CSE143X Section #18 Problems

For these problems, assume that you are using a binary tree of ints defined as:

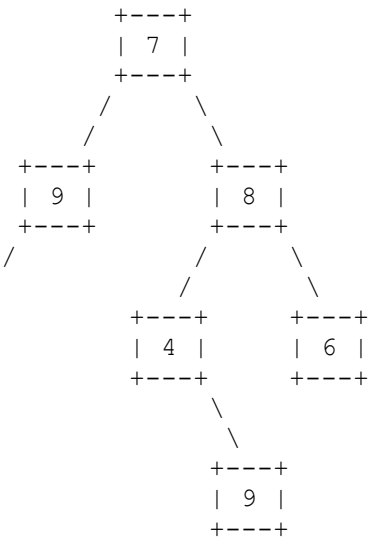
```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree

    // post: constructs a leaf node with given data
    public IntTreeNode(int data) {
        this(data, null, null);
    }

    // post: constructs a branch node with the given data and links
    public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}

public class IntTree {
    private IntTreeNode overallRoot;

    <methods>
}
```

For each problem below you will write a new public method for the IntTree class. You may define additional private methods to implement your public methods, but otherwise you may not call any other methods of the class.

1. Write a method writeTree that writes the tree to System.out in a specific format. You are to perform a preorder traversal of the tree, producing exactly one line of output for each node. Each output line should begin with a code to indicate what kind of node it is followed by the data stored in the node. The different kinds of nodes are indicated below:

Code	Node Kind
0	leaf node (no children)
1	branch node with left child only (empty right)
2	branch node with right child only (empty left)
3	branch node with nonempty left and right children

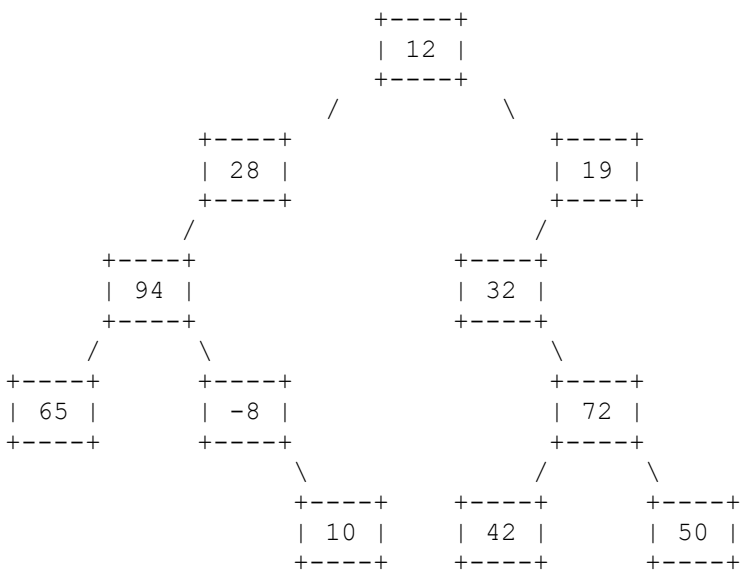
For example, below is a binary tree and the output it would produce:



Output Produced:

```
3 7
1 9
0 5
3 8
2 4
0 9
0 6
```

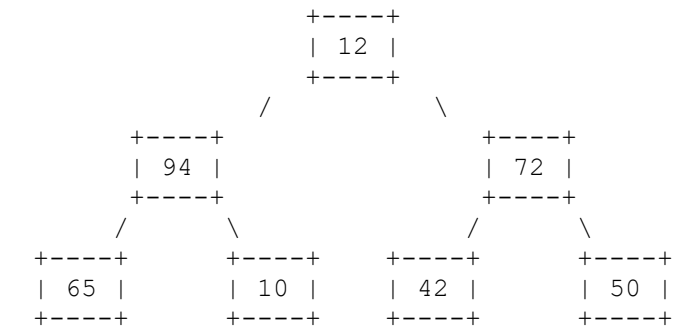
2. Write a method tighten that eliminates branch nodes that have only one child from a binary tree of integers. For example, if a variable called t stores a reference to the following tree:



then the call:

```
t.tighten();
```

should leave t storing the following tree:



Notice that the nodes that stored the values 28, 19, 32, and -8 have all been eliminated from the tree because each had one child. When a node is removed, it is replaced by its child. Notice that this can lead to multiple replacements because the child might itself be replaced (as in the case of 19 which is replaced by its child 32 which is replaced by its child 72).

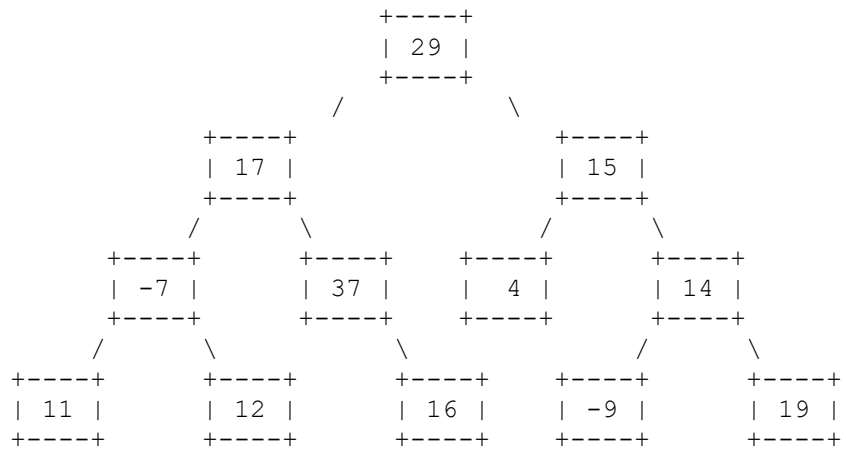
3. Write a method readTree that takes a Scanner as a parameter and that replaces the current tree with one constructed from data stored in the Scanner. The format will be the exact same one used in problem #1 where a tree has been stored using a preorder traversal with one line of input for each node. Each line of input has a code indicating the kind of node followed by the data stored in the node.

   The different kinds of nodes are indicated below:

Code	Node Kind
0	leaf node (no children)
1	branch node with left child only (empty right)
2	branch node with right child only (empty left)
3	branch node with nonempty left and right children

   See problem #1 for an example. You may assume that there is at least one line of input, meaning that the tree you are constructing will have at least one node. Remember that you are creating a new tree based on the information in the input file.

4. Write a method limitPathSum that removes nodes from a binary tree of integers to guarantee that the sum of the values on any path from the root to a node does not exceed some maximum value. For example, suppose that a variable t stores a reference to the following tree:

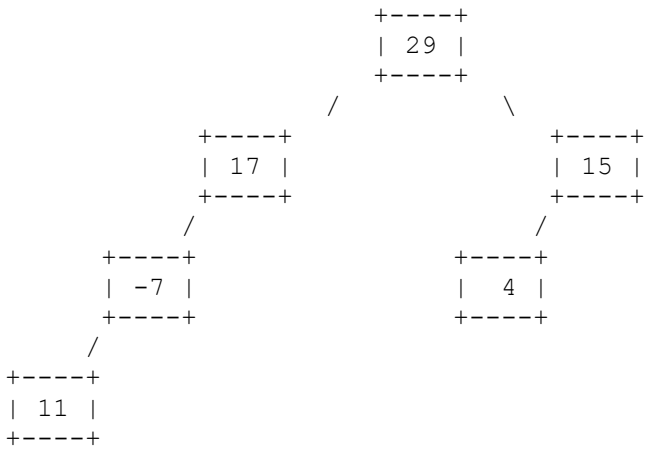


   Then the call:

   ```
   t.limitPathSum(50);
   ```

   will remove nodes so as to guarantee that no path from the root to a node has a sum that is greater than 50. This will require removing the node with 12 in it because the sum of the values from the root to that node is greater than 50 (29 + 17 + -7 + 12, which is 51). Similarly, we have to remove the node with 37 in it because its sum is too high (29 + 17 + 37, which is 83). Whenever you remove a node, you remove anything under it as well, so removing the node with 37 also removes the node with 16 in it. We also remove the node with 14 and everything under it because its sum is too high (29 + 15 + 14, which is 58).

Thus, we end up with:

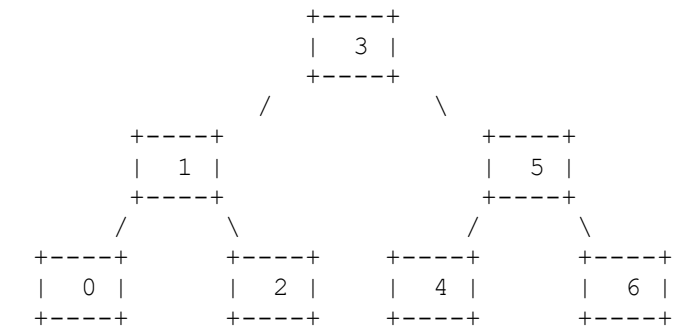


The method would be forced to remove all nodes if the data stored at the overall root is greater than the given maximum.

5. Write a method construct that takes an integer n as a parameter and that constructs a new tree of integers with n nodes. The nodes should be numbered 0 through (n - 1) such that an inorder traversal of the tree will produce the values in sequential order. The tree should be balanced in that the number of values in any node's left subtree should always be within one of the number of values in its right subtree. If there is an extra value in one of a node's two subtrees, it should appear in the right subtree. For example, given a variable t of type IntTree, the following call:

```
t.construct(7);
```

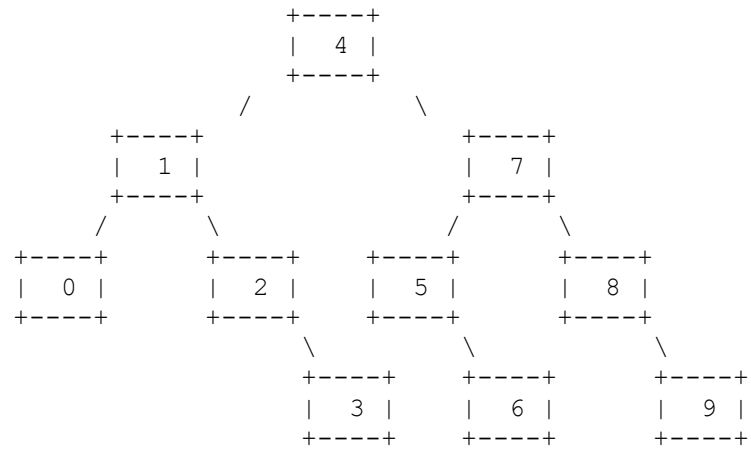
should produce the following tree:



If the call had been:

```
t.construct(10);
```

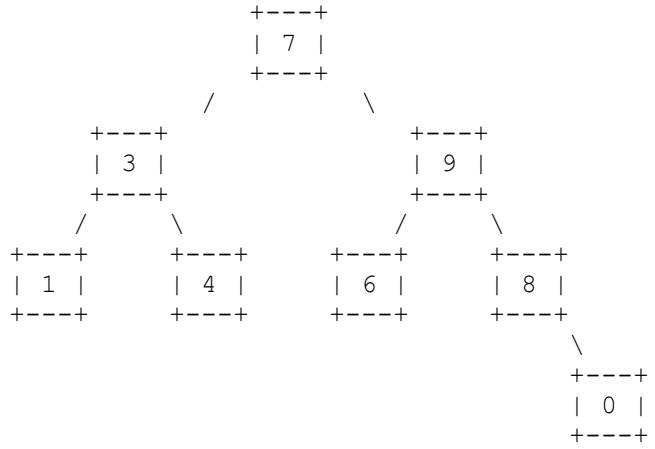
the following tree should be produced:



Your method should replace any existing tree. If should construct an empty tree if passed a value of 0 and should generate an IllegalArgumentException if passed a negative value.

6. Write a method removeLeaves that removes the leaves from a binary tree of integers. A leaf is a node that has empty left and right subtrees.

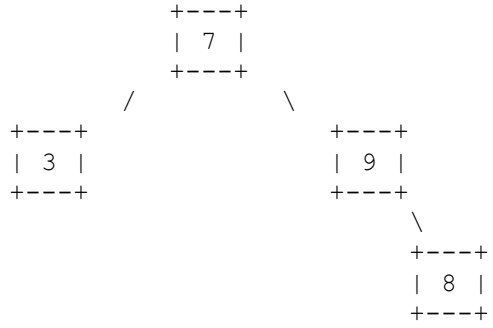
For example, if a variable t stores a reference to the following tree:



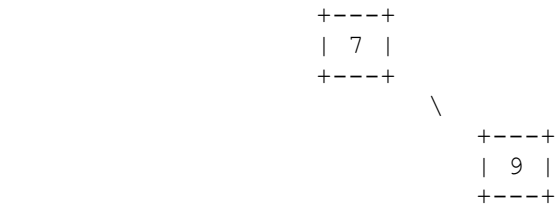
Then the call:

```
t.removeLeaves();
```

should remove the four leaves from the tree (the nodes with data values 1, 4, 6 and 0), leaving the following tree:



A second call on the method would eliminate the two leaves in this tree (the ones with data values 3 and 8):



Another call would eliminate the one leaf with data value 9:

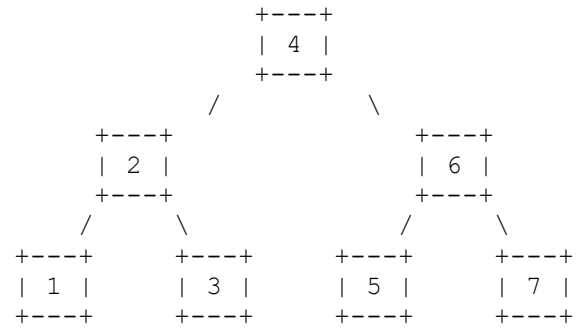
Another call would leave an empty tree because the previous tree is composed of exactly one leaf node. If called with an empty tree, the method does not change the tree because there are no nodes of any kind (leaf or not).

7. Write a method construct that takes a sorted array of integers as a parameter and that constructs a balanced binary search tree containing those integers. The constructed tree should have the property that for every node in the tree, either the left and right subtrees have the same number of nodes or the left subtree has one more node than the right subtree.

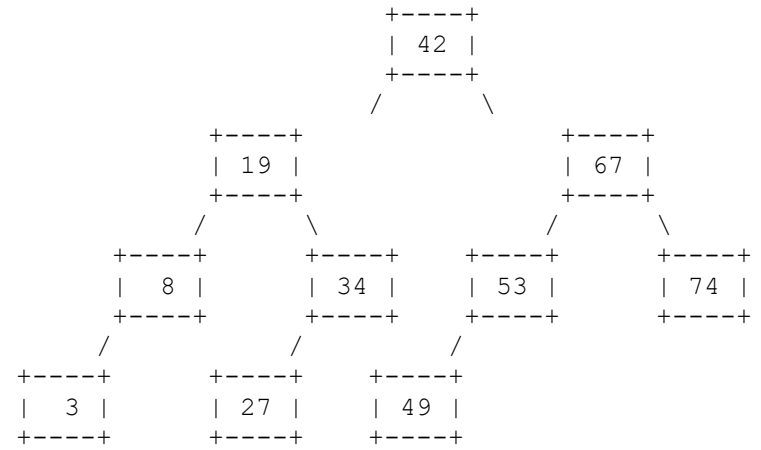
   For example, if an array called list stores the values (1, 2, 3, 4, 5, 6, 7) and the following call is made for a variable t of type IntTree:

```
t.construct(list);
```

   Then t should store the following tree after the call is made:



   If the array had instead stored (3, 8, 19, 27, 34, 42, 49, 53, 67, 74), then the following tree would have been constructed:

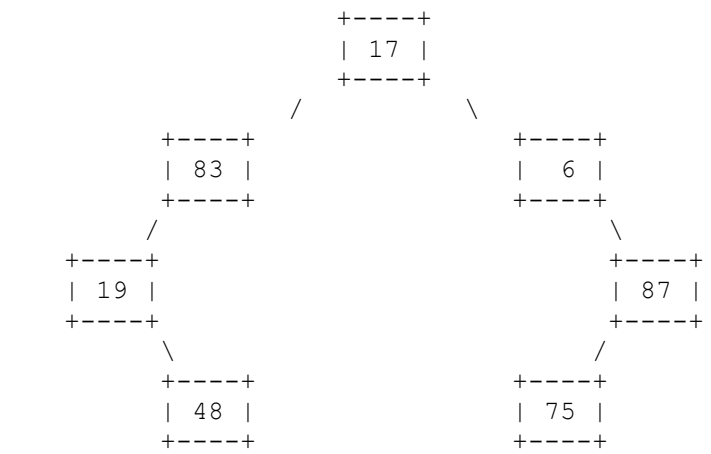


   Notice that when it is not possible to have left and right subtrees of equal size, the extra value always ends up in the left subtree, as in the overall tree which has 5 nodes in the left subtree and 4 in the right.

This problem involves constructing a new tree from an array of values. If there are "n" values in the array, then you should construct exactly "n" tree nodes. The new tree should replace any old tree. You are not allowed to use any other data structures (arrays, lists, Strings, etc) to solve this problem, you are not allowed to alter the array that you are passed and your solution must run in O(n) time. You can, however, assume that the values in the array appear in sorted (nondecreasing) order.

8. Write a method completeToLevel that takes an integer n as a parameter and that adds nodes to a binary tree so that the first n levels are complete. A level is considered complete if it has every possible node at that level. We will use the convention that the overall root is at level 1, that it's children are at level 2, and so on. You should preserve any existing nodes in the tree. Any new nodes added to the tree should have -1 as their data.

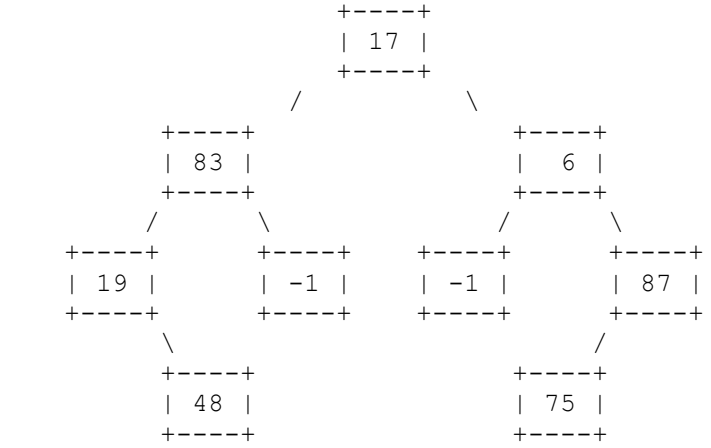
   For example, if a variable called t refers to the following tree:



   and you make the following call:

```
t.completeToLevel(3);
```

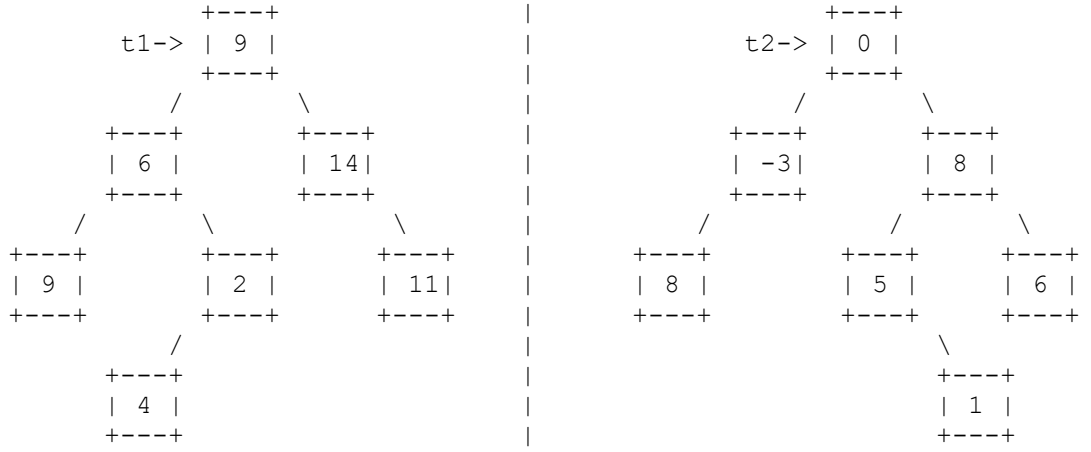
   Then t should store the following tree after the call is made:



   In this case, the request was to fill in nodes as necessary to ensure that the first 3 levels are complete. There were two nodes missing at level 3. Notice that level 4 of this tree is not complete because the call requested that nodes be filled in to level 3 only.

   Keep in mind that your method might need to fill in several different levels. Your method should throw an IllegalArgumentException if passed a value for level that is less than 1.

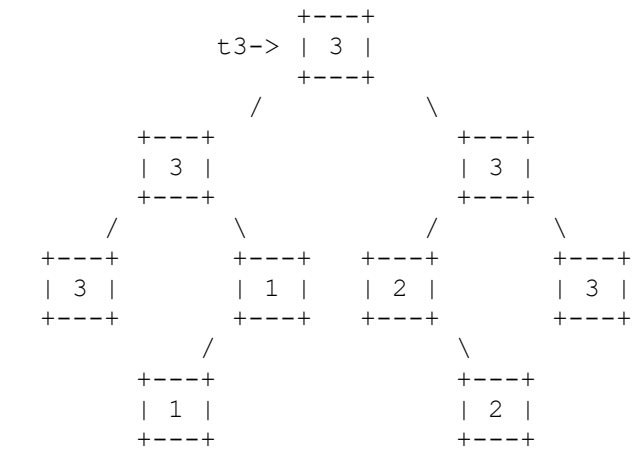
9. Write a method combineWith that constructs a binary tree of integers by combining two other trees. The combined tree should have each of the nodes that appears in either of the two trees. The nodes of the new tree should store an integer indicating which of the original trees had a node at that position (1 if just the first tree had the node, 2 if just the second tree had the node, 3 if both trees had the node). Consider, for example, the following trees.



Suppose the following call is made:

```
IntTree t3 = t1.combineWith(t2);
```

The variable t3 will refer to the following tree:



Notice that the two nodes in t1 that have no corresponding nodes in t2 (the nodes storing 2 and 4) store the value 1 in the newly constructed tree while the two nodes in t2 that have no corresponding nodes in t1 (the nodes storing 5 and 1) store the value 2 in the newly constructed tree. All other nodes store the value 3 to indicate that they appeared in both of the original trees.

You may assume the IntTree class has a zero-argument constructor that constructs an empty tree. You will construct IntTreeNode objects to build the new tree, but you should not change either of the two original trees.