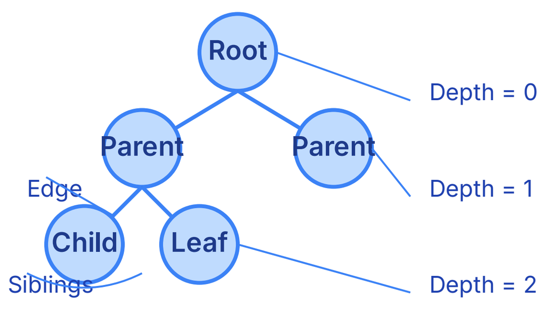
Trees in Data Structures Cheat Sheet

Essential Concepts, Traversals & Interview Highlights

What Is a Tree?

A non-linear, hierarchical data structure made of nodes connected by edges, with no cycles.



- R **Root:** Top node
- P **Parent:** Node with children
- C **Child:** Node with parent
- L **Leaf:** Node with no children
- S **Siblings:** Same parent
- H **Height:** Longest path to leaf
- D **Depth:** Distance from root

Types of Trees

Type	Structure	Use Case	Visual
Binary Tree	Max 2 children per node	Basic hierarchical structure	Basic structure
Binary Search Tree	Left < Root < Right	Fast search, insertion, deletion	Ordered
AVL Tree	Self-balancing BST (height difference ≤ 1)	Guaranteed O(log n) operations	Balanced
Heap	Complete binary tree with heap property (min/max)	Priority queues, heap sort	Min-heap
Trie	Character-wise tree for strings	Autocomplete, spell checking, prefix matching	Prefix tree

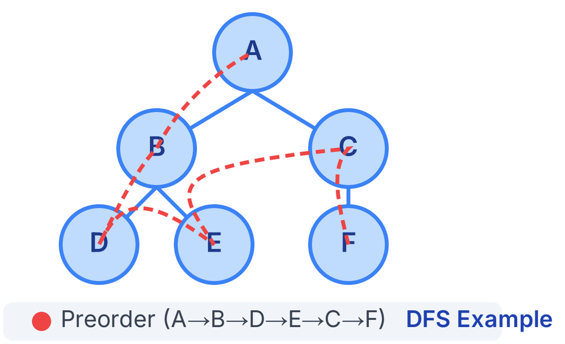
Traversal Methods

Depth-First Search (DFS)

`Preorder: Root → Left → Right`
Visit node before children (A-B-D-E-C-F)

`Inorder: Left → Root → Right`
Visit left, then node, then right (D-B-E-A-F-C)

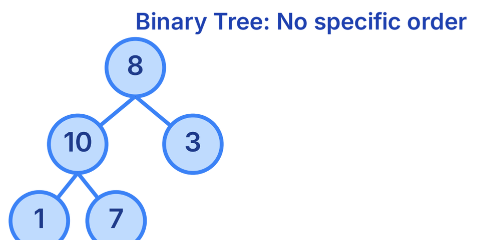
`Postorder: Left → Right → Root`
Visit node after children (D-E-B-F-C-A)

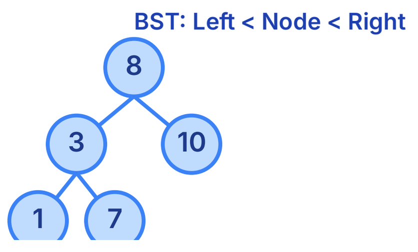


Breadth-First Search (BFS)

`Level-order: Level by level, left to right`
Visit all nodes at same depth (A-B-C-D-E-F)

⚔ Binary Tree vs Binary Search Tree





Feature	Binary Tree	Binary Search Tree
Node Order	No specific ordering rule	Left < Root < Right
Search Efficiency	O(n) - must check all nodes	O(log n) - if balanced
Usage	Expression trees, Huffman coding	Searching, sorting, database indexing
Example	File system hierarchy	Dictionary lookup

Real-World Examples

File Explorer
Folders contain subfolders and files in a tree structure

HTML DOM
Elements nested within other elements form a tree

Google Autocomplete
Uses Trie data structure for fast prefix matching

Priority Queue in CPU
Uses Heap to efficiently manage process priorities

Common Interview Questions

1. **Lowest Common Ancestor in BST**
   Find the deepest node that is an ancestor of both given nodes
2. **Serialize/Deserialize Binary Tree**
   Convert tree to string and back without losing structure
3. **Invert a Binary Tree**
   Mirror the tree by swapping left and right children
4. **Balanced Tree Check**
   Check if height difference between subtrees is at most 1
5. **Tree Diameter**
   Find longest path between any two nodes in the tree
6. **Construct Tree from Traversals**
   Build tree from preorder and inorder traversal arrays

LeetCode Problem References:

- #104 Maximum Depth of Binary Tree
- #226 Invert Binary Tree
- #235 Lowest Common Ancestor of BST
- #110 Balanced Binary Tree
- #543 Diameter of Binary Tree
- #105 Construct Tree from Traversals

Quick Practice Tips

- ✓ **Use recursion diagrams to trace logic**
  Draw the call stack at each step to understand recursion
- ✓ **Always draw the tree!**
  Visualizing helps identify patterns and edge cases
- ✓ **Dry-run your traversals**
  Trace the path node by node to verify your algorithm
- ✓ **Know when to use DFS vs BFS**
  DFS for path problems, BFS for shortest path/level-based problems
- ✓ **Revise preorder/postorder often**
  These traversals are popular in interviews and applications
- ✓ **Check edge cases**
  Empty tree, single node, unbalanced tree, duplicate values

Quick Code Template (Recursive Tree Traversal):

```
function traverse(node) {
    if (node === null) return;

    // Preorder: Process node here
    console.log(node.val);

    traverse(node.left);
    // Inorder: Process node here

    traverse(node.right);
    // Postorder: Process node here
}
```

Kaashiv Infotech

Trees in Data Structures Cheat Sheet v1.0
For educational purposes only

Print Cheat Sheet