LeetCode Problems(Trees)

56⟶(145)Binary Tree postorder traversal

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def postorderTraversal(self, root: Optional[TreeNode]) -> Li
        s=[]
        def order(root,s):
          if root:
            order(root.left,s)
            order(root.right,s)
            s.append(root.val)
        order(root,s)
        return s
```

57⟶(94)Binary tree inorder traversal

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> Lis
        s=[]
        def order(root,s):
          if root:
            order(root.left,s)
            s.append(root.val)
            order(root.right,s)
        order(root,s)
        return s
```

58⟶(104)Maximum depth of binary tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        def height(root):
            if root:
                leftnode=height(root.left)
                rightnode=height(root.right)
                return max(leftnode,rightnode)+1
            else:
                return 0
        a=height(root)
        return a
```

59⟶(111)Minimum depth of binary tree

```
# Definition for a binary tree node.
# class TreeNode:
```

```
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def minDepth(self, root: Optional[TreeNode]) -> int:
        def height(root):
            if root:
                if root.left is None:
                    return height(root.right)+1
                if root.right is None:
                    return height(root.left)+1
                leftnode=height(root.left)
                rightnode=height(root.right)
                return min(leftnode,rightnode)+1
            else:
                return 0
        a=height(root)
        return a
```

60⟶(100)Same Tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[Tree
        def same(p,q):
            if p is None and q is None:
                return True
            if p is None or q is None:
                return False
            if p.val!=q.val:
                return False
            return same(p.left,q.left) and same(p.right,q.right
        return same(p,q)
```

61→(101)Symmetric tree

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def isSymmetric(self, root: Optional[TreeNode]) -> bool:
        def same(p,q):
            if p is None and q is None:
                return True
            if p is None or q is None:
                return False
            return (p.val==q.val) and same(p.left,q.right) and s
        return same(root.left,root.right)
```

62→(222)Count complete tree nodes

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
```

```
class Solution:
    def countNodes(self, root: Optional[TreeNode]) -> int:
        s=[]
        def order(root,s):
            if root:
                order(root.left,s)
                s.append(root.val)
                order(root.right,s)
        order(root,s)
        return len(s)
```