Scott Rowe

Professor Dewin Andujar

CS-300

June 12, 2022

Pseudocode

Vector structure

Find course

Open courseFile with fstream

Create 2 function parameters

Open courseFile, read the data, THEN parse lines

THEN conduct a samples check for courseName and courseTitle

IF samples check has been complete without errors

THEN check for coursePrerequisites in courseFile

Void <Course> courses( string courseName){

Add variables to courses, THEN read courseFile

Open courseFile and read again

WHILE the courseFile is open, THEN store the courseName objective in a Vector data structure

}

Search courseNumber() {

Input variables to open courseFile

WHILE courseFile is open, THEN output courseInfo

THEN store that courseInfo in a data structure

}

Hash Structure

Open courseFile with fstream

Open courseFile, read the data, THEN parse lines

THEN conduct a samples check for courseName and courseTitle

IF samples check has been complete without errors

THEN check for coursePrerequisites in courseFile

IF coursePrerequisites found, THEN add to array

IF the course parameters are less than 2, THEN skip course

DISPLAY message ERROR: not formatted correctly, THEN End find

ELSE, Add courseName, courseNumber, And coursePrerequisites to Hash Table

IF, coursePrerequisites are found, THEN check where they start, ADD to hashTable

IF, coursePrerequisites not found, THEN skip course, DISPLAY message error

Void <Course> courses (string courseName) {

Add variables to courses, THEN read courseFile

Open courseFile and read again

WHILE the courseFile is open, THEN store the courseName objective in a hash table structure

}

Search courseNumber () {

Input variables to open courseFile

WHILE courseFile is open, THEN output courseInfo

THEN store that courseInfo in a data structure

}

Binary Tree Structure

CONSTRUCT nodes for left and right,

SET root variable to NULL,

CONSTRUCT variables for courseName, THEN integers for courseNumber

OPEN, courseFile, WHILE courseFile is open {

Read the DATA, parse each of the lines,

Void <Course> courses( string courseName){

THEN check for courseTitle and courseNumber,

IF, root is not equal to NULL

THEN, check for numPrerequisiteCourse

ADD numPrerequisiteCourse to the right

IF the courseParameters are than 2

ADD numPrerequisiteCourse to the left

Displayed error message

THEN, ADD courseName, courseNumber and numPrerequistiteCourse to the right

}

THEN read courseFile, close courseFile

Open courseFile and read again

WHILE the courseFile is open, THEN store the courseName objective in a Binary Tree data structure

}

Search courseNumber() {

Input variables to open courseFile

WHILE courseFile is open, THEN output courseInfo

THEN store that courseInfo in a data structure

}

**Menu**

CONSTRUCT an integer for a SWITCH () statement,

THEN, SET to 0 with the name courseInput;

INPUT a course variable for the loop,

WHILE, courseInput does not equal 4;

OUTPUT, Menu;

OUTPUT, 1. Load Data Structure;

OUTPUT, 2. Course List;

OUTPUT, 3. Course;

OUTPUT, 4. Exit;

SWITCH(courseInput)

Case 1:

loadData(course):

break;

Case 2:

Display Course List:

break;

Case 3:

Display Course;

Break;

Case 4:

Display message: “Thank you for choosing”

End course find;

Break;

Default:

OUTPUT: No input was found from User

Break;

**Pseudocode to print out the list of course alphanumeric order**

CONSTRUCT a sorting string that has a parameter string of R,

THEN, use Char that SETS length to ADD 1,

CONSTRUCT string to character array, THEN sort array,

CONSRUCT, 2 integers that’ll be used for Alphabetical and Number,

WHILE, loop for the Alphabetical integer when it’s less than 97,

SET, Alphabetical integer to ADD 1,

THEN, a FOR loop,

IF, the i is less than 97, THEN the number integer should be SET to ADD 1,

ELSE, SET the Alphabetical integer to ADD 1, THEN to Return to R

CONDUCT a string for the classFiles and OUTPUT courses to a Alphabetical order

END

**Runtime Analysis**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | 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) |

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

| **Binary Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all Node courses** | 1 | n | N |
| **if the course is the same as courseNumber** | 1 | n | N |
| **print out the Node course information** | 1 | 1 | 1 |
| **for each left Node course** | 1 | n | N |
| **print the prerequisite course information** | 1 | n | n |
| **for each right Node course** | 1 | n | N |
| **print the prerequisite course information** | 1 | n | N |
| **Total Cost** | | | 6n + 1 |
| **Runtime** | | | O(logn) |

**Evaluation**

When using a Vector Structure, it’s one dimensional and that might work for the course files. You can change the size, have one or more objects stored, and you can delete them whenever. The disadvantage would be the consumption of memory.

When using a Hash Table Structure, synchronization is the main advantage of this. They are also more efficient than search trees and they are used a lot in many kinds of computer software’s. The speed in lookup is very fast as well.

The disadvantage is the collisions when there is a large set of keys to be hashing, so the less the better.

When using a Binary Search Tree, the advantage is that has a faster insertion and deletion, etc. when it is balanced. A big plus in efficiency and the code is easier than link lists. Can implement order statistics and do a range query to find keys between N and M. That would be great for the alphabetic order.

The disadvantage is that depending on the shape of the tree searching can take longer.

Now for the choice I would make in choosing which code to use, I think I would go with a hash table because of the synchronization and faster than binary trees. I know that the disadvantage is the collisions but with these courses it’s not many to cause a collision. The binary sounds like it would be a good idea with order but if it’s not balanced that could cause difficulty in searches, it would be my number 2. So, for final analysis I am going with the Hash Table to give me that faster time to reduce time in finding the courses needed for students.

References

<https://www.tutorialspoint.com/advantages-of-vector-over-the-array-in-cplusplus>

<https://practice.geeksforgeeks.org/problems/advantages-and-disadvantages-of-hash-table>

<https://practice.geeksforgeeks.org/problems/advantages-and-disadvantages-of-bst>