Liam Daley

SNHU CS 300

4/9/2023

Project One

Contains Menu pseudocode, data type analysis and recommendation.

Function PrintMenu takes no arguments and returns no value

Print "Menu Options:"

Print "1. Load Data Structure"

Print "2. Print Course List"

Print "3. Print Course"

Print "4. Exit"

Declare integer variable userChoice, set to 0

While userChoice is not equal to 4

Prompt user for input

Set userChoice to input

If userChoice equals 1

LoadDataStructure()

Else If userChoice equals 2

PrintCourseList()

Else If userChoice equals 3

Prompt user for course ID

PrintCourse(courseID)

Else If userChoice equals 4

ExitProgram()

Else

Print "Invalid choice. Please enter a valid choice."

End if

End while

End function

// We learned in a previous assignment that Quicksort is a fast way to sort a vector. After this function sorts, all that is left is to for-loop through each element of the vector and print course data.

Function SortVector takes Vector of Courses, integer Begin, integer End as arguments, returns none

If begin >= end

Return

End if

Integer middle = 0

// Partition

Integer low = begin, Integer high = end

Integer midpoint = low + (high – low) / 2

string pivot = Courses[midpoint].courseId

while low <= end

while Courses[low].courseId < pivot:

low += 1;

end while

while Courses[high].couresId > pivot:

high -= 1;

end while

if low <= high:

swap values of Courses[low] and Courses[high])

low += 1, high -= 1;

end if

end while

swap values of Courses[low] and Courses[End])

SortVector(courses, start, low-1)

SortVector(courses, low+1, end)

End function

// Because we know the number of courses, we can have a perfect hash table. Therefore we need only traverse the keys in ascending order to access the data alphanumerically.

Function SortHashTable takes Table as argument, returns none

For integers 0 to table.size, incrementing by 1 each time

If Table[integer] is not a null pointer

Print course data for Table[integer]

End if

End for

End function

// Because binary search trees are already sorted, we need only do an in-order traversal to print the tree sorted.

Function SortBSTree takes Node as argument, returns none

If node is not a null pointer

SortBSTree(Node.leftNode)

Print node details

SortBSTree(Node.rightNode)

End if

End function

Analysis

After analyzing the advisor's requirements of printing a list of Computer Science courses in alphanumeric order and printing out the title and prerequisites of any given course, we are presented with three data structures variably suitable for these tasks: a vector, a hash table, and a binary search tree. Following is an analysis of each in the context of this problem.

A vector is a dynamic array that allows efficient element access and fast iteration. However, it does not maintain any inherent order, which means that searching and sorting operations can be costly. In the case of printing a list of courses in alphanumeric order, the vector would require extra steps to sort the data before outputting it. Additionally, retrieving a course's title and prerequisites based on its course number would have a time complexity of O(n), which could be inefficient for larger data sets.

A hash table is a data structure that, when properly designed, has constant-time complexity for insertion, deletion, and search operations. This makes it an ideal choice for the second requirement, random-access retrieval of a course's title and prerequisites based on its course number. However, a hash table may require additional steps to retrieve the data in sorted order, making it less suitable for the first requirement of printing a list of courses in alphanumeric order.

A binary search tree is a data structure that maintains a sorted order and provides efficient search, insertion, and deletion operations. This makes it an ideal choice for the first requirement of printing a list of courses in alphanumeric order. However, retrieving a course's title and prerequisites based on its course number would have a time complexity of O(log n), which could be less efficient than a hash table for this particular requirement.

Based on this analysis, we can easily conclude that a vector is not the proper solution. In the choice between hash tables and a binary search tree, my recommendation given the relatively large data set and frequent random lookups for course requirements is a hash table. This data structure has several advantages such as constant-time retrieval, making it ideal for situations where frequent lookups are required, which is one of the key requirements of the project. While a binary search tree was considered, the option was discarded due to it’s less efficient random lookup. Even though a BST is more efficient for in-order lookup, the random operation will be more frequent in this use case and as such is more important to optimize for.