Wurzburg Brumaire

Norman Lippincott

CS-300-11613

08 August 2024

Project One

MainMenu:

Option = Int

While True:

Print "1: Load course data"

Print "2: Print all courses”

Print "3: Print course details"

Print "9: Exit"

Print "Enter your choice:"

choice = GetInput

If 1 Then

courseData = LoadCourseData()

Else If choice = 2 Then

If courseData is not empty Then

PrintCoursesInOrder()

Else

Print "Error:"

End If

Else If

If courseData is not empty Then

Print "Enter course number:"

courseNumber = GetInput()

PrintCourseDetails()

Else

Print "Error"

End If

Else If 9 Then

ExitProgram()

Else

Print "Invalid choice"

End

Print Courses:

For each course:

Print "Course Number:"

Print "Course Title:”

End

Sort Courses

If coursesDataStructure is Vector Then

Sort vector by courseNumber

Return sorted vector

Else If coursesDataStructure is HashTable Then

Convert HashTable to List

Sort list by courseNumber

Return sorted list

End

Using a vector, each course is read line by line, with each iteration being O(1). The advantage of using a vector is that it is simple to implement and provides straightforward access to elements through their index. However, the runtime can increase significantly with larger data sizes because accessing and modifying elements, especially in cases where resizing the vector is necessary, can be costly. Additionally, modifying elements within a vector might require shifting other elements, which can be inefficient for large datasets.

A hash table, like a vector, allows for O(1) average time complexity for operations such as insertion, deletion, and lookup. However, hash tables can have memory overhead due to the need to maintain an underlying array and handle potential collisions through techniques like chaining or open addressing. The advantage of a hash table is its speed for these operations. The downside is the additional memory usage and the fact that the worst-case time complexity can degrade to O(n) if many collisions occur.

A balanced tree, such as a tree offers consistent performance with a time complexity of O(log n) for insertions, deletions, and lookups. While a balanced tree is more complex to implement and maintain compared to a vector or hash table, its efficiency in handling large datasets makes it a strong choice. The key to maximizing efficiency with a balanced tree is ensuring that the tree remains balanced.

Given this information, I recommend using a balanced tree for your data structure. It offers a good trade-off between performance and complexity, especially for larger datasets. However, you must ensure that the tree remains balanced to maintain optimal efficiency.