Homework 2: Search Trees

Handed out Tue, Oct 13. Deadline: **11:00pm on Mon, Oct 26**. Solutions will be discussed in class the next day, so no submissions will be accepted after 11:00am, Tue, Oct 27. (Please budget your time wisely. The next programming assignment will be probably handed out before this is due.)

Problem 1. (5 points) Consider the AVL tree shown in Fig. 1.

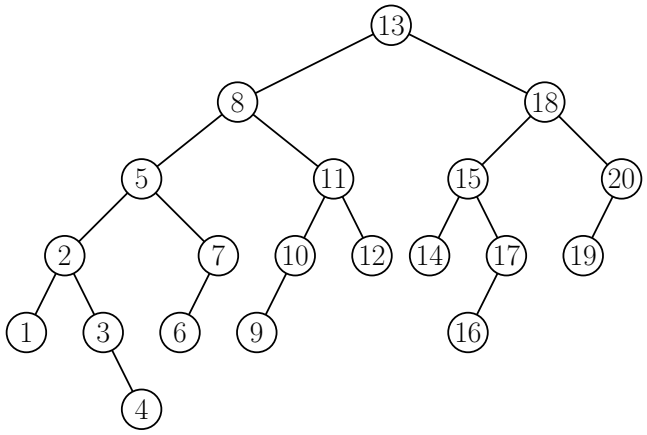

Figure 1: Problem 1: AVL Tree example.

(a) (2 points) Draw the tree again, indicating the balance factors associated with each node.

(b) (3 points) Show the tree that results from the operation `delete(19)`, after all the rebalancing has completed. (We only need the final tree. You can provide intermediate results for partial credit.)

Problem 2. (8 points) Consider the AA trees shown in Fig. 2.

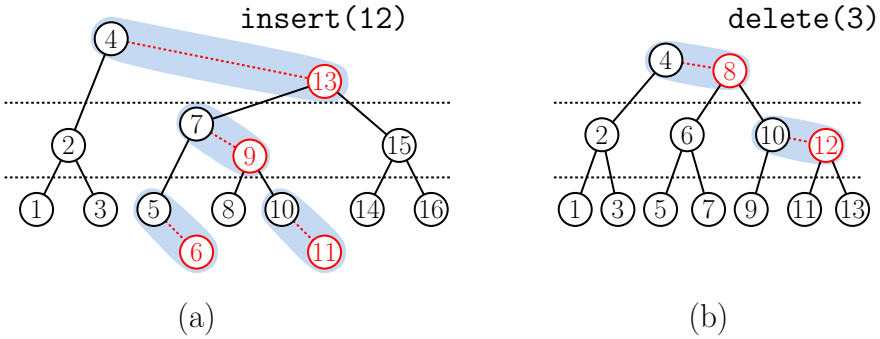

Figure 2: Problem 2. AA Tree example

(a) (4 points) Show the result of performing the operation `insert(12)` into the tree in Fig. 2(a).

(b) (4 points) Show the result of performing the operation `delete(3)` from the tree in Fig. 2(b).

(In both cases, we only need the final tree. You can provide intermediate results for partial credit. Because Gradescope is color-blind, please indicate red nodes using dashed lines, as in the figure.)

Problem 3. (8 points – 2 points each) In class, I said "trust me" that the Zig-Zig and Zig-Zag rotations for splay trees produced the results given in the lecture notes. In this problem, you will verify my assertion by working through the rotations one by one.

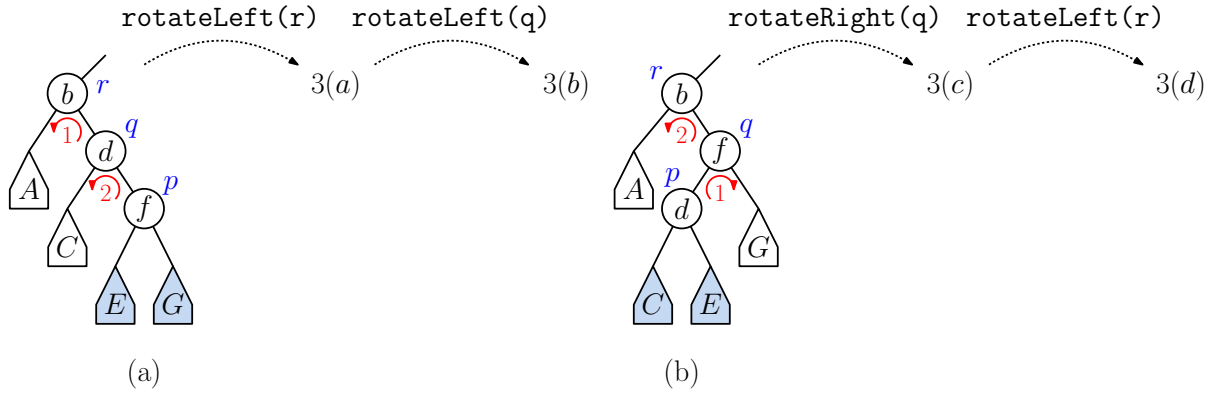

Figure 3: Problem 3: Splay rotations.

(a) Consider the tree shown in Fig. 3(a). Given node `p`, its parent `q`, and its grandparent `r`, show the result after the first rotation of the RR Zig-Zig operation at `p`, namely a left rotation about `r`.

(b) Complete the RR Zig-Zig operation by showing the result after the second rotation, namely a left rotation about `q`.

(c) Consider the tree shown in Fig. 3(b). Given node `p`, its parent `q`, and its grandparent `r`, show the result after the first rotation of the RL Zig-Zag operation at `p`, namely a right rotation about `q`.

(d) Complete the RL Zig-Zag rotation by showing the result after the second rotation, namely a left rotation about `r`.

Problem 4. (12 points) In this problem, we will assume that we have a standard (unbalanced) binary search tree. Each node stores a key, value, left-child link and right-child link. Suppose that we add one additional field, `size`, which indicates the number of nodes in the subtree rooted at the current node.

(a) (2 points) Present pseudo-code for an operation `void rotateRight(Node p)`, which performs a right rotation at node `p`, and also updates the `size` values for all nodes

whose sizes are affected by the rotation. You may assume that `p.left` is not `null` and all the size values are valid prior to the rotation.

(b) (5 points) Present pseudo-code for an operation `int countSmaller(Key x)`, which returns a count of the number of nodes in the entire tree whose key value is strictly smaller than `x`. (Hint: Write a helper function `countSmaller(Key x, Node p)`, which returns a count of the number of nodes that are smaller than `x` within the subtree rooted at `p`.) Briefly explain how your function works.

(c) (5 points) Present pseudo-code for an operation `Value getMinK(int k)`, which returns the value associated with the kth smallest key in the entire tree. If the tree has fewer than k keys, this should return `null`. (Hint: Write a helper function `getMinK(int k, Node p)`, which returns the value of the kth smallest key in the subtree rooted at `p`.) Briefly explain how your function works.

The rotation should run in $O(1)$ time and `getMinK` and `countSmaller` should both run in time proportional to the height of the tree, and neither function should alter the tree's structure.

Problem 5. (5 points) It is easy to see that, if you splay twice on the same key in a splay tree (`splay(x); splay(x)`), the tree's structure does not change as a result of the second call.

Is this true when we alternate between two keys? Let T_0 be an arbitrary splay tree, and let x and y be two keys that appear within T_0. Let:

- T_1 be the result of applying `splay(x); splay(y)` to T_0.
- T_2 be the result of applying `splay(x); splay(y); splay(x); splay(y)` to T_0.

Question: Irrespective of the initial tree T_0 and the choice of x and y, is $T_1 = T_2$? (That is, are the two trees structurally identical?) Either state this as a theorem and prove it or provide a counterexample, by giving the tree T_0 and two keys x and y for which this fails.

Problem 6. (12 points – 4 points each) For this problem, assume that the structure of a node in a skip list is as follows:

```
class SkipNode {
    Key key;           // key
    Value value;       // value
    SkipNode[] next;   // array of next pointers
}
```

The height of a node (that is, the number of levels to which it contributes) is given by the Java operation `p.next.length`.

Often, when dealing with ordered dictionaries, we wish to perform a sequence of searches in sorted order. Suppose that we have two keys, $x < y$, and we have already found the node `p` that contains the key x. In order to find y, it would be wasteful to start the search at the head of the skip list. Instead, we want to start at `p`. Suppose that there are k nodes between x and y in the skip list. We want the expected search time to be $O(\log k)$, not $O(\log n)$.

(a) Present pseudo-code for an algorithm for a function `Value forwardSearch(p, y)`, which starting a node `p` (whose key is smaller than y), finds the node of the skip list containing

key y and returns its value. (For simplicity, you may assume that key y appears within the skip list.) In addition to the pseudo-code, briefly explain how your method works.

Show that the expected number of hops made by your algorithm is $O(\log k)$, where k is the number of nodes between x and y. The proof involves showing two things:

(b) Prove that the maximum level reached is $O(\log k)$ in expectation (over random coin tosses).

(c) Prove that the number of hops per level is $O(1)$ in expectation.

(Hint: The proofs of both follow the same structure as given in the full lecture notes.)

Note: Challenge problems are just for fun. We grade them, but the grade is not used to determine your final grade in the class. Sometimes when assigning cutoffs, I consider whether you attempted some of the challenge problems, and I may "bump-up" a grade that is slightly below a cutoff threshold based on these points. (But there is no formal rule for this.)

Challenge Problem: You are given an extended binary search tree with a root note, `root`. For every node `p`, there is a boolean member, `p.isExternal`, which tells you whether this node is external. Also, every internal node has left and right pointers, `p.left` and `p.right`. Your objective is to return a pointer to *any* external node.

(a) Prove that there is a randomized algorithm that, given a pointer to the root of an extended binary tree T, returns a pointer to any external node of T. This algorithm's expected running time (averaged over all random choices made by the algorithm) must be $O(\log n)$, where n is the number of external nodes in the tree. Note that the algorithm does not know what n is. Prove your algorithm's running time.

(b) Prove that for any deterministic algorithm for the same problem, there exists an extended binary tree T such that the algorithm must visit at least $\Omega(n)$ nodes before finding an external node when run on T. (The tree's structure is allowed to depend on the algorithm.)

Your algorithms can only access nodes of the tree by starting at the root and following left-right links. Neither algorithm knows the value of n.

Submission Instructions: Please submit your assignment as a pdf file through Gradescope. Please see Homework 1 for more detailed instructions.