CONTINUOUS ASSESSMENT COMPONENT (CAC1)

DATA STRUCTURES IN C

2147255_NITHYA. S

3 MCA B

Applications of Data Structures

The data structures store the data according to the mathematical or logical model it is based on. The type of operations on a certain data structure makes it useful for specific tasks. Here is a brief discussion of different applications of data structures.

Arrays

- Storing list of data elements belonging to same data type
- Auxiliary storage for other data structures
- Storage of binary tree elements of fixed count
- Storage of matrices

Linked List

- Implementing stacks, queues, binary trees and graphs of predefined size.
- Implement dynamic memory management functions of operating system.
- Polynomial implementation for mathematical operations
- Circular linked list is used to implement OS or application functions that require round robin execution of tasks.
- Circular linked list is used in a slide show where a user wants to go back to the first slide after last slide is displayed.
- When a user uses the alt+tab key combination to browse through the opened application to select a desired application
- Doubly linked list is used in the implementation of forward and backward buttons in a browser to move backwards and forward in the opened pages of a website.
- Circular queue is used to maintain the playing sequence of multiple players in a game.

Stacks

- Temporary storage structure for recursive operations
- Auxiliary storage structure for nested operations, function calls, deferred/postponed functions
- Manage function calls
- Evaluation of arithmetic expressions in various programming languages
- Conversion of infix expressions into postfix expressions
- Checking syntax of expressions in a programming environment
- Matching of parenthesis
- String reversal
- In all the problems solutions based on backtracking.
- Used in depth first search in graph and tree traversal.
- Operating System functions
- UNDO and REDO functions in an editor.

Queues

- It is used in breadth search operation in graphs.
- Job scheduler operations of OS like a print buffer queue, keyboard buffer queue to store the keys pressed by users
- Job scheduling, CPU scheduling, Disk Scheduling
- Priority queues are used in file downloading operations in a browser
- Data transfer between peripheral devices and CPU.
- Interrupts generated by the user applications for CPU
- Calls handled by the customers in BPO

Efficiency of Algorithm

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

From the data structure point of view, following are some important categories of algorithms −

- **Search** − Algorithm to search an item in a data structure.
- **Sort** − Algorithm to sort items in a certain order.
- **Insert** − Algorithm to insert item in a data structure.
- **Update** − Algorithm to update an existing item in a data structure.
- **Delete** − Algorithm to delete an existing item from a data structure.

Algorithm efficiency

A measure of the average execution time necessary for an algorithm to complete work on a set of data. Algorithm efficiency is characterized by its order. Typically a bubble sort algorithm will have efficiency in sorting N items proportional to and of the order of N^2, usually written O (N^2). This is because an average of $N/2$ comparisons are required $N/2$ times, giving $N^2/4$ total comparisons, hence of the order of N^2. In contrast, quicksort has an efficiency O ($N \log_2 N$).

If two algorithms for the same problem are of the same order then they are approximately as efficient in terms of computation. Algorithm efficiency is useful for quantifying the implementation difficulties of certain problems. The efficiency of an algorithm depends on how efficiently it uses time and memory space.

The time efficiency of an algorithm is measured by different factors. For example, write a program for a defined algorithm, execute it by using any programming language, and measure the total time it takes to run. The execution time that you measure in this case would depend on a number of factors such as:

- Speed of the machine
- Compiler and other system Software tools
- Operating System
- Programming language used
- Volume of data required

Asymptotic Notations

Asymptotic Notations are languages that uses meaningful statements about time and space complexity. The following three asymptotic notations are mostly used to represent time complexity of algorithms:

(i) Big O

Big O is often used to describe the worst-case of an algorithm.

(ii) Big Ω

Big Omega is the reverse Big O, if Bi O is used to describe the upper bound (worst - case) of a asymptotic function, Big Omega is used to describe the lower bound (best-case).

(iii) Big Θ

When an algorithm has a complexity with lower bound = upper bound, say that an algorithm has a complexity O (n log n) and (n log n), it's actually has the complexity Θ (n log n), which means the running time of that algorithm always falls in n log n in the best-case and worst-case.

Best, Worst, and Average ease Efficiency

Let us assume a list of n number of values stored in an array. Suppose if we want to search a particular element in this list, the algorithm that search the key element in the list among n elements, by comparing the key element with each element in the list sequentially.

The best case would be if the first element in the list matches with the key element to be searched in a list of elements. The efficiency in that case would be expressed as O (1) because only one comparison is enough.

Similarly, the worst case in this scenario would be if the complete list is searched and the element is found only at the end of the list or is not found in the list. The efficiency of an algorithm in that case would be expressed as O(n) because n comparisons required to complete the search.

The average case efficiency of an algorithm can be obtained by finding the average number of comparisons as given below:

Minimum number of comparisons = 1 Maximum number of comparisons = n

If the element not found then maximum

Number of comparison = n

Therefore,   average number   of comparisons = (n + 1)/2

Hence the average case efficiency will be expressed as O (n).

Conclusion

Good choice of algorithm and data structure can make big differences to the efficiency of a program, and we looked at how it is possible for one problem to have several alternative algorithms that can solve it, or one abstract data type to have several alternative data structures that can represent it.

When writing code to solve some problem or represent some structure, it is best to write it in a way so that code can be re-used in different circumstances. We saw how inheritance helps us write more general code, and generic typing is another Java feature which can be used to write code useable with a variety of different actual types. We also saw how to make use of code from the Java library: the object-oriented programming style and the various generalisation mechanisms encourage re-use of code which already exists rather than writing your own from the beginning.