Project One

Alejandro Martinez

Southern New Hampshire University

CS-300

Mais Nijim

10/11/2025

PseudoCode

Unified Menu Pseudocode

Function displayMenu()

WHILE true

Print "1. Load Data Structure"

Print "2. Print Course List (Sorted)"

Print "3. Print Course Information"

Print "9. Exit"

Prompt user for option

IF option == 1

Call loadCourses()

ELSE IF option == 2

Call printCourseListSorted()

ELSE IF option == 3

Prompt for course number

Call displayCourse(courseNumber)

ELSE IF option == 9

Exit program

ELSE

Print "Invalid selection"

Vector Pseudocode:

Initialize Course List (Vector):

Create list courseList of type Course

Function loadCourses():

Open file for reading

WHILE not end of file

Read and split line by commas

IF line has at least 2 values

Create Course object with courseNumber and courseName

IF additional values

FOR each additional value

Add to Course.prerequisites

Add Course to courseList

ELSE

Print "Invalid format - skipping"

Close file

Function printCourseListSorted():

Sort courseList by Course.number (alphanumeric sort)

FOR each Course in sorted courseList

Print Course.number + ": " + Course.name

Function displayCourse(courseNumber):

FOR each Course in courseList

IF Course.number == courseNumber

Print "Course: " + Course.number + " - " + Course.name

IF Course.prerequisites is empty

Print "Prerequisites: None"

ELSE

Print "Prerequisites:"

FOR each prereq in Course.prerequisites

Print prereq

RETURN

Print "Course not found"

Hash Table Pseudocode:

Initialize Hash Table:

Create courseTable as new HashTable

Function loadCourses():

Open file for reading

WHILE not end of file

Read and split line by commas

IF line has at least 2 values

Create Course object

IF additional values

FOR each value after index 1

Add to Course.prerequisites

Insert Course into courseTable using Course.number as key

ELSE

Print "Invalid format - skipping"

Close file

Function printCourseListSorted():

Get all Courses from hash table into list

Sort list by Course.number

FOR each Course in sorted list

Print Course.number + ": " + Course.name

Function displayCourse(courseNumber):

Search for Course in hash table by courseNumber

IF found

Print "Course: " + Course.number + " - " + Course.name

IF Course.prerequisites is empty

Print "Prerequisites: None"

ELSE

FOR each prereq in Course.prerequisites

Search prereqCourse in hash table

IF found

Print prereqCourse.number + ": " + prereqCourse.name

ELSE

Print prereq + " not found"

ELSE

Print "Course not found"

Binary Search Tree Pseudocode:

Initialize Binary Search Tree:

Set root to null

Function loadCourses():

Open file for reading

WHILE not end of file

Read and split line by commas

IF line has at least 2 values

Create Course object

IF more values

FOR each value after index 1

Add to Course.prerequisites

Insert Course into tree using insert()

ELSE

Print "Invalid format - skipping"

Close file

Function insert(Course):

IF root is null

Set root to new Node with Course

ELSE

Call insertNode(root, Course)

Function insertNode(node, Course):

IF Course.number < node.Course.number

IF node.left is null

node.left = new Node(Course)

ELSE

insertNode(node.left, Course)

ELSE

IF node.right is null

node.right = new Node(Course)

ELSE

insertNode(node.right, Course)

Function printCourseListSorted():

Call inOrderTraversal(root)

Function inOrderTraversal(node):

IF node is not null

inOrderTraversal(node.left)

Print node.Course.number + ": " + node.Course.name

inOrderTraversal(node.right)

Function displayCourse(courseNumber):

Set current = root

WHILE current is not null

IF courseNumber == current.Course.number

Print current.Course.number + ": " + current.Course.name

IF prerequisites is empty

Print "Prerequisites: None"

ELSE

FOR each prereq in current.Course.prerequisites

Print prereq

RETURN

ELSE IF courseNumber < current.Course.number

Move to current.left

ELSE

Move to current.right

Print "Course not found"

Runtime Analysis Chart

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | Hash Table | Binary Search TreeO(1) |
| Loading Data | O(1) | O(1) – O(n)  depends on hash collisions | O(log n) – O(n)  depends on tree balance |
| Search | O(n) | O(1) – O(n)  depends on hash collisions | O(log n) – O(n)  depends on tree balance |
| Sort / Print | O(n log n)  using alphanumeric sort | O(n)  requires extracting entries | O(n)  via in-order traversal |

Advantages and Disadvantages Discussion

Each data structure in my Pseudocode has its own strengths and weaknesses when working with course information. The Vector is simple and efficent for loading and sorting data, but searching takes longer because each course must be checked one at a time. The hash table is the fastest when inserting or finding a course by its number, which does fit well with my uses of Insert and Search functions, but collisions can slow it down and sorting requires extra steps since the data isnt stored in order. The binary search tree offers a good balance by keeping courses automatically organized and easy to print alphabetically through in order traversal. But, it can become slower if the tree becomes unbalanced and uses more memory to manage its node pointers. Overall, the vector is simple, the hash table is fast, and the binary search tree is organized, each serving a different purpose depending on what the program requires most.

Final Reccomendation

After reviewing the performance and structure of each data type in my pseudocode, I would recommend using the Hash Table for this program. Out of all three options, the hash table provided the best overall performance for quickly finding and storing course information by course number. This fits well with how many pseudocode uses the Insert and Search functions, allowing the program to access data almost instantly in most cases. The hash table makes sense for this type of project because users often need to look up individual courses or their prerequisites, and it can handle those lookups efficently even as the dataset grows. While sorting the courses requires an extra step, the speed of search and insertion outweighs that disadvantage for most situations.

The Vector and Binary Search Tree structures both have their benefits, but they don’t perform as efficently for this specific task. The vector is simple and easy to implement, but searching takes longer because it has to check each course one by one. The binary search tree naturally organizes data and is great for sorted printing, but if the data becomes unbalanced, its performance can drop significantly, and it uses more memory to manage node pointers. For these reasons, **t**he hash table is the best choice overall for storing and retrieving course data in my design because it provides a good balance of speed, scalability, and reliability.