Austin Henley

CS-300

4/6/2023

CS300 Project 1 – ABCU Academics

**Previous Pseudocode with Analysis**

**Opening File and Verifying**

Open File

If open file returns -1

Return file not found error

For each line in file

Parse lines using comma as delimiter

If less than two values in line

Return missing information error

Else if more than two values in line

If prerequisite parameter not in courses

Return course does not exist error

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Open File** | 1 | 1 | 1 |
| **If open file returns -1** | 1 | 1 | 1 |
| **For each line in file** | 1 | n | n |
| **Parse lines using comma as delimiter** | 1 | n | n |
| **If less than two values in line** | 1 | n | n |
| **Else if more than two values in line** | 1 | n | n |
| **If prereq not in courses** | 1 | n | n |
| **Total Cost** | | | 5n + 2 |
| **Runtime** | | | O(n) |

**Vector – Create Course Objects**

Create new courses vector for course objects

For each line in file

Parse lines using comma as delimiter

If less than two values in line

Return missing information error

Else if more than two values in line

If pre-requisite parameter not in courses

Return pre-requisite course does not exist error

Create new Course object

Course courseNumber = line parameter 1

Course courseName = line parameter 2

While next parameter exists

Add parameter to prerequisite list in course object

Append course object to courses vector

Close file

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For each line in file** | 1 | n | n |
| **Parse line using comma delimiter** | 1 | n | n |
| **If less than two values** | 1 | n | n |
| **Else if more than two values** | 1 | n | n |
| **Create new course object** | 1 | n | n |
| **Course Number = line parameter 1** | 1 | n | n |
| **Course name = line parameter 2** | 1 | n | n |
| **While next parameter exists** | 1 | n | n |
| **Add parameter to prerequisite list** | 1 | n | N |
| **Append course object to courses vector** | 1 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 10n + 1 |
| **Runtime** | | | O(n) |

**Vector - Search for Course and print (using course ID as parameter)**

For each course in courses vector

If course ID parameter = course object -> courseNumber

Print out course information

For each prerequisite of the course

Print course information for prerequisite course

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For each course in courses vector** | 1 | n | n |
| **If course ID = course->courseNumber** | 1 | n | n |
| **Print out course information** | 1 | 1 | 1 |
| **For each prereq in course** | 1 | n | n |
| **Print course information for prereq course** | 1 | 1 | 1 |
| **Total cost** | | | 3n+2 |
| **Runtime** | | | O(n) |

**Hash Table – Create Course Objects**

For each line in file

Parse lines using comma as delimiter

If less than two values in line

Return missing information error

Else if more than two values in line

If pre-requisite parameter not in courses

Return pre-requisite course does not exist error

// Create course object

Create new Course object

Course courseNumber = line parameter 1

Course courseName = line parameter 2

While next parameter exists

Add parameter to prerequisite list in course object

// Insert course object into hash table

//Calculate the hash value of the course number

Initialize key = hash(course.courseNumber)

Initialize pointer currNode = Node at key index

While hash table node at key location is not used

currNode = next node

currNode->course = course

close file

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For each line in file** | 1 | n | n |
| **Parse line using comma delimiter** | 1 | n | n |
| **If less than two values** | 1 | n | n |
| **Else if more than two values** | 1 | n | n |
| **Create new course object** | 1 | n | n |
| **Course Number = line parameter 1** | 1 | n | n |
| **Course name = line parameter 2** | 1 | n | n |
| **While next parameter exists** | 1 | n | n |
| **Add parameter to prerequisite list** | 1 | n | n |
| **Calculate hash of course number** | 1 | n | n |
| **Initialize pointer currNode = Node at key index** | 1 | n | n |
| **While currNode is not used** | 1 | n | n |
| **currNode->course = course** | 1 | n | n |
| **Close file** | 1 | 1 | 1 |
| **Total Cost** | | | 13n + 1 |
| **Runtime** | | | O(n) |

**Search for Course and print (using course ID as parameter)**

Initialize key = hash of course ID

Initialize currNode pointer to node at key location

While currNode is not null pointer

If currNode -> course -> course ID == course ID parameter

Print course ID and course name

Print “Prerequisites:”

For each prerequisite

Print prerequisite course ID and name

currNode = currNode->next

if currNode is null pointer

Print “Course Not Found”

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize key = hash of course ID** | 1 | 1 | 1 |
| **Initialize currNode pointer to node at key location** | 1 | 1 | 1 |
| **While currNode is not null pointer** | 1 | n | n |
| **If currNode -> course -> course ID == course ID parameter** | 1 | n | n |
| **Print course ID and course name** | 1 | 1 | 1 |
| **Print “Prerequisites:”** | 1 | 1 | 1 |
| **For each prereq** | 1 | n | n |
| **Print prereq course ID and name** | 1 | n | n |
| **currNode = currNode->next** | 1 | n | n |
| **If currNode is null pointer** | 1 | 1 | 1 |
| **Print “Course not found”** | 1 | n | n |
| **Total Cost** | | | 6n+3 |
| **Runtime** | | | O(n) |

**Binary Search Tree – Create Course Objects**

For each line in file

Parse lines using comma as delimiter

If less than two values in line

Return missing information error

Else if more than two values in line

If pre-requisite parameter not in courses

Return pre-requisite course does not exist error

// Create course object

Create new Course object

Course courseNumber = line parameter 1

Course courseName = line parameter 2

While next parameter exists

Add parameter to prerequisite list in course object

// Insert course into tree

insertCourse(root, new Course)

Close file

//Method for inserting into the tree

InsertCourse(node, course)

If node is null

Create new node with course object and set it as the root node

Else If (course -> courseNumber) < (node -> course ->courseNumber)

If left node is null

Create new node with course object and set it as the left node of the current node

Else

Recursively call InsertCourse on left node and newCourse

Else

If right node is null

Create new node with course object and set it as the right node of the current node

Else

Recursively call InsertCourse on right node and newCourse

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **For each line in file** | **1** | **n** | **n** |
| **Parse lines using comma as delimiter** | **1** | **n** | **n** |
| **If less than two values in line** | **1** | **n** | **n** |
| **Return missing information error** | **1** | **1** | **1** |
| **Else if more than two values in line** | **1** | **n** | **n** |
| **If pre-requisite parameter not in courses** | **1** | **1** | **1** |
| **Return pre-requisite course does not exist error** | **1** | **1** | **1** |
| **Create new Course object** | **1** | **n** | **n** |
| **Course courseNumber = line parameter 1** | **1** | **n** | **n** |
| **Course courseName = line parameter 2** | **1** | **n** | **n** |
| **While next parameter exists** | **1** | **n** | **n** |
| **Add parameter to prerequisite list in course object** | **1** | **n** | **n** |
| **insertCourse(root, course)** | **logn** | **n** | **logn** |
| **Close file** | **1** | **1** | **1** |
| **Total Cost** | | | **9n+logn+4** |
| **Runtime** | | | **nlogn** |

| **Code for InsertCourse method\*** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **If node is null** | **1** | **1** | **1** |
| **Create new node with course object and set it as the root node** | **1** | **1** | **1** |
| **Else If (course -> courseNumber) < (node -> course ->courseNumber)** | **1** | **n-1** | **n-1** |
| **If left node is null** | **1** | **n-1** | **n-1** |
| **Create new node with course object and set it as the left node of the current node** | **1** | **n-1** | **n-1** |
| **Else** | **1** | **1** | **1** |
| **Recursively call InsertCourse on left node and newCourse** | **T(n-1)** | **n-1** | **(n-1)T(n-1)** |
| **Else** | **1** | **n-1** | **n-1** |
| **If right node is null** | **1** | **n-1** | **n-1** |
| **Create new node with course object and set it as the right node of the current node** | **1** | **n-1** | **n-1** |
| **Else** | **1** | **1** | **1** |
| **Recursively call InsertCourse on right node and newCourse** | **T(n-1)** | **n-1** | **(n-1)T(n-1)** |
| **Total Cost** | | | **2(n-1)T(n-1) + 6(n – 1) + 2** |
| **Runtime** | | | **O(nlogn)** |

**\**worst case assumed as n-1 depending on the order of inserted nodes, could result in one long linked list***

**Binary Search Tree - Search for Course (using course ID as parameter)**

Initialize currNode pointer to root node

While currNode is not null pointer

If currNode -> course -> course ID == course ID parameter

Print course ID and course name

For each prerequisite

Print “Prerequisites:”

Print prerequisite course ID and name

return

Else if course ID parameter < currNode -> course -> course ID

Set currNode to the left child of currNode

Else

Set currNode to the right child of currNode

if currNode is null pointer

Print “Course Not Found”"

| **Code (h = height of tree)** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Initialize currNode pointer to root** | 1 | 1 | 1 |
| **While currNode is not null pointer** | 1 | h+1 | h+1 |
| **If currNode -> course -> course ID == course ID parameter** | 1 | 1 | 1 |
| **Print course ID and course name** | 1 | 1 | 1 |
| **Print “Prerequisites:”** | 1 | 1 | 1 |
| **For each prereq** | 1 | 1 | 1 |
| **Print prerequisite course ID and name** | 1 | 1 | 1 |
| **Return** | 1 | 1 | 1 |
| **Else if course ID parameter < currNode -> course -> course ID** | 2 | h | h |
| **Set currNode to the left child of currNode** | 1 | h | h |
| **Else** | 1 | h | h |
| **Set currNode to the right child of currNode** | 1 | h | h |
| **If currNode is null pointer** | 1 | 1 | 1 |
| **Print “Course Not Found”** | 1 | 1 | 1 |
| **Save input in course search var** | 1 | 1 | 1 |
| **Total Cost** | | | 4h + (h+1) +10 |
| **Runtime** | | | O(logn) |

**Menu Pseudocode**

Display "Welcome to ABCU's Advising Program"

Initialize Boolean exit to false

While exit is false

Display "Please select an option:"

Display "1. Load Data Structure”

Display "2. Print Course List"

Display "3. Print Course"

Display "4. Exit"

Get user input as option

If option is 1

Load Data Structure from file

Display "Data Loaded Successfully”

Else if option is 2

Sort courses in Data Structure by course number in alphabetical order

Print all course names and IDs

Else if option is 3

Display "Please enter a course ID:”

Get user input as coursed

Print course information and prerequisites for coursed

Else if option is 4

Set exit to true

Else

Display "Invalid option. Please select a valid option."

**Printing Pseudocode**

**Vector Data structure**

Sort vector courses by alphanumeric course number from lowest to highest

For each course in courses:

Print course information

**Hash Table data structure**

Create an empty array of size n, where n is the number of courses in the hash table

// Since hash table is unordered, Extract data to an array

For each course in the hash table

hashIndex = calculateHash(course.number)

insert course into the array at index hashIndex

// now array can be sorted

Sort the array of courses by course number in alphanumeric order

For each course in the array

print course information

**Tree Data Structure:**

//Recursive method to print tree

traverseTree(node)

if node is not null

traverseTree(node.left)

print node course information

traverseTree(node.right)

//Utilizing method

traverseTree(rootNode)

**Analysis and Conclusion**

To meet the advisor's requirements, the vector, hash table, and tree data structures are all possible approaches to the needs of the program. Each data structure has its own advantages and disadvantages.

The vector is simple to implement and use and allows for easy iteration over the course list. It is good for small datasets. However, insertions and deletions can be slow, especially for large datasets. Searching can also take longer for large datasets. The vector is not ideal for frequently changing data.

The hash table provides fast access to elements and is good for large datasets. It is also easy to implement and use and is good for frequently changing data. However, the hash table can have high memory usage and collisions, which can slow down the search time. It is not good for iterating over the course list. However, if fast search and insert operations are needed and the dataset is relatively large, a hash table may be a better choice.

The tree allows for quick searches and insertions and is good for large datasets. It provides ordered access to data and is good for frequently changing data. However, the tree can have high memory usage and can be more complex to implement and use. Traversing the entire tree can also be slow, depending on the height and balance of the tree. In comparison to a hash table, a tree may have more nodes than a hash table has keys.

After analyzing the three data structures, a recommendation can be made for which data structure is the most suitable to be used in the code. Based on the Big O analysis results and analysis of the vector, hash table, and tree, the hash table is the data structure I will be using in this case.

The hash table provides fast access to elements, making it a good option for frequently changing data. Its runtime complexity of O(1) for average case operations and O(n) for worst case operations is favorable when compared to the O(n) time complexity of the vector and tree.

Although the hash table has its own set of disadvantages such as high memory usage and collisions, these can be mitigated through careful design and implementation. Additionally, the hash table is easier to implement and use, making it a practical option for this case.

In conclusion, based on the Big O analysis results and the analysis of the three data structures, the hash table is the most suitable data structure to be used in the code.