**<https://www.geeksforgeeks.org/data-structures/?ref=lbp>**

**Overview of Data Structures | Set 1 (Linear Data Structures)**

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.

[**1. Array**](https://www.geeksforgeeks.org/overview-of-data-structures-set-1-linear-data-structures/#code1)   
[**2. Linked List**](https://www.geeksforgeeks.org/overview-of-data-structures-set-1-linear-data-structures/#code2)   
[**3. Stack**](https://www.geeksforgeeks.org/overview-of-data-structures-set-1-linear-data-structures/#code3)   
[**4. Queue**](https://www.geeksforgeeks.org/overview-of-data-structures-set-1-linear-data-structures/#code4)

**Array**

The array is a data structure used to store homogeneous elements at contiguous locations. The 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. A linked list is made up of two items that are data and a reference to the next node. A reference to the next node is given with the help of pointers and data is the value of a node. Each node contains data and links to the other nodes. It is an ordered collection of data elements called a node and the linear order is maintained by pointers. It has an upper hand over the array as the number of nodes i.e. the size of the linked list is not fixed and can grow and shrink as and when required, unlike arrays.

class LinkedList {

    // The first node(head) of the linked list

    // Will be an object of type Node (null by default)

    Node head;

    class Node {

        int data;

        Node next;

        // Constructor to create a new node

        Node(int d) { data = d; }

    }

}

Types of Linked List :   
1. **Singly Linked List:**In this type of linked list, every node stores address or reference of the next node in the list and the last node has the 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. The advantage of this data structure is that we can traverse in both directions and for deletion, we don’t need to have explicit access to the 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 a doubly circular linked list. The advantage of this data structure is that any node can be made as starting node. This is useful in the implementation of the circular queues in the linked list. Eg. 1->2->3->1 [The next pointer of the last node is pointing to the first]

4. **Circular Doubly Linked List:**The circular doubly linked list is a combination of the doubly linked list and the circular linked list. It means that this linked list is bidirectional and contains two pointers and the last pointer points to the first pointer.

**Example:** Consider the previous example where we made an array of marks of students. Now if a new subject is added to the course, its marks are also to be added to the array of marks. But the size of the array was fixed and it is already full so it can not add any new element. If we make an array of a size lot more than the number of subjects it is possible that most of the array will remain empty. We reduce the space wastage Linked List is formed which adds a node only when a new element is introduced. Insertions and deletions also become easier with a linked list.   
One big drawback of a linked list is, random access is not allowed. With arrays, we can access i’th element in O(1) time. In the linked list, it takes Θ(i) time.

**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(adds an element to the collection), and pop(removes the last element). In stack both the operations of push and pop take place at the same end that is top of the stack. It can be implemented by using both array and linked list.

**Example:** Stacks are used for maintaining function calls (the last called function must finish execution first), we can always remove recursion with the help of stacks. Stacks are also used in cases where we have to reverse a word, check for balanced parenthesis, and in editors where the word you typed the last is the first to be removed when you use undo operation. Similarly, to implement back functionality in web browsers.

**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.

# Overview of Data Structures | Set 2 (Binary Tree, BST, Heap and Hash)

[**5. Binary Tree**](https://www.geeksforgeeks.org/overview-of-data-structures-set-2-binary-tree-bst-heap-and-hash/#code5)   
[**6. Binary Search Tree**](https://www.geeksforgeeks.org/overview-of-data-structures-set-2-binary-tree-bst-heap-and-hash/#code6)   
[**7. Binary Heap**](https://www.geeksforgeeks.org/overview-of-data-structures-set-2-binary-tree-bst-heap-and-hash/#code7)   
[**9. Hashing**](https://www.geeksforgeeks.org/overview-of-data-structures-set-2-binary-tree-bst-heap-and-hash/#code9)

**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 the tree. If the tree is empty, then the value of the root is NULL. A Binary Tree node contains the following parts.   
1. Data   
2. Pointer to left child   
3. Pointer to the 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

**Applications of Binary Tree:**

* Huffman coding trees are used in data compression algorithms.
* Priority Queue is another application of binary tree that is used for searching maximum or minimum in O(logn) time complexity.
* In compilers, Expression Trees are used which is an application of binary tree.

**Examples:** One reason to use binary trees or trees, in general, is for the things that form a hierarchy. They are useful in File structures where each file is located in a particular directory and there is a specific hierarchy associated with files and directories. Another example where Trees are useful is storing hierarchical objects like JavaScript Document Object Model considers HTML page as a tree with nesting of tags as parent-child relations.

**Binary Search Tree**

Binary Search Tree (BST) is a tree whose main function is to search a specific element.  
Binary Search Tree is a Binary Tree with the 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 Search Tree Declaration**

struct BinarySearchTree{

int data;

struct BinarySearchTree\* left;

struct BinarySearchTree\* right;

};

**Examples:** Its main use is in search applications where data is constantly entering/leaving and data needs to be printed in sorted order. For example in implementation in E-commerce websites where a new product is added or product goes out of stock and all products are listed in sorted order.

**Binary Heap**   
A Binary Heap is a Binary Tree with the 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 the 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 an array.

**Example:** Used in implementing efficient priority queues, which in turn are used for scheduling processes in operating systems. Priority Queues are also used in Dijkstra’s and Prim’s graph algorithms.   
The Heap data structure can be used to efficiently find the k smallest (or largest) elements in an array, merging k sorted arrays, a median of a stream, etc.   
Heap is a special data structure and it cannot be used for searching a particular element.

**Hashing:**Hashing is a popular technique for storing and retrieving data as fast as possible. The main reason behind using hashing is that it gives optimal results as it performs optimal searches.

**Why to use Hashing? :**

If you observe carefully, in a balanced binary search tree, if we try to search , insert or delete any element then the time complexity for the same is O(logn). Now there might be a situation when our applications want to do the same operations in a faster way i.e. in a more optimized way and here hashing comes into play. In hashing, all the above operations can be performed in O(1) i.e. constant time. It is important to understand that the worst case time complexity for hashing remains O(n) but the average case time complexity is O(1).

**Hash Function:** A function that converts a given big phone number to a small practical integer value. The mapped integer value is used as an index in hash table. So, in simple terms we can say that a hash function is used to transform a given key into a specific slot index. Its main job is to map each and every possible key into a unique slot index. If every key is mapped into a unique slot index, then the hash function is known as a perfect hash function. It is very difficult to create a perfect hash function but our job as a programmer is to create such a hash function with the help of which the number of collisions are as few as possible. Collision is discussed ahead.  
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).

3) Should minimize collisions

4) Should have a high load factor(number of items in table divided by size of the table).

For example for phone numbers a bad hash function is to take first three digits. A better function is consider last three digits. Please note that this may not be the best hash function. There may be better ways.

**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.  In simple terms, we can say that hash table is a generalization of array. Hash table gives the functionality in which a collection of data is stored in such a way that it is easy to find those items later if required. This makes searching of an element very efficient.

**Collision Handling:** Since a hash function gets us a small number for a key which is a big integer or string, there is the possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the 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 the hash table point to a linked list of records that have the 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.

**Example:** Hashing can be used to remove duplicates from a set of elements. Can also be used to find the frequency of all items. For example, in web browsers, we can check visited URLs using hashing. In firewalls, we can use hashing to detect spam. We need to hash IP addresses. Hashing can be used in any situation where want search() insert() and delete() in O(1) time.

# Overview of Data Structures | Set 3 (Graph, Trie, Segment Tree and Suffix Tree)

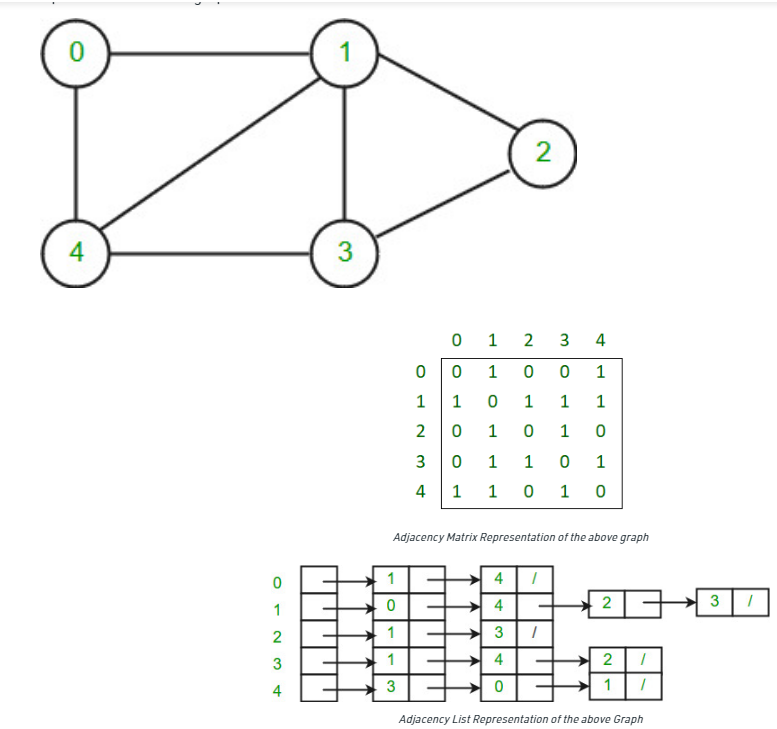
**Graph:**Graph is a data structure that consists of the following two components:

1. A finite set of vertices is also called nodes.
2. A finite set of ordered pairs of the form (u, v) is called an edge. The pair is ordered because (u, v) is not the same as (v, u) in the case of a directed graph(di-graph). The pair of forms (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. The 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 there is no weight associated with the edges.

Graphs can be represented in many ways, below are the two most common representations: Let us take the below example graph to see two representations of the graph.



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

T