* 1. Project One

**1. Resubmit pseudocode from previous pseudocode assignments and update as necessary.**

WHILE file is open

print course information

store data gathered in a data structure

}

CS-300: Milestone Tree Data Structure Pseudocode

1.

Start program

create two nodes variables for left and right;

create root variable set it to null;

create variables for course name, and an integer for course number;

open file

WHILE file is open

read data

parse each line

- check for course title

- check for course number

IF root != null

check if a prereq found

add prereq to right node

IF course parameters are < two

// Vector pseudocode

function loadCoursesVector(filename):

file = open(filename) // Open the file

courses = createEmptyVector() // Create an empty vector data structure

while not endOfFile(file): // Iterate through each line in the file

line = readLine(file) // Read a line from the file

courseData = parseLine(line) // Parse the line to extract course data

if validateCourseData(courseData): // Validate the course data

course = createCourseObject(courseData) // Create a course object

courses.append(course) // Add the course object to the vector

else:

print("Error: Invalid course data in line:", line)

return courses // Return the populated vector

// Hashtable pseudocode

function loadCoursesHashtable(filename):

file = open(filename) // Open the file

courses = createEmptyHashtable() // Create an empty hashtable data structure

while not endOfFile(file): // Iterate through each line in the file

line = readLine(file) // Read a line from the file

courseData = parseLine(line) // Parse the line to extract course data

if validateCourseData(courseData): // Validate the course data

course = createCourseObject(courseData) // Create a course object

courses.put(course.courseNumber, course) // Add the course object to the hashtable

else:

print("Error: Invalid course data in line:", line)

return courses // Return the populated hashtable

// Tree pseudocode

function loadCoursesTree(filename):

file = open(filename) // Open the file

tree = createEmptyTree() // Create an empty tree data structure

while not endOfFile(file): // Iterate through each line in the file

line = readLine(file) // Read a line from the file

courseData = parseLine(line) // Parse the line to extract course data

if validateCourseData(courseData): // Validate the course data

course = createCourseObject(courseData) // Create a course object

if not tree.contains(course.courseNumber): // Check if the course already exists in the tree

tree.insert(course.courseNumber, course) // Insert the course into the tree

else:

print("Error: Duplicate course found:", course.courseNumber)

else:

print("Error: Invalid course data in line:", line)

return tree // Return the populated tree

function parseLine(line):

tokens = splitLine(line) // Split the line into tokens

courseNumber = tokens[0] // Extract the course number

courseTitle = tokens[1] // Extract the course title

prerequisites = tokens[2:] // Extract the prerequisites

return (courseNumber, courseTitle, prerequisites) // Return the parsed course data

function validateCourseData(courseData):

courseNumber, courseTitle, prerequisites = courseData

if length(courseData) < 2: // Check if there are at least two parameters

return false

for prerequisite in prerequisites:

if not tree.contains(prerequisite): // Check if the prerequisite course exists in the tree

return false

return true

// Pseudocode for printing course information and prerequisites

function printCourseInformation(courses, courseNumber):

for course in courses:

if course.courseNumber == courseNumber:

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

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

print("Prerequisites:")

for prerequisite in course.prerequisites:

print("- Prerequisite Course:", prerequisite)

printCourseInformation(courses, prerequisite) // Print prerequisite course  
  
// Print Course List

function printCourseList(courses):

sortedCourses = sortCourses(courses) // Sort the courses in alphanumerical order

for course in sortedCourses:

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

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

print("Prerequisites:")

for prerequisite in course.prerequisites:

print("- Prerequisite Course:", prerequisite)

// Sort Courses

function sortCourses(courses):

sortedCourses = sort(courses) // Implement a sorting algorithm to sort the courses

return sortedCourses

**2. Create pseudocode for a menu**.

// Menu pseudocode

function menu():

dataStructure = None // Variable to store the selected data structure

courses = None // Variable to store the loaded courses data

while true:

print("1. Load Data Structure")

print("2. Print Course List")

print("3. Print Course")

print("4. Exit")

choice = readInput("Enter your choice: ")

if choice == "1":

filename = readInput("Enter the filename: ")

dataStructure = selectDataStructure() // Prompt user to select a data structure

if dataStructure == "vector":

courses = loadCoursesVector(filename)

else if dataStructure == "hashtable":

courses = loadCoursesHashtable(filename)

else if dataStructure == "tree":

courses = loadCoursesTree(filename)

else if choice == "2":

if courses is None:

print("Error: No data structure loaded. Please load a data structure first.")

else:

printCourseList(courses)

else if choice == "3":

if courses is None:

print("Error: No data structure loaded. Please load a data structure first.")

else:

courseNumber = readInput("Enter the course number: ")

printCourseInformation(courses, courseNumber)

else if choice == "4":

break

else:

print("Invalid choice. Please try again.")

// Select Data Structure

function selectDataStructure():

while true:

print("Select Data Structure:")

print("1. Vector")

print("2. Hashtable")

print("3. Tree")

choice = readInput("Enter your choice: ")

if choice == "1":

return "vector"

else if choice == "2":

return "hashtable"

else if choice == "3":

return "tree"

else:

print("Invalid choice. Please try again.")

// Entry point of the program

function main():

menu()

// Start the program

main()  
  
3. Design **pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order.**

// Print Sorted Course List

function printSortedCourseList(courses):

sortedCourses = sortCoursesByCourseNumber(courses) // Sort the courses by alphanumeric course number

for course in sortedCourses:

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

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

print("Prerequisites:")

for prerequisite in course.prerequisites:

print("- Prerequisite Course:", prerequisite)

// Sort Courses by Course Number

function sortCoursesByCourseNumber(courses):

sortedCourses = sort(courses, "courseNumber") // Implement a sorting algorithm to sort the courses by course number

return sortedCourses

// Modify the menu() function to include the new option

else if choice == "5":

if courses is None:

print("Error: No data structure loaded. Please load a data structure first.")

else:

printSortedCourseList(courses)

**4. Evaluate the run-time and memory of data structures that could be used to address the requirements**.

a. The pseudocode for opening the file, reading the data, parsing each line, and checking for formatting errors does not have any loops or iterations. It executes in a linear manner, processing each line once. Therefore, the worst-case running time complexity of this pseudocode is O(n), where n is the number of lines in the file.  
  
b. The pseudocode for creating course objects also does not have any loops or iterations. It executes once for each line of data read from the file. Therefore, the worst-case running time complexity of this pseudocode is also O(n), where n is the number of lines in the file.  
  
c. In the pseudocode for opening the file, reading the data, parsing each line, and checking for formatting errors, each line of code has a cost of 1, and it executes once for each line in the file. So the total cost is 1 \* n = n.  
  
d. In the pseudocode for creating course objects, the cost per line of code is again 1, and it executes once for each line in the file. So the total cost is 1 \* n = n. In both cases, the cost per line of code is assumed to be 1, as specified. The number of times each line executes is directly proportional to the number of lines in the file, which is represented by n.Therefore, the worst-case running time complexity of both pseudocode segments is O(n), where n is the number of lines in the file.

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

Vector:

* Advantages:
  + Simple and straightforward data structure to implement.
  + Elements are stored in contiguous memory locations, allowing for efficient memory access.
  + Supports dynamic resizing, allowing for flexibility in the number of courses.
* Disadvantages:
  + Insertions and deletions at differing positions can be inefficient as elements may need to be shifted.
  + Searching for a specific course requires iterating through the vector, resulting in a linear search time.
  + Sorting the vector by course number requires additional sorting algorithms, which may have a higher time complexity.

Hash Table:

* Advantages:
  + Provides fast access and retrieval of course objects based on their course number.
  + Efficient insertion and deletion operations, especially when the hash function distributes the data evenly.
  + Can handle many courses while maintaining a relatively constant time complexity for operations.
* Disadvantages:
  + The efficiency of a hash table depends on the quality of the hash function and the avoidance of errors.
  + Does not provide inherent ordering of the courses unless additional measures are taken.

Tree:

* Advantages:
  + Maintains a sorted order of courses, making it efficient for searching and traversing in a specific order.
  + Supports efficient insertion and deletion operations while maintaining the sorted property.
  + Can handle many courses without a significant impact on performance.
* Disadvantages:
  + More complex to implement compared to vectors and hash tables.
  + Requires additional memory for storing the tree structure.

6**.** **Now that you have analyzed all three data structures, make a recommendation for which data structure you will plan to use in your code.**

After thoroughly analyzing all three data structures, I would highly recommend utilizing Hash tables for the advisor's program. Despite the potential drawback of slightly slower performance, Hash tables offer numerous advantages that make them an ideal choice. One significant advantage is their ability to provide a structured and organized approach to storing and accessing data. This aspect becomes crucial, especially if there is a possibility of expanding the project's functionality in the future. While the current project focuses on essential features like sorting courses, it is essential to consider future requirements. By adopting Hash tables, the program gains flexibility and scalability. This means that incorporating additional functionalities or expanding the scope of the project becomes effortless.