Aditya Lamba

PROG20799

Professor Tiffany Antopolski

August 13, 2024

Assignment 4

A screenshot of a computer

Description automatically generated

Functions and their complexities – Binary Search Tree (BST):

1. createNode() – O(1): This is because this specific function does not perform a calculation, it just makes a single node entry each time.
2. insertNode() – O(log(n)): looking at the average insertion complexity on the chart the BST insert function will be this complex.
3. findMin() – O(log(n)): this function uses a search down the left subtrees, on average it will have this complexity for a search in a BST.
4. deleteNode() – O(log(n)): The average delete function for a BST will take O(log(n)) time according to the chart.
5. inOrderTraversal() – O(n): This function has to go through every node in the tree so it will take O(n) time, since it will be closer to the worst in terms of complexity.
6. preOrderTraversal() – O(n): This function has to go through every node in the tree so it will take O(n) time, since it will be closer to the worst in terms of complexity.
7. postOrderTraversal() – O(n): This function has to go through every node in the tree so it will take O(n) time, since it will be closer to the worst in terms of complexity.
8. calculateHeight() – O(log(n)): This function also requires traversing to the ends of the tree until a null value is reached for both left and right subtrees, a complexity of O(n) could be argued.
9. nodeCount() – O(n): This function requires going through every node in the tree, it has worst time complexity.
10. search() – O(log(n)): On average, it will take O(log(n)) time to find a searched value, worst case scenario could be O(n).
11. minimum() – O(log(n)): This function finds the value by traversing the left subtree (only) until the bottom, and will on average take O(log(n)) time.
12. Maximum() – O(log(n)): This function finds the value by traversing the right subtree (only) until the bottom, and will on average take O(log(n)) time.

Approach:

This assignment was very difficult as it is one thing to understand the logic that is required behind coding recursively, but another to comprehend what recursive logic is. Many times, I found myself thinking I understood what the code is doing, but the more I researched into it the more I learned that my thinking was off.

I found the fact that things were not explicitly coded to be of concern. Such as when a node with one child is deleted and its child is returned, only then to automatically be connected to the parent node of the node that had been deleted. Coming from working with double linked lists (and single) there were not many instances (if any) where connections could be made by default, they always involved explicitly coding where the pointers of specific nodes will be pointing.

Similarly, I had trouble when considering the logic behind the deleteNode() when there is a node with 2 children and nodeCount().

I spent a lot of time thinking about some functions and decided it would be best to just code something that may be close or pseudo and work with trial and error. Doing research to find out how logic would work, or why it wouldn’t in certain instances helped a lot too.

As difficult as the assignment was, I did find it rewarding when certain realizations dawned on me as to what the recursive process was and how the logic would be implemented in the code.