Katelynn Thompson

CS-300

James Webb

4/14/24

Project One

**Evaluation:**

After analyzing three data structures, namely vector, hash table, and tree, it was found that each has advantages and disadvantages in terms of runtime complexity and memory usage.

The vector is efficient in sequential data access and has a direct implementation, which enables it to have a runtime complexity of O (n) when reading files and creating course objects, where n represents the number of courses. However, due to its linear memory usage, it may not be optimal for large datasets, and it may require more efficiency for searching or inserting elements, especially if the dataset undergoes frequent modifications.

On the other hand, the hash table provides efficient insertion, deletion, and lookup operations on average with a runtime complexity of O(1). However, its worst-case performance can degrade if hash collisions occur frequently, impacting overall efficiency. The hash table also has a runtime complexity of O(n) for reading files and creating course objects.

Lastly, the tree offers efficient searching, insertion, and deletion operations even in worst-case scenarios with a runtime complexity of O(log n) for each operation. However, it tends to have higher memory overhead due to the need for maintaining balance, and it may be more complex to implement compared to the hash table. For reading files and creating course objects, the runtime complexity of the tree is O(n log n).

Considering ABCU's advising program's requirements, which involve efficiently storing and retrieving course information, the hash table is the most suitable choice. It strikes a balance between efficiency and simplicity, providing fast lookup operations and relatively low memory usage, making it well-suited for managing course data.

**Pseudocode for Menu:**

Function displayMenu():

Print "Menu:"

Print "1. Load Data Structure"

Print "2. Print Course List"

Print "3. Print Course Information"

Print "4. Exit"

Function main():

InitializeDataStructures()

// Loop until user chooses to exit

While true:

displayMenu() // Display the menu options

choice = getUserChoice() // Get user's choice

HandleUserChoice(choice)

Function HandleUserChoice(choice):

Switch choice:

Case 1:

LoadDataStructure()

Case 2:

PrintCourseList()

Case 3:

PrintCourseInformation()

Case 4:

ExitProgram()

Default:

Print "Error: Invalid choice. Please choose a valid option."

Function LoadDataStructure():

fileName = getFileNameFromUser() // Get file name from user

loadDataStructure(fileName) // Load data into the chosen data structure

Print "Data loaded successfully."

Function PrintCourseList():

If dataStructure is not loaded:

Print "Error: Data structure not loaded. Please load data first."

Else:

sortedCourses = SortCoursesAlphanumerically(dataStructure)

For each course in sortedCourses:

PrintCourseInfo(course)

Function PrintCourseInformation():

If dataStructure is not loaded:

Print "Error: Data structure not loaded. Please load data first."

Else:

courseNumber = getCourseNumberFromUser() // Get course number from user

course = findCourseByNumber(dataStructure, courseNumber)

If course is null:

Print "Error: Course not found."

Else:

PrintCourseInfo(course)

Function ExitProgram():

Print "Exiting program."

Exit

**Pseudocode for alphabetical order course list:**

Function PrintCourseListAlphanumeric(dataStructure):

If dataStructure is vector:

SortAndPrintVectorAlphanumeric(dataStructure)

Else If dataStructure is hash table:

SortAndPrintHashTableAlphanumeric(dataStructure)

Else If dataStructure is binary search tree:

InOrderTraversalAlphanumeric(dataStructure.root)

Function SortAndPrintVectorAlphanumeric(courseVector):

courseVector.sort(key=lambda course: course.courseNumber)

For each course in courseVector:

PrintCourseInfo(course)

Function SortAndPrintHashTableAlphanumeric(hashTable):

courses = SortCoursesAlphanumerically(GetAllCoursesFromHashTable(hashTable))

For each course in courses:

PrintCourseInfo(course)

Function GetAllCoursesFromHashTable(hashTable):

courses = []

For each key, value in hashTable:

courses.append(value)

return courses

Function SortCoursesAlphanumerically(courses):

return sorted(courses, key=lambda course: course.courseNumber)

Function InOrderTraversalAlphanumeric(node):

If node is not null:

InOrderTraversalAlphanumeric(node.left)

PrintCourseInfo(node.course)

InOrderTraversalAlphanumeric(node.right)

Function PrintCourseInfo(course):

print "Course Number:", course.courseNumber

print "Course Title:", course.courseTitle

If length(course.prerequisites) > 0:

print "Prerequisites:"

For each prerequisite in course.prerequisites:

print "-", prerequisite

Else:

print "No prerequisites"