DSA

(Data Structures and Algorithms)

WHAT IS DATA:

Data is the collection of different numbers, symbols, and alphabets to represent information.

WHAT IS DATA STRUCTURE:

The group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently.

NEED OF DATA STRUCTURES:

As applications are getting complex and amount of data is increasing day by day, there may arise the following problems:

Processor speed: To handle very large amount of data, high speed processing is required, but as the data is growing day by day to the billions of files per entity, processor may fail to deal with that much amount of data.

Data Search: Consider an inventory size of 106 items in a store, If our application needs to search for a particular item, it needs to traverse 106 items every time, results in slowing down the search process.

TYPE OF DATA STRUCTURE:

1. Linear Data Structure
2. Non-Linear Data Structure.

1. **Linear data structure:** Elements are arranged in one dimension, also known as linear dimension. Examples- lists, stack, queue, etc.
2. **Non-linear data structure:** Elements are arranged in one-many, many-one and many-many dimensions. Examples- tree, graph, table, etc.

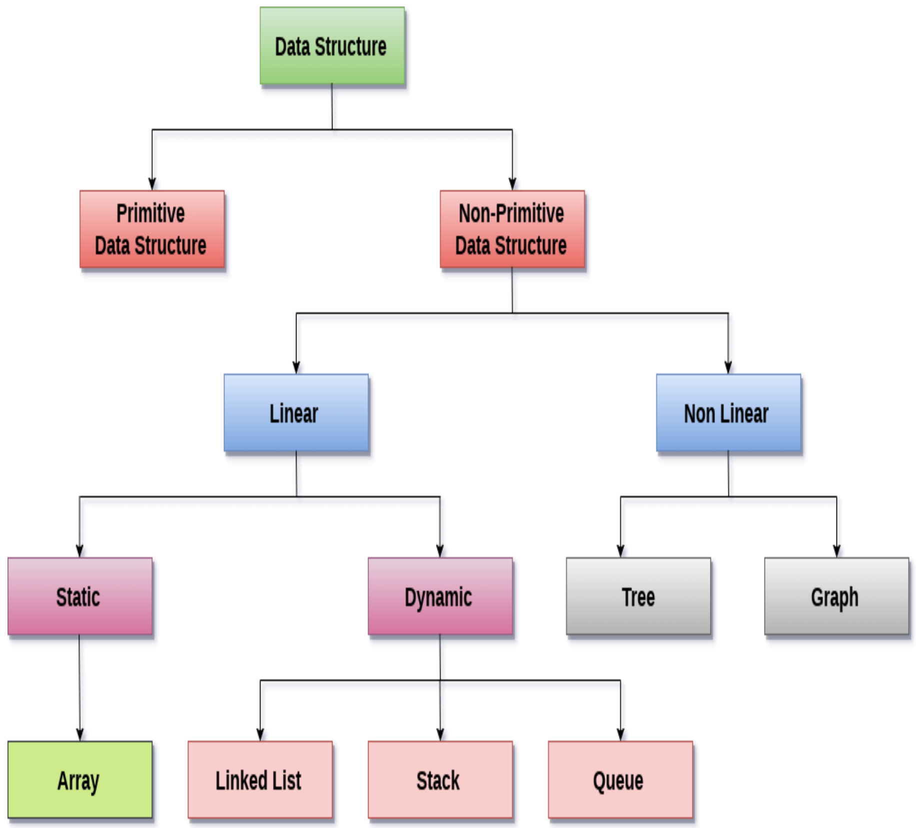
DATA STRUCTURES ARE USED IN VARIOUS FIELDS SUCH AS:

- Blockchain
- Genetics
- Graphics
- Image Processing
- Simulation
- Computer Design

ADVANTAGES OF DATA STRUCTURE:

- Helps in efficient storage of data in the storage device.
- Usage provides convenience while retrieving the data from storage device.
- Usage of proper data structure, can help programmer save lots of time or processing time while operations such as storage, retrieval or processing of data.

DATA STRUCTURE CLASSIFICATION:



COMMONLY USED DATA STRUCTURES:

1. Arrays
2. Stacks
3. Queues
4. Linked Lists
5. Trees
6. Graphs
7. Tries (they are effectively trees, but it’s still good to call them out separately).
8. Hash Tables

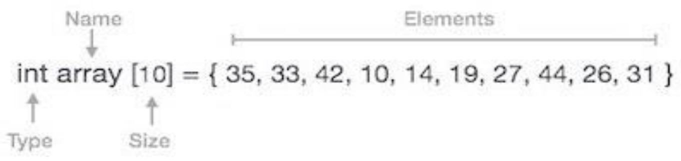
1. **ARRAY:**

Array is a container which can hold a fix number of items and these items should be of the same type. Following are the important terms to understand the concept of Array.

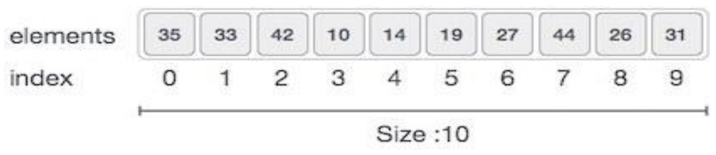
- **Element** − Each item stored in an array is called an element.
- **Index** − Each location of an element in an array has a numerical index, which is used to identify the element.

Array Representation

Arrays can be declared in various ways in different languages. For illustration, let's take C array declaration.



- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index.



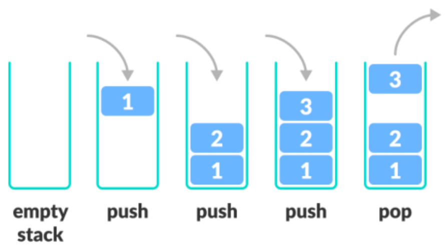
1. **STACK:**

Stack is **a linear data structure which follows a particular order in which the operations are performed**. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Basic Operations

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.

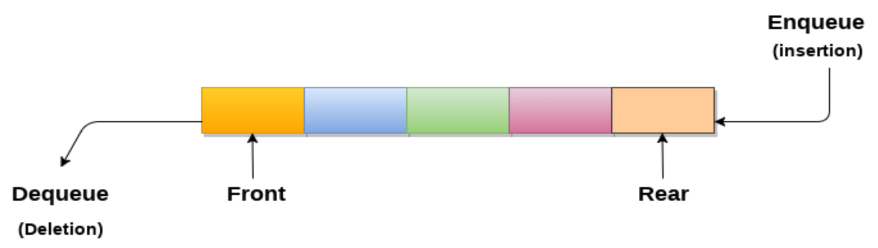
Stack Representation



2. QUEUE:

A queue can be defined as an ordered list which enables insert operations to be performed at one end called **REAR** and delete operations to be performed at another end called **FRONT**. Queue is referred to be as First In First Out list.

Queue Representation



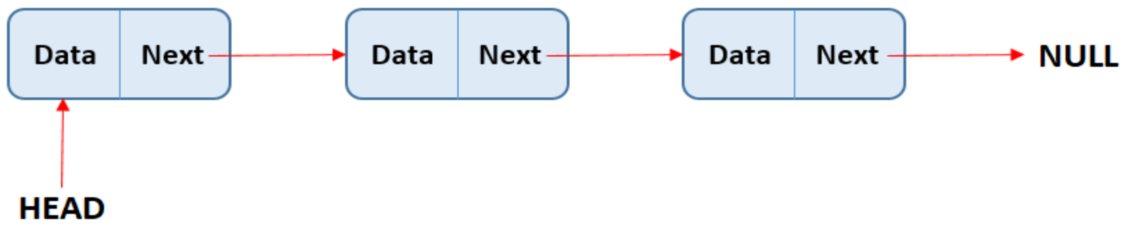
3. LINKED LIST:

A linked list is a sequence of data structures, which are connected together via links. It is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **LinkedList** − A Linked List contains the connection link to the first link called First

Linked List Representation

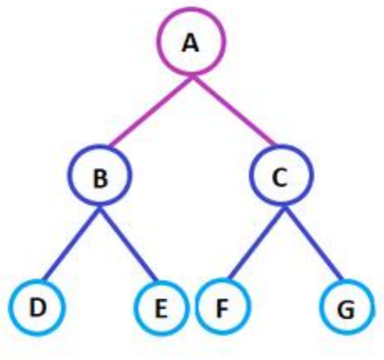
.



4. TREES:

Tree is a kind of hierarchal data arranged in a tree-like structure. It consists of a central node, structural nodes, and sub nodes, which are connected via edges. It is non-linear and a hierarchical data structure consisting of a collection of nodes such that each node of the tree stores a value, a list of references to nodes.

Tree Representation



Here, Node A is the root node. B is the parent of D and E. D and E are the siblings. D, E, and F are the leaf nodes. A and B are the ancestors of E.

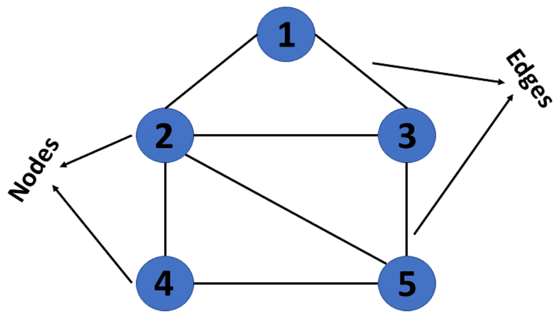
5. GRAPH:

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices.

Basic Operations

- **Add Vertex −** Adds a vertex to the graph.
- **Add Edge −** Adds an edge between the two vertices of the graph.
- **Display Vertex −** Displays a vertex of the graph.

Graph Representation



6. TRIE:

A Trie is an advanced data structure that is sometimes also known as **prefix tree or digital tree**. It is a tree that stores the data in an ordered and efficient way. We generally use trie's to store strings. Each node of a trie can have as many as 26 references (pointers).

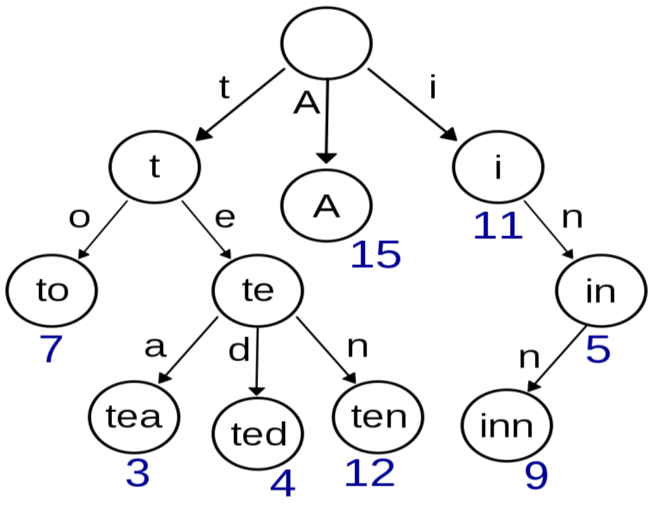
Each node of a trie consists of two things:

- A character
- A boolean value is used to implement whether this character represents the end of the word.

Basic operations of Trie

1. Insertion of a node
2. Searching a node
3. Deletion of a node

Tries Representation



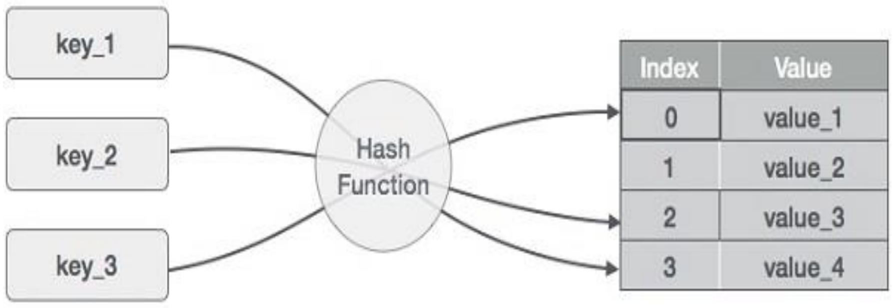
7. HASH TABLE:

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.

Basic Operations

- **Search** − Searches an element in a hash table.
- **Insert** − inserts an element in a hash table.
- **Delete** − Deletes an element from a hash table.

Hash Representation



WHAT IS AN ALGORITHM?

An algorithm is a set of well-defined instructions to solve a particular problem. It takes a set of input and produces a desired output.

CHARACTERISTICS OF AN ALGORITHM:

- **Input:** An algorithm has some input values. We can pass 0 or some input value to an algorithm.
- **Output:** We will get 1 or more output at the end of an algorithm.
- **Unambiguity:** An algorithm should be unambiguous which means that the instructions in an algorithm should be clear and simple.
- **Finiteness:** An algorithm should have finiteness. Here, finiteness means that the algorithm should contain a limited number of instructions, i.e., the instructions should be countable.
- **Effectiveness:** An algorithm should be effective as each instruction in an algorithm affects the overall process.
- **Language independent:** An algorithm must be language-independent so that the instructions in an algorithm can be implemented in any of the languages with the same output.

IMPORTANCE OF ALGORITHMS:

1. **Theoretical importance:** When any real-world problem is given to us and we break the problem into small-small modules. To break down the problem, we should know all the theoretical aspects.
2. **Practical importance:** As we know that theory cannot be completed without the practical implementation. So, the importance of algorithm can be considered as both theoretical and practical.

ISSUES OF ALGORITHMS:

- **How to design algorithms:** As we know that an algorithm is a step-by-step procedure so we must follow some steps to design an algorithm.
- **How to analyze algorithm efficiency**

THE MAJOR CATEGORIES OF ALGORITHMS ARE GIVEN BELOW:

- **Sort:** Algorithm developed for sorting the items in a certain order.
- **Search:** Algorithm developed for searching the items inside a data structure.
- **Delete:** Algorithm developed for deleting the existing element from the data structure.
- **Insert:** Algorithm developed for inserting an item inside a data structure.
- **Update:** Algorithm developed for updating the existing element inside a data structure.

ALGORITHM COMPLEXITY:

- **Time complexity:** The time complexity of an algorithm is the amount of time required to complete the execution. It is denoted by the big O notation. Here, big O notation is the asymptotic notation to represent the time complexity. The time complexity is mainly calculated by counting the number of steps to finish the execution.
- **Space complexity:** An algorithm's space complexity is the amount of space required to solve a problem and produce an output. Similar to the time complexity, space complexity is also expressed in big O notation.

TYPES OF ALGORITHMS:

1. **Search algorithm**

On each day, we search for something in our day to day life. Similarly, with the case of computer, huge data is stored in a computer that whenever the user asks for any data then the computer searches for that data in the memory and provides that data to the user. There are mainly two techniques available to search the data in an array:

   a. Linear search
   b. Binary search

      a. Linear search: Linear search is a very simple algorithm that starts searching for an element or a value from the beginning of an array until the required element is not found. It compares the element to be searched with all the elements in an array, if the match is found, then it returns the index of the element else it returns -1. This algorithm can be implemented on the unsorted list.
      b. Binary Search: A Binary algorithm is the simplest algorithm that searches the element very quickly. It is used to search the element from the sorted list. The elements must be stored in sequential order or the sorted manner to implement the binary algorithm. Binary search cannot be implemented if the elements are stored in a random manner. It is used to find the middle element of the list.

2. **Sorting algorithms**

Sorting algorithms are used to rearrange the elements in an array or a given data structure either in an ascending or descending order. The comparison operator decides the new order of the elements.