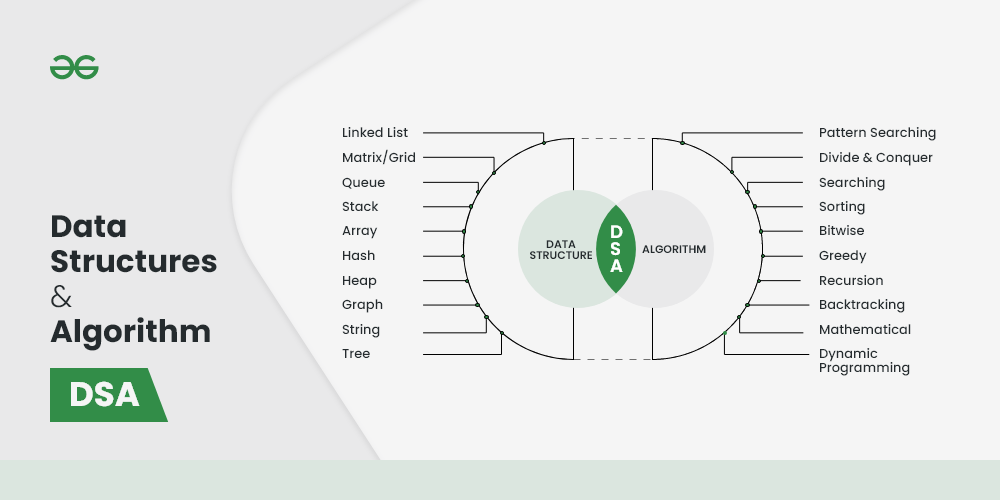
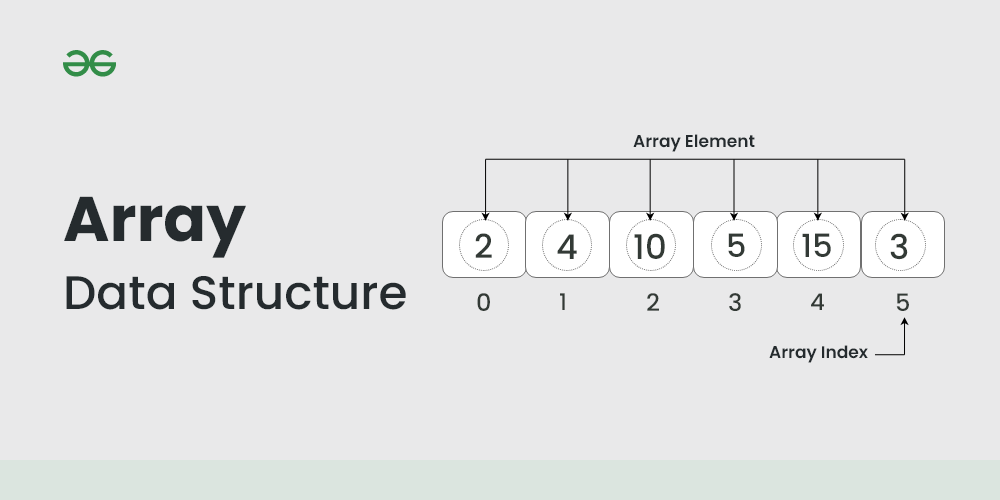
**DSA**



1. **Array:**

The most basic yet important data structure is the array. It is a linear data structure. An array is a collection of homogeneous data types where the elements are allocated contiguous memory. Because of the contiguous allocation of memory, any element of an array can be accessed in constant time. Each array element has a corresponding index number.



**Basic terminologies of array**

Array Index: In an array, elements are identified by their indexes. Array index starts from 0.

Array element: Elements are items stored in an array and can be accessed by their index.

Array Length: The length of an array is determined by the number of elements it can contain.

**Types of Array operations:**

Traversal: Traverse through the elements of an array.

Insertion: Inserting a new element in an array.

Deletion: Deleting element from the array.

Searching: Search for an element in the array.

Sorting: Maintaining the order of elements in the array.

Here are some topics about array which you must learn:

Reverse Array – Reverse an array means shifting the elements of an array in a reverse manner i.e., the last element becomes the first element, second last element becomes the second element, and so on.

[(https://www.geeksforgeeks.org/write-a-program-to-reverse-an-array-or-string/)]((https:/www.geeksforgeeks.org/write-a-program-to-reverse-an-array-or-string/))

Code : C, Java, Python

Rotation of Array – Rotation of array means shifting the elements of an array in a circular manner i.e., in the case of right circular shift the last element becomes the first element, and all other element moves one point to the right.

(<https://www.geeksforgeeks.org/array-rotation/>)

Code : C++, Java, Python

Rearranging an array – Rearrangement of array elements suggests the changing of an initial order of elements following some conditions or operations.

<https://www.geeksforgeeks.org/rearrange-array-arri/>

Code : C++, Java, Python

Advantages of using Arrays:

* Arrays allow random access to elements. This makes accessing elements by position faster.
* Arrays have better cache locality which makes a pretty big difference in performance.

Disadvantages of Array:

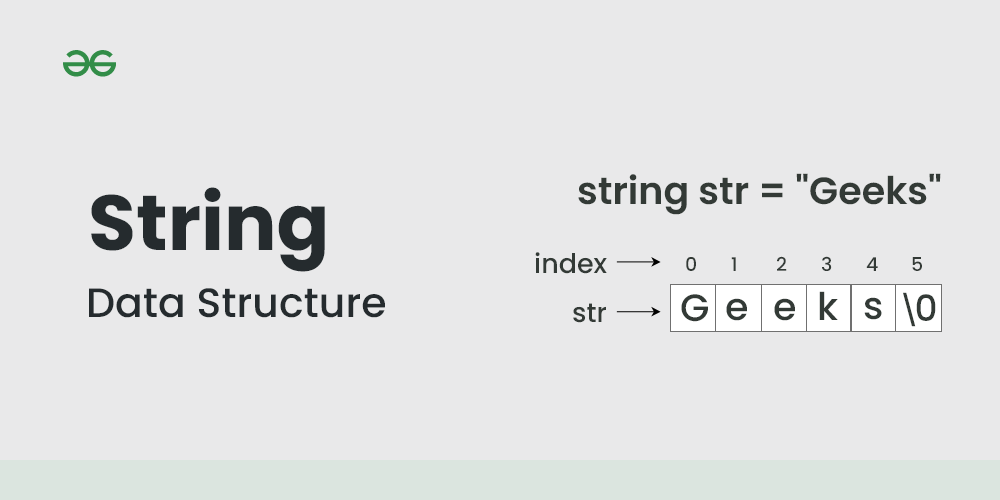
* As arrays have a fixed size, once the memory is allocated to them, it cannot be increased or decreased, making it impossible to store extra data if required. An array of fixed size is referred to as a static array.
* Allocating less memory than required to an array leads to loss of data.
* An array is homogeneous in nature so, a single array cannot store values of different data types.

Application of Array:

* They are used in the implementation of other data structures such as array lists, heaps, hash tables, vectors, and matrices.
* Database records are usually implemented as arrays.
* It is used in lookup tables by computer.

1. **String:**

A string is also a type of array. It can be interpreted as an array of characters. But it has some special characteristics like the last character of a string is a null character to denote the end of the string. Also, there are some unique operations, like concatenation which concatenates two strings into one.



Here we are providing you with some must-know concepts of string:

1. **Concatenation of Strings**

The process of combining more than one string together is known as Concatenation. String Concatenation is the technique of combining two strings.

There are two ways to concatenate two strings:

1. String concatenation without using any inbuilt methods:

Below is the algorithm for the Concatenation of two strings:

Algorithm: CONCATENATE (STR1, STR2, STR3)

LEN1 = LENGTH(STR1).

LEN2 = LENGTH(STR2).

SET I = 0.

Repeat Steps 5 and 6 while I < LEN1-1:

STR3[I] = STR1[I].

SET I = I+1.

SET J = 0.

Repeat Steps 9 to 11 while I < (LEN1 + LEN2 - 2):

STR3[I] = STR2[J].

J = J+1.

I = I+1.

Exit.

1. String concatenation using inbuilt methods:

<https://www.geeksforgeeks.org/string-concatenation-in-cpp/>

<https://www.geeksforgeeks.org/difference-between-concat-and-operator-in-java/>

<https://www.geeksforgeeks.org/python-string-concatenation/>

**2. Find in String**

* A very basic operation performed on Strings is to find something in the given whole string. Now, this can be to find a given character in a string, or to find a complete string in another string.

a) Find a character in string:

* Given a string and a character, your task is to find the first position of the character in the string. These types of problems are very competitive programming where you need to locate the position of the character in a string.

b) Find a substring in another string:

* Consider there to be a string of length N and a substring of length M. Then run a nested loop, where the outer loop runs from 0 to (N-M) and the inner loop from 0 to M. For every index check if the sub-string traversed by the inner loop is the given sub-string or not.

**3.Reverse and Rotation of a String**

Reverse operation is interchanging the position of characters of a string such that the first becomes the last, the second becomes the second last, and so on.

1. Rotations of a String:
2. Reverse a String:

**4. Palindrome String**

A string is said to be a palindrome if the reverse of the string is the same as the string.

<https://www.geeksforgeeks.org/c-program-check-given-string-palindrome/>

<https://www.geeksforgeeks.org/python-program-check-string-palindrome-not/>

<https://www.educative.io/answers/how-to-check-if-a-string-is-a-palindrome-in-java>

**Advantages of using String:**

* Versatility: Strings can store and manipulate text data, making them useful for a wide range of applications and programming tasks.
* Readability: Strings allow for clear representation and interpretation of human-readable text, making code easier to understand and maintain.

**Disadvantages of String:**

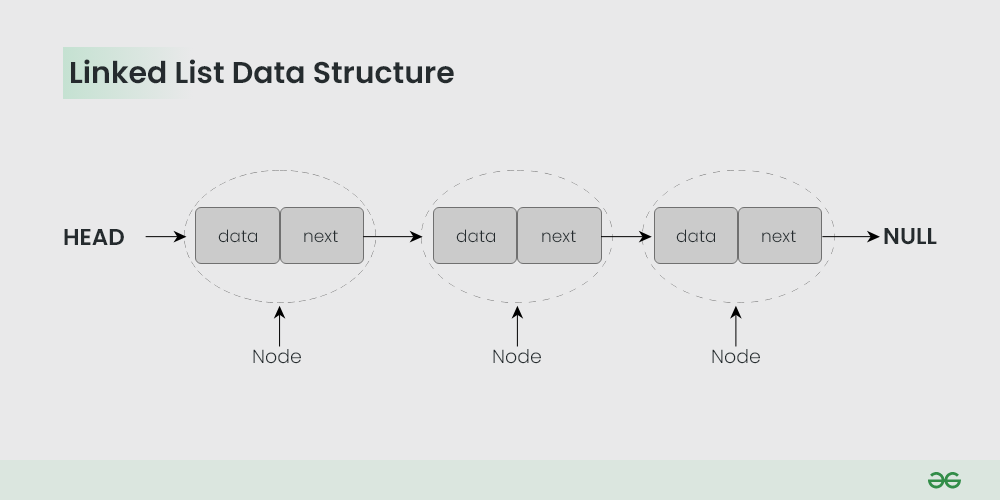
* Strings are generally slow in performing operations like input, output.
* In JAVA strings are immutable they cannot be modified or changed.

**Application of String:**

* Text Processing: Strings are extensively used for text processing tasks such as searching, manipulating, and analyzing textual data.
* Data Representation: Strings are fundamental for representing and manipulating data in formats like JSON, XML, and CSV.
* Encryption and Hashing: Strings are commonly used in encryption and hashing algorithms to secure sensitive data and ensure data integrity.

1. **Linked Lists**

The linked list is also a linear data structure. But Linked List is different from Array in its configuration. It is not allocated to contiguous memory locations. Instead, each node of the linked list is allocated to some random memory space and the previous node maintains a pointer that points to this node. So no direct memory access of any node is possible and it is also dynamic i.e., the size of the linked list can be adjusted at any time.



The topics which you must want to cover are:

Singly Linked List – In this, each node of the linked list points only to its next node.

Circular Linked List – This is the type of linked list where the last node points back to the head of the linked list.

Doubly Linked List – In this case, each node of the linked list holds two pointers, one point to the next node and the other points to the previous node.

Representation & code:

<https://www.geeksforgeeks.org/introduction-to-linked-list-data-structure-and-algorithm-tutorial/>

**Advantages of Linked Lists:**

* Dynamic nature: Linked lists are used for dynamic memory allocation.
* Memory efficient: Memory consumption of a linked list is efficient as its size can grow or shrink dynamically according to our requirements, which means effective memory utilization hence, no memory wastage.

**Disadvantages of Linked Lists:**

* Memory usage: The use of pointers is more in linked lists hence, complex and requires more memory.
* Accessing a node: Random access is not possible due to dynamic memory allocation.

**Applications of Linked List:**

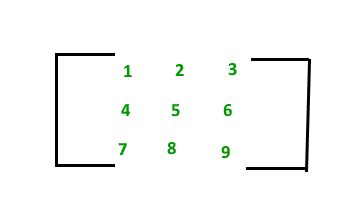
* Linear data structures such as stack, queue, and non-linear data structures such as hash maps, and graphs can be implemented using linked lists.
* Dynamic memory allocation: We use a linked list of free blocks.
* Implementation of graphs: Adjacency list representation of graphs is the most popular in that it uses linked lists to store adjacent vertices.

1. **Matrix/Grid**

A matrix represents a collection of numbers arranged in an order of rows and columns. It is necessary to enclose the elements of a matrix in parentheses or brackets.

**For example**:

A matrix with 9 elements is shown below. This Matrix M has 3 rows and 3 columns. Each element of matrix M can be referred to by its row and column number. For example, M[2][3] = 6.



**Basic Operations on Matrix:**

1. Rotate Matrix Elements

<https://www.geeksforgeeks.org/rotate-matrix-elements/>

1. Sort the given matrix

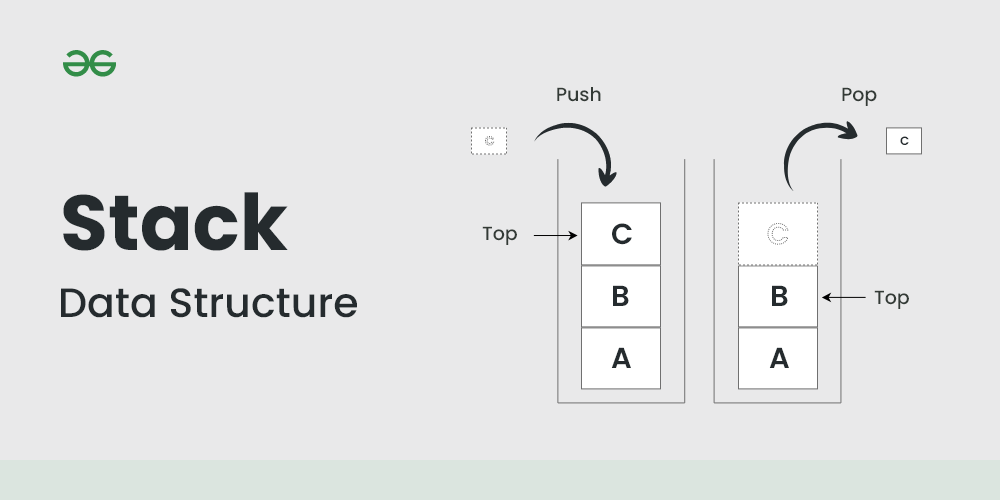
<https://www.geeksforgeeks.org/sort-given-matrix/>

1. Search element in a sorted matrix

<https://www.geeksforgeeks.org/search-element-sorted-matrix/>

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).



The reason why Stack is considered a complex data structure is that it uses other data structures for implementation, such as Arrays, Linked lists, etc. based on the characteristics and features of Stack data structure.

**Basic Operations on Stack**

In order to make manipulations in a stack, there are certain operations provided to us.

* push() to insert an element into the stack
* pop() to remove an element from the stack
* top() Returns the top element of the stack.
* isEmpty() returns true if stack is empty else false.
* size() returns the size of stack.

**Push:**

Adds an item to the stack. If the stack is full, then it is said to be an Overflow condition.

Algorithm for push:

begin

if stack is full

return

endif

else

increment top

stack[top] assign value

end else

end procedure

**Pop:**

Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

Algorithm for pop:

begin

if stack is empty

return

endif

else

store value of stack[top]

decrement top

return value

end else

end procedure

**Top:**

Returns the top element of the stack.

Algorithm for Top:

begin

return stack[top]

end procedure

**isEmpty:**

Returns true if the stack is empty, else false.

Algorithm for isEmpty:

begin

if top < 1

return true

else

return false

end procedure

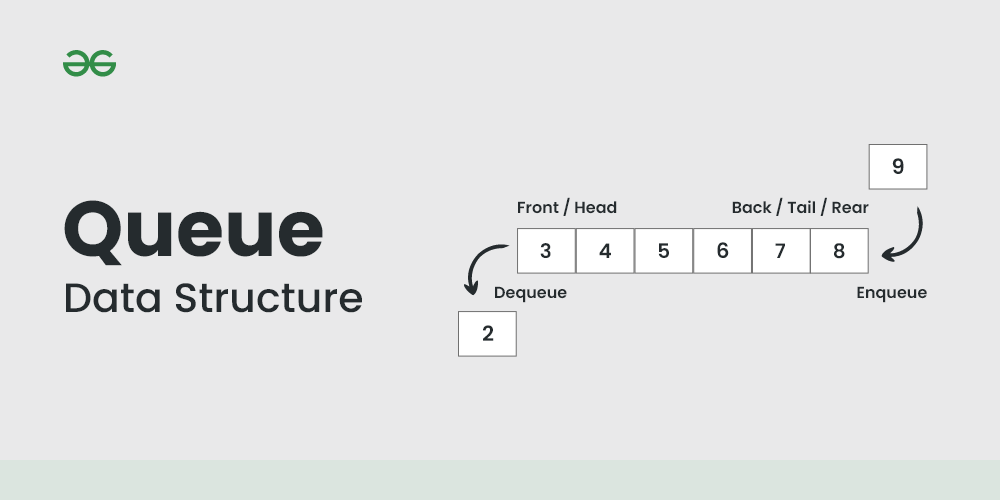
**Applications of the stack:**

* Infix to Postfix /Prefix conversion
* Redo-undo features at many places like editors, photoshop.
* Forward and backward features in web browsers

1. **Queue:**

A Queue is a linear structure which follows First In First Out (FIFO) approach in its individual operations.

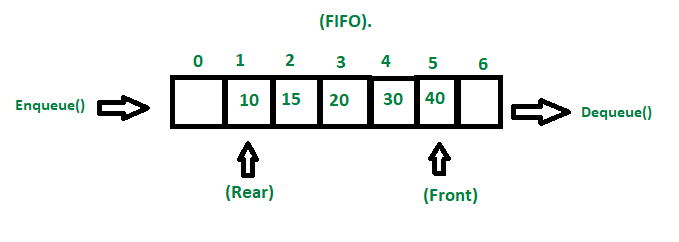
We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the delete operation is first performed on that.



**FIFO Principle of Queue:**

A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).

Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the front of the queue(sometimes, head of the queue), similarly, the position of the last entry in the queue, that is, the one most recently added, is called the rear (or the tail) of the queue. See the below figure.



**Characteristics of Queue:**

Queue can handle multiple data.

We can access both ends.

They are fast and flexible.

**Basic Operations for Queue in Data Structure:**

Some of the basic operations for Queue in Data Structure are:

* Enqueue() – Adds (or stores) an element to the end of the queue..
* Dequeue() – Removal of elements from the queue.
* Peek() or front()- Acquires the data element available at the front node of the queue without deleting it.
* rear() – This operation returns the element at the rear end without removing it.
* isFull() – Validates if the queue is full.
* isNull() – Checks if the queue is empty.

**Applications of Queue:**

Application of queue is common. In a computer system, there may be queues of tasks waiting for the printer, for access to disk storage, or even in a time-sharing system, for use of the CPU. Within a single program, there may be multiple requests to be kept in a queue, or one task may create other tasks, which must be done in turn by keeping them in a queue.

* It has a single resource and multiple consumers.
* It synchronizes between slow and fast devices.
* In a network, a queue is used in devices such as a router/switch and mail queue.
* Variations: dequeue, priority queue and double-ended priority queue.

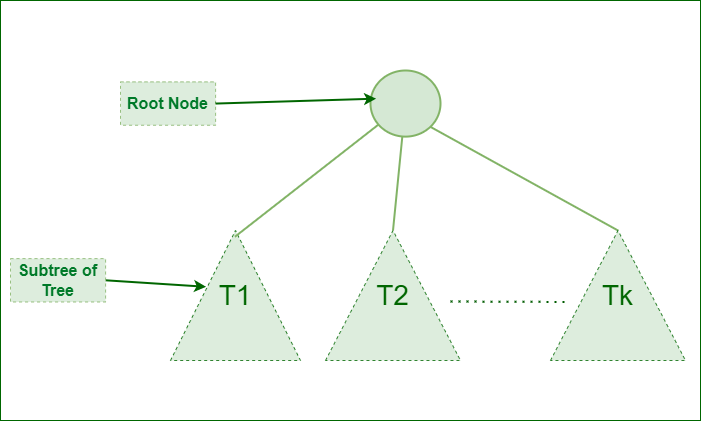
1. **Tree:**

A **tree data structure** is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.

The topmost node of the tree is called the root, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure.

## ****Representation of Tree Data Structure:****

A tree consists of a root, and zero or more subtrees T1, T2, … , Tk such that there is an edge from the root of the tree to the root of each subtree.



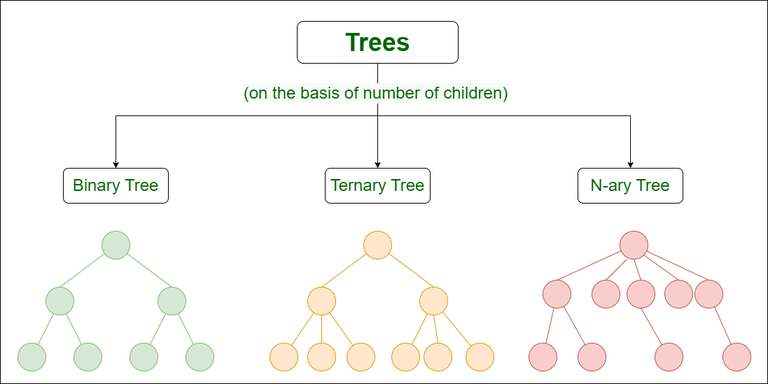
*struct Node  
{  
   int data;  
   struct Node \*first\_child;  
   struct Node \*second\_child;  
   struct Node \*third\_child;  
   .  
   .  
   .  
   struct Node \*nth\_child;  
};*

**Basic Terminologies In Tree Data Structure:**

**parent Node:** The node which is a predecessor of a node is called the parent node of that node.**{B}** is the parent node of **{D, E}**.

* **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: **{D, E}** are the child nodes of **{B}.**
* **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {A**}** is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.
* **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. **{K, L, M, N, O, P, G}** are the leaf nodes of the tree.
* **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node.**{A,B}** are the ancestor nodes of the node**{E}**
* **Descendant:** Any successor node on the path from the leaf node to that node. **{E,I}**are the descendants of the node **{B}.**
* **Sibling:** Children of the same parent node are called siblings.**{D,E}** are called siblings.
* **Level of a node:** The count of edges on the path from the root node to that node. The root node has level **0**.
* **Internal node:** A node with at least one child is called Internal Node.
* **Neighbour of a Node:** Parent or child nodes of that node are called neighbors of that node.
* **Subtree**: Any node of the tree along with its descendant.

## Types of Tree data structures:



* [**Binary tree**](https://www.geeksforgeeks.org/types-of-trees-in-data-structures/)**:** In a binary tree, each node can have a maximum of two children linked to it. Some common types of binary trees include full binary trees, complete binary trees, balanced binary trees, and degenerate or pathological binary trees.
* [**Ternary Tree**](https://www.geeksforgeeks.org/ternary-tree/)**:** A Ternary Tree is a tree data structure in which each node has at most three child nodes, usually distinguished as “left”, “mid” and “right”.
* [**N-ary Tree or Generic Tree**](https://www.geeksforgeeks.org/generic-treesn-array-trees/)**:** Generic trees are a collection of nodes where each node is a data structure that consists of records and a list of references to its children(duplicate references are not allowed). Unlike the linked list, each node stores the address of multiple nodes.

**Basic Operation Of Tree Data Structure:**

* **Create** – create a tree in the data structure.
* **Insert** − Inserts data in a tree.
* **Search** − Searches specific data in a tree to check whether it is present or not.
* **Traversal**:
  + **Preorder Traversal** – perform Traveling a tree in a pre-order manner in the data structure.
  + **In order Traversal** – perform Traveling a tree in an in-order manner.
  + **Post-order Traversal** –perform Traveling a tree in a post-order manner.

## Application of Tree Data Structure:

* **File System:** This allows for efficient navigation and organization of files.
* **Data Compression**:[Huffman coding](https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/) is a popular technique for data compression that involves constructing a binary tree where the leaves represent characters and their frequency of occurrence. The resulting tree is used to encode the data in a way that minimizes the amount of storage required.
* **Compiler Design:** In compiler design, a syntax tree is used to represent the structure of a program.
* **Database Indexing**: B-trees and other tree structures are used in database indexing to efficiently search for and retrieve data.

## Advantages of Tree Data Structure:

* Tree offer **Efficient Searching** Depending on the type of tree, with average search times of O(log n) for balanced trees like AVL.
* Trees provide a hierarchical representation of data, making it**easy to organize and navigate**large amounts of information.
* The recursive nature of trees makes them **easy to traverse and manipulate** using recursive algorithms.

## Disadvantages of Tree Data Structure:

* Unbalanced Trees, meaning that the height of the tree is skewed towards one side, which can lead to **inefficient search times.**
* Trees demand**more memory space requirements** than some other data structures like arrays and linked lists, especially if the tree is very large.
* The implementation and **manipulation of trees can be complex**and require a good understanding of the algorithms.

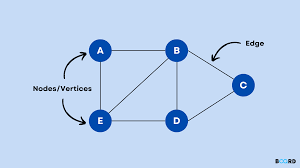
1. **Graph:**

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices( V ) and a set of edges( E ). The graph is denoted by G(V, E).

Graph data structures are a powerful tool for representing and analyzing complex relationships between objects or entities. They are particularly useful in fields such as social network analysis, recommendation systems, and computer networks.

**Components of a Graph**

* **Vertices:** Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.
* **Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph. Edges can connect any two nodes in any possible way. There are no rules. Sometimes, edges are also known as arcs. Every edge can be labelled/unlabelled.



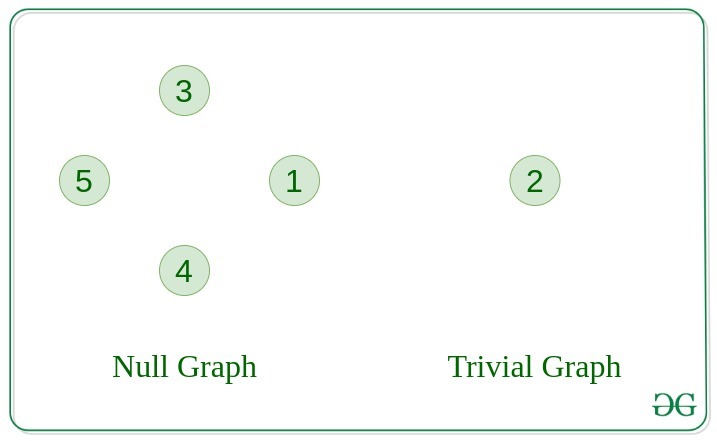
## ****Types Of Graph****

#### **1. Null Graph**

A graph is known as a null graph if there are no edges in the graph.

#### **2. Trivial Graph**

Graph having only a single vertex, it is also the smallest graph possible.

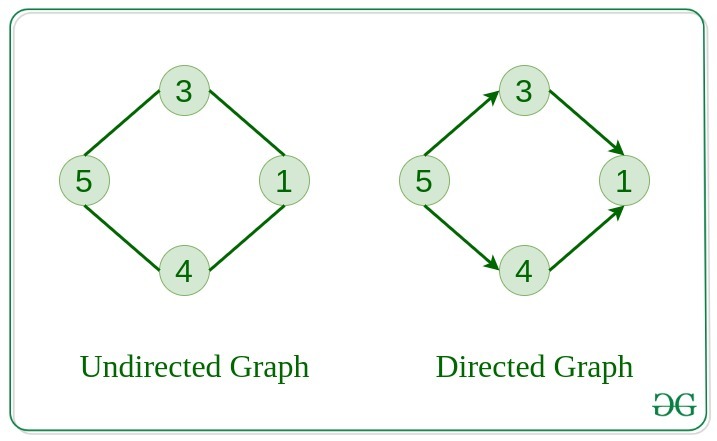
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630113942/null_graph_trivial.jpg)

#### **3. Undirected Graph**

A graph in which edges do not have any direction. That is the nodes are unordered pairs in the definition of every edge.

#### **4. Directed Graph**

A graph in which edge has direction. That is the nodes are ordered pairs in the definition of every edge.

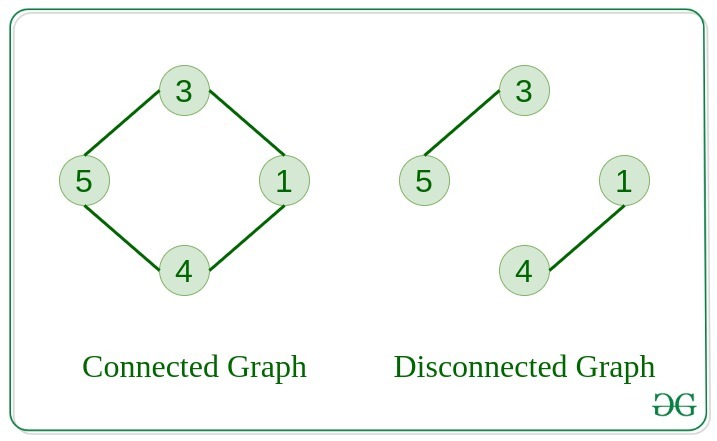
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630114438/directed.jpg)

#### **5. Connected Graph**

The graph in which from one node we can visit any other node in the graph is known as a connected graph.

#### **6. Disconnected Graph**

The graph in which at least one node is not reachable from a node is known as a disconnected graph.

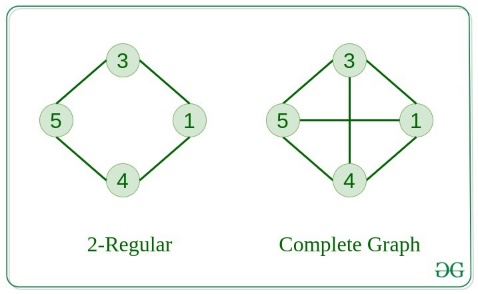
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630121400/connected1.jpg)

#### **7. Regular Graph**

The graph in which the degree of every vertex is equal to K is called K regular graph.

#### **8. Complete Graph**

The graph in which from each node there is an edge to each other node.

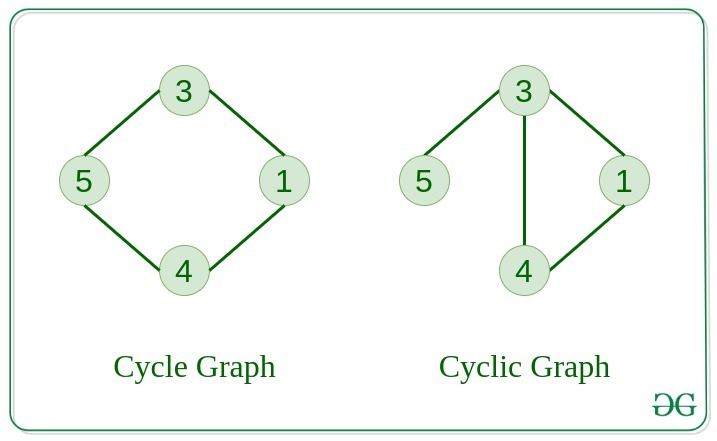
.[](https://media.geeksforgeeks.org/wp-content/uploads/20200630122008/regular.jpg)

#### **9. Cycle Graph**

The graph in which the graph is a cycle in itself, the degree of each vertex is 2.

#### **10. Cyclic Graph**

A graph containing at least one cycle is known as a Cyclic graph.

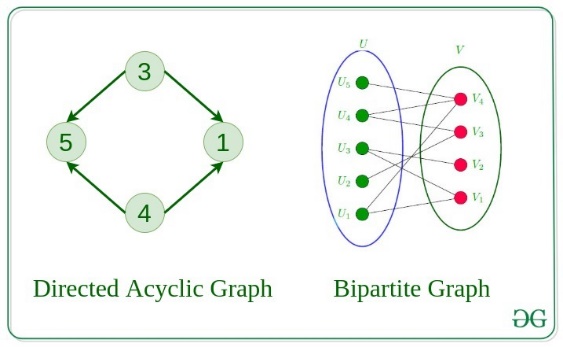
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630122225/cyclic.jpg)

#### **11. Directed Acyclic Graph**

A Directed Graph that does not contain any cycle.

#### **12. Bipartite Graph**

A graph in which vertex can be divided into two sets such that vertex in each set does not contain any edge between them.

[](https://media.geeksforgeeks.org/wp-content/uploads/20200630122552/bipartite1.jpg)

**13. Weighted Graph**

* A graph in which the edges are already specified with suitable weight is known as a weighted graph.
* Weighted graphs can be further classified as directed weighted graphs and undirected weighted graphs.

**Basic Operations on Graphs**

Below are the basic operations on the graph:

* Insertion of Nodes/Edges in the graph – Insert a node into the graph.
* Deletion of Nodes/Edges in the graph – Delete a node from the graph.
* Searching on Graphs – Search an entity in the graph.
* Traversal of Graphs – Traversing all the nodes in the graph.

### **Applications:**

* Graph data structures can be used to represent the interactions between players on a team, such as passes, shots, and tackles. Analyzing these interactions can provide insights into team dynamics and areas for improvement.
* Commonly used to represent social networks, such as networks of friends on social media.
* Graphs can be used to represent the topology of computer networks, such as the connections between routers and switches.
* Graphs are used to represent the connections between different places in a transportation network, such as roads and airports.
* **Neural Networks:**Vertices represent neurons and edges represent the synapses between them. Neural networks are used to understand how our brain works and how connections change when we learn. The human brain has about 10^11 neurons and close to 10^15 synapses.
* **Compilers:**Graphs are used extensively in compilers. They can be used for type inference, for so-called data flow analysis, register allocation, and many other purposes. They are also used in specialized compilers, such as query optimization in database languages.

### **Advantages:**

1. Graphs are a versatile data structure that can be used to represent a wide range of relationships and data structures.
2. They can be used to model and solve a wide range of problems, including pathfinding, data clustering, network analysis, and machine learning.

### **Disadvantages:**

1. Graphs can be complex and difficult to understand, especially for people who are not familiar with graph theory or related algorithms.
2. Creating and manipulating graphs can be computationally expensive, especially for very large or complex graphs.
3. Graph algorithms can be difficult to design and implement correctly, and can be prone to bugs and errors.