Serenity Rogers

CS 300 11336

Project One

8/10/24

Project One

Structure Course

String courseId

String courseName

Vector of Strings prerequisites

End Structure

Vector of Course courses

Set of String courseIds

Function loadCourses(String filePath)

File file = open(filePath)

If file is not open

Print "Error: Unable to open file."

Return

End If

While file has more lines

String line = read next line from file

Vector of String tokens = split(line, ',')

If tokens.size < 2

Print "Error: Line does not have at least two parameters."

Continue

End If

Course course

course.courseId = tokens[0].trim()

course.courseName = tokens[1].trim()

For i from 2 to tokens.size - 1

String prerequisite = tokens[i].trim()

If prerequisite is not empty

course.prerequisites.push\_back(prerequisite)

End If

End For

courses.push\_back(course)

courseIds.insert(course.courseId)

End While

Close file

End Function

Function validateCourses()

For each course in courses

For each prerequisite in course.prerequisites

If prerequisite not in courseIds

Print "Error: Prerequisite " + prerequisite + " for course " + course.courseId + " does not exist."

Remove prerequisite from course.prerequisites

End If

End For

End For

End Function

Function printCourseInfo(String courseId)

Boolean courseFound = False

For each course in courses

If course.courseId equals courseId

courseFound = True

Print "Course ID: " + course.courseId

Print "Course Name: " + course.courseName

Print "Prerequisites: "

If course.prerequisites.size > 0

For each prerequisite in course.prerequisites

Print " " + prerequisite

End For

Else

Print " None"

End If

Break

End If

End For

If not courseFound

Print "Course " + courseId + " not found."

End If

End Function

Structure Course

String courseNumber

String courseName

Vector of Strings prerequisites

End Structure

HashTable of Course courses

Set of String courseNumbers

Function loadCourses(String filePath)

File file = open(filePath)

If file is not open

Print "Error: Unable to open file."

Return

End If

While file has more lines

String line = read next line from file

Vector of String tokens = split(line, ',')

If tokens.size < 2

Print "Error: Line does not have at least two parameters."

Continue

End If

Course course

course.courseNumber = tokens[0].trim()

course.courseName = tokens[1].trim()

For i from 2 to tokens.size - 1

String prerequisite = tokens[i].trim()

If prerequisite is not empty

course.prerequisites.push\_back(prerequisite)

End If

End For

courses[course.courseNumber] = course

courseNumbers.insert(course.courseNumber)

End While

Close file

End Function

Function validateCourses()

For each course in courses

For each prerequisite in course.prerequisites

If prerequisite not in courseNumbers

Print "Error: Prerequisite " + prerequisite + " for course " + course.courseNumber + " does not exist."

Remove prerequisite from course.prerequisites

End If

End For

End For

End Function

Function printCourseInfo(String courseNumber)

If courseNumber not in courses

Print "Course " + courseNumber + " not found."

Return

End If

Course course = courses[courseNumber]

Print "Course Number: " + course.courseNumber

Print "Course Name: " + course.courseName

Print "Prerequisites: "

If course.prerequisites.size > 0

For each prerequisite in course.prerequisites

Print " " + prerequisite

End For

Else

Print " None"

End If

End Function

STRUCT Course

STRING courseNumber

STRING title

LIST STRING prerequisites

END STRUCT

STRUCT TreeNode

Course course

TreeNode left

TreeNode right

END STRUCT

STRUCT BinarySearchTree

TreeNode root

END STRUCT

FUNCTION LoadDataIntoTree(filePath, BST):

OPEN file at filePath FOR reading

CREATE a dictionary to store courses by courseNumber

WHILE NOT end of file:

READ line from file

SPLIT line by comma into fields

IF number of fields < 2:

PRINT "Error: Line does not contain enough fields"

CONTINUE TO next line

EXTRACT courseNumber, title, and prerequisites from fields

CREATE new Course object with extracted data

IF courseNumber is already in dictionary:

PRINT "Error: Duplicate courseNumber found: " + courseNumber

CONTINUE TO next line

ADD courseNumber and Course object to dictionary

CLOSE file

FOR EACH course in dictionary values:

CALL InsertIntoBST(BST, course)

END FUNCTION

FUNCTION InsertIntoBST(BST, course):

CREATE new TreeNode with course

IF BST.root is NULL:

SET BST.root to new TreeNode

ELSE:

CALL InsertNode(BST.root, new TreeNode)

END FUNCTION

FUNCTION InsertNode(currentNode, newNode):

IF newNode.course.courseNumber < currentNode.course.courseNumber:

IF currentNode.left is NULL:

SET currentNode.left to newNode

ELSE:

CALL InsertNode(currentNode.left, newNode)

ELSE IF newNode.course.courseNumber > currentNode.course.courseNumber:

IF currentNode.right is NULL:

SET currentNode.right to newNode

ELSE:

CALL InsertNode(currentNode.right, newNode)

ELSE:

PRINT "Error: Duplicate courseNumber found during insertion: " + newNode.course.courseNumber

END FUNCTION

FUNCTION PrintCourseInfo(BST, courseNumber):

CALL SearchBST(BST.root, courseNumber)

END FUNCTION

FUNCTION SearchBST(node, courseNumber):

IF node is NULL:

PRINT "Course not found: " + courseNumber

RETURN

IF courseNumber == node.course.courseNumber:

PRINT "Course Number: " + node.course.courseNumber

PRINT "Title: " + node.course.title

PRINT "Prerequisites: " + JOIN node.course.prerequisites with ", "

ELSE IF courseNumber < node.course.courseNumber:

CALL SearchBST(node.left, courseNumber)

ELSE:

CALL SearchBST(node.right, courseNumber)

END FUNCTION

Runtime Analysis Chart

| **Operation** | **Vector** | **Hash Table** | **Binary Search Tree (BST)** |
| --- | --- | --- | --- |
| **Memory Usage** | O(n) | O(n) | O(n) |
| **Insertion** | O(1) (amortized) | O(1) average, O(n) worst | O(log n) average, O(n) worst |
| **Search** | O(n) | O(1) average, O(n) worst | O(log n) average, O(n) worst |
| **Prerequisite Validation** | O(n^2) | O(n) | O(n log n) |
| **Overall Complexity** | O(n^2) | O(n) | O(n log n) |

Advantages and Disadvantages Analysis

| **Data Structure** | **Advantages** | **Disadvantages** |
| --- | --- | --- |
| **Vector** | - Simple to implement  - Fast iteration  - Cache-friendly due to contiguous memory | - Slow search operations (O(n))  - Inefficient memory usage when resized frequently  - Linear-time validation for prerequisites (O(n^2)) |
| **Hash Table** | - Fast insertion and search (O(1) on average)  - Efficient prerequisite validation (O(n))  - Handles large datasets well | - Performance can degrade to O(n) in worst-case scenarios due to collisions  - Requires careful management of hash table size  - No inherent order in elements |
| **Binary Search Tree (BST)** | - Efficient search, insertion, and deletion when balanced (O(log n))  - Maintains sorted order  - Good for range queries and ordered data | - Risk of becoming unbalanced, leading to O(n) performance  - Requires balancing mechanisms (e.g., AVL or Red-Black Tree)  - More complex to implement and manage |

The Hash Table offers the best overall performance with O(1) average-case time complexity for most operations, making it ideal for efficiently handling the course data. Despite potential collision issues, a well-designed hash function can minimize these, and the benefits far outweigh the disadvantages.