Tree Basics – Data Structures

Introduction

In computer science, many problems involve hierarchical data rather than linear data. Linear data structures such as arrays and linked lists are not efficient for representing hierarchical relationships. Tree data structures are designed specifically to model such relationships in a clear and efficient way.

What is a Tree?

A Tree is a non-linear data structure composed of nodes connected by edges. It starts with a single root node, and every node (except the root) has exactly one parent. Each node can have zero or more children, forming a hierarchical structure.

Tree Terminology

- Node: An individual element of a tree that contains data.
- Root: The topmost node of the tree.
- Parent: A node that has child nodes connected to it.
- Child: A node that descends from another node.
- Sibling: Nodes that share the same parent.
- Leaf: A node that has no children.
- Edge: A connection between two nodes.
- Subtree: A tree formed from a node and its descendants.
- Depth: The number of edges from the root to a specific node.
- Height: The number of edges on the longest downward path from a node to a leaf.

Types of Trees

Trees come in different forms depending on how many children a node can have and how nodes are arranged. The most common types include:

- **General Tree:** Each node can have any number of children. There are no structural restrictions.
- **Binary Tree:** Each node can have at most two children, usually referred to as left and right.
- **Full Binary Tree:** Each node has either zero or two children. No node has exactly one child.
- **Complete Binary Tree:** All levels are completely filled except possibly the last level, which is filled from left to right.
- **Skewed Tree:** A tree where all nodes have only one child, making it similar to a linked list.

Tree Traversals

Tree traversal is the process of visiting each node of a tree exactly once in a specific order. Traversal methods are essential for searching, printing, and processing tree data.

- **Preorder Traversal:** Root → Left → Right. Useful for copying a tree.
- **Inorder Traversal:** Left → Root → Right. Produces sorted output in Binary Search Trees.
- **Postorder Traversal:** Left → Right → Root. Useful for deleting a tree.
- **Level Order Traversal:** Visits nodes level by level using a queue.

Time Complexity

All tree traversal algorithms visit each node exactly once. Therefore, their time complexity is O(n), where n is the number of nodes in the tree.