DEPTH FIRST AND BREADTH FIRST TRAVERSALS

- **Graph traversal** (also known as **graph** search) refers to the process of visiting (checking and/or updating) each vertex in a **graph**.
- Such **traversals** are classified by the order in which the vertices are visited.

Two of the most popular traversal algorithms are

1. **Breadth-first search (BFS)**
2. **Depth-first search (DFS)**.

- Both methods visit all vertices and edges of a graph; however, they are different in the way in which they perform the traversal.

Breadth-first search (BFS)

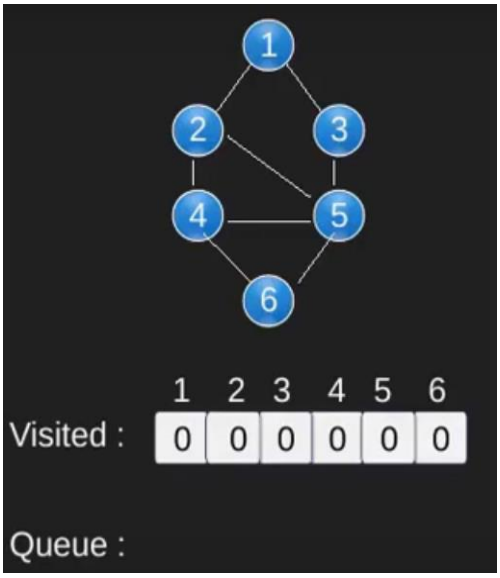
- It **uses a queue** for storing the visited vertices.
- This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node.
- Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them.
- The vertices adjacent to the visited vertex are then visited and stored in the queue sequentially.
- Similarly, the stored vertices are then treated one by one, and their adjacent vertices are visited.
- These iterations continue until all the nodes of the graph have been successfully visited and marked.
- A node is fully explored before visiting any other node in the graph.
- The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadth wise fashion.

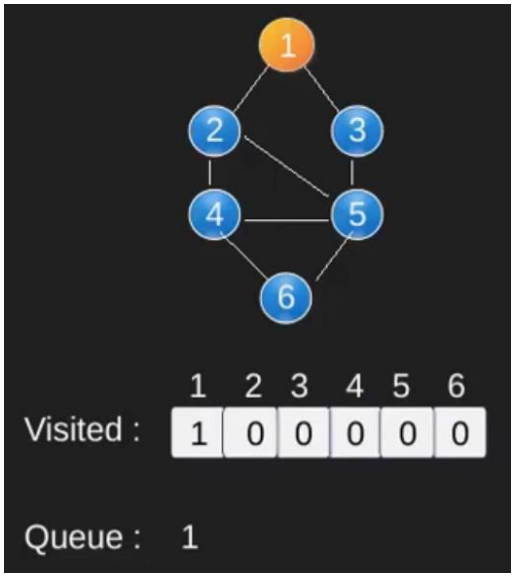
Rules of BFS Algorithm

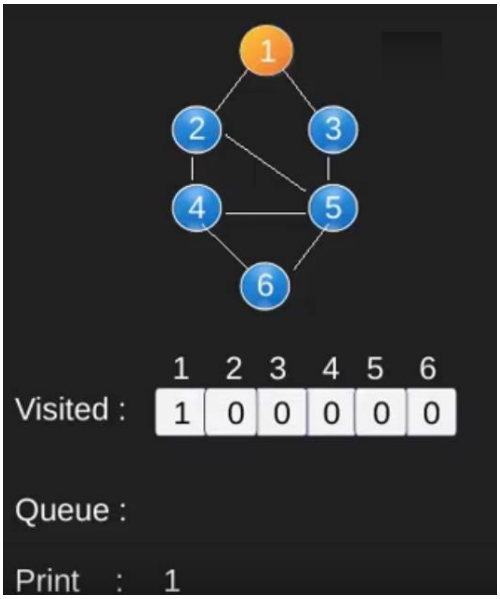
✓ A queue (FIFO-First in First Out) data structure is used by BFS.
✓ You mark any node in the graph as root and start traversing the data from it.
✓ BFS traverses all the nodes in the graph and keeps dropping them as completed.

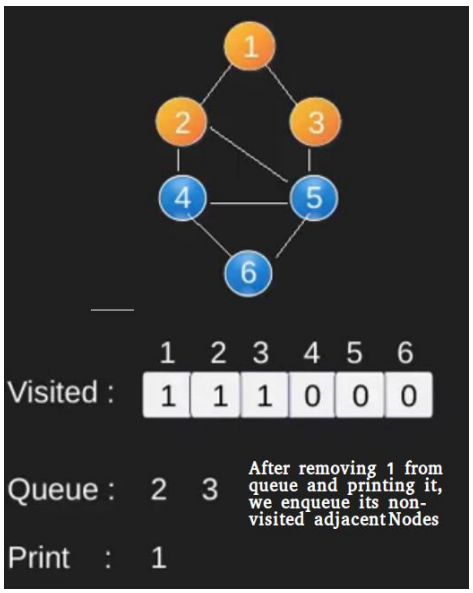
- ✓ BFS visits an adjacent unvisited node, marks it as done, and inserts it into a queue.
- ✓ Removes the previous vertex from the queue in case no adjacent vertex is found.
- ✓ BFS algorithm iterates until all the vertices in the graph are successfully traversed and marked as completed.
- ✓ There are no loops caused by BFS during the traversing of data from any node.

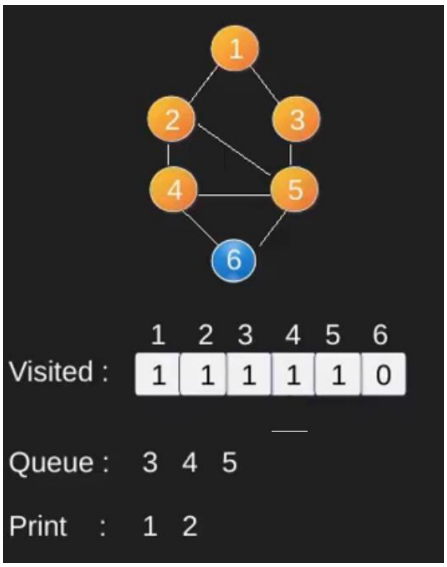
EXAMPLE

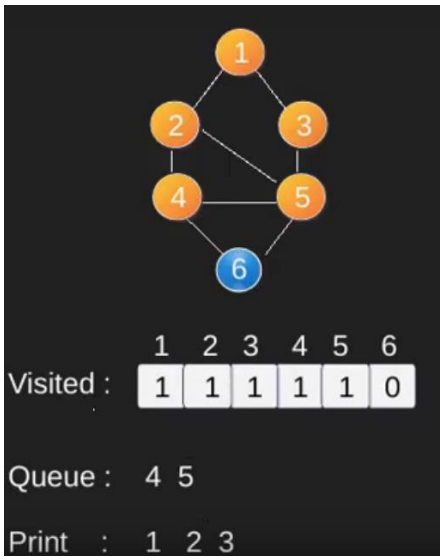


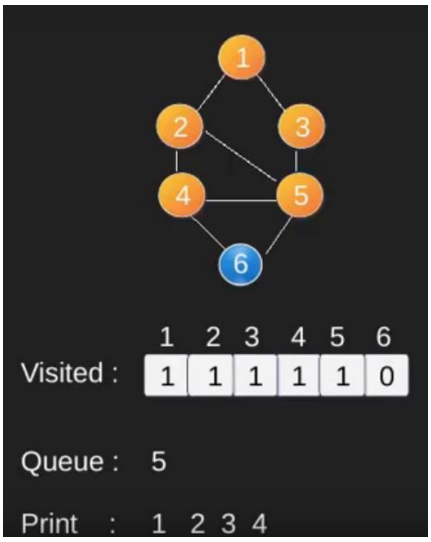


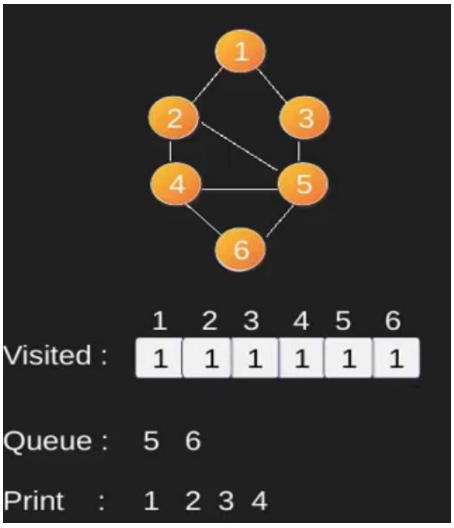


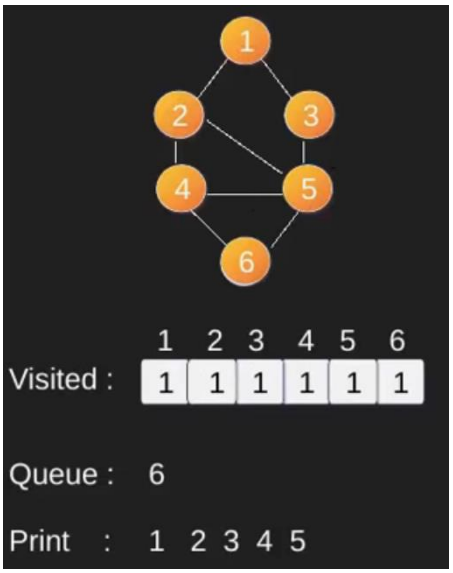


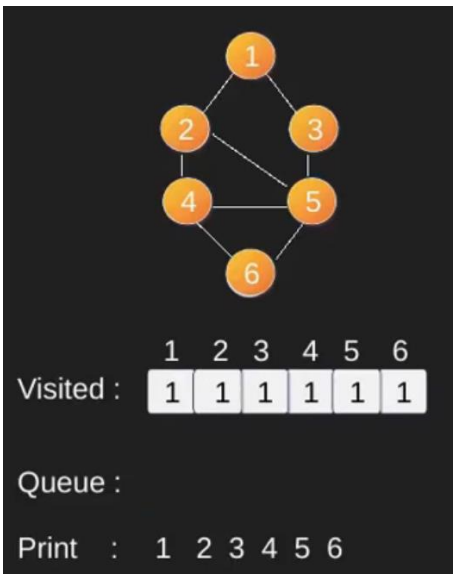

1 2 3 4 5 6
Visited : 1 1 1 1 1 0
Queue : 3 4 5
Print : 1 2


1 2 3 4 5 6
Visited : 1 1 1 1 1 0
Queue : 4 5
Print : 1 2 3


1 2 3 4 5 6
Visited : 1 1 1 1 1 0
Queue : 5
Print : 1 2 3 4


1 2 3 4 5 6
Visited : 1 1 1 1 1 1
Queue : 5 6
Print : 1 2 3 4


1 2 3 4 5 6
Visited : 1 1 1 1 1 1
Queue : 6
Print : 1 2 3 4 5


1 2 3 4 5 6
Visited : 1 1 1 1 1 1
Queue :
Print : 1 2 3 4 5 6

```
BFS (G, s) //Where G is the graph and s is the source node

      let Q be queue.

      Q.enqueue(s) //Inserting s in queue until all its
neighbour vertices are marked.

      mark s as visited.

      while (Q is not empty)

          //Removing that vertex from queue, whose neighbour
will be visited now.

           v = Q.dequeue ( )

          //processing all the neighbours of v

          for all neighbours w of v in Graph G

               if w is not visited

                         Q.enqueue (w)

            //Stores w in Q to further visit its neighbour

                         mark w as visited.
```

Depth First Search (DFS)

- The DFS algorithm is a recursive algorithm that uses the idea of backtracking.
- It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.
- Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse.
- All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.
- This recursive nature of DFS can be implemented using stacks.

The basic idea is as follows:

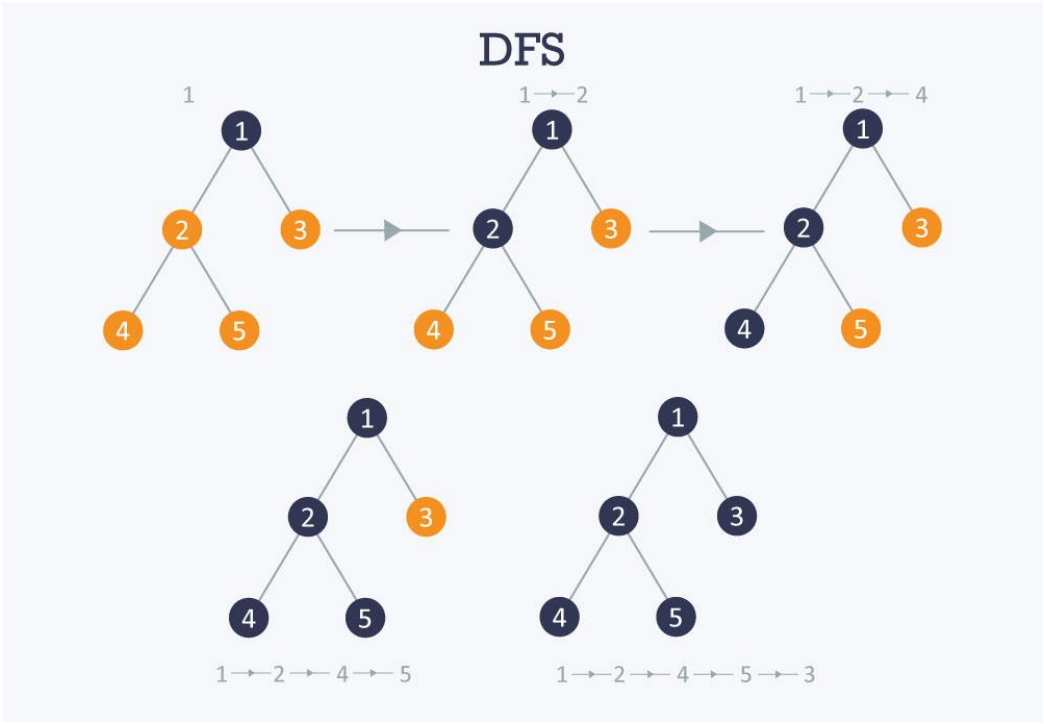
- ✓ Pick a starting node and push all its adjacent nodes into a stack.
- ✓ Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.
- ✓ Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked.

1. **DFS-iterative(G, s): //Where G is graph and s is source vertex**

```
let S be stack
S.push( s )   //Inserting s in stack
mark s as visited.
  while (S is not empty):
      //Pop a vertex from stack to visit next
    v = S.top ( )
   S.pop ( )    //Push all the neighbours of v in stack
                that are not visited
  for all neighbours w of v in Graph G:
      if w is not visited :
               S.push( w )
              mark w as visited
```

2. **DFS-recursive(G, s):**

```
  mark s as visited
  for all neighbours w of s in Graph G:
      if w is not visited:
          DFS-recursive(G, w)
```


DFS
1
1 → 2
1 → 2 → 4
1 → 2 → 4 → 5
1 → 2 → 4 → 5 → 3