Kendal Guizado O’riley

Project One

1) Pseudocode

Vector:

Function readAndStoreCourses(filename):

coursesVector = empty vector

Try:

Open the file named filename

For each line in the file:

tokens = Split the line by comma

If length of tokens < 2:

Print "Error: Insufficient parameters on line"

Continue to next line

courseNumber = tokens[0]

courseTitle = tokens[1]

prerequisites = empty list

If length of tokens > 2:

For i from 2 to length of tokens - 1:

prerequisite = tokens[i]

If courseExists(prerequisite, coursesVector):

prerequisites.append(prerequisite)

Else:

Print "Error: Prerequisite not found for course ", courseNumber

Continue to next line

course = createCourse(courseNumber, courseTitle, prerequisites)

coursesVector.append(course)

Close the file

Catch any file IO errors:

Print "Error: Unable to open or read file"

Function courseExists(courseNumber, coursesVector):

For each course in coursesVector:

If course.courseNumber == courseNumber:

Return true

Return false

Function createCourse(courseNumber, courseTitle, prerequisites):

course = new CourseObject

course.courseNumber = courseNumber

course.courseTitle = courseTitle

course.prerequisites = prerequisites

Return course

B) Pseudocode that will print out information from a data structure that meets requirements

1. Define a function to search and print course information

Input: courseNumber (string), vector data structure containing course objects

Output: Print course information and prerequisites

2. Search for the courseNumber in the vector data structure:

a. Initialize a boolean flag foundCourse to false

b. Iterate through each course object in the vector:

i. If the courseNumber matches the current course object's course number:

- Set foundCourse flag to true

- Print course information including course number, title, and prerequisites (if any)

- Break out of the loop

c. If foundCourse is still false after the loop:

- Print "Course not found."

3. End of function

Function searchAndPrintCourseInfo(courseNumber, coursesVector):

courseFound = false

For each course in coursesVector:

If course.courseNumber == courseNumber:

Print "Course Number:", course.courseNumber

Print "Course Title:", course.courseTitle

If course.prerequisites is not empty:

Print "Prerequisites:"

For each prerequisite in course.prerequisites:

Print prerequisite

Else:

Print "No prerequisites for this course."

courseFound = true

Exit loop

If courseFound is false:

Print "Error: Course not found in database"

Hash Table:

// Vector pseudocode

int numPrerequisiteCourses(Vector<Course> courses, Course c) {

totalPrerequisites = prerequisites of course c

for each prerequisite p in totalPrerequisites

add prerequisites of p to totalPrerequisites

print number of totalPrerequisites

}

void printSampleSchedule(Vector<Course> courses) {  
 for each course c in courses

print c.courseNumber, c.title, c.prerequisites

}

void printCourseInformation(Vector<Course> courses, String courseNumber) {

**for all courses**

**if the course is the same as courseNumber**

**print out the course information**

**for each prerequisite of the course**

**print the prerequisite course information**

}

// Hashtable pseudocode

int numPrerequisiteCourses(Hashtable<Course> courses) {

Initialize variable totalPrerequisitesCount to 0

// Iterate through each course in the Hashtable

for each Course c in courses

// Increment totalPrerequisitesCount by the number of prerequisites for current course

totalPrerequisitesCount += size of c.prerequisites list

Return totalPrerequisitesCount

}

void printSampleSchedule(Hashtable<Course> courses) {

// Iterate through each course in the Hashtable

for each Course c in courses

// Print course information

print c.courseNumber, c.title

}

void printCourseInformation(Hashtable<Course> courses, String courseNumber) {

// Check if the courseNumber exists in the Hashtable

if courses contains courseNumber

// Retrieve the course object associated with the courseNumber

Course course = courses.get(courseNumber)

// Print course information

print course.courseNumber, course.title

// Check if there are any prerequisites for the course

if course.prerequisites is not empty

print "Prerequisites:"

// Iterate through each prerequisite of the course

for each Prerequisite p in course.prerequisites

// Print prerequisite information

print p.courseNumber, p.title

else

print "No prerequisites"

else

// Print error message if courseNumber does not exist in the Hashtable

print "Course not found"

}

// Tree pseudocode

int numPrerequisiteCourses(Tree<Course> courses) {

// If the tree is empty, return 0

if courses.getRoot() is null

return 0

// Call a recursive function to count prerequisites starting from the root

return countPrerequisites(courses.getRoot())

}

void printSampleSchedule(Tree<Course> courses) {

void printSampleSchedule(Tree<Course> courses) {

// If the tree is empty, print "No courses available"

if courses.getRoot() is null

print "No courses available"

else

// Call a recursive function to traverse and print courses starting from the root

printSchedule(courses.getRoot())

}

void printCourseInformation(Tree<Course> courses, String courseNumber) {

// Call a recursive function to search for the course with the given courseNumber starting from the root

TreeNode<Course> courseNode = searchCourse(courses.getRoot(), courseNumber)

// If the course with the given courseNumber is found

if courseNode is not null

// Print course information

printCourseInfo(courseNode.data)

else

// Print error message if course is not found

print "Course not found"

}

Tree:

Initialize a binary tree data structure to store course information

tree = BinarySearchTree()

Define a function to open and read the data file

function readCourseDataFile(filename):

try:

// Open file for reading

file = open(filename, "r")

// Read lines of the file

for line in file:

// split the line into tokens using a delimiter (e.g., comma or tab)

tokens = line.split(",")

// Check if there are at least two parameters on each line

if length(tokens) < 2:

print("Error: Invalid format in line:", line)

continue

// Extract course information from the tokens

courseNumber = tokens[0].strip()

courseTitle = tokens[1].strip()

// Check if there are prerequisites specified

if length(tokens) > 2:

prerequisites = tokens[2:] //Extract prerequisites as a list

else:

prerequisites = [] // No prerequisites

// Create a new course object with the extracted data

newCourse = Course(courseNumber, courseTitle, prerequisites)

// Insert the course object into the binary tree

tree.insert(newCourse)

// Close the file

file.close()

except FileNotFoundError:

print("File not found.")

//Define a function to print course information and prerequisites

function printCourseInformation(courseNumber):

// Search for the course in the binary tree

foundCourse = tree.search(courseNumber)

if foundCourse is not None:

// Print course information

print("Course Number:", foundCourse.courseNumber)

print("Course Title:", foundCourse.courseTitle)

// Check if there are prerequisites

if length(foundCourse.prerequisites) > 0:

print("Prerequisites:")

for prerequisite in foundCourse.prerequisites:

print("- ", prerequisite)

else:

print("No prerequisites for this course.")

else:

print("Course not found.")

// Main

if \_\_name\_\_ == "\_\_main\_\_":

// file input

filename = input("Enter the name of file")

// Read and process the course data from the file

readCourseDataFile(filename)

// Prompt users to enter a course number for retrieval

courseNumberToRetrieve = input("Enter a course number ")

// Print course information and prerequisites

printCourseInformation(courseNumberToRetrieve)

2) Pseudocode for a menu

Menu():

display "Menu Options:"

display "1. Load Data Structure"

display "2. Print Course List"

display "3. Print Course"

display "4. Exit"

// user input

switch choice:

case 1:

LoadDataStructure(filename)

case 2:

if data structure is empty:

print "Data structure is empty. Load data."

else:

PrintCourseList()

case 3:

if data structure is empty:

print "Data structure is empty. Load data."

else:

take user input courseNumber

PrintCourse(courseNumber)

case 4:

exit program

default:

print "Invalid choice. Please enter a valid option."

repeat Menu()

3) Pseudocode that will print out the list of courses in alphanumeric order

Vector:

PrintCourseList():

sort coursesVector by course number

for each course in coursesVector:

print course title

Hash Table:

PrintCourseList():

create an empty list sortedCourses

for each course in coursesHashTable:

add course to sortedCourses

sort sortedCourses by course number

for each course in sortedCourses:

print course title

Tree:

PrintCourseList():

inorderTraversal(coursesTree)

inorderTraversal(node):

if node is not null:

inorderTraversal(node.leftChild)

print node.course.title

inorderTraversal(node.rightChild)

4) Evaluation

|  |  |  |  |
| --- | --- | --- | --- |
| Operation |  |  |  |
| Open file | 1 | 1 | 1 |
| Read line | 1 | N | N |
| Parse line | 1 | N | N |
| Create course object | 1 | N | N |
| Add course object | 1 | N | N |
| Close file | 1 | 1 | 1 |
|  |  |  |  |

Each line of code has a cost of 1, assuming no function calls. The loop iterates 'n' times, where 'n' is the number of courses in the file. Inside the loop, the operations of reading, parsing, creating course objects, and adding them to the data structure each execute 'n' times. The runtime is O(n)

5)

Vector:

Advantages: Easy to implement. Allows for random access, which is useful for operations like directly accessing a course by index.

Disadvantages: Inefficient for inserting and deleting elements, especially if done frequently or in large quantities. Sorting the vector can be expensive, leading to higher time complexity.

Hash Table:

Advantages: Quick access to elements on average (O(1) time complexity for retrieval).

Well-suited for fast retrieval, such as printing individual course information.

Disadvantages: Difficulty in maintaining sorted order. Higher memory consumption compared to vectors.

Tree:

Advantages: Maintains sorted order inherently, making it efficient for printing the list of courses in alphanumeric order. Efficient for searching, inserting, and deleting elements with a balanced tree structure.

Disadvantages: More complex implementation compared to vectors and hash tables.

Traversal operations might be slightly slower compared to hash table lookups.

6) Recommendation:

After analyzing the requirements and characteristics of each data structure. I recommend using a tree for this project. While hash tables offer fast access to data, they struggle with maintaining sorted order. Vectors are inefficient for sorting and maintaining order, especially with larger datasets. However, trees naturally maintain sorted order and provide efficient operations for printing the course list and searching for individual courses. Additionally, the time complexity of tree operations aligns well with the project's requirements. Therefore, a tree data structure appears to be the most suitable choice for this task.