# 110 Balanced Binary Tree

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

> a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1.

**Example 1:**

Given the following tree `[3,9,20,null,null,15,7]`:

```
    3
   / \
  9  20
    /  \
   15   7
```

Return true. **Example 2:**

Given the following tree `[1,2,2,3,3,null,null,4,4]`:

```
       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
```

Return false.

平衡二叉树，就是高度差< 1

首先介绍从上到下的方法。每棵树的高度是取max{left，right},如果向下的话，再+1

这种方法，每次需要判断每个节点的下面所有节点O(N),有N个节点，时间复杂度是O(N^2)

```
/**
 * 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:
    /// 求每棵树的深度
    int depth(TreeNode* root){
        if( root == NULL ) return 0;

        return max(depth(root->left),depth(root->right)) + 1;       //
return the depth of a tree.
    }

    bool isBalanced(TreeNode* root) {
        if(root == NULL) return true;

        int left = depth(root->left);
        int right = depth(root->right);

        return abs(left-right) <= 1 && isBalanced(root->left) &&
isBalanced( root->right );
    }
};
```

下面介绍一种从下到上的方法，

```
class solution {
public:
int dfsHeight (TreeNode *root) {
        if (root == NULL) return 0;

        int leftHeight = dfsHeight (root -> left);
        if (leftHeight == -1) return -1;    /// 一旦左边的有不满足的，会一直向上
传递
        int rightHeight = dfsHeight (root -> right);
        if (rightHeight == -1) return -1;

        if (abs(leftHeight - rightHeight) > 1)  return -1;  /// 从最下面的子树
开始判断
        return max (leftHeight, rightHeight) + 1;
    }
    bool isBalanced(TreeNode *root) {
        return dfsHeight (root) != -1;
    }
};
```