University of Virginia, Department of Computer Science

BSTs and AVL - Tree Implementations

Dr. Mark R. Floryan

August 20, 2019

1 Summary

For this homework, you will be implementing three classes, which are all increasingly complicated Tree objects. You will start by implementing a basic Binary Tree with the three tree traversals convered in class. Then, you will implement and test a working Binary Search Tree. Lastly, you will implement a few methods to complete an AVL Tree.

1. Download the provided starter code
2. Implement some general methods in the BinaryTree class.
3. Implement the BinarySearchTree class
4. Implement the missing methods in the AVLTree class (some of this is done for you to simplify the assignment)
5. 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.
6. **FILES TO DOWNLOAD:** BinarySearchTrees.zip
7. **FILES TO SUBMIT:** TreeImplementations.zip

1.1 BinaryTree.java

To begin, implement the BinaryTree class. These are methods that are useful for ANY binary tree (whether balanced or not). You will need to implement the following methods:

```java
public class BinaryTree<T>{
    //Prints the tree using in-order traversal
    private void printInOrder(TreeNode<T> curNode);

    //Prints the tree using pre-order traversal
    private void printPreOrder(TreeNode<T> curNode);

    //Prints the tree using post-order traversal
    private void printPostOrder(TreeNode<T> curNode);
}
```

The Binary Tree contains a few methods that are implemented for you. This includes the main print functions, that simply call the helper functions described above on the root to give off the recursive prints. In addition, a method called printTree() is provided that prints the tree in a somewhat formatted method. This may be useful when debugging your code. Lastly, a simple recursive method that computes the height of the binary tree is provided as an example of recursion in trees.

1.2 BinarySearchTree.java

Next, you will implement a binary search tree. This class will extend the binary tree class from earlier, and thus inherit the print methods defined earlier. Your binary search tree should implement the provided Tree interface, shown below.

```java
public interface Tree<T extends Comparable<T>> {

    public void insert(T data);

    public boolean find(T data);

    public void remove(T data);

    public TreeNode<T> findMax(TreeNode<T> curNode);
}

/* You BST implements the interface above */
public class BinarySearchTree<T extends Comparable<T>>
```

```java
                              extends BinaryTree<T> implements Tree<T>{

            //TODO: Implement this class
}
```

You may add other supporting methods to your binary search tree if you find that to be helpful.

1.3 AVLTree.java

Lastly, 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:

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

      //Insert and remove are partially coded for you already

      //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);

}
```

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.

When you are done, submit your entire project as a zip file.