Count Complete Tree Nodes

Given a **complete** binary tree, count the number of nodes.

Definition of a complete binary tree from Wikipedia:
In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2^h nodes inclusive at the last level h.

Solution 1

Main Solution - 572 ms

```
class Solution {
    int height(TreeNode root) {
        return root == null ? -1 : 1 + height(root.left);
    }
    public int countNodes(TreeNode root) {
        int h = height(root);
        return h < 0 ? 0 :
               height(root.right) == h-1 ? (1 << h) + countNodes(root.right)
                                         : (1 << h-1) + countNodes(root.left);
    }
}
```

Explanation

The height of a tree can be found by just going left. Let a single node tree have height 0. Find the height `h` of the whole tree. If the whole tree is empty, i.e., has height -1, there are 0 nodes.

Otherwise check whether the height of the right subtree is just one less than that of the whole tree, meaning left and right subtree have the same height.

- If yes, then the last node on the last tree row is in the right subtree and the left subtree is a full tree of height h-1. So we take the 2^h-1 nodes of the left subtree plus the 1 root node plus recursively the number of nodes in the right subtree.
- If no, then the last node on the last tree row is in the left subtree and the right subtree is a full tree of height h-2. So we take the 2^(h-1)-1 nodes of the right subtree plus the 1 root node plus recursively the number of nodes in the left subtree.

Since I halve the tree in every recursive step, I have O(log(n)) steps. Finding a height costs O(log(n)). So overall O(log(n)^2).

Iterative Version - 508 ms

Here's an iterative version as well, with the benefit that I don't recompute `h` in every step.

```
class Solution {
    int height(TreeNode root) {
        return root == null ? -1 : 1 + height(root.left);
    }
    public int countNodes(TreeNode root) {
        int nodes = 0, h = height(root);
        while (root != null) {
            if (height(root.right) == h - 1) {
                nodes += 1 << h;
                root = root.right;
            } else {
                nodes += 1 << h-1;
                root = root.left;
            }
            h--;
        }
        return nodes;
    }
}
```

A Different Solution - 544 ms

Here's one based on victorlee's C++ solution.

```
class Solution {
    public int countNodes(TreeNode root) {
        if (root == null)
            return 0;
        TreeNode left = root, right = root;
        int height = 0;
        while (right != null) {
            left = left.left;
            right = right.right;
            height++;
        }
        if (left == null)
            return (1 << height) - 1;
        return 1 + countNodes(root.left) + countNodes(root.right);
    }
}
```

Note that that's basically this:

```
public int countNodes(TreeNode root) {
    if (root == null)
        return 0;
    return 1 + countNodes(root.left) + countNodes(root.right)
```

That would be O(n). But... the actual solution has a gigantic optimization. It first walks all the way left and right to determine the height and whether it's a full tree, meaning the last row is full. If so, then the answer is just 2^height-1. And since always at least one of the two recursive calls is such a full tree, at least one of the two calls immediately stops. Again we have runtime O(log(n)^2).

written by StefanPochmann original link here

Solution 2

```
class Solution {

public:

    int countNodes(TreeNode* root) {

        if(!root) return 0;

        int hl=0, hr=0;

        TreeNode *l=root, *r=root;

        while(l) {hl++;l=l->left;}

        while(r) {hr++;r=r->right;}

        if(hl==hr) return pow(2,hl)-1;

        return 1+countNodes(root->left)+countNodes(root->right);

    }

};
```

written by victorlee original link here

Solution 3

public class Solution {

```
public int countNodes(TreeNode root) {

    int leftDepth = leftDepth(root);
    int rightDepth = rightDepth(root);

    if (leftDepth == rightDepth)
        return (1 << leftDepth) - 1;
    else
        return 1+countNodes(root.left) + countNodes(root.right);

}

private int rightDepth(TreeNode root) {
    // TODO Auto-generated method stub
    int dep = 0;
    while (root != null) {
        root = root.right;
        dep++;
    }
    return dep;
}

private int leftDepth(TreeNode root) {
    // TODO Auto-generated method stub
    int dep = 0;
    while (root != null) {
        root = root.left;
        dep++;
    }
    return dep;
}
```

}

written by mo10 original link here