Practice 11 – Linked List Implementation & its Applications

(12/13)

1. **Scan line**

   Sort processed objects with a key and then use the key as the metric of scan line to sweep all objects under processing in sorting order. During sweeping procedures, you can do any operation to realize the function you want. This kind of incremental operations can save much run time.

   A. **Example. Line overlap problem: sort all lines and scan all lines. During sweeping sorted lines, compare line overlap conditions for local lines.**

2. **Problem 2 (100 pts)**

One gardening company has to maintain lawns of his customers. The boundaries of each lawn must be rectilinear polygon. In order to systematically maintain lawns of any shape, this company develops one efficient way to partition and manage a lawn. The method is to extend each horizontal line to partition a lawn into several rectangles and then sort all sub-lawns in increasing order of the *y*-coordinate first and then *x*-coordinate of each sub-lawn, i.e., all sub-lawns are sorted in terms of bottom boundary from bottom to top and the sub-lawns with the same bottom boundary are sorted in terms of left boundary from left to right. A gardener will maintain all sub-lawns following the sorted order. During lawn partitioning, the gardener walks along the boundary of the lawn under processed counterclockwise, and then each line segment on the boundary is assigned as a walking direction, as shown in Fig. 2(a) and Fig. 3(a). The original idea is to project two vertical lines onto *y*-axis, one sub-lawn is generated if the projections of two vertical lines have an overlap. For instance, Fig. 3(b) shows one blue line and one red line that correspond to the projection of *e*1 and *e*2 on *y*-axis. Vertical lines *e*1 and *e*2 induce an overlap of 30-unit length on *y*-axis, which means one sub-lawn of 30-unit height can be produced. Actually this idea is not totally correct. For instance, although vertical lines *e*1 and *e*3 also induce a projection overlap of 20-unit length on *y*-axis, a sub-lawn cannot be generated. Similarly vertical lines *e*1 and *e*4 also induce a projection overlap of 20-unit length, but they still cannot generate a sub-lawn. Please refine this method and write a program to correctly partition each lawn. Figures 2(b) and 3(b) shows the correct partition outcome.

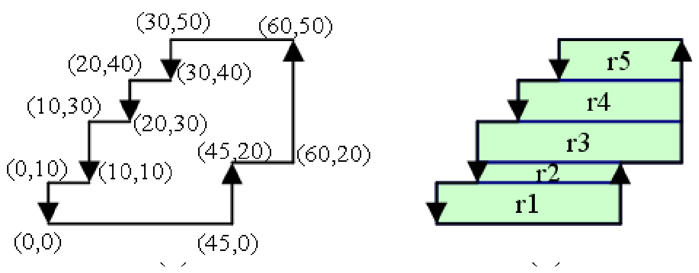
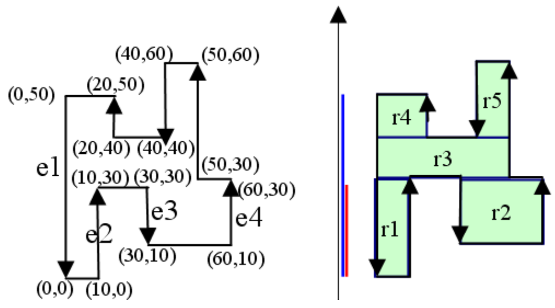


Fig. 2



(a) (b)

Fig. 3

Input Format:

The first line describes how many points a lawn has. After that each line describes one corner point with *x*-coordinate first and then *y*-coordinate separated by one space character. The first corner point can be any corner point of the lawn. The sequence of boundary corners is reported counterclockwise. The maximum number of corner points is 500. The values of *x*- and *y*-coordinates are integers between 0 and 10000. After describing the corner points, the demands for printing your results are described. Since to print all rectangles you generate is too tedious, you are given a number of sets of rectangles. For each rectangle set, you need to compute the summation of area of all rectangles in a set and report the total area for the set. In each line, the first number gives the number of rectangles in this set followed by the rectangle numbers separated by space character. Each rectangle is numbered by the order in the sorted list. The first rectangle in the sorted list is numbered as 1. For instance, a line of "3 1 3 5" means this set has three rectangles that are rectangles 1, 3 and 5. The file ends in a line starting with -1.

Input Example 1:

Assume the starting corner of the lawn in Fig. 2(a) is (0,0). The input file looks like this. Notably the starting corner and the last corner are adjacent.

```
12
0 0
45 0
45 20
60 20
```

60 50
30 50
30 40
20 40
20 30
10 30
10 10
0 10
3 1 2 3
3 1 2 5
3 2 4 5
-1

Input Example 2:

Assume the starting corner of the lawn in Fig. 3(a) is (0,0). The input file looks like this:

```
14
0 0
10 0
10 30
30 30
30 10
60 10
60 30
50 30
50 60
40 60
40 40
20 40
20 50
0 50
4 1 2 3 4
3 2 3 5
5 1 2 3 4 5
-1
```

Output Format:

Print the total number of rectangles you generated at the first line. **Compute the summation of area of rectangles of odd numbers. Print the total area at the second line.** Compute the average area for all rectangles. **Compute the summation**

of area of rectangles whose area are larger than or equal to the average area. Print the total area at the third line. Starting at the fourth line, print the total area of each rectangle set designated in input file line by line. In example 1, the total number of rectangles is 5. The total area of rectangles of odd number is 1250. Since the average area is 400 (2000/5 = 400), the area of rectangles 1, 3 and 4 is larger than or equal to 400 (their area is 450, 500, and 400). Thus the output at the third line is 1350. The first rectangle set contains rectangles 1, 2 and 3. The fourth line outputs 1300 (450 + 350 + 500). The remaining two rectangle sets are also reported similarly. **Notably your program is regarded as wrong if any one of these outputs is not correct.**

Output Example 1:

```
5
1250
1350
1300
1100
1050
```

Output Example 2:

```
5
1000
1100
1600
1300
1800
```

Main features:

A. Partition the polygon and report the output as demands illustrated in output example. (due in the midnight of 12/20)