CS 300 Project One: Pseudocode and Runtime Analysis

Introduction and Overview  
 This project designs an advising program that reads course data from a text file to (1) print an alphanumerically sorted list of computer science courses, and (2) display a selected course’s title and prerequisites. The document below presents essential pseudocode for three data structures (vector, hash table, and binary search tree), a brief runtime analysis (using the vector as an example), and an evaluation with a recommendation for the coding phase.

I. Pseudocode

*A. Common Functionality (File Handling & Course Object Creation)*

vbnet

Copy

FUNCTION loadData(fileName):

IF fileName does not end with ".txt" THEN

PRINT "Invalid file format"

RETURN

END IF

OPEN file using fileName

WHILE not end of file DO

READ line

SPLIT line by commas into tokens

IF tokens.length < 2 THEN

PRINT "Data is not valid"

CONTINUE

END IF

CREATE Course with:

courseNumber = tokens[0]

courseName = tokens[1]

prerequisites = EMPTY LIST

FOR each token from tokens[2] to tokens[end] DO

ADD token to prerequisites

END FOR

INSERT Course into chosen data structure

END WHILE

CLOSE file

END FUNCTION

*B. Data Structure–Specific Pseudocode*

Vector Implementation

vbnet

Copy

Data Structure: Vector<Course> courseList

FUNCTION loadData\_Vector(fileName):

CALL loadData(fileName)

FOR each Course created:

courseList.push\_back(Course)

END FOR

END FUNCTION

FUNCTION printCourseList\_Vector():

SORT courseList by courseNumber (alphanumeric)

FOR each course in courseList DO:

PRINT course.courseNumber, course.courseName

END FOR

END FUNCTION

FUNCTION searchCourse\_Vector(courseNumber):

FOR each course in courseList DO:

IF course.courseNumber equals courseNumber THEN:

PRINT course.courseNumber, course.courseName

FOR each prereq in course.prerequisites DO:

PRINT prereq

END FOR

RETURN

END IF

END FOR

PRINT "Course Not Found"

END FUNCTION

Hash Table Implementation

vbnet

Copy

Data Structure: HashTable<String, Course> courseTable

FUNCTION loadData\_HashTable(fileName):

CALL loadData(fileName)

FOR each Course created:

courseTable.insert(key = course.courseNumber, value = Course)

END FOR

END FUNCTION

FUNCTION printCourseList\_HashTable():

list ← EXTRACT Courses from courseTable; SORT list by courseNumber

FOR each course in list DO:

PRINT course.courseNumber, course.courseName

END FOR

END FUNCTION

FUNCTION searchCourse\_HashTable(courseNumber):

IF courseTable.contains(courseNumber) THEN:

course ← courseTable.get(courseNumber)

PRINT course.courseNumber, course.courseName

FOR each prereq in course.prerequisites DO:

PRINT prereq

END FOR

ELSE:

PRINT "Course Not Found"

END IF

END FUNCTION

Binary Search Tree (BST) Implementation

vbnet

Copy

Data Structure: Tree<Course> courseTree

FUNCTION loadData\_Tree(fileName):

CALL loadData(fileName)

FOR each Course created:

courseTree.insert(course.courseNumber, Course)

END FOR

END FUNCTION

FUNCTION printCourseList\_Tree():

sortedList ← courseTree.inOrderTraversal()

FOR each course in sortedList DO:

PRINT course.courseNumber, course.courseName

END FOR

END FUNCTION

FUNCTION searchCourse\_Tree(courseNumber):

course ← courseTree.search(courseNumber)

IF course exists THEN:

PRINT course.courseNumber, course.courseName

FOR each prereq in course.prerequisites DO:

PRINT prereq

END FOR

ELSE:

PRINT "Course Not Found"

END IF

END FUNCTION

*C. Menu for User Interaction*

vbnet

Copy

FUNCTION displayMenu():

WHILE userSelection ≠ 9 DO:

PRINT "1. Load Data"

PRINT "2. Print Course List"

PRINT "3. Print Course Details"

PRINT "9. Exit"

PROMPT "Please make a selection:"

READ userSelection

SWITCH userSelection:

CASE 1:

PROMPT "Enter file name:"

READ fileName

CALL loadData (using chosen structure's function)

BREAK

CASE 2:

CALL printCourseList (using chosen structure's function)

BREAK

CASE 3:

PROMPT "Enter course number to search:"

READ courseNumber

CALL searchCourse(courseNumber)

BREAK

CASE 9:

PRINT "Thank you. Exiting program."

EXIT LOOP

DEFAULT:

PRINT "Invalid selection. Try again."

END SWITCH

END WHILE

END FUNCTION

II. Runtime Analysis (Vector Example)

| Operation | Cost | # Executions | Total Cost |
| --- | --- | --- | --- |
| Iterate through courses | 1 | n | n |
| Check if courseNumber equals | 1 | n | n |
| Print course information | 2 | 1 | 2 |
| Loop through prerequisites (≤ n times) | 1 | ≤ n | ≤ n |
| Approximate Total |  |  | 6n + 1 |

Worst-case Runtime: O(n)  
 (Note: Similar analysis applies for the hash table and BST with slightly different constant factors.)

III. Evaluation and Recommendation

* Vector  
    • *Advantages:* Simple, low constant cost, efficient memory usage.  
    • *Disadvantages:* Deletion is less flexible; searching is linear unless sorted.
* Hash Table  
    • *Advantages:* Fast average-case lookups; efficient insertions/deletions.  
    • *Disadvantages:* Overhead from hashing; potential collisions add complexity.
* Binary Search Tree  
    • *Advantages:* Naturally sorted via in-order traversal; efficient when balanced.  
    • *Disadvantages:* More complex; worst-case performance deteriorates if unbalanced.

Recommendation:  
 Based on the analysis and simplicity, I recommend using the vector data structure. Its lower overall cost (approximately 6n + 1) and straightforward implementation best serve the advising program’s requirements.

Conclusion  
 This document provides concise pseudocode, a runtime analysis, and a brief evaluation of data structures for the advising program. The vector’s simplicity and efficiency establish a solid foundation as I move forward into coding the project.