Tree problems on LeetCode can vary widely in terms of difficulty and concept, but there are several fundamental concepts and techniques that are commonly used to solve them. Here are some of the most popular ones:

**Depth-First Search (DFS)**

DFS is a fundamental algorithm for tree traversal. It explores as far as possible along each branch before backtracking. In tree problems, DFS is often implemented recursively or using a stack. Common DFS problems include finding paths, calculating the height of a tree, and searching for specific nodes.

**Breadth-First Search (BFS)**

BFS explores the tree level by level, visiting all the nodes at the current level before moving on to the next level. It's often implemented using a queue. BFS is commonly used to solve problems like finding the shortest path, level order traversal, and finding the minimum depth of a tree.

**Binary Tree Traversal (Inorder, Preorder, Postorder)**

These traversal techniques involve visiting all nodes of a binary tree in a specific order.

**Inorder traversal**: Visit left subtree, then current node, then right subtree.

**Preorder traversal**: Visit current node, then left subtree, then right subtree.

**Postorder traversal:** Visit left subtree, then right subtree, then current node.

These traversals are used in a variety of tree problems for different purposes, such as reconstructing a tree from traversal sequences, evaluating expressions, and checking for symmetry.

**Binary Search Tree (BST) properties**

If the problem involves a binary search tree, understanding its properties is crucial. Common operations on BSTs include insertion, deletion, searching for a specific key, finding the minimum/maximum key, and validating the BST property.

**Recursion**

Many tree problems can be elegantly solved using recursion. Recursion is particularly useful for traversing or searching through the tree structure, breaking down complex problems into smaller subproblems.

**Dynamic Programming on Trees**

Sometimes, dynamic programming techniques are applied to trees. This involves solving subproblems and building up solutions recursively. Dynamic programming on trees is often used in problems involving tree diameter, maximum path sum, and subtree problems.

**Tree Rotation**

For problems involving balancing binary search trees, tree rotation techniques like left rotation, right rotation, and their combinations (e.g., AVL trees, Red-Black trees) are essential.

**LCA (Lowest Common Ancestor)**

Finding the lowest common ancestor of two nodes in a tree is a common problem. Techniques such as using parent pointers or DFS can be employed to solve LCA problems efficiently.

**Trie (Prefix Tree)**

Trie is a tree-like data structure used to store a dynamic set of strings where the keys are usually strings. It's particularly useful in problems involving string operations like autocomplete, spell-checking, and prefix search.

**Segment Trees and Fenwick Trees**

These are specialized tree data structures used for solving problems involving range queries and updates efficiently, like finding the sum or minimum/maximum value of elements in a range.

Understanding these concepts and techniques will provide a strong foundation for solving tree-related problems on LeetCode and other platforms.

Here's a list of widely used algorithms for solving tree-related problems on LeetCode:

**Depth-First Search (DFS)**

Used for traversing the tree in depth-first order, either pre-order, in-order, or post-order.

**Breadth-First Search (BFS)**

Used for traversing the tree in breadth-first order, level by level.

**Binary Search Tree (BST) operations**

Includes searching, insertion, deletion, finding the minimum/maximum, and validating the properties of the BST.

**Lowest Common Ancestor (LCA)**

Finding the lowest common ancestor of two nodes in a tree.

**Tree Traversal Algorithms**

In-order, pre-order, and post-order traversals are fundamental for various tree-related problems.

**Trie (Prefix Tree)**

Used for efficient storage and retrieval of strings, often used in string-related problems.

**Dynamic Programming on Trees**

Solving problems recursively by breaking them down into subproblems and memoizing results.

**Tree Rotations**

Used in balancing binary search trees like AVL trees and Red-Black trees.

**Segment Trees and Fenwick Trees**

Used for efficient range queries and updates in an array.

**Binary Indexed Trees (BITs)**

Also known as Fenwick Trees, used for efficient range queries and updates.

**Morris Traversal**

A space-optimized version of tree traversal algorithms, particularly in-order traversal.

**Backtracking**

Used in various tree-related problems where exhaustive search is required, such as generating all possible paths.

**Greedy Algorithms**

Occasionally used in tree-related problems, especially when optimizing for a specific condition or property.

**Graph Algorithms**

Techniques like Dijkstra's algorithm or Floyd-Warshall algorithm are sometimes applicable when the tree can be treated as an undirected graph.

**Union-Find (Disjoint Set Union)**

Used for solving problems related to connected components or disjoint sets in a tree.

These algorithms cover a wide range of tree-related problems on LeetCode and other platforms, providing a solid toolkit for tackling various challenges efficiently.