Tree

By Mani Abedii

View Full DSA Repository

A clear, compact guide covering essential data structures and algorithms, designed for easy understanding and quick learning.

A **tree** is a hierarchical data structure composed of nodes connected by edges. It can be viewed as a special type of graph—specifically, an acyclic connected graph—that satisfies the condition of having exactly one unique path between any two nodes.

Trees are widely used in computer science to organize data in a branching manner. Trees are non-linear structures and often provide efficient ways to store, search, and manipulate information. They represent data using parent–child relationships and are commonly applied in domains such as file systems, databases, and network architectures. A tree begins with a distinguished node known as the root.

Key Characteristics of a tree:

- **Root:** The topmost node of the tree. It has no parent.
- **Node:** Each element in the tree. It can hold data & references to its child nodes.
- **Edge:** The connection between two nodes (e.g. parent to child)
- **Parent:** A node that has one or more child nodes.
- **Child:** A node that has a parent.
- **Sibling:** Children that have the same parent.
- **Leaf (External Node):** A node with no children.
- **Internal Node:** A node that has at least one child (non-leaf).
- **Ancestor:** Any node on the path from a specific node toward the root (including parent, grandparent, etc.)
- **Descendant:** Any node located in the subtree rooted at a given node.
- **Subtree:** A tree consisting of a node and all its descendants.
- **Degree:** The number of children a node has. Degree of a tree = Maximum degree along all nodes.
- **Branching Factor:** The maximum number of children a node is allowed to have (e.g. 2 in binary trees)
- **Path:** A sequence of nodes connected by edges.
- **Height (of a node):** The number of edges on the longest downward path from the node to any leaf. Height of a tree = Height of the root node.
- **Depth (of a node):** The number of edges from the root to the node.
- **Level:** The root is at level 0. its children are level 1 and so on.
- **Size:** The total number of nodes in the tree.
- A tree with **N nodes** always has **N-1 edges**.
- A tree with height **h** contains **h+1 levels**.

© 2025 Mani Abedii

Special Forms of Trees:

- **Full Tree:** A tree in which every internal node has either 0 children or the maximum possible number of children.
- **Complete Tree:** A tree in which all levels are completely filled except possibly the last level which is filled left to right.
- **Perfect Tree:** A tree where all internal nodes have children and all leaves are at the same level. A perfect tree is both full and complete.

Balanced vs. Unbalanced Tree:

- **Balanced Tree:** A tree where the height is minimized relative to the number of nodes (usually O(log n)).
- **Unbalanced Tree:** A tree that is skewed, with height approaching O(n). Similar to a linked list.

Ordered vs. Unordered Tree:

- **Ordered Tree:** The children of each node have a left-to-right order.
- **Unordered Tree:** No specific ordering of children exists.

Most data structure trees (Binary Tree, BST, Heap) are **ordered.**

Traversal Concepts:

- **Depth-First Traversal (DFS)**

  Explores as far down each branch as possible before backtracking, implemented using a stack.

  Types of DFS traversal:

  - Preorder (Root → Left → Right)
  - Inorder (Left → Root → Right)
  - Postorder (Left → Right → Root)

- **Breadth-First Traversal (BFS)**

  Also called **Level-order traversal.** Visits nodes level by level from top to bottom. Implemented using a queue.

© 2025 Mani Abedii