Brian Blackman

10/31/2024

6-2 Submit Project One

Southern New Hampshire University

**Vector Pseudocode**

//Read and store course information

function readCourseInformation(fileName):

open fileName as file

for each line in file:

if line is not in correct format:

print "Error: Incorrect file format"

continue

courseDetails = parseLine(line)

addCourseToVector(courseDetails)

close file

//Store course objects

function addCourseToVector(courseDetails):

course = createCourseObject(courseDetails)

courseVector.append(course)

function createCourseObject(courseDetails):

course = new Course()

course.id = courseDetails[0]

course.name = courseDetails[1]

course.prerequisites = courseDetails[2]

return course

//Search data and print data

function searchCourse(courseVector, searchTerm):

for each course in courseVector:

if course.id == searchTerm or course.name contains searchTerm:

printCourseDetails(course)

function printCourseDetails(course):

print "Course ID: " + course.id

print "Course Name: " + course.name

print "Prerequisites: " + course.prerequisites

FUNCTION PrintSortedCoursesVector()

IF vector is empty THEN

DISPLAY "Data not loaded. Please load the data first."

RETURN

END IF

// SORT vector by courseNumber alphanumerically

FOR each course in vector DO

DISPLAY courseNumber, name

END FOR

END FUNCTION

**Hash Table Pseudocode**

// Open the File

ATTEMPT to open file "courses.txt" for reading

IF file cannot be opened THEN

DISPLAY "Error: File cannot be opened"

EXIT program

END IF

// Initialize Data Structures

DECLARE hash table "courseTable" to store course information

// Read and Parse Each Line

WHILE there are more lines to read in the file DO

READ a line from the file

SPLIT the line into parts using comma as the separator

IF number of parts < 2 THEN

DISPLAY "Error: Invalid line format"

CONTINUE to the next iteration

END IF

// Validate Prerequisites

SET courseNumber = first part

SET courseTitle = second part

SET prerequisites = remaining parts

FOR each prerequisite in prerequisites DO

IF prerequisite is not found in courseTable THEN

DISPLAY "Error: Prerequisite course not found"

CONTINUE to the next iteration

END IF

END FOR

// Create Course Object

CREATE new Course object with courseNumber, courseTitle, and prerequisites

INSERT Course object into courseTable using courseNumber as key

END WHILE

// Close the File

CLOSE the file

// Print Course Information

FOR each course in courseTable DO

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

IF course has prerequisites THEN

PRINT "Prerequisites: " + JOIN(prerequisites, ", ")

ELSE

PRINT "Prerequisites: None"

END IF

END FOR

END FUNCTION

FUNCTION PrintSortedCoursesHashTable()

IF hashTable is empty THEN

DISPLAY "Data not loaded. Please load the data first."

RETURN

END IF

//CREATE empty list sortedCourses

FOR each key in hashTable DO

ADD hashTable[key] to sortedCourses

END FOR

//SORT sortedCourses by courseNumber alphanumerically

FOR each course in sortedCourses DO

DISPLAY courseNumber, name

END FOR

END FUNCTION

**Tree Pseudocode**

// Load Data from file

Method loadCourses(String filePath):

Open file at filePath

If file is not open:

Print "Error: Unable to open file"

Return

Initialize empty list courseList

Initialize empty dictionary courseDict

While not end of file:

Read line from file

Split line by comma into tokens

If number of tokens < 2:

Print "Error: Invalid line format"

Continue

Extract courseNumber and courseTitle from tokens

Initialize empty list prerequisites

For each token after courseTitle:

Add token to prerequisites

Create Course object with courseNumber, courseTitle, and prerequisites

Add Course object to courseList

Add courseNumber to courseDict with value Course object

Close file

Return courseList, courseDict

//Create and store object

Structure Course:

String courseNumber

String courseTitle

List prerequisites

Constructor(courseNumber, courseTitle, prerequisites):

this.courseNumber = courseNumber

this.courseTitle = courseTitle

this.prerequisites = prerequisites

Method insertCourse(BinarySearchTree bst, Course course):

bst.Insert(course)

Method validatePrerequisites(courseList, courseDict):

For each course in courseList:

For each prerequisite in course.prerequisites:

If prerequisite not in courseDict:

Print "Error: Prerequisite " + prerequisite + " for course " + course.courseNumber + " does not exist"

Return False

Return True

Method loadCoursesIntoBST(String filePath):

courseList, courseDict = loadCourses(filePath)

If not validatePrerequisites(courseList, courseDict):

Print "Error: Invalid prerequisites found"

Return

Initialize BinarySearchTree bst

For each course in courseList:

insertCourse(bst, course)

Return bst

//Print information

Method printCourseInfo(Course course):

Print "Course Number: " + course.courseNumber

Print "Course Title: " + course.courseTitle

If course.prerequisites is not empty:

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

Else:

Print "Prerequisites: None"

Method printAllCourses(BinarySearchTree bst):

Define inOrderTraversal(Node node):

If node is not null:

inOrderTraversal(node.left)

printCourseInfo(node.course)

inOrderTraversal(node.right)

Call inOrderTraversal(bst.root)

FUNCTION PrintSortedCoursesTree()

IF tree is empty THEN

DISPLAY "Data not loaded. Please load the data first."

RETURN

END IF

FUNCTION InOrderTraversal(node)

IF node is not NULL THEN

InOrderTraversal(node.left)

DISPLAY node.courseNumber, node.name

InOrderTraversal(node.right)

END IF

END FUNCTION

CALL InOrderTraversal(tree.root)

END FUNCTION

**Menu Pseudocode**

BEGIN

DISPLAY "Menu:"

DISPLAY "1. Load file data into data structure"

DISPLAY "2. Print alphanumerically ordered list of all Computer Science courses"

DISPLAY "3. Print course title and prerequisites for a specific course"

DISPLAY "9. Exit"

WHILE True DO

DISPLAY "Enter your choice:"

READ choice

IF choice == 1 THEN

CALL LoadData()

ELSE IF choice == 2 THEN

CALL PrintSortedCourses()

ELSE IF choice == 3 THEN

CALL PrintCourseDetails()

ELSE IF choice == 9 THEN

EXIT

ELSE

DISPLAY "Invalid choice. Please try again."

END IF

END WHILE

END

FUNCTION LoadData()

OPEN file "course\_data.txt"

FOR each line in file DO

PARSE line into courseNumber, name, prerequisites

ADD course to data structure

END FOR

CLOSE file

DISPLAY "Data loaded successfully."

END FUNCTION

FUNCTION PrintSortedCourses()

IF data structure is empty THEN

DISPLAY "Data not loaded. Please load the data first."

RETURN

END IF

SORT courses in data structure alphanumerically by courseNumber

FOR each course in sorted data structure DO

DISPLAY courseNumber, name

END FOR

END FUNCTION

FUNCTION PrintCourseDetails()

IF data structure is empty THEN

DISPLAY "Data not loaded. Please load the data first."

RETURN

END IF

DISPLAY "Enter course number:"

READ courseNumber

IF courseNumber exists in data structure THEN

DISPLAY courseNumber, name, prerequisites

ELSE

DISPLAY "Course not found."

END IF

END FUNCTION

**EvaluationA screenshot of a computer

Description automatically generated**

The overall worst-case running time is O(nm), where ( n ) is the number of courses and ( m ) is the average number of prerequisites per course.

**A screenshot of a computer

Description automatically generated**

The overall worst-case running time is O(n), where ( n ) is the number of courses.

**A screenshot of a computer

Description automatically generated**

The overall worst-case running time is O(nm), where ( n ) is the number of courses and ( m ) is the average number of prerequisites per course.

**Analysis of Data Structures**

Vector Data Structure

Advantages:

* Simple Implementation
* Dynamic Size
* Random Access

Disadvantages:

* Insertion and Deletion
* Memory Overhead

Hash Table Data Structure

Advantages:

* Fast Access
* Efficient Use of Space

Disadvantages:

* Collisions
* Hash Function

Tree Data Structure (Binary Search Tree)

Advantages:

* Sorted Order
* Balanced Trees

Disadvantages:

* Complex Implementation
* Unbalanced Trees

Based off the Big O analysis I believe that the vector data structure would be best for this project. Because the runtime for this project would be O(n) and it is simple to implement.