Week 3 Quiz

TOTAL POINTS 10

1. Create a binary search tree by inserting the following five values one at a time: 1 point

   4 6 5 7 8

   What is the height of this tree?

   (Recall how we calculate the height of a tree or subtree: The height of a leaf node by itself is 0. The height of a non-existent tree is -1. Otherwise, the height of a tree is the longest path length from the root of that tree to any one of its leaves.)

   Enter answer here

2. Create a binary search tree by inserting the following eight values one at a time: 1 point

   3 1 2 4 6 5 7 8

   What is the balance factor of the root node of this tree? (For this question, do not perform any rotations on this tree as you insert the items. It's just a binary search tree, not necessarily a balanced BST such as an AVL tree.)

   Enter answer here

3. For the binary search tree created by inserting these items in this order: **4 3 5 1 2**, which node among 1 through 5 is the deepest node with a balance factor of magnitude two or greater? (For this question, do not perform any balancing rotations as you insert these items.) 1 point

   Enter answer here

4. Consider the binary search tree that you constructed in the previous question. If we interpret it now as an AVL tree, it has an imbalance that can be fixed with a rotation. (Remember that we focus on the deepest point of imbalance, where the magnitude of the balance factor is 2 or greater, to perform the rotation.) 1 point

   After performing the correct balancing rotation about the node that we identified in the previous question, the resulting tree is identical to which one of the following binary search trees? (We'll describe these other trees by listing the order in which you would insert items to create the trees directly.)

   - Inserting **2 1 4 3 5** one node at a time.
   - Inserting **3 2 4 1 5** one node at a time.
   - Inserting **3 5 2 4 1** one node at a time. FOO
   - Inserting **4 2 5 1 3** one node at a time.

5. The code that ensures the balance of an AVL tree after node insertion or removal only checks if the height balance factor is +2 or -2. What happens if the height balance factor of a node in an AVL tree after node insertion or removal is greater that +2 or less than -2? 1 point

   - When insertion and removal create a node whose height balance factor is greater than +2 or less than -2, that node always has a descendant with a height balance factor equal to +2 or -2 and when all of its descendant nodes are resolved, then its height balance factor will be no greater than +2 or no less than -2.
   - An AVL tree never has a node with a height balance factor greater than +2 or less than -2, even after a node insertion or removal.

- We ignore nodes in an AVL tree with height balance factor greater than +2 or less than -2 because they are statistically rare and are unstable, such that they are removed as soon as any tree balancing rotation occurs.
- There is additional code not shown that handles the cases when the height balance factor is greater than +2 or less than -2.

6. If, after inserting a new node into an AVL tree, you now have a node with a height balance factor of -2 with a child with a height balance factor of -1, which of the following operations should be performed?

   1 point

   - Left-Right Rotation
   - Right Rotation
   - Right-Left Rotation
   - Left Rotation

7. If, after inserting a new node into an AVL tree, you now have a node with a height balance factor of -2 with a child with a height balance factor of +1, which of the following operations should be performed?

   1 point

   - Left-Right Rotation
   - Right Rotation
   - Left Rotation
   - Right-Left Rotation

8. Which one of the following is NOT a valid reason to choose the B-Tree representation over a standard AVL binary search tree?

   1 point

   - B-Trees work faster in networked cloud environments than do AVL trees.
   - B-Trees run faster on large data sets than do AVL trees.
   - B-Trees have better algorithmic "Big-O" run-time complexity for the find operation.
   - B-Trees require fewer block read accesses for tree operations.

9. Which of the following statements is NOT true for a B-Tree of order m?

   1 point

   - Any node that is not the root or a leaf holds at least half of the total number of keys allowed in a node.
   - Each node can hold an ordered list of as many as m keys.
   - All leaf nodes are at the same level of the B-Tree.
   - Each node can have at most one more child than key.

10. If a B-Tree is completely filled, meaning every node holds its maximum number of keys and all non-leaf nodes has the maximum number of children, then what happens when an additional key is inserted into the B-Tree?

    1 point

    - A new leaf node is simply added to the B-Tree.
    - After searching for the leaf node where the new key should go, the leaf is split in half as two separate leaf nodes, and then the middle value is thrown up to the layer above as an inserted key, and this insertion and rebalancing repeats until a new root key rises to the top, which adds a layer to the tree.
    - A new node containing the new key is added above the previous root and becomes the new root. The new root will have one pointer leading to the old root node.
    - Every leaf node in the entire B-Tree becomes parent to a new leaf node, but all but one of these leaf nodes are "blank" placeholder nodes that contain zero key values.

Learn more about Coursera's Honor Code

Save Submit