CMSC 420: Fall 2022

Homework 2: Search Trees

Handed out Tue, Oct 11. Due **Tue, Oct 18, 9:30am** (that is, by the start of class).

Important! Solutions will be discussed in class right after the due date, so **no late submissions will be accepted**. Turn in whatever you have completed by the due date.

Problem 1. (10 points) Consider the AVL trees shown in Fig. 1.

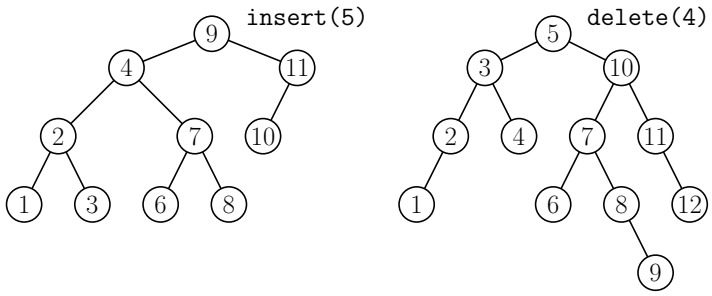

Figure 1: AVL-tree operations.

(a) (5 points) Show the result of executing the operation `insert(5)` to the tree on the left.

(b) (5 points) Show the result of executing the operation `delete(4)` to the tree on the right.

In each case, show the final tree and list (in order) all the rebalancing operations performed (e.g., "`rotateLeftRight(7)`"). Intermediate results may be shown for the sake of assigning partial credit. Draw the final tree as in Fig. 1(b) from Lecture Lecture 7. Show the balance factors at each node. (Don't bother to give the heights.)

Problem 2. (10 points) Consider the AA trees shown in Fig. 2.

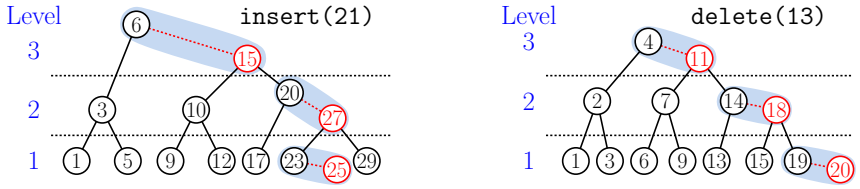

Figure 2: AA-tree operations.

(a) (5 points) Show the result of executing the operation `insert(21)` to the tree on the left.

(b) (5 points) Show the result of executing the operation `delete(13)` to the tree on the right.

In each case, show the final tree and list (in order) all the rebalancing operations (skew, split, and update-level) that result in changes to the tree (e.g., "`skew(13)`"). Intermediate results may be shown for the sake of partial credit.

Draw the tree as in Figs. 6 and 7 from Lecture 9. Indicate both the levels and distinguish red from black nodes. You do not need to color the nodes—a dashed line coming in from the parent indicates that a node is red. (Do not bother drawing `nil`.)

Problem 3. (10 points) Recall from Lecture 9 the classical red-black tree (not the AA tree) effectively models a generalization of the 2-3 tree, called the *2-3-4-tree*. Recall that this is a straightforward generalization of 2-3 trees, but nodes may have 2, 3, or 4 children (and 1, 2, or 3 keys, respectively).

(a) (5 points) Fig. 3 shows a (classical) red-black tree. Draw the associated 2-3-4 tree. (Draw your tree using the same format as in Fig. 1(c) from Lecture 8.)

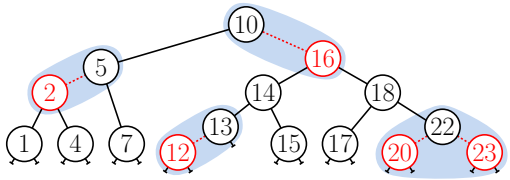

Figure 3: Converting a red-black tree to a 2-3-4 tree.

(b) (5 points) Explain what sequence of AA rebalancing operations (skews and splits) would be needed to convert this tree into an AA tree (e.g., "`skew(13), split(18), ...`"). Draw the final AA tree.

Problem 4. (10 points) Consider the splay trees shown in Fig. 4. In both cases, apply the exact algorithms described in the Lecture 12 notes.

(a) (5 points) Show the steps involved in operation `insert(9)` for the tree in Fig. 4 (left).

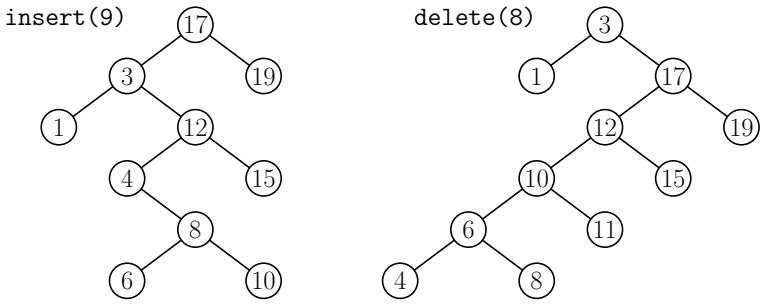

Figure 4: Splay-tree operations.

(b) (5 points) Show the steps involved in operation `delete(8)` for the tree in Fig. 4 (right).

In both cases, indicate what splays are performed and what additional alterations are performed beyond splaying. (When doing splaying, we only need to see the tree after splaying is

complete, but intermediate results may be shown for partial credit). Also show the final tree after the insert/delete operation is finished.

Problem 5. (10 points) Can you determine the structure of a binary tree based solely on a preorder enumeration of its nodes? In general, the answer is no, since there can be multiple trees that have the same preorder listing of nodes. In this problem, we will see that there are instances where this is possible.

Recall that a binary tree is said to be *full* if every node has either two (non-null) children or no children at all (both are null). An example is shown in Fig. 5.

Suppose that you have a full tree with n nodes, which have been enumerated according to a preorder traversal and stored in an array called `preorder[n]`. Suppose as well that you have a parallel boolean array `isLeaf[n]`, which tells you which nodes are leaves and which are not. In particular, `isLeaf[i]` is true if `preorder[i]` is a leaf node and false otherwise.

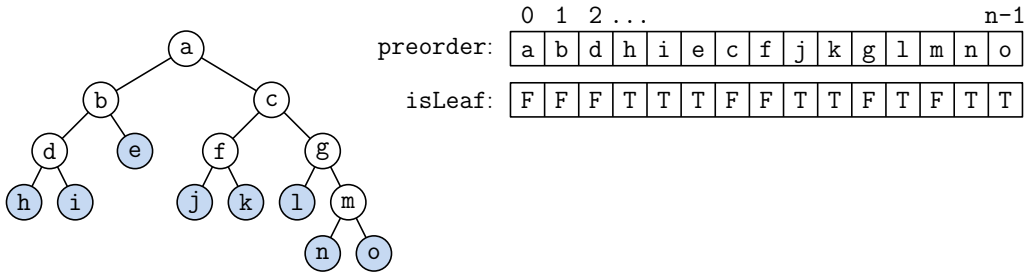

Figure 5: A full binary tree and the arrays `preorder` and `isLeaf`.

(a) (5 points) Suppose that the arrays `preorder` and `isLeaf` have been generated from some full binary tree T. Prove that T is uniquely determined from the contents of these two arrays. (Hint: Use strong induction on the number of nodes in the tree.)

(b) (5 points) Present pseudocode for a function, which given two valid arrays, `preorder` and `isLeaf`, constructs the unique tree that they represent. You may assume that the arrays are valid in the sense that they define an actual full binary tree.

Hint: Write a recursive function `buildTree(int i)` which is given an index `i`, constructs the subtree rooted at node `preorder[i]`, and returns a reference to the root of this subtree. It may be helpful to assume that `i` is passed in as a reference parameter, and as the procedure runs, it advances `i` to the first node following the generated subtree. What is the initial call to this function?

Challenge Problem: Recall that each node of an AA tree stores a key, its level, and pointer to its left and right children. (For this problem, we will ignore the node values. Also, rather than using the sentinel node nil, let's assume that we just use standard null pointers in the leaves.)

Suppose that you have been given a valid AA tree, but all the level information is missing! Knowing that the tree is a valid AA tree, is it possible to reconstruct the level information for

the tree? Take a position and justify it. If you believe that it is not possible, give an example of a binary tree which can be interpreted as two different valid AA trees (where at least one node has a different level number in the two trees). If you believe it is possible, present a proof that the assignment of level numbers is unique. (Your proof might take the form of a procedure that reconstructs the level information for the AA tree.)

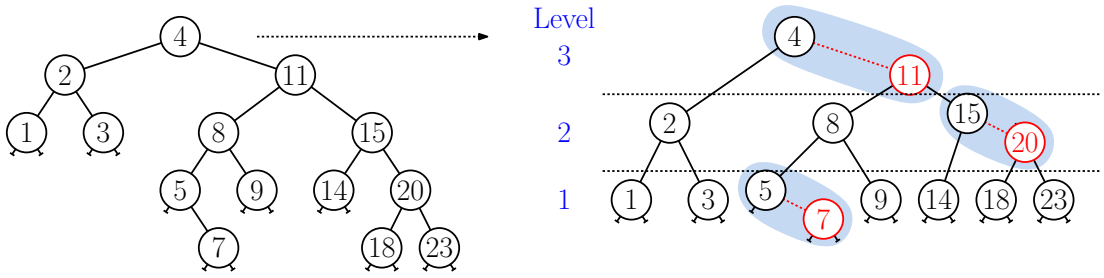

Figure 6: Can you recover the level numbers?