A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Below is an overview of some popular linear data structures.

Array

Array is a data structure used to store homogeneous elements at contiguous locations. Size of an array must be provided before storing data.

Linked List

A linked list is a linear data structure (like arrays) where each element is a separate object. Each element (that is node) of a list is comprising of two items – the data and a reference to the next node.

Types of Linked List :

1. **Singly Linked List :** In this type of linked list, every node stores address or reference of next node in list and the last node has next address or reference as NULL. For example 1->2->3->4->NULL

2. **Doubly Linked List :** In this type of Linked list, there are two references associated with each node, One of the reference points to the next node and one to the previous node. Advantage of this data structure is that we can traverse in both the directions and for deletion we don’t need to have explicit access to previous node. Eg. NULL<-1<->2<->3->NULL

3. **Circular Linked List :** Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list. Advantage of this data structure is that any node can be made as starting node. This is useful in implementation of circular queue in linked list. Eg. 1->2->3->1 [The next pointer of last node is pointing to the first]

Stack

A stack or LIFO (last in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the last element that was added. In stack both the operations of push and pop takes place at the same end that is top of the stack. It can be implemented by using both array and linked list.

Queue

A queue or FIFO (first in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: enqueue, the process of adding an element to the collection.(The element is added from the rear side) and dequeue, the process of removing the first element that was added. (The element is removed from the front side). It can be implemented by using both array and linked list.

**Binary Tree**

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.

A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. It is implemented mainly using Links.

**Binary Tree Representation:** A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL. A Binary Tree node contains following parts.

1. Data

2. Pointer to left child

3. Pointer to right child

A Binary Tree can be traversed in two ways:

Depth First Traversal: Inorder (Left-Root-Right), Preorder (Root-Left-Right) and Postorder (Left-Right-Root)

Breadth First Traversal: Level Order Traversal

**Binary Search Tree**

In Binary Search Tree is a Binary Tree with following additional properties:

1. The left subtree of a node contains only nodes with keys less than the node’s key.

2. The right subtree of a node contains only nodes with keys greater than the node’s key.

3. The left and right subtree each must also be a binary search tree.

**Binary Heap**

A Binary Heap is a Binary Tree with following properties.

1) It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to Min Heap. It is mainly implemented using array.

The Heap data structure can be used to efficiently find the k smallest (or largest) elements in an array, merging k sorted arrays, median of a stream, etc.

Heap is a special data structure and it cannot be used for searching of a particular element.

**HashingHash Function:**

A function that converts a given big input key to a small practical integer value. The mapped integer value is used as an index in hash table. A good hash function should have following properties

1) Efficiently computable.

2) Should uniformly distribute the keys (Each table position equally likely for each key)

Hash Table: An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.

Collision Handling: Since a hash function gets us a small number for a key which is a big integer or string, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

Chaining:The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.

Open Addressing: In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table.

Hashing seems better than BST for all the operations. But in hashing, elements are unordered and in BST elements are stored in an ordered manner. Also, BST is easy to implement but hash functions can sometimes be very complex to generate. In BST, we can also efficiently find floor and ceil of values.

Graph

Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

V -> Number of Vertices.

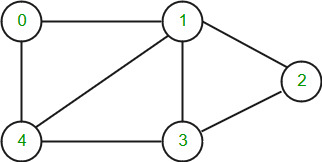
E -> Number of Edges.

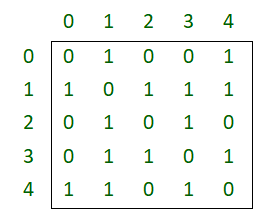
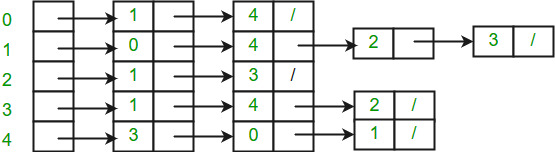
Graph can be classified on the basis of many things, below are the two most common classifications :

1. **Direction :**Undirected Graph : The graph in which all the edges are bidirectional.Directed Graph : The graph in which all the edges are unidirectional.
2. **Weight :**Weighted Graph : The Graph in which weight is associated with the edges.Unweighted Graph : The Graph in which their is no weight associated to the edges.

Graph can be represented in many ways, below are the two most common representations :

Let us take below example graph two see two representations of graph.



1.   
   *Adjacency Matrix Representation of the above graph*
2. *  
   Adjacency List Representation of the above Graph*

**Time Complexities in case of Adjacency Matrix :**

Traversal :(By BFS or DFS) O(V^2)

Space : O(V^2)

**Time Complexities in case of Adjacency List :**

Traversal :(By BFS or DFS) O(V + E)

Space : O(V+E)

**Examples :** The most common example of the graph is to find shortest path in any network. Used in google maps or bing. Another common use application of graph are social networking websites where the friend suggestion depends on number of intermediate suggestions and other things.

**Trie**

Trie is an efficient data structure for searching words in dictionaries, search complexity with Trie is linear in terms of word (or key) length to be searched. If we store keys in binary search tree, a well balanced BST will need time proportional to M \* log N, where M is maximum string length and N is number of keys in tree. Using trie, we can search the key in O(M) time. So it is much faster than BST.

Hashing also provides word search in O(n) time on average. But the advantages of Trie are there are no collisions (like hashing) so worst case time complexity is O(n). Also, the most important thing is Prefix Search. With Trie, we can find all words beginning with a prefix (This is not possible with Hashing). The only problem with Tries is they require a lot of extra space. Tries are also known as radix tree or prefix tree.

The Trie structure can be defined as follows :

struct trie\_node

{

int value; /\* Used to mark leaf nodes \*/

trie\_node\_t \*children[ALPHABET\_SIZE];

};

root

/ \ \

t a b

| | |

h n y

| | \ |

e s y e

/ | |

i r w

| | |

r e e

|

r