Homework 3

Due: September 14, 11:55 PM

Important

There are a few guidelines you must follow in this homework. If you fail to follow any of the following guidelines you will receive a **0** for the entire assignment.

1. All submitted code must compile under **JDK 7**. This includes unused code, 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. Don't include any package declarations in your classes.
3. Don't change any *existing* class headers or method signatures. (It is fine to add extra methods and classes)
4. Don't import anything that would trivialize the assignment. (e.g. don't import `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
5. You must submit your source code, the `.java` files, not the compiled `.class` files.

After you submit your files redownload them and run them to make sure they are what you intended to submit. We are not responsible if you submit the wrong files.

Assignment

In this assignment you will be coding an AVL tree. Of course your tree **MUST** remain balanced (under the rules of an AVL tree given in class/recitation). Make sure you read the javadocs as well as the remainder of this document.

Node

A suggested Node class has been provided, you do not need to change it, but may modify it in any way you wish (Including deleting it and making it a private class). Just make sure you re-submit the node class even if you choose not to change it.

Definitions

Height: (max height of children) + 1

Balance Factor: left height - right height

Fields

You must have a root field, and it's recommended to have a size field.

Methods

add: This takes in some data of a generic type and adds a new node to the tree containing that data

remove: This takes in some data of a generic type and finds the object in the tree that is equal to the given data, then removes and returns the data from the tree. If the data is not found in the tree return null.

get: This takes in some data of a generic type and finds the object in the tree that is equal to the given data, then returns the data from the tree. If the data is not found in the tree return null.

asSortedList: This method returns a generic list which contains all of the data in the list in sorted order. Note that you should not call a sort method on this list, this method must have O(n) running time. Think about what traversal might give you the correct list.

Suggested Implementation

void calcHeightAndBF(Node<T> n): Calculates the height and BF of *just* node n. There doesn't need to be any recursion in this method. Just make sure all of the descendants of n are already updated. You can make sure this is true by updating each node as you recursively rebuild the tree in **add** or **remove**

Node<T> rotate(Node<T> n): Method that determines which of the four AVL rotations need to be performed. If a rotation was performed return the new root of the rotated sub-tree. If none of the rotations need to be performed simply return n. Make sure if you do a rotation that you correctly update the heights and balance factors of the appropriate nodes.

Node<T> left (Node<T> n): Performs a single left rotation on the subtree with root n. This is where the root is right heavy, and the right child is right heavy. Return the new root of the subtree.

Node<T> right (Node<T> n): Performs a single right rotation on the subtree with root n. This is where the root is left heavy, and the left child is left heavy. Return the new root of the subtree.

Node<T> leftRight (Node<T> n): Performs a double left right rotation on the subtree with root n. This is where the root is left heavy, and the left child is right heavy. Return the new root of the subtree.

Node<T> rightLeft (Node<T> n): Performs a double right left rotation on the subtree with root n. This is where the root is right heavy, and the right child is left heavy. Return the new root of the subtree.

Null

Your tree should support adding and removing null elements. Null should be treated as positive infinity, greater than everything else. Here is a suggested way of handling nulls easily using a comparator:

```
private Comparator<T> comp = new Comparator<T>() {
   public int compare(T ths, T tht) {
      if (ths == tht) {
         return 0;
      } else if (ths == null) {
         return 1;
      } else if (tht == null) {
         return -1;
      } else {
         return ths.compareTo(tht);
      }
   }
};
```

Then you can pass in the two objects you want to compare into the comparator and not worry about calling the compareTo method on a null object.

Assumptions

1. You may assume no duplicate data is added.

Deliverables

You must submit all of the following files.

1. `AVLTree.java`
2. `Node.java`
3. Other required files (optional)

You may attach them each individually, or submit them in a zip archive.