ALDA VT12: Algoritmdesigntekniker

Leon Hennings leonh
Kamyar Sajjadi kamy-saj

23 februari 2012

Innehåll

1 Closest-Point Problem

1.1 Beskrivning

Vi har denna vecka valt att implementera en Divide and Conquer algoritm. Algoritmen vi valt är en algroritm som ska hitta de punkter som ligger närmast varandra. Problemet nämns i boken på sida 406, Closest-Points Problem. Vi valde att metoden ska returnera det lägsta avståndet. Problemet går att lösa genom att jämföra alla punkter mot varandra, detta genom att implementera en Brute Force metod som har tidskomplexitet O(N^2). Det går även att lösa problemet genom Divide and Conquer vilket då istället ger tidskomplexiteten O(n log n). Vi har valt att skriva en klass som har en ArrayList med java.awt.Point:s som element. Listan sorteras först med hänsyn till punkternas x värde. Algoritmen bestämmer en linje som delar listan på mitten i två logiska partitioner, en vänsterpartition och en högerpartition. Sedan kallas algoritmen rekursivt på de två partitionerna. Basecaset för den rekursiva algoritmen är att varje partition ska bestå av mindre än 3 element. När algoritmen nått basecaset så kommer partitionen att sökas igenom och det kortaste avståndet mellan de närmaste punkterna kommer returneras. Det lägsta av dessa sätts till smallest. Nu när vi går ur rekursionen så kollar algoritmen om det finns några punkter som ligger på kortare avstånd än smallest från den delande linjen. Dessa punkter läggs till en ny lista och denna sorteras med hänsyn till punkternas y värde. Arean består av de element som ligger inom intervallet (mittpunkten-minsta $<$ punktens X $<$ mittpunkten+minsta). Då punkterna är sorterade på y kan vi ignorera de fallen då differensen mellan punkternas y värde är större än smallest. Sedan kontrollerar vi om något par av punkter i denna area har kortare avstånd till varandra. Om detta värde är mindre än smallest sätts det till smallest som slutligen returneras.

1.2 Kod

```
1  /**
2   * Klassen för Closest-Pair problem.
3   * @author Leon Hennings
4   * @author Kamyar Sajjadi
5   */
6  public class ClosestPair{
7
8    // ArrayList för samtliga punkter
9    private ArrayList<Point> plane = new ArrayList<Point>();
10
11   /**
12    * Comperator för att jämföra i X-led
13    */
14   private class CmpX implements Comparator<Point>{
15     public int compare(Point p1, Point p2){
16       if(p1.x==p2.x)
17         return p1.y-p2.y;
18       else
19         return p1.x-p2.x;
20     }
21   }
22
23   /**
24    * Comparator för att jämföra i Y-led
25    */
26   private class CmpY implements Comparator<Point>{
27     public int compare(Point p1, Point p2){
28       if(p1.y==p2.y)
29         return p1.x-p2.x;
30       else
31         return p1.y-p2.y;
32     }
33   }
34
35   /**
36    * Metod för att lägga till punkter i arrayen
37    * @param p Som är en Point som ska läggas till i plane
38    */
39   public void addPoint(Point p){
40     plane.add(p);
41   }
42
43   /**
44    * toString för att få en bra överblick över listan.
45    * Denna metod är mest användbar när man ska debugga.
46    * @return str
47    */
48   public String toString(){
49     String str= "";
50     for(Point p : plane){
51       str+= "(" + p.x + p.y+ ")" + "\n";
52     }
```

```
53      return str;
54    }
55
56    /**
57     * Metod för att få storleken av listan.
58     * @return the size of the array
59     */
60    public int getPlaneSize(){
61      return plane.size();
62    }
63
64    /**
65     * Räknar ut avståndet mellan två punkter.
66     * @param p1 En av punkterna
67     * @param p2 den andra punkten
68     * @return en double med avståndet från punkt p1 och punkt p2
69     */
70    private double getDistance(Point p1, Point p2){
71      return Math.sqrt( Math.pow((p1.getX()-p2.getX()), 2.0) + Math.
            pow((p1.getY()-p2.getY()), 2.0) );
72    }
73
74    /**
75     * Den publika metoden för att hitta de kortaste avståndet.
76     * denna metod kommer att sortera listan i X-led och sedan
77     * anropa den privata findClosestPair.
78     */
79    public double findClosestPair(){
80      Collections.sort(plane, new CmpX());
81      return findClosestPair(0, plane.size()-1);
82    }
83
84    /**
85     * Den privata metoden för att hitta det korstaste avståndet.
86     * @param intervalStart
87     * @param instervalEnd
88     * @return smallest Kortaste avståndet.
89     */
90    private double findClosestPair(int intervalStart, int intervalEnd
        ){
91      // Basecase som vi når när listan har mindre än 3 element i sig
          .
92      if(intervalEnd-intervalStart<=2){
93        return findClosestInInterval(intervalStart, intervalEnd);
94      }
95      // Här kommer uppdelningen av listorna att ske. Vardera sida
96      // av listan kommer att anropas med findClosestPair.
97      else
98      {
99        // Detta är mitten av den lista som vi arbetar på just nu.
100       int middle = ((intervalStart+intervalEnd)/2);
101       // Detta är mitten fast i X-led.
102       double middleX = (plane.get(intervalStart).getX()+plane.get(
              intervalEnd).getX())/2;
```

```
103         // De rekursiva anropen.
104         double left = findClosestPair(intervalStart, middle);
105         double right = findClosestPair(middle+1, intervalEnd);
106         // Sätter smallest till det minsta av left och right
107         double smallest = (left < right ? left : right);
108 
109         // Här går vi igenom elementen för att hitta de element som
                ska ligga i
110         // mittenArean (strip). Dessa ska ligga i intervallet som vi
                har nämnt i beskrivningen.
111         // Sedan sorteras denna lista i Y-led.
112         ArrayList<Point> strip = new ArrayList<Point>();
113 
114         for(Point p : plane)
115           if(p.getX() > (middleX - smallest) && p.getX() < (middleX +
                  smallest))
116             strip.add(p);
117 
118         Collections.sort(strip, new CmpY());
119 
120         // En BruteForce metod för att kontrollera de element som vi
                lagt in
121         // i strip. Denna metod nämner Weiss i boken på sida 409.
122         for (int i=0; i<strip.size(); i++)
123           for (int j=i+1; j< strip.size(); j++)
124             if ((strip.get(j).y - strip.get(i).y) > smallest)
125               break;
126             else
127             {
128               double tmp = getDistance(strip.get(i), strip.get(j));
129               if(tmp < smallest)
130                 smallest = tmp;
131             }
132         return smallest;
133       }
134     }
135 
136     /**
137      * Brute force algoritm för basecase och testning av algoritmen.
138      * @param instervalStart
139      * @param intervalEnd
140      * @return delta som är det minsta avståndet mellan de närmaste
             punkterna.
141      */
142     public double findClosestInInterval(int intervalStart, int
            intervalEnd){
143       double delta=Double.MAX_VALUE;
144       for(int i=intervalStart; i<intervalEnd+1; i++){
145         Point tmp = plane.get(i);
146         for(int j = i+1; j<intervalEnd+1; j++ ){
147           double newDelta=getDistance(tmp, plane.get(j));
148           if(newDelta<delta){
149             delta=newDelta;
150           }
```

```
151         }
152       }
153       return delta;
154     }
155 }
```

2 Poolfrågor

Denna veckas fråga måste behandla giriga algoritmer.

2.1 Fråga 1

Förklara vad en girig algoritm är och ge ett par exempel på giriga algoritmer som vi har gått igenom under kursen. För ett högre betyg ska du förklara varför giriga algoritmer i vissa fall kan göra felberäkningar. Motivera ditt svar med exempel.