Elliot Putnam

CS300 DSA: Analysis and Design

Project One

2/20/2024

1. **Vector**

Function parseFileAndCreateCourses(filename):

courses = Empty Vector

Try:

Open file with 'filename'

For each line in the file:

If line is well-formatted:

courseData = Parse line

course = CreateCourseObject(courseData)

Add course to courses vector

Else:

Print "Error: Invalid formatting in line. Skipping."

Close the file

Return courses

Catch FileOpenException:

Print "Error: Unable to open the file."

Function ParseLine(line):

tokens = Split line by ","

courseCode = tokens[0]

courseTitle = tokens[1]

prerequisites = tokens[2:] // Remaining tokens as prerequisites

Return (courseCode, courseTitle, prerequisites)

Function CreateCourseObject(courseData):

Create a new course object

Set course attributes using courseData

Return the course object

1. **Hash Table**

Function parseFileAndCreateCourses(filename):

courses = Empty Hash Table

Try:

Open file with 'filename'

For each line in the file:

If line is well-formatted:

courseData = Parse line

course = CreateCourseObject(courseData)

Add course to courses hash table using course number as the key

Else:

Print "Error: Invalid formatting in line. Skipping."

Close the file

Return courses

Catch FileOpenException:

Print "Error: Unable to open the file."

Function ParseLine(line):

tokens = Split line by ","

courseCode = tokens[0]

courseTitle = tokens[1]

prerequisites = tokens[2:] // Remaining tokens as prerequisites

Return (courseCode, courseTitle, prerequisites)

Function CreateCourseObject(courseData):

Create a new course object

Set course attributes using courseData

Return the course object

1. **Tree**

Class TreeNode:

Data: course

Left: TreeNode

Right: TreeNode

Function parseFileAndCreateCourses(filename):

courses = Null

Try:

Open file with 'filename'

For each line in the file:

If line is well-formatted:

courseData = Parse line

course = CreateCourseObject(courseData)

courses = InsertIntoTree(courses, course)

Else:

Print "Error: Invalid formatting in line. Skipping."

Close the file

Return courses

Catch FileOpenException:

Print "Error: Unable to open the file."

Function ParseLine(line):

tokens = Split line by ","

courseCode = tokens[0]

courseTitle = tokens[1]

prerequisites = tokens[2:] // Remaining tokens as prerequisites

Return (courseCode, courseTitle, prerequisites)

Function InsertIntoTree(root, course):

If root is Null:

Create a new TreeNode with course as data

Return the new node

Else If course number is less than root's course number:

Set root's left child to InsertIntoTree(root's left child, course)

Else:

Set root's right child to InsertIntoTree(root's right child, course)

Return root

Function CreateCourseObject(courseData):

Create a new course object

Set course attributes using courseData

Return the course object

**MENU**

Function MainMenu():

dataStructure = Null

While true:

Print "1. Load Data Structure"

Print "2. Print Course List"

Print "3. Print Course"

Print "4. Exit"

choice = GetUserInput()

Switch choice:

Case 1:

dataStructure = LoadDataStructure()

Case 2:

If dataStructure is not Null:

PrintAlphanumericCourseList(dataStructure)

Else:

Print "Error: Data structure not loaded. Please choose option 1 first."

Case 3:

If dataStructure is not Null:

courseCode = GetUserInput("Enter course code:")

PrintCourseInformation(dataStructure, courseCode)

Else:

Print "Error: Data structure not loaded. Please choose option 1 first."

Case 4:

ExitProgram()

Default:

Print "Invalid choice. Please try again."

Function LoadDataStructure():

Print "Enter the filename:"

filename = GetUserInput()

Switch chosenDataStructure:

Case "1":

Return parseFileAndCreateCoursesVector(filename)

Case "2":

Return parseFileAndCreateCoursesHashTable(filename)

Case "3":

Return parseFileAndCreateCoursesTree(filename)

Default:

Print "Invalid data structure choice. Please try again."

Function parseFileAndCreateCoursesVector(filename):

// Pseudocode for loading data into a vector, like the previous pseudocode

Return courses

Function parseFileAndCreateCoursesHashTable(filename):

// Pseudocode for loading data into a hash table, like the previous pseudocode

Return courses

Function parseFileAndCreateCoursesTree(filename):

// Pseudocode for loading data into a tree, like the previous pseudocode

Return courses

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For all courses | 1 | n | n |
| If course is same as number | 1 | n | n |
| print out the course information | 2 | 1 | 1 |
| for each prerequisite of the course | 1 | n | n |
| print the prerequisite course information | 2 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | 1(n) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For all courses | 2 | n | n |
| If course is same as number | 1 | n | n |
| print out the course information | 1 | 1 | 1 |
| for each prerequisite of the course | 2 | n | n |
| print the prerequisite course information | 4 | n | n |
| **Total Cost** | | | 9n + 1 |
| **Runtime** | | | O(n) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For all courses | 1 | n | n |
| If course is same as number | 1 | n | n |
| print out the course information | 2 | 1 | 1 |
| for each prerequisite of the course | 1 | n | n |
| print the prerequisite course information | 4 | n | n |
| **Total Cost** | | | 8n + 1 |
| **Runtime** | | | O(n) |

I would choose to use a vector structure due to its minimized total cost to operate. Due to the limited data set, it will be performant compared to hash but will be reduced if it grows larger. It will also be easy to implement.