# CS 300 Pseudocode Document

Module 6 -2 Project One

Scott Dixon

Pseudocode for Course Menu and Prerequisites

Define a Course struct with fields for course code, name, and prerequisites.

Define a function called readCSVFile that takes in an unordered map of courses and a file name as input.

a. Open the file with the given file name for reading.

b. If the file was successfully opened:

i. Create a string variable called line to store each line of the file.

ii. While there are more lines in the file:

1. Read the next line into the line variable.

2. Create a stringstream object from the line variable.

3. Create string variables called courseCode, courseName, and prerequisite to store the parsed values.

4. Use getline to extract the courseCode and courseName from the stringstream using ',' as the delimiter.

5. If the courseCode does not already exist in the courses map:

a. Create a new Course object with the courseCode, courseName, and an empty vector of prerequisites.

b. Add the new Course object to the courses map with the courseCode as the key.

6. Use another while loop with getline to extract each prerequisite from the stringstream and add it to the prerequisites vector of the corresponding Course object in the courses map.

iii. Close the file.

c. If the file could not be opened, display an error message and exit the program.

Define a function called printMenu that displays a menu of options to the user.

a. Print a welcome message and a list of menu options, including "Load data structure", "Print course list", "Find prerequisites for a course", and "Exit".

Define a function called listAllCourseCodes that takes in an unordered map of courses as input.

a. Create a vector of Course objects.

b. Iterate over each element in the courses map:

i. Add the Course object to the vector.

c. Sort the vector by course code.

d. Iterate over each element in the vector:

i. Print the course code and name.

ii. If the Course object has any prerequisites, print them out as well.

Define a function called findPrerequisites that takes in an unordered map of courses as input.

a. Prompt the user to enter a course code.

b. If the course code exists in the courses map:

i. Print the course name and prerequisites (if any).

c. If the course code does not exist in the courses map, display an error message and prompt the user to try again.

Define the main function.

a. Declare an unordered map of courses.

b. Create a boolean variable called running and set it to true.

c. While running is true:

i. Display the main menu using the printMenu function.

ii. Prompt the user to enter a menu option.

iii. If the user enters "1":

1. Prompt the user to enter a file name.

2. Call the readCSVFile function with the courses map and the file name.

3. Display a message indicating that the data has been loaded.

iv. If the user enters "2":

1. Call the listAllCourseCodes function with the courses map.

v. If the user enters "3":

1. Call the findPrerequisites function with the courses map.

vi. If the user enters "9":

1. Set running to false to exit the loop and end the program.

End the program.

For a Vector:

Create a vector of course objects.

Open the file containing the course information.

For each line in the file, create a course object and add it to the vector.

Use a sorting algorithm such as merge sort or quicksort to sort the vector in alphabetical order based on course number.

For each course object in the sorted vector, print its course information to the console or screen.

Close the file.

HashTable:

Create a hash table of course objects.

Open the file containing the course information.

For each line in the file, create a course object and add it to the hash table, with the course number as the key.

Use a sorting algorithm such as merge sort or quicksort to sort the hash table by course number and store the sorted course objects in a list.

For each course object in the sorted list, print its course information to the console or screen.

Close the file.

Tree:

Create a binary search tree of course objects.

Open the file containing the course information.

For each line in the file, create a course object and add it to the binary search tree, with the course number as the key.

Use an in-order traversal algorithm to traverse the binary search tree of course objects and store the course objects in a list as they are visited.

For each course object in the sorted list, print its course information to the console or screen.

Close the file.

To evaluate the worst-case running time of reading the file and creating course objects:

Reading the file:

Open the file: O(1)

Loop through each line in the file: O(n)

Parse the line: O(1)

Check for formatting errors: O(1)

The overall worst-case running time of reading the file is O(n), where n is the number of lines in the file.

Creating course objects:

Create a new course object: O(1)

Parse the course information from the line: O(1)

Set the course properties: O(1)

Add the course object to the data structure: O(1)

The overall worst-case running time of creating a course object is O(1).

Memory usage:

The memory usage of each data structure will depend on the number of courses stored in the data structure and the size of each course object. The memory usage of a vector will be proportional to the number of courses stored, while the memory usage of a hash table or tree will depend on the number of courses and the distribution of keys.

Overall, the vector data structure will have O(n) worst-case memory usage, as it will store a contiguous block of memory for each course object. The hash table and tree data structures will have O(nlogn) and O(n) worst-case memory usage, respectively, due to the overhead of storing and managing the additional data structure.

Time and Space Complexity of Vector, Hash Table and Binary Tree for a Sort Function

Function Time Complexity Space Complexity

|  |  |  |
| --- | --- | --- |
| sort(v.begin( ), v.end( ))  Vector | nlog(n) | log n |
| sort(v.begin( ), v.end( ))  Hash Table | O(n) | O(n) |
| sort(v.begin( ), v.end( ))  Binary Tree | O(log N) | O(n) |

# Advantages and disadvantages of Vector, Hash Table and Binary Tree.

* "BST allows for recursion, which can be used to solve problems more elegantly and efficiently." (“Advantages of BST over Hash Table - GeeksforGeeks”) Hash tables do not allow for recursion.
* A hash table is a dictionary; it stores and retrieves objects by a key value. A vector, on the other hand, holds an ordered collection of elements.
* BST are memory efficient but Hash table is not.
* With Self-Balancing BSTs, all operations are guaranteed to work in O(Logn) time. But with Hashing, Θ(1) is average time and some operations may be costly i.e, O(n2 ), especially when table resizing happens. (“Advantages of BST over Hash Table - GeeksforGeeks”)

I think the advantages of a faster sorting algorithm make the Binary Search Tree or the Hash Table the choice for larger dictionary or database sorting. The ability to recursively sort and to be able to delete and append records in the file make the choices more apparent as I further study the different algorithms.

Sources:

Advantages of BST over Hash Table - GeeksforGeeks, https://www.geeksforgeeks.org/advantages-of-bst-over-hash-table/.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **for all courses** | 1 | n | n |
| **if the course is the same as courseNumber** | 1 | n | n |
| **print out the course information** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** | 1 | n | n |
| **Total Cost** | | | 4n + 1 |
| **Runtime** | | | O(n) |