

# Binary Trees! 🌳 (fundamental concept)

## What is a Binary Tree?

Think of a **Binary Tree** as a special kind of family tree. It starts with one ancestor at the top (the **root**). The special rule is that every person (a **node**) in this tree can have at most **two children**: a left child and a right child. That's it! Simple, right?

It's a non-linear data structure used to store data in a hierarchical way.

**Basic Structure of a Node:** A node in a binary tree typically contains three things:

1. **Data** (the value it holds)
  2. A pointer to the **left child**
  3. A pointer to the **right child**
- 

## Key Terminology 🧠

To speak the language of trees, you need to know these terms:

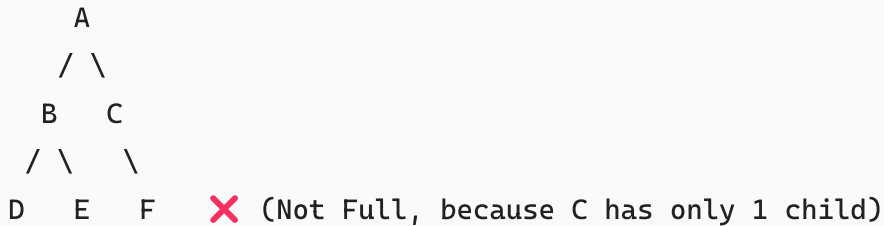
- **Node:** Any single element in the tree.
  - **Root:** The topmost node of the tree. It's the only node with no parent.
  - **Parent:** A node that has at least one child node.
  - **Child:** A node that has a parent node.
  - **Leaf Node (or External Node):** A node with no children. They are the endpoints of the tree.
  - **Internal Node:** A node with at least one child. Basically, any node that isn't a leaf.
  - **Edge:** The link connecting a parent to a child.
  - **Height of a Node:** The number of edges on the longest path from that node down to a leaf. The height of a leaf node is 0.
  - **Height of a Tree:** The height of the root node. An empty tree has a height of -1, and a tree with one node has a height of 0.
  - **Depth of a Node:** The number of edges from the root to that node. The depth of the root is 0.
  - **Level of a Node:** The depth of the node + 1. The level of the root is 1.
-

## Types of Binary Trees

Not all binary trees are the same! Here are some special types you'll encounter often:

### 1. Full Binary Tree

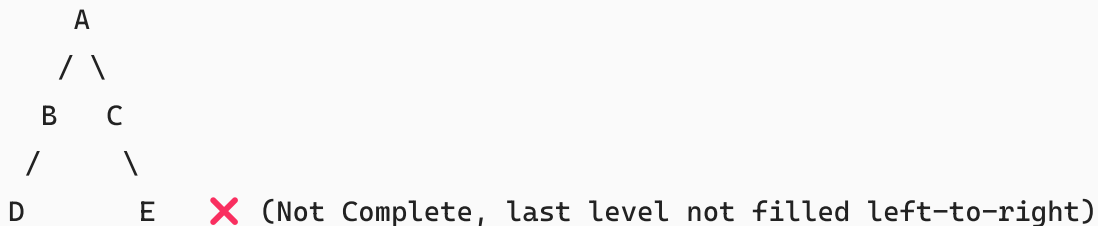
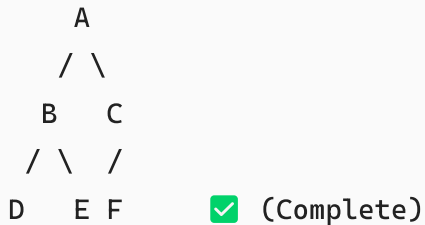
A tree where every node has either **0 or 2 children**. No node has only one child.



### 2. Complete Binary Tree

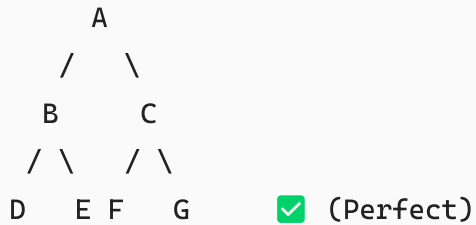
A tree where all levels are completely filled, except possibly the last level.

The last level has its nodes filled from **left to right**.



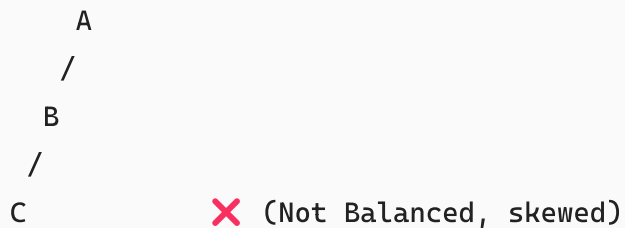
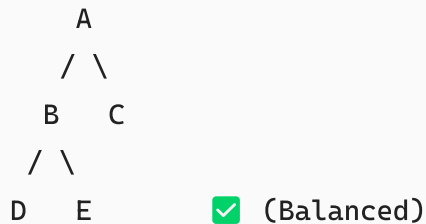
### 3. Perfect Binary Tree

A tree where all internal nodes have **2 children** and all leaf nodes are at the **same level**.



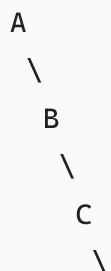
## 4. Balanced Binary Tree

A tree where the height difference between the left and right subtrees for any node is **not more than 1**.



## 5. Degenerate (Pathological) Tree

A tree where every parent node has only one child. It basically looks and behaves like a **Linked List**.



## Important Properties & Formulas 🧠

These formulas are your secret weapons for solving many problems quickly!

Let's say the height of the tree is  $h$  and the number of nodes is  $n$ .

1. **Maximum number of nodes** at any level  $l$  (where the root is at level 0) is:

$$2^l$$

- Level 0:  $2^0 = 1$  (the root)
- Level 1:  $2^1 = 2$
- Level 2:  $2^2 = 4$

2. **Maximum number of nodes** in a binary tree of height  $h$  is:

$$2^{h+1} - 1$$

- This occurs when the tree is a **Perfect Binary Tree**.

3. **Minimum possible height** (or minimum number of levels) for a binary tree with  $n$  nodes is:

$$\lfloor \log_2(n) \rfloor$$

- This occurs when the tree is as balanced and compact as possible.

4. **Minimum number of nodes** in a binary tree of height  $h$  is:

$$h + 1$$

- This happens in the worst case, a **Degenerate Tree** (like a linked list).

5. In any **non-empty Full Binary Tree**, the number of **leaf nodes** ( $L$ ) is equal to the number of **internal nodes** ( $I$ ) plus one:

$$L = I + 1$$

6. In a binary tree with  $n$  nodes, there will be exactly:

$$n - 1 \text{ edges.}$$

## Tree Traversal: Taking a Walk in the Woods 🚶

Traversal means visiting every node in the tree exactly once. There are two main ways to do this:

## 1. Depth-First Search (DFS)

This strategy goes as deep as possible down one path before backtracking.

- **In-order Traversal (Left, Root, Right)**

1. Go to the left subtree.
2. Visit the root.
3. Go to the right subtree.

- **Fun Fact:** In-order traversal of a Binary Search Tree (BST) gives you the nodes in sorted order! 🥳

- **Pre-order Traversal (Root, Left, Right)**

1. Visit the root.
2. Go to the left subtree.
3. Go to the right subtree.

- **Use Case:** Useful for creating a copy of the tree or getting an expression prefix notation.

- **Post-order Traversal (Left, Right, Root)**

1. Go to the left subtree.
2. Go to the right subtree.
3. Visit the root.

- **Use Case:** Perfect for deleting nodes from a tree (you delete children before the parent).

## 2. Breadth-First Search (BFS)

This strategy explores the tree level by level, from top to bottom.

- **Level-order Traversal**

1. Visit all nodes at the current level from left to right.
2. Move to the next level.

- **Implementation:** This is usually done using a **Queue** data structure.
- **Use Case:** Excellent for finding the shortest path between two nodes in terms of edges.

---

## Properties of Traversal (The Reconstruction Puzzle) 🧩

This is a classic and very important interview topic!

- **The Golden Rule:** You can uniquely reconstruct a Binary Tree if you have its **In-order traversal** AND one of the other two (either **Pre-order** or **Post-order**).
  - **Why?**
    - The **Pre-order** traversal's first element is *a/ways* the **root** of the tree.
    - The **Post-order** traversal's last element is *a/ways* the **root** of the tree.
    - Once you know the root, you can look for it in the **In-order** traversal. Everything to the left of the root in the In-order array belongs to the left subtree, and everything to the right belongs to the right subtree.
  - You **cannot** reconstruct a unique tree from just Pre-order and Post-order traversals.
- 

## Other Important Concepts for DSA Problems

Keep these patterns in your toolkit. They are very common problem types!

- **Lowest Common Ancestor (LCA):** Finding the first shared ancestor for any two given nodes.
- **Diameter of a Binary Tree:** The longest path between any two nodes in the tree. This path may or may not pass through the root.
- **Views of a Binary Tree:**
  - **Left View:** The nodes visible when you look at the tree from the left side.
  - **Right View:** The nodes visible when you look from the right side.
  - **Top View / Bottom View:** The nodes visible when you look from directly above or below.
- **Check if a tree is a Binary Search Tree (BST):** A common validation problem where you need to check if for every node, all nodes in its left subtree are smaller and all nodes in its right subtree are larger.