Trees in DSA: Beginner to Advanced Guide

Subtitle: Learn tree data structures, traversal techniques, and solve common interview problems step by step.

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

Introduction

Trees are hierarchical data structures widely used in programming and interviews. From file systems to databases, understanding trees is essential. This guide covers basic to advanced concepts, traversal patterns, and important problems.

Step 1: What is a Tree?

A tree is a **hierarchical structure** consisting of nodes:

- **Root** → top node
- **Child** → nodes connected below
- **Parent** → node above
- **Leaf** → node with no children

Example:

```
    1
   / \
  2   3
 / \
4   5
```

Step 2: Types of Trees

- **Binary Tree** → Each node has at most 2 children
- **Binary Search Tree (BST)** → Left < Node < Right
- **Balanced Tree (AVL, Red-Black)** → Height difference ≤ 1
- **N-ary Tree** → Node can have N children

Step 3: Tree Traversal Techniques

1. Depth-First Search (DFS)

Preorder (Root → Left → Right)

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

Inorder (Left → Root → Right)

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

Postorder (Left → Right → Root)

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

2. Breadth-First Search (BFS / Level Order)

```
function bfs(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 4: Important Tree Problems

1. Height of a Tree

```
function height(root) {
  if (!root) return 0;
  return 1 + Math.max(height(root.left), height(root.right));
}
```

2. Count Nodes

```
function countNodes(root) {
  if (!root) return 0;
  return 1 + countNodes(root.left) + countNodes(root.right);
}
```

3. Diameter of a Tree

- Longest path between two nodes

```
let diameter = 0;

function dfs(node) {
  if (!node) return 0;
  let left = dfs(node.left);
  let right = dfs(node.right);
  diameter = Math.max(diameter, left + right);
  return 1 + Math.max(left, right);
```

}

Step 5: Binary Search Tree (BST) Operations

- **Insert Node** → O(log n) average
- **Delete Node** → O(log n) average
- **Search Node** → O(log n) average

Example: Insert in BST

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

Step 6: Common Mistakes

- Forgetting **null checks**
- Confusing preorder, inorder, postorder
- Not handling **BST edge cases**
- Ignoring **empty tree scenarios**

Step 7: Practice Plan for Trees

Day 1:

- Basic tree structure + DFS → 5 problems

Day 2:

- BFS + count nodes + height → 5–7 problems

Day 3:

- BST insert, search, delete → 5 problems

Day 4–5:

- Diameter, LCA, level order variations → 8–10 problems

Day 6:

- Mixed tree problems + revision → 10 problems

Mini Exercises

1. Find the sum of all nodes in a tree
2. Count leaf nodes
3. Check if a binary tree is height-balanced
4. Print right view of a binary tree

Key Takeaways

- Trees are **hierarchical and non-linear** data structures
- Master traversal techniques: **DFS & BFS**
- BSTs are key for **search and insertion efficiency**
- Practice with **height, diameter, and LCA problems**
- Consistent practice is essential for interviews

Visit haas.dev for more DSA guides, tree problems, and interview preparation resources.

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