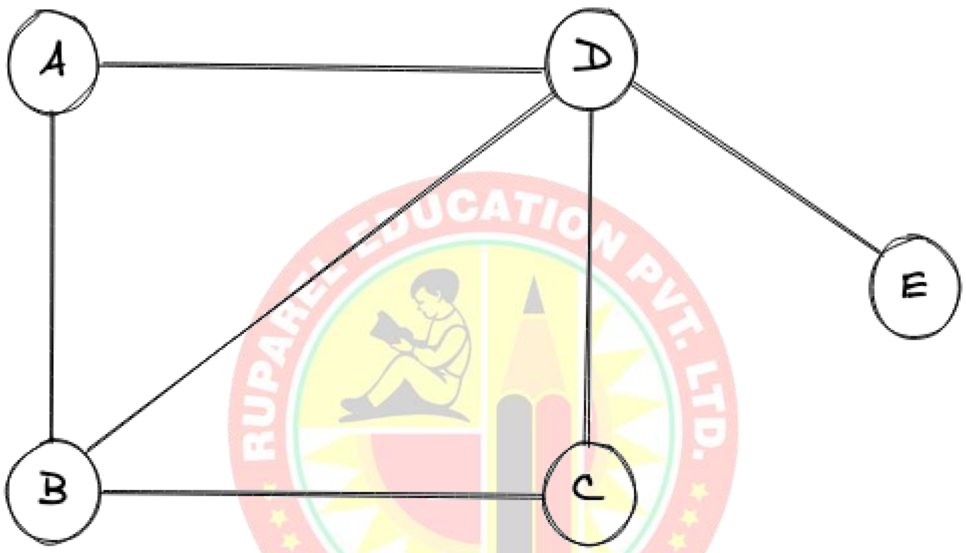
Design and Analysis of Algorithm

UNIT – 6: Exploring Graphs

Introduction to Graphs:

A **graph is an advanced data structure** that is used to organize items in an **interconnected network**. Each item in a graph is known as a **node**(or **vertex**) and these nodes are connected by **edges**.

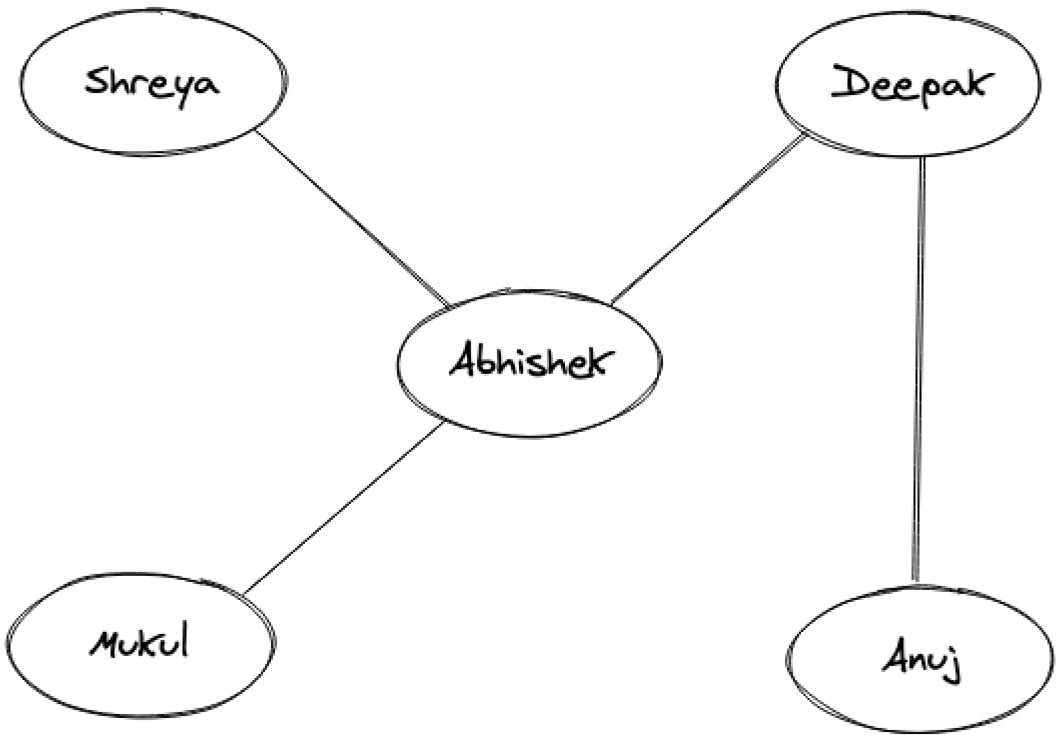
In the figure below, we have a simple graph where there are five nodes in total and six edges.



The nodes in any graph can be referred to as **entities** and the edges that connect different nodes define the **relationships between these entities**. In the above graph we have a set of nodes **{V} = {A, B, C, D, E}** and a set of edges, **{E} = {A-B, A-D, B-C, B-D, C-D, D-E}** respectively.

Real-World Example:

A very good example of graphs is a **network of socially connected people**, connected by a simple connection which is whether they know each other or not. Consider the figure below, where a pictorial representation of a social network is shown, in which there are five people in total.



A line in the above representation between two people mean that they know each other. If there's no line in between the names, then they simply don't know each other. The names here are equivalent to the nodes of a graph and the lines that define the relationship of "knowing each other" is simply the equivalent of an edge of a graph. It should also be noted that the relationship of knowing each other goes both ways like "Abhishek" knows "Mukul" and "Mukul" knows "Abhishek".

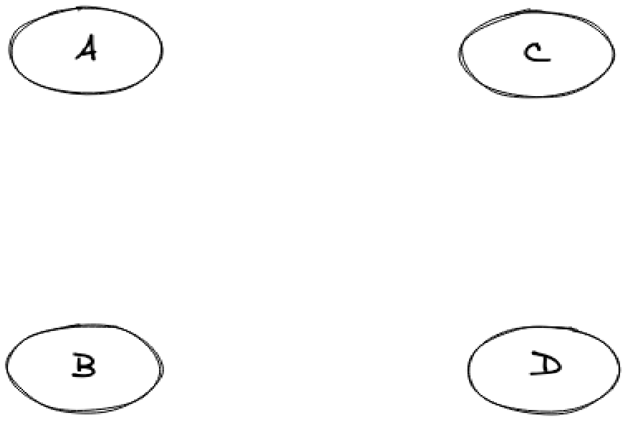
The social network depicted above is nothing but a graph.

Types of Graphs:

Let's cover various different types of graphs.

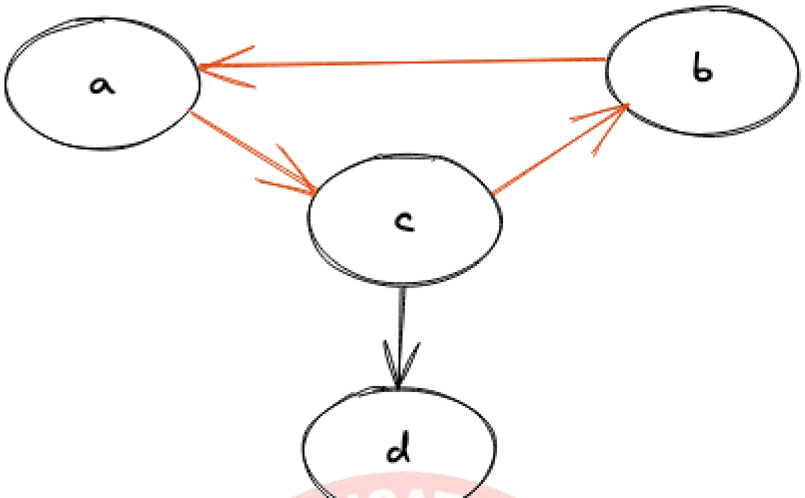
Null Graphs:

A graph is said to be null if there are no edges in that graph.



Cyclic Graph:

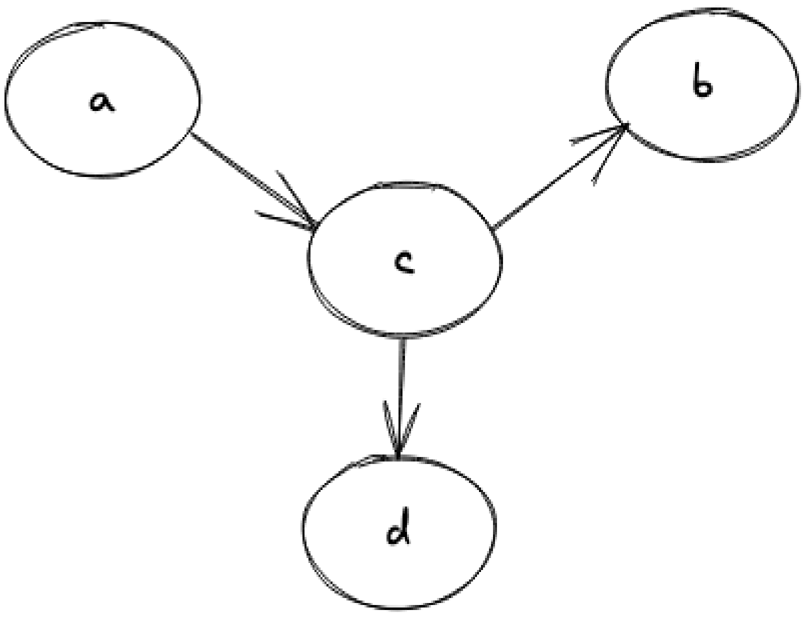
A graph that contains at least one node that traverses back to itself is known as a cyclic graph. In simple words, a graph should have at least one cycle formation for it to be called a cyclic graph.



It can be easily seen that there exists a cycle between the nodes (a, b, c) and hence it is a cyclic graph.

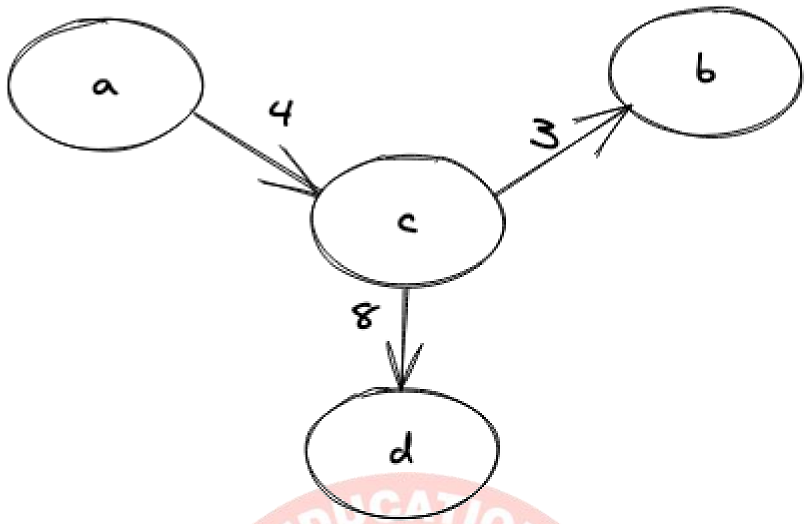
Acyclic Graph:

A graph where there's no way we can start from one node and can traverse back to the same one, or simply doesn't have a single cycle is known as an acyclic graph.



Weighted Graph:

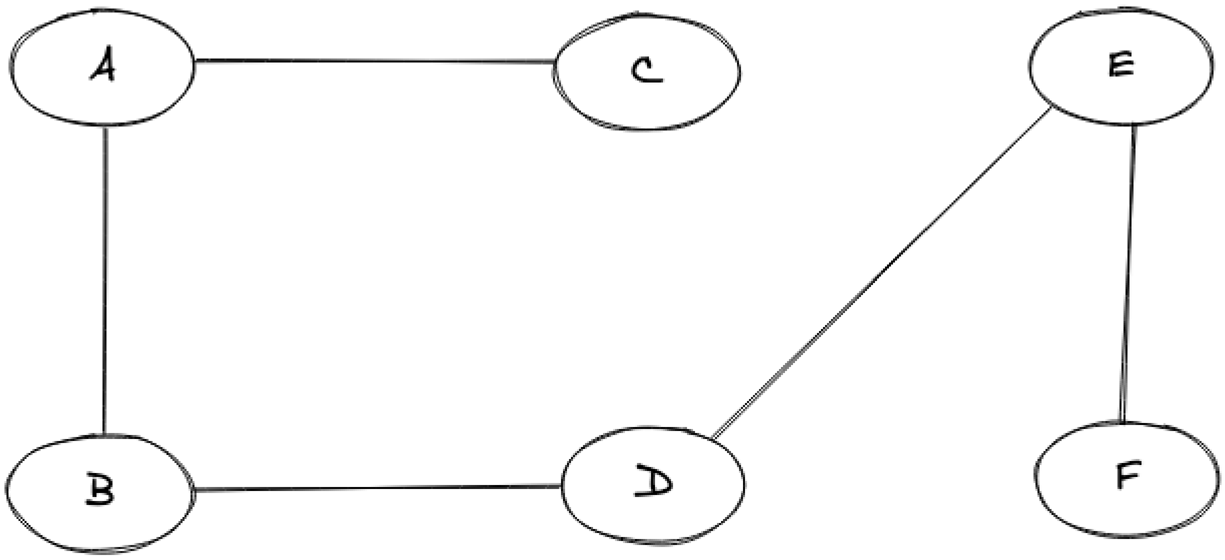
When the edge in a graph has some weight associated with it, we call that graph as a weighted graph. The weight is generally a number that could mean anything, totally dependent on the relationship between the nodes of that graph.



It can also be noted that if any graph doesn't have any weight associated with it, we simply call it an unweighted graph.

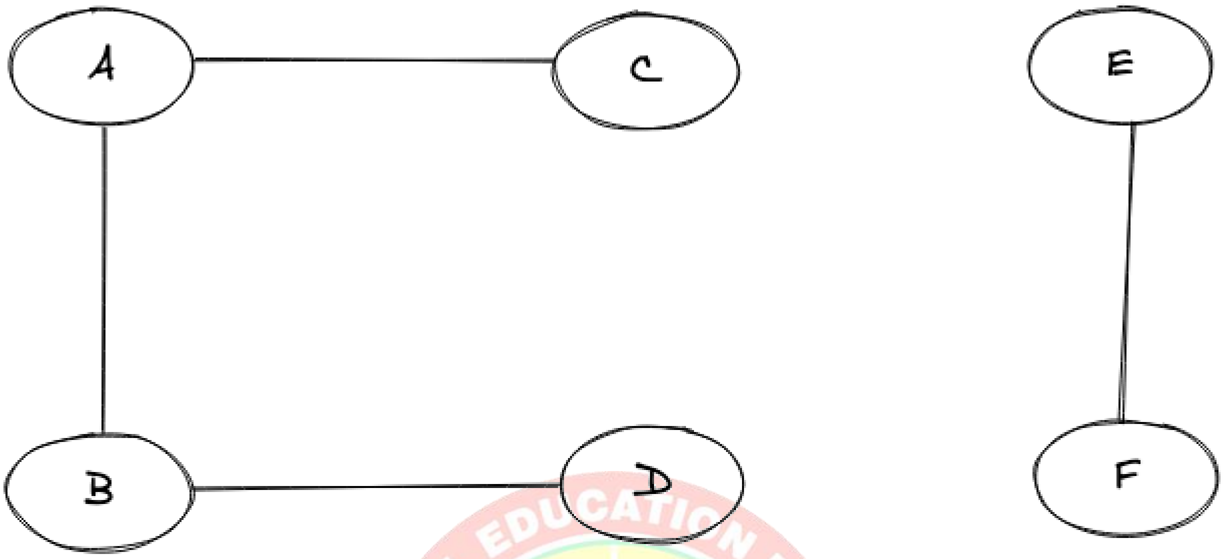
Connected Graph:

A graph where we have a path between every two nodes of the graph is known as a connected graph. A path here means that we are able to traverse from a node "A" to say any node "B". In simple terms, we can say that if we start from one node of the graph we will always be able to traverse to all the other nodes of the graph from that node, hence the connectivity.



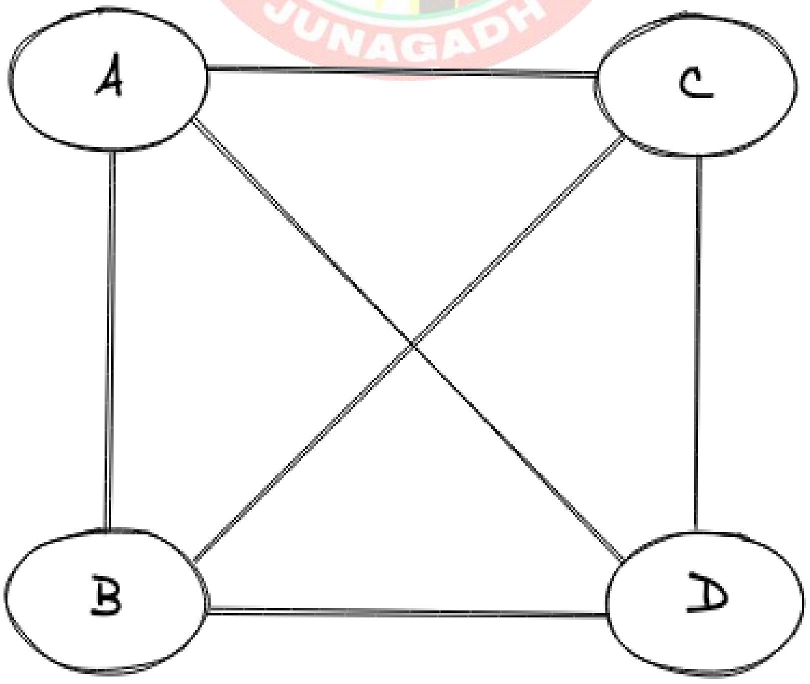
Disconnected Graph:

A graph that is not connected is simply known as a disconnected graph. In a disconnected graph, we will not be able to find a path from between every two nodes of the graph.



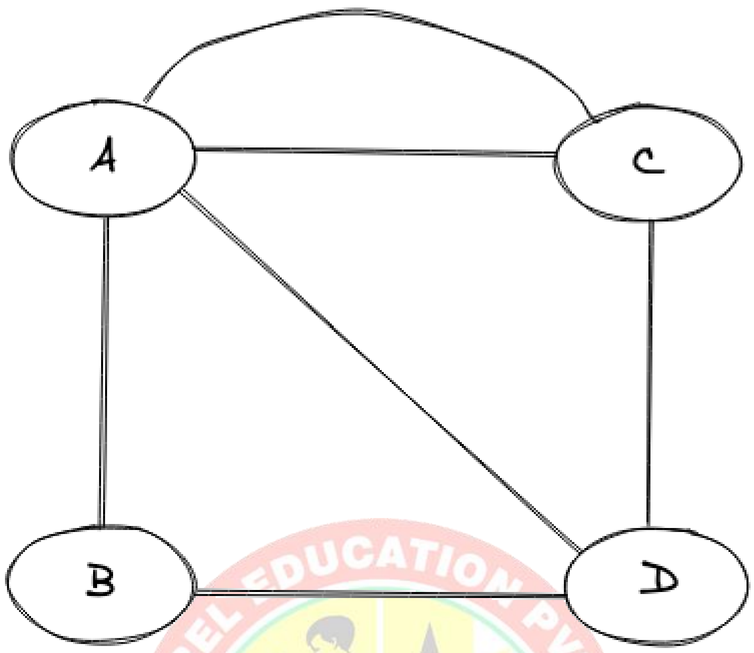
Complete Graph:

A graph is said to be a complete graph if there exists an edge for every pair of vertices(nodes) of that graph.



Multigraph:

A graph is said to be a multigraph if there exist two or more than two edges between any pair of nodes in the graph.



Commonly Used Graph Terminologies:

- **Path** - A sequence of alternating nodes and edges such that each of the successive nodes are connected by the edge.
- **Cycle** - A path where the starting and the ending node is the same.
- **Simple Path** - A path where we do not encounter a vertex again.
- **Bridge** - An edge whose removal will simply make the graph disconnected.
- **Forest** - A forest is a graph without cycles.
- **Tree** - A connected graph that doesn't have any cycles.
- **Degree** - The degree in a graph is the number of edges that are incident on a particular node.
- **Neighbour** - We say vertex "A" and "B" are neighbours if there exists an edge between them.

Breadth First Search:

*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.*

The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:

- **Visited and**
- **0.**

A boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a **queue data structure** for traversal.

How does BFS work?

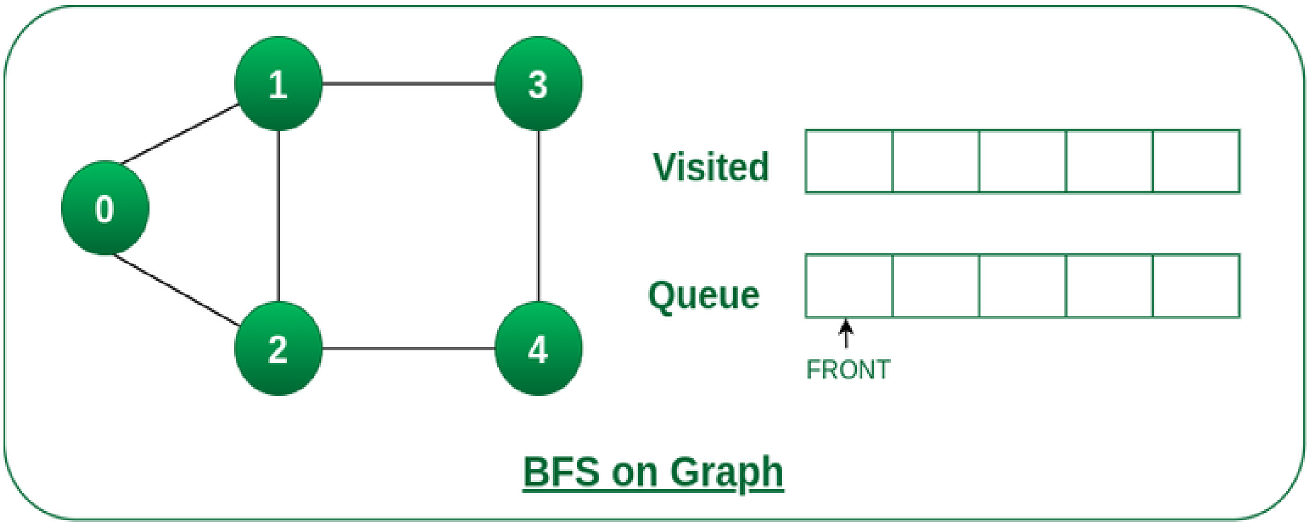
Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

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:

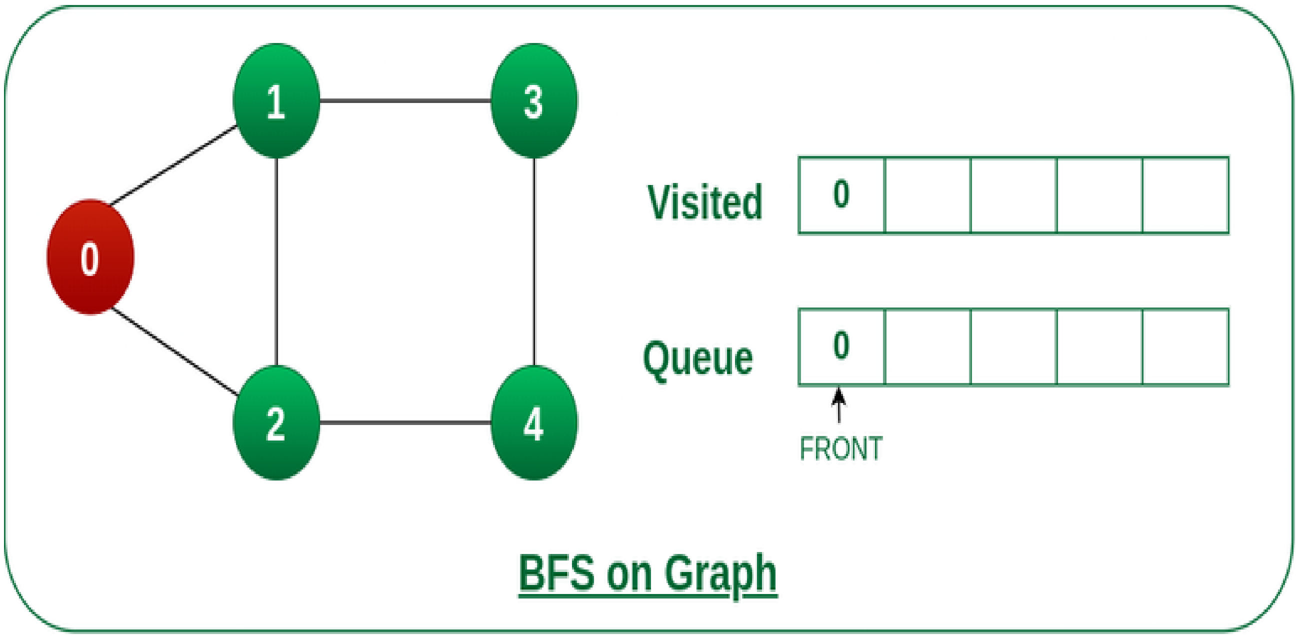
Let us understand the working of the algorithm with the help of the following example.

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



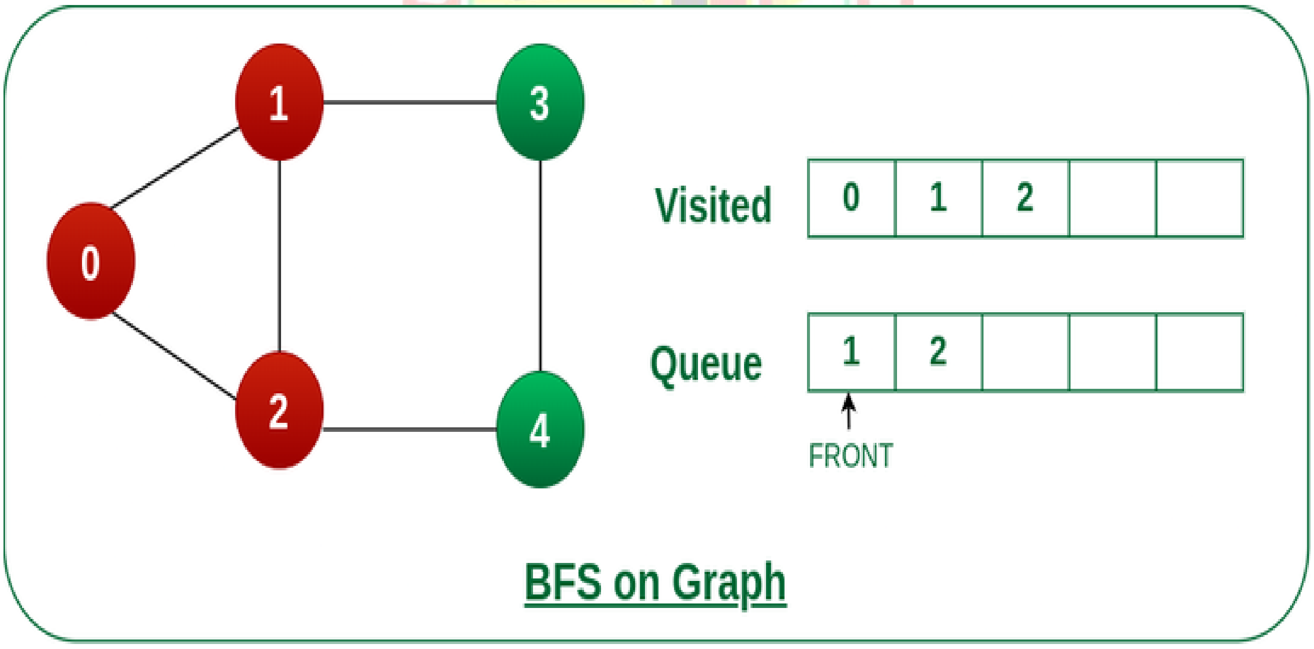
Queue and visited arrays are empty initially.

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



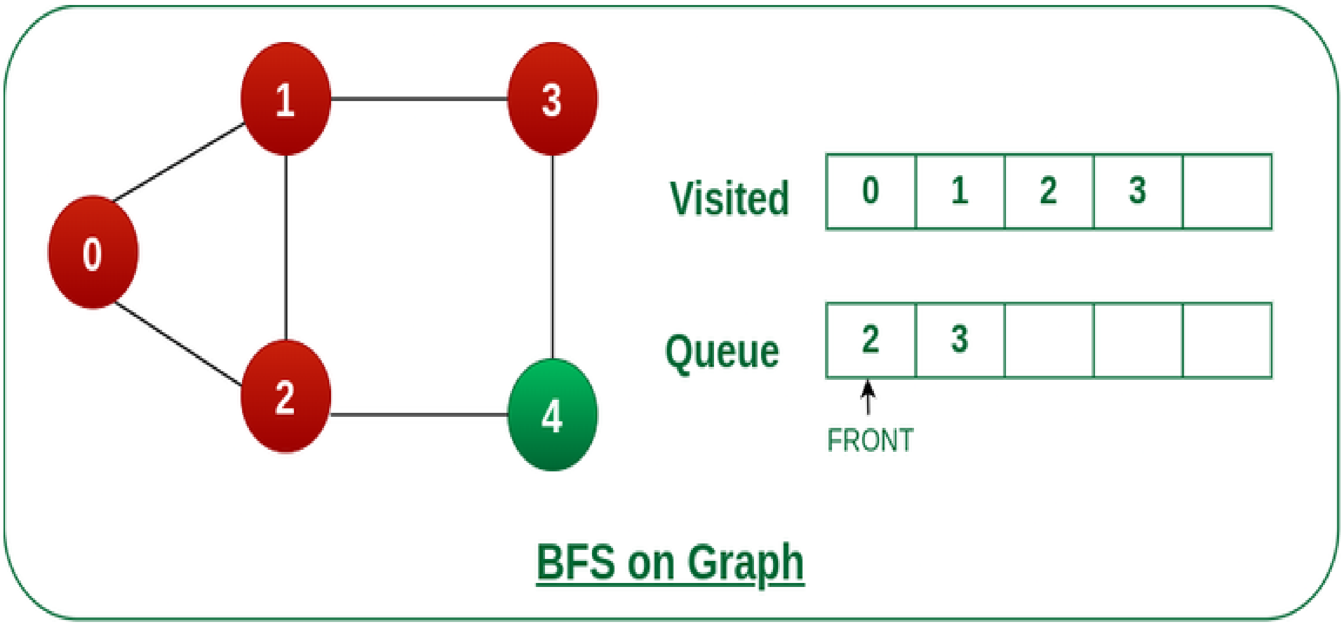
Push node 0 into queue and mark it visited.

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



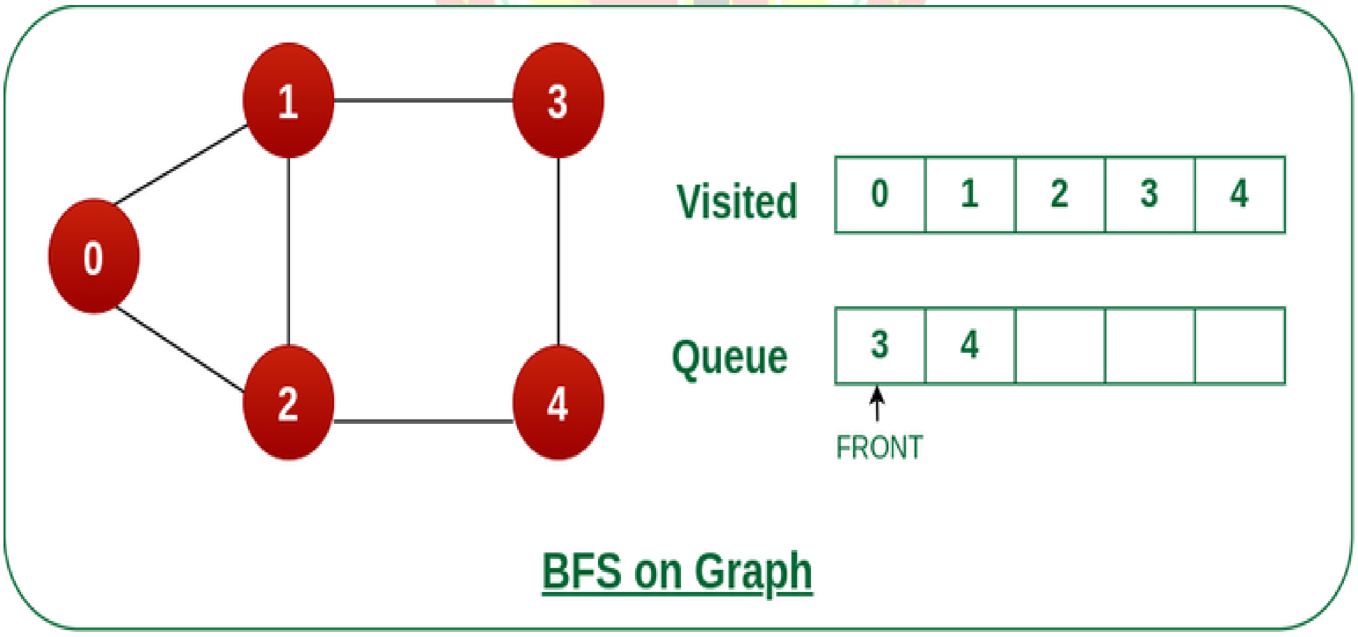
Remove node 0 from the front of queue and visited the unvisited neighbours and push into queue.

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

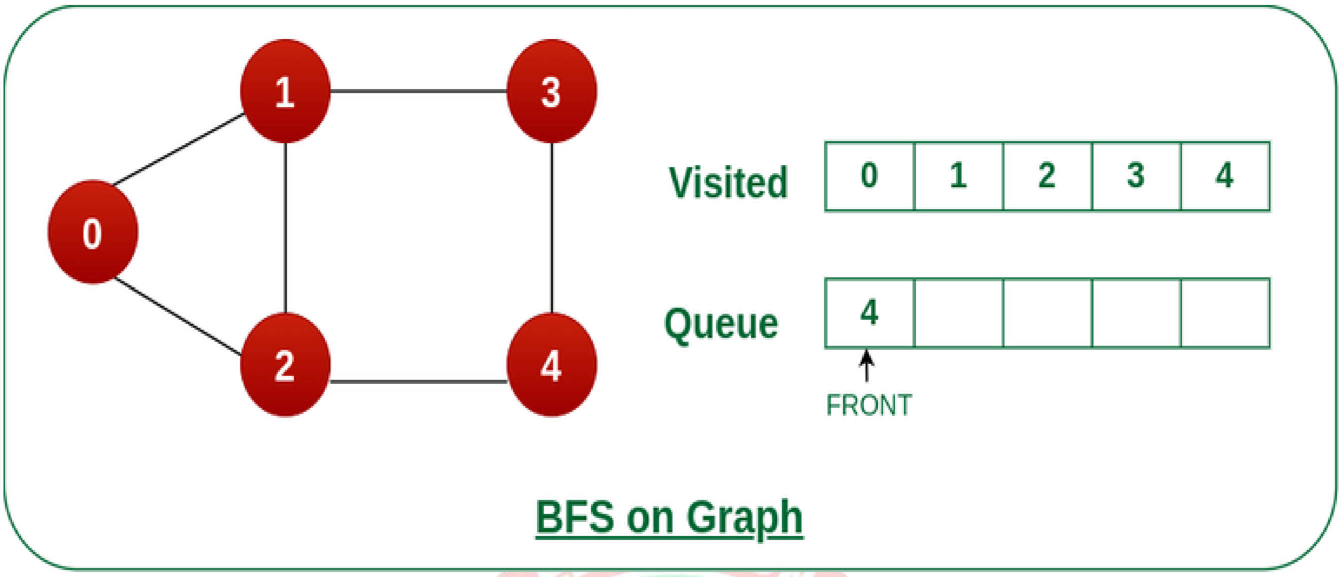
Step 5: *Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*



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.*

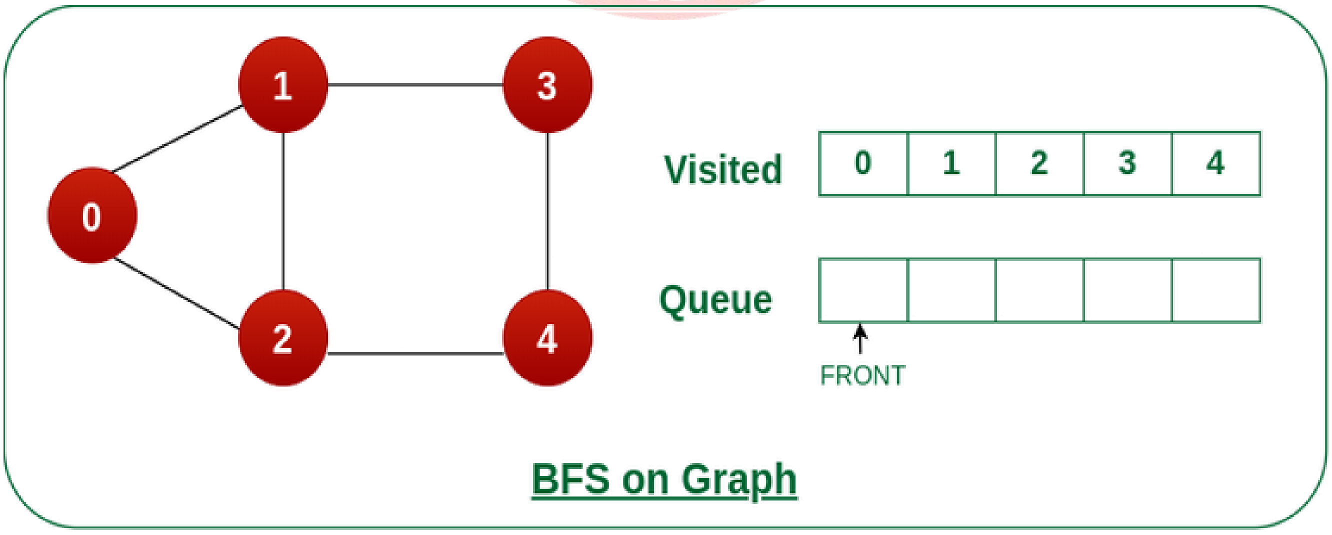
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



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.

Depth First Search:

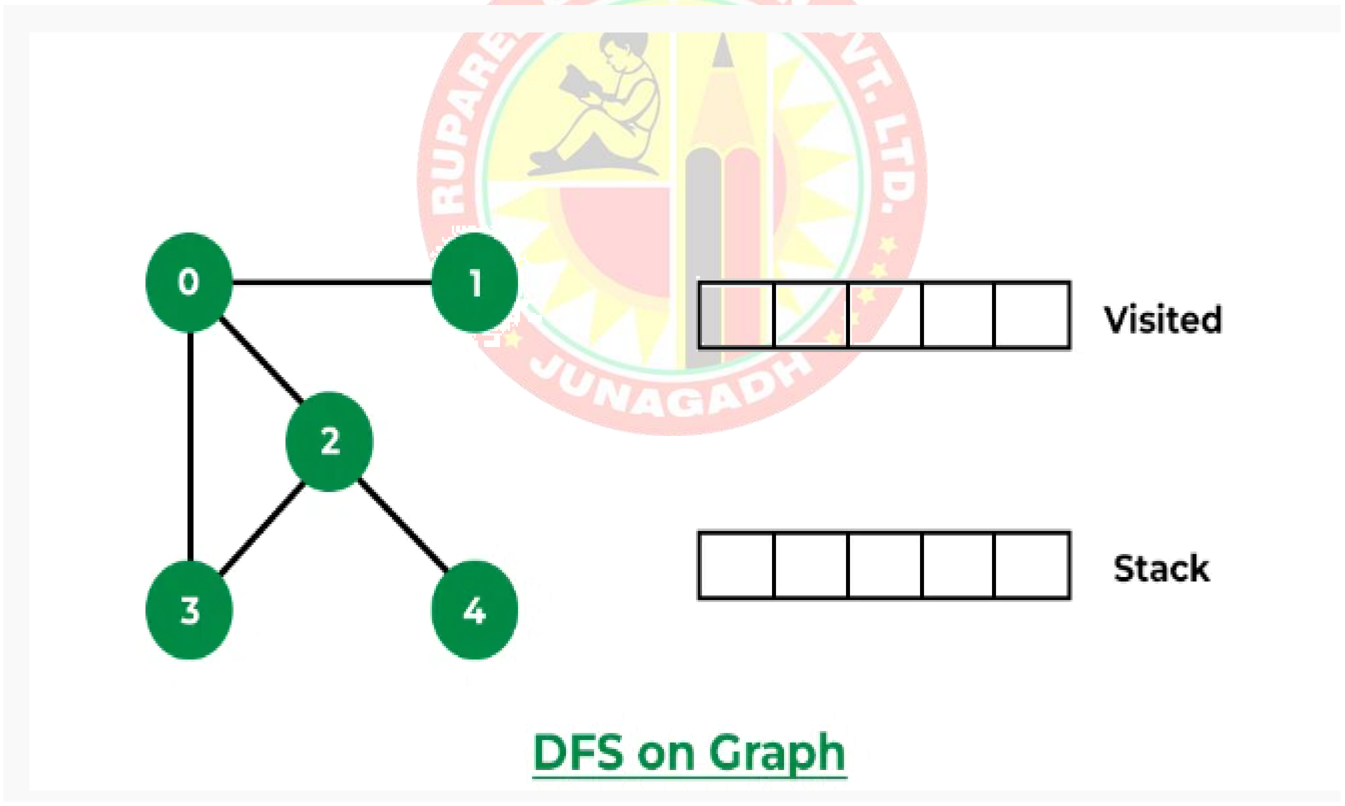
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

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.

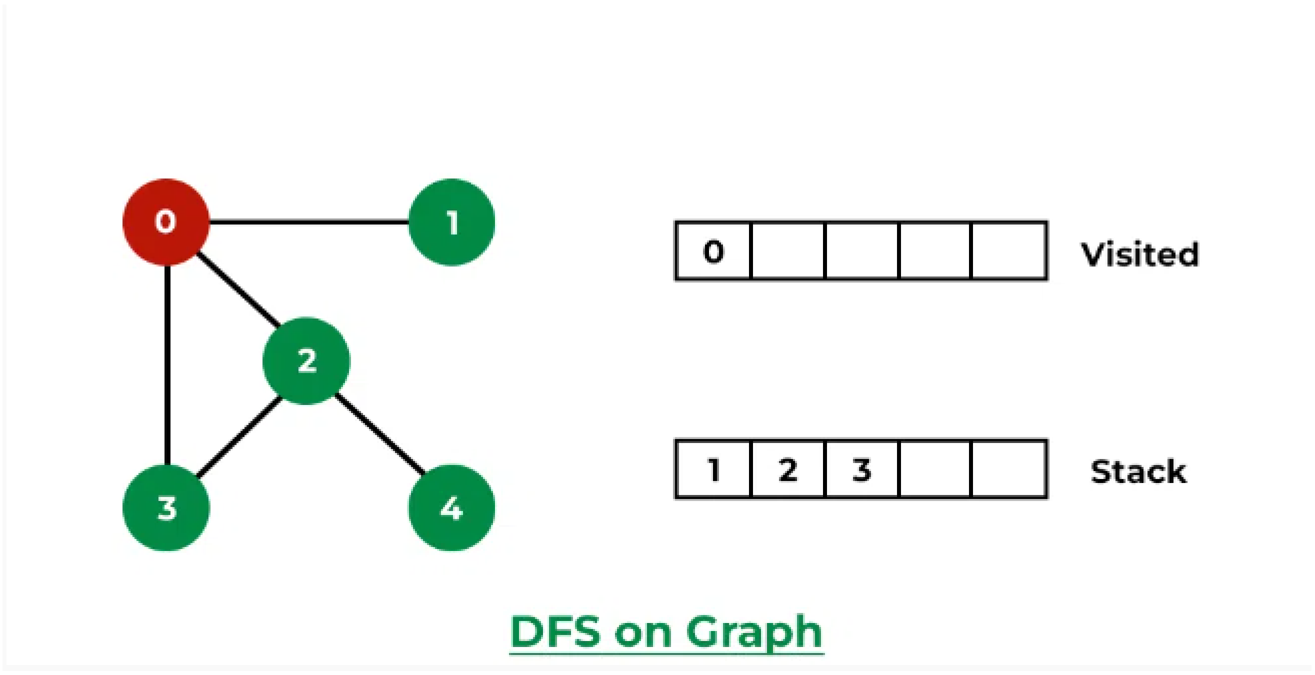
Let us understand the working of **Depth First Search** with the help of the following illustration:

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



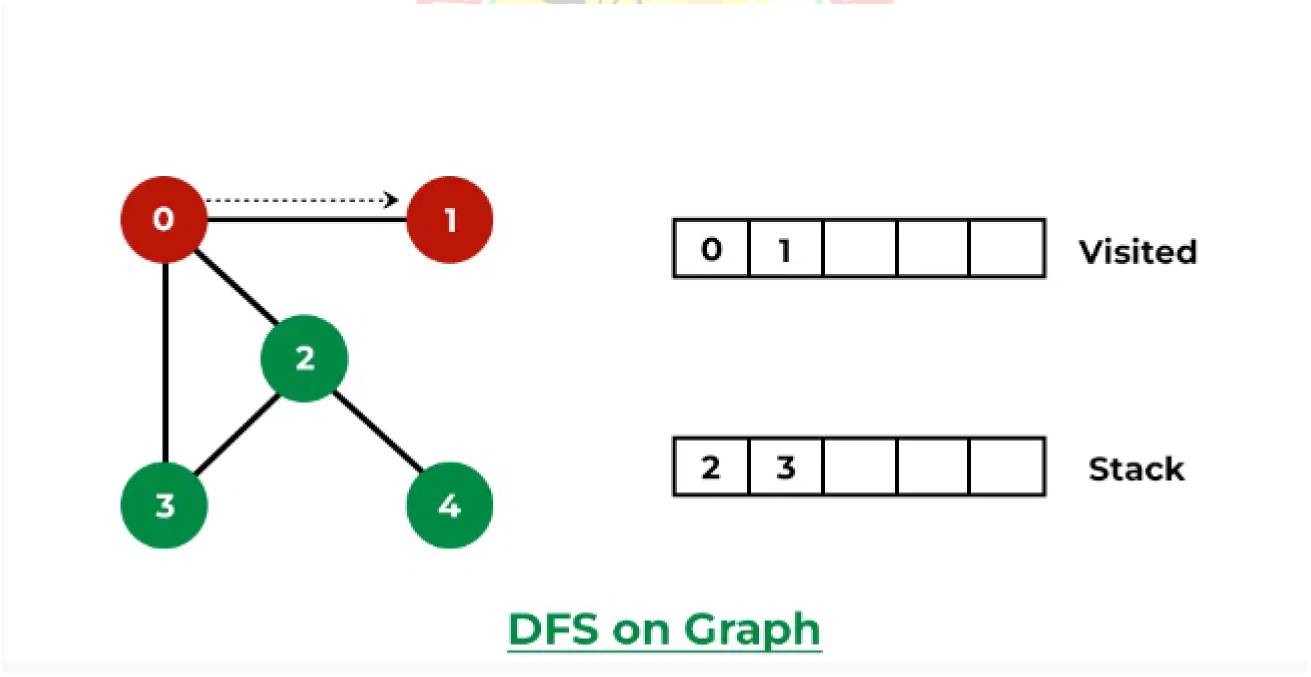
Stack and visited arrays are empty initially.

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



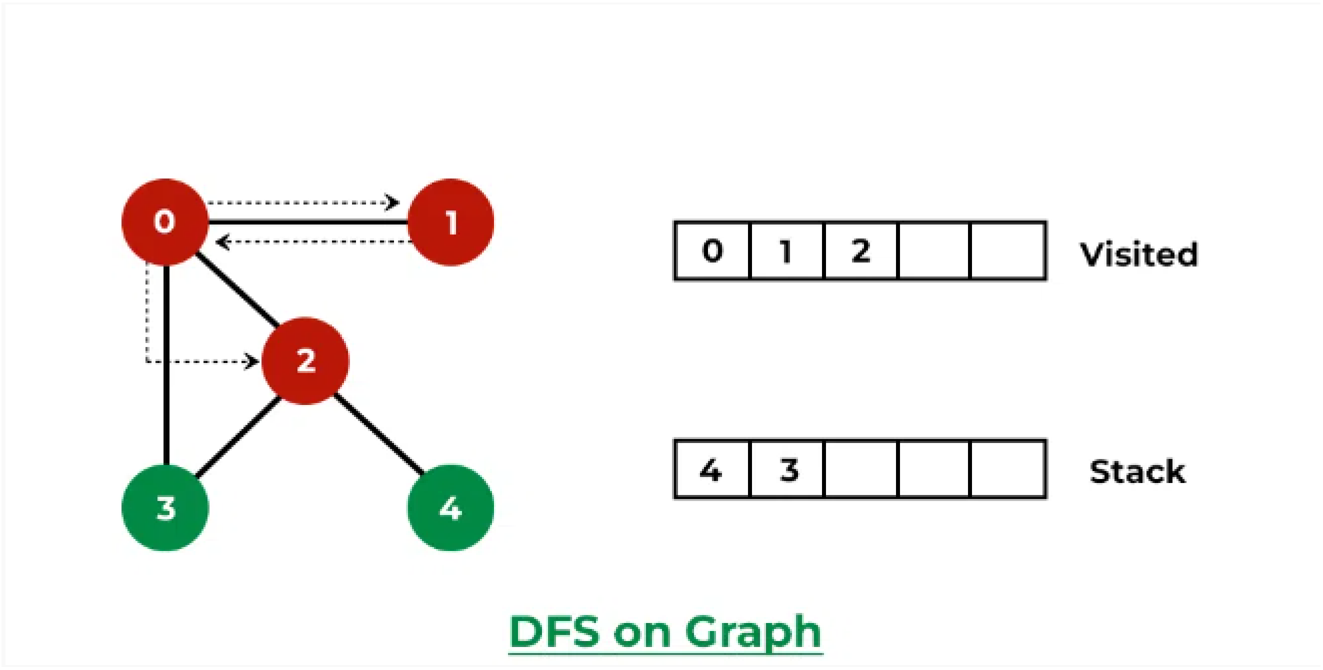
Visit node 0 and put its adjacent nodes (1, 2, 3) into the stack

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.*



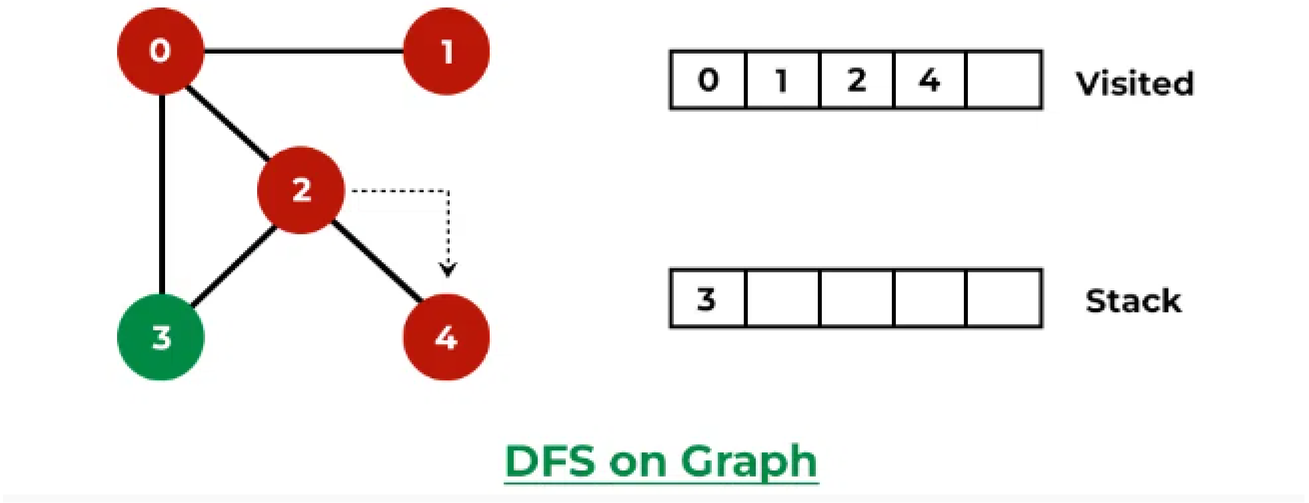
Visit node 1

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.*



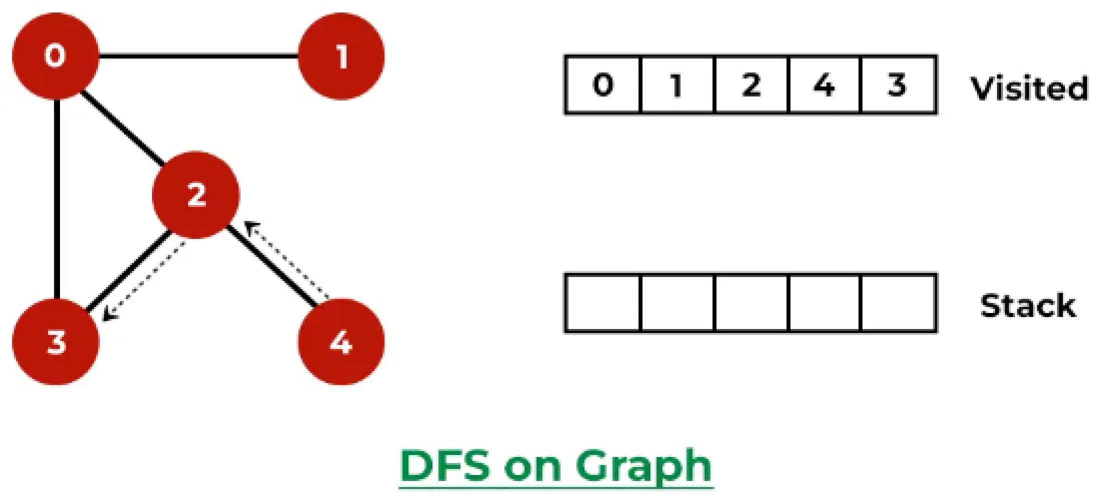
Visit node 2 and put its unvisited adjacent nodes (3, 4) into the stack

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.*



Visit node 4

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.*



Visit node 3

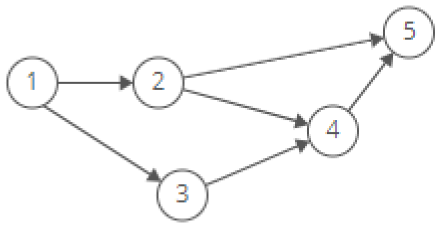
Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

Topological Sort:

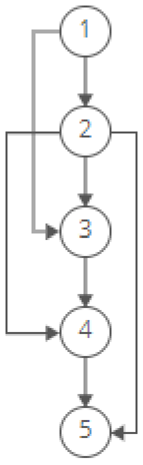
The **topological sort** algorithm takes a directed graph and returns an array of the nodes where each node appears *before* all the nodes it points to.

The ordering of the nodes in the array is called a *topological ordering*.

Here's an example:



Since node 1 points to nodes 2 and 3, node 1 appears before them in the ordering. And, since nodes 2 and 3 both point to node 4, they appear before it in the ordering.



So [1, 2, 3, 4, 5] would be a topological ordering of the graph.