Binary Tree

```
"""Tree 的题目一般先考虑 DFS，再考虑 BFS"""

""" DFS 思路
1. 画 DFS 递归调用树
2. 思考 DFS 节点之间的信息传递
    1. 向下👇传递信息：dfs() 的输入
    2. 向上👆传递信息：dfs() 的返回值
    3. 用全局变量保存全局信息
"""

"""DFS @ List"""
def dfs(node):
    dfs(node.next)

"""DFS @ Tree"""
def dfs(node):
    dfs(node.left)
    dfs(node.right)

""" BFS 思路
1. 适用于分层访问
"""

"""BFS @ List（其实就是指针遍历）"""
bfs = head
while bfs:
    bfs = bfs.next

"""BFS @ Tree"""
bfs = collections.deque([root])
while bfs:
    sz = len(bfs)
    for _ in range(sz):
        cur = bfs.popleft()
        if cur.left: bfs.append(cur.left)
        if cur.right: bfs.append(cur.right)
```

DFS

Is Same Tree
Is Symmetric Tree
Is Balanced Binary Tree
Is Binary Search Tree

Print Binary Tree
Sum of Left Leaves
Binary Tree Total Tilt
Count Univalue Subtrees
Most Frequent Subtree Sum
Lowest Common Ancestor of a Binary Tree
Count Complete Tree Nodes (lHeight, rHeight)
Smallest Subtree with all the Deepest Nodes
Find Leaves of Binary Tree **(distance to leaves)**
Second Minimum Node In a Binary Tree **(pruning)**

Invert Binary Tree
Add One Row to Tree
Merge Two Binary Trees
Construct Maximum Binary Tree
Binary Tree Upside Down (梳子形)

BFS @ Row/Level

Binary Tree Level Order Traversal

```python
def levelOrder(root):
    if not root: return []
    bfs = collections.deque([root])
    ret = []
    while bfs:
        sz = len(bfs)
        level = []
        for _ in range(sz):
            cur = bfs.popleft()
            level.append(cur.val)
            if cur.left: bfs.append(cur.left)
            if cur.right: bfs.append(cur.right)
        ret.append(level)
    return ret
```

Vertical Order Traversal `BFS: [(root, 0)]`
N-ary Tree Level Order Traversal
Maximum Width of Binary Tree
Average of Levels in Binary Tree
Find Largest Value in Each Tree Row
Find Bottom Left Tree Value
Binary Tree Right Side View
Populating Next Right Pointers in Each Node **(O(1) space using .next)**
Populating Next Right Pointers in Each Node II

DFS + Traversal

Binary Tree Preorder Traversal
Binary Tree Inorder Traversal
Binary Tree Postorder Traversal

Leaf-Similar Trees `遍历leaves`
Boundary of Binary Tree `遍历left-boundary, right-boundary, leaves`

DFS + (De)Serialization

Serialize and Deserialize Binary Tree
Construct String from Binary Tree (helper)
Construct Binary Tree from String

```python
###反/序列化###
class Codec:
    def serialize(self, root):
        vals = []
        def dfs(node):
            if node:
                vals.append(str(node.val))
                dfs(node.left)
                dfs(node.right)
            else:
                vals.append('#')
        dfs(root)
        return ' '.join(vals)

    def deserialize(self, data):
        vals = iter(data.split())  # python iterator
        def dfs():
            val = next(vals)
            if val == '#':
                return None
            node = TreeNode(int(val))
            node.left = dfs()
            node.right = dfs()
            return node
        return dfs()
```

DFS + Reconstruction

Construct Binary Tree from Preorder and Inorder Traversal
Construct Binary Tree from Preorder and Postorder Traversal
Construct Binary Tree from Inorder and Postorder Traversal

DFS + Flatten

Flatten Binary Tree to Linked List `return head, tail`
Flatten a Multilevel Doubly Linked List `dfs` `return tail`
Increasing Order Search Tree `dfs` `return head, tail`

DFS + Subtree

Find Duplicate Subtrees `subtree2id`

Subtree of Another Tree `subtree2id`
Equal Tree Partition `subtree_sums`
Pruning All Zero Subtrees

DFS + TreePath: (RL, PC, CPC ...)

(RL) Maximum Depth of Binary Tree
(RL) Minimum Depth of Binary Tree
(RL) Binary Tree Paths (Print Root to Leaf Paths)
(RL) Sum Root to Leaf Numbers

(RL) Path Sum I
(RL) Path Sum II
(PC) **Path Sum III**
(RL) Path Sum IV (pos2val, implicit tree)

(PC) Binary Tree Longest Consecutive Sequence
(CPC) **Binary Tree Longest Consecutive Sequence II**

(CPC) **Binary Tree Maximum Path Sum**
(CPC) **Longest Univalue Path**
(CPC) **Longest Path (Diameter) of Binary Tree**
(CPC) **Diameter of Binary Tree**

BFS @ TreeGraph

Build a undirected graph from a Binary Tree
All Nodes Distance K in Binary Tree
Closest Leaf in a Binary Tree
Sum of Distances in Tree

Implicit Tree

Kill Process (implicit tree)
Path Sum IV (pos2val, implicit tree)

Counting Trees

All Possible Full Binary Trees (int; list[node])
Unique Binary Search Trees (int; list[node])

N-ary Tree

N-ary Tree Preorder Traversal
(?)N-ary Tree Postorder Traversal
N-ary Tree Postorder Traversal
Encode N-ary Tree to Binary Tree & Decode back
Serialize and Deserialize N-ary Tree

```
"""如果不用 DFS, 如何对 Tree 先序/中序/后序遍历? """

"""USE separate left & right stack"""
def preorderTraversal(root):
    if not root: return []
    left, right = [root], []
    res = []
    while left or right:
        if left:  node = left.pop()
        else:  node = right.pop()
        res.append(node.val)
        if node.left: left.append(node.left)
        if node.right: right.append(node.right)
    return res

def postorderTraversal(root):
    if not root: return []
    left, right = [], [root]
    res = []
    while left or right:
        if right:  node = right.pop()
        else:  node = left.pop()
        res.append(node.val)
        if node.left: left.append(node.left)
        if node.right: right.append(node.right)
    return res[::-1]

def inorderTraversal(root):
    if not root: return []
    left, right = [root], []
    mid = []
    res = []
    while left or right:
        if left:  node = left.pop()
        else:  node = right.pop()
```

```
        if node:
            mid.append(node)
            left.append(node.left)
            right.append(node.right)
        elif mid:
            res.append(mid.pop().val)
    return res

"""Merge left & right stack"""
def preorderTraversal(root):
    if not root: return []
    stack = [root]
    res = []
    while stack:
        node = stack.pop()
        res.append(node.val)
        if node.right: stack.append(node.right)
        if node.left : stack.append(node.left)
    return res

def postorderTraversal(root):
    if not root: return []
    stack = [root]
    res = []
    while stack:
        node = stack.pop()
        res.append(node.val)
        if node.left : stack.append(node.left)
        if node.right: stack.append(node.right)
    return res[::-1]

def inorderTraversal(root):
    if not root: return []
    stack = [root]
    mid = []
    res = []
    while stack:
        node = stack.pop()
        if node:
            mid.append(node)
            stack.append(node.right)
            stack.append(node.left)
        elif mid:
            res.append(mid.pop().val)
    return res
```