John Munguia

CS 300

December 13, 2024

Pseudocode

Main Function() // Menu Loop

Read command-line arguments

Store argument as CSV file path

If no command-line arguments, load default CSV file path

Loop while choice is not equal to '9'

Output menu

Get user input; store in menuChoice

Get user input; store in dataStructureChoice

Validate user input

If menuChoice is not 1–4 or 9, throw an error

If menuChoice == '1' // Load data

If Binary Search Tree

Call loadBids() and store CSV data in BinarySearchTree bst

Else if Vector

Call loadBids() and store CSV data in vector courseList

Else if Hash Table

Call loadBids() and store CSV data in HashTable courseTable

If menuChoice == '2' // Print alphanumeric course list

If Binary Search Tree

Call printTree()

Else if Vector

Call sortList()

Call printList()

Else if Hash Table

Call printTable()

If menuChoice == '3' // Search for course

Get user input for searchKey

If Binary Search Tree

Call searchTree(searchKey)

Else if Vector

Call searchList(searchKey)

Else if Hash Table

Call searchTable(searchKey)

If menuChoice == '9'

Exit program

Output "Goodbye."

End

loadBids(filePath, BinarySearchTree tree)

Open file at filePath

Loop through each line in the file

Parse line into courseID, title, and prerequisites

Create Course object

Insert Course into tree

End

searchTree(searchKey)

Start at tree root

Traverse tree to find matching courseID

If match found

Print courseID, title, and prerequisites

Else

Output "Course not found."

End

printTree()

Perform in-order traversal of the tree

For each node:

Print courseID, title, and prerequisites

End

loadBids(filePath, vector<Course> courseList)

Open file at filePath

Loop through each line in the file

Parse line into courseID, title, and prerequisites

Create Course object

Append Course to courseList

End

searchList(searchKey)

Loop through courseList

If courseID matches searchKey

Print courseID, title, and prerequisites

Return

Output "Course not found."

End

sortList(courseList)

Use quicksort to sort courseList by courseID

End

printList(courseList)

Loop through courseList

Print courseID, title, and prerequisites

End

loadBids(filePath, HashTable table)

Open file at filePath

Loop through each line in the file

Parse line into courseID, title, and prerequisites

Create Course object

Insert Course into table using hash(courseID)

End

searchTable(searchKey)

Use hash(searchKey) to find bucket

Search linked list in bucket for matching courseID

If match found

Print courseID, title, and prerequisites

Else

Output "Course not found."

End

printTable(table)

Loop through each bucket in table

For each course in bucket:

Print courseID, title, and prerequisites

End

Runtime Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Vector | Hash Table | Binary Search Tree |
| Loading Data | O(n) | O(1) – O(n) | O(log n) – O(n) |
| Search | O(n) | O(1) – O(n) | O(log n) – O(n) |
| Sort/Print | |  | | --- | | O(n log n) |  |  | | --- | |  | | |  | | --- | | O(n) (pre-sorted insertion) |  |  | | --- | |  | | O(n) (in-order traversal) |

Vector

**Advantages**: Simple to implement, fast insertion.

**Disadvantages**: Slow searches and sorting due to linear traversal.

Hash Table

**Advantages**: Fast average-case search and insertion (O(1)).

**Disadvantages**: Potential for collisions leading to O(n) operations. Memory-intensive.

Binary Search Tree

**Advantages**: Always sorted, good search performance (O(log n)) if balanced.

**Disadvantages**: Slower than hash table for search. Worst-case O(n) for unbalanced trees.

Recommendation

Based on the runtime analysis and program requirements, the **Hash Table** is the optimal choice for ABCU’s course advising program. Its fast average-case performance for both insertion and search operations makes it ideal for frequent lookups. Proper hashing and collision handling mechanisms can ensure consistent performance.