Yoel Marquez

8-11-12

Project one

CS-300

For this project, I evaluated three data structures—vector, hash table, and binary search tree (BST)—to determine which would best serve the needs of the Computer Science department’s course management program. I analyzed each data structure based on its performance in key operations such as insertion, search, sorting, and memory usage, and conducted a comprehensive examination of the advantages and disadvantages of each structure.

Starting with the vector, it is a straightforward and efficient data structure for small to medium-sized datasets, offering fast insertion times at the end of the structure, typically O(1). However, its simplicity comes with limitations. Searching for a course within a vector requires a linear search, leading to a time complexity of O(n), which can be inefficient for larger datasets. Moreover, while vectors can hold data efficiently, sorting the courses alphanumerically requires an additional O(n log n) time complexity, which further reduces its appeal for this specific application where sorted data is crucial.

The hash table, on the other hand, excels in operations requiring fast lookups and insertions. With an average time complexity of O(1) for both operations, hash tables are highly efficient for accessing data based on keys such as course numbers. However, hash tables have a significant drawback in that they do not maintain data in any particular order, making it challenging to print a sorted list of courses without additional processing. Additionally, in scenarios where hash collisions occur frequently, the performance of a hash table can degrade to O(n), and while memory usage is generally efficient, it can increase due to the need for handling these collisions.

The binary search tree (BST) presents a compelling option for this project because it naturally maintains the sorted order of its elements. Operations such as insertion, search, and deletion in a balanced BST have a time complexity of O(log n), making it efficient for managing course data. An important advantage of using a BST is that the courses can be printed in alphanumeric order using an in-order traversal, which operates in linear time, O(n). However, a significant consideration is that the BST must be kept balanced to avoid performance degradation, which would otherwise lead to a worst-case time complexity of O(n) for operations. This can be mitigated by employing a self-balancing variant of a BST, such as an AVL tree, to ensure consistent performance.

In conclusion, while each data structure offers unique benefits, the binary search tree is the most suitable choice for the requirements of this project. It efficiently supports the necessary operations while maintaining the sorted order of courses, which is crucial for the task at hand. The use of a self-balancing BST would ensure that the tree remains balanced, providing optimal performance across all operations. This makes the BST the recommended data structure for implementation in the next phase of coding the course management program at ABCU.

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| Insertion | O(1) | Average: O(1) | Average O(log n) |
|  |  | Worst: O(n) | Worst: O(n) |
| Search | O(n) | Average: O(1) | Average: O(log n) |
|  |  | Worst: O(n) | Worst: O(n) |
| Sorting | O(n log n) |  | O(n) in order traversal |
| Memory | O(n) | O(n) | O(n) |