Jester Cruz

CS300

06/13/2025

Evaluation

For the vector-based implementation, loading courses from the file involves reading and parsing n courses, which is O(n). However, validating prerequisites requires searching the vector linearly for each prerequisite, leading to a worst-case time complexity of O(n²). Searching for a specific course in the vector is also O(n), since it requires a linear search. Printing the course list in alphanumeric order requires sorting the vector, which is O(n log n), followed by printing, which is O(n). The vector approach is simple and intuitive, making it suitable for small datasets, but it doesn’t do well on search and prerequisite validation tasks as the data grows.

The hash table-based implementation improves performance significantly. Loading the courses involves inserting each course into the hash table, which is O(1) per insert, resulting in O(n) overall. Prerequisite validation leverages a hash set for quick lookups, resulting in O(nk), where k is the average number of prerequisites per course. Searching for a course by course number is highly efficient at O(1) due to direct hash table access. However, printing the list in sorted order requires extracting keys and sorting them, which costs O(n log n), followed by quick lookups for each course. The hash table provides fast searches and validations and is scalable, but the need to sort keys for alphanumeric output introduces some overhead.

In the binary search tree (BST) approach, inserting courses is on average O(log n), but in the worst case, if the tree is unbalanced, insertion and search degrade to O(n). Prerequisite validation still relies on a map or hash set, leading to O(nk) time. Searching within the BST averages O(log n), but again can be O(n) if unbalanced. Printing courses in alphanumeric order is efficient with an in-order traversal at O(n). The BST offers efficient sorted traversal without extra sorting steps but is sensitive to insertion order, which can degrade performance if the tree becomes skewed. Additionally, prerequisite validation is not inherently fast within the BST structure.

Based on these analyses, the hash table is recommended for this application. It provides the fastest course lookups (O(1)) and efficient prerequisite validation. While sorting is necessary for printing courses alphanumerically, the O(n log n) cost is acceptable given the overall benefits. The vector, although straightforward, performs poorly with large datasets due to linear searches and validations. The BST offers sorted data traversal but risks degraded performance if not balanced and requires additional data structures for fast prerequisite checks.

Overall, the hash table balances speed and scalability, making it the best choice for implementing the ABCU Computer Science advising program.