Sergio Irianda

CS300

CS 300 Project One

Function LoadFile(fileName):

Open file with name fileName

If file cannot be opened:

Print "Error: File not found."

Return

For each line in file:

Split line by commas into fields

If fields.length is not as expected:

Print "Formatting error on line: " + line

Continue

Create Course object using fields

Add Course object to data structure

Close file

Function CreateCourse(courseNumber, title, prerequisites):

Create Course object

Set Course.courseNumber = courseNumber

Set Course.title = title

Set Course.prerequisites = prerequisites

Return Course

Function PrintCourseDetails(courseNumber):

Search for Course in data structure by courseNumber

If Course not found:

Print "Course not found."

Return

Print "Course Title: " + Course.title

Print "Prerequisites: " + Join(Course.prerequisites, ", ")

Function MainMenu():

Do:

Print "Menu Options:"

Print "1. Load File Data"

Print "2. Print All Courses in Alphanumeric Order"

Print "3. Print Course Details"

Print "9. Exit"

Get userInput

If userInput == 1:

Call LoadFile(fileName)

Else If userInput == 2:

Call PrintAllCourses()

Else If userInput == 3:

Get courseNumber from user

Call PrintCourseDetails(courseNumber)

Else If userInput == 9:

Print "Exiting program."

Exit

Else:

Print "Invalid option."

While True

**VECTOR**

Function PrintAllCourses():

Sort vector of courses by courseNumber

For each Course in vector:

Print Course.courseNumber + ": " + Course.title

**HASH TABLE**

Function PrintAllCourses():

Create a temporary list

For each Course in hash table:

Add Course to list

Sort list by courseNumber

For each Course in sorted list:

Print Course.courseNumber + ": " + Course.title

**TREE**

Function PrintAllCourses():

Perform in-order traversal of the tree

For each Course visited:

Print Course.courseNumber + ": " + Course.title

**Runtime Analysis**

**1. Vector**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open file | 1 | 1 | 1 |
| Check file open success | 1 | 1 | 1 |
| For each line in file | 1 | nnn | nnn |
| Split line into tokens | 1 | nnn | nnn |
| Check line format | 1 | nnn | nnn |
| Create course object | 1 | nnn | nnn |
| Insert course into vector | 1 | nnn | nnn |
| Close file | 1 | 1 | 1 |
| Total Cost |  |  | 5n+35n + 35n+3 |
| Runtime |  |  | O(n)O(n)O(n) |

1. **Hash Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open file | 1 | 1 | 1 |
| Check file open success | 1 | 1 | 1 |
| For each line in file | 1 | nnn | nnn |
| Split line into tokens | 1 | nnn | nnn |
| Check line format | 1 | nnn | nnn |
| Create course object | 1 | nnn | nnn |
| Insert course into hash table | O(1)O(1)O(1) | nnn | nnn |
| Close file | 1 | 1 | 1 |
| Total Cost |  |  | 5n+35n + 35n+3 |
| Runtime |  |  | O(n)O(n)O(n) |

1. **Binary Search Tree**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open file | 1 | 1 | 1 |
| Check file open success | 1 | 1 | 1 |
| For each line in file | 1 | nnn | nnn |
| Split line into tokens | 1 | nnn | nnn |
| Check line format | 1 | nnn | nnn |
| Create course object | 1 | nnn | nnn |
| Insert course into binary tree | O(log⁡(n))O(\log(n))O(log(n)) | nnn | nlog⁡(n)n \log(n)nlog(n) |
| Close file | 1 | 1 | 1 |
| Total Cost |  |  | nlog⁡(n)+4n+3n \log(n) + 4n + 3nlog(n)+4n+3 |
| Runtime |  |  | O(nlog⁡(n))O(n \log(n))O(nlog(n)) |

**Recommendations**

Each data structure has its strengths and weaknesses for this program. A **vector** is simple and quick for reading the file and adding courses. Sorting the courses is straightforward and fast, which would be useful for printing the full catalog. However, searching for a specific course can be slow because the program must check each course one by one until it finds a match. This makes it less ideal if searching happens very often.

A **hash table** is also great for searching because it can find a course quickly. However, setting it up takes more time and memory, and it lacks organization. If you need a sorted list of courses, you’d have to extract the data, sort it separately, and then print it. This methods adds extra steps.A **binary search tree** makes it easy to keep the courses sorted and is efficient at searching. However, it’s more complex to set up and maintain. It also uses more memory compared to the others.

I recommend using a **vector** for this project. The simplicity of sorting makes it the best option for printing the course catalog. It’s straightforward to work with and fits well with the program’s needs.