Alexandra Delarosa

CS 300 Project One

**Pseudocode for each data structure**

**Vector:**

Initialize empty vector to store courses

loop through file

split line into parameters

if number of parameters < 2

print error

continue to next line

initialize courseNumber as 1st parameter

initialize name as 2nd parameter

initialize prerequisites as remaining parameters

loop through prerequisites

if prerequisite not found

print error

continue to next line

Create new Course object and store data in vector

new course = Course(courseNumber, name, prerequisites)

append new course to courses vector

close file

Input a string courseNumber

For each course in courses vector

If courseNumber equals courseNumber

Print course information (courseNumber + name +prerequisites)

End loop

**Hash Table:**

Pseudocode for opening file:

open file

loop through file

split lines in file into parameters

if parameter < 2

print error

go to next line

set courseNumber to 1st parameter

set name to 2nd parameter

set prerequisite to last parameters

loop through prerequisites

if prerequisite does not exist in hash table

print error

go to next line

close file

Pseudocode for creating course object:

Initialize empty hash table

loop through lines in file

create new Course object

set course number and name to course object

add Course object to hash table

loop through prerequisites

add prerequisite to the course object required prerequisites

Pseudocode for printing course data:

Loop through courses in hash table

print course number

print course name

if course has prerequisite

loop through prerequisites

print prerequisites of the course

else

print no prerequisites

**Tree:**

loop each line in file

split lines into parameters

if # of parameters < 2

print error

go to next line

set courseNumber to 1st parameter

set name to 2nd parameter

set prerequisite to last parameters

loop through prerequisites from index 2

if prerequisite does not exist in tree’s course data

print error

go to next line

close file

Pseudocode for creating course object:

Initialize empty tree data structure to store courses

loop through lines in file

create new Course object

set course number and name to course object

add Course object to tree data structure

loop through prerequisites

add prerequisite to the course object required prerequisites

Pseudocode for printing course data:

Traverse tree data in order (in order, pre order, post order)

Loop through nodes in traversal

print course number

print course name

if course has prerequisite

loop through prerequisites

print prerequisites of the course

else

print no prerequisites

**Pseudocode for printing course data:**

Initialize empty data structure variable

function to load into data structure:

Open file

Loop through each line in the file

Parse lines to extract data (courseNumber, name, prerequisites)

Create a new Course object with the data

Load Course object into the data structure

Close file

function to print course list:

If data structure is empty

Print error

Else

Sort courses in data structure alphanumerically by courseNumber

Loop through each course in the data structure

Print courseNumber and name

End

function to print course(courseNumber):

If data structure is empty

Print error

Else

Loop to find course with matching courseNumber

If course == courseNumber

Print courseNumber and name of the course

If course has prerequisites

Loop through prerequisites of course

Print prerequisite courseNumber

Else

Print course not found

End function

**Display menu pseudocode**:

Loop until user chooses exit(4)

Print menu options:

1. Load Data Structure

2. Print Course List

3. Print Course

4. Exit

Prompt user to enter an option

If 1 is chosen

Call load data structure function

Else if 2 is chosen

Call print course list function

Else if 3 is chosen

Prompt user for a course number

Call print course(courseNumber) function

Else if 4 is chosen

Exit

Else

Print error try again

End loop

**Evaluation**

**Vector:**

for all courses: Cost = 1, # Time Executes = n, Total cost = n

if the course is the same as courseNumber: Cost = 1, # Time Executes = n, Total cost = n

print out the course information: Cost = 1, # Time Executes = 1, Total cost = 1

for each prerequisite of the course: Cost = 1, # Time Executes = n, Total cost = n

print the prerequisite course information: Cost = 1, # Time Executes = n, Total cost = n

Total Cost: 4n + 1

Runtime: O(n)

Advantages: Easy implementation, efficiency sequential access, cache friendly

Disadvantages: Insertion & deletion can be costly as it requires element shifting, not ideal for large amounts of data

**Hash Table:**

for all courses in hast table: Cost = 1, # Time Executes = n, Total cost = n

if the course is the same as courseNumber: Cost = 1, # Time Executes = n, Total cost = n

print out the course information: Cost = 1, # Time Executes = 1, Total cost = 1

for each prerequisite of the course: Cost = 1, # Time Executes = n, Total cost = n

print the prerequisite course information: Cost = 1, # Time Executes = n, Total cost = n

Total Cost: 4n + 1

Runtime: O(n)

Advantages: Efficient insertion and deletion, suitable for large number of items

Disadvantages: Potential for collisions can lower performance

**Tree**

Traverse tree in in-order : Cost = log(n), # Time Executes = n, Total cost = n log(n)  
 if the course is the same as courseNumber: Cost = 1, # Time Executes = 1, Total cost = 1  
 print out the course information: Cost = 1, # Time Executes = 1, Total cost = 1  
 for each prerequisite of the course: Cost = 1, # Time Executes = n, Total cost = n  
 print the prerequisite course information: Cost = 1, # Time Executes = n, Total cost = n  
Total Cost: n log(n) + 2  
Runtime: O(n log n)  
Advantages: Ordered structure, efficient searching, maintains data in sorted order

Disadvantages: Complex implementation, potential for imbalanced tree, requires additional memory for pointers

**Recommendation**

Based on my analysis, a hash table seems to be the most efficient data structure to be used in the code. It provides constant time insertion and retrieval of objects which makes searching and accessing data efficient. It has a linear run time for reading the file and searching through data. The tree data structure provides efficient searching, however the overhead of maintaining the tree structure is not ideal for this task. It is ideal for large data sets and while hash collisions may be a concern, but proper hashing techniques and collision handling can address this issue.