Tree Data Structure

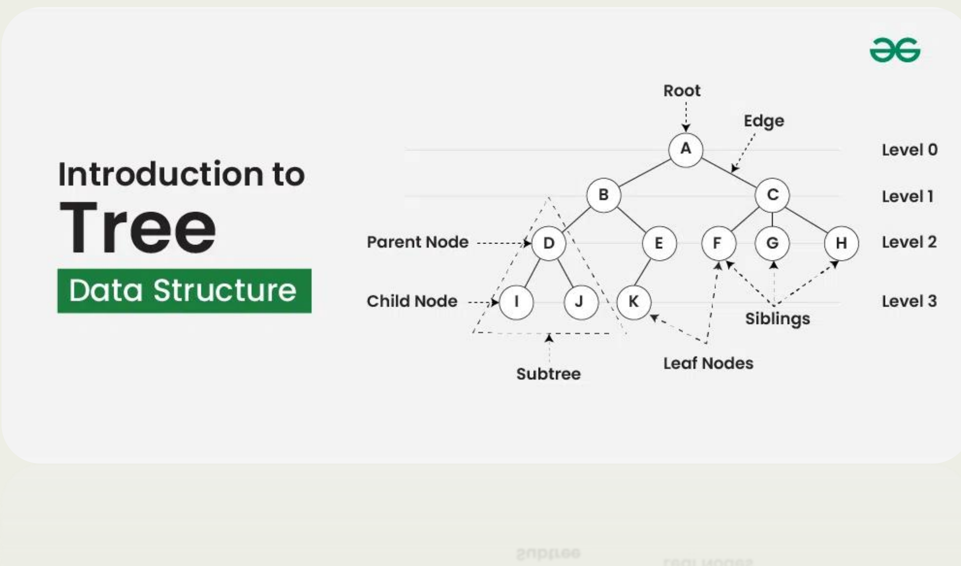
🌳 Introduction to Tree Data Structure

A **Tree** is a **non-linear**, **hierarchical data structure** that represents elements in a parent-child relationship.

📌 Real-World Examples:

- Folder structure in an Operating System
- Tag hierarchy in HTML/XML documents (like <html> as root, <head> and <body> as children)

📖 Basic Terminologies



1. **Root Node**:
   The topmost node of the tree, which has no parent.
   ➤ Example: A in a tree is the root.

2. **Parent Node**:
   A node that has one or more child nodes.
   ➤ Example: B is a parent of D and E.

3. **Child Node**:
   A node that descends from another node (its parent).
   ➤ Example: D and E are children of B.

4. **Leaf/External Node**:
   Nodes that do not have any children.
   ➤ Example: I, J, K, F, G, H.

5. **Internal Node**:
   Nodes that have at least one child.
   ➤ Example: B, C.

6. **Sibling**:
   Nodes that share the same parent.
   ➤ Example: D and E.

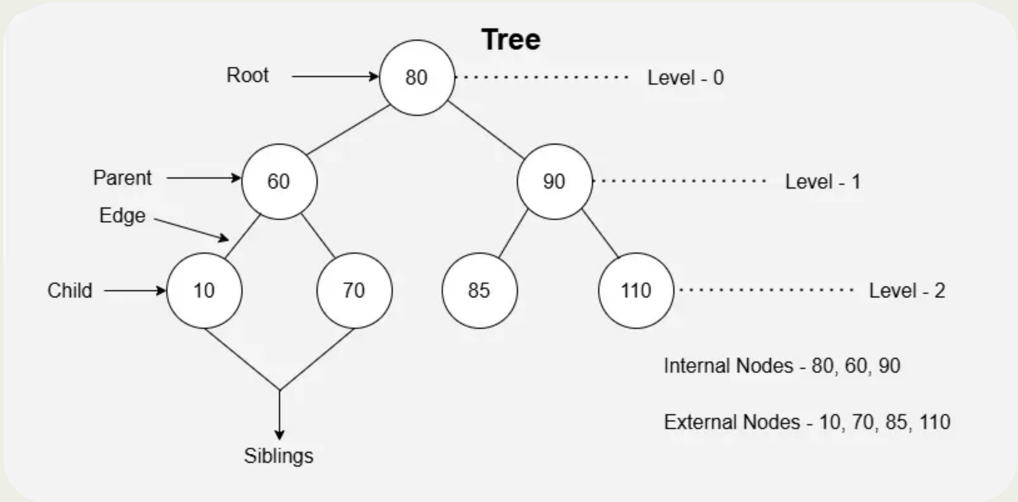
7. **Ancestor**:
   All nodes from the root to a given node (excluding the node itself).
   ➤ Example: Ancestors of E = A, B.

8. **Descendant**:
   All nodes below a given node.
   ➤ Example: Descendants of B = D, E.

9. **Level of a Node**:
   The number of edges from the root to that node.
   ➤ Root is always at level 0.

10. **Subtree**:
    A node and all its descendants form a subtree.

11. **Neighbor**:
    Parent and child nodes are considered neighbors.



🧠 Why Tree is a Non-Linear Data Structure?

Unlike arrays or linked lists that store data sequentially, trees **store data hierarchically**, with elements connected across different levels. Hence, it is a **non-linear structure**.

🧱 Representation of Tree

A tree is defined by:

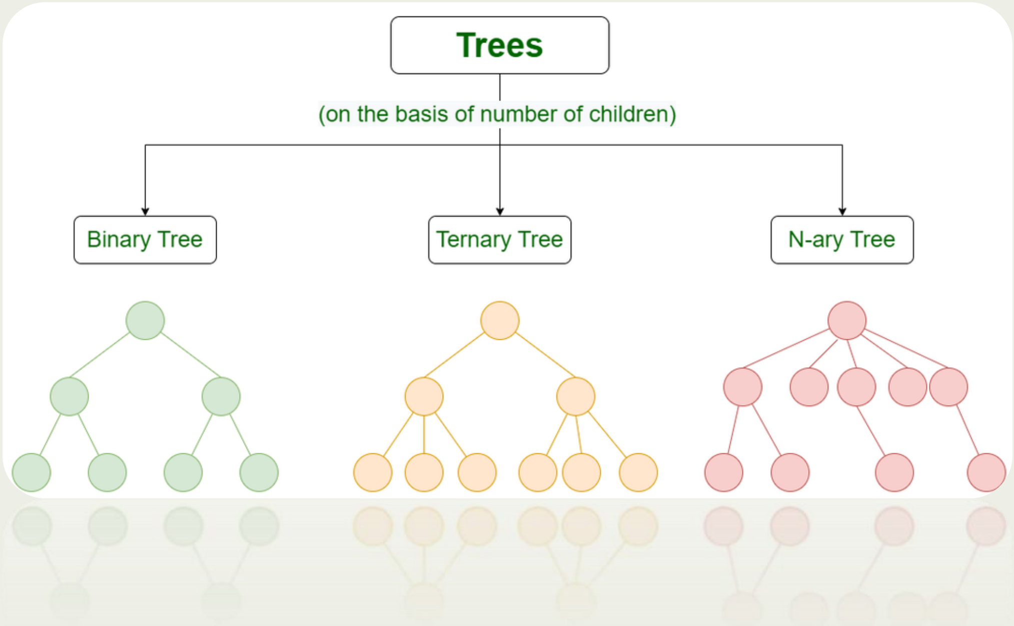
- A **root node**
- **Zero or more subtrees** (T1, T2, ..., Tn) Each subtree is connected to the root via an **edge**.

🌲 Types of Trees

1. **Binary Tree**:
   Each node has **at most two children**.
   ➤ Variants: Full, Complete, Balanced, Degenerate, Binary Search Tree (BST), Binary Heap.

2. **Ternary Tree**:
   Each node has **at most three children** (left, mid, right).

3. **N-ary Tree (Generic Tree)**:
   A node can have **N number of children**, where N is not fixed.



🔁 Basic Operations in Tree

- **Create**: Initialize a new tree.
- **Insert**: Add a new node to the tree.
- **Search**: Find a node with specific value.
- **Traversal**: Visit all nodes.
  - **Depth-First Search (DFS)**: Preorder, Inorder, Postorder.
  - **Breadth-First Search (BFS)**: Level Order Traversal.

📐 Properties of Tree

1. **Number of Edges**:
   If a tree has N nodes, it will always have N - 1 edges.
2. **Depth of a Node**:
   Number of edges from root to that node.
3. **Height of a Node**:
   Number of edges from the node to the **deepest leaf**.
4. **Height of Tree**:
   Height of the **root node** (longest path to a leaf).
5. **Degree of Node**:
   Number of direct children of that node.
6. **Degree of Tree**:
   Maximum degree among all nodes in the tree.

📌 Summary

- Trees help model real-world hierarchies like file systems or DOM structures.
- Trees allow efficient insertions, deletions, and traversals.
- Trees are non-linear and recursive in nature.
- Understanding tree structure builds a strong foundation for advanced data structures like BSTs, Heaps, Tries, etc.
