Data Structures Lab

Observation

1. Binary search trees, balanced BSTs, and hash tables each have unique structural properties. If a hypothetical data structure could achieve both the ordered property of a BST and the constant-time complexity of a hash table, what theoretical limitations might it encounter? Discuss in terms of computational complexity and trade-offs in real-world scenarios.
2. In certain cases, an unbalanced BST can outperform a balanced BST. Formulate a scenario where maintaining balance in a BST could theoretically degrade performance. What specific types of data or operations would lead to this situation, and why?
3. Hash tables and balanced BSTs are fundamentally different in handling "collisions" (for hash tables) and "balance" (for BSTs). Is there an underlying theoretical link between collision resolution in hash tables and self-balancing in trees? Discuss how these two concepts address similar computational challenges differently.
4. Balanced trees maintain height constraints to improve efficiency, yet this introduces overhead. Conceptually, if there were no height constraints (i.e., no balancing requirements), what would be the worst-case computational complexity of a search operation in a large, unbalanced BST?
5. Different balanced BSTs like AVL trees and Red-Black trees use different strategies to maintain balance. Compare and contrast how these balancing techniques impact performance in scenarios with high insertion vs. high deletion operations. Which type of balanced BST would you recommend in a database with frequent delete operations?

Execution

1. Design a data structure for a library system where each book has a unique ISBN. The structure should:

   - Support constant-time lookups by ISBN.
   - Allow range queries to retrieve books within a specified ISBN range.

   Implement methods for insertion, deletion, ISBN search, and range queries. Use a combination of hashing and balanced BSTs. Analyze the time complexity of each operation.

2. Build a system to track unique user activity intervals (defined by start and end timestamps). It should:

   - Detect and eliminate duplicate intervals.
   - Retrieve all unique intervals in sorted order.

   Implement methods to add, remove, and retrieve intervals. Use hashing for duplicate detection and a balanced BST to maintain sorted order.