Nate Bennett

Ramsey Kraya

Analysis and Design

12 April 2023

6-2 Submit Project One Evaluation

Vector

Cost per line of code: 1.

Open file: 1 time.

Read data from file: n times (once for each course).

Parse each line: n times (once for each course).

Create course object: n times (once for each course).

Add course object to vector: n times (once for each course).

Total running time: 5n + 1.

Big O: O(n).

Hash Table

Cost per line of code: 1

Open file: 1 time

Read data from file: n times (once for each course).

Parse each line: n times (once for each course).

Create course object: n times (once for each course).

Add course object to hash table: n times (once for each course).

Total running time: 5n + 1.

Big O: O(n).

Tree

Cost per line of code: Depends on the type of tree.

Open file: 1 time.

Read data from file: n times (once for each course).

Parse each line: n times (once for each course).

Create course object: n times (once for each course).

Insert course object into tree: n times (once for each course).

Total running time: 5n + 1.

Big O: Depends on the type of tree used O(n) could be used for binary search tree, and an O(log n) could be used for s balanced search tree

The run time and memory of data structures that could be utilized to address the requirements would be the following. For a vectors runtime, linear time, or O(n), can be used to read data from a file and parse each line, where n is the number of lines in the file. Since each course object must be constructed independently for each line, creating course objects would likewise need linear time, or O(n). For a vector's memory, It can take a sizable continuous block of memory to store all the course objects because a vector saves elements in a contiguous block of memory. Since each object must be saved separately in the vector, the memory usage would be O(n) in proportion to the number of course objects. For a hash tables runtime, It would still take linear time, O(n), to read data from a file and parse each line. However, because each course object must be constructed independently for each line, constructing course objects would likewise need linear time, O(n). The average constant time complexity of hash table operations, such as insertion, deletion, and retrieval, is O.(1). For storing hash values and keeping the hash table structure, a hash table needs extra RAM. The hash table size and the number of course objects will affect how much memory is used, however on average, it can be said that the complexity is O(n), or linear. A tree data structure's runtime is dependent on the particular tree implementation being utilized. For insertion, deletion, and retrieval operations, a binary search tree can have an average case time complexity of O(log n). However, in the worst scenario (for example, a tree that isn't balanced), the time complexity can drop to O(n). As with the earlier data structures, reading data from a file and parsing each line would continue to require linear time, O(n). Since each course object must be constructed independently for each line, creating course objects would likewise need linear time, or O(n). The particular tree implementation utilized affects how much memory a tree uses. For instance, in the worst case, a binary search tree can have a space complexity of O(n). (e.g., a skewed tree). The space complexity of balanced tree implementations like AVL trees and Red-Black trees, however, can be O(n log n) or better.

Utilizing a vector has some benefits, including being ideal for small to medium-sized datasets, being usually simple and easy to construct, and having random access to elements with constant time complexity. However, the insertion and deletion procedures, particularly for big datasets, can be inefficient since they may require moving of items, and in the worst-case scenarios, identifying elements may require linear search, resulting in O(n) time complexity. Using a hash table can offer some benefits, such as being appropriate for substantial datasets with efficient searching, as well as being able to provide consistent time complexity for insertion, deletion, and retrieval operations on average. The hashing function may introduce collisions, which can impair performance, and hash tables might require additional memory to store hash values and maintain the hash table structure. The use of a tree structure can be advantageous because in can provide effective insertion, deletion, and retrieval operations in a balanced tree structure, suitability for substantial datasets, and can preserve the order in which the given elements might be sorted. A tree structure can, however, also be difficult to construct, have the potential to degrade performance in unbalanced trees, and possibly require additional memory to maintain the tree structure.