Joseph Les

CS-300 Professor Lebel

Project one Evaluation

In the pseudocode, we cover how files are opened and closed, how each line of data within the file is read, and how formatting errors are verified. When opening and reading the file, the time complexity is O(1) as it doesn’t depend on the number of courses. Reading and parsing each line are O(n), where n represents the number of courses, with each line corresponding to a course. Splitting lines into tokens is O(1), assuming the number of tokens is constant and small. Creating course objects and adding them to data structures is O(1) for each course, leading to an overall O(n) for n courses. Therefore, the total complexity for reading and parsing the file is O(n).

The memory usage for reading and parsing will involve n entries corresponding to n courses for both the hash table and the vector. While iterating over each course, the time complexity is O(n). The complexity of iterating over prerequisites depends on the number of prerequisites each course has. Assuming each course has at most p prerequisites, checking if each prerequisite exists in the hash table is O(1), leading to O(p) for each course, and a total cost of O(n\*p).

The memory usage for storing course objects in hash tables and vectors is O(n). It's important to consider that sorting operations typically take O(n log n) time, where n is the number of courses. In tree operations, such as adding a course or performing an in-order traversal, the average complexity is O(log n) but can worsen to O(n) in the worst-case scenarios, such as a skewed tree.

Each data structure has its own set of advantages and disadvantages. Vectors offer fast sequential access and cache-friendliness. However, inserting or deleting elements in the middle of the vector is inefficient due to the required element shifting. Hash tables provide fast access to elements using keys, which is beneficial for quickly finding specific courses, but they can consume more memory due to the storage of keys and the hash table structure itself. Trees have advantages like an ordered structure, efficient searches, and flexibility. Their downside is the complexity in balancing and managing the tree structure to maintain efficiency.

Based on the Big O analysis and the characteristics of the three data structures my recommendation for managing course information in the code is to use a Hash Table. This recommendation is grounded in the specific requirements and operations that the application appears to prioritize, which include efficient access, search, and handling of courses by their unique identifiers.

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Tree |
| Reading and parsing file | O(n) | O(n) | O(n) |
| Iterating over courses | O(n) | O(n) | O(n) |
| Accessing courses | O(n) | O(1) | O(log n) |
| Iterating over prerequisites | O(n\*p) | O(p) | O(log n) |
| Inserting course | O(n) | O(1) | O(log n) |
| Deleting course | O(n) | O(1) | O(log n) |
| Sorting courses | O(n log n) | N/A | N/A |