# CS 300 Pseudocode Document

**Vector Pseudocode**

Function LoadAndValidateCourseData(fileName)

SET UP an empty list named courseList

SET UP an empty dictionary named courseDict

TRY

OPEN file

IF the file is open THEN

READ the file line by line

FOR each line in the file

SPLIT the line into courseNumber, courseName, and prerequisites

IF the split parts are less than two THEN

PRINT "Format error in line:", line

ELSE

IF courseNumber is in courseDict THEN

PRINT "Duplicate courseNumber found:", courseNumber

ELSE

PLACE courseNumber in courseDict as a key

ENDIF

FOR each prerequisite in prerequisites

IF prerequisite is NOT in courseDict THEN

PRINT "Prerequisite", prerequisite, "not found for course", courseNumber

ENDIF

ENDFOR

ENDIF

ADD line to courseList

ENDFOR

ELSE

PRINT "Could not open file", fileName

ENDIF

CATCH error

PRINT "Problem reading the file"

CLOSE the file

ENDTRY

RETURN courseList

End Function

Function CreateCourseObjects(courseList)

SET UP a vector named courseVector

FOR each courseData in courseList

SPLIT courseData into courseNumber, courseName, and prerequisites

CREATE a Course object with courseNumber, courseName, and prerequisites

ADD the Course object to courseVector

ENDFOR

RETURN courseVector

End Function

Class Course

Property number

Property name

Property prerequisites

Constructor(number, name, prerequisites)

SET Course number to number

SET Course name to name

IF prerequisites list is not empty THEN

SET Course prerequisites to prerequisites

ENDIF

End Constructor

End Class

Function PrintCourseDetails(courseVector, searchNumber)

SET found to False

FOR each course in courseVector

IF course's number is equal to searchNumber THEN

PRINT "Course Number:", course.number

PRINT "Course Name:", course.name

IF course has prerequisites THEN

PRINT "Prerequisites for this course are:"

FOR each prerequisite in course.prerequisites

PRINT prerequisite

ENDFOR

ELSE

PRINT "There are no prerequisites for this course."

ENDIF

SET found to True

BREAK

ENDIF

ENDFOR

IF found is False THEN

PRINT "The course", searchNumber, "was not found."

ENDIF

End Function

**Hash Table Pseudocode**

Function LoadAndValidateCourseData(fileName)

SET UP an empty hash table named courses

TRY

OPEN file

IF file is open THEN

WHILE not end of file

READ line from file

SPLIT line into parts using comma as a separator

IF number of parts is less than 2 THEN

PRINT "Error in line format: ", line

GO TO the next line

ENDIF

SET courseNumber to parts[0]

SET courseName to parts[1]

IF there are more than 2 parts THEN

SET prerequisites from the rest of the parts

ELSE

SET prerequisites to an empty list

ENDIF

IF courseNumber is already in courses THEN

PRINT "Duplicate courseNumber found: ", courseNumber

ELSE

ADD courseNumber with courseName and prerequisites to courses

ENDIF

FOR each prerequisite in prerequisites

IF prerequisite is NOT already in courses THEN

PRINT "Prerequisite course not found: ", prerequisite

ENDIF

ENDFOR

ENDWHILE

ELSE

PRINT "File cannot be opened: ", fileName

ENDIF

CATCH error

PRINT "Problem reading the file.”

CLOSE the file

ENDTRY

RETURN courses

End Function

Function CreateCourseObjects(coursesHashTable)

SET UP an empty hash table named courseObjectsHashTable

FOR each courseNumber, courseInfo in coursesHashTable

// I am assuming that courseNumber is a string, courseName is a string, and prerequisites is a list of strings

CREATE a Course object with courseNumber, courseInfo.name, and courseInfo.prerequisites

ADD the Course object to courseObjectsHashTable with courseNumber as the key

ENDFOR

RETURN courseObjectsHashTable

End Function

Class Course

Property number //String

Property name //String

Property prerequisites //List of Strings

Constructor(number String, name String, prerequisites List of Strings)

SET this.number to number

SET this.name to name

SET this.prerequisites to prerequisites IF NOT empty, IF its empty set to a new empty List of Strings

End Constructor

End Class

Function PrintCourseDetails(courseObjectsHashTable, searchNumber as String)

IF searchNumber is in courseObjectsHashTable THEN

SET course to courseObjectsHashTable[searchNumber]

PRINT "Course Number: " + course.number

PRINT "Course Name: " + course.name

IF course.prerequisites is not empty THEN

PRINT "Prerequisites for this course:"

FOR each prerequisite in course.prerequisites

PRINT " - " + prerequisite

ENDFOR

ELSE

PRINT "This course has no prerequisites."

ENDIF

ELSE

PRINT "Course " + searchNumber + " not found."

ENDIF

End Function

**Tree Pseudocode**

Function LoadAndValidateCourseData(fileName)

SET UP an empty list named coursesList

TRY

OPEN file

IF file is open THEN

WHILE not end of file

READ line from file

SPLIT line into parts using comma as a separator

IF number of parts is less than 2 THEN

PRINT "Error in line format: ", line

GO TO the next line

ENDIF

SET courseNumber to parts[0]

SET courseName to parts[1]

IF there are more than 2 parts THEN

SET prerequisites to parts from 3rd to the last

ELSE

SET prerequisites to an empty list

ENDIF

ADD a new entry to coursesList with courseNumber, courseName, and prerequisites

ENDWHILE

ELSE

PRINT "File cannot be opened: ", fileName

ENDIF

CATCH error

PRINT "Problem reading the file."

CLOSE the file

ENDTRY

RETURN coursesList

End Function

Function BuildCourseTree(coursesList)

SET UP an empty tree named coursesTree

FOR each course in coursesList

IF course.number is NOT already a node in coursesTree THEN

ADD course.number as a new node in coursesTree with course.name and course.prerequisites

ENDIF

FOR each prerequisite in course.prerequisites

IF prerequisite is NOT already a node in coursesTree THEN

ADD prerequisite as a new node in coursesTree

ENDIF

LINK prerequisite node as a child of course.number node in coursesTree

ENDFOR

ENDFOR

RETURN coursesTree

End Function  
  
Class CourseNode

Property number // String

Property name // String

Property prerequisites // List of CourseNode references

Property children // List of CourseNode references (for prerequisites)

Constructor(number String, name String, prerequisites List of Strings)

SET this.number to number

SET this.name to name

SET this.prerequisites to prerequisites

SET this.children to a new empty List of CourseNode references

End Constructor

Method AddChild(child CourseNode)

ADD child to this.children

End Method

End Class

Function PrintCourseTree(node CourseNode)

IF node is NULL THEN

RETURN

ENDIF

PRINT “Course Number: “, node.number

PRINT “Course Name: “, node.name

IF node.prerequisites is NOT empty THEN

PRINT "Prerequisites:"

FOR each prerequisite in node.prerequisites

PRINT “ – “, prerequisite.number, ": ", prerequisite.name

ENDFOR

ELSE

PRINT "No prerequisites"

ENDIF

FOR each child in node.children

CALL PrintCourseTree(child)

ENDFOR

End Function

**Menu Pseudocode**

Function MainMenu()

SET dataLoaded to FALSE

SET coursesList to an empty list

SET coursesTree to NULL

WHILE TRUE

PRINT "1. Load Data Structure"

PRINT "2. Print Course List"

PRINT "3. Print Course"

PRINT "4. Exit"

PRINT "Enter your choice: "

READ choice

SWITCH choice

CASE 1:

IF dataLoaded THEN

PRINT "Data already loaded."

ELSE

SET fileName to .csv file

SET coursesList to CALL LoadAndValidateCourseData(fileName)

IF coursesList is NOT empty THEN

SET dataLoaded to TRUE

SET coursesTree to CALL BuildCourseTree(coursesList)

PRINT "Data loaded successfully."

ELSE

PRINT "Failed to load data."

ENDIF

ENDIF

BREAK

CASE 2:

IF NOT dataLoaded THEN

PRINT "Please load data structure first."

ELSE

CALL PrintSortedCourses(coursesList)

ENDIF

BREAK

CASE 3:

IF NOT dataLoaded THEN

PRINT "Please load data structure first."

ELSE

PRINT "Enter course number: "

READ courseNumber

SET courseNode to CALL FindCourseNode(coursesTree, courseNumber)

IF courseNode is NOT NULL THEN

CALL PrintCourseDetails(courseNode)

ELSE

PRINT "Course not found."

ENDIF

ENDIF

BREAK

CASE 4:

PRINT "Exiting program..."

EXIT PROGRAM

DEFAULT:

PRINT "Invalid choice, try again."

END SWITCH

END WHILE

End Function

// Print Sorted Courses Function

Function PrintSortedCourses(coursesList)

SET sortedList to CALL SortCoursesAlphanumerically(coursesList)

FOR each course in sortedList

PRINT course.number, ": ", course.name

ENDFOR

End Function

// Print Course Details Function

Function PrintCourseDetails(courseNode)

PRINT "Course Number: ", courseNode.number

PRINT "Course Name: ", courseNode.name

IF courseNode.prerequisites is NOT empty THEN

PRINT "Prerequisites:"

FOR each prerequisite in courseNode.prerequisites

PRINT " - ", prerequisite.number, ": ", prerequisite.name

ENDFOR

ELSE

PRINT "No prerequisites"

ENDIF

End Function

// Find Course Node Function

Function FindCourseNode(coursesTree, courseNumber)

// Implement a search function that traverses the coursesTree

// If not found, return NULL.

End Function

// Sort Courses Alphanumerically Function

Function SortCoursesAlphanumerically(coursesList)

// Implement sorting algorithm to sort coursesList

End Function

**Print in Alphanumeric Order Pseudocode**

Function SortAndPrintCourses(coursesList)

CALL AlphanumericSort(coursesList)

// Print the sorted list of courses

FOR each course in coursesList

PRINT course.number, ": ", course.name

ENDFOR

End Function

Function AlphanumericSort(coursesList)

SET n to length of coursesList

FOR i from 0 to n-1

FOR j from 0 to n-i-2

// Compare course numbers alphanumerically

IF coursesList[j].number > coursesList[j+1].number THEN

// Swap coursesList[j] and coursesList[j+1]

SET temp to coursesList[j]

SET coursesList[j] to coursesList[j+1]

SET coursesList[j+1] to temp

ENDIF

ENDFOR

ENDFOR

End Function

Function PrintSortedCoursesFromHashTable(coursesHashTable)

SET coursesList to an empty list

FOR each courseNumber, courseInfo in coursesHashTable

ADD {courseNumber, courseInfo.name} to coursesList

ENDFOR

// Sort the coursesList alphanumerically by course number

CALL AlphanumericSort(coursesList)

Print the sorted list

FOR each course in coursesList

PRINT course.courseNumber, ": ", course.name

ENDFOR

End Function

Function AlphanumericSort(coursesList)

SET n to length of coursesList

FOR i from 0 to n-1

FOR j from 0 to n-i-2

IF coursesList[j].courseNumber > coursesList[j+1].courseNumber THEN

// Swap coursesList[j] and coursesList[j+1]

SET temp to coursesList[j]

SET coursesList[j] to coursesList[j+1]

SET coursesList[j+1] to temp

ENDIF

ENDFOR

ENDFOR

End Function

Function PrintSortedCourseFromBinaryTree(root)

CALL InOrder(root)

End Function

Function InOrder(node)

IF node is NULL THEN

RETURN

ENDIF

CALL InOrder(node.leftChild)

PRINT "Course Number: ", node.number

PRINT "Course Name: ", node.name

PRINT "Prerequisites: "

IF node.prerequisites is NOT empty THEN

FOR each prerequisite in node.prerequisites

PRINT " - ", prerequisite

ENDFOR

ELSE

PRINT "No prerequisites"

ENDIF

CALL InOrder(node.rightChild)

End Function

Vector Analysis

| **Vector LoadAndValidateCourseData Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Init courseList and CourseDict** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **Read the file line by line** | 1 | n | n |
| **For each line in the file. (split, validate, check duplicates,add to list)** | 1 (for each action) | n | n |
| **Check prerequisites for each course** | 1 | n | n |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |

| **Vector CreateCourseObjects Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Init courseVector** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **For each courseData in courseList (split courseData, create Course object and add to courseVector)** | 1 | n (for each course) | 2n |
| **Total Cost** | | | 2n + 1 |
| **Runtime** | | | O(n) |

Hash Analysis

| **Hash LoadAndValidateCourseData Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize hash table** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **WHILE not end of file (Read line from file, Split line into parts):** | 1 | n | n |
| **Validate line format** | 1 | n | n |
| **Check for duplicate courseNumber** | 1 | n | n |
| **FOR each prerequisite in prerequisites** | 1 | np | np |
| **Total Cost** | | | 3n + np + 2 |
| **Runtime** | | | O(np) if p scales with n  O(n) if p is constant |

| **Hash CreateCourseObjects Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **FOR each courseNumber, courseInfo in coursesHashTable** | 1 | n | n |
| **Total Cost** | | | n + 1 |
| **Runtime** | | | O(n) |

Tree Analysis

| **Tree LoadAndValidateCourseData Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize coursesList** | 1 | 1 | 1 |
| **Open file** | 1 | 1 | 1 |
| **WHILE not end of file (Read line from file, Split line into parts)** | 1 | n | n |
| **IF there are more than 2 parts (set prerequisites)** | 1 | n | n |
| **ADD a new entry to coursesList** | 1 | n | n |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |

| **Tree BuildCourseTree Function** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize coursesTree** | 1 | 1 | 1 |
| **FOR each course in coursesList** | 1 | n | n |
| **FOR each prerequisite in course.prerequisites** | 1 | np | np |
| **Total Cost** | | | n + np + 1 |
| **Runtime** | | | O(n + np) |

**Recommendation**

When comparing the vector, hash table, and tree analysis there are many advantages and disadvantages with using each one. For example with the vector its advantage is that is simple and straightforward to use because it takes a linear approach to storing data. Also it is very cache friendly. However its disadvantages are that its not scalable and when searching for duplicates it has a cost of O(n) which is not very optimal.

The advantages of hash tables are that they are great at handling duplicates more effectively and they can provide average-case constant time when it comes to insertion, lookup, and deletion operations. The disadvantage is that they may consume more memory then vectors due to the storage of things such as keys, values, and the table itself.

The advantages of trees are that they give a nice balanced search and are sutable for representing hierarchical data, however they are very complex to actually implement and manage then that of vectors and hash tables and can be unbalanced.

Based on the Big O results and the analysis I would recommend that we use the hash table for managing course data structure. This recommendation is based on the average-case time and the hash tables overall scalability. As the table grows the hash table would be the better option as it will perform better in terms of complexity when compared to that of vectors and trees. Also the hash table with its key-value relationships will handle prerequisite checking efficiently.