Homework 6: AVL Tree

Due: See T-Square

Updates

Earlier, it was mentioned that every time a node was added or removed, the heights and balance factors were updated for every node. The correct statement would be that the heights and balance factors should be updated for all the ancestors of the added/removed node, thus keeping the add and remove in O(logn).

Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 7**. This includes unused code, so don't submit extra files that don't compile. (Java is backwards compatabile so if it compiles under JDK 6 it *should* compile under JDK 7.)
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, or method signatures.
4. Do not add additional public methods when implementing an interface.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
6. You must submit your source code, the `.java` files, not the compiled `.class` files.
7. After you submit your files redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

AVL tree

You will be coding an AVL tree for this homework. It has all the properties of a BST with the added benefit of being self balancing. Adding and removing are almost identical to that of a BST but the subtree must be balanced after addition/removal. This means that you can reuse a lot of code that you used for your BST homework (especially for unchanging methods such as get(), contains(), etc.).

Constructor

The AVL class should be fine with the default parameterless constructor that Java will provide, so there is no need to write one. The constructor for Node is already provided.

Balancing

Each time you add/remove a node from the tree you must recalculate the heights and balance factors for each of the node's ancestors. Do not update the heights and balance factors for every node in the tree as this will increase the complexity to O(n) as opposed to O(logn).

Generally speaking and for the purposes of this homework, the height of a tree (and subtree) is defined as max(leftChild.height, rightChild.height) + 1. Each node in the tree will have a height field which represents the height of the subtree rooted at that node. (See the Node class provided.) A subtree consisting of only a single node (left and right are both null) has height 1. (You can think of a null as representing a 0 height subtree.) Any leaf node will have height 1 in that it is the root of its own subtree, but having null valued left and right children, its height is the max(0, 0) + 1 or just simply 1. A tree consisting of a root having a left leaf and a right leaf (50(30()())(70()())) would have 2 stored as the height at the root (the 50 node in this example), and each leaf (the 30 node and the 70 node) would have height 1 stored. You can see many more examples in the textbook on page 575. Figure 23-7 shows trees of heights varying from 1 to 5. In each case, it is the root of the tree that would have the largest of the "heights". Leaves always have height 1.

The balance factor (equivalent to height difference in the textbook) of a node is defined as (leftChild.height - rightChild.height). This is an arbitrary choice as you can define it the other way around and still have a working AVL. **However** for the purposes of this class, the balance factor will always be defined this way.

Driver

For basic testing purposes, you can use the same driver you wrote for your BST along with the User class. Many JUnits are provided with this assignment to help test your AVL. It is highly recommended that you write additional JUnits. The JUnits provided with the assignment are not intended to be complete.

Style and Formatting

It is important that your code is functional and is also written clearly and with good style. Efficiency also always matters. Be sure you are using the updated style checker located in T-Square-Resources. (It was updated on February 5, 2014.) Be sure to run your code against this latest style checker. While it is backward compatible with the original style checker, some style rules have been made easier to pass, so it is in your interest to update to this Feb 5 style checker file.) If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Kush Mansingh kmansingh3@gatech.edu with the subject header of "Style XML".

Javadocs

All public methods must have javadocs (or @Override's where appropriate). All private methods must be commented, but it is not necessarily that they be in javadoc style.

Provided

The following files have been provided to you:

1. `BSTInterface.java`
2. `AVL.java`
3. `Node.java`
4. `BasicAVLTest.java`
5. `CheckStyle.zip` - see T-Square Resources for the latest file from February 5, 2014

Deliverables

You must submit all of the following files. Please make sure the filename matches the filenames below.

1. `AVL.java`

Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc. You may attach each file individually, or submit them in a zip archive.