Jake Grubert

SNHU

CS 300

**Project One**

Vector Pseudocode

Initialize an empty vector called "courses."

Open the file named "Course Information."

If the file is not open, display an error message and exit the program.

While not reaching the end of the file:

Read a line from the file and store it in a string called "line."

Split "line" into a list named "tokens" using spaces.

If the length of "tokens" is less than 2, output an error message and move to the next line.

For each "token" in "tokens" starting from the third one to the end:

If there is no line in the file that begins with "token," display an error message and move to the next line.

Add "tokens" to the "courses" vector.

Close the file.

For each "course" in the "courses" vector:

Create a new Course object "c" with the course number as "course"[0], the title as "course"[1], and prerequisites as the remaining elements of "course."

Add "c" to the end of the "courses" vector.

Input a string named "courseNumber."

For each "course" in the "courses" vector:

If the course number of "course" matches "courseNumber":

Print the course number, title, and prerequisites of "course."

Break out of the loop.

If no course is found, display an error message.

Hash Table Pseudocode:

File Reading:

Utilize fstream to open a file.

If the file operation returns -1,

showcase an error message.

Else, proceed with the following steps.

While the file has not reached the End Of File (EOF):

Read each line.

If there are fewer than two values in a line, trigger an error.

Else, extract parameters.

If there are three or more parameters, continue processing.

Close the file.

Course Object Creation:

Initialize a HashTable.

Insert the bid into the table.

Iterate through the file until reaching the End Of File.

For each line in the file:

Extract the first and second values.

Create a temporary item to store these values.

If there is a third value,

incorporate it into the current value.

Return

Print HashTable Contents:

Prompt the user for input and assign it to a key.

If the key is found in the HashTable:

Display the course information.

For each course prerequisite, display the prerequisite course information.

Else

If the key is not found,

showcase an error message.

Return from Operation.

Tree Data Pseudocode:

Utilize fstream to enable file access.

If the file operation returns -1,

display an ERROR indicating that the file was not found.

Else, proceed with the following steps.

While the file has not reached the End Of File (EOF):

Read each line.

If there are fewer than two values in a line, return an ERROR.

Else, read and extract parameters.

If there are three or more parameters, continue processing.

Closing the File

Creating Course Objects Structure:

Create a BinarySearchTree.

Insert bids into the tree.

While not reaching EOF, loop through the file.

For each line in the file:

Extract the first and second values.

Add Course ID and Course Name.

If a third value exists,

add prerequisites until a newline is found.

Searching and Printing from the Tree:

Prompt the user for input.

Create a search and print method.

If the root is not null:

Traverse left.

If the node matches the Course ID:

Display course information.

For each course prerequisite,

print prerequisite course information.

Else,

traverse right.

If the node matches the Course ID:

Display course information.

For each course prerequisite,

print prerequisite course information.

Return from Operation.

Menu Pseudocode:

While the user's choice is not equal to 9:

Print the following menu options:

Print"1. Load Data"

Print "2. Print Course List"

Print "3. Print Course"

Print "4. Exit"

Prompt the user for input.

Switching based on user menu input:

Case 1:

Load course data in the program.

Break from the switch.

Prompt the user for input.

Case 2:

Print the course number and name.

Break from the switch.

Prompt the user for input.

Case 3:

Prompt the user to enter the course number to search.

Obtain the user input for the course number.

Print the course number, course name, and prerequisite course numbers.

Break from the switch.

Prompt the user for input.

Case 4:

Print "Goodbye."

Exit the loop.

Continue looping until the user chooses to exit the program.

Course List Alphanumeric Pseudocode:

Initialize mid to low + (high - low) / 2.

Set pivot to courseName(mid).

While courseName(low) is less than pivot:

Increment low by 1.

ENDWHILE

While pivot is less than courseName(high):

Decrement high by 1.

ENDWHILE

If low is greater than or equal to high,

return.

Else

Create a temporary variable temp and set it to courseName(low).

Set courseName(low) to courseName(high).

Set courseName(high) to temp.

Increment low by 1.

Decrement high by 1.

ENDIF

In the main function:

Call the quicksort function with parameters courseName, 0, and SIZE - 1.

Display "Sorted in alphabetical order: ".

For each course:

Display course information.

End the main function.

Vector:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Time Executed** | **Total** |
| Open | 1 | 1 | 1 |
| While not EOF read each line | 1 | n | n |
| If less than two values return error | 1 | n | n |
| Else Continue | 1 | n | n |
| Initalize course vector | 1 | 1 | 1 |
| For each line | 1 | n | n |
| Add value to vector | 1 | n | n |
| Push item back | 1 | n | n |
| **Total Cost** | | | 7n+2 |
| **Runtime** | | | 0(n) |

Hash Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Time Executed** | **Total** |
| Open | 1 | 1 | 1 |
| While not EOF read each line | 1 | n | n |
| If less than two values return error | 1 | n | n |
| Else Continue | 1 | n | n |
| Create HashTable | 1 | 1 | 1 |
| Insert big to table | 1 | n | n |
| Loop Through File | 1 | n | n |
| For each line | 1 | n | n |
| Create temporary held item | 1 | n | n |
| **Total Cost** | | | 8n+2 |
| **Runtime** | | | 0(n) |

Search Tree:

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Line Cost** | **# Time Executed** | **Total** |
| Open | 1 | 1 | 1 |
| While not EOF read each line | 1 | n | n |
| If less than two values return error | 1 | n | n |
| Else Continue | 1 | n | n |
| Create Search Tree | 1 | 1 | 1 |
| Insert bids to tree | 1 | n | n |
| While not EOF | 1 | n | n |
| Loop through file | 1 | n | n |
| For each line in file | 1 | n | n |
| for first and second, add courseID and name | 1 | n | n |
| If third value available, add prerequisite until newline | 1 | n | n |
| **Total Cost** | | | 10n+2 |
| **Runtime** | | | 0(n) |

Vector:

While the vector structure excels in quickly reading files and adding course objects efficiently, its drawback lies in reduced efficiency when searching for a specific course. This limitation arises from the need to scan through each course item, leading to slower search operations.

Hash Table:

The hash table facilitates remarkably swift searches by utilizing keys to map to items in the search list. Nonetheless, it's crucial to recognize that hash tables lack inherent sorting, and arranging the courses in alphanumeric order would demand a notably extended processing time.

Search Tree:

A key advantage of a Binary Search Tree lies in its efficient organization and sorting of items, simplifying search operations. Although this process is somewhat slower compared to a hash table, it outperforms the use of vectors. Nevertheless, a drawback of the binary search tree is the time needed for making modifications.

Recommendation:

Taking into account the advantages and disadvantages of each data structure, I recommend utilizing vectors for this project. Among the three options, vectors have the lowest runtime, specifically at 7n+2. Moreover, they excel in both the speed of file reading and object addition. I contend that the efficiency in these operations outweighs the time required for searching and printing course information.