Yash Hegde

Professor Kapoor

Data Structures & Algorithms

18 June 2021

P1 - Documentation

1. **Q: What did you learn from this assignment and what would you do differently if you had to start over?**

A: In regards to this assignment, I learned that there are many unusual ways that C++ differs from other programming languages, which affects the thought process of solving certain situations. For example, comparing two strings similarly to integers through lexicographical juxtaposition was a concept that was lightly touched on in the past, but practiced extensively in this project. Additionally, I found that there are many different ways to solve a problem (such as with rotating in a BST), which could vary greatly in the amount of time spent in order to complete them (interestingly, this is a concept that this class relies heavily on through algorithmic analysis). I found that approaching the creation of the AVL tree through building off of the unbalanced BST involved repeatedly adding on top of the node’s children to access its parents, which allowed for getting the height of both sides of the parent in order to determine what rotation was necessary. If I had to start over, I would make sure that I would start much earlier in the planning process of how to approach dealing with adding functionality to an unbalanced BST, as it was a concept that I had initially struggled with. Alternatively, I would make sure to consistently write down what and when certain test cases would not work, in order to seek out a general pattern in faulty output.

1. **Q: What is the computational complexity of each method in your implementation? Reflect for each scenario: Best, Worst and Average.**

A:

**Note: Conditions of id checks if specified id is less than or greater than certain id to know where to traverse next in the AVL.**

*Insert Method -*

*Worst Case:* O(log n), where n is the height of the tree, as the recursive calls only go through those nodes that satisfy the conditions of the inserted id, till the length of the tree if the node is inserted as a leaf node.

*Average Case:* O(log n), where n is the height of the tree, as the recursive calls only go through those nodes that satisfy the conditions of the inserted id, till the specified node can be inserted or node with a duplicate id is reached at some arbitrary point in the tree.

*Best Case:* O(1), as if the value that is trying to be inserted is a duplicate id, and is located at the root of the tree, the number of nodes in the tree would not affect the time complexity. (Subsequent rotations are O(1), not affecting time complexity)

*Remove Method -*

*Worst Case:* O(log n), where n is the height of the tree, as the recursive calls only go through those nodes that satisfy the conditions of the inserted id, till the length of the tree if the node that needs to be deleted is a leaf node, or if the node cannot be found.

*Average Case:* O(log n), where n is the height of the tree, as the recursive calls only go through those nodes that satisfy the conditions of the inserted id, till the specified node can be deleted at some arbitrary point in the tree.

*Best Case:* O(1), as if the value that is being deleted is at the root of the tree, the number of other nodes would not affect the time complexity, and subsequent rotations needed would take place in O(1), not affecting the time complexity either.

*Searching (Input of Name to Output of ID & Vice Versa) -*

*Worst Case:* O(log n), where n is the height of the tree, as the recursive calls only go through those nodes that satisfy the conditions of the inserted id, till the length of the tree if the node found is a leaf node, or if the node cannot be found.

*Average Case:* O(log n), where n is the height of the tree, as the recursive calls only go through those nodes that satisfy the conditions of the inserted id, till the specified node is found at some arbitrary point in the tree.

*Best Case:* O(1), as if the root contains the id, or if it is nullptr, the search would be successful immediately, and the number of nodes in the tree would not affect the time complexity.

*Remove Inorder N & Final -*

*Worst Case:* O(N+n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, including when the id number N is out of range. The variable N, which is the index at which the node is to be removed and is traversed to in the queue, is used for deletion.

*Average Case:* O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, and the index N being in range.

Best Case: O(n), as the function still needs to traverse the entire tree to know how many nodes there are, via the recursive methods involved. If N is 0, this part would be O(1) complexity, but would not affect the O(n) complexity earlier.

*InOrder, PreOrder, & PostOrder -*

Worst Case: O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, printing them out.

Average Case: O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, printing them out.

Best Case: O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, printing them out. (A possible contingency would be if the root is nullptr, but is not something we can assume)

*LevelCount - (Doesn’t rely on height of the tree, as explained in the following)*

*Worst Case:* O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, storing them in a queue to be accessed and traversed (also n). Within the queue loop that loops 0 to n-1 times, a count based on checks is done to know where a level ends and begins.

*Average Case:* O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, storing them in a queue to be accessed and traversed (also n). Within the queue loop that loops 0 to n-1 times, a count based on checks is done to know where a level ends and begins.

*Best Case:* O(n), where n is the number of nodes in the tree, as the recursive methods within traverse through all the nodes continually, storing them in a queue to be accessed and traversed (also n). Within the queue loop that loops 0 to n-1 times, a count based on checks is done to know where a level ends and begins. (A possible contingency would be if the root is nullptr, but is not something we can assume)