Assignment 8: AVL Trees
Part 2: AVL Tree Testing
CS3305/W01 Data Structures

Casey Hampson

October 22, 2024

Assignment Description

This assignment is super vague, so I will explain what I did and why I did it here. We are asked to confirm that the `AVLTree` class from the book spits out a valid AVL tree. An AVL tree is a binary search tree that is well balanced, meaning that all of its nodes have a balance factor, call it b, such that $|b| < 2$.

So, we need to test that 1) the tree it spits out is a binary search tree, and 2) that the tree is well balanced. If we can do that for three test cases (which I just will just randomly generate), then I think that should satisfy the requirements for this assignment. I choose to test only integers because there is no loss of generality compared to choosing any other comparable type.

To test if it is a binary search tree, we need to check that for each node, its left child's value is less and its right child's value is greater than its own value. To test if it is balanced, we can simply iterate through all the nodes and check to make sure that each node's balance factor is less than 2 and greater than -2.

Program Output

```
Inorder traversal of the tree's contents:
9 13 14 23 32 37 42 44 55 58 72 84
Binary Search Tree? Yes!
Balanced? Yes!
Is an AVL tree? Yes!

Inorder traversal of the tree's contents:
3 11 31 37 39 50 70 79 81 93
Binary Search Tree? Yes!
Balanced? Yes!
Is an AVL tree? Yes!

Inorder traversal of the tree's contents:
5 14 19 30 42 47 50 52 56 74 76 82 84 97
Binary Search Tree? Yes!
Balanced? Yes!
Is an AVL tree? Yes!

Inorder traversal of the tree's contents:
3 10 27 36 47 51 55 56 58 64 70 77 85 87 91 92 93
Binary Search Tree? Yes!
Balanced? Yes!
Is an AVL tree? Yes!

Inorder traversal of the tree's contents:
7 14 16 18 19 23 25 35 44 63 66 68 72 73 84 86 88 89
Binary Search Tree? Yes!
Balanced? Yes!
Is an AVL tree? Yes!
```

Source Code

```java
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 8-Part-2-AVL


import java.util.ArrayList;
import java.util.Random;


public class P2 {
    private static Random random;

    // we traverse through the tree to check that each node has that
    // its left child's value is less and its right child's value is greater
    // than its own value
    // then we recursively go through the left and right subtrees
    // doing the same thing until we reach a null value which returns true
    // since a null value is ``okay'' in this sense
    private static boolean IsBinaryTree(BST.TreeNode<Integer> node) {
        if (node == null) return true;
        // this is a bit complex looking but it just considers null children
        boolean check_current =
                (node.left == null ? true : node.left.element < node.element)
            &&  (node.right == null ? true : node.right.element > node.element);

        return check_current && IsBinaryTree(node.left) && IsBinaryTree(node.right);
    }
    public static boolean IsBinaryTree(AVLTree<Integer> tree) {
        BST.TreeNode<Integer> root = (AVLTree.AVLTreeNode<Integer>)tree.getRoot();
        return IsBinaryTree(root);
    }


    // this time we go through and compute the balance factors for each node
    // and ensure that its absolute value is less than 2
    private static boolean IsBalanced(BST.TreeNode<Integer> _node) {
        if (_node == null) return true;

        // polymorphism to grab the AVLTreeNode-specific height
        AVLTree.AVLTreeNode<Integer> node = (AVLTree.AVLTreeNode<Integer>)_node;

        int left_height = node.left == null ?
            0 : ((AVLTree.AVLTreeNode<Integer>)node.left).height;
        int right_height = node.right == null ?
            0 : ((AVLTree.AVLTreeNode<Integer>)node.right).height;
        boolean check_current = Math.abs(right_height - left_height) < 2;
        return check_current && IsBalanced(_node.left) && IsBalanced(_node.right);
    }
    public static boolean IsBalanced(AVLTree<Integer> tree) {
        BST.TreeNode<Integer> root = (AVLTree.AVLTreeNode<Integer>)tree.getRoot();
```

```
        return IsBalanced(root);
    }



    public static void TestAVLTree(AVLTree<Integer> tree) {
        // just print the tree's contents for completeness
        System.out.printf("Inorder traversal of the tree's contents:\n");
        tree.inorder();
        System.out.println();


        boolean is_bst = IsBinaryTree(tree);
        System.out.printf("Binary Search Tree? %s\n", is_bst ? "Yes!" : "No");

        boolean is_balanced = IsBalanced(tree);
        System.out.printf("Balanced? %s\n", is_balanced ? "Yes!" : "No");

        System.out.printf("Is an AVL tree? %s\n\n", (is_bst && is_balanced) ? "Yes!" : "No");
    }


    public static void main(String[] args) {
        random = new Random();

        // create 5 arrays of 10-20 integers ranging from 1-100
        ArrayList<Integer[]> arrs = new ArrayList<>();

        for (int i=0; i<5; i++) {
            int arr_len = random.nextInt(10, 20);
            Integer arr[] = new Integer[arr_len];
            for (int j=0; j<arr_len; j++) arr[j] = random.nextInt(1, 100);

            arrs.add(arr);
        }

        // make them all into trees
        ArrayList<AVLTree<Integer>> trees = new ArrayList<>();
        trees.add(new AVLTree<>(arrs.get(0)));
        trees.add(new AVLTree<>(arrs.get(1)));
        trees.add(new AVLTree<>(arrs.get(2)));
        trees.add(new AVLTree<>(arrs.get(3)));
        trees.add(new AVLTree<>(arrs.get(4)));

        // test each tree to make sure that it is an AVL tree
        trees.forEach(tree -> TestAVLTree(tree));
    }
}
```