**Project One**

Tremaine Rayner

Southern New Hampshire University

CS-300 DSA: Analysis and Design

Professor Jamaliannasrabadi

December 8th, 2024

**Project One**

**VECTOR**

Start

// Define a class for a course

CLASS Course

ATTRIBUTES:

courseNumber : STRING // Unique course number

courseTitle : STRING // Title of the course

prerequisites : LIST<STRING> // Prerequisite courses

END CLASS

// Function to load courses from a file

FUNCTION loadCourses(fileName : STRING) : LIST<Course>

courses : LIST<Course> // List to store courses

file : FILE

courseData : LIST<STRING> // Temporary storage for file lines

existingCourseNumbers : SET<STRING> // To store all course numbers for validation

OPEN file WITH fileName FOR reading

IF file DOES NOT EXIST THEN

PRINT "Error: File not found."

RETURN an empty list

END IF

// Read file into a list for post-processing

FOR EACH line IN file

APPEND line TO courseData

END FOR

CLOSE file

// First pass: Extract and validate course numbers

FOR EACH line IN courseData

inputStream : STRING\_STREAM INITIALIZED WITH line

courseNumber, courseTitle : STRING

READ courseNumber FROM inputStream (delimited by ',')

READ courseTitle FROM inputStream (delimited by ',')

IF courseNumber IS EMPTY OR courseTitle IS EMPTY THEN

PRINT "Skipping line due to incorrect format: " + line

CONTINUE

END IF

ADD courseNumber TO existingCourseNumbers

END FOR

// Second pass: Create courses and validate prerequisites

FOR EACH line IN courseData

course : Course

inputStream : STRING\_STREAM INITIALIZED WITH line

courseNumber, courseTitle, prereq : STRING

READ courseNumber FROM inputStream (delimited by ',')

READ courseTitle FROM inputStream (delimited by ',')

// Skip previously identified invalid lines

IF courseNumber NOT IN existingCourseNumbers THEN

CONTINUE

END IF

ASSIGN course.courseNumber = courseNumber

ASSIGN course.courseTitle = courseTitle

// Read and validate all prerequisites

WHILE NOT END OF inputStream

READ prereq FROM inputStream (delimited by ',')

IF prereq NOT IN existingCourseNumbers THEN

PRINT "Error: Prerequisite " + prereq + " not found for course " + courseNumber

END IF

ADD prereq TO course.prerequisites

END WHILE

ADD course TO courses

END FOR

RETURN courses

END FUNCTION

// Function to search for a course

FUNCTION searchCourse(courses : LIST<Course>, courseNumber : STRING)

courseNumber = TO\_LOWERCASE(courseNumber) // Case-insensitive comparison

FOR EACH course IN courses

IF TO\_LOWERCASE(course.courseNumber) == courseNumber THEN

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

IF course.prerequisites IS NOT EMPTY THEN

PRINT "Prerequisites:"

FOR EACH prereq IN course.prerequisites

PRINT "- " + prereq

END FOR

ELSE

PRINT "No prerequisites available for this course."

END IF

RETURN

END IF

END FOR

PRINT "Error: Course " + courseNumber + " not found."

END FUNCTION

// Function to sort courses by course number

FUNCTION sortCourses(courses : LIST<Course>)

// Simple sorting algorithm (e.g., bubble sort)

FOR i FROM 0 TO LENGTH(courses) - 2

FOR j FROM 0 TO LENGTH(courses) - 2 - i

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

SWAP courses[j] WITH courses[j + 1]

END IF

END FOR

END FOR

END FUNCTION

// Function to print all courses

FUNCTION printAllCourses(courses : LIST<Course>)

IF courses IS EMPTY THEN

PRINT "Courses not available."

RETURN

END IF

// Sort courses before printing

CALL sortCourses(courses)

PRINT "All Available Courses:"

FOR EACH course IN courses

PRINT "Course Number: " + course.courseNumber

PRINT "Course Title: " + course.courseTitle

IF course.prerequisites IS NOT EMPTY THEN

PRINT "Prerequisites:"

FOR EACH prereq IN course.prerequisites

PRINT "- " + prereq

END FOR

ELSE

PRINT "No prerequisites available for this course."

END IF

PRINT "------------------------"

END FOR

END FUNCTION

// Main program execution

FUNCTION main()

courses : LIST<Course>

fileName : STRING

choice : INTEGER

// Prompt user for file name

PRINT "Enter the file: "

READ fileName

// Load courses from the file

ASSIGN courses = loadCourses(fileName)

IF courses IS EMPTY THEN

PRINT "Exiting program, no courses loaded."

RETURN

END IF

// Main menu loop

DO

PRINT "\nMenu:"

PRINT "1. Load data into the data structure"

PRINT "2. Print courses"

PRINT "3. Search course"

PRINT "9. Exit"

PRINT "Enter your choice (1-3, 9 to exit): "

READ choice

SWITCH choice

CASE 1:

ASSIGN courses = loadCourses(fileName)

BREAK

CASE 2:

CALL printAllCourses(courses)

BREAK

CASE 3:

courseNumber : STRING

PRINT "Enter the course number to search for: "

READ courseNumber

CALL searchCourse(courses, courseNumber)

BREAK

CASE 9:

PRINT "Goodbye!"

BREAK

DEFAULT:

PRINT "Invalid Selection."

END SWITCH

WHILE choice != 9

END FUNCTION

// Run the program

CALL main()

END

**Hash Table**

**START**

**// Define a structure for a Course**

**STRUCT Course**

**STRING number // Course number**

**STRING name // Course name**

**LIST prerequisites // List of prerequisites**

**// Global hash table to store courses**

**hashTable AS HASH\_TABLE**

**sortedCourses AS LIST<Course> // Temporary list to store sorted courses**

**// Function: Display menu**

**FUNCTION printMenu()**

**PRINT "Menu:"**

**PRINT "1. Load data into the data structure"**

**PRINT "2. Display all courses"**

**PRINT "3. Search for a specific course"**

**PRINT "9. Exit"**

**// Function: Read and process file**

**FUNCTION readFile(fileName)**

**// Open the file for reading**

**OPEN fileName AS file**

**IF file NOT open**

**// Error handling if the file cannot be opened**

**PRINT "Error: File cannot be opened"**

**RETURN FALSE**

**END IF**

**// Temporary storage for processing courses**

**tempCourses : LIST**

**// Process each line in the file**

**FOR each line IN file**

**IF NOT processLine(line, tempCourses)**

**PRINT "Error: Processing failed: " + line**

**RETURN FALSE**

**END IF**

**END FOR**

**CLOSE file**

**// Validate prerequisites after processing all courses**

**FOR each tempCourse IN tempCourses**

**IF NOT validatePrerequisites(tempCourse)**

**PRINT "Error: Prerequisite validation failed for course " + tempCourse.number**

**RETURN FALSE**

**END IF**

**ADD tempCourse TO hashTable**

**END FOR**

**RETURN TRUE**

**END FUNCTION**

**// Function: Process a single line from the file**

**FUNCTION processLine(line, tempCourses)**

**IF line IS EMPTY THEN**

**PRINT "Error: Empty line."**

**RETURN FALSE**

**END IF**

**// Split the line into course number, course name, and prerequisites**

**SPLIT line BY ',' INTO courseNumber, courseName, prerequisites**

**// Ensure at least two parameters on each line**

**IF courseNumber IS EMPTY OR courseName IS EMPTY THEN**

**PRINT "Error: Line format is incorrect. Expected course number and name."**

**RETURN FALSE**

**END IF**

**// Create and temporarily store the course**

**CREATE newCourse**

**SET newCourse.number = courseNumber**

**SET newCourse.name = courseName**

**SET newCourse.prerequisites = prerequisites**

**ADD newCourse TO tempCourses**

**RETURN TRUE**

**END FUNCTION**

**// Function: Validate prerequisites**

**FUNCTION validatePrerequisites(course)**

**FOR each prereq IN course.prerequisites**

**IF NOT courseExists(prereq)**

**PRINT "Error: Prerequisite " + prereq + " not found for course " + course.number**

**RETURN FALSE**

**END IF**

**END FOR**

**RETURN TRUE**

**END FUNCTION**

**// Function: Add a course to the hash table**

**FUNCTION addCourse(courseNumber, courseName, prerequisites)**

**IF courseNumber IS EMPTY OR courseName IS EMPTY THEN**

**PRINT "Error: Cannot add course with missing number or name."**

**RETURN**

**END IF**

**CREATE new Course**

**SET new Course.number = courseNumber**

**SET new Course.name = courseName**

**SET new Course.prerequisites = prerequisites**

**INSERT new Course INTO hashTable WITH KEY courseNumber**

**END FUNCTION**

**// Function: Check if a course exists in the hash table**

**FUNCTION courseExists(courseNumber)**

**RETURN courseNumber EXISTS IN hashTable**

**END FUNCTION**

**// Function: Sort courses by course number**

**FUNCTION sortCourses()**

**sortedCourses = hashTable TO LIST<Course> // Convert hash table to list**

**FOR i FROM 0 TO LENGTH(sortedCourses) - 2**

**FOR j FROM 0 TO LENGTH(sortedCourses) - 2 - i**

**IF sortedCourses[j].number > sortedCourses[j + 1].number THEN**

**SWAP sortedCourses[j] WITH sortedCourses[j + 1]**

**END IF**

**END FOR**

**END FOR**

**END FUNCTION**

**// Function: Print all courses**

**FUNCTION printAllCourses()**

**IF hashTable IS EMPTY THEN**

**PRINT "No courses available."**

**RETURN**

**END IF**

**// Sort courses before printing**

**CALL sortCourses()**

**FOR each course IN sortedCourses**

**PRINT "Course Number: " + course.number**

**PRINT "Course Name: " + course.name**

**IF course.prerequisites IS EMPTY THEN**

**PRINT "Prerequisites: None"**

**ELSE**

**PRINT "Prerequisites: " + JOIN course.prerequisites BY ", "**

**END IF**

**PRINT "-----------------------------------"**

**END FOR**

**END FUNCTION**

**// Function: Search for a specific course**

**FUNCTION searchCourse(courseNumber)**

**IF courseNumber IS EMPTY THEN**

**PRINT "Course number cannot be empty."**

**RETURN**

**END IF**

**LOWERCASE searchNumber = TO\_LOWER(courseNumber) // Normalize input**

**FOR each course IN hashTable**

**IF TO\_LOWER(course.number) == searchNumber THEN // Case-insensitive match**

**PRINT "Course Number: " + course.number**

**PRINT "Course Name: " + course.name**

**IF course.prerequisites IS EMPTY THEN**

**PRINT "Prerequisites: None"**

**ELSE**

**PRINT "Prerequisites: " + JOIN course.prerequisites BY ", "**

**END IF**

**RETURN**

**END IF**

**END FOR**

**PRINT "Error: Course " + courseNumber + " not found"**

**END FUNCTION**

**// Main Function**

**FUNCTION main()**

**PRINT "Welcome to the Course Management System!"**

**PROMPT "Please enter the file name containing course data: " INTO fileName**

**IF fileName IS EMPTY THEN**

**PRINT "File name cannot be empty."**

**RETURN**

**END IF**

**IF NOT readFile(fileName)**

**PRINT "Exiting program because of file read errors."**

**RETURN**

**END IF**

**WHILE TRUE**

**CALL printMenu()**

**PROMPT "Enter your choice: " INTO choice**

**IF choice IS NOT INTEGER THEN**

**PRINT "Invalid input. Please enter a number."**

**CONTINUE**

**END IF**

**SWITCH choice**

**CASE 1:**

**IF NOT readFile(fileName)**

**PRINT "Exiting program because of file read errors."**

**RETURN**

**END IF**

**CASE 2:**

**CALL printAllCourses()**

**CASE 3:**

**PROMPT "Enter the course number to search: " INTO courseNumber**

**IF courseNumber IS EMPTY THEN**

**PRINT "Error: Course number cannot be empty."**

**CONTINUE**

**END IF**

**CALL searchCourse(courseNumber)**

**CASE 9:**

**PRINT "Goodbye!"**

**BREAK**

**DEFAULT:**

**PRINT "Invalid choice."**

**END SWITCH**

**END WHILE**

**END FUNCTION**

**CALL main()**

**END**

**Binary Search Tree**

**Start**

**// Define Course and Node Structures**

**Course:**

**String courseNumber // Unique identifier for the course**

**String courseName // Name of the course**

**List<String> prerequisites // List of prerequisite course numbers**

**Node:**

**Course course // Course information**

**Node left // Pointer to the left child node**

**Node right // Pointer to the right child node**

**// Define Binary Search Tree Class**

**BinarySearchTree:**

**Node root // Root of the BST**

**// Method to Insert a Course into the BST**

**Method insert(Node& node, Course course):**

**If node IS NULL:**

**Create a new node with course and assign it to node**

**Else if course.courseNumber < node.course.courseNumber:**

**Call insert(node.left, course)**

**Else if course.courseNumber > node.course.courseNumber:**

**Call insert(node.right, course)**

**Else:**

**Print "Error: Duplicate course detected. Skipping insertion."**

**Method insert(Course course):**

**Call insert(root, course)**

**// Method for In-Order Traversal**

**Method inOrderTraversal(Node node):**

**If node IS NOT NULL:**

**Call inOrderTraversal(node.left)**

**Print "Course Number: ", node.course.courseNumber, " - ", node.course.courseName**

**Call inOrderTraversal(node.right)**

**Method inOrderTraversal():**

**If root IS NULL:**

**Print "Error: No courses available."**

**Else:**

**Call inOrderTraversal(root)**

**// Method to Search for a Course**

**Method search(Node node, String courseNumber):**

**If node IS NULL:**

**Return NULL**

**Else if node.course.courseNumber == courseNumber:**

**Return node**

**Else if courseNumber < node.course.courseNumber:**

**Return search(node.left, courseNumber)**

**Else:**

**Return search(node.right, courseNumber)**

**Method search(String courseNumber):**

**If root IS NULL:**

**Print "Error: No courses available."**

**Return NULL**

**Return search(root, courseNumber)**

**// Function to Load Courses from File**

**Function loadCoursesFromFile(String fileName) -> BinarySearchTree:**

**Create a new BinarySearchTree courseTree**

**Open file fileName**

**If file CANNOT be opened:**

**Print "Error: Cannot open the file."**

**Return courseTree**

**For each line in file:**

**If line IS EMPTY:**

**Print "Warning: Empty line in file."**

**Continue**

**Split the line by ',' into courseNumber, courseName, and prerequisites**

**If courseNumber IS EMPTY OR courseName IS EMPTY:**

**Print "Error: Course format is invalid."**

**Continue**

**Create a new Course with courseNumber, courseName, and prerequisites**

**Add the course to courseTree**

**Close the file**

**If courseTree.root IS NULL:**

**Print "Warning: The file had no valid courses, so nothing was loaded."**

**Return courseTree**

**// Function to Search for a Course and Print Details**

**Function searchAndPrintCourse(BinarySearchTree courseTree, String courseNumber):**

**If courseNumber IS EMPTY:**

**Print "Error: Course number cannot be empty."**

**Return**

**Node node = courseTree.search(courseNumber)**

**If node IS NULL:**

**Print "No course found."**

**Else:**

**Print "Course Number: ", node.course.courseNumber**

**Print "Course Name: ", node.course.courseName**

**Print "Prerequisites: "**

**If node.course.prerequisites IS EMPTY:**

**Print "None"**

**Else:**

**For each prereq IN node.course.prerequisites:**

**Print prereq**

**// Function to Print All Courses in Order**

**Function printAllCourses(BinarySearchTree courseTree):**

**Print "Courses in alphabetical order:"**

**Call courseTree.inOrderTraversal()**

**// Main Function**

**Function main():**

**Print "ABCU Course Planner"**

**Print "Enter the file name: "**

**Read fileName**

**If fileName IS EMPTY:**

**Print "Error: The file name cannot be empty. Exiting program."**

**Return**

**BinarySearchTree courseTree = loadCoursesFromFile(fileName)**

**If courseTree.root IS NULL:**

**Print "No courses were loaded. Exiting program."**

**Return**

**Integer choice**

**Do:**

**Print "\nMenu:"**

**Print "1. Load data into the data structure"**

**Print "2. Print all courses"**

**Print "3. Search for a course"**

**Print "9. Exit"**

**Print "Enter your choice: "**

**Read choice**

**If choice IS NOT a valid integer:**

**Print "Error: Please enter a number between 1 and 3."**

**Continue**

**Switch(choice):**

**Case 1:**

**courseTree = loadCoursesFromFile(fileName)**

**Case 2:**

**Call printAllCourses(courseTree)**

**Case 3:**

**Print "Enter course number: "**

**Read courseNumber**

**Call searchAndPrintCourse(courseTree, courseNumber)**

**Case 9:**

**Print "Goodbye!"**

**Default:**

**Print "Error: Please select a valid option."**

**While choice != 9**

**END**

| **Vector Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all lines in file** | 1 | n | n |
| **read courseNumber and courseTitle from line** | 1 | n | n |
| **check if courseNumber or courseTitle is empty** | 1 | n | n |
| **add courseNumber to existingCourseNumbers** | 1 | n | n |
| **for all lines in file (second pass)** | 1 | n | n |
| **read courseNumber, courseTitle, and prereqs** | 1 | n | n |
| **check if courseNumber is valid** | 1 | n | n |
| **while not end of inputStream (read prereqs)** | 1 | n | n |
| **add prereq to course.prerequisites** | 1 | n | n |
| **add course to courses** | 1 | n | n |
| **sort courses by course number** | n^2 | 1 | n^2 |
| **search for course by courseNumber** | 1 | n | n |
| **print course information** | 1 | 1 | 1 |
| **Total Cost** | | | 12n + n^2 + 1 |
| **Runtime** | | | O(n^2) |

| **Hash Table Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all lines in file** | 1 | n | n |
| **split line into courseNumber, courseName, prereqs** | 1 | n | n |
| **check if courseNumber or courseName is empty** | 1 | n | n |
| **add newCourse to tempCourses** | 1 | n | n |
| **for all tempCourses** | 1 | n | n |
| **validate prerequisites** | 1 | n | n |
| **add tempCourse to hashTable** | 1 | n | n |
| **sort courses by course number** | n^2 | 1 | n^2 |
| **search for course by courseNumber** | 1 | 1 | 1 |
| **print course information** | 1 | 1 | 1 |
| **Total Cost** | | | **9n + n^2 + 2** |
| **Runtime** | | | **O(n^2)** |

| **Binary Search Tree Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all lines in file** | 1 | n | n |
| **split line into courseNumber, courseName, prereqs** | 1 | n | n |
| **check if courseNumber or courseName is empty** | 1 | n | n |
| **create a new Course** | 1 | n | n |
| **insert course into BST** | log(n) | n | nlog(n)\* |
| **search for course by courseNumber** | log(n) | 1 | log(n) |
| **print course information** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + nlog(n) + log(n) + 1 |
| **Runtime** | | | **O(n log(n)** |

**Recommendation**

Judging from the analysis, all three data structures have their strengths and weaknesses. Vectors are easy to use and provide dynamic sizing, but they have linear time complexity for searching and costly insertions/deletions, making them less ideal for this project. Hash tables offer average constant-time operations for insertion, deletion, and search, making them efficient for large datasets. However, they do not maintain order and require handling collisions. Binary search trees provide ordered data and efficient range queries, but maintaining balance can be complex, and unbalanced trees can degrade to linear time complexity. I recommend using the **hash table** for this project. The constant-time performance it provides for insertion, deletion, and search operations makes it the most efficient choice for managing the ABCU course data. As I mentioned the hash table does not maintain order, I believe the performance benefits will outweigh this drawback, especially for large datasets. The hash table will ensure fast response times for user queries, enhancing the overall user experience.