Trees

1. Introduction

A **tree** is a hierarchical data structure used to model relationships with a parent-child structure. Trees are fundamental because they efficiently represent **hierarchical information**, **organizational data**, and **decision-making processes**.

Real-world analogies:

- **Organizational charts:** CEO at the root, managers as parents, employees as children.
- **File systems:** Root directory, folders as nodes, files as leaves.
- **Decision-making:** Each decision branch leads to outcomes, forming a decision tree.

Unlike linear structures (arrays, lists), trees allow for **fast searching, insertion, and structured organization**.

2. Basic Terminology

- **Node:** Fundamental unit of a tree.
- **Root:** The topmost node without a parent.
- **Leaf:** A node with no children.
- **Parent / Child:** Directly connected nodes.
- **Siblings:** Nodes sharing the same parent.
- **Height:** Length of the longest path from a node to a leaf.
- **Depth / Level:** Distance from the root.

Example: In a family tree, the root could be a grandparent, children are the next generation, and leaves are the youngest generation with no children.

3. Types of Trees

3.1 Binary Trees

Each node has at most **two children** (left and right).

Variants:

- **Full Binary Tree:** Every node has 0 or 2 children.
- **Complete Binary Tree:** All levels filled except possibly the last, filled left to right.
- **Perfect Binary Tree:** All internal nodes have two children, and all leaves are at the same level.

3.2 Binary Search Tree (BST)

A tree where **left child < parent < right child**.

Applications:

- Fast search, insertion, and deletion operations (average O(log n) if balanced).
- Examples: Student databases, dictionary word lookup.

3.3 Self-Balancing Trees

- **AVL Tree:** Maintains height balance after insertions and deletions to keep operations efficient.
- **Red-Black Tree:** Maintains balance using color properties for performance guarantees.
- Used in **databases, memory management, and sets/maps**.

3.4 N-ary Trees

Nodes can have **more than two children**.

- Example: File system directories or organization charts.

3.5 Trie (Prefix Tree)

- Efficiently stores strings.
- Common in **autocomplete, spell checkers, and IP routing**.

3.6 Segment Trees / Fenwick Trees

- Specialized trees for **range queries and updates**.
- Example: Efficiently computing cumulative sums or minimum/maximum in an array.

4. Tree Traversals

Traversal is visiting all nodes systematically.

1. **Inorder (Left → Root → Right):** Sorted order for BSTs.
   Example: Printing a dictionary in alphabetical order.
2. **Preorder (Root → Left → Right):** Useful for copying trees or evaluating prefix expressions.
3. **Postorder (Left → Right → Root):** Used in deleting trees or evaluating postfix expressions.
4. **Level Order (Breadth-First):** Visits nodes level by level.
   Example: Organizational charts or hierarchical reports.

5. Common Tree Operations

5.1 Search

- BST search is O(log n) in balanced trees.
- Real-world analogy: Searching for a file in a well-organized directory tree.

5.2 Insertion

- Place a new node while maintaining tree properties.
- Example: Adding a new employee in an organizational chart.

5.3 Deletion

- Remove a node and restructure to maintain balance.
- Complex in BSTs due to three cases (leaf, one child, two children).

5.4 Height / Depth Calculation

- Used in balancing and analysis of efficiency.
- Example: Calculating levels in a tournament bracket.

6. Applications of Trees

1. **File Systems:** Directories and files stored as trees.
2. **Databases:** B-Trees and B+ Trees for indexing large datasets.
3. **Decision Making:** Decision Trees in AI and machine learning.
4. **Parsing:** Abstract Syntax Trees (ASTs) in compilers.
5. **Networking:** Routing tables and prefix tries for IP addresses.
6. **Games:** Game trees for AI, e.g., chess and tic-tac-toe strategies.

7. Advantages of Trees

- Hierarchical representation of data is natural and intuitive.
- Efficient insertion, search, and deletion (especially in balanced trees).
- Provides structure for divide-and-conquer algorithms.
- Used as a foundation for advanced structures (heaps, tries, segment trees).

8. Summary

Trees are versatile structures bridging theory and practice. They help **organize hierarchical data, model decisions, and optimize searches**.

Mastering trees enables understanding of:

- **Hierarchical relationships** (e.g., org charts, files).
- **Efficient algorithms** (BSTs, AVL, Red-Black).
- **Advanced applications** in AI, databases, and networking.

Trees are everywhere — from your file system to AI decision models — making them an essential concept for computer scientists and software engineers.