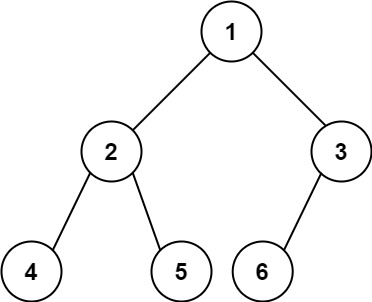
<https://leetcode.com/problems/count-complete-tree-nodes/description/>

Given the root of a **complete** binary tree, return the number of the nodes in the tree.

According to **Wikipedia**, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Design an algorithm that runs in less than O(n) time complexity.

**Example 1:**



**Input:** root = [1,2,3,4,5,6]

**Output:** 6

**Example 2:**

**Input:** root = []

**Output:** 0

**Example 3:**

**Input:** root = [1]

**Output:** 1

**Constraints:**

The number of nodes in the tree is in the range [0, 5 \* 10^4]..

0 <= Node.val <= 5 \* 10^4

The tree is guaranteed to be **complete**.

**Attempt 1: 2024-04-02**

**Solution 1: DFS (10 min)**

**Style 1: return int**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public int countNodes(TreeNode root) {

        return helper(root);

    }

    private int helper(TreeNode root) {

        if(root == null) {

            return 0;

        }

        return 1 + helper(root.left) + helper(root.right);

    }

}

**Style 2: void return + global variable**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    int count = 0;

    public int countNodes(TreeNode root) {

        helper(root);

        return count;

    }

    private void helper(TreeNode root) {

        if(root == null) {

            return;

        }

        count++;

        helper(root.left);

        helper(root.right);

    }

}

**Solution 2: DFS + Complete Binary Tree attribute usage (10 min)**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public int countNodes(TreeNode root) {

        int l\_height = leftHeight(root);

        int r\_height = rightHeight(root);

        if(l\_height == r\_height) {

            return (int)Math.pow(2, l\_height) - 1;

        }

        return 1 + countNodes(root.left) + countNodes(root.right);

    }

    private int leftHeight(TreeNode root) {

        if(root == null) {

            return 0;

        }

        return 1 + leftHeight(root.left);

    }

    private int rightHeight(TreeNode root) {

        if(root == null) {

            return 0;

        }

        return 1 + rightHeight(root.right);

    }

}

**Refer to**

<https://www.cnblogs.com/grandyang/p/4567827.html>

我们还是要来利用一下完全二叉树这个条件，不然感觉对出题者不太尊重。通过上面对完全二叉树跟完美二叉树的定义比较，可以看出二者的关系是，**完美二叉树一定是完全二叉树**，**而完全二叉树不一定是完美二叉树**。那么这道题给的完全二叉树就有可能是完美二叉树，若是完美二叉树，节点个数很好求，为2的h次方减1，h为该完美二叉树的高度。若不是的话，只能老老实实的一个一个数结点了。思路是由 root 根结点往下，分别找最靠左边和最靠右边的路径长度，如果长度相等，则证明二叉树最后一层节点是满的，是满二叉树，直接返回节点个数，如果不相等，则节点个数为左子树的节点个数加上右子树的节点个数再加1(根节点)，其中左右子树节点个数的计算可以使用递归来计算

<https://leetcode.com/problems/count-complete-tree-nodes/discuss/61958/Concise-Java-solutions-O(log(n)2)>

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**

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;

}

}

**Refer to**

<https://github.com/lampardchelsea/hello-world/blob/master/leetcode/Tree/Document/Full_Compelete_Perfect_Tree.pdf>

**完全二叉树 (Complete Binary Tree)：**

A Complete Binary Tree （CBT) is a binary tree in which every level, except possibly the last, is completely filled,

and all nodes are as far left as possible.

对于一颗二叉树，假设其深度为d（d>1）。除了第d层外，其它各层的节点数目均已达最大值，且第d层所有节点从左向右连续地紧密排列，

这样的二叉树被称为完全二叉树；换句话说，完全二叉树从根结点到倒数第二层满足完美二叉树，最后一层可以不完全填充，其叶子结点都靠左对齐。

**完美二叉树 (Perfect Binary Tree)：**

A Perfect Binary Tree(PBT) is a tree with all leaf nodes at the same depth. All internal nodes have degree 2.

二叉树的第i层至多拥有 (2 ^ i) - 1 个节点数；深度为k的二叉树至多总共有 (2 ^ (k + 1)) -1 个节点数，而总计拥有节点数匹配的，称为“满二叉树”；

**完满二叉树 (Full Binary Tree):**

A Full Binary Tree (FBT) is a tree in which every node other than the leaves has two children.

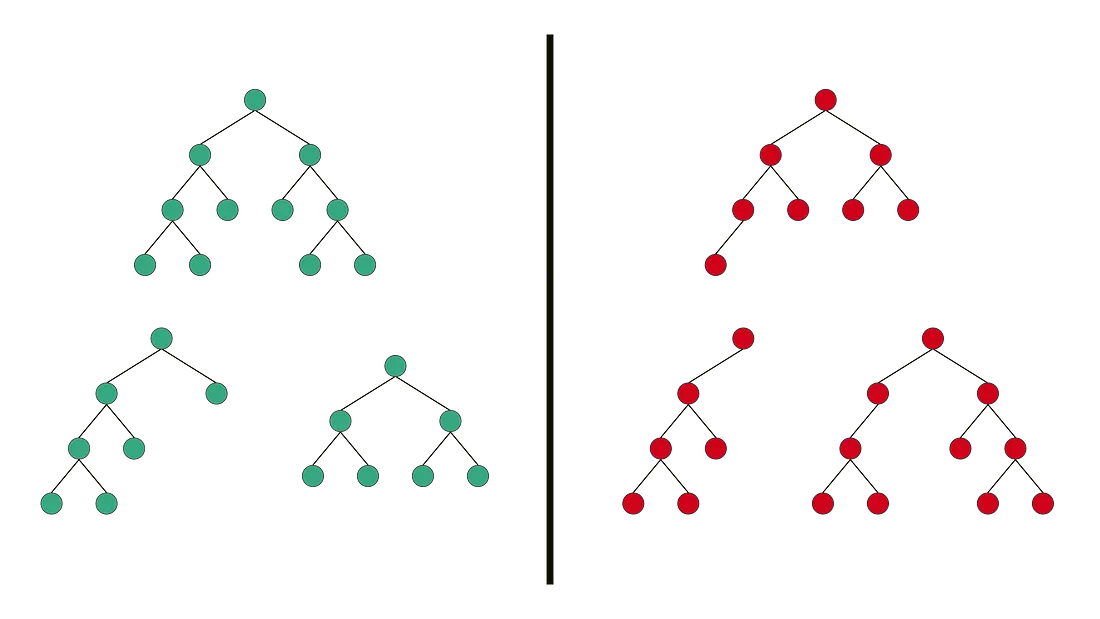
换句话说，所有非叶子结点的度都是2。（只要你有孩子，你就必然是有两个孩子）

**Refer to**

<https://towardsdatascience.com/5-types-of-binary-tree-with-cool-illustrations-9b335c430254>

**1. Full Binary Tree**

Full Binary Tree is a Binary Tree in which every node has 0 or 2 children.



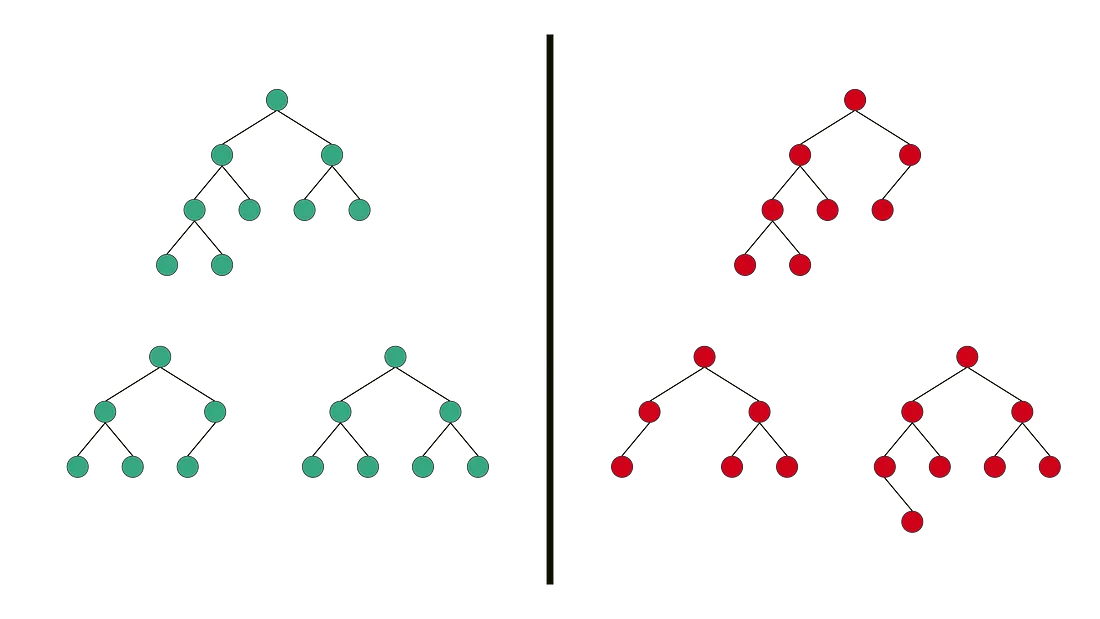
Valid and Invalid Structure of Full Binary Tree

**Interesting Fact:** For Full Binary Tree, following equation is always true.

Number of Leaf nodes = Number of Internal nodes + 1

**2. Complete Binary Tree**

Complete Binary Tree has all levels completely filled with nodes except the last level and in the last level, all the nodes are as left side as possible.

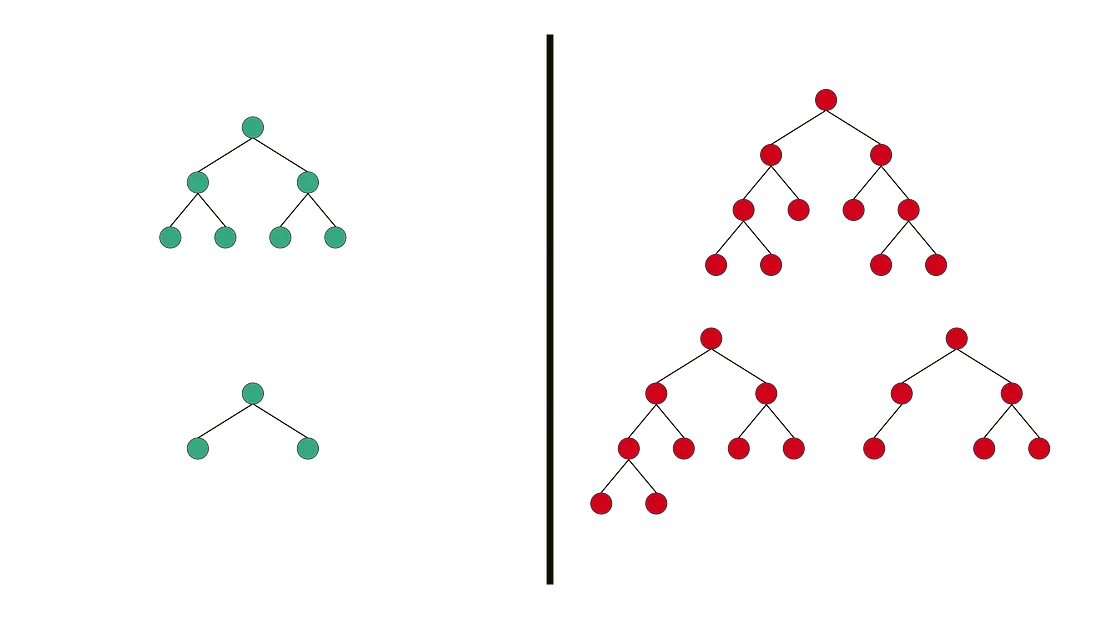


Valid and Invalid Structure of Complete Binary Tree

**Interesting Fact:** Binary Heap is an important use case of Complete Binary tree.

**3. Perfect Binary Tree**

Perfect Binary Tree is a Binary Tree in which all internal nodes have 2 children and all the leaf nodes are at the same depth or same level.

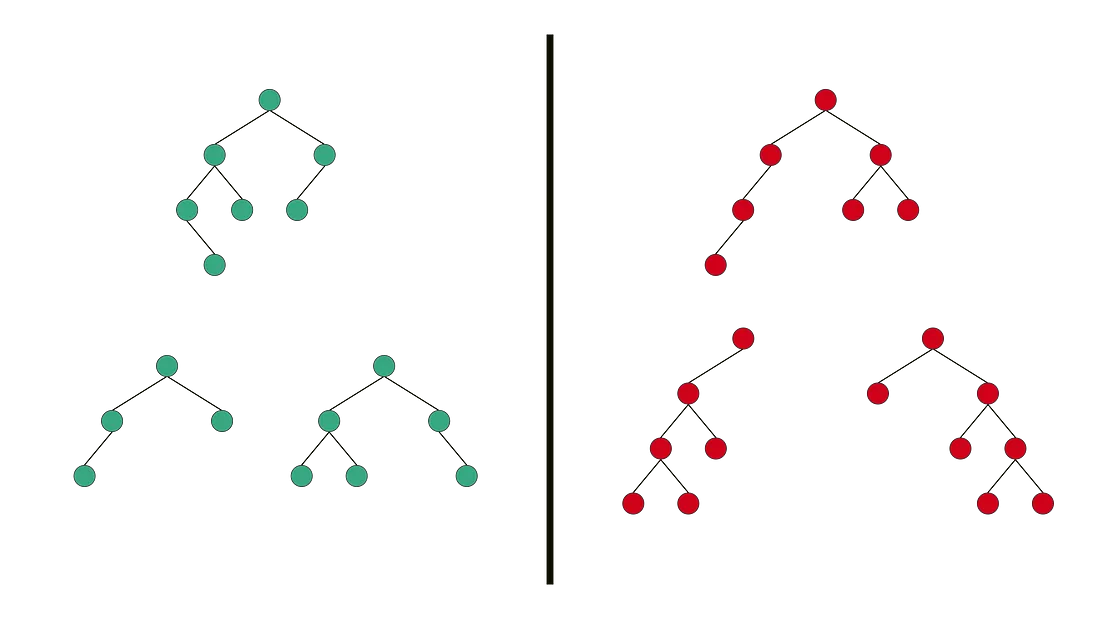


Valid and Invalid Structure of Perfect Binary Tree

**Interesting Fact:** Total number of nodes in a Perfect Binary Tree with height H is 2^H - 1.

**4. Balanced Binary Tree**

Balanced Binary Tree is a Binary tree in which height of the left and the right sub-trees of every node may differ by at most 1.

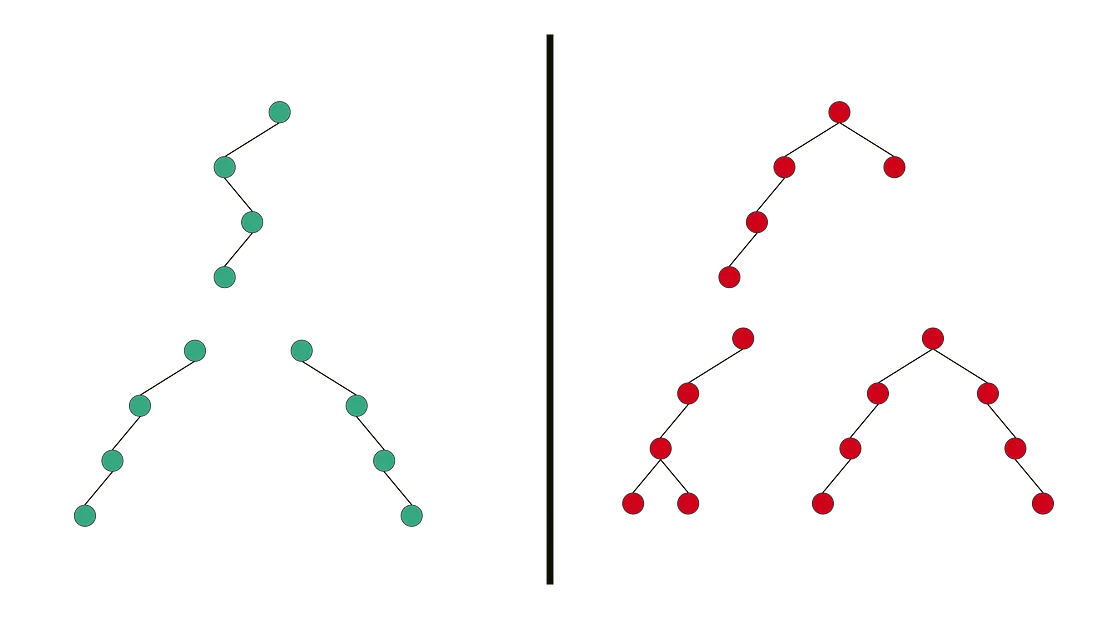


Valid and Invalid Structure of Balanced Binary Tree

**Interesting Fact:** AVL Tree and Red-Black Tree are well-known data structure to generate/maintain Balanced Binary Search Tree. Search, insert and delete operations cost O(log n) time in that.

**5. Degenerate(or Pathological) Binary Tree**

Degenerate Binary Tree is a Binary Tree where every parent node has only one child node.



Valid and Invalid Structure of Degenerate Binary Tree

**Interesting Fact:** Height of a Degenerate Binary Tree is equal to Total number of nodes in that tree.

**Refer to**

[L104.Maximum Depth of Binary Tree (Ref.L222)](note://DFEAE5072F224B7F85934A11F3381C70)

[L1448.Count Good Nodes in Binary Tree](note://WEB2d2969f579d16b378664369c32f6f5ee)

[L333.Largest BST Subtree (Ref.L98,L222)](note://27750017E06842BB91F3070EF9BC6DFB)