Tyler Pimental

CS-300

Professor Ostrowski

10/08/2023

Project 1

While reviewing this project and past work, I have chosen to review the pseudocode I have previously submitted and adjusted some in a few areas. I also have adjusted the format and line spacing to hopefully make it easier to read.

//Read from file for all datatypes

Open inputfile

While file exists

Read line in inputfile

While line in inputfile is not end

count values in line

If values is less than or equal to 1 Then

Return Error

If values equals 2

Return value

If values is greater than 2

For each value

If value does not exist

Return Error

Else

Return value

//Initialize Course Data Structure

Create construct course object

Set courseid, coursename, <vector> preqs

//Vector Data Structure for creating course object

Insert vector element function

Append value to course vector

// Hash table Data structure for creating course object

Create new hash table

Run hash table insert function

Set hash of key to id

If id is not in table

Create node from ID

Else

Add ID to table

// Binary Tree for creating course object

Create new node class

Define attributes

Node left

Node right

Run addnode function

Construct courseobj

If root is null

Create root node

Else determine if current node is less than new course

Create a left node

Else

Create a right node

// Vector Print course information

Get courseid

For each course in courses

IF cousrse.id is equal to courseid

print course

For each course.prerequisite

Print course

// Hash Table Print course information

Get courseid

Run hashid to return key

Iterate through hashtable until key matches

Print course

For each course.prerequisite

Print course

// Binary Tree Print course information

Get courseid

Run search function

Set current node to root

While the current node is not null

if the current node is requested node

return course information

if current node is less than requested node

traverse the left

Else

Traverse the right

// Menu pseudocode

Get input

If input is equal to ‘Load Data’

Set dataloaded bool to true

Run readfile function

Else if input is equal to ‘Print Course List’

If dataloaded bool is true then

Run print sorted function

Else

Return Error

Else if input is equal to ‘Print Coures

If dataloaded bool is true then

print requested course

Else

Return Error

Else

Exit

// Vector Print sorted course list using insert sort

For elements in vector

Set tempvar to current element

While tempvar charcode is less than next element charcode

swap the two

// Binary Tree sorted course list

If the current node is not a null pointer

Call this function again with the the left node

Output the node information

Call this function again with the right node

|  |  |  |  |
| --- | --- | --- | --- |
| **Code analysis for Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Read File** | 1 | n | n |
| **Insert Vector** | 1 | n | n |
| **Append Value to course** | 1 | n | n |
| **Total Cost** | | | 3n |
| **Runtime** | | | O(n) |
| **Code analysis for Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create hash table** | 1 | 1 | 1 |
| **Insert Function** | 1 | n | n |
| **Set hash** | 1 | n | n |
| **Search ID** | 1 | n | n |
| **Add ID to table** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |
| **Code ode analysis for Binary Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| **Create node class** | 1 | 1 | 1 |
| **Define attributes** | 1 | 1 | 1 |
| **Addnode Function** | 1 | n | n |
| **Construct Courseobj** | 1 | n | n |
| **Create left node** | 1 | n | n |
| **Create Root node** | 1 | n | n |
| **Create left Node** | 1 | n | n |
| **Total Cost** | | | 5n + 2 |
| **Runtime** | | | O(n) |

The vector data structure is the most simplistic of the three we analyzed. Through this data structure, you can access any arbitrary element with the same time and complexity. This makes it fast for searching, however all elements are independent of each other, meaning in a worst cast you would have to check every element.

The hash table data structure builds off the vector one previously analyzed. We have the added feature of being able to find the location of an element simply by knowing its value. This becomes useful when sorting unique elements. However, we do have to account for possible collisions, which adds additional execution costs to our code.

Our last data structure being binary trees, is an entirely different model. We can by design sort our elements. This makes displaying an alphanumerically sorted list easy, since we can simply traverse our tree in a specified way. However, the code introduces some complexity, for searching arbitrary data, it can become time and memory consuming.

My choice for a data structure will be a simple vector. All three data structures came out to O(n) runtime, leading me to rely on self confidence and current knowledge of the data structures. I find the vector’s data structure to have fastest access times and least amount of separate executions.