Karl Maloy

CS-300

Project 1

1. Display Menu Options:

- Option 1: Load file data into the data structure

- Option 2: Print an alphanumerically ordered list of all courses

- Option 3: Print course title and prerequisites

- Option 9: Exit the program

2. Initialize variables:

- loaded\_data = false

3. Loop until user selects Option 9:

Display Menu Options

Input choice from user

if choice == 1:

if loaded\_data:

Print "Data already loaded."

else:

// Implement loading data into data structure (vector, hash table, or tree)

LoadData()

loaded\_data = true

else if choice == 2:

if not loaded\_data:

Print "Please load data first."

else:

// Print sorted list of courses

PrintSortedCourses()

else if choice == 3:

if not loaded\_data:

Print "Please load data first."

else:

Input course\_number

// Print course title and prerequisites for specified course

PrintCourseInfo(course\_number)

else if choice == 9:

Exit the program

else:

Print "Invalid choice. Please select again."

4. End loop

// Function to load data into data structure

function LoadData():

// Implement according to the data structure (vector, hash table, or tree)

// Function to print alphanumerically ordered list of courses

function PrintSortedCourses():

// Implement according to the data structure used

// Function to print course title and prerequisites

function PrintCourseInfo(course\_number):

// Implement according to the data structure used

Vector-

-Advantages

- Simple and straightforward implementation.

- Allows indexing which facilitates sorting.

- Good for small to medium-sized datasets where random access and sorting are needed.

- Disadvantages:

- Insertion and deletion can be expensive if done frequently, especially in large datasets.

- Sorting can be costly for large datasets.

-Big O Analysis for Loading Data:

- Reading file: O(n)

- Creating course objects: O(n)

- Total: O(n)

Hash Table

- Advantages:

- Provides average O(1) time complexity for insertions, deletions, and lookups.

- Efficient for storing and retrieving data quickly based on keys (course numbers).

- Disadvantages:

- Does not maintain order inherently, so sorting requires additional steps.

- Hash collisions may impact performance, though good hashing functions minimize this.

- Big O Analysis for Loading Data:

- Reading file: O(n)

- Creating course objects: O(n)

- Total: O(n)

3. Binary Search Tree (BST)

- Advantages:

- Maintains order of elements based on keys (course numbers), so sorting is straightforward (inorder traversal).

- Efficient for searching (average O(log n)).

- Disadvantages:

- Can become unbalanced, leading to worst-case O(n) for operations.

- Insertion and deletion operations may require restructuring to maintain balance.

- Big O Analysis for Loading Data:

- Reading file: O(n)

- Creating course objects: O(n log n) on average

- Total: O(n log n)

Recommendation

Based on the analysis:

- Vector is suitable if the primary operation after loading is sorting and if the dataset is not expected to change frequently.

- Hash Table is suitable if quick lookups by course number are important and the dataset is relatively static.

- Binary Search Tree is suitable if maintaining sorted order and efficient search operations are crucial, even with potential restructuring overhead.

Recommendation: Given the requirement to print an alphanumerically ordered list of courses frequently, the Binary Search Tree (BST) is recommended. Despite its potential for O(n log n) time complexity for insertion and creation, its ability to maintain sorted order naturally through inorder traversal makes it well-suited for this specific task. The benefit of balancing operations ensures that subsequent operations like printing sorted courses remain efficient. Therefore, a balanced BST would provide a good balance between efficiency in sorting and reasonable performance for insertion and lookup operations.