# Introduction

The goal of this assignment is to finalize an implementation of an AVL tree. An AVL tree is a data structure that is unique in that it auto-balances after every insertion or deletion of data, keeping data searches at a time and space cost of O(log n). There are several operations that we are to have the program perform, while also printing confirmations of those operations to the console. The code was written by our CS instructor and given to us in a mostly functional state; however, it did not successfully perform all of the intended operations. The functional requirements of this AVL tree implementation are laid out in the following table:

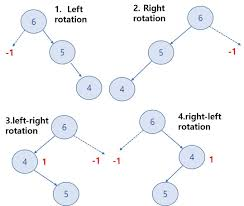
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** |  | **Functional Requirement** |  | **Value** | **Stakeholder** |
| FID001 | **I want to** | create an AVL tree | **so that** | we have a data structure to work with | student |
| FID002 | **I want to** | be able to insert data | **so that** | I can add data to the tree | student |
| FID003 | **I want to** | be able to delete data | **so that** | I can remove data from the tree | student |
| FID004 | **I want to** | have the tree balance itself | **so that** | it remains a balanced AVL tree after each insertion or deletion | student |
| FID005 | **I want to** | fill the AVL tree with 10 randomized names | **so that** | I can manipulate the names | student |
| FID006 | **I want to** | print verification for each inserted entry | **so that** | the insertion method is verified to be functional | instructor |
| FID007 | **I want to** | print a list of keys in the tree with their values | **so that** | we can see what data was inserted | instructor |
| FID008 | **I want to** | attempt to insert a duplicate key | **so that** | we can see that a duplicate cannot be inserted | instructor |
| FID009 | **I want to** | delete 1st key and print verification | **so that** | we can see that the delete function works | instructor |
| FID010 | **I want to** | print the tree data again | **so that** | we can verify that the 1st key was indeed deleted | instructor |
| FID011 | **I want to** | print the balance factors of each node | **so that** | we can confirm that the tree is balanced as the instructor intended | instructor |

As you can see most of the requirements involve printing confirmation to the console that the tree implementation is indeed working. I will attempt to explain what I was able to accomplish, and also what eluded me.

# Analysis

The first requirement, the AVL tree itself, is a sum of many parts. An AVLNode class serves as the representation of each node of the tree. The AVLTree class is made up of methods to manipulate those nodes in the unique way that an AVL tree must. The **insert()** method calls **insertNode()** to handle the fine points of inserting data into the tree. The **insertNode()** method [FID002] first checks to see if the initial root is null, in which case is will insert the new node there. It then checks the new node’s key against existing nodes in the tree, and blocks insertion of a duplicate key [FID008]. Next, it uses a *while* loop to traverse down the tree, moving left if the new node’s key is less than the key it is being compared to, or right if it is greater than that key, until it reaches a null node and inserts the new key and data. The **rebalance()** method is then called to rebalance the tree, which I’ll discuss shortly. Upon successful insertion, **insert()** confirms by printing to the console with **System.out.println()** that the insertion has been completed [FID006]

In order to delete a node, the **delete()** and **deleteNode()** methods are employed [FID003]. The **delete()** method traverses the tree in a similar fashion to the **insertNode()** method, finds the node to be deleted, and then calls **deleteNode()** to do the dirty work. It then calls the **rebalance()** method to return the tree to a balanced state, after which **delete()** prints a confirmation of the deletion to the console using **System.out.println()** [FID009].



The **rebalance()** method is at the heart of the AVLTree class [FID004]. It determines if and how the tree is unbalanced, and uses different methods to perform the appropriate rotations to achieve balance again. To be considered balanced, an AVL tree requires that the height each side of the tree and subtrees must be within 1 balance factor of each other. If they differ by more than that, the program must perform one of four rotation methods (shown to the left) around the root of the unbalanced subtree in order to regain balance.

Several methods to print the tree in various orders are included: **preOrder(), inOrder(),** and **postOrder()**. Each of these public methods calls a separate private method that uses **System.out.print()** to print the data in the correct order [FID007][FID010]. There is also a method called **balance()** that prints the balance factor of each node[FID011], which is not currently functioning properly.

# Conclusion

This program is put together well and employs several solid methods for implementing a functional AVL. It does, however, still have some bugs. Although it performs AVL operations correctly, and prints the desired information to the console, it also prints several error messages that need to be rectified. Further development is required.

After many hours of poring over the code, comparing it to the revised code that you wrote and a few examples online, I have a much better grasp of what it does and how it all works together. Once the pieces started to fall into place in my mind, though, it was too late to work out all the bugs. By putting a *try/catch* around the *for* loop in **setBalance(),** I was somehow able to print all of the necessary information that wasn’t printing before, but there were lots of lines of error messages between each line of intended output. I also wasn’t able to get the balance factor to print the way that you asked. I would like to work on it more in the future and see if I can get it figured out.

<https://github.com/cellson7170/AVL-Tree-Fix>