CS 300 – Project One: Pseudocode and Runtime Analysis

Raymundo Valencia

CS 300 – Module 6

Date: April 12, 2025

# Pseudocode – Vector Implementation

BEGIN PROGRAM

INITIALIZE CourseList AS an empty vector of Course objects

OPEN file "course\_info.txt" FOR reading

IF file cannot be opened THEN

PRINT "Error: Cannot open file."

TERMINATE program

END IF

WHILE NOT end-of-file DO

READ currentLine FROM file

IF currentLine IS empty THEN

CONTINUE

END IF

SET tokens = SPLIT(currentLine, delimiter)

IF NUMBER\_OF(tokens) < 2 THEN

PRINT "File format error: " + currentLine

TERMINATE program

END IF

SET courseNumber = tokens[0]

SET courseTitle = tokens[1]

INITIALIZE prerequisites AS empty list

FOR index FROM 2 TO NUMBER\_OF(tokens) - 1 DO

APPEND tokens[index] TO prerequisites

END FOR

CREATE Course OBJECT

Course.courseNumber = courseNumber

Course.courseTitle = courseTitle

Course.prerequisites = prerequisites

APPEND Course TO CourseList

END WHILE

CLOSE file

FOR EACH course IN CourseList DO

FOR EACH prereq IN course.prerequisites DO

IF NOT EXISTS (courseItem IN CourseList WHERE courseItem.courseNumber == prereq) THEN

PRINT "Error: Prerequisite " + prereq + " not found."

TERMINATE program

END IF

END FOR

END FOR

END PROGRAM

# Pseudocode – Hash Table Implementation

FUNCTION loadCourses(courseTable, fileName):

OPEN file with fileName

IF file fails to open THEN

PRINT "Error: Unable to open file"

RETURN

CREATE empty list rawLines

FOR EACH line IN file DO

TRIM whitespace

IF line is empty THEN CONTINUE

SPLIT line by commas INTO tokens

IF tokens.length < 2 THEN

PRINT "Invalid line format: " + line

CONTINUE

END IF

ADD tokens TO rawLines

CREATE set courseNumbers

FOR EACH tokens IN rawLines DO

ADD tokens[0] TO courseNumbers

FOR EACH tokens IN rawLines DO

FOR EACH prerequisite IN tokens[2 to end] DO

IF prerequisite NOT IN courseNumbers THEN

PRINT "Error: Prerequisite " + prerequisite + " not found"

RETURN

END IF

END FOR

END FOR

FOR EACH tokens IN rawLines DO

CREATE Course OBJECT

course.courseNumber = tokens[0]

course.courseTitle = tokens[1]

course.prerequisites = tokens[2 to end]

INSERT course INTO courseTable USING course.courseNumber AS key

END FUNCTION

# Pseudocode – Binary Search Tree Implementation

STRUCT Course:

courseNumber, courseName, prerequisites

STRUCT TreeNode:

course, left, right

FUNCTION loadCourses(filename):

OPEN file

IF cannot open THEN

PRINT "Error" and EXIT

CREATE courseMap, courseLines

FOR EACH line IN file DO

SPLIT by commas INTO tokens

IF tokens.length < 2 THEN CONTINUE

EXTRACT courseNumber, courseName, prereqs

CREATE Course OBJECT

ADD course TO courseMap AND courseLines

FOR EACH course IN courseLines DO

FOR EACH prereq IN course.prerequisites DO

IF prereq NOT IN courseMap THEN

PRINT "Error: Prerequisite " + prereq + " not found."

EXIT or SKIP course

INITIALIZE root = NULL

FOR EACH course IN courseLines DO

CALL insertCourse(root, course)

RETURN root

# Shared Menu Pseudocode

FUNCTION displayMenu():

WHILE true:

PRINT options: 1 Load, 2 Print List, 3 Search, 9 Exit

READ input

IF input == 1 THEN CALL loadCourses()

IF input == 2 THEN CALL printCourseList()

IF input == 3 THEN

PROMPT user: "Enter the course number to search:"

CALL searchCourse(courseNumber)

IF input == 9 THEN

PRINT "Exiting"

BREAK

ELSE

PRINT "Invalid option"

# Sorted Print Function (Vector Only)

FUNCTION printCourseList(CourseList):

SORT CourseList BY courseNumber

FOR EACH course IN CourseList DO

PRINT courseNumber + ", " + courseTitle

IF course.prerequisites IS NOT EMPTY THEN

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

ELSE

PRINT "Prerequisites: None"

# Runtime Analysis Table

| Operation | Vector | Hash Table | Binary Search Tree |  
|---------------------|---------------|------------|----------------------------|  
| File Read | O(n) | O(n) | O(n) |  
| Object Creation | O(1) | O(1) | O(log n) avg |  
| Course Lookup | O(n) | O(1) | O(log n) avg |  
| Sorted Print | O(n log n) | ❌ | O(n) (in-order) |

# Data Structure Evaluation

Vector:  
✅ Easy to implement  
✅ Supports sorting  
❌ Slow search (linear time)

Hash Table:  
✅ Fast lookup (average constant time)  
❌ No natural order (no sorted print)  
❌ Requires collision handling

Binary Search Tree:  
✅ Fast search and sort  
✅ Clean structure for ordered data  
❌ Slightly more complex to implement

# Final Recommendation

I recommend using the Binary Search Tree for this project. It balances fast search and supports a sorted print using in-order traversal, which meets both of the advisor's requirements efficiently.  
This recommendation also considers scalability, assuming a much larger course dataset beyond the sample file provided.  
BST operations like insert and search generally operate at O(log n), offering significant performance gains over O(n) operations in vectors, especially when scaling to hundreds of courses.