Nicholas Wyrwas

CS-300-15095-M01

Mr. Nathan Lebel

12/07/2024

6-3 Project One

**Pseudocode for Data Structures**

Pseudocode to read data into each data structure first opens the file and reads each line. If it cannot open the file, error message appears. For each line, course id, title, and prerequisites are parsed. If the line is misinterpreted, an error is logged and the line is skipped. When the prerequisites are validated (every prerequisite already exists in the structure), a course object is created and put into the associated data structure (vector, hash table, or BST). To add a course to the vector, it is necessary to write it to the end of the vector which is usually an O(1) operation. But by re-sizing the vector (if required), an insertion might be O(n) at best. The course is read from the hash table with the computed hash key (a regular O(1) operation). insertion would become O(n) if the linked list handles a hash collision (which happens extremely rarely). And finally, in the BST, insertion is just about getting the correct tree position. At worst, if the tree was unbalanced, that might take O(n) for every insertion, and at worst, it will require O(n2) time to insert all courses. When printing course information, program prints all courses alphabetically. For the vector, this consists of sorted the courses and is an O(n log n) operation. Sorting for the hash table would entail mining and filtering the data, and will take O(n log n). For the BST, an in-order traversal ensures courses are printed in the order sorted (o(n) times). It also has a menu button to load the data, print courses or see details of individual courses. The menu takes the user’s input and calls the desired functions based on the user’s choice.

**Big O Analysis**

For the vector data model, for reading the file (O(n)) and creating course objects (O(1) per course), the worst-case time complexity of loading data is O(1). Inserting each course into the vector is O(1), on average (reducing the vector to a given size might sometimes make it O(n). Thus the total time required to fit the courses into the vector is O(n). For the hash table, reading the file and writing course objects will also still take O(n), but inserting will normally be O(1) time as long as no collisions have occurred. In the worst-case, if a collision takes place, insertion might fall down to O(n), but this rarely happens. In that case, the total worst-case time complexity of getting the courses into the hash table is O(n).

**Memory Usage**

The memory consumption for all three data structures is O(n), since each course object occupies a node/ element of the structure. The vector consists of contiguous memory, the hash table contains buckets (which may require more memory to accommodate collisions), and the BST contains nodes whose children have links to other nodes.

**Advantages and Disadvantages of Each Data Structure**

The vector is simple and low memory cost, so it is quite easy to implement. But it isn't search friendly and takes more time to resize. It also doesn’t have any order of its own, so you must sort it when printing courses. The hash table is fast to find average-case lookups (O(1)) and is efficient to search courses by course number. But it takes up more memory (by hash buckets) and is not ordered by default (sorting is needed for printing). It can also get sucked up by collisions of hash functions, but it rarely happens in a good hash function.

The BST keeps courses sorted and, in the well-balanced case, ensures fast searching, inserting and deleting (O(log n)). But, at worst, if the tree is out of balance, it’s slow down to O(n) and won’t be reliable for huge datasets.

**Recommendation**

According to the Big O analysis, hash table is the best data structure for this use case. It provides O(1) average-case insertion and lookup performance, so it’s perfect for running hundreds of courses. The BST would be great for storing sorted data, but at its worst-case (O(n) for insertion and traversal) is not really good unless it is balanced. The vector is very easy to implement but not searchable and has to be sorted to keep things neat. The hash table thus is a good compromise between speed and usability, and I would suggest it for this project.