A Method to Count the Number of Leaf Nodes in a Binary Tree

We want to write a method which counts the number of leaf nodes in a given binary tree. Leaf nodes are defined as nodes which have no children.

What parts of the tree will we need to process to accomplish this?

We must process the entire tree, and cannot leave any nodes out, because all the leaves are at the bottom of the tree.

Assume we have a node root which is the root node of a tree. How can we recursively define the number of leaf nodes in that tree?

The number of leaves in a tree for which root is the root node is the sum of the number of leaves in root’s left subtree and the number of leaves in root’s right subtree. In other words, we can say that

\[
\text{leafCount}(\text{root}) = \text{leafCount}(\text{root}.\text{left}) + \text{leafCount}(\text{root}.\text{right}).
\]

What will be our base case(s)?

We need to terminate the recursion if we are either at a leaf or if we are given an empty tree or subtree.

What will our function return? An integer value.

```java
public int leafCount( Node root ) {
    if ( root == null )
        return 0;
    else if ( (root.left != null) || (root.right != null) )
        return (leafCount(root.left) +
                leafCount(root.right));
    else
        return 1; //if we got here this must be a leaf
}
```

What would we need to do if we wanted to write this method iteratively? What sort of data structures would we need?

We would need to maintain a stack onto which we could push the left and right subtree’s root nodes as we iterated.

Notice we could also implement this algorithm iteratively using a level-order traversal of the binary tree. Would this implementation be less efficient for balanced trees? Why or why not?

No, it would not be less efficient, because we would still need to touch every node in the tree once, just like we do in our depth-first recursive traversal above.