Questions for data structures

Question
(Gemini) 3. **Efficiently implementing a Least Recently Used (LRU) cache with limited memory and high throughput:** Design a highly efficient LRU cache implementation using a data structure that allows both O(1) time complexity for insertion, retrieval, and removal of the least recently used element, and supports concurrent access from multiple threads safely and efficiently. Consider the limitations of readily available concurrent data structures like ConcurrentHashMap. Explain your choice of data structure and algorithms and analyze its performance characteristics under high load.
(Gemini) 2. **Space-optimized Trie for large dictionaries with long keys:** You need to build a Trie data structure to store a very large dictionary of words (millions of entries), where many words share long common prefixes. Standard Trie implementations can consume a vast amount of memory. Describe a space-optimized Trie variant that minimizes memory usage while maintaining reasonably fast search, insertion, and deletion times. Consider techniques like compressed nodes or path compression. Analyze the space and time complexities of your optimized Trie.
(Gemini) 4. **Optimizing graph traversal for a weighted, directed graph with negative edge weights and cycles:** You're given a weighted, directed graph that may contain negative edge weights and cycles. You need to find the shortest path from a single source node to all other reachable nodes. Standard Dijkstra's algorithm doesn't work correctly with negative edge weights. Describe how you would modify Dijkstra's algorithm or utilize a different algorithm (like Bellman-Ford) to solve this problem efficiently, handling potential negative cycles appropriately. Discuss the time complexity and limitations of your chosen approach.
(Gemini) 1. **Designing a self-balancing tree for a highly dynamic environment:** You need to design a self-balancing binary search tree (BST) to handle a dataset where insertions and deletions happen frequently and randomly, with a significant bias towards deletions near the root. Standard AVL or red-black trees might exhibit poor performance in this scenario. Describe the modifications you would make to a standard self-balancing tree (or propose an entirely different structure) to optimize performance, considering factors like rebalancing frequency, worst-case time complexity for operations, and space overhead. Justify your design choices.
(Gemini) 5. **Implementing a persistent data structure for version control:** Design a persistent versioned data structure (like a persistent list or tree) that allows efficient access to all previous versions of the data. Each update should create a new version without modifying previous versions. Describe the data structure and the operations (insert, delete, access) necessary