Project 5: AVL Tree

DUE: Sunday, November 22 at 11:59 PM

Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided)
- Have a style (indentation, good variable names, etc.)
- Comment your code well in JavaDoc style (no need to overdo it, just do it well)
- Have code that compiles with the command: **`javac *.java`** in your user directory

You may:

- Add additional methods and variables, however these methods **must be private**.
- You can import and use the following classes/interfaces in Java's util package (Queue, ArrayDeque, LinkedList, ArrayList)

You may NOT:

- Copy code from the Internet/book. The code should be your own.
- Make your program part of a package.
- Add additional public methods or variables
- Alter any method signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Alter provided classes that are complete.
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the `project5.zip` and unzip it. This will create a folder `section-yourGMUUserName-p5`;
- Rename the folder replacing `section` with the `001`, `002`, `003`, `etc` based on the lecture section you are in;
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address;
- After renaming, your folder should be named something like: `001-krusselc-p5`.
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`)

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p5.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:

```
001-krusselc-p1.zip --> 001-krusselc-p5 --> JavaFile1.java
                                            JavaFile2.java
                                            JavaFile3.java
                                            ...
```

- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Overview

An AVL tree (named after its inventors Adelson-Velsky and Landis) is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

In this project we are going to implement AVL tree ADT. The implementation includes the definition of AVLNode and AVLTree classes. Already existing classes such as BinaryNode and BinarySearchTree can be used.

Implementation/Classes

The following classes are needed for this project. Here we provide a description of these classes.

CLASSNAME (FILENAME): **AVLNode.java (30 pts)**

This class extends the BinaryNode class of trees. An AVLNode contains a **height** as an attribute. The height attribute indicates the height of a subtree rooted at that specific node. The following are the list of methods in this class:

+AVLNode(): constructor - initializes all the attributes to null. and height to -1.

+AVLNode(T data): constructor - initializes the data with the accepted value and the rest attributes to null. Note that this operation may affect the height of an AVLNode.

+AVLNode(T data, AVLNode<T> leftNode, AVLNode<T> rightNode): constructor - initializes the attributes to the given values. Note that this operation affects the height of an AVLNode.

+setLeftChild() and setRightChild(): - the setter methods for the leftChild and the rightChild attributes of AVLNode. Note that this operation affects the height of an AVLNode.

+getHeight():int - This method returns the height attribute of an AVLNode.

The following methods are inherited from the superclass: getData(), setData(T newData), getLeftChild(), getRightChild(), hasLeftChild(), hasRightChild(), isLeaf(), copy(), getHeight(), getNumberOfNodes().

CLASSNAME (FILENAME): **AVLTree.java (70 pts)**

This class represents the self- balancing AVL tree. The class extends binary search tree. The following are the members of the AVLTree class.

+AVLTree() : constructor - initializes the root to null.

+AVLTree(T rootEntry): constructor - initializes the root with the accepted value.

+add (T newEntry): T - add a new entry to the AVL tree. Note that you need to define the rotate methods for the add method to work.

+ remove(T entry): T – removes and returns the removed element from the tree. Rotate methods are used for this method implementation. If the tree is empty, the method throws EmptyTreeException.

The following class are provided to you and you should NOT change them.

- BinaryNode
- TreeInterface
- BinaryTreeInterface
- SearchTreeInterface
- BinaryTree
- BinarySearchTree
- EmptyTreeException

The following link gives a good simulation on the add and remove methods of AVLTree:

https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

A Wikipedia page on AVL trees:

https://en.wikipedia.org/wiki/AVL_tree

Requirements

An overview of the requirements are listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes AVLNode and AVLTree.