

```

1 # class TreeNode:
2 #     def __init__(self, x):
3 #         self.val = x
4 #         self.left = None
5 #         self.right = None
6 ▼ class Solution:
7     # 二叉树的前序遍历: 递归 (DFS: 根-左-右)
8 ▼     def preorderTraversal(self, root: TreeNode) -> List[int]:
9         res = []
10 ▼        def helper(root):
11            if not root: return # (叶子) 节点为空即可返回
12            res.append(root.val)
13            helper(root.left)
14            helper(root.right)
15        helper(root)
16        return res
17
18    # 二叉树的前序遍历: 迭代
19 ▼    def preorderTraversal(self, root: TreeNode) -> List[int]:
20        if not root: return []
21        deque = collections.deque()
22        deque.append(root)
23        res = []
24 ▼        while deque:
25 ▼            for _ in range(len(deque)):
26                node = deque.pop()
27                res.append(node.val)
28                if node.right: deque.append(node.right)
29                if node.left: deque.append(node.left)
30        return res

```

#二叉树的中序遍历：递归（DFS：左-根-右）

```
def inorderTraversal(self, root: TreeNode) -> List[int]:  
    res = []  
    def helper(root):  
        if not root: return  
        helper(root.left)  
        res.append(root.val)  
        helper(root.right)  
    helper(root)  
    return res
```

#来一个超级吊的写法：

```
def helper(root):  
    return helper(root.left) + [root.val] + helper(root.right) if root else []
```

#二叉树的中序遍历：迭代

```
def inorderTraversal(self, root: TreeNode) -> List[int]:  
    res = []  
    stack = []  
    while root or stack:  
        while root:  
            stack.append(root)  
            root = root.left  
        root = stack.pop()  
        res.append(root.val)  
        root = root.right  
    return res
```

```
60 #二叉树的后序遍历 (DFS: 左-右-根)
61 def postorderTraversal(self, root: TreeNode) -> List[int]:
62     res = []
63     def helper(root):
64         if not root: return
65         helper(root.left)
66         helper(root.right)
67         res.append(root.val)
68     helper(root)
69     return res
70
71 #二叉树的后序遍历: 迭代
72 def postorderTraversal(self, root: TreeNode) -> List[int]:
73     if not root: return []
74     deque = collections.deque()
75     deque.append(root)
76     res = []
77     while deque:
78         for _ in range(len(deque)):
79             node = deque.pop()
80             res.append(node.val)
81             if node.left: deque.append(node.left)
82             if node.right: deque.append(node.right)
83     return res[::-1]
```

#二叉树的层序遍历 (DFS)

```
def levelOrder(self, root: TreeNode) -> List[List[int]]:
    res = []
    def helper(cur_layer, root):
        if not root: return []
        if len(res) < cur_layer: res.append([])
        res[cur_layer-1].append(root.val)
        if root.left: helper(cur_layer+1, root.left)
        if root.right: helper(cur_layer+1, root.right)
    helper(1, root)    #一般来说，树的顶层从第1层开始，不是第0层
    return res
```

#二叉树的层序遍历 (BFS)

```
def levelOrder(self, root: TreeNode) -> List[List[int]]:
    if not root: return []
    deque = collections.deque()
    deque.append(root)
    res = []
    while deque:
        cur_layer = []
        for _ in range(len(deque)):
            node = deque.popleft()
            cur_layer.append(node.val)
            if node.left: deque.append(node.left)
            if node.right: deque.append(node.right)
        res.append(cur_layer)
    return res
```