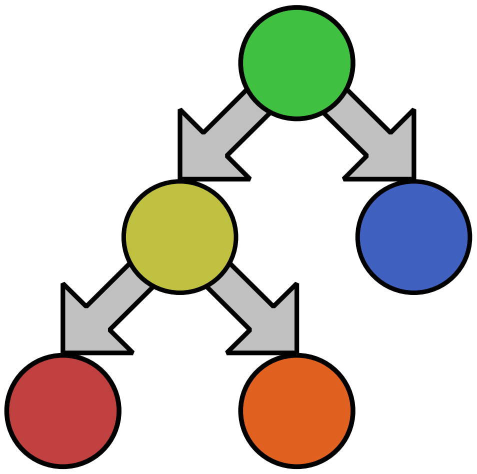
California State University, Sacramento
College of Engineering and Computer Science

Computer Science 130: Data Structures and Algorithm Analysis

Assignment #3 – Binary Search Tree ADT

Overview



For this assignment, you are going to create a Binary Search Tree (BST) with a minimal interface. You don't have to balance the tree. In fact, don't even try it yet. We are creating a very basic tree class.

For the sake of testing, we are going to restrict this tree to use Integers rather than the generic Object class. Like all programming assignments, work on it in parts. You can use my source code for a basic tree a basis for this assignment.

Part 1: BinarySearchTree Class

Overview

The BinarySearchTree is a tad more sophisticated than the last version.

Now, nodes aren't simply linked together and assigned to the tree's node. In fact, the root **must** be private. So, the constructor, in the last assignment that assigned the root, must be removed. All values are added to the using the add() method. It will find the correct position in the tree and store it there.

Interface

Just like before, the BinarySearchTree's methods will start recursion on the Node class.

`public class BinarySearchTree`		
	`BinarySearchTree ()`	Constructor.
`String`	**`about()`**	Returns text about you – the author of this class.
`void`	**`printValues()`**	Starts recursion from the root node.
`void`	**`printTree()`**	Starts recursion from the root node.
`void`	**`add(int value)`**	Adds the value to the correct position in the BST. If the value already exists, do nothing. So, basically, you are creating a proper-set of numbers.
`boolean`	**`contains(int value)`**	Returns True if the specified value is in the tree
`void`	**`remove(int value)`**	Search the tree and remove the value if it is found. If it is not found, do nothing.
`void`	**`clear()`**	Removes all the nodes from the tree. It is best to use a recursive postorder. Set left and right to null.

Part 2: Node Class

Overview

In recursively defined structures, like trees, all the coding (and complexity) is found in the recursive structure itself. In the case of trees, the node will contain the vast amount of the logic and behavior.

The node has changed slightly. In particular, for this assignment, let's use integers rather than the generic Object class.

To implement the remove() method, you will need to find (and remove) the max or minimum node. The algorithm uses the same basic logic as contains(), but takes special actions if the node is found. You only need to implement one of the two: removeMax() or removeMin().

Interface

`public class Node`		
	`Node()`	Constructor
	`Node(Object, Node, Node)`	Constructor that assigns the data, left and right nodes.
`Node`	**`left`**	
`Node`	**`right`**	
`int`	**`data`**	The value that the node contains.
`void`	**`printValues()`**	Prints the contents of the tree using an **infix tree traversal**. They should be sent to standard out with spaces between each value. Feel free to redirect the stream if you like.
`void`	**`printTree(int indent)`**	Unlike the other print method, this one will print the structure of the tree. One node will be printed per line. You should use prefix tree traversal. Feel free to redirect the stream if you like.
`void`	**`add(int value)`**	Adds the value to the correct position in the BST. If the value already exists, do nothing. So, basically, you are creating a proper-set of numbers.
`boolean`	**`contains(int value)`**	Returns True if the specified value is in the tree
`Node`	**`removeMin()`**	Needed to implement remove.
`Node`	**`removeMax()`**	Needed to implement remove.
`void`	**`remove(int value)`**	Search the tree and remove the value if it is found. If it is not found, do nothing.
`void`	**`clear()`**	Removes all the nodes from the tree. It is best to use a recursive postorder. Set left and right to null.

Part 3: Testing

Once you have finished your code, you need to test it using some good test data. Now you can see why you wrote the PrintTree method. It is vital to seeing if your methods are working correctly.

For example, the following is a basic demonstration of the add() method.

```
BinarySearchTree tree = new BinarySearchTree();

tree.add(10);
tree.add(43);
tree.add(18);
tree.add(6);
tree.add(50);
tree.add(8);

tree.printTree();
```

Should produce the following tree:

```
+-- 10
  +-- 6
    +-- 8
  +-- 43
    +-- 18
    +-- 50
```

Turn in

- The source code.
- The main program that runs the tests. These tests should be hard-coded. I will run your programs and make modifications to test them.
- Output generated by your tests.