July Wellman

CS 300: Analysis and Design

Southern New Hampshire University

February 17, 2024

**Project One**

*// Vector Data Structure Pseudocode*

Reading and validating the ABCU CSV Course file and prerequisites

Create a vector<Course> to store objects from parameters

Course Id equals parameter at element(0)

Course Name equals the parameter at element (1)

prerequisites equals any additional parameters that are read on the file()

Open the ABCU CSV Course file

For each line read on the CSV file:

Check if the line has two or more parameters

Split each parameter into elements with a “,”

For each prerequisite in prerequisites:

Add to prerequisites()

Else

Skip to the next line

End reading and validating the ABCU CSV file and prerequisites

Creating course objects and storing them in the appropriate data structure

Create an object called Course

For each parameter extracted:

Course Id equals parameter at element (0)

Course Name equals the parameter at element (1)

prerequisites equal any additional parameters that are read on the file and will be appended to the end of the list ()

Return the object in the format (course ID, Course Name, prerequisites())

Append by adding the object as an element to the end of the list on the vector<Course>

End creating course objects and storing them in the appropriate data structure

Search and print course information

Take user input

For each course in the vector

If the course ID matches the user input

Print course information

Print course Id, course title

If there are any prerequisites

Print the prerequisites

Else

Check the next node

If the course number does not match the user input

Print an error message “Invalid Course ID, please try again.”

End searching the data structure for a course and printing out course information and prerequisites

*// Vector Data Structure Run Time Analysis*

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For each line read on the CSV file | 1 | n | n |
| Check if the line has two or more parameters | 1 | n | n |
| Split each parameter into elements with a “,” | 1 | n | n |
| For each prerequisite in prerequisites: | 1 | n | n |
| Add to prerequisites() for the course object | 1 | n | n |
| Append the course object to the end of the list | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

*// Vector Data Structure: Advantages and Disadvantages*

Using a vector is memory efficient because vectors allow access to particular elements. Because vectors have a single block design, they require no complicated traversal system such as hash tables or search trees, making them simpler to use and implement. Using a vector is less desirable in the case you frequently add or remove elements, and it may be costly to navigate the list to modify elements in the middle. The size of vectors is set at the moment of their creation. Resizing the vector requires time and memory resources.

*// Hash Table Data Structure Pseudocode*

Reading and validating the ABCU CSV Course file and prerequisites

Create a vector<Course> to store objects from parameters

Course Id equals parameter at element(0)

Course Name equals the parameter at element (1)

prerequisites equals any additional parameters that are read on the file()

Open the ABCU CSV Course file

For each line read on the CSV file:

Check if the line has two or more parameters

Split each parameter into elements with a “,”

For each prerequisite in prerequisites:

Add to prerequisites()

Else

Skip to the next line

End reading and validating the ABCU CSV file and prerequisites

Creating course objects and storing them in the data structure

Create a new hash table<Course>

Initialize a size table to store the course objects as keys

Declare an unsigned int key

Initialize a node pointer to the next node

Set a variable to store the course id as course Id(0)

Set a variable to store the course name as course Name(1)

Set a variable to store the prerequisites as prerequisites()

Set a function that maps a course number to a bucket on the list

For each course object created from the CSV file

If the current node is empty

Create a new node with the course object as the data node

Else

Go to the next node

End creating course objects and storing them in the data structure

Search and print course information

Take user input

Set user input to node pointer

For each node in the hash table

If the course ID matches the node pointer

Print course information in the format (course id, course name, prerequisite())

If no perquisites

Print course information in the format (course id, course name)

If the course number does not match the user input

Print an error message “Invalid course Id, please try again.”

End search and print course information

*// Hash Table Run Time Analysis*

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For each line read on the CSV file | 1 | n | n |
| Check if the line has two or more parameters | 1 | n | n |
| Split each parameter into elements with a “,” | 1 | 1 | 1 |
| For each prerequisite in prerequisites: | 1 | n | n |
| Add to prerequisites() for the course object | 1 | 1 | 1 |
| For each course object created from the CSV file | 1 | n | n |
| If the current node is empty | 1 | n | n |
| Create a new node with the course object as the data node | 1 | 1 | 1 |
| Go to the next node | 1 | n | n |
| **Total Cost** | | | 6n + 3 |
| **Runtime** | | | O(1) |

*// Hash Tables: Advantages and Disadvantages*

The benefit of a hash table is the table may be resized to hold more or fewer elements. Hash tables support large data sets and do not require key and value ordering with constant time complexity. The benefits of using a hash table are that hash tables aren't suitable for ordered lists and are difficult to implement. A disadvantage of a hash table is that it prohibits null values. The worst-case run time of a hash table will be 0 (n).

*// Binary Search Tree Pseudocode*

Reading and validating the ABCU CSV Course file and prerequisites

Open the ABCU CSV Course file

For each line read on the CSV file:

If the line has two or more parameters

Split each parameter into elements with a “,”

For each prerequisite in prerequisites:

Add to prerequisites()

Else

Skip to the next line

End reading and validating the ABCU CSV file and prerequisites

Create course objects and store them in the tree data structure

Construct the course in format (course id, course name, prerequisites)

Create the binary tree by assigning the root, left, and right nodes

Set all nodes to null

Start the pointer at the root node

For each course object created from the CSV file

Create the course object and add it to the tree

While the node is not empty

If the course id is less than the current node course id

Traverse to the left node

else

Traverse to the right node

End create course objects and store them in the tree data structure

Print out course information and prerequisites

Take input from the user for a course id

Set the node pointer to the root of the tree

While the current node is not null

If the current course id matches the input course id

Print the course details

If the course id is smaller than the current node course id

Traverse to the left node

Else

Traverse to the right node

Store the course details

*// Binary Search Tree Run Time Analysis*

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| For each line read on the CSV file | 1 | n | n |
| Check if the line has two or more parameters | 1 | n | n |
| Split each parameter into elements with a “,” | 1 | 1 | 1 |
| For each prerequisite in prerequisites: | 1 | n | n |
| Add to prerequisites() for the course object | 1 | 1 | 1 |
| For each course object created from the CSV file | 1 | n | n |
| While the node is not empty | 1 | n | n |
| If the course id is less than the current node course id | 1 | n | n |
| Traverse to the left node | 1 | Log n | Log n |
| Else traverse to the right node | 1 | Log n | Logn n |
| Store course object | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 2Logn + 3 |
| **Runtime** | | | O(log n) |

*// Binary Search Trees: Advantages and Disadvantages*

The benefits of binary search trees are they provide sorted access to data and can be used for searching, sorting, and indexing. A search tree can also be adapted to increasing and decreasing nodes. Disadvantages are that the tree can get unbalanced, and this causes memory and performance issues. The search tree may degrade to a 0 (n) in a worst-case runtime scenario.

*// Recommendation for which data structure to use to code ABCU Course Information Planner*

Based on all three data structures and the client's needs, I will recommend using a binary search tree data structure. The client wants the Computer Science course to be in alphanumeric order. A BST has this advantage in the sense that will sort the courses as the objects are being added to the tree allowing for efficiency. A vector will work just as well, a binary search tree will be more effective in searching due to 0(log n) which lets the program run for only half the size instead of the full size.

*// Pseudocode for the menu*

Display Menu

Create the Menu

Print “1. Load Data Structure”

Print “2. Print Course List”

Print “3. Print Course”

Print “9. Exit”

Load Data Structure

Case 1:

Read form Course Information CSV file

While reading each line

If parameters are two or more

Parse with “,”

Else

Skip

While line read contains prerequisite

Add to prerequisites()

Insert node to binary search tree

Else

Print “Failed to load. Please try again.”

Print Course List

Case 2:

Take user input to print all of the courses in the search tree

Print all of the courses in the list from the binary search tree

Print Course

Case 3:

Take user input for Course Id

While the current Course Id is equal to the user input course Id

Print “Couse Id, Course Name”

Else if

Course Id is less than current course id

Traverse to the left

Else

Traverse to the right

Return Course ID

Exit

Case 4:

Take user input to exit

Break and close the program

*// Pseudocode for printing out the list of courses*

Point the pointer to start at the root and print the root node

Print course node to the left

Print the course node to the right

**References**

Olawanle, J. (October 5, 2022). *Big O Cheat Sheet – Time Complexity Chart*. Free Code Camp.

[Big O Cheat Sheet – Time Complexity Chart (freecodecamp.org)](https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/)

(Accessed February 17, 2024). *What does Constant Time Complexity or Big 0(1) mean?*. GeeksforGeeks.org.

[What does Constant Time Complexity or Big O(1) mean? - GeeksforGeeks](https://www.geeksforgeeks.org/what-does-constant-time-complexity-or-big-o1-mean/)

(Accessed February 17, 2024). *Applications, Advantages and Disadvantages of Hash Data Structure*. GeeksforGeeks.org.

[Applications, Advantages and Disadvantages of Hash Data Structure - GeeksforGeeks](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-hash-data-structure/)