104. Maximum Depth of Binary Tree

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: TreeNode) -> int:
        if not root:
            return 0

        left_level = self.maxDepth(root.left) + 1
        right_level = self.maxDepth(root.right) + 1

        return max(left_level, right_level)
```

617. Merge Two Binary Trees

Merge Two Binary Trees

```
# 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 mergeTrees(self, t1: TreeNode, t2: TreeNode) -> TreeNode:
        if t1 is None:
            return t2

        if t2 is None:
            return t1

        if t1 and t2 is None:
            return 0

        t3=TreeNode(t1.val+t2.val)
        t3.left=self.mergeTrees(t1.left,t2.left)
        t3.right=self.mergeTrees(t1.right,t2.right)
        return t3
```

559. Maximum Depth of N-ary Tree

Maximum Depth of N-ary Tree

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val=None, children=None):
        self.val = val
        self.children = children
"""

class Solution:
    def maxDepth(self, root: 'Node') -> int:
        if not root:
            return 0

        max_depth=0
        for child in root.children:
            max_depth=max(max_depth,self.maxDepth(child))
        return 1+max_depth
```

589. N-ary Tree Preorder Traversal

N-ary Tree Preorder Traversal

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val=None, children=None):
        self.val = val
        self.children = children
"""

class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        if root is None:
            return None

        tree=[]
        tree.append(root.val)
        if root.children:
            for child in root.children:

                tree+=self.preorder(child)


        return tree
```

590. N-ary Tree Postorder Traversal

N-ary Tree Postorder Traversal

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val=None, children=None):
        self.val = val
        self.children = children
"""

class Solution:
    def postorder(self, root: 'Node') -> List[int]:
        if root is None:
            return None

        tree=[]
        if root.children:
            for child in root.children:
                tree+=self.postorder(child)

        tree+=[root.val]
        return tree
```

700. Search in a Binary Search Tree

Search in a Binary Search 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 searchBST(self, root: TreeNode, val: int) -> TreeNode:
        if root==None:
            return

        if root.val==val:
            return root

        if root.val>val:
            return self.searchBST(root.left,val)

        elif root.val<val:
            return self.searchBST(root.right,val)
```

897. Increasing Order Search Tree

Increasing Order Search 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 increasingBST(self, root: TreeNode) -> TreeNode:
        nodes = []
        def in_order(root):
            #base case
            if root is None:
                return
            in_order(root.left)
            nodes.append(root.val)  # save the node values
            in_order(root.right)

        in_order(root)
        for i in range(len(nodes)):     # iterate over the nodes and create a new tree
            if i == 0:
                Node = TreeNode(nodes[i])
                root = Node
            else:
                Node.right = TreeNode(nodes[i])
                Node = Node.right

        return root
```

938. Range Sum of BST

Range Sum of BST

```
# 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 rangeSumBST(self, root: TreeNode, L: int, R: int) -> int:
        if not root:
            return 0

        sum=root.val if root.val>=L and root.val<=R else 0

        return sum + self.rangeSumBST(root.left,L,R) + self.rangeSumBST(root.right,L,R)
```

965. Univalued Binary Tree

Univalued 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 isUnivalTree(self, root: TreeNode) -> bool:
        def helper(node,value):
            if not node:
                return True

            if node.val!=value:
                return False

            return helper(node.left,value) and helper(node.right,value)

        value=root.val
        return helper(root,value)
```