UNIVERSITY OF VIRGINIA, DEPARTMENT OF COMPUTER SCIENCE

AVL - Tree Implementations

Nada Basit and Mark Floryan

February 22, 2022

1 SUMMARY

For this homework, you will be extending your Binary Search Tree to include tree rotations, and self-balancing. You will implement a few methods to complete this AVL Tree.

1. Grab your working code from the *Binary Search Trees* assignment (the project / starter code is the same).

2. Implement the missing methods in the AVLTree class (some of this is done for you to simplify the assignment)

3. Use the provided tester files to verify your implementation works. Note that you should test your code more so than the provided tester does this time. The tester is NOT as thorough as in previous homeworks.

4. **FILES TO DOWNLOAD:** *None, but use your code from the previous assignment for this one.*

5. **FILES TO SUBMIT:** BinaryTree.java (from last week), BinarySearchTree.java (from last week), AVLTree.java (new)

1.1 AVLTREE.JAVA

You will implement an AVL tree that inherits from your binary search tree. An AVL tree can take advantage of the insert and remove methods from the class it inherits from (i.e., BinarySearchTree.java). Thus, to insert into an avl tree, you can call super.insert() and then simply check if the current node needs to be balanced. Some of this implementation is provided for you, but you will have to implement the following methods yourself:

```
public class AVLTree<T extends Comparable<T>>
                    extends BinarySearchTree<T>{

    //Insert and remove
    protected TreeNode<T> insert(T data, TreeNode<T> curNode);
    protected TreeNode<T> remove(T data, TreeNode<T> curNode);


    //figures out whether a double or single rotation is
    //needed and in which direction(s)
    private TreeNode<T> balance(TreeNode<T> curNode);

    //rotate right on the curNode provided
    private TreeNode<T> rotateRight(TreeNode<T> curNode);

    //rotate left on the curNode provided
    private TreeNode<T> rotateLeft(TreeNode<T> curNode);

    //compute the balance factor of the given node
    private int balanceFactor(TreeNode<T> node);

}
```

1.2 TESTING YOUR CODE

Once you are done, you can look at the two provided tester files to check your implementation. As stated earlier, these files do not rigorously check your implementations, so you should be writing your own test cases in addition to the few provided.

You should submit **three files** for this homework: **BinaryTree.java**, **BinarySearchTree.java**, and **AVLTree.java**. The former two are resubmissions of your work from last week.

1.3 GRADESCOPE

You should submit your code to *Gradescope*. If you are having trouble with your submission, you should double check the following common problems:

1. Make sure you are only submitting the three requested files, and they are named *BinaryTree.java*, *BinarySearchTree.java*, and *AVLTree.java* exactly.

2. Make sure you keep any *package* statements in your code before submitting. The autograder expects your files to have the package statements that are provided in the downloaded project.

3. Make sure your output is in the correct format. You should not be printing ANYTHING else or the autograder will think your output is incorrect.