Jas King

Prof. Leslie Gruner

CS-300 DSA: Analysis & Design

June 14, 2025

6-2 Project One:

Pseudocode & Runtime Analysis

**Part One:**

**// File Handling and Initialization**

Utilize fstream to open the file "Course Information".

Define a method void **loadCourses(string csvPath, vector<Course>& courseVector)** to handle the loading process

Initialize an empty vector **courseVector** to store course objects.

**// File Opening and Error Handling**

Open file "Course Information" using **fstream**

If the file cannot be opened:

Print "Error: Unable to open file."

Exit the program

**// Reading and Parsing Data**

While the file can be opened and read:

For each line in the file:

Split the line into tokens using comma as the separator

If the number of tokens < 2:

Print "Invalid line format"

Continue to the next line

If the tokens 2:

Create a new **Course** object

Set **Course ID** to the first token

Set **Course Name** to the second token

For each token from the third to the last token:

Set token to the Course's Prerequisites

Add the **Course** object to the **courseVector**

Close the file

**// Prerequisite Validation**

For each **Course** in **courseVector**:

For each prerequisite in the Course's prerequisites:

If there is no **Course** in **courseVector** with the Course ID equal to the prerequisite:

Print "Invalid prerequisite for Course ID: [Course ID]"

**// Course Struct Definition**

Define struct **Course**:

Fields:

String **id**

String **name**

Vector<string> **prerequisites**

**// Course Object Creation**: **A Course object is created for each line read, its fields populated from the tokens, and added to courseVector.**

**// Function to Print Course Information**

Define function **printCourseInfo(string courseNumber, vector<Course> courseVector)**:

For each **Course** object in **courseVector**:

If the **Course** object's ID is equal to **courseNumber**:

Print "Course ID: [Course ID], Title: [Course Name], Prerequisites: [Prerequisites]"

Return

If no course is found, Print "Course not found"

**Part Two:**

**// Vector Data Structure**

Struct Course

String **id**

String **name**

Vector<string> **prerequisites**

Function **loadCourses(csvPath, Vector<Course>& courseVector)**

Open file at **csvPath**

If file cannot be opened

Print "Error: Unable to open file."

Return

For each line in file

Split line by commas into tokens

If **tokens.length** < 2

Print "Invalid line format"

Continue

Initialize Course **newCourse**

Set **newCourse.id** = tokens[0]

Set **newCourse.name** = tokens[1]

For i FROM 2 TO **tokens.length** - 1

Add tokens[i] TO **newCourse.prerequisites**

Add **newCourse** TO **courseVector**

Close file

**// Validate Prerequisites**

For each course IN **courseVector**

For each prereq IN **course.prerequisites**

Set found = FALSE

For each **checkCourse** IN **courseVector**

If **checkCourse.id** == prereq

Set found = TRUE

Break

If found == FALSE

Print "Invalid prerequisite for Course ID: " + **course.id**

Function **printCourseInfo(courseID, Vector<Course> courseVector)**

For each course IN **courseVector**

If **course.id** == **courseID**

Print "Course ID: " + **course.id**

Print "Title: " + **course.name**

Print "Prerequisites: " + **JOIN(course.prerequisites, ", ")**

Return

Print "Course not found."

Function **printCourseList(Vector<Course> courseVector)**

Sort **courseVector** BY course.id

For each course IN **courseVector**

Print course.id + ": " + course.name

Function menu()

While TRUE

Print "1. Load Data"

Print "2. Print Course List"

Print "3. Print Course Info"

Print "9. Exit"

Input choice

SWITCH choice

CASE 1: CALL **loadCourses("courses.csv", courseVector)**

CASE 2: CALL **printCourseList(courseVector)**

CASE 3: Input **courseID**

CALL **printCourseInfo(courseID, courseVector)**

CASE 9: Exit Program

DEFAULT: Print "Invalid option"

**Big O Analysis for Vector:**

Loading courses: O(n) for reading lines, O(n2) for validating prerequisites (in the worst case).

Printing course information: O(n) for searching.

Sorting course list: O(n log n).

**// Hash Table Data Structure**

Struct **Course**

String **id**

String **name**

Vector<String> **prerequisites**

Function hash(key)

**// Return index based on key (e.g., simple mod or custom hash)**

Function **loadCourses(csvPath, Hash\_Table<Course>& courseTable)**

Open file at **csvPath**

If file cannot be opened

Print "Error: Unable to open file."

Return

For each line IN file

Split line by commas into tokens

If **tokens.length** < 2

Print "Invalid line format"

Continue

Initialize Course **newCourse**

Set newCourse.id = tokens[0]

Set newCourse.name = tokens[1]

For i FROM 2 TO tokens.length - 1

Add tokens[i] TO **newCourse.prerequisites**

Insert **newCourse** INTO **courseTable** with key newCourse.id

Close file

**// Validate prerequisites**

For each key, course IN **courseTable**

For each prereq IN **course.prerequisites**

If NOT **courseTable.CONTAINS(prereq)**

Print "Invalid prerequisite for Course ID: " + course.id

Function **printCourseInfo(courseID, Hash\_Table<Course> courseTable)**

If **courseTable.Contains(courseID)**

Set course = **courseTable**[courseID]

Print course.id + ": " + course.name

Print "Prerequisites: " + **JOIN(course.prerequisites, ", ")**

Else

Print "Course not found"

Function **printCourseList(Hash\_Table<Course> courseTable)**

Create **courseList** = EMPTY VECTOR

For each key, course IN **courseTable**

Add course TO **courseList**

Sort **courseList** BY course.id

For each course IN **courseList**

Print course.id + ": " + course.name

Function menu()

**// Same structure as vector**

**Big O Analysis for Hash Table:**

Loading courses: O(n) for reading lines, O(1) for inserting into the hash table.

Validating prerequisites: O(n) for iterating through courses, O(1) for checking existence.

Printing course information: O(1) for access.

Sorting course list: O(n log n).

**// Binary Search Tree Data Structure**

Struct **Course**

String **id**

String **name**

Vector<String> **prerequisites**

Struct **BSTNode**

Course course

BSTNode left

BSTNode right

Function **insertNode(BSTNode\* &root, Course newCourse)**

If root is NULL

root = NEW **BSTNode(newCourse)**

Else If newCourse.id < root.course.id

insertNode(root.left, newCourse)

Else

insertNode(root.right, newCourse)

Function findCourse(BSTNode\* root, String courseID)

If root is **NULL**

Return **NULL**

If root.course.id == **courseID**

Return root.course

If **courseID** < root.course.id

Return **findCourse(root.left, courseID)**

Else

Return **findCourse(root.right, courseID)**

Function **loadCourses(csvPath, BSTNode\* &root)**

Open file

If file cannot be opened

Print "Error"

Return

For each line IN file

Parse line into Course **newCourse**

**insertNode**(root, **newCourse**)

Close file

Function **validatePrerequisites(BSTNode\* root, BSTNode\* fullRoot)**

If root is NULL

Return

For each prereq IN root.course.prerequisites

If **findCourse(fullRoot, prereq)** == NULL

Print "Invalid prerequisite for " + root.course.id

**validatePrerequisites(root.left, fullRoot)**

**validatePrerequisites(root.right, fullRoot)**

Function **printCourseListInOrder(BSTNode\* root)**

If root is NULL

Return

**printCourseListInOrder(root.left)**

Print root.course.id + ": " + root.course.name

**printCourseListInOrder(root.right)**

Function **printCourseInfo(courseID, BSTNode\* root)**

Set course = **findCourse(root, courseID)**

If course IS NOT NULL

Print course.id + ": " + course.name

Print "Prerequisites: " + **JOIN(course.prerequisites, ", ")**

Else

Print "Course not found"

**Big O Analysis for Binary Search Tree:**

Loading courses: O(n) for reading lines, O(log n) for the insertion of each course (in the average case).

Validating prerequisites: O(n) for tree traversal, O(log n) for searching each prerequisite (in the average case).

Printing course information: O(log n) for the search operation.

Printing the course list: O(n) for in-order traversal.

**Conclusion:**

Selecting the appropriate data structure is crucial for our course management system's efficiency. Vectors are user-friendly but inefficient for large datasets, with O(n2) complexity in prerequisite validation. Hash tables provide average O(1) time complexity for access and insertion, making them dependable. Binary search trees allow for ordered data storage and efficient searching, but can degrade to O(n) in the worst case if unbalanced.