Mitchell Fontaine

CS-300

08/07/2024

6-2: Project One

Runtime Analysis:

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| Loading Data and Reading | O(n) | O(n) | O(n log n) |
| Searching | O(n) | O(1) | O(log n) |
| Sorting | O(n log n) | O(n log n) | O(n) |

Advantages and Disadvantages:

Vector:

Vectors offer several advantages, including simplicity and ease of use, making them great for straightforward applications. They support random access, allowing for constant-time access to any element if the index is known. Additionally, vectors are memory-efficient, with lower overhead than complex data structures like trees or hash tables, and store elements in contiguous memory locations, making them cache-friendly. However, vectors also have some disadvantages. Searching for an element can be inefficient, requiring O(n) time on average, making them less suitable for applications with frequent searches. Furthermore, sorting a vector takes O(n log n) time, which can be a drawback if frequent sorting is necessary, and maintaining a sorted order dynamically can be challenging and inefficient with vectors.

Hash Table:

Hash tables offer several advantages, including fast lookups with an average-case time complexity of O(1) for insertions, deletions, and searches, which makes them great for applications requiring frequent searches and updates. Also, hash tables can dynamically resize, maintaining efficient operations as the number of elements grows. However, hash tables also have some disadvantages. One major drawback is their inability to maintain order among elements, making sorting a costly O(n log n) operation requiring additional steps to convert the data into a list. Furthermore, hash tables can have significant memory overhead due to the need for extra space for buckets and collision handling, and poorly implemented hash functions can lead to clustering, increasing collisions, and degrading performance.

Binary Search Tree:

Binary Search Trees (BSTs) offer efficient search, insert, and delete operations with a time complexity of O(log n), and provide efficient sorting capabilities through in-order traversal. Their ordered structure facilitates range queries and ordered iterations, which makes them great for applications that require frequent sorted data retrievals. BSTs are more complex to implement though, as they require balancing mechanisms to maintain efficiency, and typically use more memory than simpler data structures like vectors.

Recommendation:

Based on the big O analysis, the recommended data structure for this project is the Binary Search Tree (BST). While vectors are suitable for small, static datasets with infrequent searches, and hash tables excel in scenarios with frequent insertions and lookups they have no need for ordered data, the BST offers a balanced approach that meets the program's requirements. It provides efficient searching, sorting, and ordered data retrieval, making it ideal for managing and querying course information dynamically. Although BSTs are more complex to implement, their performance benefits justify the added complexity, making them the best choice for this project.