Data Structures and Algorithms

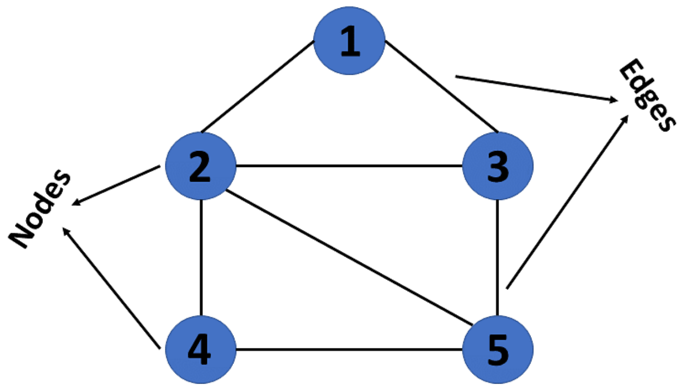
Lecture 35: Graphs

Graphs

A graph can be defined as group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship. A graph can be directed or undirected.

Nodes: Nodes are the fundamental units of the graph. Sometimes, vertices are also known as vertices or nodes.

Edges: Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph.

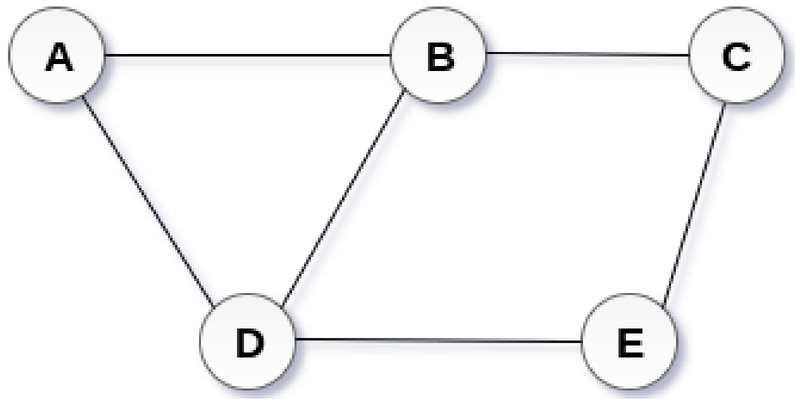


Real-Time Applications of Graph

- Social media analysis
- Network monitoring
- Financial trading
- Internet of Things (IoT) management
- Autonomous vehicles
- Disease surveillance

Undirected Graph

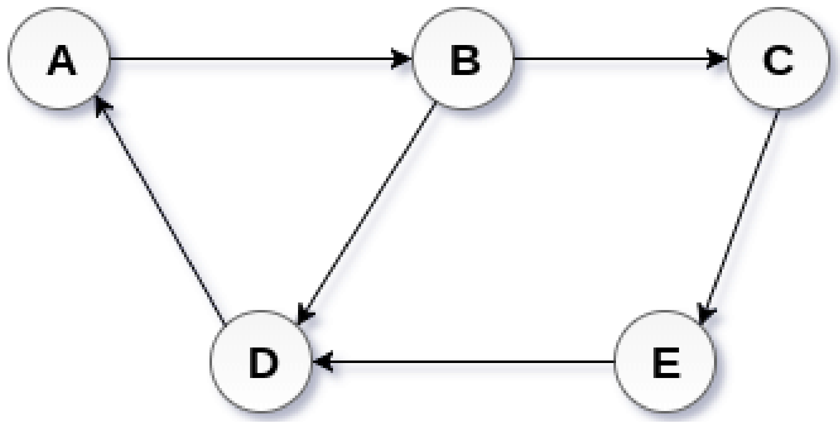
In the undirected graph, edges are not associated with the directions with them. An undirected graph is shown in the figure since its edges are not attached with any of the directions. If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.



Undirected Graph

Directed Graph

In a directed graph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called initial node while node B is called terminal node.



Directed Graph

Graph Terminology

Path - A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

Closed Path - A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if V0=VN.

Simple Path - If all the nodes of the graph are distinct with an exception V0=VN, then such path P is called as closed simple path.

Cycle - A cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices.

Graph Terminology

Connected Graph - A connected graph is the one in which some path exists between every two vertices (u, v) in V. There are no isolated nodes in connected graph.

Complete Graph - A complete graph is the one in which every node is connected with all other nodes. A complete graph contain n(n-1)/2 edges where n is the number of nodes in the graph.

Weighted Graph - In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.

Graph Terminology

Digraph - A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in the specified direction.

Loop - An edge that is associated with the similar end points can be called as Loop.

Adjacent Nodes - If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.

Degree of the Node - A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

Representations of Graphs

The following two are the most commonly used representations of a graph.

Adjacency Matrix

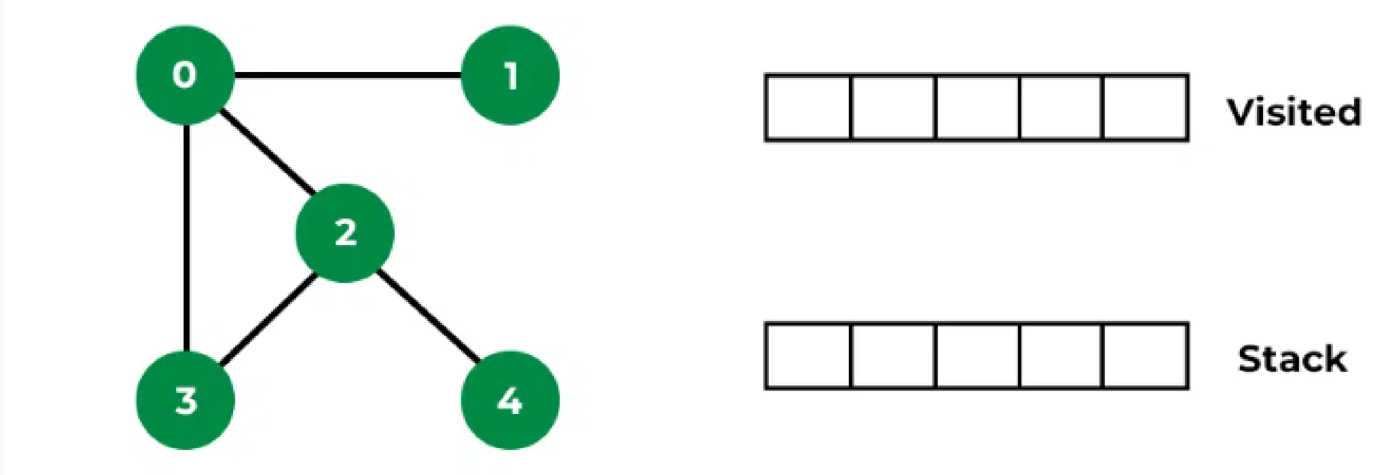
Adjacency List

Graph Traversal - DFS

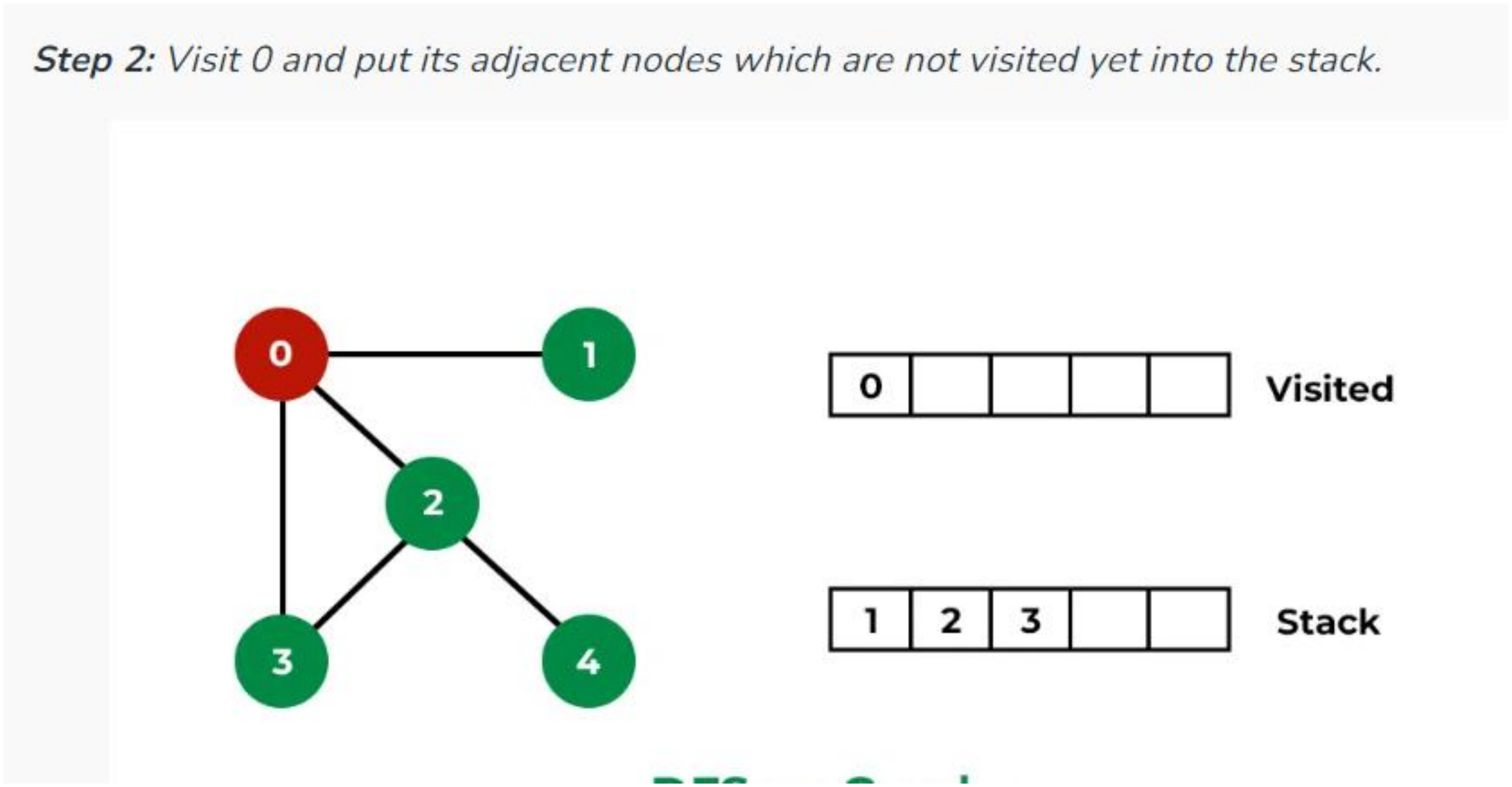
Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

Graph Traversal - DFS

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



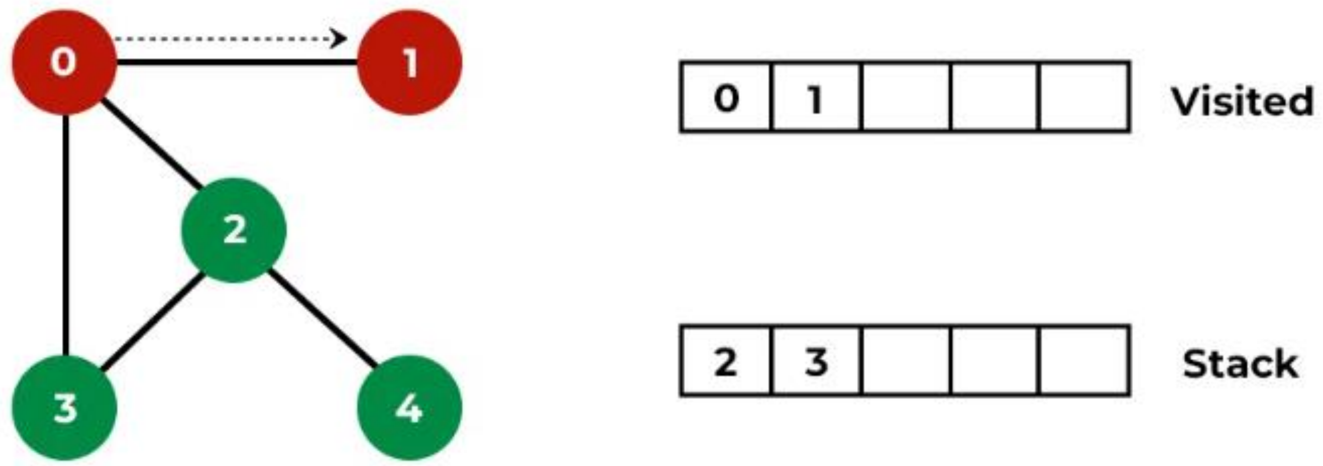
Graph Traversal - DFS



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

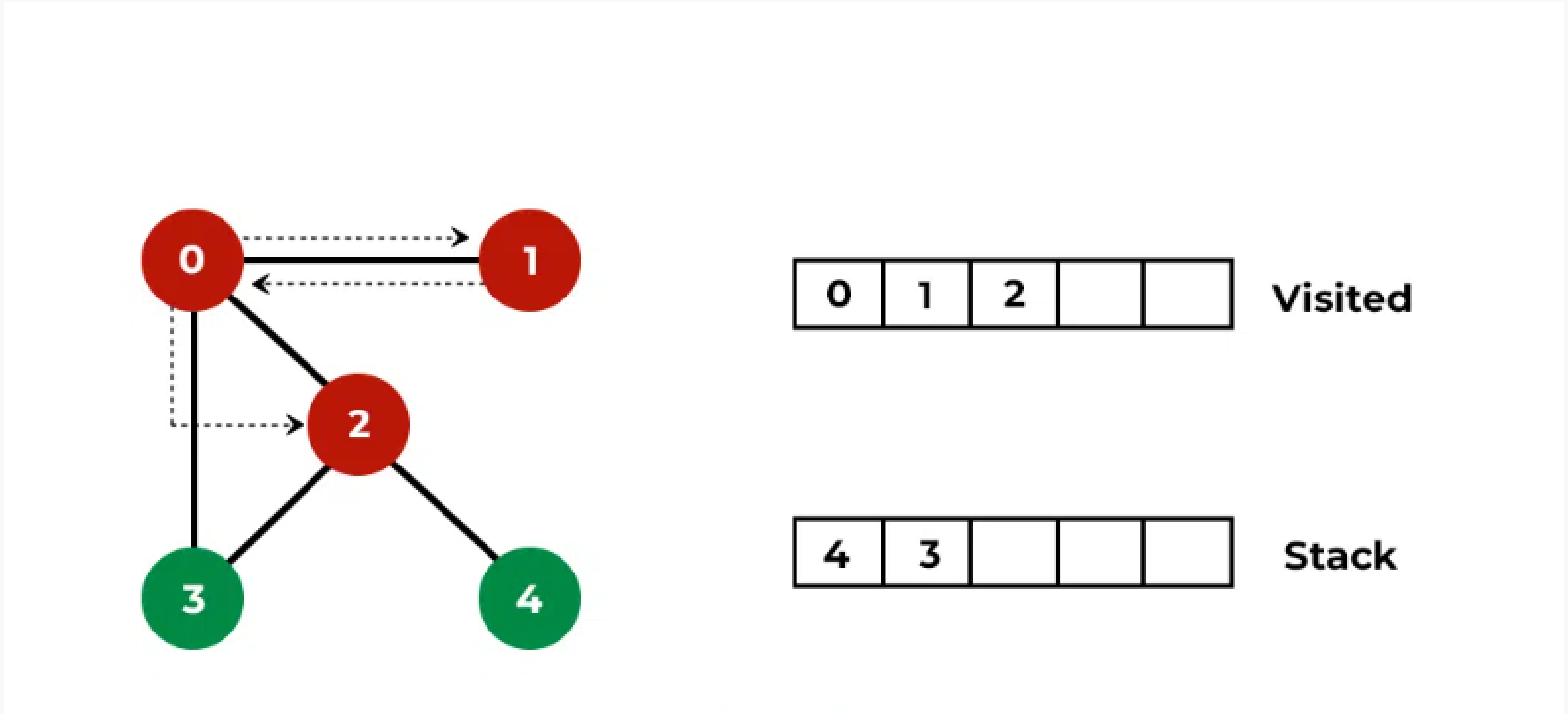
Graph Traversal - DFS

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.*



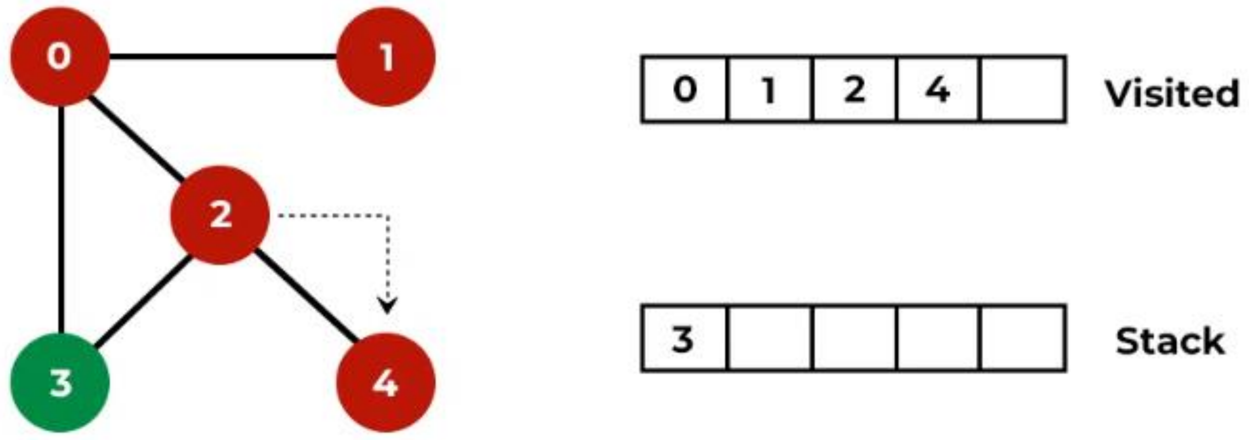
Graph Traversal - DFS

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.*



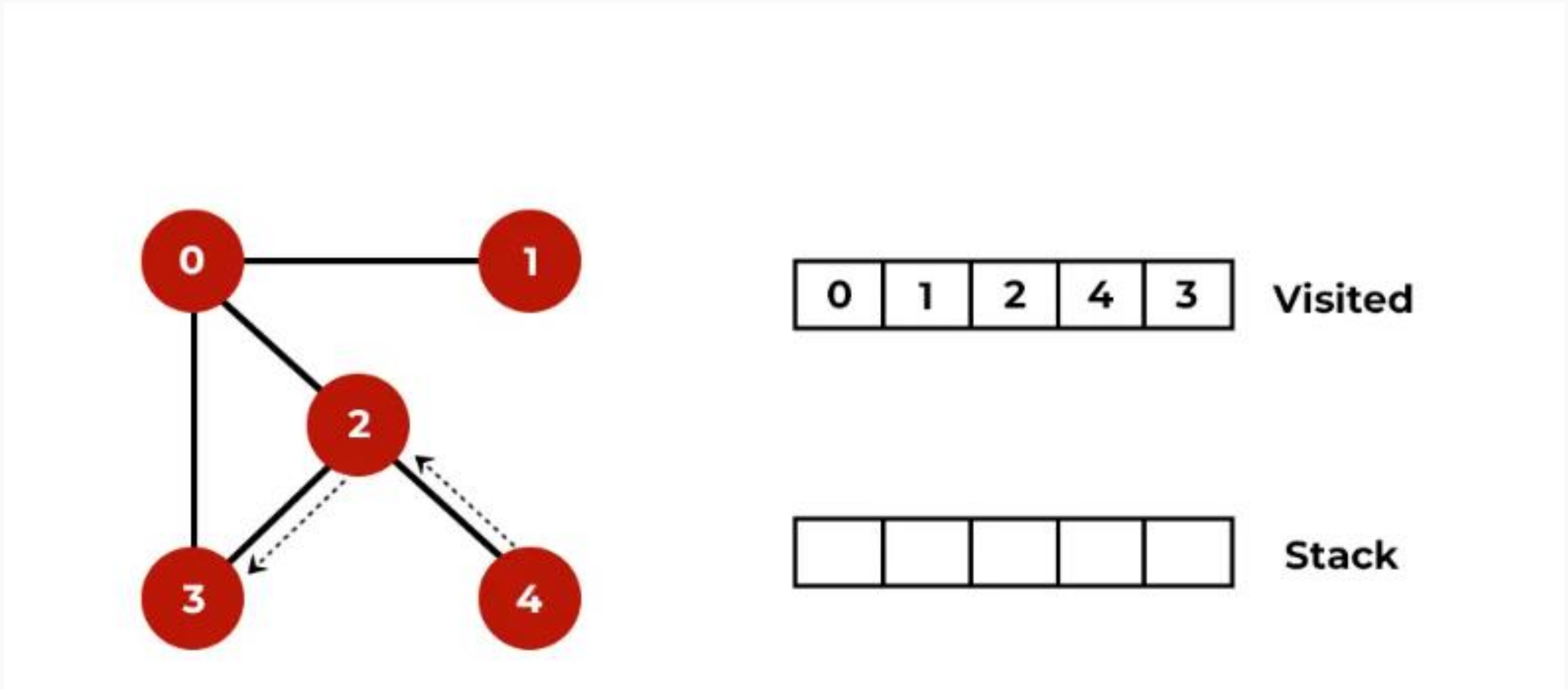
Graph Traversal - DFS

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.*



Graph Traversal - DFS

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.

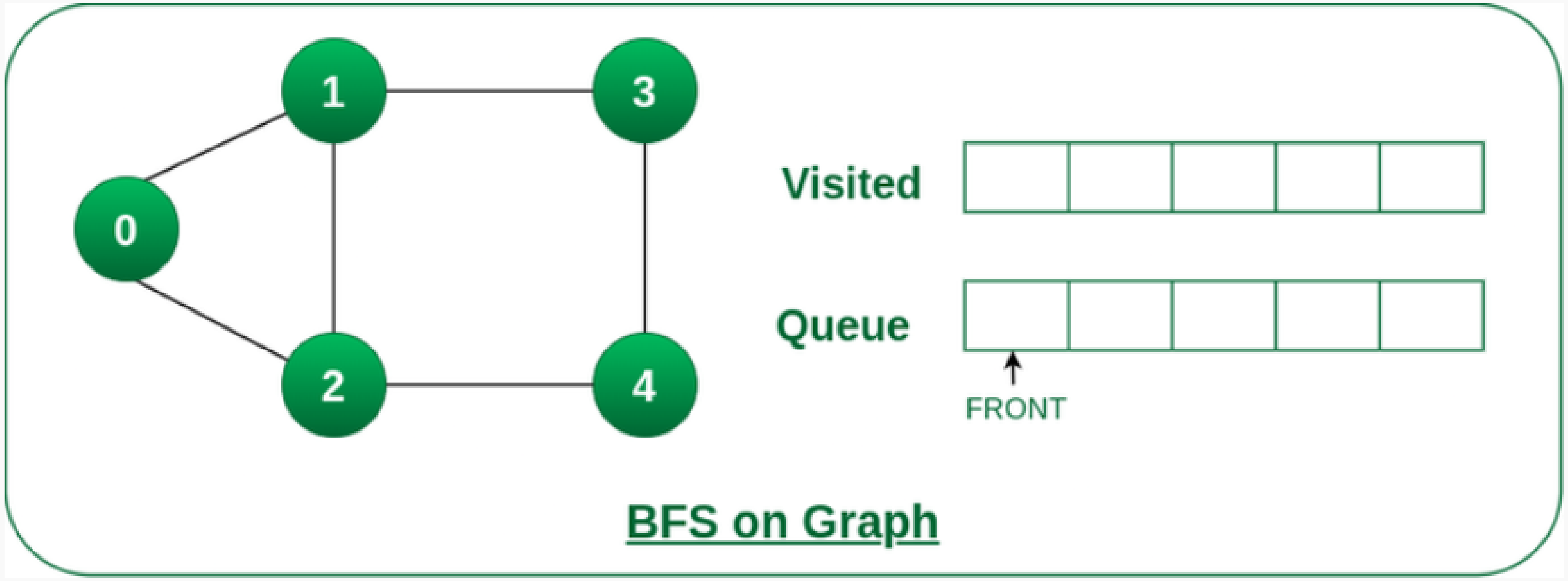
Graph Traversal - BFS

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

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.

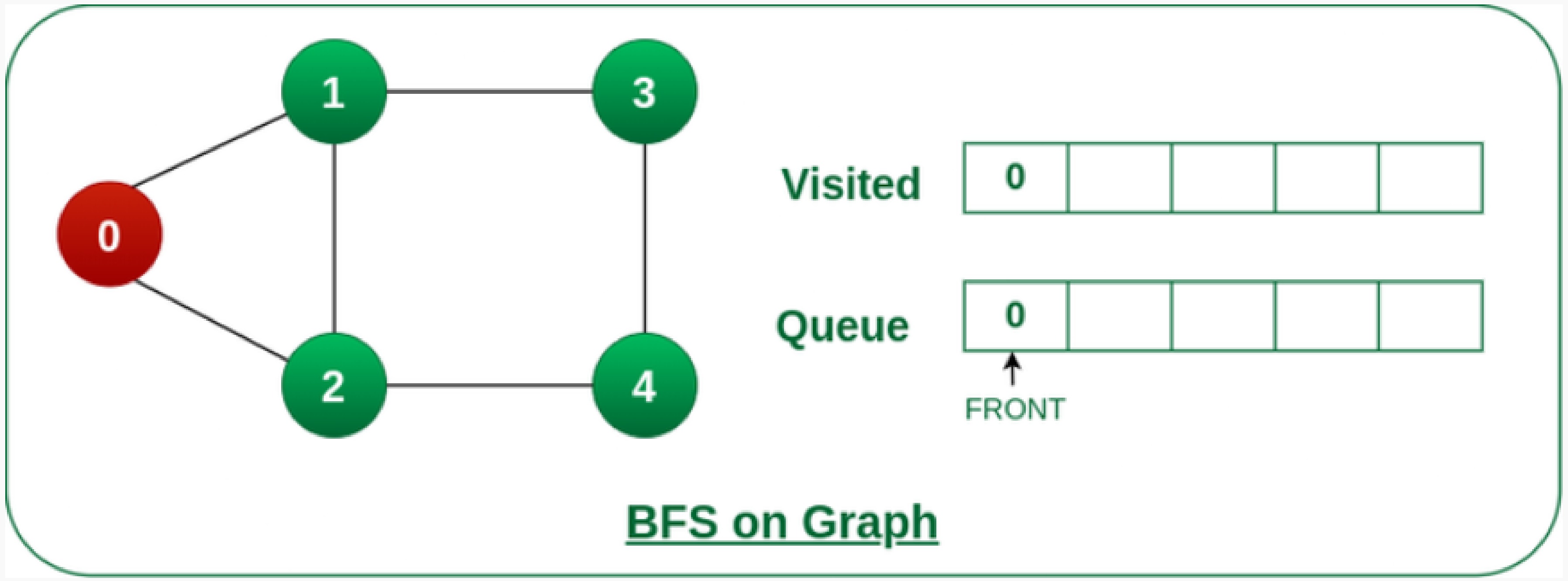
Graph Traversal - BFS

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



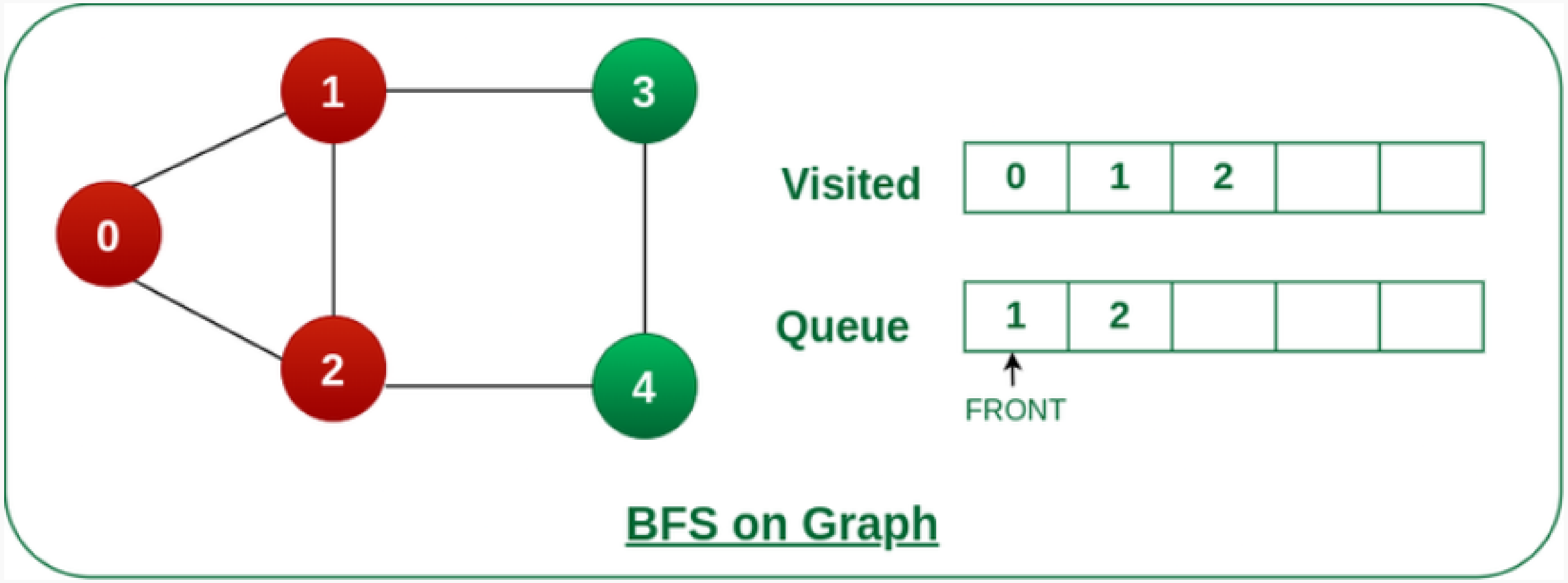
Graph Traversal - BFS

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



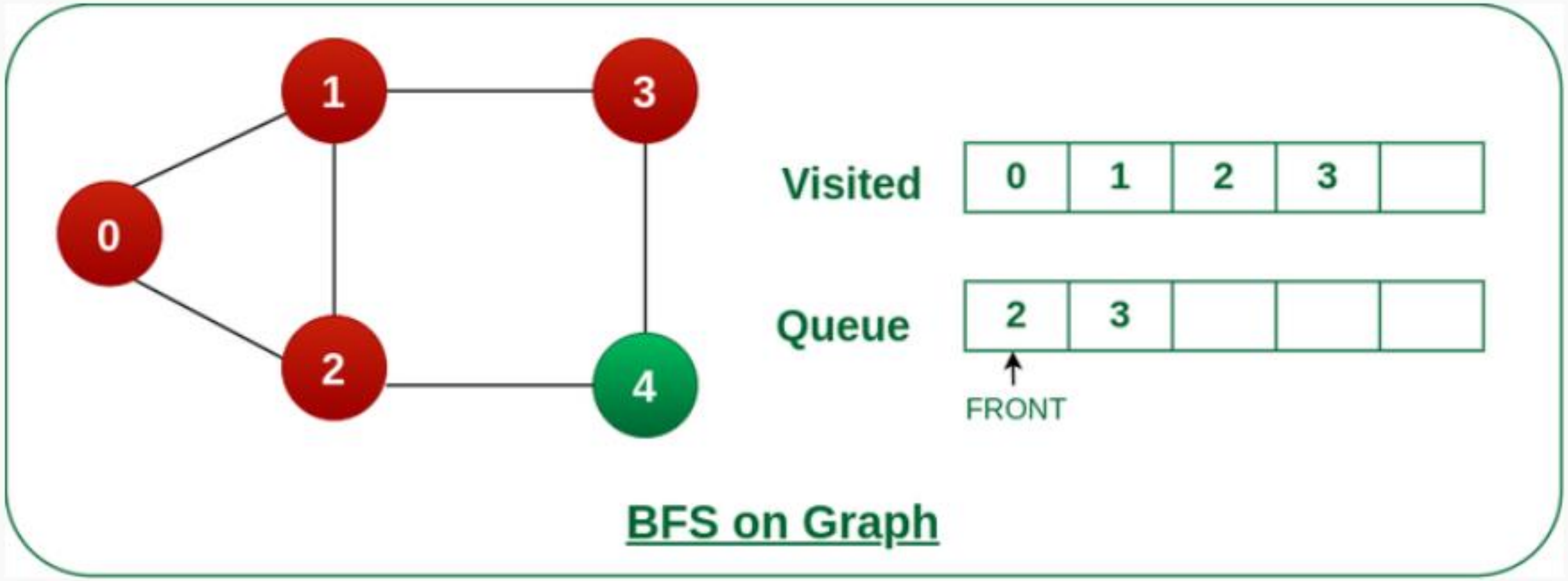
Graph Traversal - BFS

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



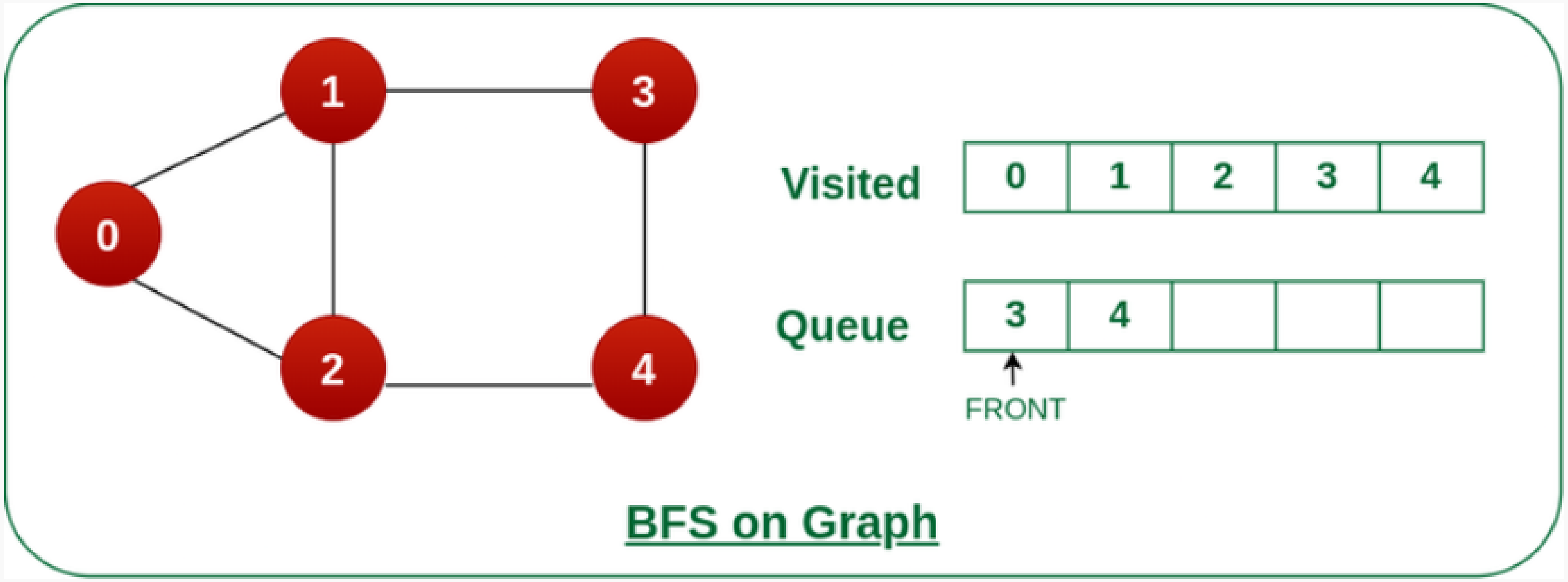
Graph Traversal - BFS

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



Graph Traversal - BFS

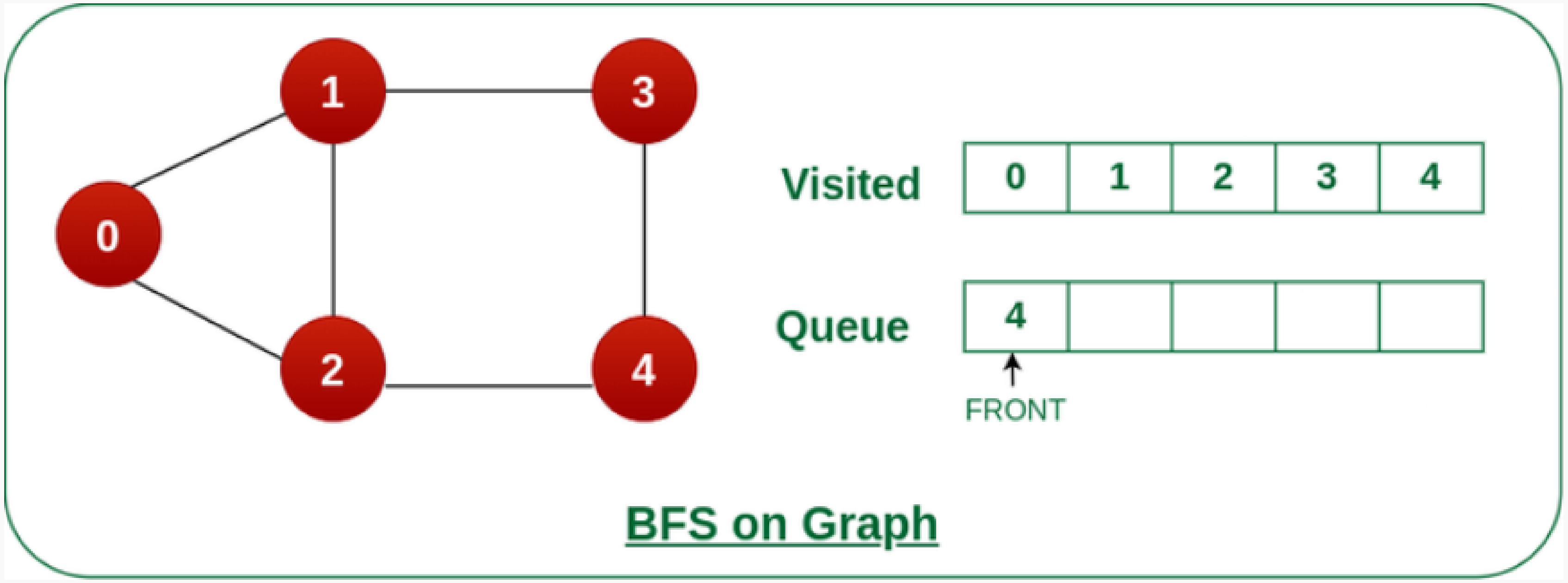
Step 5: *Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*



Graph Traversal - BFS

Step 6: *Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.*

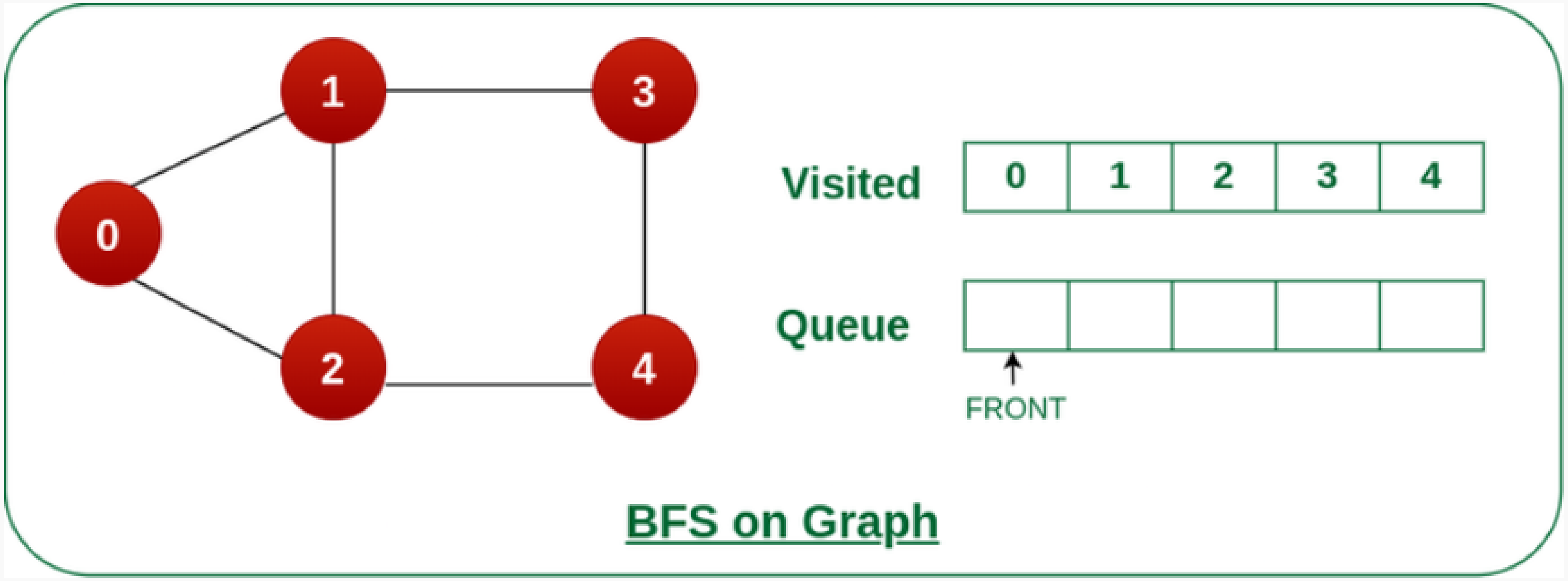
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



Graph Traversal - BFS

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.


0
1
2
3
4
Visited
0 1 2 3 4
Queue
FRONT
BFS on Graph