Problem Set 5

Data Structures and Algorithms, Fall 2020

Due: October 22, in class.

Problem 1

(a) 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?

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

(b) Professor Bunyan thinks 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. (That is, $A = (\cup_{b\in B}\{\text{nodes in the left subtree of } b\})\backslash B$ and $C = (\cup_{b\in B}\{\text{nodes in the right subtree of } b\})\backslash B$.) Professor Bunyan claims that any three keys $a \in A$, $b \in B$, and $c \in C$ must satisfy $a \le b \le c$ (if there are such a, b, and c). Do you think the claim is true? You need to justify your answer.

Problem 2

Recall that for each node x in a binary search tree, it keeps a pointer $x.left$ to its left child, a pointer $x.right$ to its right child, and a pointer $x.p$ to its parent. Suppose that instead of each node x keeping the pointer $x.p$, 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 the tree T.

Problem 3

Equal keys pose a problem for the implementation of binary search trees.

(a) Recall the TREEINSERT procedure introduced in Section 12.3 in the textbook CLRS. What is the asymptotic total runtime of TREEINSERT when used to insert n items with identical keys into an initially empty binary search tree?

We propose to improve TREEINSERT by testing before line 5 to determine whether $z.key = x.key$ and by testing before line 11 to determine whether $z.key = y.key$. If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic total runtime of inserting n items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of z and x. Substitute y for x to arrive at the strategies for line 11.)

(b) Keep a boolean flag $x.b$ at node x, and set x to either $x.left$ or $x.right$ based on value of $x.b$, which alternates between FALSE and TRUE each time we visit x while inserting a node with same key as x.

(c) Keep a list of nodes with equal keys at x, and insert z into the list.

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) Argue that an arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using $O(n)$ rotations. *(Hint: First show that at most $n-1$ right rotations suffice to transform the tree into a right-going chain.)*

Problem 5

(a) Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree. Note, you need to show the red-black tree after *each* insertion.

(b) Following part (a), now show the red-black trees that result from the successive deletion of the keys in the order 8, 12, 19, 31, 38, 41. Again, you need to show the red-black tree after *each* deletion.

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.

Problem 7

(a) Assume you are given a size n sorted array $A = (x_1, \cdots, x_n)$. Design an algorithm that builds a random treap containing elements in A in $O(n)$ worst-case time.

(b) Recall that we say an event happens "with high probability (in n)" if it happens with probability at least $1 - 1/n$. Prove the following basic facts about skip lists, where n is the number of keys.

1. The max level is $O(\log n)$ with high probability.
2. The number of nodes is $O(n)$ in expectation.
3. **(Bonus Question)** The number of nodes is $O(n)$ with high probability.

(Hint: You may need to apply concentration inequalities like Chernoff bounds to solve the bonus question. You can learn more about these concentration inequalities from Chapter 4 of the "Probability and Computing" book [original version, translated version].)