**Project One**

Southern New Hampshire University

CS-300: Data Structures & Algorithms

Instructor: Saba Jamaliannasrabadi

D.D. Sessions

August 5, 2023

**Previous Pseudocode Resubmission**

**// VECTOR:**

START

Call function 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

} ENDFOR

} ENDIF

} ENDFOR

}

Call function void loadFile(csvFileName) {

SET file equal to csvFileName

DECLARE vector<Course> courses

// Read data from file and parse each line

FOR all lines in file {

// Create course object

DECLARE Course course

DEFINE Vector<Prerequisite> prerequisite

// Check for format errors

IF course number and name exist on the line {

SET course.CourseId equal to courseId

SET course.CourseName equal to courseName

IF additional args exist(prerequisites) {

FOR all prerequisites {

IF prerequisite matches any of the course ids in the file {

prerequisite.push\_back(prerequisite courseId)

} ENDIF

} ENDFOR

} ENDIF

SET courses.Prerequisites equal to prerequisite

// Store object in data structure

INSERT courses.push\_back(course)

} ENDIF

} ENDFOR

}

END

**// HASH TABLE:**

START

Call function void printCourseInformation(String courseNumber) {

FOR loop to iterate all nodes beginning to end {

IF courseId is equal to courseNumber {

// cout << "Course ID: " << courseId << endl;

// cout << "Course Name: " << courseName << endl;

PRINT course information

IF course has prerequisites {

// cout << "Course Prerequisite(s): " << prerequisite << endl;

PRINT course prerequisite(s)

FOR each prerequisite of the course {

Call printCourseInformation with the current prerequisite courseId

} ENDFOR

} ENDIF

} ENDIF

} ENDFOR

}

Call function loadFile(String csvFileName, HashTable\* hashTable) {

// Open file

DECLARE fstream fileStream

OPEN csvFileName with fileStream

// Read data from file and parse each line

FOR all lines in fileStream {

// Create course object

DECLARE Course course

DEFINE Vector<Prerequisite> prerequisites

// Check for format errors

IF course number and name exist on the line {

SET course.CourseId equal to courseId

SET course.CourseName equal to CourseName

IF additional args exist(prerequisites) {

FOR all prerequisites {

IF prerequisite matches any of the course ids in the file {

INSERT prerequisite in prerequisites vector

} ENDIF

} ENDFOR

} ENDIF

} ENDIF

SET courses.Prerequisites equal to prerequisite

// Store object in data structure

INSERT course into hashTable

} ENDFOR

}

END

**// TREE**

START

DEFINE function Course searchCourse(String courseNumber) {

DECLARE current Node\* equal to root

// Keep looping downwards until bottom is reached or match is found

WHILE current is not null {

IF current->course.CourseId compared to courseNumber equals 0 {

RETURN current->course

ELSEIF current->course.CourseId compared to courseNumber less than 0 {

SET current equal to current->left

ELSE {

SET current equal to current->right

} ENDIFELSE

} ENDWHILE

DECLARE Course course

RETURN course

}

DEFINE function void printCourseInformation(String courseNumber) {

DECLARE BinarySearchTree bst

DECLARE Course course

SET course equal to CALL searchCourse(course.CourseId)

IF course.CourseId is not empty {

// cout << "Course ID: " << courseId << endl;

// cout << "Course Name: " << courseName << endl;

PRINT course information

IF course has prerequisites {

// cout << "Course Prerequisite(s): " << prerequisite << endl;

PRINT "course prerequisite(s)"

FOR each prerequisite of the course {

Call printCourseInformation with the current prerequisite courseId

} ENDFOR

} ENDIF

} ENDIF

}

DEFINE function void loadFile(String csvFileName, BinarySearchTree\* bst) {

// Open file

DECLARE fstream fileStream

OPEN csvFileName with fileStream

// Read data from file and parse each line

FOR all lines in fileStream {

// Create course object

DECLARE Course course

DECLARE Vector<Prerequisite> prerequisites

// Check for format errors

IF course number and name exist on the line {

SET course.CourseId equal to courseId

SET course.CourseName equal to CourseName

IF additional args exist(prerequisites) {

FOR all prerequisites {

IF prerequisite matches any of the course ids in the file {

INSERT prerequisite in prerequisites vector

} ENDIF

} ENDFOR

} ENDIF

} ENDIF

SET courses.Prerequisites equal to prerequisite

// Store object in data structure

INSERT course into bst

} ENDFOR

}

END

**Menu Pseudocode: Vector**

START

// Define vector data structure

DEFINE vector to hold all courses

SET integer input equal to 0

// Display menu

WHILE input is not equal to 9 {

PRINT Menu:

PRINT 1. Load Courses

PRINT 2. Display All Courses

PRINT 3. Find Course

PRINT 9. Exit

PRINT Enter choice

// Switch statement to handle input choice

SWITCH

// Load all courses

CASE 1:

Call function to load courses

PRINT course.size() courses read

BREAK

// Display all courses

CASE 2:

FOR loop to iterate course.size() {

PRINT all courses

} ENDFOR

BREAK

// Find a course

CASE 3:

FOR loop to iterate course.size() {

IF courseId equals current courseId {

PRINT courseId

} ENDIF

} ENDFOR

BREAK

} ENDWHILE

END

**Menu Pseudocode: Hash Table**

START

// Define hash table data structure

DEFINE HashTable to hold all courses

SET courseTable equal to new HashTable

SET integer input equal to 0

// Display menu

WHILE input is not equal to 9 {

PRINT Menu:

PRINT 1. Load Courses

PRINT 2. Display All Courses

PRINT 3. Find Course

PRINT 9. Exit

PRINT Enter choice

// Switch statement to handle input choice

SWITCH

// Load all courses

CASE 1:

Call function to load courses

PRINT number of courses read

BREAK

// Display all courses

CASE 2:

Call the courseTable PrintAll() function

BREAK

// Find a course

CASE 3:

IF the courseId it not empty {

PRINT courseId

}

ELSE {

PRINT courseId not found

} ENDIFELSE

BREAK

} ENDWHILE

END

**Menu Pseudocode: Tree**

START

// Define tree data structure

DEFINE BinarySearchTree bst to hold all courses

SET bst equal to new BinarySearchTree

SET integer input equal to 0

// Display menu

WHILE input is not equal to 9 {

PRINT Menu:

PRINT 1. Load Courses

PRINT 2. Display All Courses

PRINT 3. Find Course

PRINT 9. Exit

PRINT Enter choice

// Switch statement to handle input choice

SWITCH

// Load all courses

CASE 1:

Call function to load courses

PRINT number of courses read

BREAK

// Display all courses

CASE 2:

Call the bst InOrder function

BREAK

// Find a course

CASE 3:

SET course equal to bst Search courseKey

IF the courseId it not empty {

PRINT courseId

}

ELSE {

PRINT courseKey not found

} ENDIFELSE

BREAK

} ENDWHILE

END

**Alphanumeric Order Course List Pseudocode: Vector**

START

AlphaNumSort(vector<Course>& courses) {

DEFINE integer min

DEFINE size equal to courses size // number of courses in list

FOR size\_t i equal to 0; i < size - 1; ++I {

SET min equal to i

FOR size\_t j = i + 1; j < size - 1; ++j {

IF compare courses[j].courseId to courses[min].courseId is less than 0 {

SET min equal to j

} ENDIF

} ENDFOR

IF min not equal to I {

swap courses at I with courses at min

} ENDIF

} ENDFOR

FOR each course in courses {

Call printCourseInformation passing in course.courseId

} ENDFOR

}

END

**Alphanumeric Order Course List Pseudocode: Hash Table**

START

AlphaNumSort(){

DEFINE vector of courses

FOR loop to iterate all nodes beginning to end {

Pushback node to courses

} ENDFOR

DEFINE integer min

DEFINE size equal to courses size // number of courses in list

FOR size\_t i equal to 0; i < size - 1; ++i {

SET min equal to i

FOR size\_t j = i + 1; j < size - 1; ++j {

IF compare courses[j].courseId to courses[min].courseId is less than 0 {

SET min equal to j

} ENDIF

} ENDFOR

IF min not equal to i {

Swap courses at i with courses at min

} ENDIF

} ENDFOR

FOR each course in courses {

CALL printCourseInformation passing in course.courseId

} ENDFOR

}

END

**Alphanumeric Order Course List Pseudocode: Tree**

START

void BinarySearchTree::InOrder() {

CALL inOrder passing in root

}

void BinarySearchTree::inOrder(Node\* node) {

IF node not equal to nullptr {

// Recursively call in order not left

Call inOrder passing node->left

// Print course information

Call printCourseInformation passing in node->course.courseId

// Recursively call in order right

Call inOrderpassing node->right

} ENDIF

}

AlphaNumSort() {

CALL InOrder method

}

END

**Run-Time Analysis Chart**

**VECTOR:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | 1 | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n2) |

**HASH TABLE:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | 1 | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n2) |

**TREE:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | 1 | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |

**Advantages & Disadvantages of Each Data Structure**

There are advantages and disadvantages to vectors, hash tables, and binary trees. Advantages of vectors are that you can easily add items to or remove items from the data structure, and an element of the vector can be accessed quickly by its index. Disadvantages are that it can be slow to insert or delete elements, and they require contiguous memory allocation which can lead to wasted memory. Some advantages of hash tables include quick searches, insertions, and deletions. However, they can take up more space than other structures, and you may run into resizing and null value issues. Binary trees store nodes with a key and a value. You can efficiently traverse, search, and sort within a tree. Some drawbacks include poor performance and bulkier code due to requiring pointers.

**Data Structure Recommendation**

In this instance dealing with course information in the Computer Science department at ABCU, I would recommend a hash table data structure because it is fast to search, insert, and delete items. Using a hash table would allow you to map course numbers to objects associated with that course, and you could efficiently search for a course number.