Joel Woeste

DSA

10/01/2024

Insert

The insert method has a worst-case time complexity of O(log n). This is because the AVL tree continuously auto balances, ensuring the height of the tree is proportional on both sides aka the logarithm of the nodes. After inserting a node, the balancing operations (rotateLeft and rotateRight) are called, and each of these rotations takes only O(1) time.

Remove

The remove method has a worst-case time complexity of O(log n). Similar to insert, the removal process involves traversing the tree to find the node to remove, which takes O(log n) time. The balancing operations afterward also contributes the same time complexity.

Search by Name

The searchName method performs an in-order traversal of the entire tree using a stack for iteration. This results in a time complexity of O(n), where n is the number of nodes in the tree. Since every node needs to be checked in the worst case, it takes linear time. It was made non-recursively but has the exact same time complexity to if it was iterated recursively.

Search by ID

The searchID method also performs an in-order traversal using a stack. Almost the same as searchName, it has a worst-case time complexity of O(n), as it may need to traverse the entire tree to find the ID. It was made non-recursively but has the exact same time complexity to if it was iterated recursively.

In-Order Traversal

The printInOrder method recursively visits each node in the tree, making its time complexity O(n) since each node is visited exactly once.

Pre-Order Traversal

The printPreOrder method works like the printInOrder method, visiting each node exactly once. Therefore, its time complexity is O(n).

Post-Order Traversal

The printPostOrder method follows the same pattern as the previous traversal methods, leading to a time complexity of O(n).

Print Level Count

The printLevelCount method uses a queue to traverse the tree level by level. Since every node in the tree must be visited, its time complexity is O(n).

Remove In-Order

The removeInorder method has a time complexity of O(n), as it first performs an in-order traversal to find the Nth node, which takes O(n) time. Afterward, it calls the remove method, which takes O(log n) time. The overall complexity is dominated by the traversal, resulting in a time complexity of O(n).

What I learned and What I would do Differently:

I learned a lot about trees and also a lot about recursive functions, I had never really understood recursive functions until this project, and being able to use them has made me realize how much better my older programs could have been. Also I got a lot of review for pointers and nodes which was good, and I think overall the project allowed me to get a better grasp of a lot of the CS concepts I missed or had a loose knowledge of.

If I were to start over, I would likely start making the functions for remove first, as these were the most complex and that would also force me to work on the balance functions to start as well. Additionally I probably wouldn't make my search functions the way I did this time, as out of curiosity I made them non recursive and it ended up taking way longer than it would have if I just made them recursive. Also I would probably separate AVL class into a separate .cpp and instead call it in main, instead of what I had to do which was make another copy of just the code from AVL and place it into a class separate of main but still similar.