Cory Doak

Southern New Hampshire University

CS300: Analysis and Design

October 1, 2022

Module 6: Project 1

**// Vector pseudocode**

//Open Files

OpenFiles(two params) {

File = open (filename)

While file is not null

Line = readline(file)

Parse each line

Check for course title and ID number

//Check for errors

if file is correct {

check what prereqs are for course

}

//Create Course Objects Function

CreateCourse (parameters for creation) {

//variables for courses and read files

Initialize course and read file variables;

Open file;

Read file;

//Store course object

While file is open {

Course object stored in vector;

}

}

//Course Search Function and Print out Course Info

CourseSearch () {

//variables to open file

Initialize open file variables;

Open file;

//Output course info and store in vector

While file is open {

Output course info;

Data stored in vector;

}

}

**// HashTable Pseudocode**

//Open Files

Open file;

Read file;

Parse each line;

//Check for course info

Check for course title

If course title found {

Check for course id;

//if course title and id found check for prereq

If course id found {

Check for prereq for course;

//if prereq present then add to array

If prereq is present {

Add to array;

}

//check for errors

If course parameters is less than 2 {

Skip course;

Print format error message;

End;

}

Else {

add course info (name, number, and prereq) to hash table;

}

//if prerequisite exists and comes before course, add to hash table

If prereq is present{

If prereq is vefore course {

Add to hash table;

}

Else {

If prereq is null {

Skip course;

Print error message;

}

}

}

//Create Course Function

//initialize course variables and read

Initialize course variables;

Open file;

Read file;

While file open {

Store information in data structure;

}

//Output Course Information Function

Initialize variables to open file;

Open file;

//output course info

While file is open {

for all courses {

if course matches courseID {

print course info;

for each HashTable[course] {

print prereq course info;

}

}

}

}

**// Binary Search Tree Pseudocode**

//Total quantity of Course Prerequisites Function

numCoursePrereqs(Tree<Course> courses Node n) {

//Variable for # of prereqs

totalNumPrereqs = all children to Node n (left and right)

for each prereq “p” in totalNumPrereqs {

//add all nodes to get # of prerequisites

add right and left Nodes of node “p” to totalNumPrereqs

//Print Total of prerequisites

print quantity of totalNumPrereqs

}

}

//Output Course Schedule Function

printCourseSchedule(Tree<Course> courses) {

for all Nodes that are courses {

print the name of the course;

//printing the left and right node prereqs (if present)

if course has a node to the left{

print this node as a course prereq;

if course has a node to the right

print this node as a course prereq;

}

}

}

//Output Course Information Function

printCourseInfo(Tree<Course> courses, String course ID) {

for all present nodes {

if the course number and the course match (==) {

\\printing the course prereqs (if present)

Print the information for that course prereq

\\printing the course prereqs (if present)

If course has a node to the left

Print this node as course prereq and its information;

If course has a node to the right

Print this node as course prereq and its information;

}

Else {

If left node for course is present

Move to left node;

If right node for course is present

Move to right node;

}

}

}

**//Menu Pseudocode**

//Displaying menu for user

WHILE user choice != 9 {

//display menu options 1-3 and 9

Output << “Choose an option: “

Output << “1. Load Data”

Output << “2. Print Course List”

Output << “3. Print Course”

Output << “9. Exit”

//functions for each choice

Switch for menu

//user chooses 1

If user chooses 1{

Course Data is printed;

}

//user chooses 2

If user chooses 2{

Course information is printed;

}

//user chooses 3

If user chooses 3 {

Output << “Enter course ID: “

User input >> course ID;

Output << “course information (number, title, and prereqs)”

}

//user chooses 9

If user chooses 9 {

Output << “Goodbye!”

End program;

}

**//Printing Sorted List**

//Vector Sorting

//create sorting string with s param

sortingString( string s)

//implement partition method

//initialize first and last element

low is first element in list.

High is last element in list.

//find midpoint

Midpoint is low+ (high – low) / 2.

//set pivot to midpoint

Pivot = midpoint;

While pivot < high {

High -1;

}

//Swap values to left of pivot is lower and right of pivot is higher.

Use temporary val to swap low.

Set low to high;

Set high to temp;

//Quick sort

Mid is equal to 0;

Low is start;

High is finish;

If start is greater or equal to finish {

Return;

}

//implement low finish partition

Quick sort with recursive call;

quickSort(courses array, low index, low index of end)

quickSort(courses array, low index of end + 1, high index)

//Output display

Ouutput << course ID << “ : “ << course name << “ | “ << prereq << new line;

//loop vector to output course info;

For (int i = 0; i < courses.size; ++i){

Call course display method;

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Vector Code** | **Line Cost** | **# of Executions** | **Total Cost** |
| For all courses | 1 | n | n |
| If course == courseID | 1 | n | n |
| Output course info | 2 | 1 | 1 |
| For each prereq for course | 1 | N | n |
| Output prereq info | 2 | N | n |
| **Total Cost** | | | 5n + 1 |
| **Run Time** | | | O(n) |

NOTES:

Vector data structure has total cost of 5n+1 and a run time of O(n).

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Code** | **Line Cost** | **# of Executions** | **Total Cost** |
| for all courses | 2 | N | N |
| If course == coursed | 1 | N | N |
| Output course info | 1 | 1 | 1 |
| For each prereq for course | 2 | N | n |
| Output prereq info | 4 | N | n |

|  |  |
| --- | --- |
| **Total Cost** | 9n + 1 |
| **Run Time** | O(n) |

NOTES:

Hash Table data structure has a total of 9n+1 and a run time of O(n).

|  |  |  |  |
| --- | --- | --- | --- |
| **Tree Code** | **Line Cost** | **# of Execution** | **Total Cost** |
| for all courses | 1 | N | N |
| If course == coursed | 1 | N | N |
| Output course info | 2 | 1 | 1 |
| For each prereq for course | 1 | N | N |
| Output prereq info | 4 | N | n |

|  |  |
| --- | --- |
| **Total Cost** | 7(n) +1 |
| **Run Time** | O(n) |

NOTES:

Tree data structure has a total 7n+1 and a run time of O(n).

**Evaluation:**

For our given assignment, we want to list the courses required as well as the prerequisite courses needed for these courses. We are determined to find the most efficient way possible to ensure satisfaction and reduce memory use by testing out different data structures (Vector, Tree, and Hash Table). Every data structure has a function that gives the same results. Each of these data structures have their own advantages and disadvantages and are preferred over the other depending on the functionality of the application. We have been able to analyze and determine what is best for us with the given information.

First, we looked at vector data structures and were able to see that the advantage of using a vector structure is that it is straightforward. Although with arrays you set the length, it is possible to resize it to better fit more information if needed. In this case, we found that using the vector method gave us a total line cost and run time of 5n+1. Although it had the fastest run time, there is a disadvantage to using the vector method when it comes to searching for specific courses. This method will search each item in the array until it finds its match which could be time costly depending on length.

Another data structure we considered was a hash table. With hash tables we map out ‘keys’ to ‘values’. An advantage to using hash tables, unlike the vector method, is the search function. Finding information within the structure is easier, however it does require slower implementing. In our assignment we needed to print the courses in alphanumeric order, and this is where our disadvantage came in. To do so, each item needed to be pulled out, sorted through, and then finally printed. After analyzing this method, we found that the line cost and run time was 9n+1.

Finally, we tested out a binary tree method. Now, in my opinion it is more visually appealing on paper. But we are not working with paper are we. Binary trees consist of a root and nodes with, as the name (bi-) mentions, no more than two children per node. Binary trees are quicker when it comes to searching compared to the vector method. But will require more lines of code due to its nodes and directions. The searching will search a node and determine there if it needs to continue to one or the other children from there. The line cost and run time for this methods analysis gave us a 7n+1. With this information given, it is evident that a vector method would be best to utilize when printing course and prerequisite course information. It showed to have the quickest run time compared to the other methods, which is what we are looking for with this analysis.