**6-2: Project One**

Winnie Kwong

Southern New Hampshire University

CS-300 DSA: Analysis and Design

Professor Griffith

August 5th, 2023

**Resubmit Pseudocode**

**Vector**

***Defining the Program***

START

Open files with fstream

Creating a function with two parameters

Call to open file, WHILE file is not empty

Read data

Parse each line

Check for course title and number

IF file format is free from errors

IF prerequisites exist

ELSE BREAK

ELSE put “error” to OUTPUT

Close file

END

***Course Objects***

START

Open file

Check if file was successfully opened

Loop through each line in the input file

Initialize variables for the course

Parse line and extract course data

Check for any prerequisites

IF prerequisites exist, put prerequisites to OUTPUT

ELSE put courses to OUTPUT

Close file

END

***Data Structure***

START

INPUT course number

Loop through vector

IF INPUT equates to course number

Print out the course information

IF prerequisites exist

Print prerequisites course information

ELSE display course cannot be found to OUTPUT

END

**Hash Table**

***Defining the Program***

START

Open files with fstream

Creating a function with two parameters

Call to open file, WHILE file is not empty

Read data

Parse each line

Check for course title and number

IF file format is free from errors

IF prerequisites exist

ELSE BREAK

ELSE put “error” to OUTPUT

Close file

END

***Course Objects***

START

Initialize Course Vector vector<Node> nodes

Create HashTable ClassCreate

Insert method to insert items to HashTable

Open file

Check if file was successfully opened

Loop through each line in the input file

FOR each line in file

FOR first and second value

Use pushback to add value to vector

IF third value exists

Add to current value with pushback

Insert method for each value

Close file

END

***Data Structure***

START

RETRIEVE INPUT

Assign INPUT to key

IF key is found

Print out the course information

IF prerequisites exist

Print prerequisites course information

ELSE display course cannot be found to OUTPUT

END

**Tree**

***Defining the Program***

START

Open files with fstream

Creating a function with two parameters

Call to open file, WHILE file is not empty

Read data

Parse each line

Check for course title and number

IF file format is free from errors

IF prerequisites exist

Add node as parent node

ELSE BREAK

ELSE put “error” to OUTPUT

Close file

END

***Create Course Objects and Store them into Data Structure***

START

Open file

Check if the file was successfully opened.

Loop through each line in the input file

Initialize variables for the course

Parse line and extract course data

Check for any prerequisites

IF prerequisites exist, put prerequisites to OUTPUT

ELSE put courses to OUTPUT

Close file

END

***Print Out Course Information and Prerequisites***

START

RETRIEVE INPUT

Assign INPUT to key

IF key is found

PRINT course information of current node

IF node has child

FOR each child

PRINT course information

ELSE BREAK

ELSE display course cannot be found to OUTPUT

END

**Pseudocode for a Menu**

START

Initialize choice for the menu option

WHILE choice does not equal 4

DISPLAY “1. Load Data Structure”

DISPLAY “2. Print Course List”

DISPLAY “3. Print Course”

DISPLAY “4. Exit”

INPUT choice for menu option

SWITCH (choice)

CASE 1:

Load the file data into the data structure.

BREAK

CASE 2:

PRINT an alphanumerically ordered list.

BREAK

CASE 3:

PRINT course title and the prerequisites.

BREAK

CASE 4:

EXIT program

DISPLAY “Goodbye.”

default:

DISPLAY “Invalid option, please try again.”

Return to menu option.

END

**Pseudocode that Prints Courses in the Alphanumeric Order**

**Vector**

START

Set void function to print courses with vector array

Use sort method for the courses

FOR each course

PRINT course information.

END

**Hash Table**

START

Set void function to print courses with hash table

Set sorted courses to be equivalent to sorted coursed in hash table

FOR each course in sorted courses

PRINT course information

Sorted coursed in hash table

Convert the hash table to an array of key-value pairs

Sort the course array based on keys of course numbers

Sorted hash table EQUALS a create new hash table function

FOR each pair of courses in the array

INSERT pair into a sorted hash table

RETURN sorted hash table

Use the array of key-value pairs to convert the courses in the array of key-value pairs

Set the course in the array into the array of key-value pairs

FOR each key-value pair in courses

APPEND pair to the course in the array

RETURN the course in the array

END

**Tree**

START

Set void function to print courses with binary search tree

Use in-order traversal to print courses in alphanumeric order

Set void function to execute inorder for nodes

IF node is null

RETURN

Traverse left subtree

PRINT course information

Traverse right subtree

END

**Evaluation**

| **Vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open files with fstream | 1 | n | n |
| Creating a function with two parameters | 1 | n | n |
| Call to open file, WHILE file is not empty | 1 | n | n |
| Parse each line | 1 | n | n |
| IF file format is free from errors | 1 | n^2 | n^2 |
| Loop through each line in the input file | 1 | n | n |
| Parse line and extract course data | 1 | n | n |
| IF prerequisites exist, put prerequisites to OUTPUT | 1 | n | n |
| Loop through vector | 1 | n | n |
| IF INPUT equates to course number | 1 | n^2 | n^2 |
| Print prerequisites course information | 1 | 1 | 1 |
| **Total Cost** | | | 2n^2+9n + 1 |
| **Runtime** | | | O( n^2) |

| **Hash Table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open files with fstream | 1 | n | n |
| Creating a function with two parameters | 1 | n | n |
| Call to open file, WHILE file is not empty | 1 | n | n |
| Parse each line | 1 | n | n |
| IF file format is free from errors | 1 | n^2 | n^2 |
| Insert method to insert items to HashTable | 1 | n | n |
| Loop through each line in the input file | 1 | n | n |
| FOR each line in file | 1 | n^2 | n^2 |
| Insert method for each value | 1 | n | n |
| IF key is found | 1 | n^2 | n^2 |
| Print out the course information | 1 | 1 | 1 |
| Print prerequisites course information | 1 | 1 | 1 |
| **Total Cost** | | | 3n^2+7n + 2 |
| **Runtime** | | | O(n^2) |

| **Tree** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Open files with fstream | 1 | n | n |
| Creating a function with two parameters | 1 | n | n |
| Call to open file, WHILE file is not empty | 1 | n | n |
| Parse each line | 1 | n | n |
| IF file format is free from errors | 1 | n^2 | n^2 |
| Add node as parent node | 1 | 1 | 1 |
| Loop through each line in the input file | 1 | n | n |
| Initialize variables for the course | 1 | n | n |
| IF prerequisites exist | 1 | n | n |
| Assign INPUT to key | 1 | n | n |
| IF key is found | 1 | log N | log N |
| PRINT course information | 1 | 1 | 1 |
|  |  |  |  |
|  |  |  |  |
| **Total Cost** | | | log N +n^2+8n+2 |
| **Runtime** | | | O(log N) |

**Advantage and Disadvantage of each Structure**

**Advantage and Disadvantage of Vector**

Vectors are known for their ability to store elements within a container. One of the most significant advantages of using a vector is its flexibility to increase or decrease size using memory allocation. Having flexibility enables the vector to have potential growth to fit the capacity of what needs to be stored. Also, since vectors use dynamic memory allocation, iterators make it easier to traverse elements. Since iterators provide the means to access data in a stored container class, it makes accessing the target element easier. Furthermore, vectors can use a built-in sorting function. The vector's sorting function helps reduce the program's complexity. With flexibility, iterators, and sorting advantages, vectors can manage changing data elements.

However, vectors also come with complications in performance and memory usage. Vectors are inefficient when inserting or deleting in the middle of a list. With vectors, the list would need to be split to add an element to the center, then create a new variable to combine both lists. Vectors also occupy a lot of memory. Even after elements are removed, the vector will retain its capacity, so it does not need to reallocate when an element is inserted constantly. Furthermore, with memory usage, because the capacity does not reduce, the memory usage will either remain the same or continue to grow. This means the cost would become expensive because of its continuous growth opportunity. With performance and memory usage issues, vectors may only be suitable for some list programs.

**Advantage and Disadvantage of Hash Table**

Hash tables provide an easier way to compute an index due to its operation handling, space efficiency, and flexibility. Hash tables are efficient when inserting and deleting elements because only the array index needs to update per operation. In this case, the hash table provides constant time to search, insert, and delete operations. Also, hash tables have excellent space efficiency to maintain data. Hash tables only store key-value pairs and the arrays that hold them, making it easier to handle data structures. Since hash tables can keep any data and objects, the hash table can be flexible. Flexibility within the hash tables enables the opportunity to complete simple and complex data structures. With its advantages of operation handling, space efficiency, and flexibility, the hash table makes it efficient when creating data structures.

Unfortunately, hash tables may not be suitable because of their inability to null values, limited capacity, and time deficiency. One critical issue is that hash tables cannot have null values because null is not an object. However, null values are essential because they indicate the database's nonexistent data values. Unlike vectors, hash tables have a limited capacity. With a limited capacity, hash tables will eventually max out, making it tedious and expensive to rebuild another hash table that should be doubled in size. Also, hash tables can need more time efficiency when creating, deleting, and calling within the project. Hash tables suffer from cache performance that makes it time deficient to iterate through each value within the bucket to locate the key. Users must be aware that hash tables hinder because of their inability to null values, limited capacity, and time deficiency can affect the program's performance.

**Advantage and Disadvantage of Binary Search Tree**

Binary search trees (BST) are data structures that can maintain a sorted list with fast time complexity, efficiency, and organization. With BST, when balanced, it can insert and delete nodes as other functions, such as searching quickly due to its O(log n). With time efficiency, less recourse consumption is used, as well as elevating the quality of the software. BST is also efficient when executed properly. BST was designed to make searching specific data; with each search, nodes are eliminated by half to reduce the search time. Since the structure of BST is organized alphabetically or numerically, it is always sorted. Having a data structure always managed to be sorted makes it easier to retrieve data. Having a fast data structure, time complexity, efficiency, and organization enhances the quality of the software.

Some complications also come with BTS, such as the effect of imbalanced trees, search deficiency, and memory usage. When trees are imbalanced, it may be costly for operation. The operation can avoid being logarithmic, and inefficient search operations will occur. As mentioned, the search algorithm is to cut nodes in half and travel to the bottom of the deepest leaf. However, as the BTS grows, search time becomes inadequate, going down one level at a time and uncovering that the search key may not even exist, wasting valuable time and resources. With BTS, it also requires a lot more memory to operate correctly. BTS requires each node to store two pointers on top of the initial element, making it difficult for users with limited memory. Having complications within BTS, such as the effects of an imbalanced tree, search time deficiency, and memory usage, can hinder the overall program, making it more complex than it needs to be.

**Recommended Data Structure**

The recommended data structure I plan to use is vector. The vector’s ability to resize allows opportunities to include more courses if needed with the course data file. From the Big O analysis, vector had the least cost and lowest runtime, meaning it would have the least time complexity in the algorithm as the input grows. Therefore, the vector would also be the simplest to execute among the hash table and binary search tree.