Todd Johnson

08/13/2024

CS-300-11613-M01 DSA: Analysis and Design

CS 300 Project One

***Pseudocode***

***Vector Data Structure:***

Open file with course data

Initialize an empty vector `courseList`

While there are lines in the file:

Read line

Parse the line into course information

Check for formatting errors

If no errors:

Create a course object

Append course object to `courseList`

Else:

Output an error message

Close the file

Function CreateCourse(line):

Split the line by delimiter (e.g., comma)

Extract course number, course title, and prerequisites

Create a course object with extracted data

Return course object

Function PrintCourseInfo(course):

Print "Course Number: ", course.number

Print "Course Title: ", course.title

Print "Prerequisites: ", course.prerequisites

***Hash Table Data Structure:***

Open file with course data

Initialize an empty hash table `courseTable`

While there are lines in the file:

Read line

Parse the line into course information

Check for formatting errors

If no errors:

Create a course object

Add course object to `courseTable` using course number as key

Else:

Output an error message

Close the file

Function CreateCourse(line):

Split the line by delimiter (e.g., comma)

Extract course number, course title, and prerequisites

Create a course object with extracted data

Return course object

Function PrintCourseInfo(course):

Print "Course Number: ", course.number

Print "Course Title: ", course.title

Print "Prerequisites: ", course.prerequisites

***Tree Data Structure:***

Open file with course data

Initialize an empty binary search tree `courseTree`

While there are lines in the file:

Read line

Parse the line into course information

Check for formatting errors

If no errors:

Create a course object

Insert course object into `courseTree`

Else:

Output an error message

Close the file

Function CreateCourse(line):

Split the line by delimiter (e.g., comma)

Extract course number, course title, and prerequisites

Create a course object with extracted data

Return course object

Function PrintCourseInfo(course):

Print "Course Number: ", course.number

Print "Course Title: ", course.title

Print "Prerequisites: ", course.prerequisites

A white sheet with black text

Description automatically generated

Analysis of Advantages and Disadvantages

1)Vector:

Advantages:

- Simple to implement and manage.

- Efficient for sequential access.

Disadvantages:

- Linear search time makes it less efficient for large datasets.

- Sorting is required to maintain order, adding additional overhead.

2. Hash Table:

Advantages:

- Fast average-case insertion and lookup (O(1)).

- Efficient for large datasets where search operations are frequent.

Disadvantages:

- Worst-case performance can degrade to O(n) if many collisions occur.

- Higher memory overhead due to storage of keys and management of collisions.

3. (Binary Search Tree):

Advantages:

- Efficient search, insertion, and deletion operations with O(log n) time complexity (assuming the tree is balanced).

- Automatically maintains sorted order, eliminating the need for additional sorting.

Disadvantages:

- More complex to implement and manage.

- Unbalanced trees can degrade to O(n) performance for search and insertion.

***Conclusion***

After carefully analyzing the Vector, Hash Table, and Tree data structures, I recommend using a balanced Binary Search Tree (BST) for the implementation. The Tree structure offers a worst-case time complexity of O(log n) for search, insertion, and deletion operations, making it highly efficient even with larger datasets. This efficiency is crucial for maintaining fast and responsive search operations. Additionally, the Tree's inherent ability to maintain sorted order eliminates the need for additional sorting steps, which is particularly advantageous for applications requiring the data to be displayed in alphanumeric order.

In comparison to a Hash Table, which provides O(1) average-case time complexity but can degrade to O(n) in the worst case due to collisions, the Tree offers more consistent and reliable performance. Furthermore, the Tree data structure uses O(n) memory, similar to the Vector, but without the added overhead associated with managing hash table collisions. The Tree's balanced performance and scalability make it a robust choice for long-term use, ensuring that the program remains efficient and responsive as the dataset grows. Therefore, the Tree data structure is the most suitable option for this application.

.