6-2 Project One

Gavin Bish

SNHU

CS 300: Analysis and Design

October 11th, 2024

**Main Menu**

// Main menu function

Function DisplayMenu()

While true

Print "1. Load Courses"

Print "2. Print Sorted Courses"

Print "3. Print Course Details"

Print "9. Exit"

Input userChoice

If userChoice equals 1

Input fileName

courseDataList = ReadFile(fileName)

// Choose appropriate data structure to load courses

// e.g., vector = LoadCoursesIntoVector(courseDataList)

Else If userChoice equals 2

// Call appropriate print function based on data structure

// e.g., PrintSortedCoursesVector(vector)

Else If userChoice equals 3

Input courseID

// Call appropriate search function based on data structure

// e.g., PrintCourseDetailsVector(vector, courseID)

Else If userChoice equals 9

Break

Else

Print "Invalid option. Try again."

End While

**Vector Sorting**

// Vector-specific pseudocode

// Function to load courses into a vector

Function LoadCoursesIntoVector(courseDataList)

vector = CreateCourseObjects(courseDataList)

Return vector

// Function to sort and print the courses stored in a vector

Function PrintSortedCoursesVector(vector)

Sort vector by courseID

For each Course in vector

Print Course.courseID and Course.courseTitle

End For

// Function to find and print details of a specific course in a vector

Function PrintCourseDetailsVector(vector, courseID)

For each Course in vector

If Course.courseID equals courseID

Print Course.courseTitle and prerequisites

Return

End For

Print "Course not found”

|  |  |  |  |
| --- | --- | --- | --- |
| **Line/Operation** | **Description** | **Cost Per Line** | **Big O Notation** |
| ReadFile(fileName) | Read file line by line | O(1) per line | O(n) |
| Add Course to vector | Append to vector | O(1) | O(n) |
| Sort vector by courseID | Sort vector using a comparison-based algorithm | O(n log n) | O(n log n) |
| For each Course in vector | Iterating through the vector | O(1) per course | O(n) |
| Print Course.courseID and Course.courseTitle | Printing each course | O(1) per course | O(n) |
| Find Course in vector (linear search) | Iterating through vector to find a course | O(1) per comparison | O(n) |
| Print Course.courseTitle and prerequisites (if found) | Printing details | O(1) | O(1) if found |

**Hash Tables**

// Hash Table-specific pseudocode

// Function to load courses into a hash table

Function LoadCoursesIntoHashTable(courseDataList)

Initialize empty hashTable

For each courseData in courseDataList

Create a Course object

Insert Course into hashTable with Course.courseID as the key

End For

Return hashTable

// Function to print all courses in the hash table (unordered)

Function PrintAllCoursesHashTable(hashTable)

For each key in hashTable

Print key and hashTable[key].courseTitle

End For

// Function to find and print details of a specific course in a hash table

Function PrintCourseDetailsHashTable(hashTable, courseID)

If courseID exists in hashTable

Print hashTable[courseID].courseTitle and prerequisites

Else

Print "Course not found"

|  |  |  |  |
| --- | --- | --- | --- |
| **Line/Operation** | **Description** | **Cost Per Line** | **Big O Notation** |
| ReadFile(fileName) | Read file line by line | O(1) per line | O(n) |
| CreateCourseObjects(courseDataList) | Create a list of Course objects | O(1) per object | O(n) |
| Insert Course into hashTable | Insert using hash function | O(1) (average) | O(n) for all insertions |
| For each key in hashTable | Iterating through the hash table | O(1) per course | O(n) |
| Print key and hashTable[key].courseTitle | Printing each course | O(1) per course | O(n) |
| Check if courseID exists in hashTable | Search using hash function | O(1) (average) | O(1) |
| Print course details (if found) | Printing details | O(1) | O(1) if found |

**Binary Search Tree**

// Binary Search Tree-specific pseudocode

// Function to insert a Course into the BST

Function InsertNode(root, course)

If root is null

Set root to new Node with course as data

Else If course.courseID < root.data.courseID

InsertNode(root.left, course)

Else

InsertNode(root.right, course)

// Function to load courses into a BST

Function LoadCoursesIntoBST(courseDataList)

Initialize root as null

For each courseData in courseDataList

Create a Course object

InsertNode(root, Course)

End For

Return root

// Function to perform in-order traversal and print sorted courses

Function InOrderTraversalPrint(root)

If root is not null

InOrderTraversalPrint(root.left)

Print root.data.courseID and root.data.courseTitle

InOrderTraversalPrint(root.right)

// Function to find and print details of a specific course in the BST

Function SearchCourseBST(root, courseID)

If root is null

Print "Course not found"

Else If courseID equals root.data.courseID

Print root.data.courseTitle and prerequisites

Else If courseID < root.data.courseID

SearchCourseBST(root.left, courseID)

Else

SearchCourseBST(root.right, courseID)

|  |  |  |  |
| --- | --- | --- | --- |
| **Line/Operation** | **Description** | **Cost Per Line** | **Big O Notation** |
| ReadFile(fileName) | Read file line by line | O(1) per line | O(n) |
| CreateCourseObjects(courseDataList) | Create a list of Course objects | O(1) per object | O(n) |
| InsertNode(root, course) | Insert into BST (logarithmic in balanced tree) | O(log n) (average) | O(n log n) for all insertions |
| InOrderTraversalPrint(root) | Traverse and print in sorted order | O(1) per node | O(n) |
| SearchCourseBST(root, courseID) | Search BST for a course | O(log n) (average) | O(log n) |
| Print course details (if found) | Printing details | O(1) | O(1) if found |

**Advantages and Disadvantages of Each Data Structure**

* Vector
  + Advantages: Simple to implement, allows for easy sorting.
  + Disadvantages: Inefficient for searching and inserting in the middle.
* Hash Table
  + Advantages: Fast lookup and insertion.
  + Disadvantages: Poor ordering, requires handling collisions.
* Binary Search Tree
  + Advantages: Efficiently ordered, good for dynamic sorting.
  + Disadvantages: Performance degrades to O(n) if unbalanced.

**Recommended Data Structure**

* Recommend the best data structure based on the analysis.
* Justify the recommendation considering the given requirements, Big O analysis, and trade-offs.