给出一个**完全二叉树**，求出该树的节点个数。

说明：

完全二叉树的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2h 个节点。

示例:

输入：

```
    1
   / \
  2   3
 / \  /
4  5 6
```

输出： 6

分析，自己想的时候，用了最暴力的直接遍历全树来解，然后果不其然，时间超限，这题看了别人的方法，遍历左右子树最深的度，相等，则这个子树为完二叉树，值等于2的度次幂-1，若不是，遍历左右子树，在判断。该方法的好处在于若是左右子树不是完二叉树，那么进入子树时，左子树或者右子树必有一个为完二叉树，减少了对这部分的计算。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
   int countNodes(TreeNode* root) {
        if(root == NULL) return 0;
        int lheight = getLeftHeight(root);//(1)左子树最左边高度
        int rheight = getRightHeight(root);//(2)右子树最右边高度
        //(3)判断是否为满二叉树
        if(lheight == rheight)
        {//满二叉树。节点一共有2^h - 1个，h为高度
            return (1<<lheight) - 1;//左移一位相当于乘2,2^h即等于1<<h！！！
        }
        else
        {//非满二叉树。递归求左右子树节点数+根节点
            return countNodes(root->left) + countNodes(root->right)
+1;//加上根节点
```

```
        }
    }
    //递归版本
    int getLeftHeight(TreeNode* root)
    {//得到完全二叉树左子树最左边深度（高度）
        return root==NULL?0:1+getLeftHeight(root->left);
    }
    int getRightHeight(TreeNode* root)
    {//得到完全二叉树右子树最右边深度（高度）
        return root==NULL?0:1+getRightHeight(root->right);
    }
};
```