Data Structures past year Questions 2023

In the context of balanced binary search trees, provide a detailed comparison between AVL trees and Red-Black trees, specifically focusing on their balancing mechanisms, rotation operations, and scenarios where one would be preferred over the other. How do these differences impact their performance in real-world applications involving frequent insertions and deletions?

When implementing a hash table for a large-scale system, multiple collision resolution techniques exist (chaining, linear probing, quadratic probing, and double hashing). Compare these approaches in terms of their space-time trade-offs, performance degradation under high load factors, and practical implementation challenges. How would you choose the most appropriate method for a specific application?
In the implementation of priority queues, explain the fundamental differences between Binary Heaps and Fibonacci Heaps. How do their time complexities differ for various operations (insert, delete-min, decrease-key), and what makes Fibonacci Heaps theoretically superior but practically challenging to implement?

Compare and contrast B-trees and B+ trees in the context of database management systems. How do their structural differences affect disk I/O operations, range queries, and overall performance? What specific characteristics make B+ trees more suitable for database indexing?

When dealing with graph traversal algorithms, analyze the situations where Depth-First Search (DFS) would be more advantageous than Breadth-First Search (BFS) and vice versa. How do these algorithms need to be modified when handling weighted edges and directed graphs, and what are their space complexity implications for very large graphs?

Examine the Skip List data structure as an alternative to balanced binary search trees. How does its probabilistic balancing approach differ from the deterministic balancing of AVL or Red-Black trees? What makes Skip Lists particularly suitable for concurrent programming environments?

In the context of string processing and pattern matching, analyze how Trie data structures can be optimized for space efficiency. What are the trade-offs between using a standard Trie versus a compressed Trie, and how do these choices affect the performance of operations like prefix matching and autocomplete functionality?