**Evaluation**

**Vector**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **FOR EACH line IN file:** | 1 | n | n |
| **SPLIT line BY COMMA INTO courseData** | 1 | n | n |
| **FOR EACH courseData IN coursesParsed** | 1 | n | n |
| **FOR i FROM 2 TO LENGTH OF courseData - 1** | 1 | n | n |
| **INSERT course OBJECT INTO coursesVector** | n | n | n |
| **Total Cost** | | | 3n + 1 |
| **Runtime** | | | O(n) |

**Hash Table**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **FOR EACH line IN file:** | 1 | n | n |
| **SPLIT line BY COMMA INTO courseData** | 1 | n | n |
| **FOR EACH courseData IN coursesParsed** | 1 | n | n |
| **FOR i FROM 2 TO LENGTH OF courseData - 1** | 1 | n | n |
| **INSERT course OBJECT INTO hashTable** | n | n | n |
| **Total Cost** | | | 3n + 1 |
| **Runtime** | | | O(n) |

**Tree**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **FOR EACH line IN file:** | 1 | n | n |
| **SPLIT line BY COMMA INTO courseData** | 1 | n | n |
| **FOR EACH courseData IN coursesParsed** | 1 | n | n |
| **FOR i FROM 2 TO LENGTH OF courseData - 1** | 1 | n | n |
| **INSERT course OBJECT INTO treeDataStructure** | n | n | n |
| **Total Cost** | | | 3n + 1 |
| **Runtime** | | | O(n) |

**Recommendation**

The hash table offers constant-time average-case lookup, making it ideal for quick access to course information. It efficiently handles large datasets and insertion/deletion operations, although it may require additional steps for sorting. Trees provide efficient searching and sorting but are more complex to implement and can suffer performance loss due to unbalanced structures. Vectors are simple but less efficient for searching and sorting, making them less suitable for this scenario. Therefore, I recommend using a hash table because it is simple and efficient in meeting the requirements of printing courses and retrieving course information.

**Vector Pseudocode**

FUNCTION main():

DECLARE coursesVector AS Vector<Course>

CALL loadDataStructure("courseInformation.csv", coursesVector)

REPEAT:

PRINT "Menu:"

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "4. Exit"

INPUT choice

SWITCH choice:

CASE 1:

CALL loadDataStructure("courseInformation.csv", coursesVector)

CASE 2:

CALL printSortedCourseList(coursesVector)

CASE 3:

PRINT "Enter course number:"

INPUT courseNumber

CALL printCourse(coursesVector, courseNumber)

CASE 4:

PRINT "Exiting program"

EXIT

DEFAULT:

PRINT "Invalid choice. Please select again."

END SWITCH

UNTIL choice EQUALS 4

END FUNCTION

FUNCTION readFile():

TRY:

OPEN file "courseInformation.csv" FOR READING

FOR EACH line IN file:

IF line IS NOT EMPTY:

APPEND line TO fileData LIST

CLOSE file

RETURN true

EXCEPT IOError:

RETURN false

FUNCTION parseFile():

coursesParsed = EMPTY LIST

courseNumbers = EMPTY LIST

FOR EACH line IN fileData:

SPLIT line BY COMMA INTO courseData

IF LENGTH OF courseData < 2:

PRINT "Error: Insufficient parameters in line"

CONTINUE

APPEND courseData TO coursesParsed LIST

APPEND courseData[0] TO courseNumbers LIST

FOR EACH courseData IN coursesParsed

courseNumber = courseData[0]

courseTitle = courseData[1]

prerequisites = []

FOR i FROM 2 TO LENGTH OF courseData - 1:

IF courseData[i] NOT IN courseNumbers:

PRINT "Error: Prerequisite not found for course"

RETURN false

ELSE:

APPEND courseData[i] TO prerequisites

CREATE course OBJECT WITH courseNumber, courseTitle, AND prerequisites

INSERT course OBJECT INTO coursesVector

RETURN true

FUNCTION printSortedCourseList(coursesVector):

DECLARE sortedCourses AS Array

sortedCourses = SORT coursesVector BY courseNumber

FOR EACH course IN sortedCourses:

PRINT "Course Number:", course.courseNumber

PRINT "Title:", course.title

IF course.prerequisites NOT EMPTY:

PRINT "Prerequisites:", course.prerequisites

END IF

END FOR

END FUNCTION

FUNCTION printCourse(coursesVector, courseNumber):

DECLARE foundCourse AS Boolean

foundCourse = FALSE

FOR EACH course IN coursesVector:

IF course.courseNumber EQUALS courseNumber:

foundCourse = TRUE

PRINT "Course Number:", course.courseNumber

PRINT "Title:", course.title

IF course.prerequisites NOT EMPTY:

PRINT "Prerequisites:", course.prerequisites

ELSE:

PRINT "No prerequisites for this course"

END IF

BREAK

END IF

END FOR

IF NOT foundCourse:

PRINT "Error: Course not found"

END IF

END FUNCTION

**Hash Table Pseudocode**

FUNCTION readFile():

TRY:

OPEN file "courseInformation.csv" FOR READING

FOR EACH line IN file:

IF line IS NOT EMPTY:

APPEND line TO fileData LIST

CLOSE file

RETURN true

EXCEPT IOError:

RETURN false

FUNCTION parseFile():

coursesParsed = EMPTY LIST

courseNumbers = EMPTY LIST

FOR EACH line IN fileData:

SPLIT line BY COMMA INTO courseData

IF LENGTH OF courseData < 2:

PRINT "Error: Insufficient parameters in line"

CONTINUE

APPEND courseData TO coursesParsed LIST

APPEND courseData[0] TO courseNumbers LIST

FOR EACH courseData IN coursesParsed

courseNumber = courseData[0]

courseTitle = courseData[1]

prerequisites = []

FOR i FROM 2 TO LENGTH OF courseData - 1:

IF courseData[i] NOT IN courseNumbers:

PRINT "Error: Prerequisite not found for course"

RETURN false

ELSE:

APPEND courseData[i] TO prerequisites

CREATE course OBJECT WITH courseNumber, courseTitle, AND prerequisites

INSERT course OBJECT INTO hashTable

RETURN true

FUNCTION printCourseInfo(hashTable):

FOR EACH courseNumber IN hashTable:

PRINT "Course Number:", courseNumber

courseObject = GET hashTable VALUE AT courseNumber

PRINT "Course Title:", courseObject.title

IF courseObject.prerequisites IS NOT EMPTY:

PRINT "Prerequisites:"

FOR EACH prerequisite IN courseObject.prerequisites:

PRINT " ", prerequisite

END FOR

END IF

END FOR

FUNCTION printSortedCourseList(hashTable):

sortedCourseNumbers = SORT keys of hashTable ALPHABETICALLY

FOR EACH courseNumber IN sortedCourseNumbers:

courseObject = GET hashTable VALUE AT courseNumber

PRINT "Course Number:", courseNumber

PRINT "Course Title:", courseObject.title

IF courseObject.prerequisites IS NOT EMPTY:

PRINT "Prerequisites:"

FOR EACH prerequisite IN courseObject.prerequisites:

PRINT " ", prerequisite

END FOR

END IF

END FOR

FUNCTION printCourseInfoByNumber(hashTable, courseNumber):

IF courseNumber NOT IN hashTable:

PRINT "Course number not found."

RETURN

END IF

PRINT "Course Number:", courseNumber

courseObject = GET hashTable VALUE AT courseNumber

PRINT "Course Title:", courseObject.title

IF courseObject.prerequisites IS NOT EMPTY:

PRINT "Prerequisites:"

FOR EACH prerequisite IN courseObject.prerequisites:

PRINT " ", prerequisite

END FOR

END IF

FUNCTION displayMenu():

PRINT "Menu:"

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "4. Exit"

FUNCTION main():

dataLoaded = FALSE

hashTable = EMPTY

REPEAT:

displayMenu()

choice = GET user input

IF choice == "1":

filePath = GET file path from user

hashTable = readCourseData(filePath)

IF hashTable IS NOT NULL:

dataLoaded = TRUE

PRINT "Data loaded successfully."

ELSE:

dataLoaded = FALSE

ELSE IF choice == "2":

IF NOT dataLoaded:

PRINT "Data structure not loaded. Please load data first."

CONTINUE TO NEXT ITERATION

PRINT "Sorted Course List:"

printSortedCourseList(hashTable)

ELSE IF choice == "3":

IF NOT dataLoaded:

PRINT "Data structure not loaded. Please load data first."

CONTINUE TO NEXT ITERATION

courseNumber = GET course number from user

printCourseInfoByNumber(hashTable, courseNumber)

ELSE IF choice == "4":

PRINT "Exiting program."

BREAK FROM LOOP

ELSE:

PRINT "Invalid choice. Please select a valid option."

END REPEAT

**Tree Pseudocode**

FUNCTION main():

INITIALIZE treeDataStructure AS EMPTY TREE

MENU()

FUNCTION MENU():

WHILE true:

PRINT "Menu:"

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "4. Exit"

INPUT choice

SWITCH choice:

CASE "1":

IF readFile():

PRINT "File read successfully"

ELSE:

PRINT "Error reading file"

BREAK

CASE "2":

IF isEmpty(treeDataStructure):

PRINT "Data structure is empty. Please load data first."

ELSE:

SORT treeDataStructure BY courseNumber

PRINT "Printing alphanumerically ordered list of all courses in Computer Science department:"

printTree(treeDataStructure)

BREAK

CASE "3":

IF isEmpty(treeDataStructure):

PRINT "Data structure is empty. Please load data first."

ELSE:

PRINT "Enter course number:"

INPUT courseNumber

printCourse(treeDataStructure, courseNumber)

BREAK

CASE "4":

PRINT "Exiting program..."

RETURN

DEFAULT:

PRINT "Invalid choice. Please choose a valid option."

FUNCTION readFile():

TRY:

OPEN file "courseInformation.csv" FOR READING

FOR EACH line IN file:

IF line IS NOT EMPTY:

APPEND line TO fileData LIST

CLOSE file

RETURN true

EXCEPT IOError:

RETURN false

FUNCTION parseFile():

coursesParsed = EMPTY LIST

courseNumbers = EMPTY LIST

FOR EACH line IN fileData:

SPLIT line BY COMMA INTO courseData

IF LENGTH OF courseData < 2:

PRINT "Error: Insufficient parameters in line"

CONTINUE

APPEND courseData TO coursesParsed LIST

APPEND courseData[0] TO courseNumbers LIST

FOR EACH courseData IN coursesParsed

courseNumber = courseData[0]

courseTitle = courseData[1]

prerequisites = []

FOR i FROM 2 TO LENGTH OF courseData - 1:

IF courseData[i] NOT IN courseNumbers:

PRINT "Error: Prerequisite not found for course"

RETURN false

ELSE:

APPEND courseData[i] TO prerequisites

CREATE course OBJECT WITH courseNumber, courseTitle, AND prerequisites

INSERT course OBJECT INTO treeDataStructure

RETURN true

FUCNTION nodeConstructor(value):

CREATE a new node with data to value

SET left child to NULL

SET right child to NULL

RETURN the new node

FUNCTION insert(root, value):

IF root IS NULL:

SET root TO nodeConstructor(value)

RETURN root

IF value < root.data:

SET root.left TO insert(root.left, value)

ELSE IF value > root.data:

SET root.right TO insert(root.right, value)

RETURN root

FUNCTION printTree(tree):

FOR EACH node IN tree:

PRINT node.courseNumber, node.courseTitle

IF node HAS prerequisites:

PRINT "Prerequisites:"

FOR EACH prerequisite IN node.prerequisites:

PRINT prerequisite

ELSE:

PRINT "No prerequisites"

FUNCTION printCourse(tree, courseNumber):

course = findCourse(tree, courseNumber)

IF course IS NOT NULL:

PRINT course.courseTitle

IF course HAS prerequisites:

PRINT "Prerequisites:"

FOR EACH prerequisite IN course.prerequisites:

PRINT prerequisite

ELSE:

PRINT "Course not found."

FUNCTION findCourse(tree, courseNumber):

FOR EACH node IN tree:

IF node.courseNumber EQUALS courseNumber:

RETURN node

RETURN NULL