Veronica Guzman

November 29, 2023

CS 300

Psuedocode

# Vector Operations

Function LoadDataIntoVector(filename):

# Pseudocode to load data from file into a vector

Function SortCoursesAlphanumeric(vector):

# Pseudocode to sort courses in the vector alphabetically

Function PrintCourseList(vector):

# Pseudocode to print the list of courses in the vector

Function PrintCourseDetails(vector, courseNumber):

# Pseudocode to print the details of a specific course in the vector

# Hash Table Operations

Function LoadDataIntoHashTable(filename, hashtable):

# Pseudocode to load data from file into a hash table

Function PrintCourseDetailsFromHashTable(hashtable, courseNumber):

# Pseudocode to print the details of a specific course from the hash table

# Tree Operations

Function LoadDataIntoTree(filename, tree):

# Pseudocode to load data from file into a tree structure

Function PrintCourseListFromTree(tree):

# Pseudocode to print the list of courses in the tree

Function PrintCourseDetailsFromTree(tree, courseNumber):

# Pseudocode to print the details of a specific course from the tree

# Menu Functionality

Function DisplayMenu():

# Pseudocode to display the menu options

Function PerformMenuAction(selection, dataStructure):

# Pseudocode to perform actions based on menu selection

# File Operations and Course Handling

Function OpenFile(file\_path):

# Pseudocode to open a file

Function ParseCourseData(line):

# Pseudocode to parse data from each line in the file

Function CreateCourseObject(courseNumber, courseTitle, prerequisites):

# Pseudocode to create a course object

|  |  |  |
| --- | --- | --- |
| Operation | Time Completxity | Description |
| ‘LoadDataIntoVector(filename)’ | O(n) | Reading each line from the file and inserting into the vector where n is the number of courses/lines in the file. |
| ‘SortCoursesAlphanumeric(vector)’ | O(n log n) | Sorting courses in the vector using an efficient sorting algorithm (like merge sort or quicksort). |
| ‘PrintCourseList(vector)’ | O(n) | Printing the list of courses stored in the vector by traversing through each course. |
| ‘PrintCourseDetails(vector, courseNumber)’ | O(n) | Searching for and printing details of a specific course from the vector where n is the number of courses. |
| ‘LoadDataIntoHashTable(filename, hashtable)’ | O(n) | Reading each line from the file and inserting into the hash table where n is the number of courses/lines in the file. |
| ‘PrintCourseDetailsFromHashTable(hashtable, courseNumber)’ | O(1) to O(n) | Printing details of a specific course from the hash table. Average case is O(1) (constant time) due to hash table lookup, but worst-case (hash collisions) could be O(n). |
| ‘LoadDataIntoTree(filename, tree)’ | O(n log n) | Reading each line from the file and inserting into a tree structure (like AVL or Red-Black tree) with log n insertion time for each course in the average case. |
| ‘PrintCourseListFromTree(tree)’ | O(n) | Printing the list of courses stored in the tree by traversing the entire tree. |
| ‘PrintCourseDetailsFromTree(tree, courseNumber)’ | O(log n) | Searching for and printing details of a specific course from the tree in a balanced tree structure (like AVL or Red-Black tree). |
| ‘DisplayMenu()’ | O(1) | Displaying the menu options typically takes constant time. |
| ‘PerformMenuAction(selection, dataStructure)’ | Varies | The time complexity will depend on the action performed and the associated data structure. For instance, performing actions like printing courses could be O(n) based on the data structure used. |
| ‘OpenFile(file\_path)’ | Varies | The time complexity might vary based on the file size and operating system considerations but typically could be O(n) where n is the file size. |
| ‘ParseCourseData(line)’ | O(1) | Parsing a line to extract course details usually involves constant time unless the parsing operation is complex. |
| ‘CreateCourseObject(courseNumber, courseTitle, prerequisites)’ | O(1) | Creating a course object usually involves constant time unless additional data processing is performed. |

The runtime analysis chart outlines the estimated time complexities associated with various operations presented in the pseudocode provided for managing course data. It attempts to predict how the execution time of these operations scales concerning the input size or specific actions performed. The chart provides insights into the efficiency of different functionalities implemented within the context of data structures like vectors, hash tables, and trees. It indicates that certain operations exhibit different time complexities based on the data structure employed and the nature of the task. For instance, when considering loading data from a file, most operations seem to have a linear time complexity (O(n)), which means the time taken tends to increase linearly with the number of courses or lines in the file. However, sorting courses alphabetically shows a more efficient performance of O(n log n), indicating a slightly faster execution time for larger datasets. The comparison between different data structures reveals interesting nuances. Hash tables, for example, display a typical average-case constant lookup time (O(1)) for printing specific course details. Still, this efficiency might degrade in worst-case scenarios due to hash collisions, potentially leading to linear time complexity (O(n)). The analysis also highlights the effectiveness of tree structures for certain tasks. Searching for specific course details in a balanced tree structure exhibits logarithmic time complexity (O(log n)), indicating faster search times as compared to linear searches in vectors or potential performance drawbacks in hash tables. While the chart provides a valuable understanding of how different operations perform in various scenarios, it's important to note that the actual runtime may vary based on implementation details, data characteristics, and specific algorithms used. In conclusion, the runtime analysis chart serves as a useful guide for selecting suitable data structures and understanding the expected efficiency of different operations. However, practical implementation and real-world testing will provide a more accurate assessment of performance.