

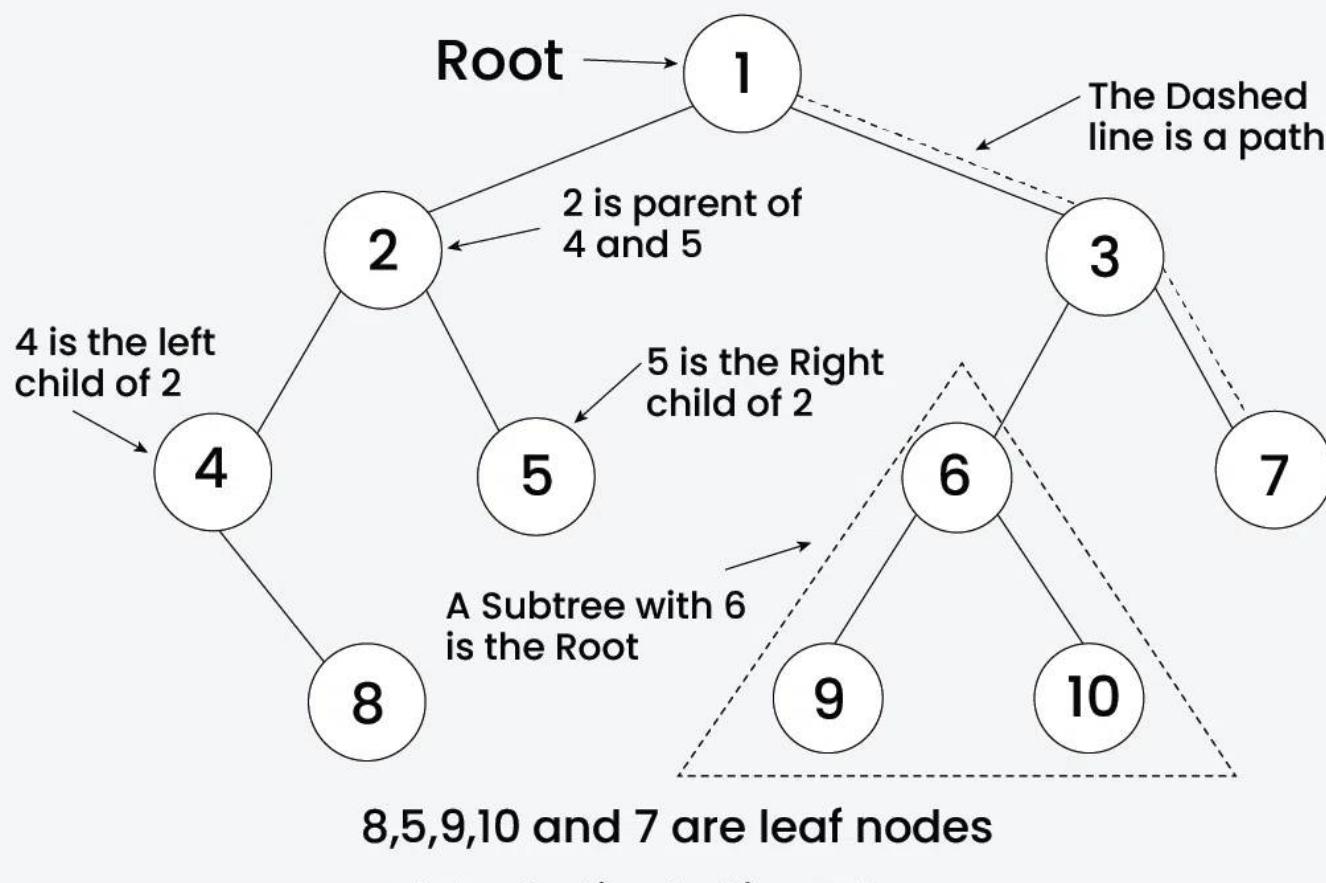
ChatGPT - DSA



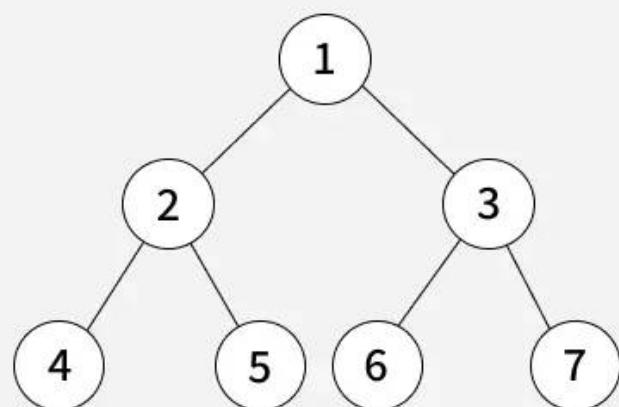
Tree Detailed Cheatsheet with all Patterns in Python DSA Mastery

Below is **the most complete, exam-ready, interview-ready Tree Cheatsheet** you'll ever need for **Python DSA mastery** — covering **all patterns, intuitions, templates, and common LC problems**.

TREE DETAILED CHEATSHEET (ALL PATTERNS) — PYTHON DSA MASTERY



Introduction to Binary Tree



Inorder Traversal						
4	2	5	1	6	3	7

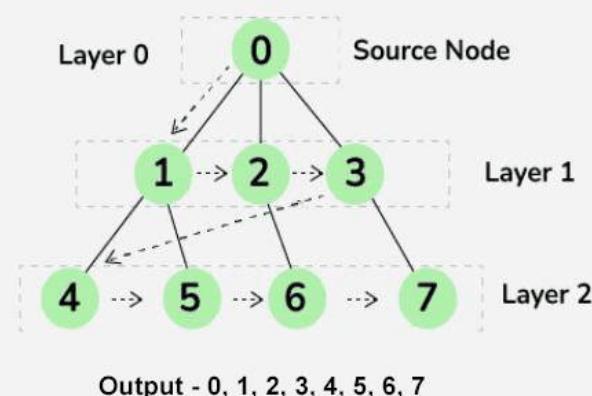
Preorder Traversal						
1	2	4	5	3	6	7

Postorder Traversal						
4	5	2	6	7	3	1

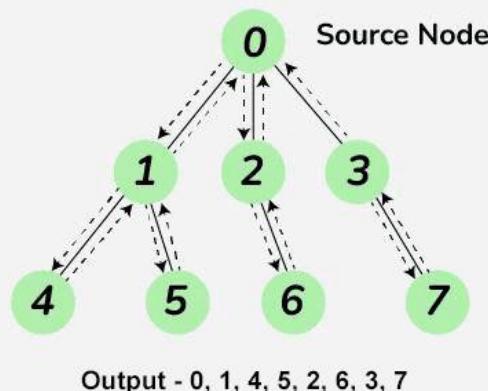
Level order Traversal						
1	2	3	4	5	6	7



BFS



DFS



Difference Between BFS and DFS



✓ 1. TREE BASICS & TERMINOLOGY

✓ Types of Trees

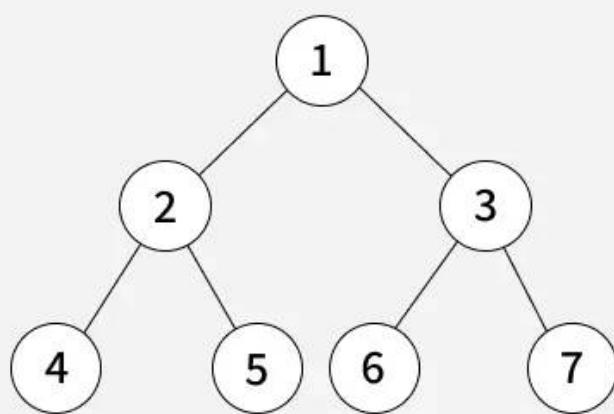
- **Binary Tree:** each node has ≤ 2 children
- **Binary Search Tree:** left $<$ root $<$ right
- **Complete Tree:** filled except last level left→right
- **Full Tree:** every node has 0 or 2 children
- **Perfect Tree:** all leaves at same level
- **N-ary Tree:** >2 children allowed
- **Trie:** prefix tree
- **Segment Tree / Fenwick Tree:** for range queries

✓ Core Node Definition (Binary Tree)

```
python

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

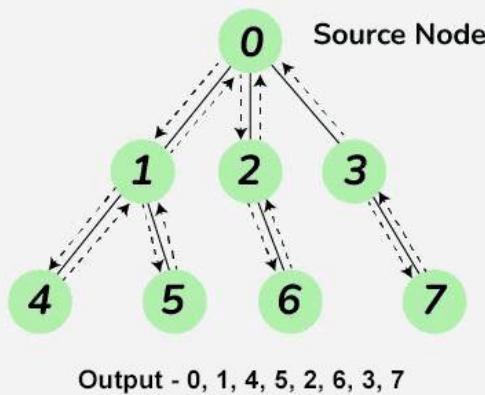
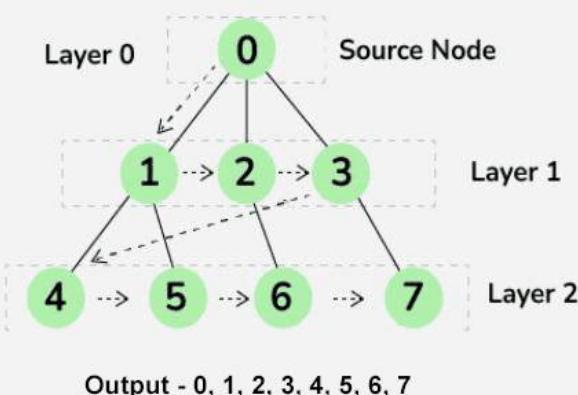
✓ 2. TRAVERSAL PATTERNS



Inorder Traversal
4 2 5 1 6 3 7
Preorder Traversal
1 2 4 5 3 6 7
Postorder Traversal
4 5 2 6 7 3 1
Level order Traversal
1 2 3 4 5 6 7



BFS



Difference Between BFS and DFS



2.1 DFS Traversals (Recursive) — CORE TEMPLATES

Preorder (Root → Left → Right)

```

python

def preorder(root):
    if not root: return
    print(root.val)
    preorder(root.left)
    preorder(root.right)
  
```

Inorder (Left → Root → Right)

```
python

def inorder(root):
    if not root: return
    inorder(root.left)
    print(root.val)
    inorder(root.right)
```

Postorder (Left → Right → Root)

```
python

def postorder(root):
    if not root: return
    postorder(root.left)
    postorder(root.right)
    print(root.val)
```

2.2 DFS Traversal (Iterative Templates)

Iterative Preorder

```
python

def preorder_iter(root):
    if not root: return []
    stack = [root]
    result = []

    while stack:
        node = stack.pop()
        result.append(node.val)
        if node.right: stack.append(node.right)
        if node.left: stack.append(node.left)

    return result
```

Iterative Inorder

```
python

def inorder_iter(root):
    stack, result = [], []
    cur = root

    while cur or stack:
        while cur:
            stack.append(cur)
            cur = cur.left

        cur = stack.pop()
        result.append(cur.val)
        cur = cur.right

    return result
```

2.3 BFS Traversal (Level Order)

Simple BFS

```
python

from collections import deque

def level_order(root):
    if not root: return []
    q = deque([root])
    result = []

    while q:
        level = []
        for _ in range(len(q)):
            node = q.popleft()
            level.append(node.val)

            if node.left: q.append(node.left)
            if node.right: q.append(node.right)

        result.append(level)

    return result
```

✓ 3. TREE PROBLEM-SOLVING PATTERNS (INTERVIEW GOLD)

★ PATTERN 1 — DFS RETURNING VALUES UPWARD (BOTTOM-UP)

Used for:

- Height of tree
- Diameter of tree
- Balanced tree
- Max path sum
- LCA
- Good nodes

TEMPLATE

```
python

def dfs(node):
    if not node:
        return base_value

    left = dfs(node.left)
    right = dfs(node.right)

    # combine left, right, and node.val
    return something
```

Example: Height

```
python

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

★ PATTERN 2 — DFS USING EXTERNAL GLOBAL (TOP-DOWN)

Used for:

- Sum of root-to-leaf paths
- Maximum depth (top-down version)
- Count paths
- Path sum problems

TEMPLATE

```
python

def dfs(node, path_value):
    if not node: return

    new_value = path_value + node.val

    if leaf(node):
        handle

    dfs(node.left, new_value)
    dfs(node.right, new_value)
```

★ PATTERN 3 — PATH PROBLEMS IN TREES

Path Sum I

```
python

def hasPathSum(root, target):
    if not root: return False
    if not root.left and not root.right:
        return target == root.val

    return hasPathSum(root.left, target - root.val) or \
           hasPathSum(root.right, target - root.val)
```

Max Path Sum (Hard)

```
python

class Solution:
    def maxPathSum(self, root):
        self.max_sum = float('-inf')

        def dfs(node):
```

```

if not node: return 0

left = max(dfs(node.left), 0)
right = max(dfs(node.right), 0)

self.max_sum = max(self.max_sum, node.val + left + right)

return node.val + max(left, right)

dfs(root)
return self.max_sum

```

★ PATTERN 4 — TREE + DP (Memoization on Nodes)

Used for:

- Rob house III
- Count BSTs (Catalan)
- DP on subtrees

Example: House Robber III

```

python

def rob(root):
    def dfs(node):
        if not node: return (0, 0)    # (rob, not_rob)

        left = dfs(node.left)
        right = dfs(node.right)

        rob_node = node.val + left[1] + right[1]
        not_rob = max(left) + max(right)

        return (rob_node, not_rob)

    return max(dfs(root))

```

★ PATTERN 5 — Lowest Common Ancestor (LCA)

Recursive LCA (Binary Tree)

```

python

def lca(root, p, q):
    if not root or root in (p, q):
        return root

    left = lca(root.left, p, q)
    right = lca(root.right, p, q)

    if left and right:
        return root

    return left or right

```

★ PATTERN 6 — Binary Search Tree Patterns

✓ Validate BST

```
python

def isValidBST(root):
    def dfs(node, low, high):
        if not node: return True
        if not (low < node.val < high): return False
        return dfs(node.left, low, node.val) and \
            dfs(node.right, node.val, high)
    return dfs(root, float('-inf'), float('inf'))
```

✓ Search, Insert, Delete

All operations use BST ordering rules.

★ PATTERN 7 — Serialization / Deserialization

Preorder + Null Markers

```
python

def serialize(root):
    if not root: return "N"
    return f"{root.val},{serialize(root.left)},{serialize(root.right)}"
```

```
python

def deserialize(data):
    values = iter(data.split(","))
    def helper():
        val = next(values)
        if val == "N": return None
        node = TreeNode(int(val))
        node.left = helper()
        node.right = helper()
        return node
    return helper()
```

★ PATTERN 8 — Tree Construction (Very Important)

Build Tree from Inorder + Preorder

```
python

def build(preorder, inorder):
    if not preorder: return None

    root = TreeNode(preorder[0])
    mid = inorder.index(preorder[0])

    root.left = build(preorder[1:1+mid], inorder[:mid])
```

```
root.right = build(preorder[1+mid:], inorder[mid+1:])
return root
```

★ PATTERN 9 — Tree Diameter (Longest Path)

Bottom-up DFS

```
python

def diameter(root):
    diameter = 0

    def dfs(node):
        nonlocal diameter
        if not node: return 0

        left = dfs(node.left)
        right = dfs(node.right)

        diameter = max(diameter, left + right)
        return 1 + max(left, right)

    dfs(root)
    return diameter
```

★ PATTERN 10 — N-ARY Tree Patterns

DFS

```
python

def dfs(root):
    if not root: return []
    res = [root.val]
    for child in root.children:
        res.extend(dfs(child))
    return res
```

BFS

```
python

def bfs(root):
    if not root: return []
    q = deque([root])
    result = []
    while q:
        node = q.popleft()
        result.append(node.val)
        for c in node.children:
            q.append(c)
    return result
```

★ PATTERN 11 — Trie (Prefix Tree)

python

```

class TrieNode:
    def __init__(self):
        self.children = {}
        self.end = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        cur = self.root
        for ch in word:
            cur = cur.children.setdefault(ch, TrieNode())
        cur.end = True

    def search(self, word):
        cur = self.root
        for ch in word:
            if ch not in cur.children:
                return False
            cur = cur.children[ch]
        return cur.end

```

★ PATTERN 12 — Morris Traversal (Inorder O(1) Space)

python

```

def morris_inorder(root):
    cur = root
    result = []

    while cur:
        if not cur.left:
            result.append(cur.val)
            cur = cur.right
        else:
            pred = cur.left
            while pred.right and pred.right != cur:
                pred = pred.right

            if not pred.right:
                pred.right = cur
                cur = cur.left
            else:
                pred.right = None
                result.append(cur.val)
                cur = cur.right

    return result

```

★ PATTERN 13 — Tree + Graph Conversion (Undirected)

Useful for:

- Burning tree
- Tree distance problems
- BFS from random node

```
python

from collections import defaultdict, deque

def treeToGraph(root):
    graph = defaultdict(list)

    def dfs(node, parent):
        if not node: return
        if parent:
            graph[node.val].append(parent.val)
            graph[parent.val].append(node.val)
        dfs(node.left, node)
        dfs(node.right, node)

    dfs(root, None)
    return graph
```

★ PATTERN 14 — Vertical / Horizontal Traversal

Vertical Order:

```
python

from collections import defaultdict, deque

def vertical_order(root):
    if not root: return []
    q = deque([(root, 0)])
    cols = defaultdict(list)

    while q:
        node, col = q.popleft()
        cols[col].append(node.val)
        if node.left: q.append((node.left, col - 1))
        if node.right: q.append((node.right, col + 1))

    return [cols[x] for x in sorted(cols)]
```

★ PATTERN 15 — Zigzag Traversal

```
python

def zigzag(root):
    if not root: return []
    q = deque([root])
    result = []
    left_to_right = True

    while q:
        level = []
        for _ in range(len(q)):
            node = q.popleft()
            level.append(node.val)
            if node.left: q.append(node.left)
            if node.right: q.append(node.right)

        result.append(level if left_to_right else level[::-1])
        left_to_right = not left_to_right

    return result
```

BONUS: MASTER TEMPLATE — SOLVE ANY TREE PROBLEM

```
python

def solve(root):
    def dfs(node):
        if not node:
            return base_case

        left = dfs(node.left)
        right = dfs(node.right)

        # logic using node.val, left, right
        return something

    return dfs(root)
```

👉 90% of LC tree questions are variations of this.

WANT MORE?

I can generate for you:

- 60 Tree Coding Templates**
- 50 Hard Tree Problems List**
- Binary Tree + Graph Hybrid Patterns**
- DP on Trees Mastermind Sheet**
- Tree Interview Question Bank (500+)**

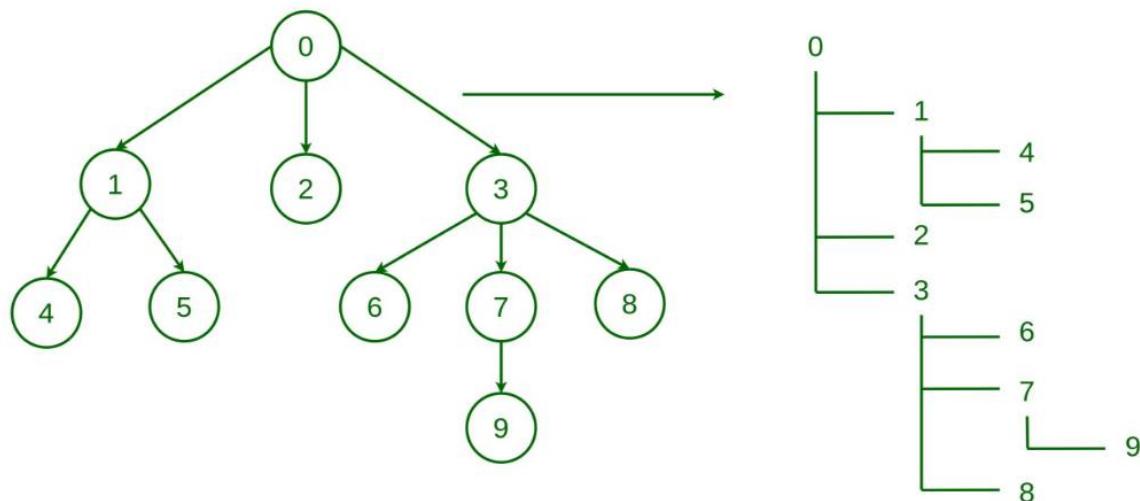
Just tell me “**Generate all templates**” or what you want next.



n-ary tree

Below is a **complete N-ARY TREE DSA MASTER CHEATSHEET** with **definitions, traversal patterns, templates, tricks, and interview patterns** — everything you need to master N-ary Trees.

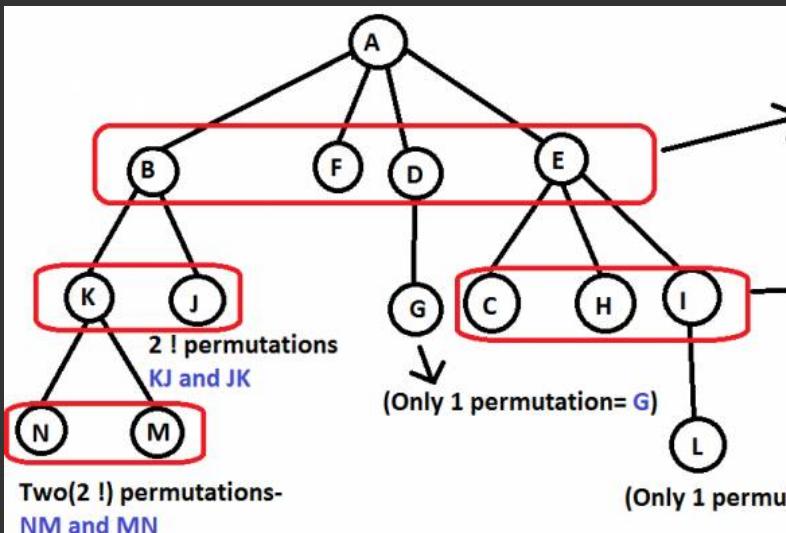
Print N-ary Tree Graphically

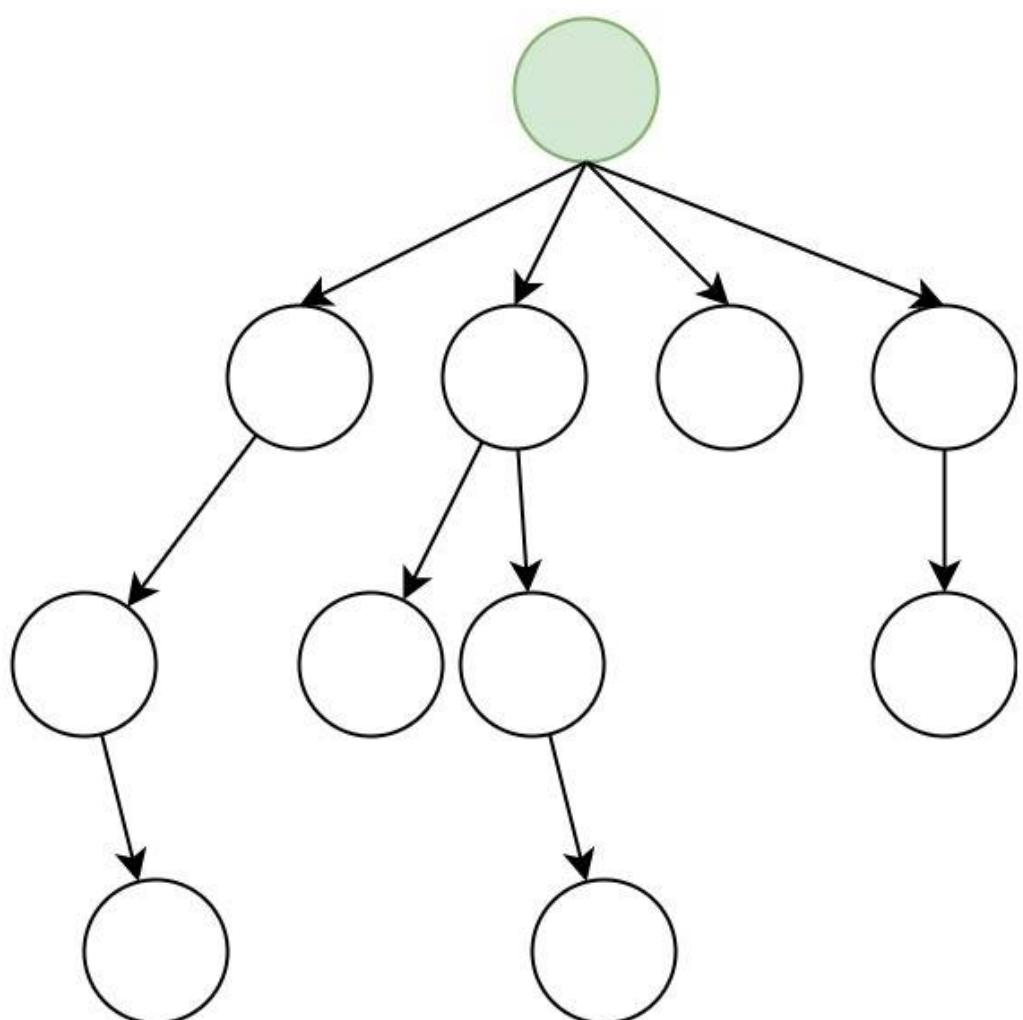


DG

Different permutations of children of A-
BFDE,BFED,BDFE,BDEF,BEDF.....24 (4 !)
permutations.

Different permutations of children of E ->
3 != 6 permutations. These are-
CHI
CIH
HCI
HIC
IHC
ICH





General Tree

牢固树立 N-ARY TREE — FULL CHEATSHEET (Python DSA Mastery)

✓ 1. N-ary Tree Node Definition

Standard LeetCode-style:

```
python

class Node:
    def __init__(self, val=None, children=None):
        self.val = val
        self.children = children if children is not None else []
```

✓ 2. Traversal Patterns (Core)

★ 2.1 Preorder Traversal (Root → Children)

Recursive (most common)

```
python

def preorder(root):
    result = []

    def dfs(node):
        if not node:
            return
        result.append(node.val)
        for child in node.children:
            dfs(child)

    dfs(root)
    return result
```

Iterative

```
python

def preorder_iter(root):
    if not root: return []
    result = []
    stack = [root]

    while stack:
        node = stack.pop()
        result.append(node.val)
        # reverse to maintain left-to-right order
        stack.extend(node.children[::-1])

    return result
```

★ 2.2 Postorder Traversal (Children → Root)

Recursive

```
python

def postorder(root):
    result = []

    def dfs(node):
        if not node:
            return
        for child in node.children:
            dfs(child)
        result.append(node.val)

    dfs(root)
    return result
```

Iterative

python

```
def postorder_iter(root):
    if not root: return []
    stack = [root]
    result = []

    while stack:
        node = stack.pop()
        result.append(node.val)
        stack.extend(node.children)

    return result[::-1]
```

★ 2.3 Level Order (BFS)

python

```
from collections import deque

def levelOrder(root):
    if not root: return []
    q = deque([root])
    result = []

    while q:
        level = []
        for _ in range(len(q)):
            node = q.popleft()
            level.append(node.val)
            for child in node.children:
                q.append(child)
        result.append(level)

    return result
```

✓ 3. Depth Calculations

★ 3.1 Maximum Depth

python

```
def maxDepth(root):
    if not root: return 0
    if not root.children: return 1
    return 1 + max(maxDepth(c) for c in root.children)
```

★ 3.2 Minimum Depth

python

```
def minDepth(root):
    if not root: return 0
    if not root.children: return 1
    return 1 + min(minDepth(c) for c in root.children)
```

✓ 4. Counting Nodes

```
python

def countNodes(root):
    if not root: return 0
    return 1 + sum(countNodes(c) for c in root.children)
```

✓ 5. Searching in N-ary Tree

DFS Search

```
python

def search(root, target):
    if not root: return False
    if root.val == target: return True
    return any(search(child, target) for child in root.children)
```

BFS Search

```
python

def search_bfs(root, target):
    if not root: return False
    from collections import deque
    q = deque([root])

    while q:
        node = q.popleft()
        if node.val == target:
            return True
        q.extend(node.children)

    return False
```

✓ 6. N-ary Tree Serialization / Deserialization

Most common format:

- **value, number_of_children, ...children...**

Serialize

```
python

def serialize(root):
    res = []

    def dfs(node):
        if not node:
            return
        res.append(str(node.val))
        res.append(str(len(node.children)))
        for c in node.children:
            dfs(c)

    dfs(root)
    return res
```

```
dfs(c)

dfs(root)
return " ".join(res)
```

Deserialize

```
python

def deserialize(data):
    if not data:
        return None

    vals = iter(data.split())
    def dfs():
        val = next(vals)
        size = int(next(vals))
        node = Node(int(val))
        for _ in range(size):
            node.children.append(dfs())
        return node

    return dfs()
```

✓ 7. Common Interview Patterns in N-ary Trees

★ Pattern A — DFS Bottom-Up (Return Values Upward)

Used for:

- Sum of subtree values
- Count leaf nodes
- Height problems

Example: Total sum of tree

```
python

def treeSum(root):
    if not root: return 0
    return root.val + sum(treeSum(c) for c in root.children)
```

★ Pattern B — DFS Top-Down (Path Problems)

Used for:

- Path sum
- Path count
- Accumulate values

Example: Root-to-leaf paths

```
python

def allPaths(root):
    paths = []

    def dfs(node, path):
        if not node: return
        path.append(node.val)

        if not node.children:
            paths.append(path[:])

        for c in node.children:
            dfs(c, path)

        path.pop()

    dfs(root, [])
    return paths
```

★ Pattern C — Degree, Height, Width Analysis

- Degree = number of children
- Height = longest depth
- Width = maximum nodes in a level (use BFS)

★ Pattern D — Tree → Graph Conversion

Useful for problems like:

- Burning tree
- Find nodes at distance K

```
python

from collections import defaultdict

def toGraph(root):
    graph = defaultdict(list)

    def dfs(node, parent):
        if not node: return
        if parent:
            graph[parent.val].append(node.val)
            graph[node.val].append(parent.val)
        for c in node.children:
            dfs(c, node)

    dfs(root, None)
    return graph
```

✓ 8. Building an N-ary Tree from Parent Array

Example input:

```
ini

parents = [-1, 0, 0, 1, 1, 2]
```

```
python

def buildTree(parents):
    nodes = [Node(i) for i in range(len(parents))]
    root = None

    for i, p in enumerate(parents):
        if p == -1:
            root = nodes[i]
        else:
            nodes[p].children.append(nodes[i])

    return root
```

✓ 9. Advanced N-ary Problems (LC Patterns)

★ 1. Clone N-ary Tree

```
python

def cloneTree(root):
    if not root: return None
    new = Node(root.val)
    new.children = [cloneTree(c) for c in root.children]
    return new
```

★ 2. Check Symmetry

```
python

def isMirror(a, b):
    if not a and not b: return True
    if not a or not b or a.val != b.val: return False

    if len(a.children) != len(b.children): return False

    for x, y in zip(a.children, reversed(b.children)):
        if not isMirror(x, y):
            return False
    return True

def isSymmetric(root):
    return not root or isMirror(root, root)
```

★ 3. N-ary Tree Diameter

```
python

def diameter(root):
    dia = 0
```

```

def dfs(node):
    nonlocal dia
    if not node: return 0

    heights = [dfs(c) for c in node.children]
    heights.sort(reverse=True)

    if len(heights) >= 2:
        dia = max(dia, heights[0] + heights[1])
    elif len(heights) == 1:
        dia = max(dia, heights[0])

    return 1 + (heights[0] if heights else 0)

dfs(root)
return dia

```

✓ 10. MASTER TEMPLATE — SOLVE 95% OF N-ary TREE PROBLEMS

python

```

def solve(root):

    def dfs(node):
        if not node:
            return base_case

        results = [dfs(child) for child in node.children]

        # combine node.val with results from children
        return combine(node.val, results)

    return dfs(root)

```

🎯 WANT MORE?

I can generate:

- ✓ 50 N-ary Tree Practice Problems (Easy → Hard)
- ✓ Visual diagrams for each algorithm
- ✓ N-ary Tree DP Patterns
- ✓ Interview-Ready Notes (1-page)

Just tell me what you want next:

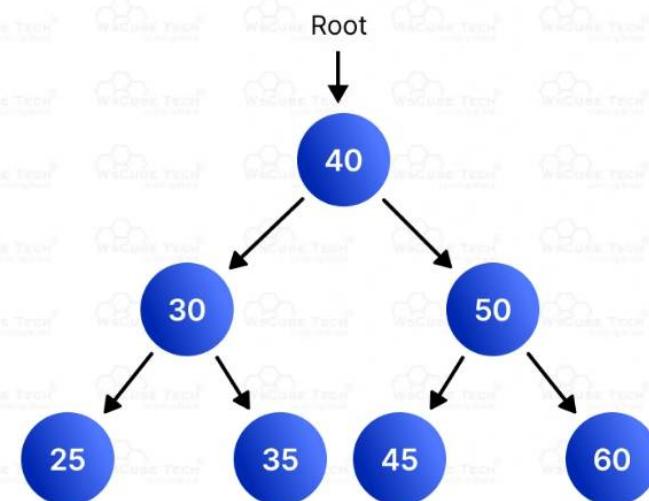
“N-ary DP”, “50 Problems”, “Interview Sheet”, or “Visual Notes”.



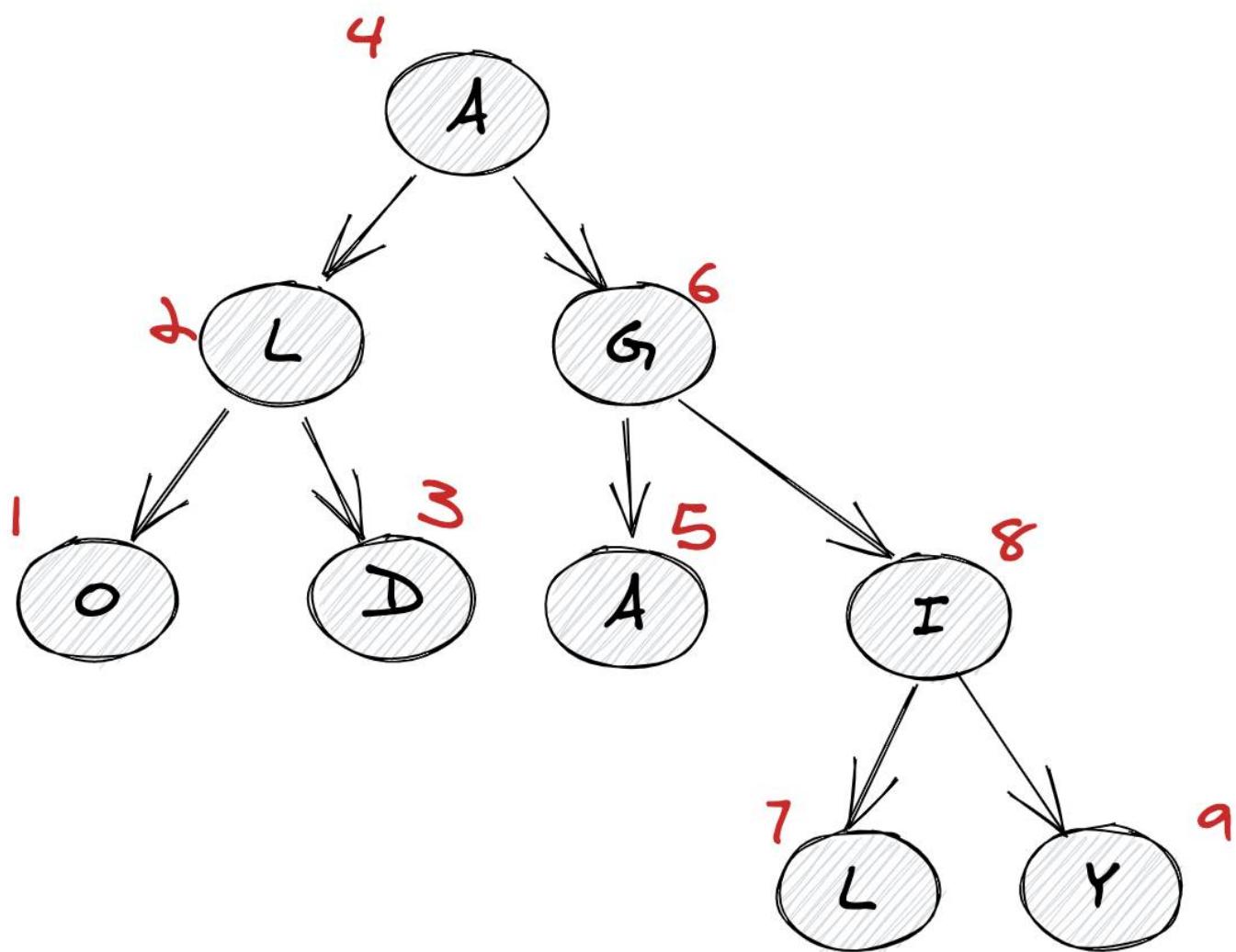
Binary Search Tree

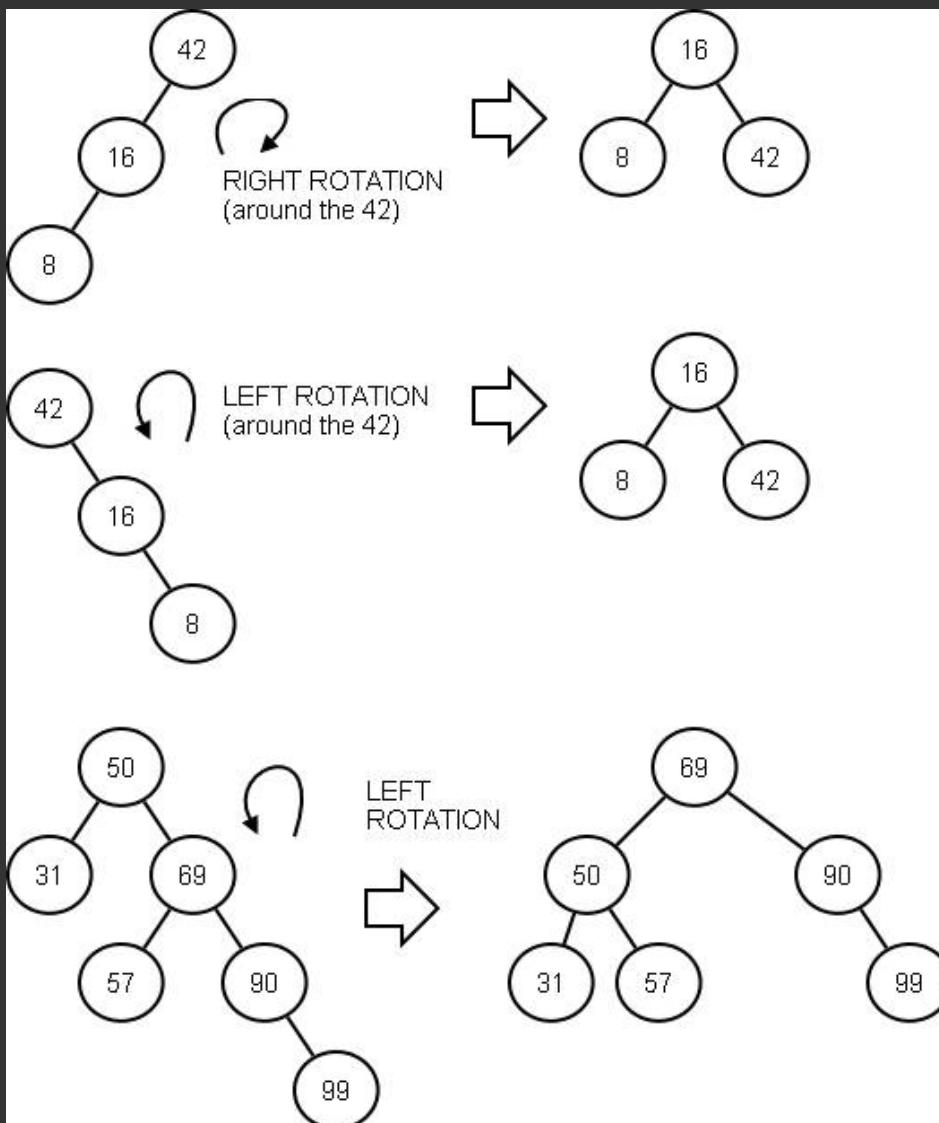
Below is the **ULTIMATE Binary Search Tree (BST) Cheatsheet** — everything you need for **DSA mastery** + **LeetCode interviews** with **patterns, templates, examples, and tricks**.

Binary Search Tree



IN-ORDER





🌳 BINARY SEARCH TREE (BST) — FULL CHEATSHEET (Python DSA Mastery)

✓ 1. BST Definition

BST property:

```
css
Left subtree < Node < Right subtree
```

Python Structure:

```
python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

✓ 2. Core BST Operations

★ 2.1 Search

Iterative (FAST, preferred):

```
python

def searchBST(root, val):
    while root:
        if val == root.val:
            return root
        root = root.left if val < root.val else root.right
    return None
```

Recursive:

```
python

def search(root, val):
    if not root or root.val == val: return root
    if val < root.val: return search(root.left, val)
    return search(root.right, val)
```

★ 2.2 Insertion

```
python

def insertIntoBST(root, val):
    if not root:
        return TreeNode(val)
    if val < root.val:
        root.left = insertIntoBST(root.left, val)
    else:
        root.right = insertIntoBST(root.right, val)
    return root
```

★ 2.3 Deletion (MOST ASKED)

Cases:

1. No child → remove node
2. One child → replace node with child
3. Two children → replace with **inorder successor** (smallest in right subtree)

```
python

def deleteNode(root, key):
    if not root:
        return None
```

```

if key < root.val:
    root.left = deleteNode(root.left, key)
elif key > root.val:
    root.right = deleteNode(root.right, key)
else:
    if not root.left:
        return root.right
    if not root.right:
        return root.left

    # find inorder successor
    cur = root.right
    while cur.left:
        cur = cur.left

    root.val = cur.val
    root.right = deleteNode(root.right, cur.val)

return root

```

✓ 3. BST Traversals

★ 3.1 Inorder Traversal = Sorted Output

```

python

def inorder(root):
    if not root: return []
    return inorder(root.left) + [root.val] + inorder(root.right)

```

Iterative:

```

python

def inorder_iter(root):
    stack = []
    cur = root
    result = []

    while cur or stack:
        while cur:
            stack.append(cur)
            cur = cur.left
        cur = stack.pop()
        result.append(cur.val)
        cur = cur.right

    return result

```

✓ 4. Validate BST (VERY IMPORTANT)

Check BST property using ranges:

```

python

def isValidBST(root):
    def dfs(node, low, high):
        if not node: return True
        if not (low < node.val < high):

```

```

        return False
    return dfs(node.left, low, node.val) and dfs(node.right, node.val, high)

return dfs(root, float('-inf'), float('inf'))

```

✓ 5. BST Min/Max

python

```

def findMin(root):
    while root.left:
        root = root.left
    return root.val

def findMax(root):
    while root.right:
        root = root.right
    return root.val

```

✓ 6. Lowest Common Ancestor (LCA) — BST Optimization

BST gives huge speed-up:

python

```

def lca(root, p, q):
    while root:
        if p.val < root.val and q.val < root.val:
            root = root.left
        elif p.val > root.val and q.val > root.val:
            root = root.right
        else:
            return root

```

✓ 7. Floor & Ceil in BST

Floor (largest \leq target)

python

```

def floor(root, x):
    ans = None
    while root:
        if root.val == x:
            return x
        if root.val > x:
            root = root.left
        else:
            ans = root.val
            root = root.right
    return ans

```

Ceil (smallest \geq target)

```
python
```

```
def ceil(root, x):
    ans = None
    while root:
        if root.val == x:
            return x
        if root.val < x:
            root = root.right
        else:
            ans = root.val
            root = root.left
    return ans
```

✓ 8. Kth Smallest in BST

Using Inorder Count

```
python
```

```
def kthSmallest(root, k):
    stack = []
    while True:
        while root:
            stack.append(root)
            root = root.left
        root = stack.pop()
        k -= 1
        if k == 0:
            return root.val
        root = root.right
```

✓ 9. Convert Sorted Array → BST

Balanced BST:

```
python
```

```
def sortedArrayToBST(nums):
    if not nums: return None
    mid = len(nums) // 2
    root = TreeNode(nums[mid])
    root.left = sortedArrayToBST(nums[:mid])
    root.right = sortedArrayToBST(nums[mid+1:])
    return root
```

✓ 10. BST Iterator (LC Premium)

Iterative inorder generator:

```
python
```

```
class BSTIterator:
    def __init__(self, root):
        self.stack = []
        self.push_left(root)
```

```

def push_left(self, node):
    while node:
        self.stack.append(node)
        node = node.left

def next(self):
    node = self.stack.pop()
    if node.right:
        self.push_left(node.right)
    return node.val

def hasNext(self):
    return len(self.stack) > 0

```

✓ 11. Balance a BST

Convert to sorted list → build balanced BST:

```

python

def balanceBST(root):
    arr = []

    def inorder(node):
        if not node: return
        inorder(node.left)
        arr.append(node.val)
        inorder(node.right)

    inorder(root)
    return sortedArrayToBST(arr)

```

⚡ 12. BST Important Properties

Property	Meaning
Inorder → Sorted	BST always outputs sorted values
Height	Best = $O(\log n)$, Worst = $O(n)$
Search	$O(\log n)$
Insert	$O(\log n)$
Delete	$O(\log n)$
Balanced BST	AVL, Red-Black → Guarantees $O(\log n)$

★ 13. MASTER TEMPLATE — BST DFS

```

python

def dfs(node):
    if not node:
        return base_case

    left = dfs(node.left)

```

```
right = dfs(node.right)
# combine
return something
```

★ 14. When to Use BST?

Use BST when you need:

- Ordered data
- Insert/Delete/Search in O(log n)
- Find floor/ceil
- Find predecessor/successor
- Kth smallest/largest
- Range queries

★ 15. BST Problems (LeetCode Set)

I can generate:

- **40 Easy**
- **40 Medium**
- **40 Hard BST Problems**
with solutions + templates.

🎁 Want More?

Say:

- 👉 "BST 60 Templates"
- 👉 "BST Visual Guide"
- 👉 "BST Interview Sheet"
- 👉 "BST Balanced Tree (AVL/RB) Cheatsheet"

Which one would you like next?



LC Problems to Master all Trees as Checklist

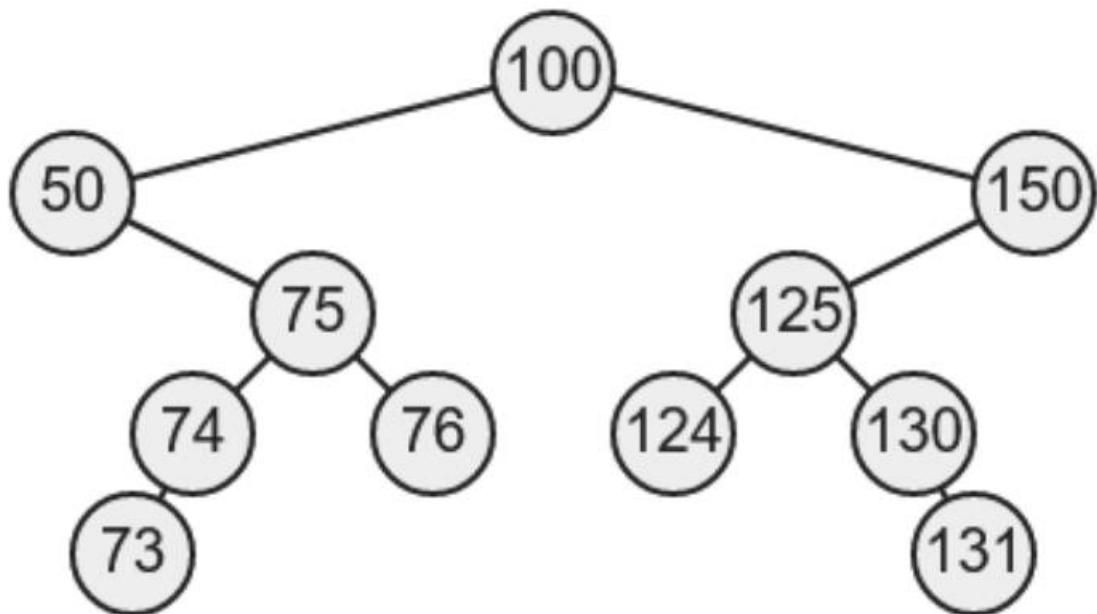
Below is the **ULTIMATE LeetCode TREE MASTERY CHECKLIST** — organized by **topic** → **pattern** → **difficulty**, covering **ALL binary tree, BST, n-ary, trie, and graph-tree hybrid problems** you must master for top-tier interviews (Google/Amazon/Meta).

This is a **complete roadmap**.

Use it as a **checklist** → mark each problem as you solve.

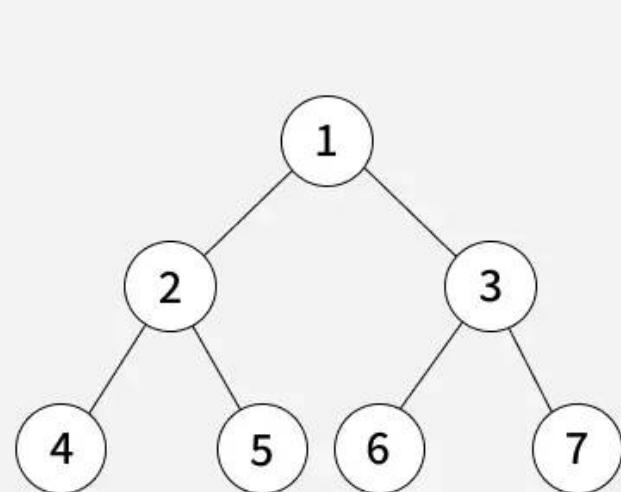
leetcode tree mastery checklist (full DSA roadmap)

Binary Search Tree Data Structure



1. Nodes on left side lower than root

2. Nodes on right side higher than root



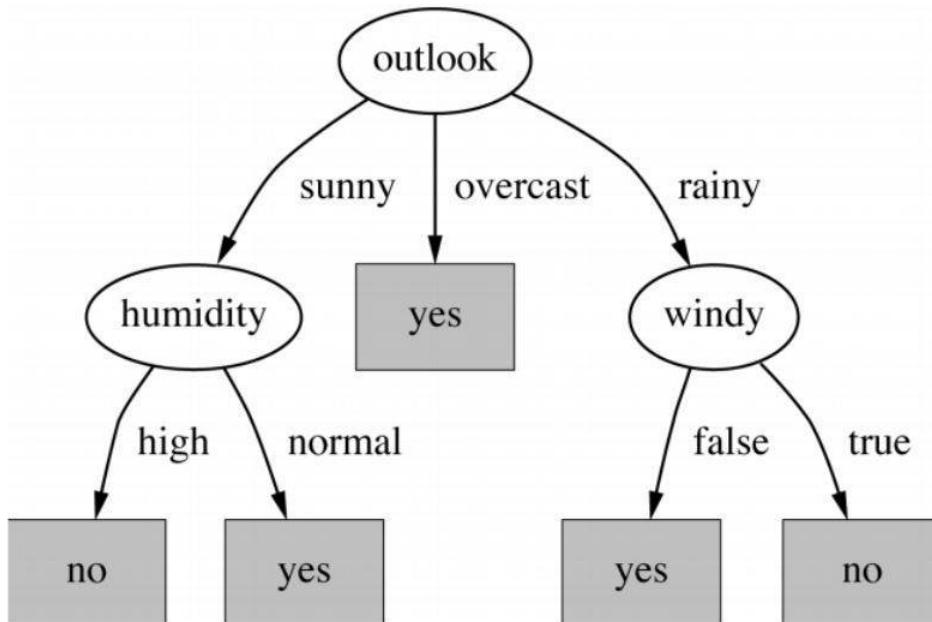
Inorder Traversal						
4	2	5	1	6	3	7

Preorder Traversal						
1	2	4	5	3	6	7

Postorder Traversal						
4	5	2	6	7	3	1

Level order Traversal						
1	2	3	4	5	6	7

Final decision tree



✓ SECTION A — BASIC TRAVERSALS (DFS/BFS)

Goal: Master Preorder, Inorder, Postorder, Level Order.

✓ MUST SOLVE (BEGINNER → EASY)

1. [144. Binary Tree Preorder Traversal](#)
2. [94. Binary Tree Inorder Traversal](#)
3. [145. Binary Tree Postorder Traversal](#)
4. [102. Binary Tree Level Order Traversal](#)
5. [107. Binary Tree Level Order II](#)
6. [103. Zigzag Level Order Traversal](#)
7. [429. N-ary Tree Level Order Traversal](#)
8. [589. N-ary Preorder Traversal](#)
9. [590. N-ary Postorder Traversal](#)

✓ SECTION B — DEPTH + HEIGHT + COUNT

Pattern: Return upward (bottom-up DFS)

-
- 10. **104. Maximum Depth of Binary Tree**
 - 11. **559. Maximum Depth of N-ary Tree**
 - 12. **111. Minimum Depth of Binary Tree**
 - 13. **110. Balanced Binary Tree**
 - 14. **543. Diameter of Binary Tree**
 - 15. **222. Count Complete Tree Nodes**
-

SECTION C — PATH PATTERNS

Pattern: Path sum, root-to-leaf paths, global maxima.

- 16. **112. Path Sum**
 - 17. **113. Path Sum II**
 - 18. **437. Path Sum III**
 - 19. **129. Sum Root to Leaf Numbers**
 - 20. **257. Binary Tree Paths**
 - 21. **124. Binary Tree Maximum Path Sum (HARD)**
 - 22. **988. Smallest String Starting From Leaf**
-

SECTION D — TREE CONSTRUCTION

MUST LEARN FOR ALL INTERVIEWS

- 23. **105. Construct Binary Tree from Preorder & Inorder**
 - 24. **106. Construct Binary Tree from Inorder & Postorder**
 - 25. **654. Maximum Binary Tree**
 - 26. **889. Construct Tree from Preorder & Postorder**
 - 27. **1008. Construct BST from Preorder**
-

SECTION E — TREE COMPARISON

- 28. **100. Same Tree**
 - 29. **101. Symmetric Tree**
 - 30. **572. Subtree of Another Tree**
 - 31. **617. Merge Two Binary Trees**
-

SECTION F — TREE SERIALIZATION / DESERIALIZATION

32. **297. Serialize and Deserialize Binary Tree (HARD)**

33. **449. Serialize and Deserialize BST**

34. **428. Serialize and Deserialize N-ary Tree (HARD)**

SECTION G — LOWEST COMMON ANCESTOR (LCA)

Critical Interview Pattern

Binary Tree LCA

35. **236. Lowest Common Ancestor of a Binary Tree**

BST LCA (easier)

36. **235. Lowest Common Ancestor of BST**

N-ary Tree LCA

37. **1676. LCA of N-ary Tree Nodes**

SECTION H — BINARY SEARCH TREE (BST)

Pattern: BST property — $\text{left} < \text{node} < \text{right}$

38. **700. Search in a BST**

39. **701. Insert into a BST**

40. **450. Delete Node in BST (HARD)**

41. **98. Validate BST**

42. **530. Minimum Absolute Difference in BST**

43. **783. Min Diff in BST**

44. **230. Kth Smallest in BST**

45. **538. BST to Greater Sum Tree**

46. **897. Increasing Order Search Tree**

47. **653. Two Sum IV — BST**

48. **1305. All Elements in Two BSTs**

49. **108. Convert Sorted Array to BST**

50. **109. Sorted List to BST**

SECTION I — ITERATOR PATTERN / ORDERED TREE STRUCTURES

51. **173. BST Iterator**

52. **341. Flatten Nested List Iterator** (concept related)

✓ SECTION J — GRAPH + TREE HYBRID PROBLEMS

Pattern: Convert tree → graph (adj list) → BFS/DFS

53. **863. All Nodes Distance K in Binary Tree (HARD)**

54. **2385. Amount of Time to Burn a Binary Tree**

55. **742. Closest Leaf in a Binary Tree**

✓ SECTION K — VERTICAL / HORIZONTAL / SPECIAL VIEWS

56. **987. Vertical Order Traversal of Binary Tree (HARD)**

57. **314. Binary Tree Vertical Order (Premium)**

58. **199. Right Side View**

59. **513. Bottom Left Tree Value**

60. **662. Maximum Width of Binary Tree**

✓ SECTION L — TREES + DP (DYNAMIC PROGRAMMING ON TREES)

Most advanced pattern

61. **337. House Robber III (DP on Trees)**

62. **124. Binary Tree Maximum Path Sum**

63. **333. Largest BST Subtree (HARD)**

64. **979. Distribute Coins in Binary Tree**

65. **968. Binary Tree Cameras (HARD)**

✓ SECTION M — ADVANCED TREE TRICKS

66. **865. Smallest Subtree with all Deepest Nodes**

67. **1609. Even Odd Tree**

68. **1302. Deepest Leaves Sum**

69. **1053. Is Graph a Tree? (tree properties)**

70. **1026. Maximum Difference Between Node & Ancestor**

▲ SECTION N — TRIE (PREFIX TREE)

Tries are considered Tree Problems in LC interviews.

-
- 71. [208. Implement Trie](#)
 - 72. [720. Longest Word in Dictionary](#)
 - 73. [212. Word Search II \(HARD, TOP-5\)](#)
 - 74. [1268. Search Suggestions System](#)
 - 75. [1032. Stream of Characters \(HARD\)](#)
-

SECTION O — HARD TREE PROBLEMS (TOP INTERVIEW PRIORITY)

If you want FAANG-level mastery, solve these:

- 76. [124. Max Path Sum](#)
 - 77. [297. Serialize/Deserialize](#)
 - 78. [236 + 1676 LCA](#)
 - 79. [863 Nodes Distance K](#)
 - 80. [987 Vertical Order](#)
 - 81. [968 Cameras](#)
 - 82. [333 Largest BST Subtree](#)
 - 83. [428 N-ary Serialization](#)
 - 84. [450 Delete BST Node](#)
 - 85. [212 Word Search II](#)
-

SECTION P — STRUCTURAL / FORMULA TREES

- 86. [331. Verify Preorder Serialization](#)
 - 87. [653. Two Sum IV](#)
 - 88. [671. Second Minimum Node](#)
-

FULL TREE MASTERY PROGRESSION

Phase 1 — Traversals

→ Problems 1-15

Phase 2 — Paths & Depth

→ Problems 16-22

Phase 3 — Construction

→ Problems 23–27

Phase 4 — Comparison & Serialization

→ Problems 28–34

Phase 5 — LCA

→ Problems 35–37

Phase 6 — BST

→ Problems 38–50

Phase 7 — Vertical/Horiz. Views

→ Problems 56–60

Phase 8 — DP on Trees

→ Problems 61–65

Phase 9 — Hard Trees

→ Problems 76–85

Phase 10 — Tries

→ Problems 71–75



Want this as a PDF or Notion-style checklist?

I can convert this into:

- A printable **PDF**
- A **tickable checklist table**
- A **Notion template with progress tracking**
- A **difficulty-based study plan (7–30 days)**

Just tell me:

“Make this a PDF” or “Make a Notion checklist”.