);

           rightHull.setUpperTangent(rightTangent);

       }

       private void findBT(ConvexHull leftHull, ConvexHull rightHull)

       {

           // O(n)

           PointF leftTangent = leftHull.getRightMost();

           PointF rightTangent = rightHull.getLeftMost();

           double slope = (leftTangent.Y - rightTangent.Y) / (leftTangent.X - rightTangent.X);

           bool done = false;

           while (!done)

           {

               done = true;

               double nextSlope;

               do

               {

                   PointF rightNext = rightHull.getCounterWise(rightTangent);

                   nextSlope = (leftTangent.Y - rightNext.Y) / (leftTangent.X - rightNext.X);

                   if (nextSlope > slope)

                   {

                       slope = nextSlope;

                       rightTangent = rightNext;

                       done = false;

                   }

                   if (rightNext == rightHull.getCounterWise(rightTangent))

                   {

                       break;

                   }

               } while (slope == nextSlope);

               do

               {

                   PointF leftNext = leftHull.getClockWise(leftTangent);

                   nextSlope = (rightTangent.Y - leftNext.Y) / (rightTangent.X - leftNext.X);

                   if (nextSlope < slope)

                   {

                       slope = nextSlope;

                       leftTangent = leftNext;

                       done = false;

                   }

                   if (leftNext == leftHull.getClockWise(leftTangent))

                   {

                       break;

                   }

               } while (slope == nextSlope);

           }

           leftHull.setBottomTangent(leftTangent);

           rightHull.setBottomTangent(rightTangent);

       }

   }

   class Comparor : IComparer<PointF>

   {

       public int Compare(PointF x, PointF y)

       {

           if (x.X < y.X)

               return -1;

           else

               return 1;

       }

   }

   class ConvexHull

   {

       private PointF leftMost;

       private PointF rightMost;

       private PointF upperTangent;

       private PointF bottomTangent;

       private List<PointF> points;

       public ConvexHull(PointF point)

       {

           points = new List<PointF>();

           points.Add(point);

           leftMost = point;

           rightMost = point;

       }

       public void add(PointF point)

       {

           if(point.X < leftMost.X)

           {

               leftMost = point;

           }

           else if(point.X > rightMost.X)

           {

               rightMost = point;

           }

           points.Add(point);

       }

       public PointF getCounterWise(PointF point)

       {

           int index = points.IndexOf(point) + 1;

           if(index >= points.Count)

           {

               return points[0];

           }

           return points[index];

       }

       public PointF getClockWise(PointF point)

       {

           int index = points.IndexOf(point) - 1;

           if (index < 0)

           {

               return points[points.Count - 1];

           }

           return points[index];

       }

       public PointF getLeftMost()

       {

           return leftMost;

       }

       public PointF getRightMost()

       {

           return rightMost;

       }

       public PointF getUpperTangent()

       {

           return upperTangent;

       }

       public void setUpperTangent(PointF tangent)

       {

           upperTangent = tangent;

       }

       public PointF getBottomTangent()

       {

           return bottomTangent;

       }

       public void setBottomTangent(PointF tangent)

       {

           bottomTangent = tangent;

       }

       public int size()

       {

           return points.Count;

       }

       public List<PointF> getPoints()

       {

           return points;

       }

   }

}

[2] Explain time and space complexity

After we receive the list of points, we perform a merge sort with the help of C# default library.

O(nlogn)

Begin the divide and conquer section of the algorithm.  The sorted list is divided into two parts, each one half the size.  This recursive division continues until we reach the base case, lists of size one, the convex hulls are recombined with time complexity O(n).  At any given time the worst case scenario for the space complexity is every point, therefore the space complexity is O(n).

By entering the information for my algorithm (a=2, b=2, d=1) into the master algorithm for divide and conquer algorithms, we get total time complexity of O(nlogn) and a space complexity of O(n).

[3] Experimental Outcomes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | Time in Seconds | | | | | AVG |
| 10 | 0.017913 | 0.003403 | 0.002941 | 0.002684 | 0.002969 | 0.005982 |
| 100 | 0.00338 | 0.005091 | 0.003347 | 0.00299 | 0.002968 | 0.003555 |
| 1000 | 0.007663 | 0.007398 | 0.006896 | 0.007756 | 0.006931 | 0.007329 |
| 10000 | 0.045398 | 0.040349 | 0.039612 | 0.039128 | 0.040424 | 0.040982 |
| 100000 | 0.394319 | 0.384296 | 0.381226 | 0.409926 | 0.383283 | 0.39061 |
| 500000 | 1.859774 | 1.871539 | 1.888877 | 1.898005 | 1.890847 | 1.881808 |
| 1000000 | 3.783367 | 3.783167 | 3.797517 | 3.788614 | 3.794267 | 3.789387 |

Clearly this appears to be progressing at a logarithmic scale. By doing simple tests and calculating the average we can see the rate the algorithm progressing and I would expect to continue to progress logarithmically.

[4] Comparisons

The algorithm appears to be progressing about as we expected. We calculated a O(nlogn) computation time and we see something similar in the empirical data. I am surprised that the time from 10 to 100 and then again to 1000 does not increase as I would have expected, especially with the average time for 100 actually being less than 10. It’s not till we get to about 100,000 that we start to see that logarithmic growth that I expected.

[5]  Screen Shots



