Luis Diaz

CS 300

12 October 2025

Project One

Pseudocode

Main Function

Read command arguments

Stores arguments in CSV path. Uses default CSV path if there are no arguments.

Loop when choice is not 9.

Output menu

Get user input, both menu choice and data to be used.

Validate user input.

If the choice is not 1-4 or 9, throw an error.

If choice is 1.

Call file parser and load data into each data structure.

(Binary Search Tree, Hash Table, Vector)

If choice is 2.

Validate the list

For BST call validateTree

For vector call validateList

For Hash Table call validateTable

If choice is 3.

Search and print the course.

Get the value from the user and store it in userSearch.

For BST call printCourseTree

For Vector call printCourseList

For Hash Table call printCourseTable

If choice is 4.

Print courses in alphabetical order

For BST call printTree

For Vector call printList

For Hash Table call printTable.

If choice is 9

Exit the application

Output ‘Good bye’

End

Course Structure

CourseID

CourseName

PreCount

PreList

Binary Tree class

Node structure

course

right pointer

left pointer

Root

PrintTree

BinaryTree

Hash Table class

Bucket structure

coure

key

next pointer

Hash

PrintTable

List hash Table

SortList

Get vector to sort, lowest and highest index

If lowest index is higher or equal to highest index, return nothing

Call partition

Set low index equal to partition value

Call quicksort passing the vector, lowestindex and lowEndindex

Call quicksort passing the vector, lowEndIndex plus one, and highest index.

End

Partition

Get vector to partition, lowest and highest index.

Deterine vector element at midpoint between lowest and highest index

Set pivot equal to vector element

Loop until lowest index is equal to or greater than highest index

Loop through vector until an element larger than the pivot is found.

Overwrite lowest index with element

Loop through vector until an element smaller than the pivot is found

Overwrite highest index with element

Swap elements at new highest and lowest index

Overwrite lowest index by increasing it by 1

Overwrite highest index by decreasing it by 1

Return highest index

End

PrintList

Loop courseList

Output courseID and courseName to console

Loop 0 to preCount

For all courses in preList

output courseID to console

End

PrintTree

Create a pointer node named root

Set root to NULL

Check if Node is null

Call Node’s left pointer to find left most node

Output courseID and courseName to console

Loop 0 to preCount

for all courses in preList

Output courseID to console

Call Node’s right pointer to find right most Node

End

PrintTable

Create new Node pointer set to nodes beginning

Loop through list from the beginning

Output courseID to console

Output courseName to console

Loop 0 to preCount

for al courses on preList

call printCourse

End

Run Time Analysis

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Hash Table | Binary Tree |
| Loading Data | O(1) | O(1)-O(N) | O(log N) |
| Search | O(n) | O(1)-O(N) | O(log N)-O(N) |
| Sort/Print | O(N log N) | O(N) | O(N) |

Advantage Analysis

All three data structures have their benefits as well as their disadvantages that will affect how we access our data in the application. The three data structures we are implementing here are Vectors, Hash Tables, and Binary Search Trees.

The Vector data structure can access elements directly by their index with a constant time and can grow or shrink in size as needed. It can be memory intensive to make modifications to the middle of the list and to grow larger as they have to allocate memory to copy everything on the list and then copy it back.

A Hash Table data structure is very fast when it comes to doing common tasks like insertion, deletion, and search operations. Performance can be negatively affected when there is a large number of collisions. It also requires extra memory for the hash function and table.

Finally, a Binary Search Tree structure uses logarithmic time, 0(log N), which is much faster than linear time. They can also grow and shrink dynamically when data is added and removed and can be more memory efficient than other structures.

Recommendation

I would recommend the use of a hash table data structure because the data will be searched often, which this type of structure is very fast at. This also means that we need to try to keep the code operating at O(1) as often as possible so that it can conduct its operations optimaly.