Project 1

Joseph Eaton

Southern New Hampshire University

CS-300 DSA: Analysis and Design

October 20th, 2024

1. Vector Pseudo code

BEGIN

FUNCTION loadCourses(String filename)

OPEN file with filename

IF file does not exist THEN

PRINT "Error: File not found."

RETURN empty vector

END IF

CREATE vector<Course> courses

WHILE not at the end of the file

READ line from file

SPLIT line by comma into tokens

IF tokens.size < 2 THEN

PRINT "Error: Line does not contain enough parameters."

CONTINUE

END IF

SET courseNumber = tokens[0]

SET courseTitle = tokens[1]

CREATE vector<String> prerequisites

FOR i = 2 TO tokens.size - 1

ADD tokens[i] to prerequisites

END FOR

// Create a Course object and add to the courses vector

CREATE course object using createCourse(courseNumber, courseTitle, prerequisites)

ADD course object to courses vector

END WHILE

CLOSE file

RETURN courses

END FUNCTION

// Validating the files format

BEGIN

FUNCTION validateFile(vector<Course> courses)

CREATE set<String> courseNumbers

FOR each course inside of courses

ADD course.courseNumber to courseNumbers

END FOR

FOR each course inside of courses

FOR each prerequisite that is in course.prerequisite

IF prerequisite is NOT IN courseNumbers THEN

PRINT “Error Prerequisite does not exist”

RETURN FALSE

END IF

END FOR

END FOR

RETURN TRUE

END FUNCTION

BEGIN

FUNCTION createCourse( String courseNumber, String CourseTitle, vector<String> prerequisite)

CREATE Course Object

SET object.courseNumber = courseNumber

SET object.courseTitle = courseTitle

Set object.prerequisetes = prerequisites

RETURN object

END Function

BEGIN

FUNCTION printCourses(vector<Course> courses)

SORT courses by courseNumber

FOR each course in courses

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

IF course.prerequisites is empty THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites: " + JOIN course.prerequisites WITH ", "

END IF

END FOR

END FUNCTION

END

2. Hash Tables

**//** Function to load course data into program from a selected file

FUNCTION LoadCourses( filename):

OPEN file FOR reading

IF file cant be opened THEN

PRINT “File cannot be opened ”

RETURN

END IF

FOR each line in the file:

SPLIT line into tokens by comma

SET courseNumber = the first token

SET courseTitle = the second token

IF line does not have 2 tokens THEN

PRINT “Error missing course title or number”

CONTINUE

END IF

CREATE new Course object with courseNumber and courseTitle

IF there are additional tokens

FOR each prerequisite token

SET prerequisite = to token

IF prerequisite does not exist as courseNumber in the hashtable THEN

PRINT “Preequisite does not exist.” THEN

CONTINUE to next line

END IF

ADD prerequisite to the Course prerequisites list

END FOR

END IF

CALL AddCourseToHashTable(hashTable, Course Object)

END FOR

CLOSE file

END FUNCTION

// Function to create the Course Object

FUNCTION CreateCourse(courseNumber, courseTitle, prerequisites)

CREATE Course object

SET object.courseNumber = courseNumber

SET object.courseTitle = courseTitle

SET object.prerequisites = prerequisites

RETURN object

END FUNCTION

// Function to add selected courses to the hash table

FUNCTION Add CourseToHashTable( hashtable, course )

CALCULATE hash value from course.courseNumber

IF hashTable [hashValue] is empty THEN

SET hashTable[hashValue] = course

ELSE

HANDLE collision

ADD course to the appropriate location within the hashTable

END IF

END FUNCTION

// Function to print the course information that is stored in the hash table.

FUNCTION PrintCourseInformation(hashTable)

FOR each course in hashTable

IF course is not null then

ADD course to courseList

END IF

END FOR

SORT courseList by courseNumber

FOR each course in courseList

**PRINT “Course Number: ” + course.courseNumber**

PRINT “Course Title : “ + corse.courseTitle

IF course.prerequistes is empty THEN

PRINT “No Prerequisites”

**ELSE**

PRINT “Prerequistes: “ + JOIN course.prerequisites with “, “

END IF

END FOR

END FUNCTION

3.Binary Search Tree

// Function to open file and check for any formatting errors

FUNCTION LoadCoursesFromFile(fileName)

OPEN file FROM fileName

IF file cannot be opened THEN

PRINT “Error cannot open file”

**RETURN**

END IF

WHILE lines in file

READ current line

PARSE current line

IF number of elements in line less than 2 THEN

PRINT “Invalid Format ”

CONTINUE to next LINE

**END IF**

// Extracts the course number title and the prerequisites.

SET courseNumber = first part of line

SET courseTitle = second part of line

SET prerequisites = last part of line

// FOR loop to validate the prerequisites.

FOR each prerequisite in prerequisites

IF prerequisite is not in file THEN

PRINT “Error prerequisite is not found ”

CONTINUE

END FOR

// Creates a course object for the line

Course = CreateCourseObject (courseNumber, courseTitle, prerequisites ) **CALL InsertCourseInTree(course)**

**END WHILE**

CLOSE FILE

END FUNCTION

**//** struct definition for the course object

STRUCT Course

**courseNumber**

**CourseTitle**

**Prerequisites**

**END STRUCT**

// function to create the course object

FUNCTION CreateCourseObject

Course = new Course

Course.courseNumber = courseNumber

Course.courseTitle = courseTitle

Course.prerequisites = prerequisites

**RETURN course**

END FUNCTION

//Function that inserts courses into a binary search tree

FUNCTION InsertCourseInTree(course)

**IF tree is empty** THEN

SET root equal to course

**ELSE**

​ SET currentNode = root

WHILE currentNode is not null

IF course.courseNumber is greater than currentNode.courseNumber IF left child of currentNode is null THEN

SET left child of currentNode to course

RETURN

ELSE

SET currentNode = currentNode.left

END IF

ELSE

IF right child of currentNode is null THEN

SET right child of currentNode to course

RETURN

Else

SET currentNode = currentNode.right

**END IF**

**END IF**

**END WHILE**

**END IF**

**END FUNCTION**

**//**In-order traversal to print the courses in a specific order.

FUNCTION PrintCourses(node)

IF node is not null

CALL PrintCourses(node.left)

PRINT “ Course Number: “ + node.courseNumber

PRINT “Course Title: “ + node.courseTitle

IF node.prerequisites is empty THEN

PRINT “Prerequisites: None”

ELSE

PRINT “Prerequisites: “ + join(node.prerequisites) WITH ”, ”

END IF

CALL PrintCourses(node.right)

END IF

END FUNCTION

// Function to print from the root of the tree

FUNCTION DisplayCourseInformation():

IF tree is empty THEN

PRINT “No courses available currently”

ELSE:

CALL PrintCourses(tree.root)

END IF

END Function

FUNCTION DisplayMenu()

SET fileLoaded to False

WHILE TRUE

PRINT “Menu:”

PRINT “1: Load the file data into the data structure.”

PRINT “2: Print all courses in alphanumeric order"

PRINT “3: Print course details”

PRINT “9: Exit”

SET choice to user input

IF choice is 1 THEN

PRINT “Enter file name”

set fileName to user input

CALL LoadCourses(fileName)

SET fileLoaded to True

PRINT “Course file loaded successfully.”

ELSE IF choice is 2 THEN

IF fileLoaded is FALSE THEN

PRINT “Error you must load a file first”

ELSE

CALL PrintCourses()

END IF

ELSE IF choice is 3 THEN

IF fileLoaded is FALSE THEN

PRINT “Error you must load a file first”

ELSE

PRINT "Enter the course number:"

SET courseNumber = to user input

CALL PrintCourseDetails(courseNumber)

END IF

ELSE IF choice == 9 THEN

PRINT "Closing the program."

BREAK

ELSE

PRINT "Invalid option. Please try again."

END IF

END WHILE

END FUNCTION

Vector Complexity Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Number of times executed | Total Cost |
| Open File | 1 | 1 | 1 |
| IF file does not exist | 1 | 1 | 1 |
| CREATE vector<course | 1 | 1 | 1 |
| WHILE not at the end of the file | 1 | n | n |
| READ line from file | 1 | n | n |
| SPLIT line by comma into tokens | 1 | n | n |
| IF tokens.size < 2 | 1 | n | n |
| SET courseNumber = tokens[0] | 1 | n | n |
| SET courseTitle = tokens[1] | 1 | n | n |
| CREATE vector<String> prerequisites | 1 | n | n |
| FOR i = 2 TO token.size-1 | 1 | n | n |
| ADD tokens[i] to prerequisites | 1 | n | n |
| CREATE course object | 1 | n | n |
| ADD course object to courses vector | 1 | n | n |
| CLOSE file | 1 | 1 | 1 |
| CREATE set<String> courseNumbers | 1 | 1 | 1 |
| FOR each course inside of courses | 1 | n | n |
| ADD course.courseNumber to courseNumbers | 1 | n | n |
| FOR each course inside of courses | 1 | n | n |
| FOR each prerequisite in course.prerequisite | 1 | (n\*p) | (n\*p) |
| IF prerequisite is NOT IN courseNumbers THEN PRINT "Error Prerequisite does not exist" | 1 | (n\*p) | (n\*p) |
| CREATE Course Object | 1 | n | n |
| SET object.courseNumber = courseNumber | 1 | n | n |
| SET object.courseTitle = courseTitle | 1 | n | n |
| SET object.prerequisites = prerequisites | 1 | n | n |
| RETURN object | 1 | n | n |
| Worst Case time complexity | | | O(n) |

Hash Table Complexity Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Number of times executed | Total Cost |
| Open File | 1 | 1 | 1 |
| IF file cannot be opened | 1 | 1 | 1 |
| CREATE vector<course | 1 | n | n |
| FOR each line in the file | 1 | n | n |
| SPLIT line into tokens | 1 | n | n |
| SET courseNumber = tokens[0] | 1 | n | n |
| SET courseTitle = tokens[1] | 1 | n | n |
| IF tokens.size < 2 | 1 | n | n |
| CREATE course object | 1 | n | n |
| IF there are additional tokens | 1 | n | n |
| FOR each prerequisite token | 1 | n | n |
| SET prerequisite = token | 1 | n | n |
| IF prerequisite does not exist in hash | 1 | n | n |
| ADD prerequisite to course prerequisites | 1 | n | n |
| CALL AddCourseToHashTable | 1 | n | n |
| CLOSE file | 1 | 1 | 1 |
| Worst Case time complexity | | | O(n^2) |

Binery Search Tree Complexity Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | Number of times executed | Total Cost |
| OPEN file FROM fileName | 1 | 1 | 1 |
| IF file cannot be opened | 1 | 1 | 1 |
| WHILE lines in file | 1 | n | n |
| READ current line | 1 | n | n |
| PARSE current line | 1 | n | n |
| IF number of elements in line < 2 | 1 | n | n |
| SET courseNumber = first part of line | 1 | n | n |
| SET courseTitle = second part of line | 1 | n | n |
| SET prerequisites = last part of line | 1 | n | n |
| FOR each prerequisite in prerequisites | 1 | (n\*p) | (n\*p) |
| IF prerequisite is not in file THEN PRINT "Error prerequisite not found" | 1 | (n\*p | (n\*p |
| CREATE Course object = CreateCourseObject(courseNumber, courseTitle, prerequisites) | 1 | n | n |
| CALL InsertCourseInTree(course) | 1 | 1 | 1 |
| CLOSE FILE | 1 | n | n |
| CREATE Course object | 1 | n | n |
| SET object.courseNumber = courseNumber | 1 | n | n |
| SET object.courseTitle = courseTitle | 1 | n | n |
| SET object.prerequisites = prerequisites | 1 | n | n |
| RETURN object | 1 | n | 1 |
| IF tree is empty THEN SET root equal to course | 1 | 1 | 1 |
| WHILE currentNode is not null | 1 | n | n |
| IF course.courseNumber > currentNode.courseNumber | 1 | n | n |
| SET left or right child | 1 | n | n |
| Worst Case time complexity | | | O(n^2) |

**Explain the advantages and disadvantages of each structure in your evaluation.**

**When taking a look at the three choices between hashtable, vector and binary search tree we can look at the advantages and disadvantages that they all bring when looking at our overall goal of the program. When it comes to sorting out our list of course binary search trees are going to be the best at doing that as they naturally maintain a sorted order while hash tables and vectors are required to be sorted. But when it comes to looking up a specific course Hash Tables are going to be the quickest in a constant time of O(1) versus O(n) for a unbalanced Binary search tree. And Vectors use a linear search O(n) making them not efficient at looking up induvial courses due to these requirements while we will need to account sorting for the Hash Table its ability to quickly find a course and its prerequisites make it the top pick.**