**Southern New Hampshire University**

**Wasan Alabed**

**CS300 Analysis and Design**

**October 5, 2023**

**Project One**

1. **Pseudocode opens the file, reads the data from the file, parses each line, and checks for formatting errors.**

**// Vector**

Create an empty vector to store new course.

Courses vector = Vector ()

FUNCTION read file (file path)

// Open the Course Information file for reading

FILE = OPEN ("course File.txt", "r")

// Check if the file was successfully opened

IF File IS OPEN

//Read the file line by line

WHILE NOT END\_OF\_FILE (file)

Increment line Number

READ\_LINE (file)

SPLIT line, “,”

Course code

Course title

Course prerequisite

// Check if there are at least two parameters on each line

IF length (parts) >= 2

//Create a new Course object

If the course has Prerequisites

PRINT prerequisites

ELSE

Print NO prerequisites.

ADD new course to courses vector.

Course vector. Push back (course)

Else PRINT ERROE “Unable to open or read the file."

Close file

//Function to search and print course information and prerequisites

FUNCTION print ()

IPUT new course

//Use FOR loop to Search through the vector list

For the course in course vector

IF new course = Course

Found course =course

Break

IF found course is not equal to null

PRINT course information

If the found course has Prerequisites

PRINT prerequisites

ELSE

Print NO prerequisites.

ELSE

PRINT course not found.

**// Hash- Table**

Create an empty hash table to store course objects

Course table = Hash Table ()

FUNCTION num Prerequisite Courses (Hash table<Course> courses)

// Open the Course Information file for reading

FILE = OPEN ("course File.txt", "r")

// Check if the file was successfully opened and not null

IF File IS not null

WHILE NOT END\_OF\_FILE (file)

READ\_LINE (file)

SPLIT line, “,”

//Create a new Course object using the course number and title

New course = Course (course number, course title)

// Check if there are at least two parameters on each line

IF length (parts) >= 2

Create a new Course.

// check if the course has prerequisites

If the course has Prerequisites

ADD prerequisites for the new Course object.

//Store the new Course object in the hash table.

Course table. Insert (course number, new course)

ELSE

Print NO prerequisites.

Else

PRINT ERROE message

Close file

//Function to search and print course information and prerequisites

**Function** print Course Information (Hash table <Course> courses, String course number) {

IPUT new course

//Use FOR loop to Search through the hash table

For the course in course table

IF new course = Course

Found course =course

Break

If the course is not equal to null

Print course information.

IF found course has prerequisites

Print prerequisite.

ELSE

Print NO prerequisites.

ELSE

PRINT course not found.

**// Tree Data Structure**

**Function read file (file path)**

Create an empty BST to store course objects.

Tree Node root = null

// Open the Course Information file for reading

FILE = OPEN ("course File.txt", "r")

// Check if the file was successfully opened

IF File IS OPEN

//Read the file line by line

WHILE NOT END\_OF\_FILE (file)

Increment line Number

READ\_LINE (file)

SPLIT line, “,”

// Check if there are at least two parameters on each line

IF length (parts) >= 2

Create a new Course.

If the course has Prerequisites

PRINT prerequisites

ELSE

Print NO prerequisites.

Call Insert a new course object to BST.

Else PRINT ERROE message

Close file

**Function search course (node, course number)**

Node\* current =course

While (current is not null)

if current Course == course number:

return Course

if (course Number < current course)

current = current left

else

current = current right

**Function print Sample Schedule(node)// in order traversal**

if (node is not null)

print Sample Schedule (node. left)

print course number, title, and prerequisites.

Print Sample Schedule (node. right)

**Function print Course Information (node, course Number)**

IF node is not null

IF (node course number = course number)

Print course number, title, prerequisites

IF (course number < node course number)

Print course information for left child

Else

Print course information for the right child

1. **Pseudocode for Sort the course information by alphanumeric and print course list.**

**Function print course list**

Sort Data Structure

**Function quicksort (course code, begin, end)**

IF begin >= end

Return

Mid = partition (bids, begin, end)

Quicksort (bids, begin, mid)

Quicksort (bids, mid + 1, end)

**Function Print sort course list**

For each course in course list

Print "Course Code:", “Course Title”, “Prerequisites:", Endl.

**Pseudocode for Menu**

Declare choice = 0

Loop to display menu.

Print “Welcome to the course planner.”

Print “1. Load Data Structure”

Print “2. Print Course List”

Print “3. Print Course”

Print “4. Exit”

Print “Enter your choice.”

Input Choice

Switch (choice)

Case 1:

Call **read file (file path)**

Break

Case 2:

Print “Here is a sample schedule:”

Call Print Sorted Course List

Break

Case3:

Print “What course do you want to know about?”

Input course code

Call Print course information function

Break

Case 4:

Print “Thank you for using the course planner!”

Break

Default:

Print “Invalid choice. Please try again."

Print “Goodbye”

Return 0

**Evaluation**

**Evaluate the run-time and memory of data structures that could be used to address the requirements.**

| **vector** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Reading the file(line by line)** | 1 | n | n |
| **Loop to splits each line into parts to check the length** | 1 | 1 | 1 |
| **Creating course objects and adding them to the vector** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |

| **Hash table** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Reading the file(line by line)** | 1 | n | n |
| **Loop to splits each line into parts to check the length** | 1 | 1 | 1 |
| **Creating course objects and adding them to the hash table** | 1 | 1 | 1 |
| Store the new Course object in the hash table | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 3n + 3 |
| **Runtime** | | | O(n) |

| **Tree Data Structure** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Reading the file(line by line)** | 1 | n | n |
| **Loop to splits each line into parts to check the length** | 1 | 1 | 1 |
| **Creating course objects and adding them to the vector** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 3n + 2 |
| **Runtime** | | | O(n) |

The worst-case running time for (vector, hash table, and tree) of each, reading the file and creating course objects, will be O (N).

**Advantages and disadvantages of (vector, hash table, and tree.).**

Vectors are a great option for managing data that needs to be resized often, but they are not very efficient for inserting or deleting elements, and they do not have built-in search functionality. Hash tables, on the other hand, are perfect for quickly retrieving course information based on course codes, despite the possibility of collisions and the lack of a guaranteed order for the data. Trees are ideal for managing ordered data and developing ordered programs, but maintaining balanced trees can be challenging. The best option for printing course information in alphabetical order based on course numbers is a tree data structure.