**Project One**

**Vector:**

Open file

If file is successfully opened

Read each line from the file

Parse the line into course number, title, and prerequisites

Validate line format

If number of parameters < 2

Print error message and skip the line

Else

Store course information and prerequisites in data structure

**Hashtable:**

Create a Hashtable class

Create a Node structure that holds information about a course

Define a constant variable called tableSize to determine the size of the hashtable

Class Hashtable

Node[] nodes

tableSize

Hashtable()

Create an empty array called nodes with size tableSize

Unsigned hash

Return the key

Void Insert(Course course)

Calculate the key for the given course

Calculate the index where the node should be placed using the hash

If the node at nodes is empty

Create a new node with the given course and key, and assign it to nodes

Else

Start from the node at nodes

While there is a node present

If the node's key matches the given key

Update the course information of the current node and exit

End If

Move to the next node

Create a new node with the given course and key

Assign the new node as the next node of the previous node

Void loadCourses(string csvPath, Hashtable hashTable)

For each row in the CSV file

Create a new course object

Assign the course ID from the current row to the course object

Assign the name from the current row to the course object

Assign the prerequisites from the current row to the course object's prerequisites

Insert the course object into the hashTable

End

**Tree:**

Open file

If file is successfully opened:

Initialize tree data structure

Read each line:

Parse the line.

If the number of parameters is less than 2:

Display an error, proceed to the next line.

Else:

Save the course in tree data structure.

For each prerequisite in course prerequisite:

If prerequisite is not present in the tree data structure:

Print error

Create a new course object

Assign the course number, title, and prerequisites to the corresponding fields of the course object.

Insert the course object into the tree data structure, using the course number as the key.

**Menu:**

Load course info:

Print course list:

Print course:

menu:

display "Menu:"

display "1. Load Course Info"

display "2. Print Course List"

display "3. Print Course"

display "4. Exit"

while true:

display "Enter your choice: "

choice = input

if choice is "1":

display "Enter the file name: "

File name = input

Load course info

display "Data structure loaded successfully."

else if choice is "2":

Print course list

else if choice is "3":

display "Enter the course code: "

Course code = input

Print course(course code)

else if choice is "4":

display "Exiting the program"

****Design pseudocode that will print out the list of the courses in the Computer Science program in alphanumeric order:****

**Vector:**

Initialize an empty vector called courses.

Iterate through each course in the Computer Science program:

Add the course to the courses.

Sort the course in alphanumeric order based on the course number.

Iterate through each course in the courses:

Print the course to the display.

**Hash Table:**

Initialize an empty hash table called course.

Iterate through each course in the Computer Science program

Add the course to the courses using the course number as the key.

Sort the keys of the courses in alphanumeric order.

Iterate through each key in the sorted keys of the courses:

Retrieve the course from the courses using the key.

Print the course to the display.

**Tree:**

Initialize an empty tree called courses.

Iterate through each course in the Computer Science program:

Insert the course into the courses using the course number as the key.

Traverse the courses in alphanumeric order

Print the course to the display.

**Vector:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open file | 1 | 1 | 1 |
| If file is successfully opened | 1 | 1 | 1 |
| Read each line from the file | 1 | n | n |
| Parse the line into course info | 1 | n | n |
| Validate line format | 1 | n | n |
| If number of parameters < 2 | 1 | n | n |
| Print error message and skip the line | 1 | n | n |
| Else | 1 | n | n |
| Store course information and prerequisites in data structure | 1 | n | n |
|  |  |  |  |
| Total Cost | | | 3n + 1 |
| Runtime | | | O(n) |

**Hashtable:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Hashtable | 1 | 1 | 1 |
| Unsigned hash | 1 | n | n |
| Insert course | 0 | 0 | 0 |
| Calculate the key for the given course | 1 | n | n |
| Calculate using hash | 1 | n | n |
| Else | 0 | 0 | 0 |
| While node is present | 1 | n | n |
| Update the course information of current node and exit. | 1 | n | n |
| Move to next node | 1 | n | n |
| Create new node with the course and key | 0 | 0 | 0 |
| Assign the new node as the next node of the previous | 1 | n | n |
| For each row in the CSV file | 1 | n | n |
| Create new course object | 1 | n | n |
| Assign course ID from the current to the course object | 1 | n | n |
| Assign the name from current row to the course object | 1 | n | n |
| Assign prerequisites from current to the course object prerequisites. | 1 | n | n |
| Insert course object into the hashtable | 1 | n | n |
| Total Cost | | | 6n + 2 |
| Runtime | | | O(n) |

**Tree:**

|  |  |  |  |
| --- | --- | --- | --- |
| Code | Line Cost | # Times Executes | Total Cost |
| Open file | 1 | 1 | 1 |
| If file is successfully opened | 1 | 1 | 1 |
| Initialize tree data structure | 1 | 1 | 1 |
| Read each line from the file | 1 | n | n |
| Parse the line into course info | 1 | n | n |
| If number of parameters < 2 | 1 | n | n |
| Print error message and skip the line | 1 | n | n |
| Else | 1 | n | n |
| Store course information and prerequisites in tree data structure | 1 | n | n |
| For each prerequisite in course prerequisite. | 1 | n | n |
| If prerequisite is not present in tree | 1 | n | n |
| Print error | 1 | n | n |
| Create new course | 1 | n | n |
| Assign the course number, title, and prerequisite | 1 | n | n |
| Insert the course into tree | 1 | n | n |
| Total Cost | | | 9n + 3 |
| Runtime | | | O(n) |

Each data structure has its own set of advantages and disadvantages. A vector is easy to implement and would be the best approach for its quick ability to read files and add course objects. Vectors come with a disadvantage. Being vectors need to search the complete list for a certain course by checking each course until a specific course is found. Hash tables create keys which allow the courses to be searched by the key. This is also a disadvantage of a hash table; hash tables cannot be sorted. Lastly, Binary trees are efficient at searching for a specific element. Binary trees have nodes with at the most two childs per node which is why it is efficient to search. Binary trees can have poor performance if the tree is not properly balanced. Based on the Big O analysis I would recommend using the vector method. Using a vector would allow for sorting the courses quickly and to print each course. The vector pros outweigh the cons of the other methods and would be the best option for the academic advisors in the Computer Science department at ABCU.