Yakuba Conde

CS-300-17233-M01 DSA: Analysis and Design 2024 C-6 (Oct - Dec)

Dec 08, 2024

Pseudocode and Runtime Analysis for ABCU Course Management System

**Introduction**

This document explains how the ABCU Course Management System will handle course data using three structures: vector, hash table, and binary search tree. It includes pseudocode, runtime analysis, and a recommendation for the best approach.

**Pseudocode**

**Vector**

Here’s how the system works with a vector. It’s simple but not the fastest for some tasks.

// Load courses into vector  
function loadData(filePath, vector<Course> &courses) {  
 file = open(filePath)  
 if file is null {  
 print "File not found."  
 return  
 }  
 while (line = readLine(file)) {  
 tokens = split(line, ",")  
 if length(tokens) < 2 continue  
 Course course  
 course.courseNumber = tokens[0]  
 course.name = tokens[1]  
 for i from 2 to length(tokens) - 1 course.prerequisites.push\_back(tokens[i])  
 courses.push\_back(course)  
 }  
 close(file)  
}  
  
// Display course details  
function displayCourse(vector<Course> courses, courseNumber) {  
 for course in courses {  
 if course.courseNumber == courseNumber {  
 print course details and prerequisites  
 return  
 }  
 }  
 print "Course not found."  
}

**Hash Table**

The hash table speeds up lookups but doesn’t keep things in order.

// Load courses into hash table  
function loadData(filePath, hashTable<Course> &courses) {  
 file = open(filePath)  
 if file is null {  
 print "File not found."  
 return  
 }  
 while (line = readLine(file)) {  
 tokens = split(line, ",")  
 if length(tokens) < 2 continue  
 Course course  
 course.courseNumber = tokens[0]  
 course.name = tokens[1]  
 for i from 2 to length(tokens) - 1 course.prerequisites.push\_back(tokens[i])  
 hashTable[course.courseNumber] = course  
 }  
 close(file)  
}  
  
// Display course details  
function displayCourse(hashTable<Course> courses, courseNumber) {  
 if courseNumber in hashTable {  
 print course details and prerequisites  
 } else {  
 print "Course not found."  
 }  
}

**Binary Search Tree (BST)**

BST keeps courses sorted and is efficient for searching, but it’s more complex.

// Insert course into BST  
function insertCourse(TreeNode &root, Course course) {  
 if root is null root = new TreeNode(course)  
 else if course.courseNumber < root.course.courseNumber insertCourse(root.left, course)  
 else insertCourse(root.right, course)  
}  
  
// Search and display course  
function searchCourse(TreeNode root, courseNumber) {  
 if root is null print "Course not found."  
 else if courseNumber == root.course.courseNumber print course details  
 else if courseNumber < root.course.courseNumber searchCourse(root.left, courseNumber)  
 else searchCourse(root.right, courseNumber)  
}  
  
// Print all courses in order  
function traverseInOrder(TreeNode root) {  
 if root is not null {  
 traverseInOrder(root.left)  
 print course details  
 traverseInOrder(root.right)  
 }  
}

**Runtime Analysis**

Here’s how the three structures compare in terms of speed. The table shows worst-case runtimes.

| Operation | Vector | Hash Table | BST |  
|----------------|--------------|--------------|---------------------|  
| Load Data | O(n) | O(n) | O(n log n) |  
| Search Course | O(n) | O(1) | O(log n) |  
| Print Courses | O(n log n)\* | O(n)\*\* | O(n) |  
  
**Vector** requires sorting first. **Hash table** needs traversal and sorting.

**Recommendation**

The BST is the best choice. It keeps courses sorted and is efficient for searching and printing. While the hash table is faster for lookups, it doesn’t maintain order, which is important here.