Project One

Zaffar Shiekh

CS-300-R3288 DSA: Analysis and Design 24EW3

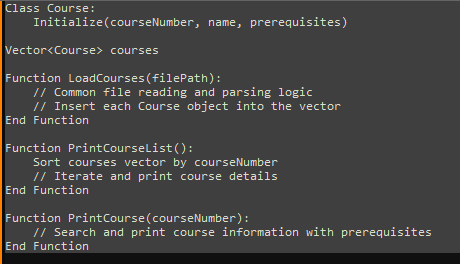
Prof. David Ostrowski

02/18/2024

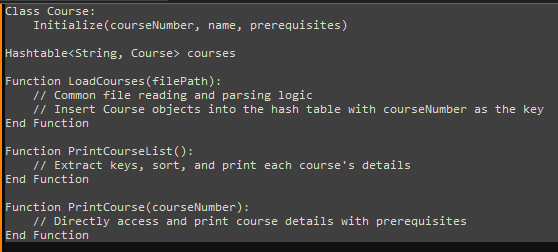
To fulfill the requirements of this project for the Computer Science department at ABCU, I have designed a system that prints a list of all Computer Science courses in alphanumeric order and also retrieves specific course information, including prerequisites. This system uses three distinct data structures: a vector, a hash table, and a tree. Below is the comprehensive pseudocode for each data structure, including a menu system to interact with the program, followed by a Big O runtime analysis and evaluation of each data structure.

**Pseudocode for Data Structure Operations**

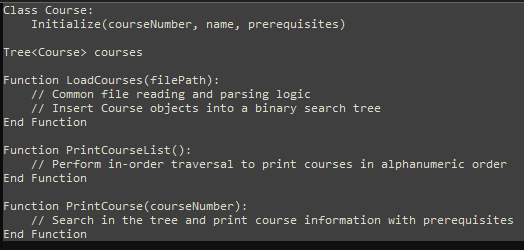
**Vector Data Structure:**



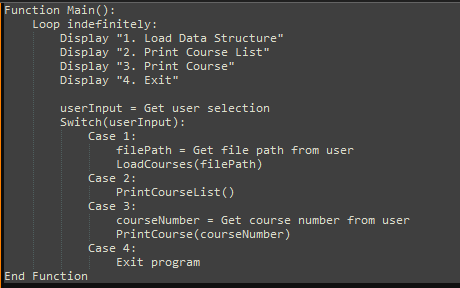
**Hash Table Data Structure:**



**Tree Data Structure:**



**Pseudocode for Menu System:**



**Runtime Analysis and Evaluation:**

1. Vector
   1. Loading Data: *O*(*n*) for appending n courses.
   2. Printing Course List: *O*(*n* log *n*) due to sorting.
   3. Printing Course Information: *O*(*n*) for linear search.

**Advantages**: Simple and contiguous memory usage.

**Disadvantages**: Slow search and sorting for large datasets.

1. Hash Table
   1. Loading Data: *O*(*n*) average for inserting n courses.
   2. Printing Course List: *O*(*n* log *n*) due to sorting keys.
   3. Printing Course Information: *O*(1) average for direct access.

**Advantages**: Fast average search and access times.

**Disadvantages**: Unordered, potential collisions, and slower ordered iteration.

1. Tree
   1. Loading Data: *O*(*n* log *n*) inserts n courses in a balanced tree.
   2. Printing Course List: *O*(*n*) for in-order traversal.
   3. Printing Course Information: *O*(log *n*) for balanced tree search.

**Advantages**: Maintains order, faster search than vector, and log-scale access times.

**Disadvantages**: Potentially complex implementation and balance maintenance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation** | **Vector** | **Hash** | **Tree** | **Notes** |
| **Load Data** | *O*(*n*) | *O*(*n*) | *O*(log *n*) | n is the number of courses |
| **Print List (Unsorted)** | *O*(*n*) | *O*(*n*) | *O*(*n*) | n is the number of courses |
| **Print List (Sorted)** | *O*(*n* log *n*) | *O*(*n* log *n*) | *O*(*n*) | Sorting required for vector and hash table |
| **Print Course** | *O*(*n*) | *O*(1) | *O*(log *n*) | Assuming no hash collisions for the hash table |
| **Search for a Course** | *O*(*n*) | *O*(1) | *O*(log *n*) | Assuming a balanced tree for tree structure |
| **Insert a Course** | *O*(1) | *O*(1) | *O*(log *n*) | Assuming a balanced tree and good hash function |

**Recommendation:**

Considering the requirements of the system and the Big O analysis, I recommend using a **balanced binary search tree**. This data structure strikes the best balance between maintaining order for listing courses and providing efficient access to individual course information. Its log-scale search is particularly beneficial for quickly finding course details when dealing with many courses, which is common in a university setting. The tree's in-order traversal naturally supports the printing of courses in alphanumeric order, meeting one of the primary requirements without additional sorting algorithms.

In contrast, while vectors are simple to implement, their linear search time and the need for sorting make them less efficient for our use case. Hash tables offer fast average search times but do not maintain the order of courses, requiring additional sorting steps, which can be inefficient for large datasets.

Therefore, the balanced tree's ability to maintain order and provide efficient search capabilities leads me to choose it as the most suitable data structure for ABCU's Computer Science Department advising program.

The balanced binary search tree provides an elegant solution that meets the performance and functionality requirements of the advising system. Its ordered nature and efficient search and access operations make it ideal for managing the course data efficiently.

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). The MIT Press. This book is a comprehensive resource on algorithms, which has guided the development of efficient pseudocode and the understanding of the Big O notation used in runtime analysis.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley. This text has provided insights into data structures and algorithm analysis, informing the evaluation of vectors, hash tables, and trees in the context of the project.