Homework 3: Binary Search Tree

Due: January 27, 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, constructors, 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.

Binary Search Tree

For this assignment you will be implementing a binary search tree (BST). This means that each node in the tree might potentially have a left and right child. For this specific search tree, the potential left child of any parent node is less than that parent, and the potential right child is greater than the parent.

For this BST, no duplicate data entries will be allowed. (this means you can assume we won't give you two duplicate entries)

Null should be treated as positive infinity.

Although many of the methods you will be coding can be implemented using iteration, we highly recommend that you use recursion.

Recursion can be used to find the proper location in the tree for add, remove, and get operations. You can simply check to see if you are in the proper location and if not, call the same method on the appropriate child (depending on if you want to go left or right). To avoid complex base cases, we recommend that you check to see if your incoming argument is null, instead of checking what state your current node is in (one left child, one right child, etc.).

You will also be implementing tree traversals as well. There is not agreed upon way of serializing (linear representation) a tree. There are three different methods that you will write to traverse the tree and return the order in which data entries are visited. Pre-order, in-order, and post-order. All of these should be implemented recursively.

Add

To add to the BST you might want to create a new private method. This will be a recursive helper that will call itself. From the provided add method, you first call this helper with the root of the tree as the argument. From here you can recursively find the correct spot to stick on a new node.

The important part of this process is to assign the appropriate reference from your current node to the output of your recursive call at every level of your recursive search. This means that at the end of the line (when you finally get null as an argument) you can simply return the new node that contains the appropriate data. All of the recursive calls will bounce back up until they hit the root, returning to the first helper call that you made from the provided add method.

Contains

This method is very similar to the add method. In this operation you might want to use the same technique as adding, with a distinction that you are looking for an existing piece of data.

You will return a boolean indicating whether or not the generic data passed in as an argument exists in the tree.

Remove

Removing an item from a BST is a more complicated procedure. Once again, you can use the same recursive technique to find the node to be removed. If the node does not exist, return null.

There are several cases to handle when removing from a BST. In general, it is easy to understand that, if we are removing a node in the tree, we will need to replace it with another node. This is like shifting things down when you remove an item from a sorted list.

Just like the sorted list, if the item is at the end, there is no shifting involved. The same goes for the BST removal. If the node to be removed had no children then we can simply remove it with no further complication.

There are two other cases. The node to be removed may have one child. In this case it is very much like removing from a sorted list. We can easily trash the node and replace it with its single child (if implemented recursively, this is done by just returning its child).

The most complicated case is when the node to be removed has both a right and a left child. Here we have to ask, how do we find the right value to replace the node that we are removing? There are actually two such nodes in the tree, the predecessor and successor. The node that we are trying to remove is basically a middle value. It is greater than everything to its left and less than everything to its right. So naturally, it makes sense that the proper replacement value should be either, the largest value of the lesser side (predecessor) or the least value of the greater side (successor).

For this assignment use the **PREDECESSOR**, or your grade will not be happy!

The predecessor is found by looking at the left child of the current node, and then traversing all the way right until you hit a null right reference.

The trick to removing in this situation is to find the node to be removed, find its predecessor, replace the data in the node that you want to remove with the data of the predecessor, and remove the predecessor node (which by definition is a single or zero child case).

Traversals

For these methods you will traverse the tree in the appropriate order, adding each data entry to a list and returning this list at the end of the traversal.

You are also given a reconstruct method to complete. In this method you are given several traversals and are expected to clear the tree, and then rebuild it so that it would produce the given traversals. This is something you need to figure out, don't ask the TA's explicitly how to do this method. It will probably help to do a few cases by hand first.

Deliverables

You must submit all of the following files.

1. `BST.java`

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