Binary Search Tree pt. 2 – The AVL Tree

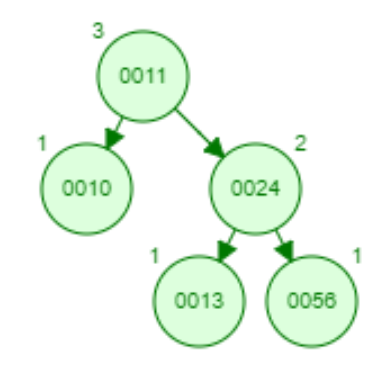
A binary search tree (BST) is great and all, but it’s prone to some serious worst-case scenarios. For instance, what if I add numbers in a sorted order? What if I add 100 instances of the same value? A simple BST will quickly lose its log2n runtime advantage. However, what if the tree was guaranteed to maintain its average runtime? We can do this by changing our simple BST to an AVL Tree.

Side Note: AVL is the initialism representing the last names of the TWO men who discovered the data structure behavior. That’s right; one man has two last names (they’re hyphenated).

# Requirements

Once again, we will start with your existing AlgoDataStructures library. To it, add the following features:

1. Create a new class, AVLTree. You may choose to create an independent class or simply subclass your existing BinarySearchTree class. Add the new class to the same namespace/package as your previous data structures.
2. Nodes now have balance factors. The balance factor of any node is the difference in height between its left and right child (or rather, Balance Factor = Left Child Height – Right Child Height). You may choose to subclass your original Node or simply refactor it. It’s internal to the tree, so you’re safe either way.
3. The tree can now “rotate” nodes in any one of four ways:
   1. Single Right Rotation
   2. Single Left Rotation
   3. Right Left Rotation
   4. Left Right Rotation
4. The tree re-balances as needed after EVERY add or remove operation. Balancing takes place at the first node closest to the leaves where the node’s balance factor has an absolute value greater than 1.
5. Additionally, change the tree’s ToArray function to return a Breadth-First representation of the tree. In other words, if I add 24, 10, 11, 56, 13, the Breadth-First array would contain values ordered as 11, 10, 24, 13, 56 (where 11 is at index 0). The array should have NO empty spaces, unless a node in the tree holds the value null.



1. All other features and functions of the original BST should function as before.

# Rubric

**Automatic Zero:** Your data structure is not a binary search tree, your final product is not a library (JAR or DLL), or your library cannot be tested.

(10 points) ToArray now returns a correct breadth-first array representation of the tree’s values

(40 points) The node balance-factors and tree rotations work correctly