**CS 300 Project One – Pseudocode & Runtime Analysis**

**Andriana Djurdjevic**

**Oct 8, 2025**

**Southern New Hampshire University**

**Vector Data Structure**

STRUCT Course

STRING number

STRING name

LIST<STRING> prereqs

END STRUCT

FUNCTION loadVector(path) RETURNS (LIST<Course> courses, LIST<STRING> errors)

courses <- []

errors <- []

OPEN file at path

IF open failed

ADD "Could not open file" TO errors

RETURN (courses, errors)

ENDIF

FOR EACH line IN file

line <- TRIM(line)

IF line = "" THEN CONTINUE

parts <- SPLIT line BY ","; TRIM each part

IF LENGTH(parts) < 2

ADD "Format error" TO errors

CONTINUE

ENDIF

c <- new Course

c.number <- parts[0]

c.name <- parts[1]

c.prereqs <- []

FOR i FROM 2 TO LENGTH(parts)-1

IF parts[i] <> "" THEN ADD parts[i] TO c.prereqs

END FOR

ADD c TO courses

END FOR

CLOSE file

RETURN (courses, errors)

END FUNCTION

FUNCTION findPrintVector(courses, target)

found <- FALSE

FOR EACH c IN courses

IF c.number == target

PRINT c.number + ": " + c.name

IF LENGTH(c.prereqs) == 0

PRINT "Prerequisites: none"

ELSE

PRINT "Prerequisites: " + JOIN c.prereqs WITH ", "

ENDIF

found <- TRUE

BREAK

ENDIF

END FOR

IF NOT found THEN PRINT "Course '" + target + "' not found."

END FUNCTION

**Hash Table Data Structure**

FUNCTION loadHash(path) RETURNS (MAP<STRING, Course> table, LIST<STRING> errors)

table <- {}

errors <- []

OPEN file at path

IF open failed

ADD "Could not open file" TO errors

RETURN (table, errors)

ENDIF

FOR EACH line IN file

line <- TRIM(line)

IF line = "" THEN CONTINUE

parts <- SPLIT line BY ","; TRIM each part

IF LENGTH(parts) < 2

ADD "Format error" TO errors

CONTINUE

ENDIF

c <- new Course

c.number <- parts[0]

c.name <- parts[1]

c.prereqs <- []

FOR i FROM 2 TO LENGTH(parts)-1

IF parts[i] <> "" THEN ADD parts[i] TO c.prereqs

END FOR

table[c.number] <- c

END FOR

CLOSE file

RETURN (table, errors)

END FUNCTION

FUNCTION findPrintHash(table, target)

IF target IN table

c <- table[target]

PRINT c.number + ": " + c.name

IF LENGTH(c.prereqs) == 0

PRINT "Prerequisites: none"

ELSE

PRINT "Prerequisites: " + JOIN c.prereqs WITH ", "

ENDIF

ELSE

PRINT "Course '" + target + "' not found."

ENDIF

END FUNCTION

**Tree Data Structure**

STRUCT Node

Course course

Node left

Node right

END STRUCT

FUNCTION bstInsert(root, course) RETURNS Node

IF root == NULL

n <- new Node

n.course <- course

n.left <- NULL

n.right <- NULL

RETURN n

IF course.number < root.course.number

root.left <- bstInsert(root.left, course)

ELSE IF course.number > root.course.number

root.right <- bstInsert(root.right, course)

ELSE

root.course <- course

RETURN root

END FUNCTION

FUNCTION loadBST(path) RETURNS (Node root, LIST<STRING> errors)

root <- NULL

errors <- []

OPEN file at path

IF open failed

ADD "Could not open file" TO errors

RETURN (root, errors)

ENDIF

FOR EACH line IN file

line <- TRIM(line)

IF line = "" THEN CONTINUE

parts <- SPLIT line BY ","; TRIM each part

IF LENGTH(parts) < 2

ADD "Format error" TO errors

CONTINUE

ENDIF

c <- new Course

c.number <- parts[0]

c.name <- parts[1]

c.prereqs <- []

FOR i FROM 2 TO LENGTH(parts)-1

IF parts[i] <> "" THEN ADD parts[i] TO c.prereqs

END FOR

root <- bstInsert(root, c)

END FOR

CLOSE file

RETURN (root, errors)

END FUNCTION

FUNCTION findPrintBST(root, target)

cur <- root

WHILE cur <> NULL

IF target == cur.course.number

PRINT cur.course.number + ": " + cur.course.name

IF LENGTH(cur.course.prereqs) == 0

PRINT "Prerequisites: none"

ELSE

PRINT "Prerequisites: " + JOIN cur.course.prereqs WITH ", "

ENDIF

RETURN

ELSE IF target < cur.course.number

cur <- cur.left

ELSE

cur <- cur.right

ENDIF

END WHILE

PRINT "Course '" + target + "' not found."

END FUNCTION

**Vector Data Structure**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Vector** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create course item** | 1 | n | n |
| **While prereq exists** | 1 | n | n |
| **Append prereq** | 1 | n | n |
| **Pushback course item** |  |  |  |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | 0(n) |

**Hash Table Data Structure**

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create hash table** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create course item** | 1 | n | n |
| **While prereq exists** | 1 | n | n |
| **Append prereq** | 1 | n | n |
| **Compute hash for key** | 1 | n | n |
| **Insert into table** | 1 | n | n |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | 0(n) |

**Tree Data Structure**

| **Binary Search Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create empty BST** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Create course item** | 1 | n | n |
| **While prereq exists** | 1 | n | n |
| **Append Prereq** | 1 | n | n |
| **Insert into BST** | log n | n | n log n |
| **Total Cost** | | | log n +4n + 1 |
| **Runtime** | | | 0(n log n) |

**Advantage Analysis**

Vector – It is simple to implement and debug but can be inefficient when inserting or deleting elements in the middle.

Hash – It is efficient for key-value storage, and space efficient, but elements are not stored in a specific order, which can slow down searching.

Binary Search Tree –Elements are stored in sorted order, are typically efficient and have dynamic sizing, but can face problems balancing.

I recommend a Vector here. It keeps the code straightforward and memory frugal, and scanning the entire list of courses to load them is O(n). The size of the project (a file for a single course) is small enough that the O(n) scan to print one course is acceptable, and printing the whole list is just an O(n log n) sort. If quicker lookup is required, we may sort the vector once during load and perform binary search (O(log n)) without incurring data structure modifications.