Rene Guzman

Southern New Hampshire University

CS-300 DSA: Analysis and Design

August 10, 2025

**Evaluation**

Problem Context  
Advisors must (1) load a course catalog from file, (2) print the entire course list in alphanumeric order, and (3) print a specific course with its prerequisites. The data set is modest (hundreds to a few thousand courses), and the operations are mostly read-heavy (frequent prints/queries; occasional loads).  
  
Candidate Data Structures Evaluated  
1) Vector (dynamic array)  
- Insert: Amortized O(1) via push\_back; O(n) if kept sorted during insert.  
- Find: O(n) linear scan (unless maintained sorted + binary search).  
- Print all (alphanumeric): Requires O(n log n) sort each time (or maintain sorted invariant → O(n) insert).  
- Memory: O(n) contiguous; cache-friendly; simplest to implement.  
  
2) Hash Table (separate chaining, key = courseNumber)  
- Insert: Average O(1); worst O(n) under heavy collisions.  
- Find: Average O(1); worst O(n).  
- Print all (alphanumeric): Requires gathering + O(n log n) sort (hash buckets are not ordered).  
- Memory: O(n + M) (nodes + buckets); pointer overhead; depends on load factor.  
  
3) Binary Search Tree (unbalanced; key = courseNumber)  
- Insert: Average O(log n); worst O(n) if skewed.  
- Find: Average O(log n); worst O(n) if skewed.  
- Print all (alphanumeric): O(n) with in-order traversal (naturally ordered).  
- Memory: O(n) nodes; pointer overhead.  
  
Complexity Summary (per operation)  
  
Structure | Insert | Find | Print All (A–Z) | Extra Memory  
--------- | ------ | ---- | --------------- | -------------  
Vector | Amortized O(1); O(n) if kept sorted | O(n) | O(n log n) | O(n)  
Hash Table | Avg O(1); worst O(n) | Avg O(1); worst O(n) | O(n log n) | O(n + M)  
BST (unbalanced) | Avg O(log n); worst O(n) | Avg O(log n); worst O(n) | O(n) | O(n)  
  
Edge Cases & Risks  
- Skewed BST: If data arrives already sorted, an unbalanced BST can degrade to O(n). Mitigations: randomize insert order, or (future work) switch to a balanced tree.  
- CSV inconsistencies: Missing columns or malformed lines can crash naive loaders. Mitigation: guards and safe fallbacks.  
- Hash table tuning: With poor hash or high load factor, performance degrades.  
  
Recommendation & Justification  
Choose: Binary Search Tree (BST). It directly supports ordered prints (O(n)), has acceptable lookup times (O(log n) average), and balances implementation simplicity with project needs.  
  
When would a hash table be better? If the workflow were dominated by random lookups with rare full prints.  
  
Memory Considerations  
All three fit comfortably in memory given project scope. BST’s overhead is a fair trade for ordered printing.  
  
Implementation Notes  
- File handling: Loader validates columns and skips malformed rows.  
- Data structure choice: BST justified with trade-offs.  
- Operations covered: Load, print all in order, print single course.

**CS 300 Project One Milestone One**

Reflection

This milestone helped me understand how to translate a set of input requirements into structured pseudocode and plan for object-oriented design using vectors. I learned how to process each line from a file while accounting for potential format errors and how to validate prerequisites as course dependencies. Designing the pseudocode for storing courses in a vector improved my understanding of how objects relate to data structures. Lastly, creating the logic to search and print course details gave me insight into structuring efficient lookups. This foundation will help as I move toward full program implementation in future milestones.

Pseudocode

1. Pseudocode to open and parse a course file (with error checking)

Function LoadCourseFile(filePath)  
 Open the file at filePath  
 If file cannot be opened  
 Display error and exit  
 EndIf  
  
 For each line in file  
 Split the line into tokens using ',' as delimiter  
 If line has fewer than 2 tokens  
 Display format error and skip line  
 EndIf  
  
 courseNumber = tokens[0]  
 courseTitle = tokens[1]  
 prerequisites = tokens[2] to end (if any)  
  
 For each prerequisite in prerequisites  
 If prerequisite does not exist in file  
 Display error and skip line  
 EndIf  
 EndFor  
  
 Create a Course object using courseNumber, courseTitle, and prerequisites  
 Add Course object to vector  
 EndFor  
 Close the file  
EndFunction

2. Pseudocode to create course objects and store them in a vector

Structure Course  
 courseNumber: String  
 courseTitle: String  
 prerequisites: List of Strings  
EndStructure  
  
Function CreateCourse(courseNumber, courseTitle, prereqList)  
 course = new Course  
 course.courseNumber = courseNumber  
 course.courseTitle = courseTitle  
 course.prerequisites = prereqList  
 Return course  
EndFunction  
  
Function StoreCoursesInVector(filePath)  
 Create empty vector<Course> courseList  
 Open file at filePath  
 For each line in file  
 Parse courseNumber, courseTitle, prereqs  
 course = CreateCourse(courseNumber, courseTitle, prereqs)  
 Add course to courseList  
 EndFor  
 Close file  
EndFunction

3. Pseudocode to search for a course and print course + prerequisite info

Function PrintCourseInfo(courseList, targetCourseNumber)  
 For each course in courseList  
 If course.courseNumber == targetCourseNumber  
 Display course.courseNumber and course.courseTitle  
 If course.prerequisites is empty  
 Display "No prerequisites"  
 Else  
 For each prereq in course.prerequisites  
 For each c in courseList  
 If c.courseNumber == prereq  
 Display c.courseNumber and c.courseTitle  
 EndIf  
 EndFor  
 EndFor  
 EndIf  
 EndIf  
 EndFor  
EndFunction

**Milestone 2: Pseudocode**

// Function to load course data from a file into a hash table

function loadCoursesFromFile(filePath, HashTable<Course> courses) {

open file at filePath

for each line in file {

split line by commas into tokens

// validate line has at least 2 tokens

if number of tokens < 2

print error: "Invalid line format"

continue

create new Course object

set Course.courseNumber = tokens[0]

set Course.courseName = tokens[1]

// store remaining tokens as prerequisites

for i from 2 to length(tokens) - 1 {

add tokens[i] to Course.prerequisites

}

// insert the course into the hash table using courseNumber as the key

courses.insert(Course.courseNumber, Course)

}

close file

}

// Function to display course information and prerequisites

function printCourseInfo(HashTable<Course> courses, string courseNumber) {

course = courses.search(courseNumber)

if course is null {

print "Course not found."

return

}

print course.courseNumber + ", " + course.courseName

for each prereq in course.prerequisites {

prereqCourse = courses.search(prereq)

if prereqCourse is not null {

print " " + prereqCourse.courseNumber + ", " + prereqCourse.courseName

} else {

print " " + prereq + " (prerequisite not found in file)"

}

}

}

**Milestone 3: Pseudocode**

**1. Open, Read, and Validate File**

FUNCTION LoadCourseData(filePath, bst)  
 PRINT "Loading course data from", filePath  
  
 TRY  
 OPEN filePath FOR reading  
 CATCH  
 PRINT "Error: Cannot open file."  
 RETURN  
 END TRY  
  
 FOR EACH line IN file  
 IF line IS empty  
 CONTINUE  
 END IF  
  
 PARSE line INTO: courseNumber, courseTitle, prerequisitesList  
  
 IF courseNumber IS empty OR courseTitle IS empty  
 PRINT "Invalid line format. Skipping line."  
 CONTINUE  
 END IF  
  
 FOR EACH prereq IN prerequisitesList  
 IF prereq NOT found in file  
 PRINT "Warning: Prerequisite", prereq, "not found in course list."  
 END IF  
 END FOR  
  
 CREATE Course object:  
 Course.courseNumber = courseNumber  
 Course.courseTitle = courseTitle  
 Course.prerequisites = prerequisitesList  
  
 CALL bst.Insert(Course)  
 END FOR  
  
 CLOSE file  
END FUNCTION

**2. Create and Store Course Objects in BST**  
FUNCTION Insert(bst, course)  
 IF bst.root IS NULL  
 bst.root = NEW Node(course)  
 ELSE  
 CALL AddNode(bst.root, course)  
 END IF  
END FUNCTION  
  
FUNCTION AddNode(currentNode, course)  
 IF course.courseNumber < currentNode.course.courseNumber  
 IF currentNode.left IS NULL  
 currentNode.left = NEW Node(course)  
 ELSE  
 CALL AddNode(currentNode.left, course)  
 END IF  
 ELSE  
 IF currentNode.right IS NULL  
 currentNode.right = NEW Node(course)  
 ELSE  
 CALL AddNode(currentNode.right, course)  
 END IF  
 END IF  
END FUNCTION

**3. Print Course Information and Prerequisites**  
FUNCTION PrintCourseListInOrder(node)  
 IF node IS NULL  
 RETURN  
 END IF  
  
 CALL PrintCourseListInOrder(node.left)  
 PRINT node.course.courseNumber, node.course.courseTitle  
 IF node.course.prerequisites IS NOT empty  
 PRINT "Prerequisites:", node.course.prerequisites  
 ELSE  
 PRINT "Prerequisites: None"  
 END IF  
 CALL PrintCourseListInOrder(node.right)  
END FUNCTION  
  
FUNCTION FindAndPrintCourse(bst, courseNumber)  
 currentNode = bst.root  
 WHILE currentNode IS NOT NULL  
 IF courseNumber == currentNode.course.courseNumber  
 PRINT currentNode.course.courseNumber, currentNode.course.courseTitle  
 IF currentNode.course.prerequisites IS NOT empty  
 PRINT "Prerequisites:", currentNode.course.prerequisites  
 ELSE  
 PRINT "Prerequisites: None"  
 END IF  
 RETURN  
 ELSE IF courseNumber < currentNode.course.courseNumber  
 currentNode = currentNode.left  
 ELSE  
 currentNode = currentNode.right  
 END IF  
 END WHILE  
 PRINT "Course not found."  
END FUNCTION