MacRichard Risma

CS 300 Project One

**MENU PSEUDOCODE**

START

DECLARE and INITIALIZE variables

WHILE user choice does not equal 4

DISPLAY menu options

// 1 = Load data, 2 = print all courses, 3 = print specific course, 4 = exit

GET input for user choice

IF user choice equals 1

LOAD data into data structure

ELSE IF user choice equals 2

PRINT courses in alphanumeric order

ELSE IF user choice equals 3

GET input for course number for course

PRINT course name and prerequisites if any

ELSE

PRINT “Invalid input”

PRINT “Good bye”

END

**VECTOR PRINT COURSES PSEUDOCODE**

START

DECLARE and INITIALIZE variables

FOR course in courses vector

SORT courses using selection sort logic by course number from lowest to highest

FOR course in courses vector

PRINT course number, course name, prerequisites

END

**HASH TABLE PRINT COURSES PSEUDOCODE**

START

DECLARE and INITIALIZE variables

SORT using sort() function to sort array by course number from lowest to highest

LOOP through array list of courses

FOR course in courses array

PRINT course number, course name, prerequisites

END

**BST PRINT COURSES PSEUDOCODE**

START

DECLARE and INITIALIZE variables

PRINT courses using inOrder() function

// if node does not equal NULL

// RECURSION through inOrder(left of current node)

// PRINT course number, course name, course prerequisites

// RECURSION through inOrder(right of current node)

END

**CREATING DATA STRUCTURE ANALYSIS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **Line Cost** | **# Times Executed** | **Total Cost** |
| DECLARE a structure that holds course information | 1 | 1 | 1 |
| DECLARE string for course number | 1 | 1 | 1 |
| DECLARE string for course name | 1 | 1 | 1 |
| DECLARE string for prerequisites | 1 | 1 | 1 |
|  |  | **Total Cost** | 6 |
|  |  | **Runtime** | O (1) |

**VECTOR RUNTIME ANALYSIS TO LOAD DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **Line Cost** | **# Times Executed** | **Total Cost** |
| OPEN FILE | 1 | 1 | 1 |
| WHILE end of file is not reached | 1 | n | n |
| Input file stream variable >> string variable | 1 | n | n |
| DECLARE stringstream variable “ss” | 1 | n | n |
| PASS string variable into “ss” | 1 | n | n |
| INVOKE getline with “,” as the delimiter with “ss” and String substring variable | 1 | n | n |
| ASSIGN first substring to course number variable | 1 | n | n |
| INVOKE getline with “,” as the delimiter with “ss” and String substring variable | 1 | n | n |
| ASSIGN second substring to course name variable | 1 | n | n |
| WHILE “ss” is still good | 1 | n | n |
| INVOKE getline with “ss” and String substring variable | 1 | n | n |
| ASSIGN substring to one whole string named course prerequisites | 1 | n | n |
| PUSH BACK Course into Courses vector | 1 | n | n |
| CLOSE file | 1 | 1 | 1 |
|  |  | **Total Cost** | 12n + 2 |
|  |  | **Runtime** | O(n) |

**HASH TABLE RUNTIME ANALYSIS TO LOAD DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **Line Cost** | **# Times Executed** | **Total Cost** |
| OPEN FILE | 1 | 1 | 1 |
| WHILE end of file is not reached | 1 | n | n |
| Input file stream variable >> string variable | 1 | n | n |
| DECLARE stringstream variable “ss” | 1 | n | n |
| PASS string variable into “ss” | 1 | n | n |
| INVOKE getline with “,” as the delimiter with “ss” and String substring variable | 1 | n | n |
| ASSIGN first substring to course number variable | 1 | n | n |
| INVOKE getline with “,” as the delimiter with “ss” and String substring variable | 1 | n | n |
| ASSIGN second substring to course name variable | 1 | n | n |
| WHILE “ss” is still good | 1 | n | n |
| INVOKE getline with “ss” and String substring variable | 1 | n | n |
| ASSIGN substring to one whole string named course prerequisites | 1 | n | n |
| SET initial value | 1 | 1 | n |
| SET hash multiplier | 1 | 1 | 1 |
| CONVERT string object to string using c\_string | 1 | 1 | 1 |
| GET ASCII number value for each character in the string of course number | 1 | n | n |
| DETERMINE bucket using course number as key | 1 | n | n |
| INSERT course into bucket using Linear probing method | 1 | n | n |
| CLOSE file | 1 | 1 | 1 |
|  |  | **Total Cost** | 16n + 5 |
|  |  | **Runtime** | O(n) |

**BST RUNTIME ANALYSIS TO LOAD DATA**

|  |  |  |  |
| --- | --- | --- | --- |
| **CODE** | **Line Cost** | **# Times Executed** | **Total Cost** |
| OPEN FILE | 1 | 1 | 1 |
| WHILE end of file is not reached | 1 | n | n |
| Input file stream variable >> string variable | 1 | n | n |
| DECLARE stringstream variable “ss” | 1 | n | n |
| PASS string variable into “ss” | 1 | n | n |
| INVOKE getline with “,” as the delimiter with “ss” and String substring variable | 1 | n | n |
| ASSIGN first substring to course number variable | 1 | n | n |
| INVOKE getline with “,” as the delimiter with “ss” and String substring variable | 1 | n | n |
| ASSIGN second substring to course name variable | 1 | n | n |
| WHILE “ss” is still good | 1 | n | n |
| INVOKE getline with “ss” and String substring variable | 1 | n | n |
| ASSIGN substring to one whole string named course prerequisites | 1 | n | n |
| INSERT course into Tree using insert() function/method | 1 | n | O (log n) |
| CLOSE file | 1 | 1 | 1 |
|  |  | **Total Cost** | 11n + 2 + O (log n) |
|  |  | **Runtime** | O (n) + O (log n) |

**ANALYSIS**

All three data structures can be used for this task, and all have their advantages and disadvantages. Based on the complexity of the set up the code, a Vector would be the simplest, then a BST, and then a Hash Table. Now, memory and time complexity are the two main factors when analyzing algorithms. Based on the analysis of loading the data, both the Vector and the Hash Table have a linear time complexity, while the BST has a logarithmic time complexity. However, taking into consideration the other parts of total lines of code for memory and sorting the data, Hash Table would both incorporate a logarithmic time complexity to sort the data, and the logic would take up more lines of code, and the Vector would have a quadratic time complexity when using the selection sort logic. While the BST, despite having a logarithmic time complexity with its recursion when adding the data, takes up less lines of code, which takes up less memory. For the data structure, they all have the same set up for the courses data structure, but the Hash Table and BST would have another data structure for the nodes. Which again, takes up more lines of code and memory.

**RECOMMENDATION**

Based on the prompt of how the code will be used by ABCU’s advising program, they want to search and print the information of a specific course or print all the courses. For this small batch of courses, a Vector would be the easiest to write and complete the necessary needs, despite having the worst run time complexity. However, despite the simplicity of a Vector, a BST would be better in the long run, because it does take up less lines of code/memory, it is scalable, and has better run time complexities.

**RESOURCES**

“Analysis of Algorithms | Big-O Analysis - GeeksforGeeks.” *GeeksforGeeks*, 28 Feb. 2018, [www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/](http://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/).

‌ “Std::Sort() in C++ STL.” *GeeksforGeeks*, 2 Apr. 2016, [www.geeksforgeeks.org/sort-c-stl/](http://www.geeksforgeeks.org/sort-c-stl/).

‌ “ZyBooks.” *Learn.zybooks.com*, learn.zybooks.com/zybook/CS-300-T5501-OL-TRAD-UG.23EW5/chapter/3/section/7. Accessed 5 June 2023.

‌