**CS300 Project One**

**Kevin Jordan**

**Vector Pseudo Code**

FUNCTION importCourseData(fileName)

CREATE courseVector ← new Vector

OPEN file at fileName

IF file doesn't open THEN

PRINT "Couldn't open file"

RETURN

END IF

REPEAT

READ entry FROM file

IF entry IS NULL THEN

BREAK

END IF

SPLIT entry using commas → lineParts

IF count of lineParts < 2 THEN

PRINT "Line doesn't have enough info"

RETURN

END IF

SET id ← lineParts[0]

SET name ← lineParts[1]

CREATE prereqList ← empty List

// if there are prereqs, collect them

FOR i FROM 2 TO length of lineParts DO

ADD lineParts[i] TO prereqList

END FOR

// make a new course object

CREATE c ← new Course

SET c.id ← id

SET c.title ← name

SET c.prereqs ← prereqList

ADD c TO courseVector

UNTIL end of file

CLOSE file

// double-check that all prereqs actually exist in the list

FOR each course IN courseVector DO

FOR each pre IN course.prereqs DO

SET valid ← false

// yeah we loop again just to be sure

FOR each courseCheck IN courseVector DO

IF courseCheck.id = pre THEN

SET valid ← true

BREAK

END IF

END FOR

IF valid = false THEN

PRINT "Missing prereq → " + pre

RETURN

END IF

END FOR

END FOR

RETURN courseVector

END FUNCTION

CLASS Course

STRING id

STRING title

LIST prereqs

END CLASS

FUNCTION showCourseDetails(courseVector, searchCode)

SET matchFound ← false

FOR each item IN courseVector DO

IF item.id = searchCode THEN

PRINT "Course ID: " + item.id

PRINT "Course Title: " + item.title

IF item.prereqs IS EMPTY THEN

PRINT "No prerequisites"

ELSE

PRINT "Prerequisites:"

FOR each preq IN item.prereqs DO

PRINT "- " + preq

END FOR

END IF

SET matchFound ← true

BREAK

END IF

END FOR

IF matchFound = false THEN

PRINT "No course found"

END IF

END FUNCTION

**Hash Table Pseudocode**

4-3 Pseudo Code

Courses

// Open and check the course file

FUNCTION openCourseFile(fileName)

TRY to open file with fileName

IF file can't be opened THEN

PRINT "Error opening file"

RETURN

END IF

FOR EACH line in file DO

SPLIT line by commas

SET courseNumber = first value

SET courseTitle = second value

SET prereqList = remaining values

IF courseNumber or courseTitle is missing THEN

PRINT "Missing course info"

CONTINUE to next line

END IF

FOR EACH prereq in prereqList DO

IF prereq does not appear as a course in the file THEN

PRINT "Warning: " + prereq + " not found"

END IF

END FOR

END FOR

CLOSE file

END FUNCTION

// Load course data and store it in a hash table

FUNCTION loadCoursesToHashTable(fileName, hashTable)

OPEN file with fileName

FOR EACH line in file DO

SPLIT line by commas

SET courseNumber = first value

SET courseTitle = second value

SET prereqs = remaining values

CREATE Course object

SET Course.courseNumber = courseNumber

SET Course.title = courseTitle

FOR EACH prereq in prereqs DO

ADD prereq TO Course.prerequisites

END FOR

SET hashKey = calculateHash(courseNumber)

ADD Course to hashTable[hashKey]

END FOR

CLOSE file

END FUNCTION

// Print all course info from the hash table

FUNCTION printCourseInfo(hashTable)

FOR EACH bucket IN hashTable DO

SET current = start of bucket list

WHILE current IS NOT NULL DO

PRINT current.courseNumber + ": " + current.title

IF current has prerequisites THEN

PRINT "Prerequisites: "

FOR EACH prereq IN current.prerequisites DO

PRINT prereq

END FOR

END IF

SET current = next course in list

END WHILE

END FOR

END FUNCTION

**Binary Search Tree Pseudocode**

Week 5 Pseudo Code

Courses

// Open and validate the course file

FUNCTION openCourseFile(fileName)

TRY to open file with fileName

IF file can't be opened THEN

PRINT "Error opening file"

RETURN

END IF

FOR EACH line IN file DO

SPLIT line by commas

SET courseNumber ← first value

SET courseTitle ← second value

SET prereqList ← remaining values

IF courseNumber OR courseTitle is missing THEN

PRINT "Missing course info"

CONTINUE to next line

END IF

FOR EACH prereq IN prereqList DO

IF prereq does not appear as a course in the file THEN

PRINT "Warning: " + prereq + " not found"

END IF

END FOR

END FOR

CLOSE file

END FUNCTION

// Load course data into a binary search tree

FUNCTION loadCoursesToTree(fileName, tree)

OPEN file with fileName

FOR EACH line IN file DO

SPLIT line by commas

SET courseNumber ← first value

SET courseTitle ← second value

SET prereqs ← remaining values

CREATE Course object

SET Course.courseNumber ← courseNumber

SET Course.title ← courseTitle

FOR EACH prereq IN prereqs DO

ADD prereq TO Course.prerequisites

END FOR

INSERT Course INTO tree

END FOR

CLOSE file

END FUNCTION

// Print all course info from the binary search tree in order

FUNCTION printCourseInfo(tree)

CALL inOrderTraversal(tree.root)

END FUNCTION

FUNCTION inOrderTraversal(node)

IF node IS NULL THEN

RETURN

END IF

CALL inOrderTraversal(node.left)

PRINT node.course.courseNumber + ": " + node.course.title

IF node.course.prerequisites IS NOT EMPTY THEN

PRINT "Prerequisites: "

FOR EACH prereq IN node.course.prerequisites DO

PRINT prereq

END FOR

END IF

CALL inOrderTraversal(node.right)

END FUNCTION

**Menu Pseudocode**

FUNCTION displayMenu()

DO

PRINT "1. Load course data"

PRINT "2. Print course list"

PRINT "3. Search for course"

PRINT "9. Exit"

GET userInput

SWITCH userInput

CASE 1:

CALL loadCourses(fileName)

BREAK

CASE 2:

CALL printAllCourses()

BREAK

CASE 3:

PRINT "Enter course ID:"

GET searchID

CALL showCourseDetails(searchID)

BREAK

CASE 9:

PRINT "Exiting program."

BREAK

DEFAULT:

PRINT "Invalid selection"

END SWITCH

WHILE userInput ≠ 9

END FUNCTION

**Sort and Print**

**Vector**

FUNCTION printAllCourses()

SORT courseVector BY course.id

FOR EACH course IN courseVector DO

PRINT course.id + ": " + course.title

END FOR

END FUNCTION

**Hash Table**

FUNCTION printAllCourses()

CREATE courseList ← empty List

FOR EACH bucket IN hashTable DO

SET current ← head of bucket

WHILE current IS NOT NULL DO

ADD current TO courseList

SET current ← current.next

END WHILE

END FOR

SORT courseList BY course.id

FOR EACH course IN courseList DO

PRINT course.courseNumber + ": " + course.title

END FOR

END FUNCTION

**Binary Search Tree**

FUNCTION printSortedCourses(tree)

CALL inOrderTraversal(tree.root)

END FUNCTION

**Runtime Analysis**

| **Data Structure** | **Step** | **Cost per Line** | **Number of Executions** | **Total Cost** | **Big O** |
| --- | --- | --- | --- | --- | --- |
| **Vector** | Open and read file | 1 | *n* | *n* | O(n) |
|  | Parse and split line | 1 | *n* | *n* |  |
|  | Create course object | 1 | *n* | *n* |  |
|  | Store in vector | 1 | *n* | *n* |  |
|  | Validate prerequisites (nested loop) | *n* | *n* | *n²* | **O(n²)** |
| **Hash Table** | Open and read file | 1 | *n* | *n* | O(n) |
|  | Parse and split line | 1 | *n* | *n* |  |
|  | Create course object | 1 | *n* | *n* |  |
|  | Hash and store course | 1 | *n* | *n* | **O(n)** |
| **BST** | Open and read file | 1 | *n* | *n* | O(n log n) |
|  | Parse and split line | 1 | *n* | *n* |  |
|  | Create course object | 1 | *n* | *n* |  |
|  | Insert into tree | log(n) | *n* | *n*log(n)\* | **O(n log n)** |

**Vector Worst Case**

| **Pseudocode Line** | **Cost** | **Executions (Worst-Case)** | **Total Cost Expression** |
| --- | --- | --- | --- |
| Open file, check if it opens | 1 | 1 | 1 |
| Loop through each line in file | 1 | n | n |
| Split line by commas | 1 | n | n |
| Validate format | 1 | n | n |
| Create course object, set fields | 1 | n | n |
| Add course to vector | 1 | n | n |
| Nested loop to validate prerequisites | n | n | n² |
| **Total Big O** |  |  | **O(n²)** |

**Hash Table Worst Case**

| **Pseudocode Line** | **Cost** | **Executions (Worst-Case)** | **Total Cost Expression** |
| --- | --- | --- | --- |
| Open file, check if it opens | 1 | 1 | 1 |
| Loop through each line in file | 1 | n | n |
| Split line by commas | 1 | n | n |
| Validate format | 1 | n | n |
| Create course object, set fields | 1 | n | n |
| Calculate hash (assume cost is n worst-case, due to collision) | n | n | n² |
| Add course to hash table (insert in list at bucket) | 1 | n | n |
| Prereq validation requires scanning buckets (worst: n each) | n | n | n² |
| **Total Big O** |  |  | **O(n²)** |

**Binary Tree Worst Case**

| **Pseudocode Line** | **Cost** | **Executions (Worst-Case)** | **Total Cost Expression** |
| --- | --- | --- | --- |
| Open file, check if it opens | 1 | 1 | 1 |
| Loop through each line in file | 1 | n | n |
| Split line by commas | 1 | n | n |
| Validate format | 1 | n | n |
| Create course object, set fields | 1 | n | n |
| Insert course into unbalanced tree (degenerates to list) | n | n | n² |
| Validate prerequisites | n | n | n² |
| **Total Big O** |  |  | **O(n²)** |

**Evaluation**

Vector. Some advantages are that it is very simple, and easy to do. Works well without too much data and has direct access. The main problem is that validating the prerequisites means it has to go over the entire list each time, which is extremely inefficient and more so the larger the dataset becomes.

Hash Table. Advantages include, and mainly, the fast insert and lookup times . O(1). You can check for courses and prerequisites using a hash key. This makes it great for direct retrieval. Disadvantages include insertion time becoming worse if there are a lot of collisions. It is harder to implement. It is possible for the worst case to degrade to O(n^2) but unlikely.

Binary Search tree. Some advantages include it maintaining order already via in-order traversal. If balanced you get O(log n) insertion which can be efficient. This makes it fast for looking things up and sorting. Some disadvantages include the insertion time becoming worse if unbalanced. Managing it is a more complex endeavor than vectors or hash tables.

Suggestion. After comparing the 3 I would suggest the hash table. Mainly because it has the best overall performance when you search and store course information. This most closely aligns with the needs of the advisor. With constant times O(1) for lookup and retrieval it is the best choice. It is possible to degrade further but unlikely.

Vectors are easy to implement, but the nested loops can make them spiral into inefficiency quickly. The binary search tree has the potential to be slower when unbalanced. I believe the hash table strikes the best balance between practicality and efficiency and provides the best average times out of the options.