Written Exercise for HW6 BST

Name: Rong Huang

Date:03/12/2024

1. What does it mean if a binary search tree is a balanced tree?

A binary search tree (BST) is considered balanced if, for every node, the height difference between its left and right subtrees is at most one. The main goal of keeping a tree balanced is to ensure that operations like search, insertion, and deletion can be performed as efficiently as possible. In a balanced tree, these operations can be performed in logarithmic time, making the process much quicker compared to an unbalanced tree where these operations could degrade to linear time in the worst case.

2. What is the big-Oh search time for a balanced binary tree? Give a logical argument for your response. You may assume that the binary tree is perfectly balanced and full.

The big-Oh search time for a balanced binary tree is O(log n), where n is the number of nodes in the tree. The logic behind this is fairly straightforward: in a balanced binary tree, each time you move from a node to one of its children (either left or right), you effectively halve the number of potential nodes to search through. This is because, at each step, you choose one subtree and ignore the other, cutting down the search space by half. Since dividing the search space in half repeatedly is the essence of logarithmic complexity, the search operation in a balanced binary tree takes O(log n) time. This makes searching in a balanced binary tree much faster compared to linear search algorithms, especially as the number of nodes grows large.

For instance, in a perfectly balanced BST with 15 nodes, a search might proceed as follows: starting at the root (1 comparison), then moving to one of its children (2nd comparison), and so on, until the desired node is found or not. In this case, the maximum number of comparisons would not exceed 4 (since log2(15) is approximately 3.9), demonstrating how each step effectively halves the search space, leading to logarithmic search time.

3. Now think about a binary search tree in general, and that it is basically a linked list with up to two paths from each node. Could a binary tree ever exhibit an O(n) worst-case search time? Explain why or why not. It may be helpful to think of an example of operations that could exhibit worst-case behavior if you believe it is so.

Yes, a binary search tree could indeed exhibit an O(n) worst-case search time. This happens in an unbalanced BST, particularly when the tree becomes skewed. A skewed tree can happen in two ways: either all nodes are inserted in ascending or descending order, which makes the tree take the form of a straight line (like a linked list). In such cases, every insertion just adds another node to the end of this "line," making the tree's height equal to the number of nodes, n.

When searching for a value in this skewed BST, in the worst case, we might need to traverse from the root to the leaf node at the very end of this "line." This means we would have to go through every node, making the search operation take O(n) time since we're effectively performing a linear search. This is why balancing a BST is crucial for maintaining efficient search times, as it ensures the tree's height is kept to a minimum, allowing operations to stay within logarithmic time complexity.