Anthony Arnett

Project One

CS300 – DSA: Analysis and Design

October 12, 2025

**Pseudocode**

**Common data model**

STRUCT Course

courseNumber: STRING

title: STRING

prerequisities: LIST<STRING>

END STRUCT

**Vector Implementation**

FUNCTION LoadCoursesFromFile(filePath)

IF file cannot be opened THEN

PRINT “File open failed”

RETURN FALSE

END IF

SET coursesVector to an empty list

SET courseIdSet to an empty set

SET errors to an empty list

FOR EACH line IN file

TRIM the line

IF line is empty OR starts with “#” THEN

CONTINUE

END IF

SPLIT line by comma into tokens

TRIM each token

IF number of tokens < 2 THEN

ADD “Line missing course number or title” to errors

CONTINUE

END IF

SET courseNum = tokens[0]

SET title = tokens[1]

SET prereqs = empty list

FOR i = 2 TO (length of tokens – 1)

IF tokens[i] is not empty THEN

ADD tokens[i] to prereqs

END IF

END FOR

CREATE new Course object named course

SET course.courseNumber = courseNum

SET course.title = title

SET course.prerequisites = prereqs

ADD course to coursesVector

ADD courseNum to courseIdSet

END FOR

CLOSE file

RETURN TRUE

END FUNCTION

FUNCTION ValidatePrerequisites()

FOR EACH course IN coursesVector

FOR EACH prereq IN course.prerequisites

IF prereq not in courseIdSet THEN

ADD “Missing prerequisite: “ + prereq to errors

END IF

END FOR

END FOR

END FUNCTION

FUNCTION FindCourseByNumber(target)

FOR EACH c IN coursesVector

IF c.courseNumber == target THEN

RETURN c

END IF

END FOR

RETURN NULL

END FUNCTION

FUNCTION PrintCourse(course)

PRINT course.courseNumber + “, “ + course.title

IF course.prerequisites is empty THEN

PRINT “Prerequisites: None”

ELSE

PRINT “Prerequisities: “ + JOIN(course.prerequisites, “, “)

END IF

END FUNCTION

FUNCTION PrintCourseInfo(courseNumber)

Course = FindCOurseByNumber(courseNumber)

If course is NULL THEN

PRINT “Course not found”

RETURN

END IF

PrintCourse(course)

IF course.prerequisites not empty THEN

PRINT “Prerequisite details:”

FOR EACH pid IN course.prerequisites

prereqCourse = FindCourseByNumber(pid)

IF prereqCourse is NOT NULL THEN

PRINT “- “ + prereqCourse.courseNumber + “, “ + prereqCourse.title

ELSE

PRINT “- “ + pid + “ (missing)”

END IF

END FOR

END IF

END FUNCTION

FUNCTION PrintAllCoursesSorted()

COPY coursesVector to tempList

SORT tempList by courseNumber

FOR EACH course IN tempList

PRINT course.courseNumber + “, “ + course.title

END FOR

END FUNCTION

FUNCTION ShowMenu()

PRINT “1. Load data file”

PRINT “2. Print course list”

PRINT “3. Print single course information”

PRINT “9. Exit”

END FUNCTION

FUNCTION Main()

SET loaded = FALSE

WHILE TRUE

ShowMenu()

INPUT choice

IF choice == 1 THEN

INPUT “Enter file path: “ into path

Loaded = LoadCoursesFromFile(path)

IF loaded THEN PRINT “File loaded successfully”

ELSE IF choice == 2 THEN

IF NOT loaded THEN PRINT “Please load the data first”

ELSE PrintAllCoursesSorted()

ELSE IF choice == 3 THEN

IF NOT loaded THEN PRINT “Please load the data first”

ELSE

INPUT “Enter course number: ” into num  
 PrintCourseInfo(num)

ELSE IF choice == 9 THEN

PRINT “Goodbye”

BREAK

ELSE

PRINT “Invalid input”

END IF

END WHILE

END FUNCTION

**Hash Table Implementation**

CLASS HashTable

FUNCTION Insert(course)

FUNCTION Search(courseNumber)

FUNCTION ForEach(action)

END CLASS

FUNCTION LoadCoursesToHash(filePath, table)

SET allIds to empty set

SET rawLines to empty list

OPEN file at filePath

IF file cannot be opened THEN

PRINT “Cannot open file”

RETURN FALSE

END IF

WHILE NOT end of file

READ line

TRIM line

IF line is empty THEN CONTINUE

SPLIT line by comma into tokens

TRIM each token

IF number of tokens < 2 THEN

PRINT “Error” Incorrect format in “ + line

CONTINUE

END IF

CREATE new Course c

SET c.number = tokens[0]

SET c.title = tokens[1]

SET c.prereqs = empty list

FOR I = 2 TO (length of tokens – 1)

IF tokens[i] is not empty THEN

ADD tokens[i] to c.prereqs

END IF

END FOR

ADD tokens to rawLines

ADD c.number to allIds

CALL table.insert(c)

END WHILE

CLOSE file

FOR EACH tokens IN rawLines

courseNum = tokens[0]

FOR I = 2 TO (length of tokens – 1)

Prereq = TRIM(tokens[i])

IF prereq == “” THEN CONTINUE

IF prereq not in allIds THEN

PRINT “Missing prerequisite “ + prereq + “ for course “ + courseNum

END IF

END FOR

END FOR

RETURN TRUE

END FUNCTION

**Binary Search Tree**

CLASS Course

courseNumber

courseName

prerequisites

END CLASS

CLASS TreeNode

data

left

right

END CLASS

FUNCTION InsertCourse(root, course)

IF root is null THEN

RETURN new TreeNode containing course

END IF

IF course.courseNumber < root.data.courseNumber THEN

Root.left = InsertCourse(root.left, course)

ELSE

Root.right = InsertCourse(root.right, course)

END IF

RETURN root

END FUNCTION

FUNCTION LoadCourses(filePath)

OPEN file at filePath

SET courseList to empty list

FOR EACH line IN file

SPLIT line by comma

IF number of tokens < 2 THEN CONTINUE

CREATE Course object with tokens

ADD it to courseList

END FOR

CLOSE file

SET allIds to set of all course numbers in courseList

FOR EACH course IN courseList

FOR EACH prereq IN course.prerequisites

IF prereq not in allIds THEN

PRINT “Missing prerequisite: “ + prereq

END IF

END FOR

END FOR

SET root = null

FOR EACH course IN courseList

Root = InsertCourse(root, course)

END FOR

RETURN root

END FUNCTION

FUNCTION FindCourse(root, courseNumber)

IF root is null THEN RETURN null

IF courseNumber == root.data.courseNumber THEN RETURN root

IF courseNumber < root.data.courseNumber THEN

RETURN FindCourse(root.left, courseNumber)

ELSE

RETURN FindCourse(root.right, courseNumber)

END IF

END FUNCTION

FUNCTION PrintCourseInfo(root, courseNumber)

node = FindCourse(root, courseNumber)

IF node is null THEN

PRINT "Course not found"

RETURN

END IF

PRINT node.data.courseNumber + ", " + node.data.courseName

IF node.data.prerequisites is empty THEN

PRINT "Prerequisites: None"

ELSE

PRINT "Prerequisites: " + JOIN(node.data.prerequisites, ", ")

END IF

END FUNCTION

FUNCTION InOrderTraversal(node)

IF node is null THEN RETURN

InOrderTraversal(node.left)

PRINT node.data.courseNumber + ", " + node.data.courseName

InOrderTraversal(node.right)

END FUNCTION

**Runtime Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Average Time** | **Worst Case** | **Notes** |
| Vector | O(n) | O(n) | Simple – fast access but slow lookups |
| Hash Table | O(n) | O(n2) | Fast inserts/lookups, but not ordered, sorting must be implemented |
| Binary Search Tree | O(n log n) | O(n2) | Sorted automatically, efficient lookups |

**Advantages & Disadvantages**

*Vector*

* Easy to build, low memory usage
* Sorting required

*Hash Table*

* No order, extra memory overhead

*Binary Search Tree*

* Keeps data sorted, more complex to write the logic

**Recommendation**

Binary Search Tree – It’s the only structure that naturally supports both printing all courses alphabetically and easily searching for one course. Vectors are slow but simple, and hash tables need sorting for output. A binary search tree gives the best balance of efficiency and functionality.