Lecture 6.42

Tree Traversal: Preorder, Inorder, and Postorder

Agenda

- What is Tree Traversal?
- Depth-First Search (DFS) in Trees
- Preorder Traversal: Concept, Example, and Code
- Inorder Traversal: Concept, Example, and Code
- Postorder Traversal: Concept, Example, and Code
- Complexity Analysis
- Summary and Next Steps

What is Tree Traversal?

- Definition: The process of visiting every node in a tree exactly once.
- Unlike linear data structures (arrays, linked lists) which have one logical way to traverse, trees can be traversed in multiple ways.
- Traversal is necessary for operations like searching, printing, or modifying all nodes.
- Two main approaches: Depth-First Search (DFS) and Breadth-First Search (BFS).

Depth-First Search (DFS) in Trees

- DFS goes as deep as possible down one path before backtracking.
- For binary trees, DFS naturally leads to three traversal strategies based on when we visit the Root node relative to its children.
- 1. Preorder: Visit Root, then Left subtree, then Right subtree (N-L-R).
- 2. Inorder: Visit Left subtree, then Root, then Right subtree (L-N-R).
- 3. Postorder: Visit Left subtree, then Right subtree, then Root (L-R-N).

Preorder Traversal Theory

- Sequence: Root -¿ Left Subtree -¿ Right Subtree.
- Recursively perform Preorder on the left child, then the right child.
- Primary Use Case: Creating a duplicate or copy of the tree.
- Prefix expression (Polish notation) generation from expression trees.

Preorder Traversal Example

- Consider a tree with Root A, Left Child B, Right Child C.
- B has children D and E. C has right child F.
- Step 1: Visit A (Root).
- Step 2: Traverse Left to B. Visit B. Then Left to D. Visit D.
- Step 3: D has no children. Backtrack to B. Traverse Right to E. Visit E.
- Step 4: Backtrack to A. Traverse Right to C. Visit C.
- Step 5: C has no left child. Traverse Right to F. Visit F.
- Result: A, B, D, E, C, F.

Preorder Traversal: Python Implementation

- class Node:
- def __init__(self, key):
- self.left = None
- self.right = None
- self.val = key
-
- def printPreorder(root):
- if root:
- print(root.val, end=' ')
- printPreorder(root.left)
- printPreorder(root.right)

Inorder Traversal Theory

- Sequence: Left Subtree -¿ Root -¿ Right Subtree.
- Recursively perform Inorder on the left child, visit the root, then Inorder on the right child.
- Primary Use Case: Retrieving elements in non-decreasing (sorted) order from a Binary Search Tree (BST).
- Infix expression generation from expression trees.

Inorder Traversal Example

- Same tree: Root A, Left B (children D, E), Right C (right child F).
- Step 1: Go to A's left (B), then B's left (D). D has no left child.
- Step 2: Visit D. Then D's right (none).
- Step 3: Backtrack to B. Visit B.
- Step 4: Go to B's right (E). Visit E.
- Step 5: Backtrack to A. Visit A.
- Step 6: Go to A's right (C). C has no left child. Visit C.
- Step 7: Go to C's right (F). Visit F.
- Result: D, B, E, A, C, F.

Inorder Traversal: Python Implementation

- def printInorder(root):
- if root:
- # First recur on left child
- printInorder(root.left)
- # Then print the data of node
- print(root.val, end=' ')
- # Finally recur on right child
- printInorder(root.right)

Postorder Traversal Theory

- Sequence: Left Subtree -¿ Right Subtree -¿ Root.
- Recursively perform Postorder on the left child, then the right child, and finally visit the root.
- Primary Use Case: Safely deleting an entire tree.
- Used to evaluate postfix expressions (Reverse Polish Notation).

Postorder Traversal Example

- Same tree: Root A, Left B (children D, E), Right C (right child F).
- Step 1: Go deep left to D. Visit D.
- Step 2: Backtrack to B, then go right to E. Visit E.
- Step 3: Now both children of B are visited. Visit B.
- Step 4: Backtrack to A, go right to C. Go left of C (none), go right to F. Visit F.
- Step 5: Both children of C visited. Visit C.
- Step 6: Both children of A visited. Visit A.
- Result: D, E, B, F, C, A.

Postorder Traversal: Python Implementation

- def printPostorder(root):
- if root:
- # First recur on left child
- printPostorder(root.left)
- # Then recur on right child
- printPostorder(root.right)
- # Finally print the data of node
- print(root.val, end=' ')

Time and Space Complexity

- Time Complexity: O(N) for all three traversals, where N is the number of nodes.
- Reasoning: Every node is visited exactly once, doing a constant O(1) amount of work at each node.
- Space Complexity: O(H), where H is the maximum height of the tree.
- Reasoning: The space is determined by the maximum size of the function call stack due to recursion.
- Worst-case Space: O(N) for a skewed tree.
- Best-case Space: O(log N) for a perfectly balanced tree.

Summary & Next Steps

- Traversals dictate the order in which we visit nodes in a tree structure.
- Preorder (N-L-R): Visit Node, Left, Right.
- Inorder (L-N-R): Visit Left, Node, Right. (Produces sorted output for BST).
- Postorder (L-R-N): Visit Left, Right, Node. (Good for tree deletion).
- Next Lecture: We will explore practical applications of Binary Trees and how these traversals are used in real-world scenarios.