

INTRODUCTION TO DATA STRUCTURES

Basic Concepts: Introduction to Data Structures:

A data structure is an arrangement of data in computer's memory in such a way that it could make the data quickly available to the processor for required calculations.

A data structure should be seen as a logical concept that must address two fundamental concerns.

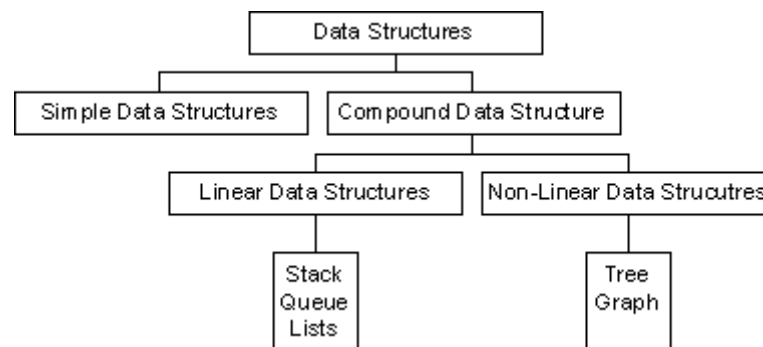
1. First, how the data will be stored, and
2. Second, what operations will be performed on it.

Some of the more commonly used data structures include lists, arrays, stacks, queues, heaps, trees, and graphs.

Classification of Data Structures:

Data structures can be classified as

- Simple data structure
- Compound data structure
- Linear data structure
- Non linear data structure



[Classification of Data Structures]

Simple Data Structure:

Simple data structure can be constructed with the help of primitive data structure. A primitive data structure used to represent the standard data types of any one of the computer languages. Variables, arrays, pointers, structures, unions, etc. are examples of primitive data structures.

Compound Data structure:

Compound data structure can be constructed with the help of any one of the primitive data structure and it is having a specific functionality. It can be designed by user. It can be classified as

- Linear data structure
- Non-linear data structure

Linear Data Structure:

Linear data structures can be constructed as a continuous arrangement of data elements in the memory. It can be constructed by using array data type. In the linear Data Structures the relationship of adjacency is maintained between the data elements.

Operations applied on linear data structure:

The following list of operations applied on linear data structures

1. Add an element
2. Delete an element
3. Traverse
4. Sort the list of elements
5. Search for a data element

For example Stack, Queue, Tables, List, and Linked Lists.

Non-linear Data Structure:

Non-linear data structure can be constructed as a collection of randomly distributed set of data item joined together by using a special pointer (tag). In non-linear Data structure the relationship of adjacency is not maintained between the data items.

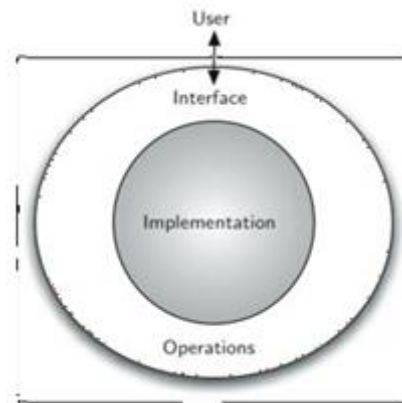
Operations applied on non-linear data structures:

The following list of operations applied on non-linear data structures.

1. Add elements
2. Delete elements
3. Display the elements
4. Sort the list of elements
5. Search for a data element

Abstract Data Type:

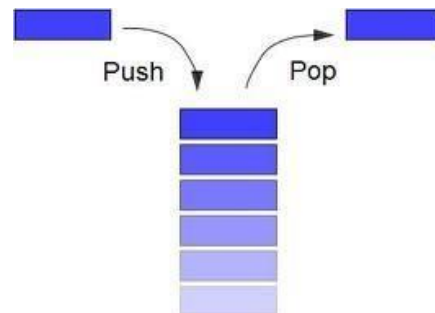
An abstract data type, sometimes abbreviated ADT, is a logical description of how we view the data and the operations that are allowed without regard to how they will be implemented. This means that we are concerned only with what data is representing and not with how it will eventually be constructed. By providing this level of abstraction, we are creating an encapsulation around the data. The idea is that by encapsulating the details of the implementation, we are hiding them from the user's view. This is called information hiding.



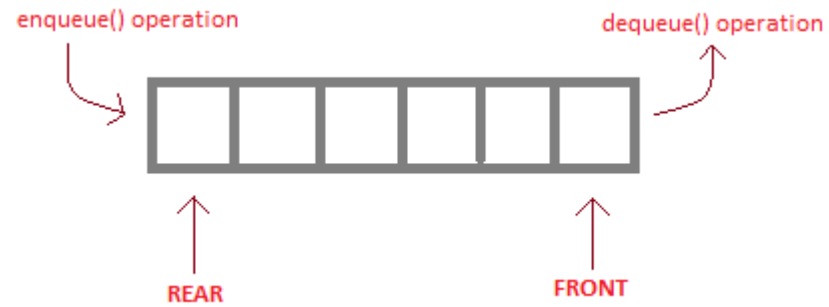
[Abstract Data Type (ADT)]

Description of various Compound Data Structures:

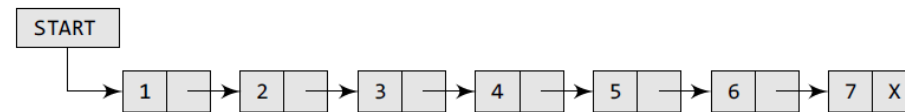
Stack:



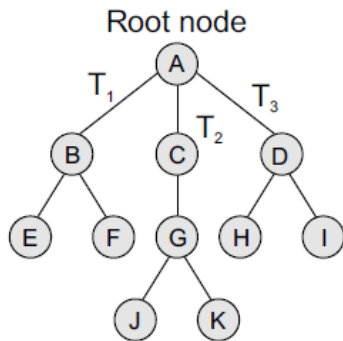
Queue:



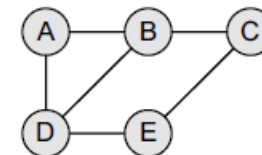
Linked List:



Tree:



Graph:



Algorithms:

Structure and Properties of Algorithm:

An algorithm has the following structure

1. Input Step
2. Assignment Step
3. Decision Step
4. Repetitive Step
5. Output Step

An algorithm is endowed with the following properties:

1. **Finiteness:** An algorithm must terminate after a finite number of steps.
2. **Definiteness:** The steps of the algorithm must be precisely defined or unambiguously specified.
3. **Generality:** An algorithm must be generic enough to solve all problems of a particular class.
4. **Effectiveness:** the operations of the algorithm must be basic enough to be put down on pencil and paper.
5. **Input-Output:** The algorithm must have certain initial and precise inputs, and outputs that may be generated both at its intermediate and final steps.

Different Approaches to Design an Algorithm:

An algorithm does not enforce a language or mode for its expression but only demands to meet its properties.

Practical Algorithm Design Issues:

1. **To save time (Time Complexity):** A program that runs faster is a better program.
2. **To save space (Space Complexity):** A program that saves space over a competing program is considerable desirable.

Efficiency of algorithms:

The performances of algorithms can be measured on the scales of **time** and **space**. The performance of a program is the amount of computer memory and time needed to run a program. We use two approaches to determine the performance of a program. One is analytical and the other is experimental. In performance analysis we use analytical methods, while in performance measurement we conduct experiments.

Time Complexity: The time complexity of an algorithm or a program is a function of the running time of the algorithm or a program. In other words, it is the amount of computer time it needs to run to completion.

Space Complexity: The space complexity of an algorithm or program is a function of the space needed by the algorithm or program to run to completion.