**Introduction to Graphs – Data Structure and Algorithms**

### Introduction:

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.

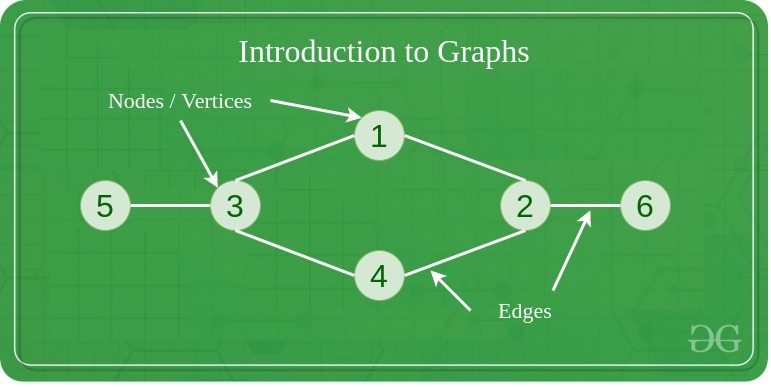
More formally a [Graph](https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/) is composed of a set of vertices( **V**) and a set of edges( **E**). The graph is denoted by **G(V, E).**

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, and computer

## 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.

[](https://media.geeksforgeeks.org/wp-content/uploads/20200630111809/graph18.jpg)

**The basic properties of a graph include:**

1. Vertices (nodes): The points where edges meet in a graph are known as vertices or nodes. A vertex can represent a physical object, concept, or abstract entity.
2. Edges: The connections between vertices are known as edges. They can be undirected (bidirectional) or directed (unidirectional).
3. ***Weight***: A weight can be assigned to an edge, representing the cost or distance between two vertices. A weighted graph is a graph where the edges have weights.
4. ***Degree***: The degree of a vertex is the number of edges that connect to it. In a directed graph, the ***in-degree*** of a vertex is **the number of edges that point to it**, and the ***out-degree*** is **the number of edges that start from it. Max degree of any vertex is n-1 if n is #vertices.**
5. ***Path***: A path is a sequence of vertices that are connected by edges. A ***simple path* does not contain any repeated vertices or edges**.
6. ***Cycle***: A cycle is a path that starts and ends at the same vertex. A simple cycle does not contain any repeated vertices or edges.

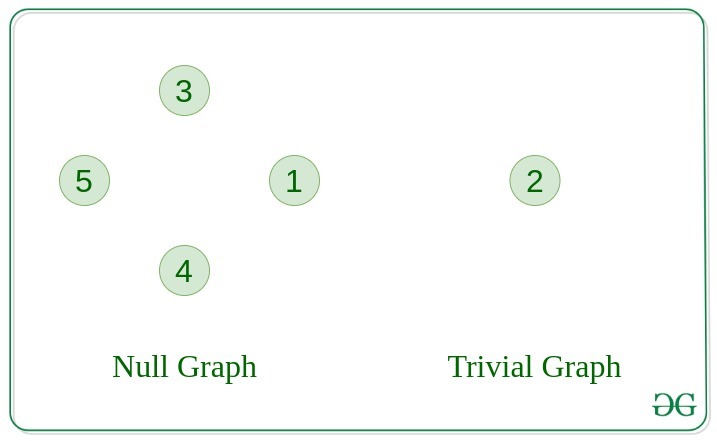
## ****Types Of Graph****

#### 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.

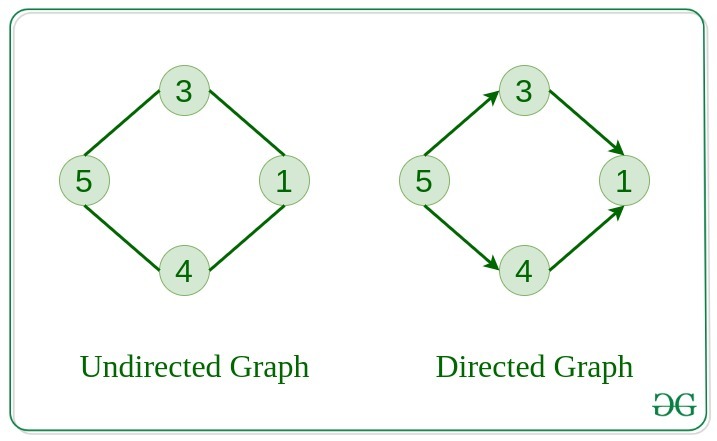
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630113942/null_graph_trivial.jpg)

#### 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.

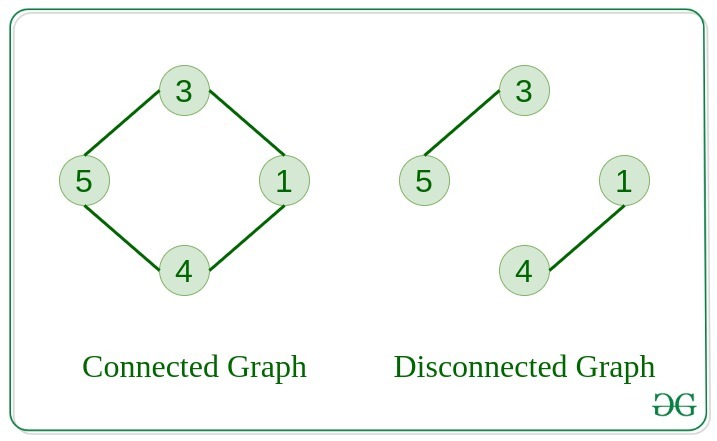
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630114438/directed.jpg)

#### 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)

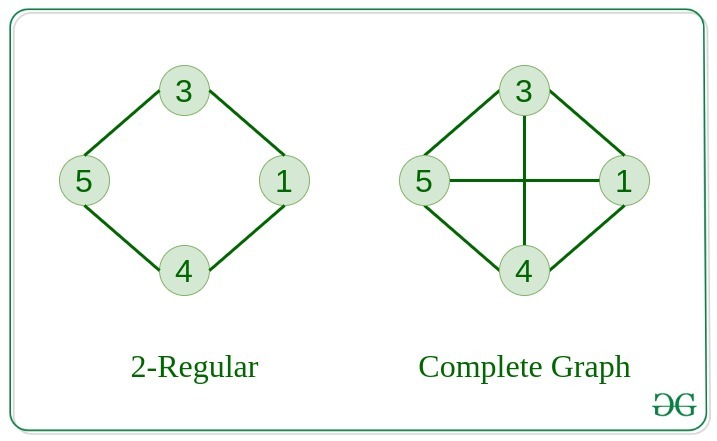
#### 7. Regular Graph

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.**

complete graph would have **n(n-1)/2 edges.**

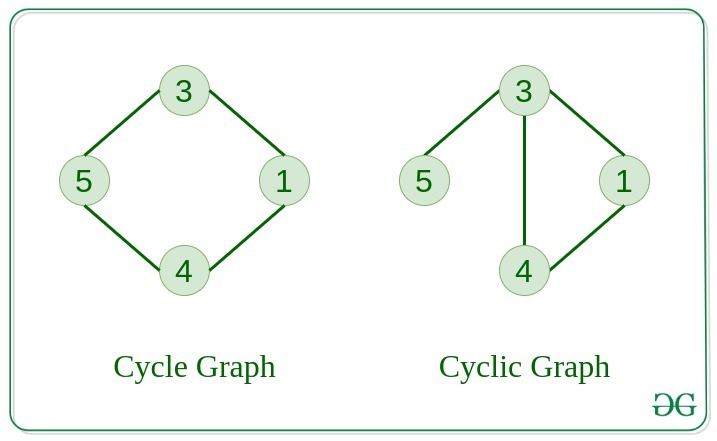
.[](https://media.geeksforgeeks.org/wp-content/uploads/20200630122008/regular.jpg)

#### 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.

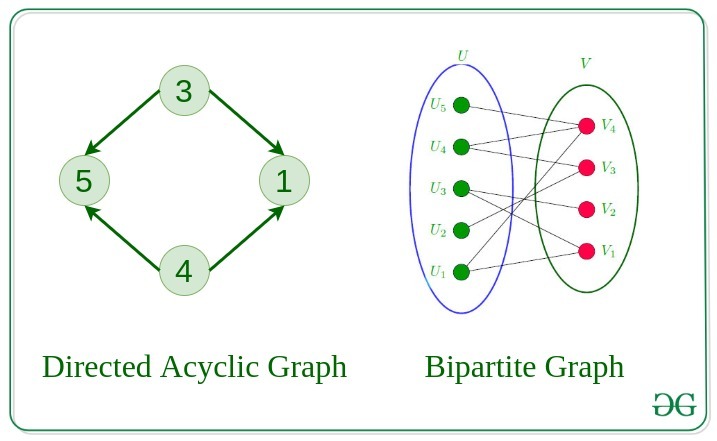
[](https://media.geeksforgeeks.org/wp-content/uploads/20200630122225/cyclic.jpg)

#### 11. Directed Acyclic Graph

A Directed Graph that does not contain any cycle.

#### 12. Bipartite Graph

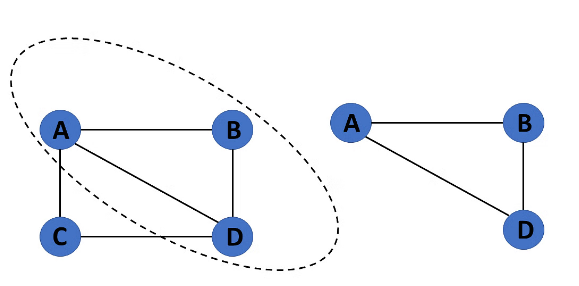
A graph in which vertex can be divided into two sets such that vertex in each set does not contain any edge between them.

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

***13. Weighted Graph***

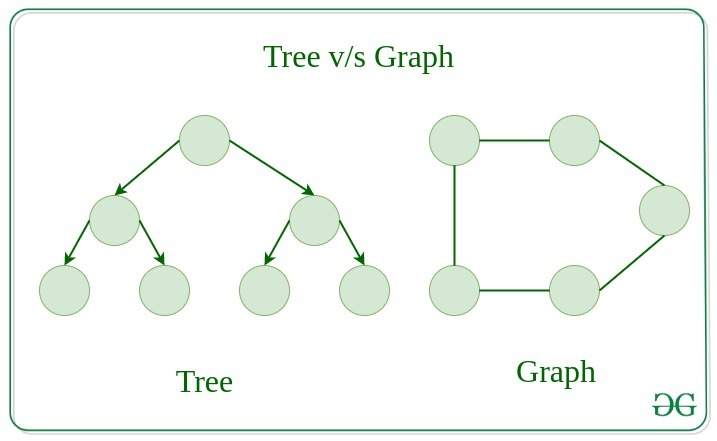
* A graph in which the edges are already specified with suitable weight is known as a weighted graph.
* Weighted graphs can be further classified as directed weighted graphs and undirected weighted graphs.

### ****14. Subgraph:****

A graph G1 = (V1, E1) is called a subgraph of a graph G(V, E) if V1(G) is a subset of V(G) and E1(G) is a subset of E(G) such that each edge of G1 has same end vertices as in G.

## Tree v/s Graph

Trees are the restricted types of graphs, just with some more rules. Every tree will always be a graph but not all graphs will be trees. [Linked List](https://www.geeksforgeeks.org/data-structures/linked-list/), [Trees](https://www.geeksforgeeks.org/binary-tree-data-structure/), and [Heaps](https://www.geeksforgeeks.org/heap-data-structure/) all are special cases of graphs.

[](https://media.geeksforgeeks.org/wp-content/uploads/20200630123458/tree_vs_graph.jpg)

| **The basis of Comparison** | **Graph** | **Tree** |
| --- | --- | --- |
| **Definition** | **Graph is a non-linear data structure.** | **Tree is a non-linear data structure.** |
| **Structure** | **It is a collection of vertices/nodes and edges.** | **It is a collection of nodes and edges.** |
| **Structure cycle** | **A graph can be connected or disconnected, can have cycles or loops, and does not necessarily have a root node.** | **A tree is a type of graph that is connected, acyclic (meaning it has no cycles or loops), and has a single root node.** |
| **Edges** | **Each node can have any number of edges.** | **If there is n nodes then there would be n-1 number of edges** |
| **Types of Edges** | **They can be directed or undirected** | **They are always directed** |
| **Root node** | **There is no unique node called root in graph.** | **There is a unique node called root(parent) node in trees.** |
| **Loop Formation** | **A cycle can be formed.** | **There will not be any cycle.** |
| **Traversal** | **For graph traversal, we use**[Breadth-First Search (BFS)](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/)**, and**[Depth-First Search (DFS)](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/)**.** | **We traverse a tree using**[in-order, pre-order, or post-order](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)**traversal methods.** |
| **Applications** | **For finding shortest path in networking graph is used.** | **For game trees, decision trees, the tree is used.** |
| **Node relationships** | **In a graph, nodes can have any number of connections to other nodes, and there is no strict parent-child relationship.** | **In a tree, each node (except the root node) has a parent node and zero or more child nodes.** |
| **Commonly used for** | **Graphs are commonly used to model complex systems or relationships, such as social networks, transportation networks, and computer networks.** | **Trees are commonly used to represent data that has a hierarchical structure, such as file systems, organization charts, and family trees.** |
| **Connectivity** | **In a graph, nodes can have any number of connections to other nodes.** | **In a tree, each node can have at most one parent, except for the root node, which has no parent.** |

## Representation of Graphs

There are two ways to store a graph:

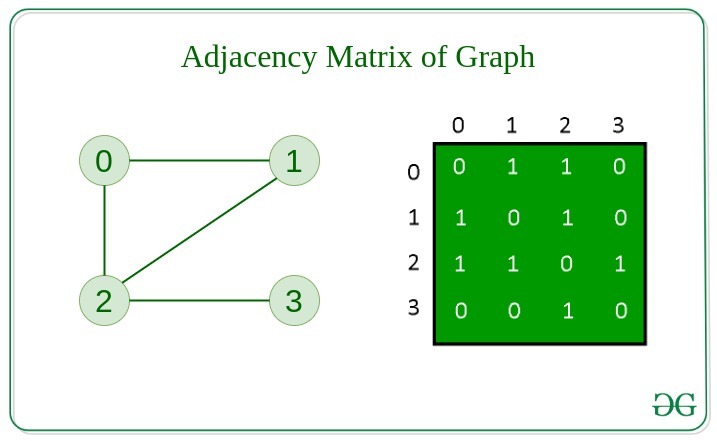
* Adjacency Matrix
* Adjacency List

#### Adjacency Matrix

An adjacency matrix is a way of representing a graph as a matrix of boolean (0’s and 1’s).

Let’s assume there are **n** vertices in the graph So, create a 2D matrix **adjMat[n][n]** having dimension n x n.

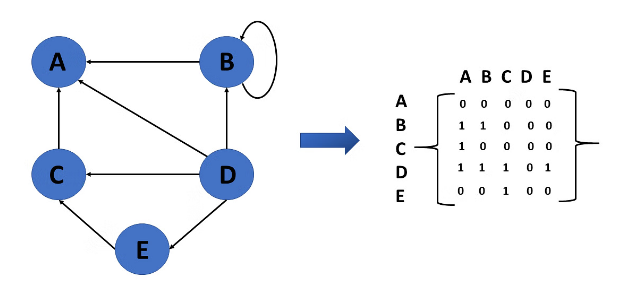
* *If there is an edge from vertex****i****to****j****, mark****adjMat[i][j]****as****1****.*
* *If there is no edge from vertex****i****to****j****, mark****adjMat[i][j]****as****0****.*

[](https://media.geeksforgeeks.org/wp-content/uploads/20200630124726/adjacency_mat1.jpg)

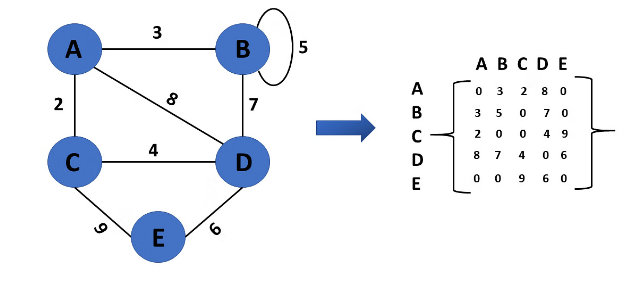
#### Undirected ghragh

#### 

#### Directed ghragh



#### Weighted Undirected graph



#### Adjacency List

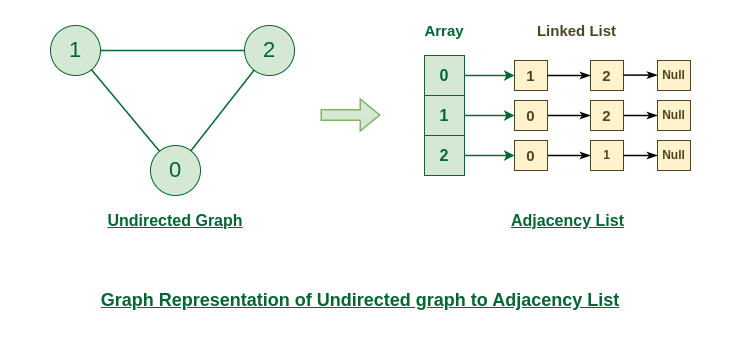
An array of Lists is used to store edges between two vertices. The size of array is equal to the number of **vertices (i.e, n)**. Each index in this array represents a specific vertex in the graph. The entry at the index i of the array contains a linked list containing the vertices that are adjacent to vertex **i**.

Let’s assume there are **n** vertices in the graph So, create an **array of list** of size **n** as **adjList[n].**

* *adjList[0] will have all the nodes which are connected (neighbour) to vertex****0****.*
* *adjList[1] will have all the nodes which are connected (neighbour) to vertex****1****and so on.*

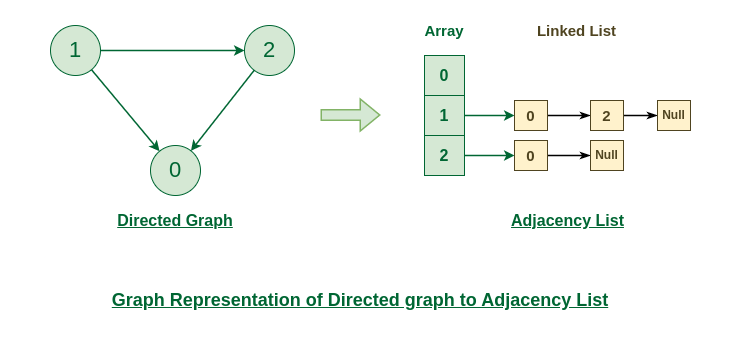
### Representation of Undirected Graph to Adjacency list:

The below undirected graph has 3 vertices. So, an array of list will be created of size 3, where each indices represent the vertices. Now, vertex 0 has two neighbours (i.e, 1 and 2). So, insert vertex 1 and 2 at indices 0 of array. Similarly, For vertex 1, it has two neighbour (i.e, 2 and 0) So, insert vertices 2 and 0 at indices 1 of array. Similarly, for vertex 2, insert its neighbours in array of list.



### Representation of Directed Graph to Adjacency list:

The below directed graph has 3 vertices. So, an array of list will be created of size 3, where each indices represent the vertices. Now, vertex 0 has no neighbours. For vertex 1, it has two neighbour (i.e, 0 and 2) So, insert vertices 0 and 2 at indices 1 of array. Similarly, for vertex 2, insert its neighbours in array of list.



#### Comparison between Adjacency Matrix and Adjacency List

When the graph contains a large number of edges then it is good to store it as a matrix because only some entries in the matrix will be empty. An algorithm such as [Prim’s](https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/) and [Dijkstra](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/) adjacency matrix is used to have less complexity.

| **Action** | **Adjacency Matrix** | **Adjacency List** |
| --- | --- | --- |
| Adding Edge | O(1) | O(1) |
| Removing an edge | O(1) | O(N) |
| Initializing | O(N\*N) | O(N) |

## Basic Operations on Graphs

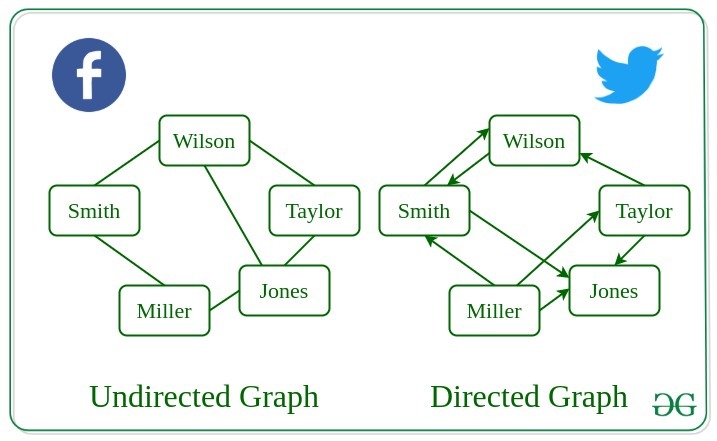
Below are the basic operations on the graph:

* 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.

## Usage of graphs

* Maps can be represented using graphs and then can be used by computers to provide various services like the shortest path between two cities.
* When various tasks depend on each other then this situation can be represented using a Directed Acyclic graph and we can find the order in which tasks can be performed using
* topological sort.
* State Transition Diagram represents what can be the legal moves from current states. In-game of tic tac toe this can be used.

## Real-Life Applications of Graph

[](https://media.geeksforgeeks.org/wp-content/uploads/20200630130949/applications_graph.jpg)

## Advantages and Disadvantages:

### 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.
3. Graph algorithms are often very efficient and can be used to solve complex problems quickly and effectively.
4. Graphs can be used to represent complex data structures in a simple and intuitive way, making them easier to understand and analyze.

### 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.
4. Graphs can be difficult to visualize and analyze, especially for very large or complex graphs, which can make it challenging to extract meaningful insights from the data.

##### **Graph Traversal Algorithms**

The most frequent methods when traversing a graph:

* Breadth first search (BFS)
* Depth first search (DFS)
* **Breadth First Search or BFS for a Graph**

*The****Breadth First Search (BFS)****algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.*

## How does BFS work?

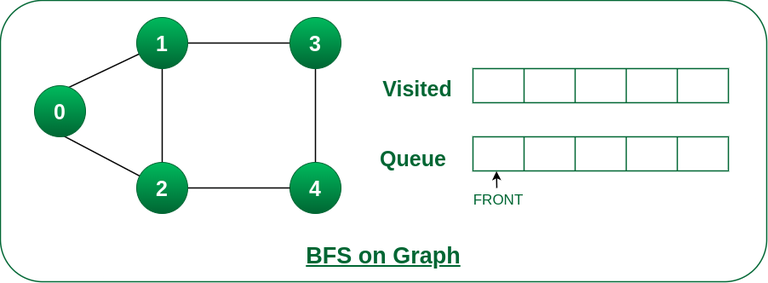
Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

**Illustration:**

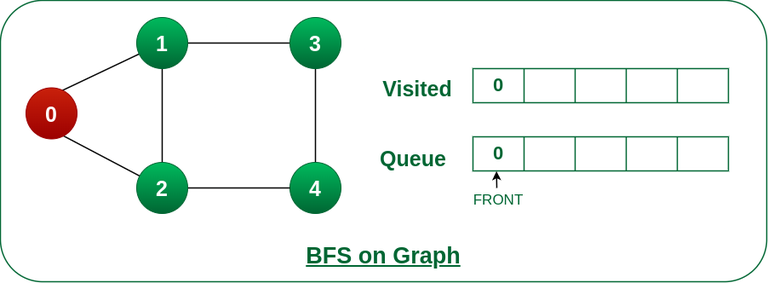
Let us understand the working of the algorithm with the help of the following example.

***Step1:****Initially queue and visited arrays are empty.*

**

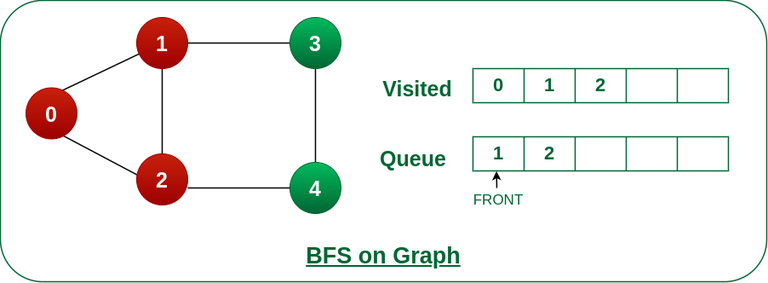
*Queue and visited arrays are empty initially.*

***Step2:****Push node 0 into queue and mark it visited.*

**

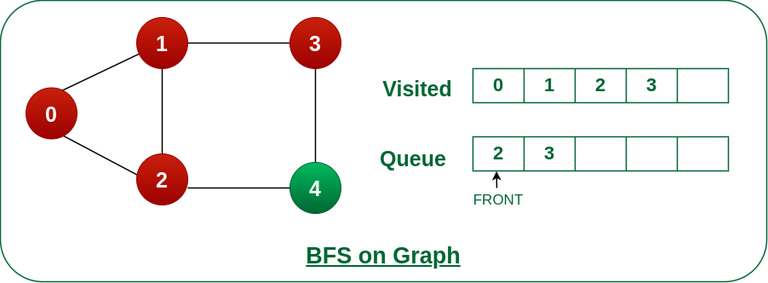
*Push node 0 into queue and mark it visited.*

***Step 3:****Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.*

**

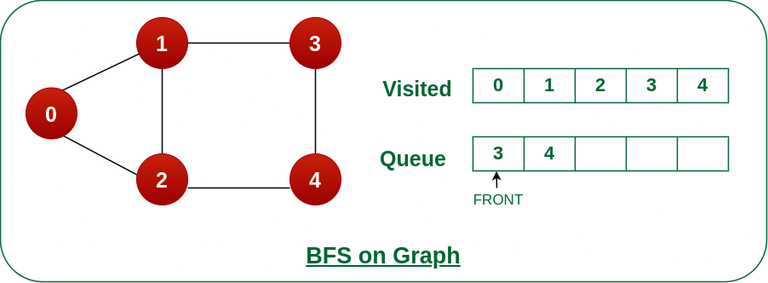
*Remove node 0 from the front of queue and visited the unvisited neighbours and push into queue.*

***Step 4:****Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.*

**

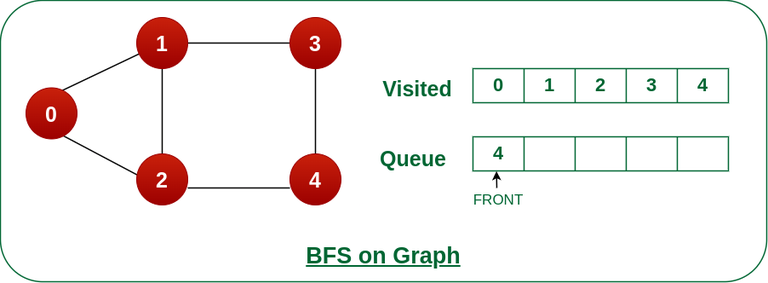
*Remove node 1 from the front of queue and visited the unvisited neighbours and push*

***Step 5:****Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*

**

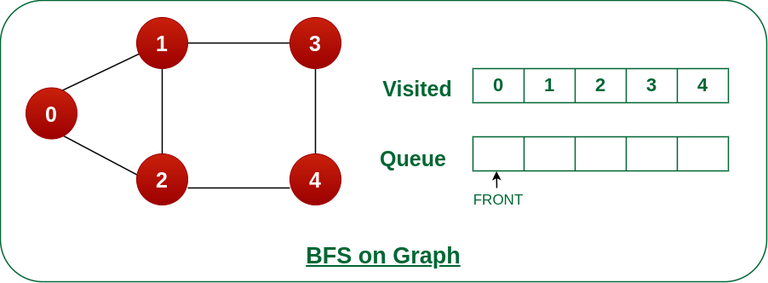
*Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*

***Step 6:****Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.   
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.*

**

*Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.*

***Steps 7:****Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.   
As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.*

**

*Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.*

*Now, Queue becomes empty, So, terminate these process of iteration.*

**Time Complexity:**O(V+E), where V is the number of nodes and E is the number of edges.  
**Auxiliary Space:**O(V)

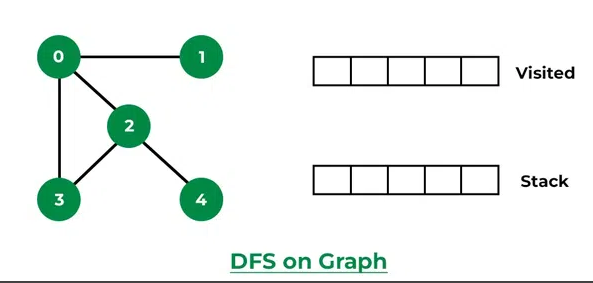
* **Depth First Search or DFS for a Graph**

## How does DFS work?

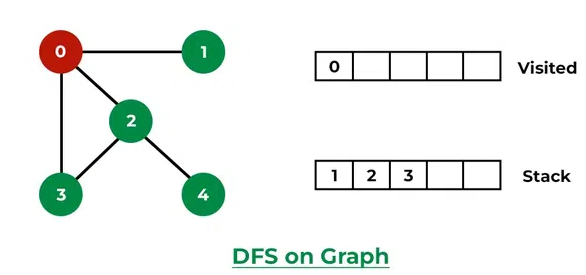
Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Let us understand the working of **Depth First Search** with the help of the following illustration:

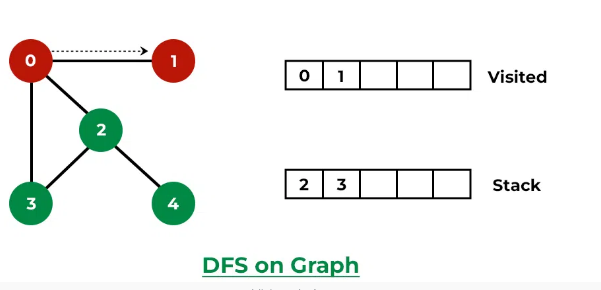
***Step1:****Initially stack and visited arrays are empty.*



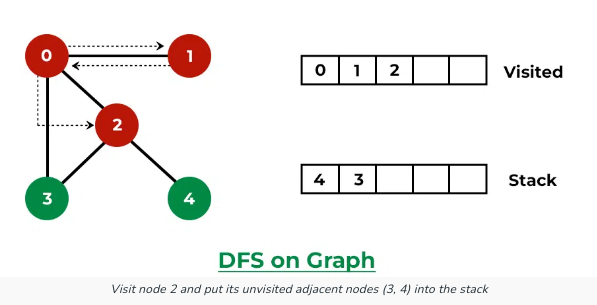
***Step 2:****Visit 0 and put its adjacent nodes which are not visited yet into the stack.*



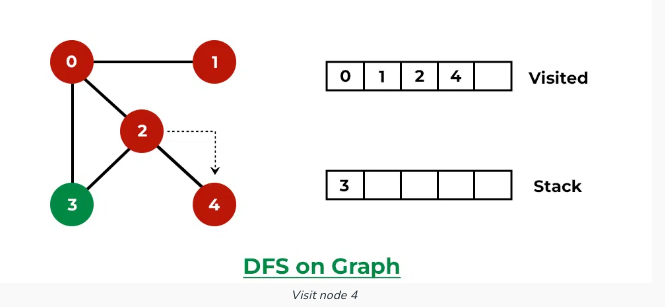
***Step 3:****Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.*

**

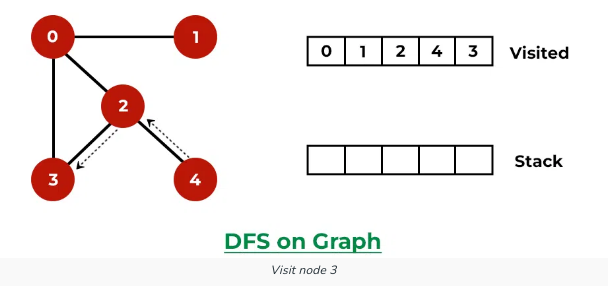
***Step 4:****Now,**Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (i.e, 3, 4) in the stack.*

**

***Step 5:****Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.*

**

***Step 6:****Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.*

**

*Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.*

**Time complexity:**O(V + E), where V is the number of vertices and E is the number of edges in the graph.  
**Auxiliary Space:** O(V + E), since an extra visited array of size V is required, And stack size for iterative call to DFS function.

## ****Difference Between BFS and DFS:****

| **Parameters** | **BFS** | **DFS** |
| --- | --- | --- |
| **Stands for** | BFS stands for Breadth First Search. | DFS stands for Depth First Search. |
| **Data Structure** | BFS(Breadth First Search) uses Queue data structure for finding the shortest path. | DFS(Depth First Search) uses Stack data structure. |
| **Definition** | BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level. | DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes. |
| **Conceptual Difference** | BFS builds the tree level by level. | DFS builds the tree sub-tree by sub-tree. |
| **Approach used** | It works on the concept of FIFO (First In First Out). | It works on the concept of LIFO (Last In First Out). |
| **Suitable for** | BFS is more suitable for searching vertices closer to the given source. | DFS is more suitable when there are solutions away from source. |
| **Applications** | BFS is used in various applications such as bipartite graphs, shortest paths, etc. | DFS is used in various applications such as acyclic graphs and finding strongly connected components etc. |