1 Project 1

Dathan Pompa

CS 300 Analysis and Design

6/23/2024

Project 1

Step 1. Part 1. Binary Search Tree

Open file, reads data, parses each line and checks for errors

Function LoadCourses(filename):

Open file with name 'filename'

Initialize an empty list 'lines'

For each line in the file:

Strip any leading/trailing whitespace from the line

Append the line to 'lines'

Close the file

Return 'lines'

Function ValidateAndParseLines(lines):

Initialize an empty object 'courseObj'

For each line in 'lines':

Split the line by comma into 'tokens'

If the number of 'tokens' is less than 2:

Print error message "Error: Line does not contain at least two parameters." to output

Return False

courseNumber = tokens[0]

courseTitle = tokens[1]

prerequisites = tokens[2:]

courseObj[courseNumber] = (courseTitle, prerequisites)

//prerequisites

For each course in 'courseObj':

courseNumber, (courseTitle, prerequisites) = course

For each prerequisite in 'prerequisites':

If prerequisite is not in 'courseObj':

Print error message "Error: Prerequisite course <prerequisite> for course <courseNumber> does not exist." to output

Return False

Return 'courseObj'

Part 2.

Creating course objects and storing in data structure.

```
Function CreateCourseObjects(courseObj, bst):
    For each courseNumber in 'courseObj':
        courseTitle, prerequisites = courseObj[courseNumber]
        Create a new Course object 'course'
        Set course.courseNumber = courseNumber
        Set course.title = courseTitle
        Set course.prerequisites = prerequisites
        bst.Insert(course)
```

Part 3.

Print our course info and prerequisites

```
Function PrintCourseInformation(bst):
```

```
Function InOrderTraversal(node):
    If node is not None:
        InOrderTraversal(node.left)
        Print "Course Number: " + node.course.courseNumber to output
        Print "Course Title: " + node.course.title to output
        If node.course.prerequisites is not empty:
            Print "Prerequisites: " + ", ".join(node.course.prerequisites) to output
        Else:
            Print "Prerequisites: None" to output
        InOrderTraversal(node.right)
InOrderTraversal(bst.root)
```

(The Vector)

File Opener

Courses

Set course number as string

Set course name as string

Vector of type string to store prerequisites

5 Project 1

Make hashtable

Struct node

create course pointer with name course

create node pointer with next

create integer key

node with

set key with max units

set next to null ptr

node(course pointer and new course) with node method

set current course to newcourse

node course pointer and newcourse with key to node method with parameters newCourse

Readfile(file and hashtable

Initialize the fstream and filestream

Then the string to hold data

Then string stream and line stream for its contents

Then a token for each word

Open file with filestream

fill each line with linestream

create counter initialize to 1

make new course pointer with new course for each line in the file

get the token up to a comma ‘,’ and continue until end of file

if counter is 1 || 2

set newcourse and course number to the token and counter ++

else

if token is present in table

add token to prerequisites

if counter is less than 1

print format error within file

add the course to table

clear line stream

Object Creator

Add course with parameters course pointer with newCourse

Make key for the new course, hash the course number

Make node to get the node by its key

If the node is null

make new node with add course, new course, and its key

push the new course into table at key position

If nodes key is at max units

set the node key to key, node course to newcourse, next node to null

Else

while next node is not null

set node to next node

make a new node called addcourse with newcourse and its key

set the next node to addcourse

Data Search

Search by course number

Create empty course object

For each loop of object

if courses course number is equal to our created course object

return course

create a list node and set it to next node

while not nullptr

if listnode course number equals our course object

return course

the list node points to next node

At the end return created object

Print

Create empty queue and enqueue the table

While queue is not empty

dequeue a node from the queue

print node course key and name

For each loop of course object

enqueue next node

While next node is not empty continue

Else return.

(The Hash Table)

Part 1. Read and validate the file

Function readFile(filePath);

    Open the file at filePath in read mode as file

    Set courses = empty hash table

    AllCourses = emptyList;

    For each line in the file;

        Remove the leading and trailing space from the line

        If the line is not empty;

```
                tokens = split line with comma
                if length of tokens is less than 2;
                        print error
                        continue
        courseNum = tokens[0].strip()
        title = tokens[1].strip()
        prerequisites = []
        if length of tokens > 2;
                for i =0 from 2 to length of tokens –1:
                        prerequisite = tokens[i].strip()
                        prerequisites.append(prerequisite)
```

Part 2. creating course objects and storing them

```
    Defining course class
    Class Course;
Course(courseNum, title, prerequisites);
    This.courseNum = courseNum
    This.title = title
    This.prerequisites = prerequisites
```

```
newCourse = Course(courseNum, title, prerequisites)

courses[courseNum] = newCourse

add courseNum to allCourses

// Check for missing prerequisites

For course in courses:

For course in courses

for prerequisite in course.prerequisites:

if prerequisite is not in courses;

print error prerequisite in courseNum doesnt exist

remove prerequisite from course.prerequisites

Return courses;
```

Part 3. Printing course information and prerequisites

```
Function printCourses(courses);

For courseNum, course in courses

print course.courseNum

print course.title

if length of prerequisite is 0
```

print no prerequisite

else

print join(course.prerequisites)

Step 2.

The menu

Main menu Function

Function MainMenu():

Initialize bst as a new binary search tree

Initialize courseObj as an empty object

Declare filename as an empty string

While True:

Print "1: Load the file data into the data structure"

Print "2: Print an alphanumerically ordered list of all the courses in the Computer Science department"

Print "3: Print the course title and the prerequisites for any individual course"

Print "9: Exit the program"

Print "Enter your choice: "

Read choice from user input

```
If choice == 1:
        Print "Enter the filename: "
        Read filename from user input
        lines = LoadCourses(filename)
        If lines is not None:
            courseObj = ValidateAndParseLines(lines)
            If courseObj is not False:
                CreateCourseObjects(courseObj, bst)
                Print "Courses loaded successfully."
             Else:
                Print "Failed to validate and parse lines."
        Else:
            Print "Failed to load courses."
Else If choice == 2:
        If bst.root is not None:
            PrintCourseInformation(bst)
        Else:
            Print "Please load the course data first (Option 1)."
```

```
Else If choice == 3:
    If bst.root is not None:
        Print "Enter the course number: "
        Read courseNumber from user input node =
        bst.Search(courseNumber)
        If node is not None:
            Print "Course Number: " + node.course.courseNumber
            Print "Course Title: " + node.course.title
            If node.course.prerequisites is not empty:
                Print "Prerequisites: " + ",
            ".join(node.course.prerequisites)
            Else:
                Print "Prerequisites: None"
        Else:
            Print "Course not found."
    Else:
        Print "Please load the course data first (Option 1)."
Else If choice == 9:
    Print "Exiting the program."
    Break
```

```
        Else:
            Print "Invalid choice. Please try again." // Error handling
// sorting in alphabetical order
 Function PrintCourseInformation(bst):
    Function InOrderTraversal(node):
        If node is not None:
            InOrderTraversal(node.left)
            Print "Course Number: " + node.course.courseNumber
            Print "Course Title: " + node.course.title
            If node.course.prerequisites is not empty:
                Print "Prerequisites: " + ", ".join(node.course.prerequisites) Else:
                Print "Prerequisites: None"
            InOrderTraversal(node.right)
    InOrderTraversal(bst.root)
```

Evaluation

Operation	Vector	Hash table	Binary search tree
Loading courses	O(n)	O(n)	O(n)
Validate and parse lines	O(nm)	O(nm)	O(nm)

Create and insert course objects	O(n)	O(n)	O(n log n)

Loading courses is going to be the runtime O(n) for all three structures.

Validating and parsing lines: runtime is O(nm) with n as the number and m being the token.

Creating and inserting course objects: Vector and Hash table are O(n) because vector takes a constant time O(1) and Hash table takes an average constant time of O(1) also. Binary Search Tree is O(n log n) because it inserts each object into a bst which takes an average time of O(log n).

Vectors: It's easy to use with O(n) inserts time but cannot search efficiently.

Hash Table: This gives us an insert of O(1) and search as well but can collide which would give it O(n) worst case time.

BST: Gives O(log n) insert and search, it keeps a sorted order which would be the best choice for our ordered data traversal, this makes Binary Search Tree the best choice for our circumstances.