Lecture 6.40

Binary Search Tree (BST): Concepts and Insertion

Agenda for Today

- 1. Recap of Binary Tree Properties
- 2. Introduction to Binary Search Tree (BST)
- 3. The BST Property and Rules
- 4. Insertion Operation in a BST
- 5. Algorithm and Time Complexity
- 6. Implementation in Python
- 7. Applications and Edge Cases

What is a Binary Search Tree?

- A Binary Search Tree (BST) is a node-based binary tree data structure.
- Rule 1: The left subtree of a node contains only nodes with keys lesser than the node's key.
- Rule 2: The right subtree of a node contains only nodes with keys greater than the node's key.
- Rule 3: The left and right subtrees each must also be a binary search tree.

BST Property Illustrated

- Valid BST Example:
- Root: 50
- Left child of 50: 30 (30 ¡ 50)
- Right child of 50: 70 (70 ¿ 50)
- Invalid BST Example:
- Root: 50, Left child: 60 (Violation: 60 is not ¡ 50)

Advantages of BST over Arrays and Linked Lists

- Arrays: Fast search (binary search), but slow insertion/deletion O(N).
- Linked Lists: Fast insertion/deletion, but slow search O(N).
- BST: Combines the benefits of both.
- Average time complexity for Search, Insert, and Delete is O(log N).
- Maintains elements in a sorted order dynamically.

Insertion in a BST: Conceptual Overview

- Goal: Insert a new key into the BST while maintaining properties.
- Step 1: Start at the root node.
- Step 2: Compare the new key with the current node's key.
- Step 3: If new key ¡ current key, move to the left child.
- Step 4: If new key ¿ current key, move to the right child.
- Step 5: Repeat until an empty spot (None) is found, attach node.

Step-by-Step Insertion Example 1

- Initial Tree: Root=50, L=30, R=70
- Insert 40:
- 1. Compare 40 with 50. 40 ¡ 50, go left.
- 2. Compare 40 with 30. 40 ¿ 30, go right.
- 3. Right of 30 is None. Insert 40 as the right child of 30.

Recursive Insertion Algorithm

- Function Insert(node, key):
- If node is None:
- return new Node(key)
- If key ¡ node.key:
- node.left = Insert(node.left, key)
- Else if key ¿ node.key:
- node.right = Insert(node.right, key)
- return node

Python Implementation: Node Structure

- class Node:
- def __init__(self, key):
- self.left = None
- self.right = None
- self.val = key
-
- Each node holds a value, and references to its left and right children.

Python Implementation: Insert Logic

- def insert(root, key):
- if root is None:
- return Node(key)
- else:
- if root.val == key:
- return root
- elif root.val ¡ key:
- root.right = insert(root.right, key)
- else:
- root.left = insert(root.left, key)
- return root

Iterative Insertion Approach

- Recursion uses implicit stack space O(H). Iterative approach uses O(1) space.
- def insert_iterative(root, key):
- if not root: return Node(key)
- curr = root
- while True:
- if key ¡ curr.val:
- if not curr.left: curr.left = Node(key); break
- curr = curr.left
- else:
- if not curr.right: curr.right = Node(key); break
- curr = curr.right
- return root

The Problem of Skewed Trees

- What happens if we insert data in sorted order? (e.g., 10, 20, 30, 40)
- Tree Structure:
- 10
- \
- 20
- \
- 30
- \
- 40
- This becomes effectively a Linked List!

Time and Space Complexity

- Time Complexity (Average Case): O(log N) - where N is number of nodes.
- Time Complexity (Worst Case): O(N) - occurs when the tree becomes highly unbalanced.
- Space Complexity (Recursive): O(H) - where H is the height of the tree, due to the recursion call stack.
- Space Complexity (Iterative): O(1).

Applications of Binary Search Trees

- 1. Implementing dictionary systems for fast lookups.
- 2. Database indexing structures (often generalized into B-Trees).
- 3. Priority Queues (using variations like heaps, but BSTs can also serve).
- 4. Handling dynamic data sorting and maintaining sorted streams of data.

Summary & Key Takeaways

- BST enforces a strict ordering rule: Left ¡ Root ¡ Right.
- Insertion always adds a new node as a leaf.
- BST allows for fast average-case O(log N) search, insert, and delete operations.
- Unbalanced trees can degrade performance to O(N).
- Next up: Deleting nodes from a BST.