## SEPARATING POINTS BY AXIS-PARALLEL LINES using greedy approach:

Separating n points in the plane, no two of which have the same x- or y-coordinate, using a minimum number of vertical and horizontal lines avoiding the points, so that each cell of the subdivision contains at most one point.

**INPUT:** An integer "max_files" which has the number of input instances to be read. The program will read instance files each containing the coordinates of the points. The points are sorted along x-axis and they won't share each other's axes.

**OUTPUT:** A Feasible solution containing the set 'S' of axis-parallel lines that separate the given instance points such that |S| is minimum. The results are printed in a file.

**Pseudocode:**

Greedy_approach_method():

```
1.def disconnect(line,link,verticality=True):
2.      x=line
3.      y=x
4.      x1=link[0][0]
5.      y1=link[0][1]
6.      x2=link[1][0]
7.      y2=link[1][1]
8.      if verticality: #compare x coordinates and y coordinates
9.          if(x<x1 and x>x2)or (x>x1 and x<x2):
10.             return True
11.         else:
12.             return False
13.     else:
14.         if (y<y1 and y>y2) or (y>y1 and y<y2):
15.             return True
16.           else:
17.             return False
18.def FUNCT(line,lineCur,vertical):
19.     cut = 0
20.     cutMax = 0
21.     for link in links:
22.         if link!=None:
23.             if disconnect(line, link, verticality=vertical):
24.                 cut = cut + 1
25.     if (cut >= cutMax):
26.         cutMax = cut
27.         lineCur = line
28.     return(cutMax)
```

```
29.def findlinewithmostcuts(links,left,right,down,up):
30.     cutMax=0 #records the max cuts
31.     lineCur_h=0
32.     lineCur_v = 0
33.     lineCur = 0
34.     line_orientation ="h"
35.     h_cutmax = 0
36.     v_cutmax = 0
37.     h1 = down +0.5
38.     v1 = left + 0.5
39.     while (True):
40.         if h1 > up:
41.             break
42.         h_line_cutmax = FUNCT(h1, lineCur, vertical=False)
43.         if h_cutmax<h_line_cutmax:
44.             h_cutmax = h_line_cutmax
45.             lineCur_h = h1
46.         h1 = h1 + 1
47.     while(True):
48.         if v1 > right:
49.             break
50.         v_line_cutmax = FUNCT(v1,lineCur,vertical=True)
51.         if v_cutmax < v_line_cutmax:
52.             v_cutmax = v_line_cutmax
53.             lineCur_v = v1
54.         v1 = v1 +1
55.     lineCur = lineCur_h
56.     cutMax = h_cutmax
57.     if h_cutmax <= v_cutmax:
58.         cutMax = v_cutmax
59.         line_orientation = "v"
60.         lineCur = lineCur_v
61.     return(lineCur,cutMax,line_orientation)
62.if __name__ == '__main__':
63.     directory = listdir("input")    # array
    # this is used to load each instance file given for further use
64.     for t in directory:    # from variable name to name of the list
65.         points = []    # stores (x,y) for all points # reads the instance files from input folder
66.         fread = open("input/" + t, 'r')    # read is r, read and write is rw
67.         fread.readline()    # reading the first line of the file
68.         for L in fread:
69.             pnt = L.split(" ")
70.             points.append([int(pnt[0]), int(pnt[1])])
 71.         res = []    # stores the number of lines chosen
```

```
72.         links = []    # stores pair( p1,p2) for all pair of points
        # NOTE: Order doesn't matter here. (p1,p2) and (p2,p1) are the same
73.         left, right, up, down = 0, 0, 0, 0    # bounds
74.         for p in points:
75.             left = min(left, p[0])    # 0 since it is minimum and is stored as that
76.             right = max(right, p[1])    # 1 since it is maximum and is stored as that
77.             up = max(up, p[1])
78.             down = min(down, p[0])
79.         for i,p1 in enumerate(points):
80.             for j in range(i,len(points)):
81.                 if p1!=points[j]:
82.                     links.append((p1, points[j]))    # tuple is appended
83.         sum =0
84.         while((len(links)-sum)!= 0):
85.             line,cutMax,line_orientation=findlinewithmostcuts(links,left,right,down,up)
86.             res.append((line,line_orientation))
87.             if line_orientation == 'v':
88.                 vertical = True
89.             else:
90.                 vertical = False
91.             sum = sum + cutMax
92.             for s in range(len(links)):
93.                 if links[s] != None:
94.                     if disconnect(line,links[s],verticality=vertical):
95.                         links[s] = None #Print the results to the output files.
96. Print the results to the output files.
```

**Explanation:**

The implementation for Greedy_approach_method ( ) is a standalone algorithm that reads coordinate values from multiple files, finds their axis parallel lines, separates the given coordinates with the minimal number of axis-parallel lines and stores the result into multiple files. **Time complexity:**

Function FUNCT contains 1 while-loop for n-points, which cost $O(n^2)$ time. Function findlinewithmostcuts contains 2 while-loops for n-points, which cost $2O(n^2)$ time. Main function contains at least 1 while-loop and it has an inner while-loop for n-points, which cost $O(n^3)$ time. As given in the problem statement, the input instances would have at most 100 points and we can have up to 100 instances, and we have a while-loop that cycles over the given number of instance files, which means at the maximum we have 'n' time complexity for this loop. Hence the running time of the greedy algorithm is $n*(O(n^2)+ 2O(n^2)+ O(n^3))=O(n^4)$.

**Fails to return the optimum solution:**

We have assumed in our algorithm that no 2 points share the same 'x' or 'y coordinate axis. But, if we alter this assumption the algorithm won't find an optimal solution, as the greedy approach fails for arbitrary inputs that have significant overlaps. For example, consider an instance with 5 points:A(2, 2), B(9, 8), C(10, 5), D(10, 10), E(11, 8). We can see that there exists overlaps for the points B & E, and C & D for coordinate values y = 8 and x = 10, respectively.

The Greedy method gives the following solution (with 4 lines) :
v 10.0
h 8.0
v 9.5
h 6.5
While an optimum solution for the same would be:
v 10.5
v 9.5
h 6.5
But if we ensure that the input points have no-overlap coordinates, we get an optimal solution.
Let the altered points be:
1 10
2 6
3 8
4 1
5 3
6 7
7 2
8 9
9 5
10 4
For this input instance, we get:
v 5.5
h 4.5
h 6.5
v 7.5
v 1.5
v 4.5 - which is an optimum solution.
So we can draw a conclusion that the Greedy approach would fail to return an optimum solution if we have inputs of the graph with overlaps.

**References:**


1. https://core.ac.uk/download/pdf/185287459.pdf, COVERING POINTS WITH AXIS PARALLEL LINES.
2. https://github.com/nickziv/sep_pts
3. https://github.com/raj-kotak/Separating-Points-by-Axis-Parallel-Lines