Project 1 Justin Fanesi

CS 300

In the development of ABCU's advising program, the choice of an appropriate data structure is pivotal to the program's efficiency and responsiveness. I will talk about three primary data structures vector, hash table, and tree and evaluate their runtime complexities, memory considerations, and advantages and disadvantages.

Runtime and Memory Analysis:

Vector Data Structure:

File Reading and Course Object Creation:

* Cost per line: 1
* Number of lines executed: n (number of courses)
* Big O: O(n)
* Memory: O(n) for storing course objects in a vector.

Hash Table Data Structure:

File Reading and Course Object Creation:

* Cost per line: 1
* Number of lines executed: n (number of courses)
* Big O: O(n)
* Memory: O(n) for storing course objects in a hash table.

Tree Data Structure:

File Reading and Course Object Creation:

* Cost per line: 1
* Number of lines executed: n (number of courses)
* Big O: O(n log n) - assuming a balanced tree in a general case
* Memory: O(n) for storing course objects in a tree.

Advantages and Disadvantages:

Vector Data Structure:

The vector data structure presents a straightforward and easy-to-implement solution, ensuring the maintenance of order. However, its disadvantage is its comparatively slower lookup times, especially when handling large datasets.

Hash Table Data Structure:

The hash table offers swift lookup times and is highly efficient for managing extensive datasets. Nevertheless, it lacks inherent order and might encounter collisions, potentially leading to performance issues.

Tree Data Structure:

The tree structure demonstrates efficiency for certain operations like range queries. However, it comes with a higher memory overhead and involves complexity in maintaining a balanced tree.

Each data structure possesses unique strengths and weaknesses, requiring careful consideration based on the specific requirements of ABCU's advising program.

Recommendation:

Considering the advisor's requirements and the comprehensive analysis, the hash table data structure is the most suitable choice for ABCU's advising program. Its efficient lookup times, constant time complexity for adding and retrieving elements, and ability to handle large datasets make it ideal for the program's requirements. While the vector and tree structures are viable options, the hash table's performance advantages outweigh the drawbacks and is a better suitor for the role of ABCU’s advising system.

In conclusion, the hash table data structure aligns with the program's need for quick access to course information. Its advantages in efficiency and scalability make it the recommended choice, helping to ensure an effective and responsive advising system for ABCU.

Vector Pseudocode:

Function to Open File, Read Data, Parse Lines, and Check Formatting Errors:

function loadCoursesFromFile(filename):  
 courses = createEmptyVector() // Create an empty vector to store Course objects  
  
 try:  
 file = open(filename, "r") // Open the file in read mode  
 for line in file.readlines():  
 courseData = parseCourseData(line) // Parse the course data from the line  
 if validateCourseData(courseData): // Validate the course data  
 course = createCourseObject(courseData) // Create a Course object  
 courses.append(course) // Add the Course object to the vector  
 else:  
 print("Error: Invalid course data on line:", line)  
 file.close()  
 except FileNotFoundError:  
 print("Error: File not found")  
  
 return courses

Function to Print Course Information and Prerequisites:

function printCourseInformation(courses, courseNumber):  
 found = False  
  
 for course in courses:  
 if course.courseNumber == courseNumber:  
 found = True  
 print("Course Information:")  
 print("Course Number:", course.courseNumber)  
 print("Course Name:", course.courseName)  
 print("Prerequisites:")  
 for prerequisite in course.prerequisites:  
 print("- ", prerequisite)  
  
 if not found:  
 print("Error: Course not found")

Function to Print Sorted List of Computer Science Courses:

function printSortedComputerScienceCourses(courses):  
 computerScienceCourses = filterCoursesByDepartment(courses, "Computer Science")  
 sortedCourses = sortCoursesAlphanumerically(computerScienceCourses)  
  
 print("Alphanumerically Sorted Computer Science Courses:")  
 for course in sortedCourses:  
 printCourseInformation(courses, course.courseNumber)

Tree Pseudocode:

Function to Open File, Read Data, Parse Lines, and Check Formatting Errors:

function loadCoursesFromFile(filename):  
 coursesTree = createEmptyTree() // Create an empty tree to store Course objects  
  
 try:  
 file = open(filename, "r") // Open the file in read mode  
 for line in file.readlines():  
 courseData = parseCourseData(line) // Parse the course data from the line  
 if validateCourseData(courseData): // Validate the course data  
 course = createCourseObject(courseData) // Create a Course object  
 addCourseToTree(coursesTree, course) // Add the Course object to the tree  
 else:  
 print("Error: Invalid course data on line:", line)  
 file.close()  
 except FileNotFoundError:  
 print("Error: File not found")  
  
 return coursesTree

Function to Print Course Information and Prerequisites:

function printCourseInformation(coursesTree, courseNumber):  
 course = searchCourseInTree(coursesTree, courseNumber)  
 if course is not None:  
 print("Course Information:")  
 print("Course Number:", course.courseNumber)  
 print("Course Name:", course.courseName)  
 print("Prerequisites:")  
 for prerequisite in course.prerequisites:  
 print("- ", prerequisite)  
 else:  
 print("Error: Course not found")

Function to Print Sorted List of Computer Science Courses:

function printSortedComputerScienceCourses(coursesTree):  
 computerScienceCourses = filterCoursesByDepartment(coursesTree, "Computer Science")  
 sortedCourses = sortCoursesAlphanumerically(computerScienceCourses)  
  
 print("Alphanumerically Sorted Computer Science Courses:")  
 for course in sortedCourses:  
 printCourseInformation(coursesTree, course.courseNumber)