Binary Trees & Tree Traversals in DSA: Beginner to Advanced Guide

Subtitle: Learn how to represent trees, perform traversals, and solve tree-based problems efficiently.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Binary trees are hierarchical data structures where each node has at most **two children**. Trees are used in **expression evaluation, searching, and hierarchical data**. This guide explains tree representation, traversals, and common problems.

Step 1: Binary Tree Basics

- **Node structure:** value + left child + right child
- **Root node:** topmost node
- **Leaf node:** node with no children
- **Height/Depth:** longest path from root to leaf
- **Applications:** expression trees, binary search trees, heaps

Example (Node in JS):

```
class Node {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
  }
}


let root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
```

Step 2: Tree Traversals

1. Preorder Traversal (Root → Left → Right)

```
function preorder(node) {
  if (!node) return;
  console.log(node.val);
  preorder(node.left);
  preorder(node.right);
}
```

2. Inorder Traversal (Left → Root → Right)

```
function inorder(node) {
  if (!node) return;
  inorder(node.left);
  console.log(node.val);
  inorder(node.right);
}
```

3. Postorder Traversal (Left → Right → Root)

```
function postorder(node) {
  if (!node) return;
  postorder(node.left);
  postorder(node.right);
  console.log(node.val);
}
```

4. Level Order / BFS Traversal

```
function levelOrder(root) {
  if (!root) return;
  let queue = [root];
  while (queue.length) {
    let node = queue.shift();
```

```
    console.log(node.val);
    if (node.left) queue.push(node.left);
    if (node.right) queue.push(node.right);
  }
}
```

Step 3: Common Binary Tree Problems

1. Height / Depth of a tree → DFS
2. Count nodes / leaf nodes
3. Diameter of a tree → longest path between two nodes
4. Lowest Common Ancestor (LCA)
5. Symmetric tree check → mirror recursion

Step 4: Binary Search Tree (BST) Basics

- Left child < node < right child
- Supports **search, insert, delete** in O(log n) average
- Traversals can produce **sorted order**

Example: Insert in BST

```
function insertBST(root, val) {
  if (!root) return new Node(val);
  if (val < root.val) root.left = insertBST(root.left, val);
  else root.right = insertBST(root.right, val);
  return root;
}
```

Step 5: Common Mistakes

- Confusing left/right child for BST property
- Forgetting null checks during recursion
- Not maintaining queue for level order
- Overlooking base cases in traversal

Step 6: Practice Plan for Trees

Day 1:

- Build tree, basic traversals → 5 problems

Day 2:

- Count nodes, height, diameter → 5 problems

Day 3:

- BST insert/search/delete → 5 problems

Day 4:

- LCA, symmetric tree, mirror tree → 5 problems

Day 5–6:

- Mixed tree problems → 10 problems

Mini Exercises

1. Print all leaf nodes
2. Find maximum value in a BST
3. Convert sorted array → BST
4. Level order traversal in zigzag pattern

Key Takeaways

- Binary trees are **hierarchical** and widely applicable
- Traversals: **preorder, inorder, postorder, level order**
- BSTs support **efficient search, insert, delete**
- Tree recursion is foundational for **graph and DP problems**
- Practice helps **visualize tree structures and recursion paths**

Visit haas.dev for more DSA binary tree guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app