110. Balanced Binary Tree

作者：qianrong wu

思路一

采用top-down， 从上到下求节点的depth，然后看看节点是不是balance

解法一

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def isBalanced(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if not root:
            return True
        if abs(self.getDepth(root.left) - self.getDepth(root.right)) > 1:
            return False
        return self.isBalanced(root.left) and self.isBalanced(root.right)

    def getDepth(self, root):
        if not root:
            return 0
        return 1 + max (self.getDepth(root.left), self.getDepth(root.right))
```

思路二

buttom-up：先递归再做事 # 解法二

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def isBalanced(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        if self.checkDepth(root) == -1:
            return False
        else:
            return True

    def checkDepth(self, root):
        if not root:
            return 0
        left = self.checkDepth(root.left)
        right = self.checkDepth(root.right)
        if left == -1:
            return -1
        if right == -1:
            return -1
        if abs(left - right) > 1:
            return -1
        else:
            return 1 + max(left, right)
```

总结

方法一：Time Complexity - O(n2)， Space Complexity - O(n) 方法二：Time Complexity - O(n)， Space Complexity - O(n) 在不满足条件的时候就直接剪枝，提高了运算效率。