Michael Donaldson

10/13/2024

Menu and Print Pseudocode

Menu Design

Display Menu:

1. Load course data

2. Print all courses in alphanumeric order

3. Print a course's title and prerequisites

9. Exit

Prompt user for input

If input == 1:

Load course data into the chosen data structure

If input == 2:

Sort and print courses in alphanumeric order

If input == 3:

Prompt for course number

Search for course and print its title and prerequisites

If input == 9:

Exit program

Else:

Print "Invalid input. Try again."

Print list in Alphanumeric Order

Vector

Function PrintCourses\_Alphanumeric\_Vector(Vector courses):

Sort the vector by course number (Assume course numbers are strings)

For each course in the sorted vector:

Print course number, course title, and prerequisites

Function SortVectorByCourseNumber(Vector courses):

Use a sorting algorithm based on alphanumeric comparison of course numbers

Return sorted vector

Hash

Function PrintCourses\_Alphanumeric\_HashTable(HashTable courses):

Convert the hash table values (courses) to a list

Sort the list by course number (Assume course numbers are strings)

For each course in the sorted list:

Print course number, course title, and prerequisites

Function ConvertHashTableToList(HashTable courses):

Create an empty list

For each course in the hash table:

Add the course to the list

Return the list

Tree

Function PrintCourses\_Alphanumeric\_Tree(Tree courses):

Perform an in-order traversal of the tree

For each course encountered during traversal:

Print course number, course title, and prerequisites

Function InOrderTraversal(TreeNode node):

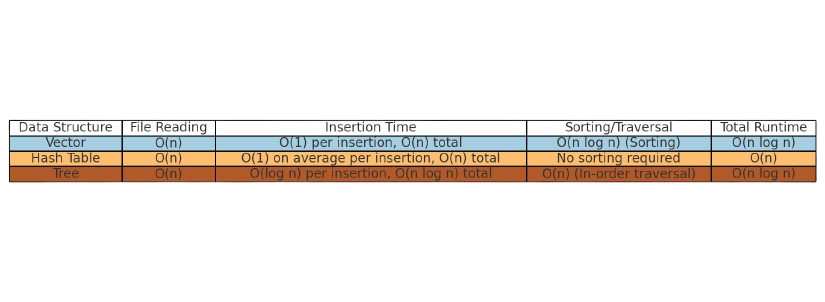
If node is not null:

InOrderTraversal(node.left)

Print course number, course title, and prerequisites

InOrderTraversal(node.right)

Runtime Analysis



**Advantages and Disadvantages**

Vector

One of the main benefits of using a vector is its simplicity and ease of use. Vectors store data in a continuous block of memory, which allows you to access any element quickly with O(1) time complexity. This makes vectors ideal when you need to quickly access courses by their position in the list. Vectors are also easy to sort with standard sorting algorithms like quicksort which runs in O(n log n) time. This works well when you need to organize the courses in alphanumeric order for display.

However, inserting or removing elements in the middle of the vector is slow because it requires shifting the other elements, taking O(n) time. Searching for a specific course also requires scanning the entire vector, which can be inefficient for large lists. Since vectors don’t maintain order on their own, you need to sort them separately to print the courses in alphanumeric order. This extra sorting step can slow things down when working with larger datasets.

Hash Table

A major advantage of using a hash table is its speed. Hash tables allow you to insert and find courses quickly, with an average time complexity of O(1). This makes them great for storing and retrieving data when speed is important. Hash tables don’t need to be sorted when inserting data, so you can quickly add new courses without worrying about the order.

The downside of hash tables is that they don’t keep data in order. If you need to print the courses in alphanumeric order, you will first need to convert the hash table to a list and then sort it, which takes O(n log n) time. Additionally, hash tables use extra memory to store data, which can increase their space usage compared to other data structures like vectors.

Tree

One of the biggest advantages of a tree, especially a balanced tree, is that it keeps the data sorted automatically. As courses are added, they are organized in order based on their course numbers. This means you don’t need to spend extra time sorting the data when it comes time to print the courses alphanumerically. A balanced tree also provides efficient insertion, deletion, and lookup times, each taking O(log n). This makes trees a good choice when you need to maintain sorted data and still have relatively fast access.

However, trees are more complicated to implement compared to vectors or hash tables. Balancing the tree can be tricky. Insertions and lookups, while relatively fast, are slower than the average time for a hash table. Trees also require additional memory for pointers to the left and right child nodes, which increases space usage.

Recommendation

After analyzing the data structures, I recommend using the tree for this program. The tree provides the best balance between keeping the courses sorted and ensuring efficient insertions and lookups.

From the Big O analysis, we know that a tree offers O(log n) time complexity for insertions, deletions, and lookups. This is slower than a hash table’s average O(1) time, but the tree’s major advantage is that it maintains the data in sorted order as elements are added. This eliminates the need for additional sorting when printing the courses in alphanumeric order, which would be required with both the vector and hash table. Additionally, an in-order traversal of the tree allows us to print the courses in O(n) time, making the overall efficiency for printing sorted data O(n log n).