**Vector Pseudocode**

//Data structure to store course information

STRUCT Course

STRING courseNumber

STRING courseName

VECTOR of STRING prerequisites

END STRUCT

--Existing functions

// Function to load courses from a file into a vector

FUNCTION LoadCoursesFromFile(filePath)

INITIALIZE empty VECTOR of Course called courseList

OPEN course file at filePath FOR reading

FOR EACH LINE in course file DO

SPLIT line by commas INTO tokens

IF number of tokens < 2 THEN

DISPLAY "Format error: Each line must have at least a course number and course name."

CONTINUE to next line

ENDIF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

INITIALIZE empty VECTOR of STRING prereqs

FOR i FROM 2 TO LENGTH(tokens) - 1 DO

ADD tokens[i] TO prereqs

ENDFOR

CREATE Course object newCourse

SET newCourse.courseNumber = courseNumber

SET newCourse.courseName = courseName

SET newCourse.prerequisites = prereqs

ADD newCourse TO courseList

ENDFOR

CLOSE course file

// Validate that all prerequisites exist in the file

FOR EACH Course IN courseList DO

FOR EACH prereq IN Course.prerequisites DO

SET found = FALSE

FOR EACH c IN courseList DO

IF c.courseNumber == prereq THEN

SET found = TRUE

BREAK

ENDIF

ENDFOR

IF found == FALSE THEN

DISPLAY "Data error: Prerequisite " + prereq + " for course " + Course.courseNumber + " not found in file."

ENDIF

ENDFOR

ENDFOR

RETURN courseList

END FUNCTION

// Function to print all courses and their prerequisites

FUNCTION PrintAllCourses(courseList)

FOR EACH Course IN courseList DO

DISPLAY Course.courseNumber + ": " + Course.courseName

IF Course.prerequisites is not empty THEN

DISPLAY " Prerequisites:"

FOR EACH prereq IN Course.prerequisites DO

DISPLAY " " + prereq

ENDFOR

ELSE

DISPLAY " Prerequisites: None"

ENDIF

ENDFOR

END FUNCTION

// Function to search for and print a specific course and its prerequisites

FUNCTION PrintCourseInfo(courseList, searchNumber)

SET found = FALSE

FOR EACH Course IN courseList DO

IF Course.courseNumber == searchNumber THEN

DISPLAY Course.courseNumber + ": " + Course.courseName

IF Course.prerequisites is not empty THEN

DISPLAY " Prerequisites:"

FOR EACH prereq IN Course.prerequisites DO

DISPLAY " " + prereq

ENDFOR

ELSE

DISPLAY " Prerequisites: None"

ENDIF

SET found = TRUE

BREAK

ENDIF

ENDFOR

IF found == FALSE THEN

DISPLAY "Course not found: " + searchNumber

ENDIF

END FUNCTION

//Additions required by final project

// Helper to split an alphanumeric course number into a sortable key

FUNCTION AlphanumericKey(courseNumber)

INITIALIZE prefix = ""

INITIALIZE numberPart = ""

FOR EACH char IN courseNumber DO

IF char is alphabetic THEN

APPEND char TO prefix

ELSE IF char is digit THEN

APPEND char TO numberPart

ENDIF

ENDFOR

IF numberPart == "" THEN

SET numeric = 0

ELSE

SET numeric = INTEGER(numberPart)

ENDIF

RETURN (prefix, numeric)

END FUNCTION

// Creates a sorted copy of the course list by alphanumeric course number

FUNCTION GetSortedCoursesByNumber(courseList)

CREATE new VECTOR of Course called sortedList

FOR EACH c IN courseList DO

ADD c TO sortedList

ENDFOR

// Comparison sort using the AlphanumericKey comparator

FOR i FROM 0 TO LENGTH(sortedList) - 2 DO

FOR j FROM i + 1 TO LENGTH(sortedList) - 1 DO

SET keyI = AlphanumericKey(sortedList[i].courseNumber)

SET keyJ = AlphanumericKey(sortedList[j].courseNumber)

// Compare by prefix first, then by numeric value

IF keyJ.prefix < keyI.prefix OR

(keyJ.prefix == keyI.prefix AND keyJ.numeric < keyI.numeric) THEN

SWAP sortedList[i] WITH sortedList[j]

ENDIF

ENDFOR

ENDFOR

RETURN sortedList

END FUNCTION

// Print an alphanumerically ordered list of all CS courses

FUNCTION PrintCourseListSorted(courseList)

IF courseList is empty THEN

DISPLAY "No courses loaded."

RETURN

ENDIF

SET sortedList = GetSortedCoursesByNumber(courseList)

DISPLAY "Computer Science Courses (Alphanumeric Order):"

FOR EACH c IN sortedList DO

DISPLAY c.courseNumber + ": " + c.courseName

ENDFOR

END FUNCTION

// Main program with menu

START PROGRAM

INITIALIZE courseList as empty VECTOR of Course

SET dataLoaded = FALSE

WHILE TRUE DO

DISPLAY ""

DISPLAY "====== ABCU Advising Menu (Vector) ======"

DISPLAY "1. Load course data from file"

DISPLAY "2. Print all courses (alphanumeric order)"

DISPLAY "3. Print a course's title and prerequisites"

DISPLAY "9. Exit"

DISPLAY "Enter option: "

ACCEPT user input AS option

IF option == 1 THEN

DISPLAY "Enter file path (e.g., courses.txt): "

ACCEPT user input AS filePath

SET courseList = LoadCoursesFromFile(filePath)

SET dataLoaded = TRUE

DISPLAY "Course data loaded."

ELSE IF option == 2 THEN

IF dataLoaded == FALSE THEN

DISPLAY "Please load course data first using option 1."

ELSE

CALL PrintCourseListSorted(courseList)

ENDIF

ELSE IF option == 3 THEN

IF dataLoaded == FALSE THEN

DISPLAY "Please load course data first using option 1."

ELSE

DISPLAY "Enter course number to lookup (e.g., CS200): "

ACCEPT user input AS searchNumber

CALL PrintCourseInfo(courseList, searchNumber)

ENDIF

ELSE IF option == 9 THEN

DISPLAY "Exiting program."

BREAK

ELSE

DISPLAY "Invalid option. Please try again."

ENDIF

ENDWHILE

END PROGRAM

**Hash Table Pseudocode**

// Data structure to store course information

STRUCT Course

STRING courseNumber

STRING courseName

LIST of STRING prerequisites

END STRUCT

//Storage

// Initialize empty HASH TABLE to store courses (key: courseNumber, value: Course object)

INITIALIZE hashTable as empty HASH TABLE (STRING, Course)

// Initialize empty LIST of STRING allCourses for prerequisite validation

INITIALIZE allCourses as empty LIST of STRING

// Existing functions

// Function to load courses from a file into a hash table

FUNCTION LoadCoursesFromFile(filePath)

OPEN course file at filePath FOR reading

FOR EACH line in course file DO

SPLIT line by commas INTO tokens

IF number of tokens < 2 THEN

DISPLAY "Format error: Each line must have at least a course number and course name."

CONTINUE to next line

ENDIF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

INITIALIZE prereqList as empty LIST of STRING

FOR i FROM 2 TO LENGTH(tokens) - 1 DO

ADD tokens[i] TO prereqList

ENDFOR

CREATE Course object newCourse

SET newCourse.courseNumber = courseNumber

SET newCourse.courseName = courseName

SET newCourse.prerequisites = prereqList

INSERT newCourse INTO hashTable WITH KEY = courseNumber

ADD courseNumber TO allCourses

ENDFOR

CLOSE course file

// Validate that all prerequisites exist

FOR EACH course IN hashTable DO

FOR EACH prereq IN course.prerequisites DO

IF prereq NOT IN allCourses THEN

DISPLAY "Data error: Prerequisite " + prereq + " for course " + course.courseNumber + " not found in file."

ENDIF

ENDFOR

ENDFOR

END FUNCTION

FUNCTION PrintAllCourses()

FOR EACH course IN hashTable DO

DISPLAY course.courseNumber + ": " + course.courseName

IF course.prerequisites is not empty THEN

DISPLAY " Prerequisites:"

FOR EACH prereq IN course.prerequisites DO

DISPLAY " " + prereq

ENDFOR

ELSE

DISPLAY " Prerequisites: None"

ENDIF

ENDFOR

END FUNCTION

FUNCTION PrintCourseInfo(courseNumber)

IF courseNumber IN hashTable THEN

SET course = hashTable[courseNumber]

DISPLAY course.courseNumber + ": " + course.courseName

IF course.prerequisites is NOT EMPTY THEN

DISPLAY " Prerequisites:"

FOR EACH prereq IN course.prerequisites DO

DISPLAY " " + prereq

ENDFOR

ELSE

DISPLAY " Prerequisites: None"

ENDIF

ELSE

DISPLAY "Course not found: " + courseNumber

ENDIF

END FUNCTION

//Additions required by final project

// Helper to split an alphanumeric course number into a sortable key

FUNCTION AlphanumericKey(courseNumber)

INITIALIZE prefix = ""

INITIALIZE numberPart = ""

FOR EACH char IN courseNumber DO

IF char is alphabetic THEN

APPEND char TO prefix

ELSE IF char is digit THEN

APPEND char TO numberPart

ENDIF

ENDFOR

IF numberPart == "" THEN

SET numeric = 0

ELSE

SET numeric = INTEGER(numberPart)

ENDIF

RETURN (prefix, numeric)

END FUNCTION

// Create a sorted list of courses by alphanumeric course number

FUNCTION GetSortedCoursesByNumber()

CREATE new LIST of Course called sortedList

FOR EACH course IN hashTable DO

ADD course TO sortedList

ENDFOR

// Comparison sort using the AlphanumericKey comparator

FOR i FROM 0 TO LENGTH(sortedList) - 2 DO

FOR j FROM i + 1 TO LENGTH(sortedList) - 1 DO

SET keyI = AlphanumericKey(sortedList[i].courseNumber)

SET keyJ = AlphanumericKey(sortedList[j].courseNumber)

IF keyJ.prefix < keyI.prefix OR

(keyJ.prefix == keyI.prefix AND keyJ.numeric < keyI.numeric) THEN

SWAP sortedList[i] WITH sortedList[j]

ENDIF

ENDFOR

ENDFOR

RETURN sortedList

END FUNCTION

// Print all courses in alphanumeric order

FUNCTION PrintCourseListSorted()

IF hashTable is empty THEN

DISPLAY "No courses loaded."

RETURN

ENDIF

SET sortedList = GetSortedCoursesByNumber()

DISPLAY "Computer Science Courses (Alphanumeric Order):"

FOR EACH c IN sortedList DO

DISPLAY c.courseNumber + ": " + c.courseName

ENDFOR

END FUNCTION

//Main program with menu

START PROGRAM

SET dataLoaded = FALSE

WHILE TRUE DO

DISPLAY ""

DISPLAY "====== ABCU Advising Menu (Hash Table) ======"

DISPLAY "1. Load course data from file"

DISPLAY "2. Print all courses (alphanumeric order)"

DISPLAY "3. Print a course's title and prerequisites"

DISPLAY "9. Exit"

DISPLAY "Enter option: "

ACCEPT user input AS option

IF option == 1 THEN

DISPLAY "Enter file path (e.g., courses.txt): "

ACCEPT user input AS filePath

CALL LoadCoursesFromFile(filePath)

SET dataLoaded = TRUE

DISPLAY "Course data loaded."

ELSE IF option == 2 THEN

IF dataLoaded == FALSE THEN

DISPLAY "Please load course data first using option 1."

ELSE

CALL PrintCourseListSorted()

ENDIF

ELSE IF option == 3 THEN

IF dataLoaded == FALSE THEN

DISPLAY "Please load course data first using option 1."

ELSE

DISPLAY "Enter course number to lookup (e.g., CS200): "

ACCEPT user input AS searchNumber

CALL PrintCourseInfo(searchNumber)

ENDIF

ELSE IF option == 9 THEN

DISPLAY "Exiting program."

BREAK

ELSE

DISPLAY "Invalid option. Please try again."

ENDIF

ENDWHILE

END PROGRAM  
  
**BST Pseudocode**

//Data structures to store course information

STRUCT Course

STRING courseNumber

STRING courseName

LIST of STRING prerequisites

END STRUCT

// Structure for the BST node

STRUCT Node

Course course

Node\* left

Node\* right

END STRUCT

// Initialize empty BST root and a LIST to track all courseNumbers for

INITIALIZE root as NULL

INITIALIZE allCourses as empty LIST of STRING

// Existing functions

// Function to insert a course into the BST

FUNCTION InsertNode(root, newCourse)

IF root == NULL THEN

CREATE newNode as Node

SET newNode.course = newCourse

SET newNode.left = NULL

SET newNode.right = NULL

RETURN newNode

ENDIF

IF newCourse.courseNumber < root.course.courseNumber THEN

root.left = InsertNode(root.left, newCourse)

ELSE

root.right = InsertNode(root.right, newCourse)

ENDIF

RETURN root

END FUNCTION

// Function to load courses from a file into the BST

FUNCTION LoadCoursesFromFile(filePath)

// Reset current data

SET root = NULL

SET allCourses = empty LIST

OPEN course file at filePath FOR reading

WHILE not end of file DO

READ line from file

SPLIT line by commas INTO tokens

IF number of tokens < 2 THEN

DISPLAY "Format error: Each line must have at least a course number and course name."

CONTINUE to next line

ENDIF

SET courseNumber = tokens[0]

SET courseName = tokens[1]

INITIALIZE prereqList as empty LIST of STRING

FOR i FROM 2 TO LENGTH(tokens) - 1 DO

ADD tokens[i] TO prereqList

ENDFOR

CREATE Course object newCourse

SET newCourse.courseNumber = courseNumber

SET newCourse.courseName = courseName

SET newCourse.prerequisites = prereqList

root = InsertNode(root, newCourse)

ADD courseNumber TO allCourses

ENDWHILE

CLOSE course file

// Validate that all prerequisites exist in the file

CALL ValidatePrerequisites(root, allCourses)

RETURN root

END FUNCTION

// Function to validate that all prerequisites exist

FUNCTION ValidatePrerequisites(node, allCourses)

IF node == NULL THEN

RETURN

ENDIF

FOR EACH prereq IN node.course.prerequisites DO

IF prereq NOT IN allCourses THEN

DISPLAY "Data error: Prerequisite " + prereq + " for course " + node.course.courseNumber + " not found in file."

ENDIF

ENDFOR

CALL ValidatePrerequisites(node.left, allCourses)

CALL ValidatePrerequisites(node.right, allCourses)

END FUNCTION

// Function to print all courses in order (sorted by courseNumber)

FUNCTION PrintAllCourses(node)

IF node == NULL THEN

RETURN

ENDIF

CALL PrintAllCourses(node.left)

DISPLAY node.course.courseNumber + ": " + node.course.courseName

IF node.course.prerequisites is not empty THEN

DISPLAY " Prerequisites:"

FOR EACH prereq IN node.course.prerequisites DO

DISPLAY " " + prereq

ENDFOR

ELSE

DISPLAY " Prerequisites: None"

ENDIF

CALL PrintAllCourses(node.right)

END FUNCTION

// Function to search for and print a specific course by number

FUNCTION PrintCourseInfo(node, searchNumber)

IF node == NULL THEN

DISPLAY "Course not found: " + searchNumber

RETURN

ENDIF

IF searchNumber == node.course.courseNumber THEN

DISPLAY node.course.courseNumber + ": " + node.course.courseName

IF node.course.prerequisites is not empty THEN

DISPLAY " Prerequisites:"

FOR EACH prereq IN node.course.prerequisites DO

DISPLAY " " + prereq

ENDFOR

ELSE

DISPLAY " Prerequisites: None"

ENDIF

RETURN

ELSE IF searchNumber < node.course.courseNumber THEN

CALL PrintCourseInfo(node.left, searchNumber)

ELSE

CALL PrintCourseInfo(node.right, searchNumber)

ENDIF

END FUNCTION

//Main Program with menu

START PROGRAM

SET root = NULL

SET allCourses = empty LIST

SET dataLoaded = FALSE

WHILE TRUE DO

DISPLAY ""

DISPLAY "====== ABCU Advising Menu (BST) ======"

DISPLAY "1. Load course data from file"

DISPLAY "2. Print all courses (alphanumeric order via in-order traversal)"

DISPLAY "3. Print a course's title and prerequisites"

DISPLAY "9. Exit"

DISPLAY "Enter option: "

ACCEPT user input AS option

IF option == 1 THEN

DISPLAY "Enter file path (e.g., courses.txt): "

ACCEPT user input AS filePath

SET root = LoadCoursesFromFile(filePath)

SET dataLoaded = TRUE

DISPLAY "Course data loaded."

ELSE IF option == 2 THEN

IF dataLoaded == FALSE THEN

DISPLAY "Please load course data first using option 1."

ELSE

CALL PrintAllCourses(root)

ENDIF

ELSE IF option == 3 THEN

IF dataLoaded == FALSE THEN

DISPLAY "Please load course data first using option 1."

ELSE

DISPLAY "Enter course number to lookup (e.g., CS200): "

ACCEPT user input AS searchNumber

CALL PrintCourseInfo(root, searchNumber)

ENDIF

ELSE IF option == 9 THEN

DISPLAY "Exiting program."

BREAK

ELSE

DISPLAY "Invalid option. Please try again."

ENDIF

ENDWHILE

END PROGRAM

**For The Following Tables:**

Let:

* n = number of courses (lines in file)
* P = total number of prerequisite entries across the whole file. If each course has at most k prereqs, then P ≤ k·n

**Vector cost-per-line table**

|  |  |  |  |
| --- | --- | --- | --- |
| Step from pseudocode | Cost per execution | Executions | Total Cost |
| Initialize empty vector ‘courseList’ | 1 | 1 | 1 |
| Open File | 1 | 1 | 1 |
| For each line: split by commas into tokens | 1 | n | n |
| For each line check ‘tokens < 2’ | 1 | n | n |
| For each line set ‘courseNumber = tokens[0]’ | 1 | n | n |
| For each line set ‘courseName = tokens[1]’ | 1 | n | n |
| For each line init empty ‘prereqs’ vector | 1 | n | n |
| For each prerequisite token ‘ADD tokens[i] TO prereqs’ | 1 | P | P |
| For each line create ‘Course newCourse’ | 1 | n | n |
| For each line set ‘newCourse.courseNumber’ | 1 | n | n |
| For each line set ‘newCourse.courseName’ | 1 | n | n |
| For each line set ‘newCourse.prerequisites = prereqs’ | 1 | n | n |
| For each line append ‘newCourse to courseList’ | 1 | n | n |
| Close file | 1 | 1 | 1 |

Sum (load only)  
Total T(n, P) = 1 + 1 + [10·n] + P + 1 = 10n + P + 3

If prerequisites per course are bounded by a small constant k, then P ≤ k·n, so  
T(n, P) ≤ (10 + k)·n + 3 which simplifies to O(n).

Vector load phase worst case Big-O: **O(n)  
  
Hash Table cost-per-line table**

|  |  |  |  |
| --- | --- | --- | --- |
| Step from pseudocode | Cost per execution | Executions | Total Cost |
| Initialize hashTable (empty) | 1 | 1 | 1 |
| Initialize allCourses (empty) | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| For each line: split by commas into tokens | 1 | n | n |
| For each line ‘check tokens < 2’ | 1 | n | n |
| For each line set ‘courseNumber = tokens[0]’ | 1 | n | n |
| For each line set ‘courseName = tokens[1]’ | 1 | n | n |
| For each line init empty ‘prereqList’ | 1 | n | n |
| For each prereq token ‘ADD tokens[i] TO prereqList’ | 1 | P | P |
| For each line create ‘Course newCourse’ | 1 | n | n |
| For each line set ‘newCourse.courseNumber’ | 1 | n | n |
| For each line set ‘newCourse.courseName’ | 1 | n | n |
| For each line set ‘newCourse.prerequisites = prereqList’ | 1 | n | n |
| Insert ‘newCourse’ into ‘hashTable’ | InsertHT | n | Σ InsertHT(i) |
| For each line ‘ADD courseNumber TO allCourses’ | 1 | n | n |
| Close file | 1 | 1 | 1 |

Sum (load only)

T(n, P) = constants + linear terms (≈ c·n) + Σ InsertHT(i) + P

Worst case: Σ InsertHT(i) = O(n²) ⇒ T(n, P) = O (n² + n + P) = O(n²)  
If prerequisites per course are bounded (P ≤ k·n), it stays) O(n²)

Hash table load phase worst case Big-O: **O(n²)**

**BST cost-per-line table**

|  |  |  |  |
| --- | --- | --- | --- |
| Step from pseudocode | Cost per execution | Executions | Total Cost |
| Reset ‘root = NULL’, ‘allCourses = empty’ | 1 | 1 | 1 |
| Open file | 1 | 1 | 1 |
| While not EOF: read line | 1 | n | n |
| Split by commas into tokens | 1 | n | n |
| Check ‘tokens < 2’ | 1 | n | n |
| Set ‘courseNumber = tokens[0]’ | 1 | n | n |
| Set ‘courseName = tokens[1]’ | 1 | n | n |
| Init empty ‘prereqList’ | 1 | n | n |
| For each prereq token ‘ADD tokens[i]’ | 1 | P | P |
| Create ‘Course newCourse’ | 1 | n | n |
| Set ‘newCourse.courseNumber’ | 1 | n | n |
| Set ‘newCourse.courseName’ | 1 | n | n |
| Set ‘newCourse.prerequisites = prereqList’ | 1 | n | n |
| Insert into BST ‘root = InsertNode(root, newCourse)’ | InsertBST(h) | n | Σ InsertBST(i) |
| Add ‘courseNumber’ to ‘allCourses’ | 1 | n | n |
| Close file | 1 | 1 | 1 |

Sum (load only)

T(n, P) = constants + linear terms (≈ c·n) + Σ InsertBST(i) + P

Worst case: Σ InsertBST(i) = O(n²) ⇒ T(n, P) = O(n²)

BST load phase worst case Big-O: **O(n²)**

**Advantages and Disadvantages**

**Vector**

Advantages:

 Simple to implement and understand

 Predictable O(n) load time in all cases

 Excellent cache locality; iterating over elements is fast

 Minimal memory overhead compared to other structures

Disadvantages:

 Searching for a specific course requires scanning the list, which is O(n) without

additional indexing

 Must perform a sort (O(n log n)) to get alphanumeric order for Option 2

 Inserting in the middle of the vector is costly (O(n)), so it’s not suited for

frequent ordered insertions

**Hash Table**

Advantages:

 Very fast lookups for single courses on average (O(1))

 Efficient average-case load time of O(n)

 Insertion performance doesn’t degrade with dataset size in average scenarios

 Built-in library support in many languages simplifies implementation

Disadvantages:

 No inherent ordering; extra work needed to produce sorted lists

 Performance depends heavily on a good hash function and proper resizing

strategy

 Worst-case O(n²) load time if extreme collisions occur

 Uses more memory due to bucket array and possible unused slots

**BST**

Advantages:

 In-order traversal naturally returns courses in alphanumeric order with no extra

sorting step

 Average-case search, insert, and delete operations are O(log n)

 Works well for datasets where ordered traversal is common

 Can handle dynamic insertions and deletions while maintaining order

Disadvantages:

 Without balancing, inserting sorted or nearly sorted data leads to O(n²)

performance

 More complex to implement than vectors or hash tables

 Higher memory usage due to node pointer storage

 Traversal costs more than a simple array iteration because of pointer chasing in

memory

**Recommendation**For ABCU’s advising system, I’d recommend the hash table as the best choice for primary data storage. It offers fast average case lookups for individual courses and efficient load times, which makes it ideal for quick retrieval of course titles and prerequisites. Since a hash table does not maintain any inherent ordering, the best approach for producing an alphanumerically sorted course list is to extract the values into a vector and sort them when needed. If maintaining a constantly sorted dataset is a higher priority than lookup speed, then implementing a balanced binary search tree, such as an AVL tree or a Red-Black Tree, would be a better alternative to the unbalanced BST used in the pseudocode.