Lecture 6.43

Applications of Binary Tree

Agenda

- Expression Trees in Compiler Design
- Data Compression: Huffman Coding Algorithm
- Database Indexing Foundations with BSTs
- Priority Queues using Binary Heaps
- Decision Trees and Network Routing
- Complexity Analysis and Summary

Expression Trees: Compiler Design

- Abstract Syntax Trees (ASTs): Compilers use binary trees to parse expressions and statements.
- Internal Nodes: Represent operators (e.g., $+$, -, *, $/$).
- Leaf Nodes: Represent operands (e.g., constants, variables).
- Evaluation: Solved via recursive bottom-up evaluation (Postorder Traversal).
- Prefix and Postfix: Traversing the tree in Preorder or Postorder yields Prefix or Postfix notations respectively, eliminating the need for parentheses.

Expression Tree: Example Evaluation

- Expression: (5 + 3) * (9 - 2)
- Tree Structure:
- - Root: *
- - Left Subtree: Root '+', Leaves '5', '3'
- - Right Subtree: Root '-', Leaves '9', '2'
- Postorder Traversal: 5 3 + 9 2 - *
- Execution: Stack-based evaluation. Push operands, apply operator on pop.

Data Compression: Huffman Coding

- Objective: Lossless data compression using variable-length prefix codes.
- Concept: Assign shorter binary codes to frequently occurring characters and longer codes to less frequent ones.
- Binary Tree Usage: The Huffman Tree is a strictly binary tree where left branches represent '0' and right branches represent '1'.
- Prefix Property: No encoded character is a prefix of another, allowing unambiguous decoding without a delimiter.

Building a Huffman Tree

- 1. Calculate frequency of each character.
- 2. Create leaf nodes for each character and push them to a priority queue (min-heap).
- 3. Extract two nodes with the lowest frequencies.
- 4. Create a new internal node with a frequency equal to the sum of the two nodes.
- 5. Insert the new node back into the priority queue.
- 6. Repeat until only one node (the root) remains.

Database Indexing Foundations

- Problem: Linear search through massive database tables is highly inefficient O(N).
- Solution: Binary Search Trees (BSTs) allow O(log N) lookup, insertion, and deletion.
- Self-Balancing Trees: AVL Trees and Red-Black Trees (variations of BSTs) ensure the tree height remains logarithmic.
- B-Trees: Generalized binary trees optimized for block storage, minimizing disk I/O reads.

Heaps and Priority Queues

- Binary Heap: A complete binary tree fulfilling the heap property.
- Max-Heap: The parent is always greater than or equal to its children.
- Min-Heap: The parent is always less than or equal to its children.
- Applications: Implementation of Priority Queues, where elements are dequeued based on priority rather than insertion order.
- Real-world Usage: CPU thread scheduling, Dijkstra's Shortest Path Algorithm, Heap Sort.

Heap Operations Complexity

- Insertion (Push): O(log N) - Append to end and 'bubble up'.
- Deletion (Pop): O(log N) - Replace root with last element and 'bubble down'.
- Peek (Find Min/Max): O(1) - The root element.
- Heapify (Build from array): O(N) - Optimized bottom-up construction.

Decision Trees and Routing

- Machine Learning: Decision trees split data points based on feature thresholds to classify outcomes.
- Networking: Tries (Prefix Trees, a variant of binary trees) are used in routers for IP routing tables.
- Longest Prefix Match: Routers use variations of binary trees to quickly determine the optimal path for an IP packet.
- Cryptography: Merkle Trees (Hash trees) verify data integrity in blockchains and peer-to-peer networks.

Summary of Tree Complexities

- BST (Average): Search O(log N), Insert O(log N).
- AVL/Red-Black: Search O(log N), Insert O(log N) (Strictly bounded).
- Binary Heap: Access Max/Min O(1), Insert O(log N).
- Huffman Tree: Construction O(N log N) utilizing priority queues.
- Expression Tree: Evaluation O(N) where N is number of nodes.

Next Steps & Lab Preparation

- Lab Focus: Implementing a Binary Search Tree (BST) in Python.
- Tasks:
- - Write insertion and deletion functions.
- - Implement Inorder traversal to retrieve sorted data.
- - Build a menu-driven Python script.
- Next Topic: Review of Data Structures and preparation for final evaluations.