Homework 3[*]

Data Structures
Fall 2019 CS203@IITG

---

(1) (a) Using strong induction, prove that the inorder traversal of any BST T traverses nodes of T in sorted order of their key values.

(b) Devise an algorithm for finding the inorder predecsessor of a given node in the input BST without ever comparing any pair of keys.

(c) Prove the correctness of BST split algorithm presented in class.

(2) As part of the adversary argument for giving a worst-case lower bound on the number of comparisons in finding the maximum among n distinct integers, give a concrete example generated by the adversary when the algo terminates after $r < n - 1$ comparisons for $n = 6$.

(3) Prove the correctness of the adversary argument for giving a worst-case lower bound on the number of comparisons in simultaneously finding the minimum and the maximum among n distinct integers. Especially, show that it is always possible for the adversary to adjust values where necessary in all the cases while being consistent with the earlier comparison outcomes the algorithm learned.

(4) Give a detailed argument for the amortized analysis using accounting method for the insertion algorithm of dynamic array that (only) expands: when the array is full and a new element is to be inserted, double the size of the dynamic array.

(5) (a) Argue that having single NIL node (instead of having more than one NIL node) does not affect the correctness of red-black tree insertion and deletion algorithms.

(b) Design and analyze algorithm for the split operation for red-black tree.

(c) Determine the significance of setting $T.root = black$ in line 16 of the insertion algorithm (page 316 of CLRS) and $x.color = black$ in line 23 (page 326 of CLRS) of delete algorithm for the red-black tree.

(6) (a) Let key x be inserted into an AVL tree T. Supposing that T continues to be an AVL after this insertion, devise an algorithm to update the balance information at the nodes along the path from x to $root$. (Note that this is only a part of the insertion/deletion algorithm for AVL trees.)

(b) Design and analyze algorithm for the join operation for the red-black tree. Further, devise an algorithm for the split operation for the red-black tree.

(7) (a) Analyze the correctness of the algorithm to delete a key in the AVL tree. (Note that this algorithm is posted in the course webpage. Like in the case of proof of correctness for the insertion algorithm, the correctness needs to be argued by associating heights to subtrees before and after insertion.)

(b) Give a counterexample for the following: In the deletion algorithm for the AVL trees, after fixing the height balance condition violation for the first node at a node, say z, balance condition is not violated at any proper ancestor of z.

[*]Prepared by R. Inkulu, Department of Computer Science, IIT Guwahati, India. http://www.iitg.ac.in/rinkulu/