Gator AVL Documentation

Worst Case Complexities

Insert NAME ID

- O (log n), where n is the number of nodes in the tree. The function recursively calls itself and the rotation operations are constant.

Remove ID

- O (log n), where n is the number of nodes in the tree. The tree is balanced, and the function recursively calls itself.

Search ID

- O (log n), where n is the number of nodes in the tree because the tree is balanced and recursively calls itself.

Search NAME

- O (n), where n is the number of nodes. This function calls another function which preorder traverses the tree and push all nodes into the vector. Then this function loops through all nodes in the vector and select nodes with desired name, pushing them into a vector. Finally, it iterates through the vector to print their IDs, but this is much smaller than n and can be ignored. So, it is O(n+n), which is O(n).

printInorder

- O (n), where n is the number of nodes. This function calls another function which inorder traverses the tree and push all nodes into the vector. Then it iterates thourgh the vector to print names. O(n+n) is O(n).

printPreorder

- O (n), where n is the number of nodes. This function calls another function which preorder traverses the tree and push all nodes into the vector. Then it iterates thourgh the vector to print names. O(n+n) is O(n).

printPostorder

- O (n), where n is the number of nodes. This function calls another function which postorder traverses the tree and push all nodes into the vector. Then it iterates thourgh the vector to print names. O(n+n) is O(n).

printLevelCount

- O (log n), where n is the number of nodes. This function recursively goes to the bottom and count number of nodes in the longest path.

removeInorder N

- O (n), where n is the number of nodes. The function first calls another function that inorder traverses all nodes and find Nth node in the vector, which is O (n). After knowing the ID of Nth node, it calls the remove function, which is O (log n). O (n) + O (log n) is still O (n).

Things I learned

- I am more familiar with recursion functions and how they link together.
- I learned how to come up with test cases by myself and considering edge cases, like when the tree is empty, we cannot remove nodes.
- I learned to sketch the problem and writing pseudocode before implementing it in my IDE.
- I learned to debug my functions with unit testing using the Catch2 framework inside the Clion.
- I secured significant pointers and avoided modifying them unconsciously by making them private.
- I learned to stay calm when there are strange bugs. Take some rest and refresh the brain instead of sticking to it painfully.
- When working with trees, make sure I managed all data attributes to avoid stupid mistakes.

Things I would do differently if start over

- I would think of edge cases before writing the general case. Adding many code snippets later easily interrupts my logic.
- I would consider if there were more efficient ways for search Name. Although mine also works, I think it could be better.
- I would try to remove some code that are potentially redundant, making my program more efficient.
- I would like to experiment different techniques to solve the problem, like finding another way to remove a node.
- I may want to utilize some dynamic memeory.