Sydney Porter

CS 300

12th April 2024

Project One

Menu Pseudocode

Function load\_data\_structure(file):

Open file

Read line

While not end of file

Parse line for course information

Create course object

Store course object in data structure

Read next line

Close file

Function print\_course\_list(data\_structure):

Sort data\_structure alphabetically by course title

For each course in data\_structure

Print course.title

Print course.prerequisites

Function print\_course(data\_structure, course\_title):

For each course in data\_structure

If course.title equals course\_title

Print course.title

Print course.prerequisites

Function display\_menu():

Display menu options:

1. Load Data Structure

2. Print Course List

3. Print Course

4. Exit

Prompt user for choice

Return user choice

Course List Pseudocode

Function print\_alphanumeric\_course\_list(data\_structure):

Sort data\_structure alphabetically by course title

For each course in data\_structure

Print course.title

Certainly! Here's the combined paragraph:

In assessing the runtime complexities of the three proposed data structures—vector, hash table, and tree—several factors were considered. For the vector structure, which maintains a linear list of courses, the time complexity for reading the file and creating course objects is linear (O(n)), with printing the course list requiring additional time due to sorting (O(nlogn)). However, accessing individual courses also requires a linear search (O(n)), making it less efficient for large datasets. Conversely, the hash table, while also requiring linear time to read the file and create objects (O(n)), provides constant-time access (O(1) on average) to individual courses, crucial for efficient printing of course information. Nevertheless, hash tables may suffer from memory overhead and potential collisions, impacting performance, particularly in worst-case scenarios (O(n^2) for printing the course list). Similarly, the tree structure offers efficient searching and sorting, with a runtime complexity of O(nlogn) for reading and creating objects and printing the course list, assuming a balanced tree. Accessing individual courses has a logarithmic time complexity (O(logn)), faster than the vector but slower than the hash table. However, implementing and maintaining a balanced tree can be complex and resource-intensive, potentially slowing down insertion and deletion operations compared to hash tables. Therefore, while each data structure has its strengths, the hash table emerges as the most suitable choice for this project, balancing performance and ease of implementation.

The requirements and performance study suggest that the hash table is the best data structure. For printing course materials and requirements, it offers quick access to particular courses (O(1) typical case). Although there might be some memory overhead and a chance of accidents, these are outweighed by its efficiency benefits.

Click here to enter text.