CSE 373: Extra Assignment - Code

AVL Tree

Code due: Friday, June 2nd, 11:59 PM to Canvas

Introduction

In this assignment, you will implement the AVL tree. This assignment is out of 25 points, but there are 50 points possible. If completed, it will replace the lowest half assignment provided that you perform better on this assignment from your lowest score. **Your total course grade cannot go down from completing this assignment**.

Completing the extra credit on this assignment makes it possible to earn over 100% in homework portion of the course. The 25 base points for this assignment revolve around successfully being able to implement the AVL tree. Although your implementation must correctly implement dictionary behavior, points will be awarded for properly maintaining the AVL property. You will not earn any points for just submitting your BST file from HW3.

There are two extra credit portions. First, there will be ten points possible for correctly implementing a BFSIterator and fifteen possible points for implementing an AVL testing suite. Unlike previous testing suites, there is only one function. It will receive an object which has the functions listed in StringTree, without knowing anything about the object. You may assume that size() will return the correct answer, but no other assumption applies. The toTest object may not be empty when the static function is called.

You may not use the java libraries for this assignment

1 Provided Files

There are three provided files in this assignment. If you choose not to do the testing suite extra credit, you need only modify and submit AVLTree.java:

- `StringTree.java`: A java interface which indicates which functions your AVL Tree will support. These are also the only functions you may use in your test suite. *Do not modify this file*

- `AVLTree.java`: This is your implementation of the AVLTree. To make scripting easier, use the variable names and private class provided. You may add other fields and private functions as necessary. Leave `getBFSIterator()` as a `return null;` if you do not wish to attempt these points.
- `AVLTester.java`: This is an abstract class which has a single static function. It should return true if and only iff the object is an AVLTree. The only thing you may assume about the input object is that it has the functions listed in StringTree.java and that the function `size()` returns accurate values. Because this is an extra credit portion, there will be no trivial cases. All the test cases will test some interesting AVL bug.

2 Functionality

For the base points, your AVLTree needs to support the following functions

- `makeEmpty()`: Empties the AVLTree so no elements are stored.
- `size()`: Returns the number of elements in the tree.
- `insert(String key, String value)`: Inserts the `<key,value>` pair into the AVLTree. If the client attempts to insert a duplicate key, throw an `IllegalArgumentException`.
- `find(String key)`: Returns the value affiliated with the key. If the key is not in the AVLTree, it should throw a `NoSuchElementException`.
- `getBFSIterator` **Implementing the following is not a part of the 25 base points. If you do not wish to attempt the extra credit, leave this function as a** `return null;` This function should return an object which implements the Iterator interface. For behavior, it needs only to implement `hasNext()` and `next()`. `hasNext()` returns true if and only if there is another element left to return. `next()` returns the next key in a BFS traversal. Your traversal should start at the root and output left children before right.

Deliverables

To canvas, submit a `.zip` folder containing the following files. If you do not attempt any extra credit, submit a compressed folder with only your AVLTree.java file. If you also attempt the testing suite, include AVLTester.java. You may submit other helper java files, if necessary.