Project One

Joshua Shults

SNHU

CS 300: Analysis and Design

Instructor Name

10/13/2024

Project One

PSUEDOCODE

//Psuedocode for the Menu

Void main() {

//Initialize all the variables for the differing data structures

Vector<course> coursesVector = new Vector<Course>()

HashTable<String, Course> courseHashTable = new HashTable<String, Course>()

TreeNode root = null

//Now initialize the list, but with the change to request which data table the user would like to use

WHILE TRUE{

Print “Menu”

Print “Enter a choice (1, 2, 3, 9)”

Print “1. Load the course data.”

Print “2. Print all courses in alphanumeric order”

Print “3. Search for a course and print its information”

Print “9. Exit the program.”

String choice = Get user’s next input

IF choice == “1”{

Print “Select a data structure to load the list into: “

Print “1. Vector”

Print “2. Hash Table”

Print “3. Binary Search Tree”

String choiceStructure = Get user’s next input

If choiceStructure == “1”{

loadCourseData(coursesVector, “courses.csv”)

}

ELSE IF choiceStructure == “2” {

loadCourseData(courseHashTable, “courses.csv”)

}

ELSE IF choiceStructure == “3” {

loadCourseData(root, “courses.csv”

}

IF choice == “2” {

If choiceStructure == “1” {

printCourses(coursesVector)

If choiceStructure == “2”

printCourses(courseHashTable)

If choiceStructure == “3”

inOrderTraversal(root)

IF choice == “3”

Print “Enter a courseID to search for: “

String courseID = Get user’s next input

IF choiceStructure == “1”

searchAndPrintCourse (coursesVector, courseID)

IF choiceStructure == “2”

searchAndPrintCourse (coursesHashTable, courseID)

IF choiceStructure == “3”

searchAndPrintCourse (root, courseID)

IF choice == “9”

Print “Exiting program.”

Break

ELSE{

Print “Invalid option, please choose from the options on the Menu.”

//Funciton to print courses alphanumerically with the Vector

Void printCourses(Vector<Course> courses) {

//sort the vector by each courseID

Sort(courses, courseID)

For each course in courses {

Print “Course ID: “ + course.courseID

Print “Course Name: “ + course.courseName

IF course.prerequisites == 0:

Print “No prerequisites”

Else

Print “Prerequisites: “

For each prerequisite in prerequisitesList

Print prerequisite + “ “

//Function to print courses alphanumerically with a Hash Table

void printCourses (unordered\_map<String, Course>& courses)

//create a temporary vector to store the courses to be sorted

Vector<Course> courseList

//extract the courses from the hash table into this vector

FOR (const auto& pair : courses) {

courseLIst.push\_back(pair.second)

}

//now sort the vector by courseID

Sort(courseList.begin(), courseLIst.end(), [](Course& a, Course& b)

Return a.courseID < b.courseID

//Print the sorted course list along with any prerequesites

For (const auto& course : courseList) {

Print “Course ID: “ + course.courseID

Print “Course Name: “ + course.courseName

//Print any prerequisites

If (!course.prerequisites.empty())

Print “Prerequisites: “

For (const auto& prereq : course.prerequisites) {

Print prereq + “ “

//Funciton to alphanumerically sort the BST

Void inOrderTraversal (TreeNode\* root)

If (root != nullptr)

//recursively sort the tree by going left

inOrderTraversal(root->left)

Print “Course ID: “ root->data.courseID

Print “Course Name: “ root->data.courseName

//print any prerequisites if they exist

If (!root->data.prerequisites.empty())

Print “Prerequisites: “

For (const auto& prereq : root->data.prerequisites)

Print prereq + “ “

Else

Print “No prerequisites.”

//recursively go right

InOrderTraversal(root->right)

**EVALUATION**

Vector

As for file reading and parsing with a vector data structure for every course reading and parsing the course ID, name, and their prerequisites will be done once per line. The cost of each of these operations is 0(1). Insertion into the vector is also simple push operations that are at a complexity of 0(1). The total cost for insertion from the csv file into a vector is O(n) with n being the number of courses.

Hash Table

Reading and parsing by course ID, name, and prerequisites occur n times, once for each line, and it has the same cost as the vector: O(1) for each parsed line. This means that the total cost for reading and parsing will be O(n) as well. Insertion into a hash table has a worst-case scenario due to collisions that can occur with hashing, but with a well written hash function the best case scenario would be O(1).

Binary Search Tree

For a binary search tree reading and parsing is also done “n” times, once per line read. The cost for splitting and parsing each line is also O(1). Insertion into a BST has a time complexity of O(log n), which means that in the worst case scenario the insertion time will be O(n^2).

**Advantages and disadvantages**

Vector Advantages:

A vector is a simple data structure with efficient memory usage, and for this program insertion would be fast at O(1) on average.

Vector Disadvantages:

Sorting the vector alphanumerically requires additional steps with a O(n log n) time complexity. Also using linear search would be slower while using a vector.

Hash Table Advantages:

A hash table would have fast insertion and search capabilities with the average complexity being O(1). Also, a hash table would have fast access to courses by the courseID.

Hash Table Disadvantages:

There are additional steps for alphanumerically sorting the Hash table with a complexity of O(n log n). Also, due to hashing collisions time complexity can be lengthened to O(n) instead of O(1) in the worst case.

BST advantages:

Courses are automatically sorted in order for the easiest scenario for alphanumeric sorting at a complexity of O(log n) due to recursively traversing the list. Also, searching the BST would be much quicker than a linear search due to a tree’s pointers.

BST disadvantages:

There could be a worst case insertion scenario with a time complexity of O(n).

**Recommendations**

For this program I would recommend using a Hash Table, as it provides the most efficient average time complexity when it comes to insertion and searching. This works with the advisor’s requirements for efficiently accessing course information and printing their information. The only real disadvantage of using the Hash Table is the additional step that is required to print courses in alphanumeric order, but as per the advisor’s requirements a Hash Table would still be the most efficient.