Tree Data Structures

Tree Data Structure

A tree is a non-linear data structure consisting of nodes connected by edges. It is used to represent hierarchical relationships.

Binary Tree

A binary tree is a type of tree in which each node has at most two children, referred to as the left child and the right child.

Tree Data Structure

A tree is a non-linear data structure consisting of nodes connected by edges. It is used to represent hierarchical relationships.

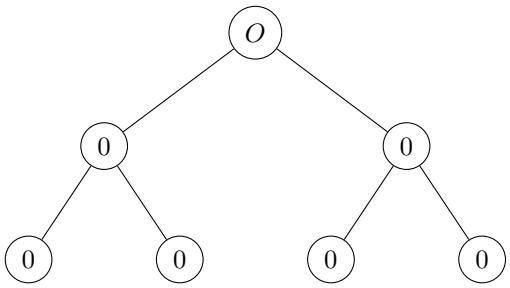
Binary Tree

A binary tree is a type of tree in which each node has at most two children, referred to as the left child and the right child.

- **Level 0**: Root Node
- **Level 1**: Nodes that are children of the root node
- **Level 2**: Nodes that are children of the nodes in Level 1

Types of Nodes

- **Root Node**: The topmost node in a tree.
- **Internal Node**: A node that has at least one child.
- **Leaf Node**: A node that does not have any children.



Terminology

- **Root Node**: The top node of the tree.
- **Leaf Node**: A node that does not have any children.
- **Internal Node**: A node that has at least one child.
- **Levels**: The root node is at level 0, its children are at level 1, and so on.

Tree Traversal Algorithms

Tree traversal algorithms are used to visit all the nodes in a tree.

Inorder Traversal

1. Traverse the left subtree.
2. Visit the root node.
3. Traverse the right subtree.

```
def inorderTraversal(root):
    if root:
        inorderTraversal(root.left)
        print(root.val, end=" ")
        inorderTraversal(root.right)
```

Preorder Traversal

1. Visit the root node.
2. Traverse the left subtree.
3. Traverse the right subtree.

```
def preorderTraversal(root):
    if root:
        print(root.val, end=" ")
        preorderTraversal(root.left)
        preorderTraversal(root.right)
```

Postorder Traversal

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root node.

```
def postorderTraversal(root):
    if root:
        postorderTraversal(root.left)
        postorderTraversal(root.right)
        print(root.val, end=" ")
```

Recurrence Relation

The time complexity for tree traversal is generally $O(n)$, where n is the number of nodes in the tree.

Balanced Binary Tree

For a balanced binary tree, the recurrence relation is:

$$T(n) = 2T\left(\frac{n}{2}\right) + c$$

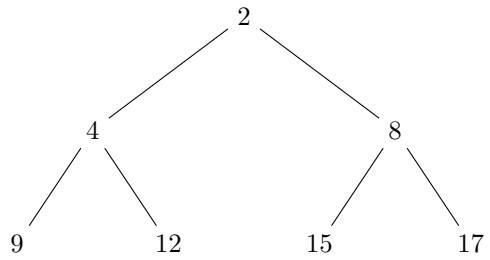
Skewed Binary Tree

For a skewed binary tree, the recurrence relation is:

$$T(n) = T(n-1) + c = O(n)$$

Examples

Example Tree



Traversal of Example Tree

Inorder Traversal

For the above tree, the inorder traversal is: 9, 4, 12, 2, 15, 8, 17.

Preorder Traversal

For the above tree, the preorder traversal is: 2, 4, 9, 12, 8, 15, 17.

Postorder Traversal

For the above tree, the postorder traversal is: 9, 12, 4, 15, 17, 8, 2.

Python Implementation

```
class TreeNode:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.val = data

def inorderTraversal(root):
    if root:
        inorderTraversal(root.left)
        print(root.val, end=" ")
        inorderTraversal(root.right)

def preorderTraversal(root):
    if root:
        print(root.val, end=" ")
        preorderTraversal(root.left)
        preorderTraversal(root.right)

def postorderTraversal(root):
    if root:
        postorderTraversal(root.left)
        postorderTraversal(root.right)
        print(root.val, end=" ")

# Driver code
root = TreeNode(2)
root.left = TreeNode(4)
root.right = TreeNode(8)
root.left.left = TreeNode(9)
root.left.right = TreeNode(12)
root.right.left = TreeNode(15)
root.right.right = TreeNode(17)

print("Inorder traversal of a given tree is:")
inorderTraversal(root)
print("\nPreorder traversal of a given tree is:")
```

```
preorderTraversal(root)
print("\nPostorder traversal of a given tree is:")
postorderTraversal(root)
```