CS 300 Pseudocode Document

Function Signatures

Below are the function signatures that you can fill in to address each of the three program requirements using each of the data structures. The pseudocode for printing course information, if a vector is the data structure, is also given to you below (depicted in bold).

```
// Vector pseudocode
int numPrerequisiteCourses(Vector<Course> courses, Course c) {
      totalPrerequisites = prerequisites of course c
      for each prerequisite p in totalPrerequisites
            add prerequisites of p to totalPrerequisites
      print number of totalPrerequisites
}

Function openFile()

Open file from source
If the file does not open, display error message and exit the function
      While the file has lines
      Read and load the lines of data
            Split the lines into tokens
                  If the number of tokens is less than 2, display an
format error, continue to next line

                        Use the tokens to extract line course number,
title and prerequisites
                  If the line has perquisites
                        Check lines for matching course numbers to ensure
that the course exists as a course in the file
                        If no prerequisite exists in the course list
                  display error message

//Creating and storing course object
      While lines have no error message
            Create classObject from token information using
            courseObject constructor store in vector.


Close file

Class Course:
Create variables
      courseNumber
      title
      perquisites


void printSampleSchedule(Vector<Course> courses) {
```

```
}

void printCourseInformation(Vector<Course> courses, String
courseNumber) {
      for all courses
            if the course is the same as courseNumber
                  print out the course information
                  for each prerequisite of the course
                        print the prerequisite course information
}

// Hashtable pseudocode
int numPrerequisiteCourses(Hashtable<Course> courses) {

Function openFile()

Open file from source
If the file does not open, display error message and exit the function

Initialize an empty hash table to store course objects

While the file has lines
      Read and load the lines of data
      Split the lines into tokens

      If the number of tokens is less than 2, display an format error,
      continue to next line

      Use the tokens to extract line course number, title and
      prerequisites

      If the line has perquisites
            Check lines for matching course numbers to ensure that the
            course exists as a course in hash table
                  If no prerequisite exists in the course list display
                  error message

      //Creating and storing course object in the hash table
      While lines have no error message
            createCourse from token information using createCourse
            function
            Add course to the hash table


Close file

return the course hash table



Function createCourse()
```

```
      Create a new course object
      Set parameters (courseNumber, title and prerequisites)
      Return created course object


Class Course:
Create variables
      courseNumber
      title
      perquisites


}

void printSampleSchedule(Hashtable<Course> courses) {

}

void printCourseInformation(Hashtable<Course> courses, String
courseNumber) {
      for all courses
            get course number from hash table using the course
            number/key
            if the course is the same as courseNumber
                  print out the course information
                  for each prerequisite of the course
                        print the prerequisite course information
            if course is not found in the course table
                  print error message


}

// Tree pseudocode
int numPrerequisiteCourses(Tree<Course> courses) {

Function openFile()

Open file from source
If the file does not open, display error message and exit the function

Initialize an empty tree to store course objects

While the file has lines

      Read and load the lines of data
      Split the lines into tokens

      If the number of tokens is less than 2, display an format error,
      continue to next line
```

```
      Use the tokens to extract line course number, title and
      prerequisites

      If the line has perquisites
            Check lines for matching course numbers to ensure that the
            course exists as a course in tree
                  If no prerequisite exists in the course list display
                  error message

      //Creating and storing course object in the tree table
      While lines have no error message
            createCourse from token information using createCourse
            function
            Add course to the tree in order
      Traverse down the tree

Close file

return the course tree



}

void printSampleSchedule(Tree<Course> courses) {

}
void printCourseInformation(Tree<Course> courses, String courseNumber)
{
for all courses
      Traverse down tree in order
            get course number from tree using the course number/key
            if the course is the same as courseNumber
                  print out the course information
                  for each prerequisite of the course
                        print the prerequisite course information
            if course is not found in the course table
                  print error message


}

Function print course list(choose data structure)
      Sort data
      For each element in list
            Print element
      End function at end of list

Function search and print selected course(choose data structure)
      Use function to find course in data structure
            If course is found
```

```
                  Print the course and course information
      Else
            Print message not found
End function

Function main()
      Initialize data structure

      Print Menu:
            1. Load data structure
            2. Print course list
            3. Search and print course
            4. Exit
Read in user choice

      If user chooses case 1
            Read file choice into data structure &
            Return data structure
      If user chooses case 2
            Use print course list function
      If user chooses case 3
            Prompt for course name
            Use search and print selected course function
      If User chooses case 4
            Exit the program

      End loop
End function
```

Runtime Analysis

Vector

Code	Line Cost	# Times Executes	Total Cost
for all courses in vector	1	n	n
Print out the course info	1	n	n
		Total Cost	2n
		Runtime	O(n)

Code	Line Cost	# Times Executes	Total Cost
Check for matching course in vector	1	n	n
Print out the course info	1	1	1
For each prerequisite of the course	1	n	n

Code	Line Cost	# Times Executes	Total Cost
`Print the prerequisite course info`	1	n	n
		Total Cost	3n + 1
		Runtime	O(n)

Advantages of a vector data structure:

- Simple logic that is easy to code and use.
- Fast printing and memory allocation runtimes.

Disadvantages:

- Slow runtimes for searching through vector.

Hash Table

Code	Line Cost	# Times Executes	Total Cost
`for all courses in the hash table`	1	n	n
`Print out the course info`	1	n	n
		Total Cost	2n
		Runtime	O(n)

Code	Line Cost	# Times Executes	Total Cost
`Check for matching course in hash table`	1	1	1
`Print out course information`	1	1	1
`For each prerequisite of the course`	1	n	n
`Print the prerequisite course info`	1	n	n
		Total Cost	2n + 2
		Runtime	O(n)

Advantages of a hash table data structure:

- Fast access runtime.
- Fast printing runtime.

Disadvantages:

- Do not maintain specific order, may require sorting.
- Memory usage/ collision handling.

Binary Search Tree

Code	Line Cost	# Times Executes	Total Cost
`for all courses in the binary search tree`	1	n	n
`Print out the course info`	1	n	n
		Total Cost	2n
		Runtime	O(n)

Code	Line Cost	# Times Executes	Total Cost
`Check for matching course number in tree`	1	Log(n)	Log(n)
`Check to see if course matches`	1	Log(n)	Log(n)
`Print out course info`	1	1	1
`For each prerequisite of course`	1	n	n
`Print the prerequisite course info`	1	n	n
		Total Cost	2n + 2log(n) + 1
		Runtime	Log(n)

Advantages of a binary search tree data structure:

- Ordered structure.
- Fast print all runtime.

Disadvantages:

- Potentially long runtime for finding chosen course.
- Tree structure requires more memory than other data structures.

Recommendation

My recommended data structure for the data structure that I will most likely use in my code is a hash table. A hash table is the most efficient data structure for the project because of its capability to access data the quickest. All of the data structures have similar runtimes for a print all function, but a hash table will access and print a chosen course the fastest.