Karina Aronov

CS-300

Professor Ostrowski

June 16, 2024

6-2 Project One

**Vector Data Structure**

For the vector data structure, the pseudocode involves opening the file, checking if the file is open, reading each line until the end of the file, parsing each line into its components (courseNumber, courseName, and preReqs), creating a new course object with the parsed data, adding the course object to the list, and finally closing the file. This program uses a while loop to run over each line while performing the crucial activities of reading and parsing the file. For every n, which is the number of courses, this indicates that the loop's operations will run n times. Reading the file and producing course objects has an O(n) overall complexity since each operation in the loop has a constant time complexity. Due to this, the process's worst-case vector running time is linear, or O(n).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Opening the file | O(1) | 1 | O(1) |
| File open check | O(1) | 1 | O(1) |
| While loop (reading n lines) | O(1) | n | O(n) |
| * Read line from file | O(1) | n | O(n) |
| * Parse line | O(1) | n | O(n) |
| * Create a new Course | O(1) | n | O(n) |
| * AddCourse | O(1) | n | O(n) |
| Closing the file | O(1) | 1 | O(1) |
| **Total worst-case running time** |  |  | **O(n)** |

**Hash Table Data Structure**

For the hash table data structure, the pseudocode is slightly more complicated. It also starts by opening the file, checking if the file is open, and initializing a list to keep track of all course numbers. As the file is read line by line, each line is parsed into courseNumber, courseName, and preReqs. The pseudocode then checks for missing course numbers or names and validates the prerequisites against the list of all course numbers. If the prerequisites are valid, the course number is added to the list of all course numbers, a new course object is created, and this object is added to the hash table using the AddCourse function. The loop's operations, such as insertion and validation, happen n times. Because hash functions are efficient, adding to the hash table typically takes the same amount of time as creating and validating course objects. Therefore, if there are a fixed and limited number of prerequisites for each course, the overall complexity stays O(n). This indicates that utilizing a hash table to create course objects and reading the file will have a worst-case running time of linear, O(n).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Opening the file | O(1) | 1 | O(1) |
| File open check | O(1) | 1 | O(1) |
| Define List of Strings allCourseNumbers | O(1) | 1 | O(n) |
| While loop (reading n lines) | O(1) | n | O(n) |
| * Read line from file | O(1) | n | O(n) |
| * Parse line | O(1) | n | O(n) |
| * Empty check for courseNumber or courseName | O(1) | n | O(n) |
| * If preReqs are provided | O(1) | n | O(n) |
| * For each preReq in preReqs | O(1) | n | O(n) |
| * Check if preReq in allCourseNumbers | O(1) | n | O(n) |
| * Add courseNumber to allCourseNumbers | O(1) | n | O(n) |
| * Create a new Course | O(1) | n | O(n) |
| * AddCourse | O(1) | n | O(n) |
| Closing the file | O(1) | 1 | O(1) |
| **Total worst-case running time** |  |  | **O(n)** |

**Binary Search Tree Data Structure**

For the binary search tree (BST) data structure, the process starts similarly by opening the file, checking if the file is open, and initializing a list to keep track of all course numbers. The file is read line by line, each line is parsed, and missing course numbers or names are checked. Prerequisites are validated against the list of all course numbers, and if valid, the course number is added to the list. A new course object is created and inserted into the BST. In a BST, the insertion operation typically takes logarithmic time, or O(log n). In the worst scenario, however, the insertion time may drop to O(n), or linear time, when the tree becomes unbalanced. The average complexity of reading the file and constructing course objects using a BST is O(n log n), but in the worst case, it can be O(n^2) if the tree is significantly disorganized, since each line is processed and a course object is produced and inserted n times.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Line Cost** | **# Times Executed** | **Total Cost** |
| Opening the file | O(1) | 1 | O(1) |
| File open check | O(1) | 1 | O(1) |
| Define List of Strings allCourseNumbers | O(1) | 1 | O(n) |
| While loop (reading n lines) | O(1) | n | O(n) |
| * Read line from file | O(1) | n | O(n) |
| * Parse line | O(1) | n | O(n) |
| * Empty check for courseNumber or courseName | O(1) | n | O(n) |
| * If preReqs are provided | O(1) | n | O(n) |
| * For each preReq in preReqs | O(1) | n | O(n) |
| * Check if preReq in allCourseNumbers | O(1) | n | O(n) |
| * Add courseNumber to allCourseNumbers | O(1) | n | O(n) |
| * Create a new Course | O(1) | n | O(n) |
| * Insert (average case) | O(long n) | n | O(n log n) |
| Closing the file | O(1) | 1 | O(1) |
| **Total worst-case running time** |  |  | **O(n long n) (average), O(n^2)** |

**Advantages & Disadvantages**

* **Vector Data Structure**

The main reason vectors are useful is because they are straightforward and simple to utilize. Vectors can be easily implemented and automatically handle dynamic scaling, making them an adaptable tool for handling different data volumes without the need for human intervention. In addition, vectors internally handle memory allocation and deallocation, lowering the possibility of memory leaks and making memory management easier for developers (Mudadla, 2023). Vectors do, however, have a number of disadvantages, especially in terms of performance. The slow search performance, with an O(n) time complexity, is the most noticeable disadvantage. This means that as the number of courses increases, the amount of time needed to locate a particular course increases linearly, potentially becoming inefficient as the dataset expands. An alphanumerically ordered list of courses must be produced by additional sorting processes because vectors do not preserve order by default. An O(n log n) overhead is added during the sorting process. Because they require shifting components and have an O(n) time complexity, insertion and deletion operations are also expensive, particularly when items need to be added or removed from the middle of the vector (Mudadla, 2023).

* **Hash Table Data Structure**

There are several benefits to hash tables, especially when it comes to performance. With an average time complexity of O(1) for search, insert, and delete operations, they offer quick access to data. This performance is made possible by the use of hash functions, which permit direct key-value mapping and speedy course information lookup by course number. Because hash tables involve constant time operations, they are also ideal for managing big datasets. Collision handling strategies like open addressing and chaining guarantee that the hash table maintains its efficiency as the number of entries increases (GeeksforGeeks, 2024b). Hash tables have a few downsides despite their advantages in terms of effectiveness. Their need for extra memory to manage collisions and preserve the table structure may provide a problem in settings with limited memory. Hash tables do not preserve any underlying order between the items, a separate sorting technique is required to show courses in alphabetical order. Compared to vectors, implementing a hash table is also more complicated, particularly when considering the collision prevention methods that must be included to guarantee stability (Tutorlix, 2023).

* **Binary Search Tree Data Structure**

The specific advantage of using binary search trees (BSTs) is that they keep the components arranged in a sorted manner. Because of this fundamental order, elements in the BST will be displayed in alphabetical order during an in-order search. This is very helpful for publishing courses alphabetically without the need for extra sorting (GeeksforGeeks, 2024a). If the tree maintains its balance, BSTs also give effective average time complexities for search, insert, and delete operations, which are normally O(log n). Compared to vectors and hash tables, they are more difficult to put into place. In the worst-case situation, a BST's performance might quickly begin to decline to O(n) if the tree becomes unbalanced. The advantages of using a BST may be outweighed by this possible performance loss. Moreover, compared to simpler data structures like vectors, each node in a BST needs pointers to its offspring, resulting in extra memory overhead (GeeksforGeeks, 2024a).

**Recommendation**

Based on the information above, the hash table is recommended because it provides quick access to course information by course number. It is useful for finding course information fast because it offers O(1) average time complexity for search, insert, and remove operations. Although sorting cannot be done automatically by hash tables, it can be done independently when printing every course, which results in a more effective combination of processes.

**Resources**

GeeksforGeeks. (2024a, February 27). *Applications, Advantages and Disadvantages of Binary Search Tree*. GeeksforGeeks. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/

GeeksforGeeks. (2024b, May 8). *Hash table data structure*. GeeksforGeeks. https://www.geeksforgeeks.org/hash-table-data-structure/

Mudadla, S. (2023, October 31). What is Vector Database?advantages and disadvantages of vector databases.Explain what companies are providing vector databases?In what kind of applications we can use this database? *Medium*. https://medium.com/@sujathamudadla1213/what-is-vector-database-advantages-e637fa7b80ef

Tutorlix. (2023, October 7). What are the advantages and disadvantages of hashing. *Medium*. https://medium.com/@tutorlix69/what-are-the-advantages-and-disadvantages-of-hashing-8ba6fd8fadef