Problem Set 6 (Revision 1)

Data Structures and Algorithms, Fall 2022

Due: October 20, in class.

Problem 1

(a) For the set of $\{1, 3, 5, 10, 16, 18, 22\}$ of keys, draw binary search trees of heights 2, 3, 4, 5, and 6.

(b) Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could *not* be the sequence of nodes examined? You do *not* to justify your answer.

a) 2, 252, 401, 398, 330, 344, 397, 363.
b) 924, 220, 911, 244, 898, 258, 362, 363.
c) 925, 202, 911, 240, 912, 245, 363.
d) 2, 399, 387, 219, 266, 382, 381, 278, 363.
e) 935, 278, 347, 621, 299, 392, 358, 363.

Problem 2

(a) Professor F. Lake believes he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Lake claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \leq b \leq c$ (if there are such a, b, and c). Do you think the claim is true? You need to prove your answer.

(b) For binary search trees, is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? You need to prove your answer.

Problem 3

Suppose that instead of each node x keeping the attribute $x.p$, pointing to x's parent, it keeps $x.succ$, pointing to x's successor. Give pseudocode for SEARCH, INSERT, and DELETE on a binary search tree T using this representation. These procedures should operate in time $O(h)$, where h is the height of T.

Problem 4

(a) Describe a red-black tree that realizes the largest possible ratio of red internal nodes to black internal nodes. What is this ratio? What tree has the smallest possible ratio, and what is the ratio? (You do *not* need to prove the described trees have the largest or the smallest ratio.)

(b) Show the red-black trees that result after successively inserting the keys 99, 80, 77, 11, 31, 2 into an initially empty red-black tree. You need to show the red-black tree after *each* insertion.

Problem 5

Show that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations.

Problem 6

An *AVL tree* (named after inventors Adelson-Velsky and Landis) is a binary search tree that is height balanced: for each node x, the heights of the left and right subtrees of x differ by at most 1. To implement an AVL tree, we maintain an extra attribute in each node: $x.h$ is the height of node x. As for any other binary search tree T, we assume that $T.root$ points to the root node.

(a) Prove that an AVL tree with n nodes has height $O(\log n)$. *(Hint: Prove that an AVL tree of height h has at least F_h nodes, where F_h is the h^{th} Fibonacci number.)*

(b) To insert into an AVL tree, first place a node into the appropriate place in binary search tree order. Now, the tree might no longer be height balanced: the heights of the left and right children of some node might differ by 2. Describe a procedure BALANCE(x), which takes a subtree rooted at x whose left and right children are height balanced and have heights that differ by at most 2, i.e., $|x.right.h - x.left.h| \leq 2$, and alters the subtree rooted at x to be height balanced. *(Hint: Use rotations.)*

(c) Using part (b), describe a recursive procedure AVLINSERT(x, z) that takes a node x within an AVL tree and a newly created node z (whose key has already been filled in), and adds z to the subtree rooted at x, maintaining the property that x is the root of an AVL tree. As in TREEINSERT from the textbook, assume that $z.key$ has already been filled in and that $z.left = NULL$ and $z.right = NULL$; also assume that $z.h = 0$. Thus, to insert the node z into the AVL tree T, we call AVLINSERT($T.root, z$).

(d) Argue that AVLINSERT, run on an n-node AVL tree, takes $O(\lg n)$ time and performs $O(1)$ rotations.