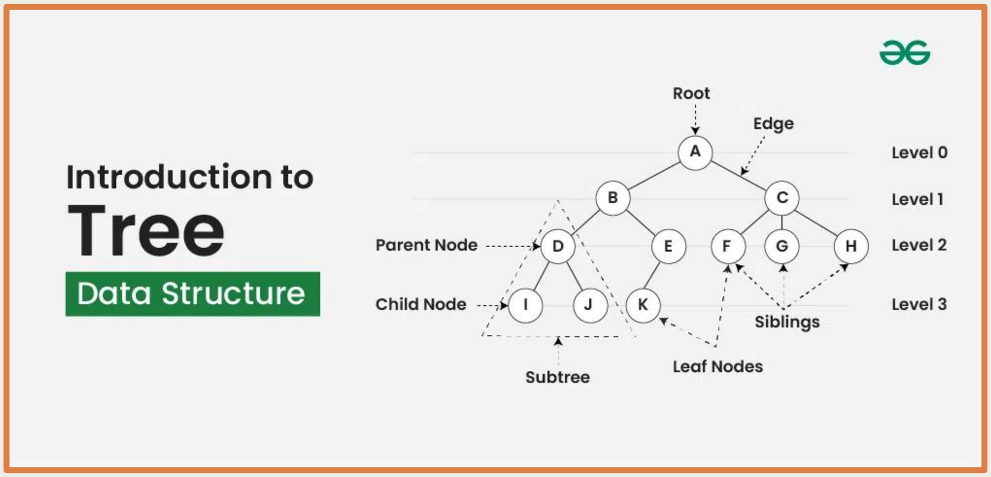
🌳 Introduction to Tree Data Structure

A **Tree** is a **non-linear**, hierarchical data structure that consists of nodes connected by edges. It mimics a real-world tree structure with a root and branches. The tree starts with a root node and extends into subtrees formed by child nodes.

🌿 Real-Life Examples of Tree Structure

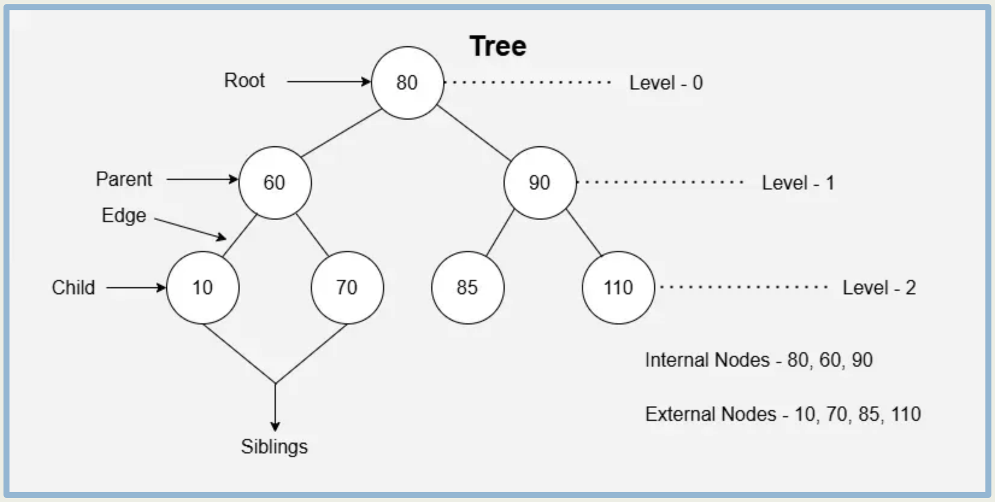
- File system in an Operating System.
- HTML/XML document structure.
- Family trees or organizational charts.

🧱 Basic Terminologies



- **Root Node**: The topmost node without a parent.
- **Parent Node**: A node that has children.
- **Child Node**: A node descended from another node.
- **Leaf Node**: A node with no children.
- **Internal Node**: A node with at least one child.
- **Siblings**: Nodes sharing the same parent.

- **Ancestors**: All nodes along the path from the root to that node.
- **Descendants**: All nodes that branch from a given node.
- **Level**: The distance from the root (root has level 0).
- **Subtree**: A part of the tree that forms its own tree from a given node.
- **Degree of Node**: Number of children.



- **Height of Tree**: Longest path from root to a leaf.
- **Depth of Node**: Path length from root to that node.
- **Edge**: A connection between two nodes (Tree with N nodes has N-1 edges).

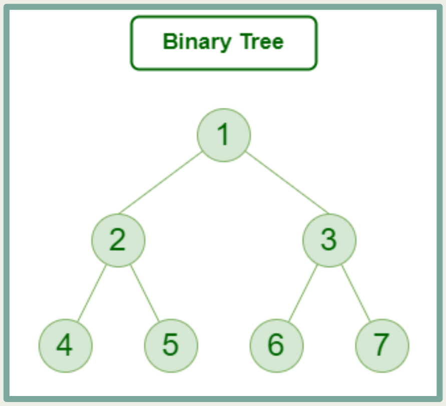
📌 Why Tree is Non-Linear?

Tree nodes are not arranged sequentially. Instead, they are placed hierarchically across levels, unlike arrays or linked lists which are linear.

🌲 Types of Trees

1. 🌱 Binary Tree

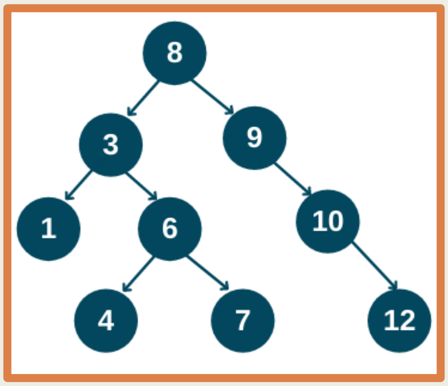
- Each node can have at most **two children**: left and right.
- No ordering rule.
- Can be:
  - **Full Binary Tree**: All nodes have 0 or 2 children.
  - **Complete Binary Tree**: All levels are fully filled except possibly the last, filled left to right.
  - **Perfect Binary Tree**: All internal nodes have 2 children, and all leaf nodes are at the same level.



---

2. 🔍 Binary Search Tree (BST)

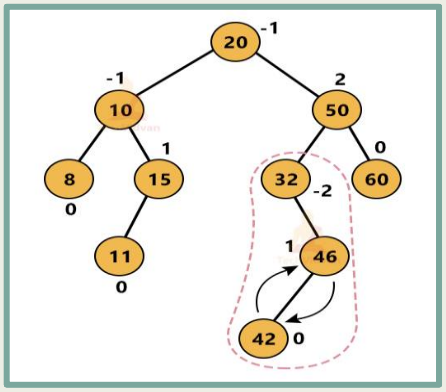
- A **binary tree with ordering rules**:
  - Left subtree contains values **less than** the node.
  - Right subtree contains values **greater than** the node.
- Operations like search, insertion, and deletion can be done efficiently if the tree is balanced.



---

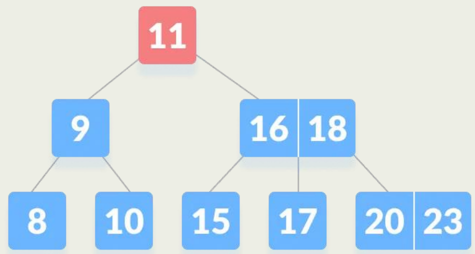
3. ⚖️ AVL Tree (Adelson-Velsky and Landis Tree)

- A **self-balancing binary search tree**.
- For every node, the **difference in height** of left and right subtree (balance factor) is at most **1**.
- Rotations (LL, RR, LR, RL) are used to maintain balance after insertion or deletion.



4. 🗂️ B Tree

- A **self-balancing search tree** used in **databases and file systems**.
- Nodes can have **more than two children** (multi-way tree).
- Keeps data sorted and allows searches, sequential access, insertions, and deletions in **logarithmic time**.
- All leaf nodes are at the same level.



Properties:

- Each node has a minimum and maximum number of keys.
- Insertion and deletion cause node splitting or merging to maintain balance.

---

5. ➕ B+ Tree

- A **variant of B Tree**, optimized for **disk-based storage systems**.
- **All values are stored in leaf nodes** only.
- Internal nodes **only store keys** and act as routing paths.
- Leaf nodes are linked like a **linked list** to support fast **range queries** and **sequential access**.

🧰 Basic Operations on Trees

- **Create** a tree.
- **Insert** a new node.
- **Delete** a node.
- **Search** for a node.
- **Traverse** the tree:
    - **Depth-First Search (DFS)**: Preorder, Inorder, Postorder.
    - **Breadth-First Search (BFS)**: Level order traversal.

🧠 Importance of Tree Data Structure

- Trees model hierarchical relationships efficiently.
- Used in:
    - File systems (folders).
    - Databases (B Tree/B+ Tree).
    - Routing algorithms (AVL/BST).
    - Compilers (syntax trees).
    - AI and ML (decision trees).
