CS60003 Algorithm Design and Analysis, Autumn 2010–11

Class test 2

Maximum marks: 20 | Time: 08-Nov-2010 | Duration: 1 hour

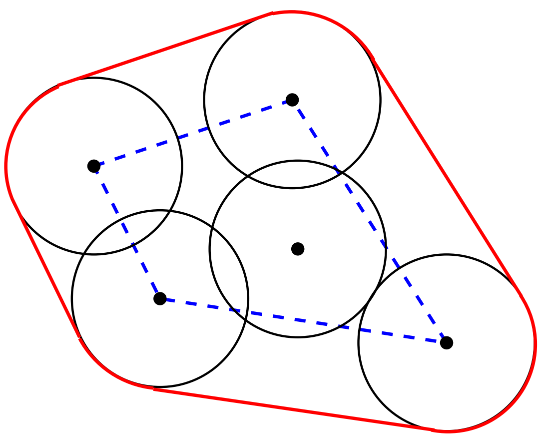
Roll no: ____________________ **Name:** ____

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

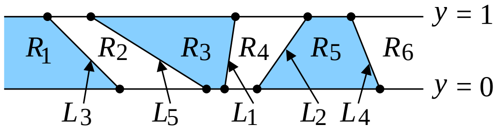
1. [*Flattest Pair*] You are given n points $P_1, P_2, \ldots, P_n$ in general position in the plane. Your task is to find out the pair P_i, P_j (with $i \neq j$) such that the absolute slope of the straight line segment P_iP_j is as small as possible. Propose an $O(n \log n)$-time algorithm to do this. Explain the correctness of your algorithm. **(4)**

Solution The less the absolute slope of P_iP_j (that is, the angle between P_iP_j and the x-axis) is, the more is the angle between P_iP_j and the y-axis, and conversely. That is, P_iP_j is flattest with respect the x-axis (slope) if and only if P_iP_j is steepest with respect to the y-axis. Thus, we follow an analogous algorithm as used for the steepest-pair problem. We sort the input points with respect to their y-coordinates ($O(n \log n)$ time). Subsequently, we consider only consecutive pairs in this sorted list ($O(n)$ time).

2. The following figure shows five circles (of the same radius) in the plane. Draw the (boundary of the) smallest convex *region* enclosing these circles. Also indicate the convex hull of the centers (shown as solid dots) of the five circles. **(4)**



3. You are given n straight line segments $L_1, L_2, \ldots, L_n$ each connecting a point on $y = 0$ to a point on $y = 1$. It is given that these lines are non-intersecting with one another. These segments partition the strip between $y = 0$ and $y = 1$ into $n + 1$ regions $R_1, R_2, \ldots, R_{n+1}$. The following figure illustrates a case of $n = 5$ segments. Each region is specified by the left and the right bounding segments. The first and the last regions are unbounded on one side, denoted by $-$. The six regions (shaded alternately) in the following figure are $R_1 = (-, L_3)$, $R_2 = (L_3, L_5)$, $R_3 = (L_5, L_1)$, $R_4 = (L_1, L_2)$, $R_5 = (L_2, L_4)$ and $R_6 = (L_4, -)$.



(a) Write an $O(n \log n)$-time algorithm to output the regions $R_1, R_2, \ldots, R_{n+1}$ from left to right. **(4)**

Solution Sort the input segments with respect to their *left* endpoints. Let this sorted list be $L_{i_1}, L_{i_2}, \ldots, L_{i_n}$. Output the regions in the following order: $R_1 = (-, L_{i_1})$, $R_2 = (L_{i_1}, L_{i_2})$, $R_3 = (L_{i_2}, L_{i_3})$, $\ldots$, $R_n = (L_{i_{n-1}}, L_{i_n})$, and $R_{n+1} = (L_{i_n}, -)$.

(b) Describe an $O(n \log n)$-time algorithm to convert the sorted output of Part (a) to a binary search tree. **(4)**

Solution The output produced by Part (a) is already sorted. So we should prepare a height-balanced BST (like the AVL tree). Notice that the sorted order also relieves us from making any comparison during insertion and rotation (height balancing) of a new region in the tree.

An alternative is to create the root to store the middle region R_m (where $m = \lfloor (n + 1)/2 \rfloor$ or $\lceil (n + 1)/2 \rceil$), and then recursively build the left subtree on $R_1, R_2, \ldots, R_{m-1}$ and the right subtree on $R_{m+1}, R_{m+2}, \ldots, R_{n+1}$. This will, in fact, take $O(n)$ time.

(c) You are given a point P in the strip between $y = 0$ and $y = 1$, but not on any of the input segments L_i. Describe an $O(\log n)$-time algorithm to identify the region R_k to which P belongs. Should you use the BST of Part (b), explain how the search for P is exactly carried out. **(4)**

Solution We use the standard search routine in the height-balanced BST of Part (b). It only needs to be established how a comparison is made in the search path of the tree with respect to P. Suppose that at some point, a tree-node labeled (L_i, L_j) is visited. Treat each of the input segments as directed from bottom to top. If P lies to the right of L_i and to the left of L_j, then P belongs to the region (L_i, L_j), and the search terminates. Otherwise, if P lies to the left of L_i, the search proceeds to the left subtree of the current node. In the remaining case (P lies to the right of L_j), the search proceeds to the right subtree. Branching decisions (if encountered) for the nodes $(-, L_i)$ and $(L_i, -)$ are analogously handled.