Separating Points By Axis Parallel Line

Sagar Prakash Mane

Illinois Institute of Technology

CWID : A20379756

smane1@hawk.iit.edu

Contents

Introduction

The goal of the project to implement several algorithms for the following problem: SEPARATING POINTS BY AXIS PARALLEL LINES.

Input: Set of n points in the two-dimensional plane, point I given by coordinates x_i and y_i. No two points have the same x or y coordinates.

Output: Set S of vertical or horizontal lines, each given by the direction and one coordinate, such that any two points of the input are separated by a line of S.

Algorithm

Step 1: Initially divide the plane by vertical and horizontal line, i.e. divided into 4 parts.

1.1 If part contains 2 points, then divide that point horizontally by finding medium of that point.

1.2 If part contains more than 2 points, then divide that plan again horizontally and vertically.

1.3 If points are not divided by horizontal line then remove that line.

Step 2: Repeat Step 1 until each divided part contains only a single point.

Input and Output Formats

The program read the input from a sequence of files called “instance01” and output solutions in the files “greedy_solution01”.

Each input file starts with n, the number of points, followed by n lines, each containing integers: the x and y coordinates of the point. The points are sorted by the x coordinates.

Example:

Instance01:

```
5
1 1
2 4
3 3
4 2
5 5
```

Greedy_soution01:

```
3
H 4.5
V 2.5
H 2.5
```

Pseudocode

Pseudocode for Main class

Main

```
1.      for(Files f exits)
2.      do arr ← ReadFile(filename)
3.      Divide(arr, 0, 0, arr.length, arr.length)
4.      WriteFile(result,filename)
```

Pseudocode for Divide Method/Function

Divide(coor[][], int x1, int y1, int x2, int y2)

```
1.      c1 ← 0; c2 ← 0; c3 ← 0; c4 ← 0
2.      for (i = 0 to i < coor.length)
3.          do if(coor[i][0] > x1 and coor[i][1] > y1 and coor[i][0] <= ((x2 + x1) / 2) and
                    coor[i][1] <= ((y2 + y1) / 2) )
4.              then do c1 ← c1+1
5.                      X_arr.add(coor[i][1])
        End for

6.      If(c1 =2) and result.contains("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)
7.      Then do result.add("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)

8.      X_arr.clear()

9.      for (i = 0 to i < coor.length)
10.         do if(coor[i][0] > x1 and coor[i][1] > (y1+y2)/2 and coor[i][0] <= ((x2 + x1) / 2) and
                    coor[i][1] <= y2 )
11.             then do c2 ← c2+1
12.                     X_arr.add(coor[i][1])
        End for

13.     If(c2 =2) and result.contains("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)
14.     Then do result.add("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)

15.     X_arr.clear()

16.     for (i = 0 to i < coor.length)
17.         do if(coor[i][0] > (x1+x2)/2 and coor[i][1] > (y1+y2)/2 and coor[i][0] <= x2 and
                    coor[i][1] <= y2 )
18.             then do c3 ← c3+1
19.                     X_arr.add(coor[i][1])
        End for
```

```
20.     If(c3 =2) and result.contains("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)
21.     Then do result.add("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)

22.     X_arr.clear()

23.     for (i = 0 to i < coor.length)
24.         do if(coor[i][0] > (x1+x2)/2 and coor[i][1] > y1 and coor[i][0] <= x2 and
                  coor[i][1] <= (y2+y1)/2 )
25.             then do c4 ← c4+1
26.                     X_arr.add(coor[i][1])
        End for

27.     If(c4 =2) and result.contains("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)
28.     Then do result.add("H " + ((x_arr.get(0) + x_arr.get(1)) / 2) + 0.5)

29.     X_arr.clear()

30.     if (!result.contains("V " + ((x1 + x2) / 2) + 0.5))
31.     Then do result.add("V " + ((x1 + x2) / 2) + 0.5)

32.     if (!result.contains("H " + ((y1 + y2) / 2) + 0.5) and c4!=0)
33.     Then do result.add("H " + ((y1 + y2) / 2) + 0.5)

34.     If(c1>2)
35.     The do Divide(coor, x1, y1, (x1+x2)/2, (y1+y2)/2)

36.     If(c2>2)
37.     The do Divide(coor, x1, (y1+y2)/2, (x1+x2)/2, y2)

38.     If(c3>2)
39.     The do Divide(coor,(x1+x2)/2, (y1+y2)/2, x2, y2)

40.     If(c4>2)
41.     The do Divide(coor, (x1+x2)/2, y1, x2, (y1+y2)/2)
```

Running time analysis

Running time Divide method/function

- Running time for line 2-5 is $O(n^2)$ as for loop run at most n times and adding operation in list take O(n) time.

-Running time for line 6-7 is O(n) as adding operation in list take O(n) time.

-Running time for line 8 is O(n) as clear operation in list take O(n) time.

- Running time for line 9-12 is $O(n^2)$ as for loop run at most n times and adding operation in list take O(n) time.

-Running time for line 13-14 is O(n) as adding operation in list take O(n) time.

-Running time for line 15 is O(n) as clear operation in list take O(n) time.

- Running time for line 16-19 is $O(n^2)$ as for loop run at most n times and adding operation in list take O(n) time.

-Running time for line 20-21 is O(n) as adding operation in list take O(n) time.

-Running time for line 22 is O(n) as clear operation in list take O(n) time.

- Running time for line 23-26 is $O(n^2)$ as for loop run at most n times and adding operation in list take O(n) time.

-Running time for line 27-28 is O(n) as adding operation in list take O(n) time.

-Running time for line 29 is O(n) as clear operation in list take O(n) time.

-Total running time from line 1 to 29 is $O(n^2)$

-For line 34-41 Divide function execute recursively, therefore we have following recurrence relationship

$T(n) = 4T(n/4) + O(n^2)$

Using Master theorem, case 3 apply. therefore, time complexity is

$T(n) = \theta(n^2)$

Therefore, total running time for Divide function is **$\theta(n^2)$.**

Running time for main class

-Running time for line 1 is m, where m is constant numbers of files.

-Running time for line 2 is O(2n)=O(n) as numbers of columns are 2 which is constant.

-Running time for line 3 is $\theta(n^2)$.

-Running time for line 4 is O(n).

Therefore, total running time for algorithm is $m(O(n) + \theta(n^2) + O(n))$ which equal to **$\theta(n^2)$.**

Example

Instance

```
10
1 10
2 6
3 8
4 1
5 3
6 7
7 2
8 9
9 5
10 4
```

After running above algorithm solution is following

Greedy_solution

```
7
h 2.5
h 8.5
v 5.5
h 5.5
v 2.5
h 4.5
v 7.5
```

More optimal solution for above instance is following

```
6
v 5.5
h 4.5
h 6.5
v 7.5
v 1.5
v 4.5
```