Kevin Alexander

CS-3000-11292

DSA:Analysis and Design

Instructor: Oluwayomi Adamo

6-2 Submit Project One

**Pseudocode for the Menu:**

BEGIN

WHILE true

PRINT “Menu:”

PRINT Load file data into the data structure”

PRINT Print an alphanumerically ordered list of all the courses in the Computer Science department course”

PRINT Print the course title and the prerequisites for any individual course”

PRINT Exit the program”

INPUT choice

IF choice == 1 THEN

CALL LoadData()

ELSE IF choice == 2 THEN

CALL PrintSortedCourses()

Else IF choice == 3 THEN

CALL PrintCourseDetails()

Else IF choice == 9 THEN

EXIT

ELSE

PRINT “Invalid choice. Please try again.”

END IF

END WHILE

END

**Pseudocode for Loading Data**

PROCEDURE LoadData

OPEN file

WHILE NOT end of file

READ line

PARSE line into course information

ADD course information to data structure

END WHILE

CLOSE file

END PROCEDURE

**Pseudocode for Printing Sorted Courses**

**Using Vector**

PROCEDURE PrintSortedCourses()

SORT vector by course number

FOR EACH course IN vector

PRINT course

END FOR

END PROCEDURE

**Using Hash Table**

PROCEDURE PrintSortedCourses()

CREATE empty list

For EACH key IN hash table

ADD hash table[key] to list

END For

SORT list by course number

FOR EACH course IN list

PRINT course

END FOR

END PROCEDURE

**PART 3 Pseudocode:**

Pseudocode for Vector:

1.Initialize vector courses.

2.Add course information to courses

3.Sort course in alphanumeric order.

4.For each course in courses:

Print course

Pseudocode for Hash Table:

1.Initialize hash table courses

2.Add course information to courses

3.Extract keys from courses and store

4.Sort keys in alphanumeric order

5. For each key in keys:

Print course[key]

Pseudocode for Tree:

1.Initialize binary search tree courses

2.Insert course information into courses

3. Define function inOrderTraversal(node);

1. If node is not null;

i.inOrderTraversal(node.left)

ii.Print node.course

iii.inOrderTraversal(node.right)

4.Call inOrderTraversal(courses.root)

**Evaluation**

4. A.

i. Cost per line of code and number of times the line will execute:

-Opening the file:0(1)

-Reading the file:0(n)

-Parsing each line:0(n)

-Checking for formatting errors:0(n)

-Creating course objects:0(n)

ii.Worst-case running time(Big O value):

-Opening the file:0(1)

-Reading the file0(n)

-Parsing each line:0(n)

-Checking for formatting errors:0(n)

-Creating course objects:0(n)

Analyze each line:

-Open file: This operation is typically 0(1) because it is a single action.

-while not end of file:This loop will run n times,where n is the number of lines(or courses) in the file.

-read line:Reading a line is0(1) per line,so for n lines,it is 0(n).

-parse Line:Parsing a line is0(1) per line,so for n lines,so for n lines,it is 0(n).

-add course object to data structure

-close file:This operation is typically 0(1) because it is a single action.

Overall worst-case running time:

-The overall worst-case running time is the sum of the individual steps. Since the dominant trem is 0(n), the overall worst-case running time is 0(n).

Step-by-Step

1.Open the File

-Cost per line:1

-Number of executions:1

-Total cost:1

2.Read Each Line from the File

While not end of file:

read line

-Cost per line:1

-Number of executions:n(assuming there are n lines/courses)

-Total cost:n

3.Parse Each line

For each line:

Parse line

-Cost per line:1

-Number of executions:n

-Total cost:n

4.Check for Formatting Errors

For each parsed line:

check formatting

cost per line:1

Number of executions:n

Total cost:n

5.Create Course Object

for each valid line;

create course object

-Cost per line:1

-Number of executions:n

-Total cost:n

Total Cost Analysis

To find the total cost,we sum up the costs of each step:

1.Open the file:1

2.Read each line:n

3.Parse each line:n

4.Check for formatting errors:n

5.Create course object:n

Total Cost = 1 + n + n + n + n = 1 + 4n

Vector:

Advantages:

1.Dynamic Size: Vectors can dynamically resize themselves when elements are added or removed.

2.Random Access:Provide constant time complexity 0(1) for accessing elements by index.

3,Cache Friendly:Elements are stored in contiguous memory locations,which makes them cache-friendly and can lead to performance improvements.

Disadvantages:

1.Resizing Overhead:When the vector resizes,it may need to allocate new memory and copy all elements,which can be costly.

2.Insertion/Deletion:Inserting or deleting elements in the middle of the vector has a time complexity of 0(n),as it may require shifting elements.

HashTable

Advantages:

1.Average Case Performance:Provides average time complexity of 0(1) for insertions,deletions, and lookups.

2.Flexible Keys:Can use various types of keys,not just integers.

Disadvantages:

1.Worst Case Performance: In the worst case,operations can degrade to 0(n) due to collisions.

2.Memory Overhead:Requires extra memory for storing hash values and handling collisions(e,g,through chaining or open addressing).

3.Non-Ordered:Does not maintain any order of elements,which can be a disadvantage if ordered data is required.

Tree(Binary Search Tree,BST)

Advantages:

1.Ordered Data: Maintains elements in a sorted order,which is useful for range queries and ordered traversals.

2.Balanced Trees:Self-balancing trees (e.g.,AVL,Red-Black trees)provide guaranteed 0(log n) time complexity for insertions,deletions,and lookups.

Disadvantages:

1.Complexity:More complex to implement and maintain compared to vectors and hash tables.

2.Performance:Even balanced trees have higher constant factors compared to hash tables for average case operations.

Recommendation

-Need fast average-case performance for insertions,deletions,and lookups and do not require ordered data,a hash table is the best choice.

-Need ordered data and can tolerate o(log n) performance,a balanced tree(e.g., AVL or Red-Black tree)is recommended.

-Need dynamic resizing and fast random access,and your operations are mostly appended and accessed by index,a vector is suitable.

Given the Big O analysis and the typical use cases,I recommend using a hash table if the primary operations are insertions,deletions,and lookups,and ordered data is not a requirement. This recommendation is based on the average-caseo(1) performance for the best efficiency for a wide general and provides the best efficiency for a wide range of applications.