Adam Benoit

7/17/2022

CS 300 DSA: Analysis and Design

6-2 Project One

**Vector Data Structure Pseudocode**:

Create Course name variable as a string

Create Course number variable as a string

Create Course prerequisite variable as a string

Load csv file of courses

Create a vector called courses and set size to number of lines in csv file

Have each line set to break at the commas for parameters

Verify that each line has at least 2 parameters

If line has prerequisite

Verify course number matches to a line course number at the beginning of the line

For each line

Assign the csvfile i, 0 to course number

Assign the csvfile i, 1 to course name

If line has 3 parameters

Assign the cvsfile i, 2 to prerequisite

Store in vector

Create a function search

Show on screen please enter course number

Get input from user

Search vector for course number

If course number in vector

Print to screen course information

For each prerequisite of the course

Print course information

**Hash Table Data Structure Pseudocode:**

Create Course name variable as a string

Create Course number variable as a string

Create Course prerequisite variable as a string

Load csv file of courses

Create a hash table called courses and set size to number of lines in csv file

Have each line set to break at the commas for parameters

Verify that each line has at least 2 parameters

If line has prerequisite

Verify course number matches to a line course number at the beginning of the line

For each line

Assign the csvfile i, 0 to course number

Assign the csvfile i, 1 to course name

If line has 3 parameters

Assign the cvsfile i, 2 to prerequisite

Store in hash table bucket

Create a function for printing

Search hash table for course numbers and names

Print to screen course information

For each prerequisite of the course

Print course information

**Tree Data Structure Pseudocode:**

Create Course name variable as a string

Create Course number variable as a string

Create Course prerequisite variable as a string

Load csv file of courses

Create a tree data structure called courses and set root to first input

Have each line set to break at the commas for parameters

Verify that each line has at least 2 parameters

If line has prerequisite

Verify course number matches to a line course number at the beginning of the line

For each line

Assign the csvfile i, 0 to course number

Assign the csvfile i, 1 to course name

If line has 3 parameters

Assign the cvsfile i, 2 to prerequisite

While lines remain

If course number is less than current node

Move to left sub tree

Else (meaning if course number is greater)

move to right subtree

Create a function for printing

Search tree data structure for course numbers and names

Print to screen course information

For each prerequisite of the course

Print course information

**Menu:**

While option is not equal to 4

Print to screen 1 load courses

Print to screen 2 print courses

Print to screen 3 Search for a course

Print to screen

Case 1

Load file of courses listed into data structure

Case 2

Pull from print function to print courses in alphanumeric order

Case 3

Print to screen “Enter course number”

Take input from user

If course number is true

Print course information including prerequisites information

Else

Print course not found

Print to screen good bye and close program

**Alphanumeric order Pseudocode:**

Pull from data structure function for variables

Partition function with int begin and end

Set low high and midpoint to 0

Create Boolean done and set to false

Have midpoint calculate the midpoint of the function

While done is false

While low course number are less than midpoint course number

increment low courses

While high course number are less than midpoint course number

decrement high courses

IF high course number are more than or equal to low course number

Done is true

ELSE

swap low and high course number

move high and low closer

return high

Print sorted list to screen of courses in order in order

**Vector:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Course name variable as a string** | **1** | **1** | **1** |
| **Create Course number variable as a string** | **1** | **1** | **1** |
| **Create Course prerequisite variable as a string** | **1** | **1** | **1** |
| **Load csv file of courses** | **1** | **1** | **1** |
| **Create a vector called courses and set size to number of lines in csv file** | **1** | **1** | **1** |
| **Have each line set to break at the commas for parameters** | **1** | **n** | **N** |
| **Verify that each line has at least 2 parameters** | **1** | **n** | **N** |
| **If line has prerequisite** | **1** | **n** | **N** |
| **Verify course number matches to a line course number at the beginning of the line** | **1** | **n** | **N** |
| For each line | 1 | n | N |
| Assign the csvfile i, 0 to course number  Assign the csvfile i, 1 to course name | 2 | n | 2N |
| If line has 3 parameters | 1 | n | N |
| Assign the cvsfile i, 2 to prerequisite | 1 | N | n |
| Store in vector | 1 | n | N |
| Create a function search | 1 | 1 | 1 |
| Search vector for course number | 1 | n | N |
| If course number in vector | 1 | n | N |
| Print to screen course information  For each prerequisite of the course  Print course information | 3 | n | 3n |
| **Total Cost** | | | 15n + 6 |
| **Runtime** | | | O(n) |

**Hash Table:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Course name variable as a string** | **1** | **1** | **1** |
| **Create Course number variable as a string** | **1** | **1** | **1** |
| **Create Course prerequisite variable as a string** | **1** | **1** | **1** |
| **Load csv file of courses** | **1** | **1** | **1** |
| **Create a hash table called courses and set size to number of lines in csv file** | **1** | **1** | **1** |
| **Have each line set to break at the commas for parameters** | **1** | **n** | **n** |
| **Verify that each line has at least 2 parameters** | **1** | **n** | **n** |
| **If line has prerequisite** | **1** | **n** | **n** |
| Verify course number matches to a line course number at the beginning of the line | 1 | n | n |
| For each line | 1 | n | n |
| **Assign the csvfile i, 0 to course number**  **Assign the csvfile i, 1 to course name** | 2 | n | 2n |
| **If line has 3 parameters** | 1 | n | n |
| Assign the cvsfile i, 2 to prerequisite | 1 | n | n |
| Store in hash table bucket | 1 | n | n |
| Create a function for printing | 1 | 1 | 1 |
| Search hash table for course numbers and names | 1 | n | n |
| Print to screen course information  For each prerequisite of the course  Print course information | 3 | 1 | 3 |
| **Total Cost** | | | 10n + 9 |
| **Runtime** | | | O(1) |

**Tree:**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create Course name variable as a string** | **1** | **1** | **1** |
| **Create Course number variable as a string** | **1** | **1** | **1** |
| **Create Course prerequisite variable as a string** | **1** | **1** | **1** |
| **Load csv file of courses** | **1** | **1** | **1** |
| **Create a tree data structure called courses and set root to first input** | **1** | **1** | **1** |
| **Have each line set to break at the commas for parameters** | **1** | **n** | **n** |
| **Verify that each line has at least 2 parameters** | **1** | **n** | **n** |
| **If line has prerequisite** | **1** | **n** | **n** |
| **Verify course number matches to a line course number at the beginning of the line** | **1** | **n** | **n** |
| **For each line** | **1** | **n** | **n** |
| **Assign the csvfile i, 0 to course number**  **Assign the csvfile i, 1 to course name** | **2** | **n** | **2n** |
| **If line has 3 parameters** | **1** | **n** | **n** |
| **Assign the cvsfile i, 2 to prerequisite** | **1** | **n** | **n** |
| **While lines remain** | **1** | **n** | **n** |
| **If course number is less than current node** | **1** | **n** | **n** |
| **Move to left sub tree** | **1** | **n** | **n** |
| Else (meaning if course number is greater) | 1 | n | n |
| move to right subtree | 1 | n | n |
| Create a function for printing | 1 | 1 | 1 |
| Search tree data structure for course numbers and names | 1 | n | n |
| Print to screen course information  For each prerequisite of the course  Print course information | 3 | 1 | 3 |
| **Total Cost** | | | 15n + 9 |
| **Runtime** | | | O(log(n)) |

The vector data structure has a couple advantages and disadvantages when compared the other data structures. Vectors are easy to implement and consume low memory. Since for this project we are just dealing with courses a vector may be good to use. With little change and adjustments with courses the disadvantages I will list may not be a problem with this project. A downside of using vectors is it is easier to remove from the outside of the vector. Meaning the beginning and the end. Also, another downside when searching the vector is searched from the beginning to the end which can be time consuming compared to the others. So, with larger databases this would be to time consuming in comparison to the other options. Also, you can only add to the end of a vector then would have to run a sorting function to rearrange them in order. Another disadvantage would eb that the size of the vector has to be declared at the beginning when it is created. Meaning to add to it you would have to adjust the size first.

The hash table data structure has a couple advantages and disadvantages when compared the other data structures. Hash tables are able to store large amounts of data and keep their speed constant when searching unlike the other 2. This means overall each search will take the same amount time. Another advantage of the hash table is that compared to the other two data structures inserting and deleting new data into the structure is much easier and faster. A disadvantage of using a has table is that more than one value can have the same slot in the table resulting in a collision. The greater the size or amount of data being stored in the hash table increases the rate at which collisions occur. Which can eventually lead to the database degrading overtime and eventually not being functional.

The binary tree structure has a couple advantages and disadvantages when compared the other data structures. One of the advantageous of the data tree is that it automatically sorts data into lower and higher. For this it is better for the data tree to get data that is already out of order. If the file contains data already in order from low to high or high to low this would create a tree that is just a straight line from the root going down to the left or the right. Trees also have and advantage with adding aspects to the tree. If the school had to adjust the curriculum to include additional classes the tree would auto sort into the right branch. But a disadvantage would be that the tree is more complicated at removing nodes from the tree and adjusting the after the node is removed. The tree is also quick at finding a particular courses since they are already sorted in the tree when searching for a particular course it would start at the root and follow a less or more process when searching moving to the left for a lower value or moving to the left when the value is greater than the current value.

Out of the data structures we have been introduced to I would suggest using the hash table data structures. One of the reasons I am choosing the hash table is that according to my tables it has the lowest total cost to implement. Also, with it being a constant speed it would be faster than the vector method at there worse. Meaning the vector would have to search all the way through checking each section of the vector until it got to the last one making it, its worse time. While in the other hand the hash table would perform a constant time for all searches. It is also easier to modify the hash table meaning if in the future some classes need to be removed and replace by other classes the hash table allows for the best flexibility.