Ivette Cerpa

Dr. Webb

CS-300

February 25, 2024

|  |  |  |  |
| --- | --- | --- | --- |
| **Runtime Analysis of Reading the File and Creating Course Objects with a Vector** | | | |
| Code | Line Cost | # Times Executes | Total Cost |
| open file | 1 | 1 | 1 |
| if (file is not open) | 1 | 1 | 1 |
| -display error message | 1 | 1 (worst case) | 1 (worst case) |
| -end program | 1 | 1 (worst case) | 1 (worst case) |
| while (there are lines in the file) | 1 | n+1 | n+1 |
| split line into parts | 1 | n | n |
| if (parts size is less than 2) | 1 | n | n |
| -print error message | 1 | 1 (worst case) | 1 (worst case) |
| -end program | 1 | 1 (worst case) | 1 (worst case) |
| courseID = first element in parts | 1 | n | n |
| course name = second element in parts | 1 | n | n |
| Appending remaining elements to prerequisites vector | 1 | n | n |
| create new course object | 1 | n | n |
| add the course object to the courses vector | 1 | n | n |
| close the file | 1 | 1 | 1 |
| Best Case Runtime | | | **O(n)** |
| Worst Case Runtime | | | **O(n)** |

**Advantages**: The use of a vector as the data structure in this program brings several benefits. First and foremost, a vector’s size is dynamic, meaning it can adjust its size to accommodate the varying number of courses. This flexibility is key when the total number of courses isn’t known beforehand. Additionally, vectors offer constant time access to their elements, a feature that proves invaluable when we need to access courses for printing. Lastly, even though vectors consume more memory due to their dynamic resizing capability, they manage their own memory allocation. This is advantageous for our program, which doesn’t require intricate control over memory allocation.

**Disadvantages**: Despite its advantages, using a vector also presents some challenges. While appending elements to the end of the vector is efficient, we encounter difficulties when adding or removing elements from the beginning or middle. This situation arises when we go to sort the courses alphanumerically. The issue stems from the fact that modifying the beginning or middle of the vector necessitates shifting of elements, a process that can be time-consuming.

|  |  |  |  |
| --- | --- | --- | --- |
| **Runtime Analysis of Reading the File and Creating Course Objects with a Hash Table** | | | |
| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| Open file | 1 | 1 | 1 |
| Check if file is not open | 1 | 1 | 1 |
| -Display error message (if file not open) | 1 | 1 (worst case) | 1 (worst case) |
| -End program (if file not open) | 1 | 1 (worst case) | 1 (worst case) |
| While there are lines in the file | 1 | n+1 | n+1 |
| Splitting each line into a vector | n | n | n2 |
| Checking if size of vector is less than two | 1 | n | n |
| -Printing an error message | 1 | 1 (worst case) | 1 (worst case) |
| -Ending program | 1 | 1 (worst case) | 1 (worst case) |
| Setting courseId to first element in vector | 1 | n | n |
| Setting courseName to second element in vector | 1 | n | n |
| Appending remaining elements to prerequisites vector | 1 | n | n |
| Calling hashing function | 1 | n | n |
| Appending course object to hash table | 1 | n | n |
| Best Case Runtime | | | **O(n)** |
| Worst Case Runtime | | | **O(n2)** |

**Advantages**: One of the primary benefits of using a hash table in this program is its constant time complexity for operations such as searching, inserting, or deleting an element. This feature enables rapid access to course information whenever necessary.

**Disadvantages**: However, using a hash table also presents certain challenges. A key issue is the handling of collisions. Although I plan to use chaining to address this, a worst-case scenario involving an excessive number of collisions could slow down search times. The effectiveness of a hash table also depends on the number of courses to be added. For a small number of courses, the memory usage of a hash table might outweigh its benefits. The choice of hashing function is another factor to consider. A simple hashing function may suffice for a small number of courses, but a larger course load would necessitate a more robust hashing function.

|  |  |  |  |
| --- | --- | --- | --- |
| **Runtime Analysis of Reading the File and Creating Course Objects with a BST** | | | |
| Code | Line Cost | # Times Executes | Total Cost |
| Open file | 1 | 1 | 1 |
| If file is not open | 1 | 1 | 1 |
| Display error message | 1 | 1 (worst case) | 1 (worst case) |
| End program | 1 | 1 (worst case) | 1 (worst case) |
| While there are lines in the file | 1 | n | n |
| Split line into parts | 1 | n | n |
| If size of vector is less than two | 1 | n | n |
| Print an error message | 1 | 1 (worst case) | 1 (worst case) |
| End program | 1 | 1 (worst case) | 1 (worst case) |
| Set courseId to first element in vector | 1 | n | n |
| Set courseName to second element in vector | 1 | n | n |
| If more than two elements in vector | 1 | n | n |
| For each element starting at third element until end of vector | 1 | n | n |
| Add each remaining element to the prerequisites vector | 1 | n | n |
| Create a Course object with courseId, courseName, and prerequisites | 1 | n | n |
| Call insert method of BinarySearchTree Class to insert the Course object into the tree | log n (average case), n (worst case) | n | n log n (average case), n2 (worst case) |
| Close file | 1 | 1 | 1 |
| Best Case Runtime | | | **O(n log n)** |
| Worst Case Runtime | | | **O(n2)** |

**Advantages**: One of the key advantages of using a BST in our program is the inherent ordering of data. This characteristic proves particularly beneficial when we need to print the courses alphanumerically. Furthermore, operations such as searching, adding, or deleting a node are highly efficient when the BST is balanced and ordered.

**Disadvantages:** On the downside, maintaining the balance of the tree requires extra care. In its worst-case scenario, a BST can degenerate into a linked list, leading to very inefficient traversal. Additionally, a BST tends to consume more memory due to the need to track two pointers for each node’s child nodes. Lastly, unlike operations with hash tables and vectors, none of the node access operations in a BST work in constant time.

**Overall Recommendation**

There are a lot of factors to consider when making a recommendation for the best data structure for this program. Ultimately, I think the biggest part to consider is the size of the data set in the file.

**Small to Medium Dataset**: For a small or medium dataset, a Binary Search Tree (BST) could be an excellent choice. The inherent ordering of elements in a BST is a significant advantage for this program, which aims to load data quickly, perform rapid searches for specific courses, and print either a specific course’s information or an alphanumerically sorted course list. A BST can efficiently handle these tasks, and if the dataset is small, maintaining the tree’s balance shouldn’t pose a significant challenge.

**Large Dataset:** In this specific case, a hash table could be more suitable. Although a hash table doesn’t inherently maintain the order of its elements, the program only needs to sort the data once since it loads the data just once. The trade-off of implementing a sorting operation could be worthwhile in exchange for constant time access to elements. Given a robust hashing function, the search times for specific courses will be highly efficient, as will the time to initially load the data.