Juan Aviles

06/15/2025

CS-300

6-2 Project One

Vector Pseudocode

Int numPrerequisiteCourse(Vector<Course> courses, Course c){

totalPrereq = prerequisites of course c

for each prerequisite p in totalPrereq

add prerequisites of p to totalPrereq

Print number of totalPreq

}

Void printSampleSchedule(Vector<Course> courses) {

For all courses print course name

if(course has prerequisites)each prerequisite

print prerequisite

)

Void printCourseInformation(Vector<Course> courses, String courseNumber){

For all courses

if the course is the same as courseNumber

print out the course information

for each prerequisite of the course

print the prerequisite course information }

Hash table Pseudocode

Int numPrerequisiteCourses(Hashtable courses, Course c){

TotalPrereq = Hashtable[c]

For each prerequisite p in totalPrereq

Add prerequisites in Hashtable[p] to totalPrereq

Print number of totalPrereq

}

Void printSampleSchedule(hashtable, courses){

For all key value pair in courses

print key course name

if value has prerequisite

for each prerequisite

print prerequisite

}

Void printCourseInformation(hashtable courses, string courseNumber){

For all courses

if course is same as courseNumber

print the course information

for each prerequisite of hashtable[course]

print prerequisite course info

}

Tree pseudocode

OpenFile()

Initialize BinaryTree

Initialize ifstream instream

Open file with fileName

if file opens

Continue

else

Print “file is not opening”

exit

Check file

If two params

if int and string correct order

Continue parsing list

else

print “invalid params”

exit

CourseCreate()

String Course

int courseNum

open file

while open

If node > courseNum

if left node = nullptr

left node = new node

else

this add node(left node, courseNum)

else

if right node = nullptr

right node = new node

else

this add node(right node, courseNum)

close file

return

SearchCourse()

Open file

While open

int courseNumSearch

input courseNumSearch

courseNumSearch

current node = root

while current node doesnt have nullptr

if current node = courseNum

return current course

if course id is < current courseNum

Traverse left subtree

else

traverse right subtree

return bid

PrintCourse()

If node != nullptr

inOrder left node

Output course info

inOrder right node

Else

return

Menu

Print menu:

add course

delete course

check course info and prerequisite

exit

MenuSelection

Selection = 0

If option == 1

Cin course name

Cin course number

Else if == 2

Cin course to delete

Parse file

If delete course = course name

delete course

Else

print “course not available”

Else if == 3

Parse file

If courseNumSearch == courseNum

print course info

Else if == 4

“Bye”

exit

Runtime Analysis

Vector

void printCourseInfo(Vector<Course>courses, String courseNum){

for all course

if the course is the same as courseNum

print out the course info

for each prerequisite of the course

print the prerequisite course info

}

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | # executes | cost |
| For all courses | 1 | n | N |
| If course is same as courseNum | 1 | n | n |
| Print course info | 1 | 1 | 1 |
| For each prerequisite of couse | 1 | n | n |
| Prerequisite course info | 1 | n | n |
| Total cost | | | 4n+1 |
| Runtime | | | O(n) |

Hashtable

Void printCourseInfo(hashtable courses, string courseNum){

for all courses

if course is same as courseNum

print the course info

for each prerequisite of hashtable[course]

print prerequisite course info

}

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line cost | #times | Total |
| For all courses | 1 | n | n |
| Course same as courseNum | 1 | n | n |
| Print out course info | 1 | 1 | 1 |
| For each prerequisite of the hashtable[course] | 1 | n | n |
| Prerequisite course info | 1 | n | n |
| Total cost | | | 4n-1 |
| Runtime | | | O(n)\* |

Binary Search

OPEN file "course\_data.txt"  
IF file does not open:  
 PRINT "Error: Could not open file."  
 EXIT

CREATE empty list "course\_list" // To store all course numbers for prerequisite validation  
CREATE BinarySearchTree bst // Binary Search Tree for storing courses

WHILE not end of file:  
 READ next line  
 SPLIT line by commas into course\_data[]  
   
 // Validate the format of each line  
 IF course\_data[] length is less than 2:  
 PRINT "Error: Invalid format. Each line must have at least a course number and title."  
 CONTINUE // Skip to the next line if format is incorrect  
   
 // Extract course data  
 course\_number = course\_data[0]  
 course\_title = course\_data[1]  
 prerequisites = course\_data[2:] // Remaining items are prerequisites, can be empty  
   
 // Add course number to the course\_list for later prerequisite validation  
 ADD course\_number to "course\_list"  
   
 // Create a new course object  
 CREATE new Course object  
 SET Course.course\_number = course\_number  
 SET Course.course\_title = course\_title  
 SET Course.prerequisites = prerequisites  
   
 // Insert the course object into the binary search tree  
 bst.INSERT(new Course)

// After processing all lines, validate that all prerequisites exist in the course\_list  
FOR each course in bst:  
 FOR each prerequisite in course.prerequisites:  
 IF prerequisite not in "course\_list":  
 PRINT "Error: Prerequisite", prerequisite, "for course", course.course\_number, "is not a valid course."

// Define a Course structure to hold each course's information  
DEFINE struct Course:  
 course\_number // Unique course number  
 course\_title // Course title  
 prerequisites = [] // List of prerequisite course numbers (could be empty)

// Insert each Course object into a binary search tree (BST)  
FUNCTION bst.INSERT(new\_course):  
 IF bst is empty:  
 SET root = new\_course  
 ELSE:  
 CALL bst\_insert\_recursive(new\_course, root)  
   
FUNCTION bst\_insert\_recursive(new\_course, current\_node):  
 IF new\_course.course\_number < current\_node.course\_number:  
 IF current\_node.left is NULL:  
 current\_node.left = new\_course  
 ELSE:  
 CALL bst\_insert\_recursive(new\_course, current\_node.left)  
 ELSE:  
 IF current\_node.right is NULL:  
 current\_node.right = new\_course  
 ELSE:  
 CALL bst\_insert\_recursive(new\_course, current\_node.right)

// Print course information and prerequisites using in-order traversal  
FUNCTION printCourses(BinarySearchTree bst):  
 IF bst is empty:  
 PRINT "No courses available."  
 ELSE:  
 CALL bst.INORDER\_TRAVERSAL(printCourse)

FUNCTION bst.INORDER\_TRAVERSAL(current\_node, printCourse):  
 IF current\_node is not NULL:  
 CALL bst.INORDER\_TRAVERSAL(current\_node.left, printCourse)  
 CALL printCourse(current\_node)  
 CALL bst.INORDER\_TRAVERSAL(current\_node.right, printCourse)

FUNCTION printCourse(Course c):  
 PRINT "Course Number:", c.course\_number  
 PRINT "Course Title:", c.course\_title  
 IF c.prerequisites is not empty:  
 PRINT "Prerequisites:", c.prerequisites  
 ELSE:  
 PRINT "No prerequisites."

|  |  |  |  |
| --- | --- | --- | --- |
| code | Line cost | #times | Total cost |
| For all nodes | 1 | n | n |
| Course same as courseNum | 1 | n | n |
| Print node info | 1 | 1 | 1 |
| Print left node | 1 | n | n |
| Course has right node | 1 | n/2 | n/2 |
| Prereq info | 1 | 1 | 1 |
| Total cost | | | 2(n/2)+3n+2 |
| runtime | | | O(n)\* |

Summary

Creating the pseudocode and analyzing the runtimes, I decided to go with the binary search. While hashtable is fastest for direct lookup, it requires additional steps for displaying. A vector is not as efficient for frequent lookups.