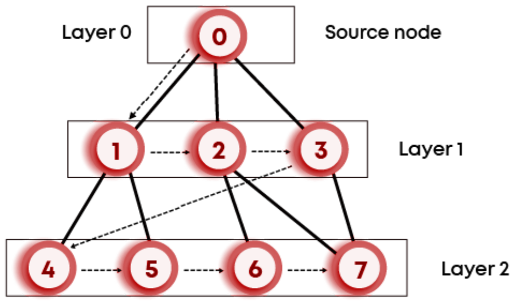
Breadth-first search (BFS)

Breadth-first search(BFS) is a graph search algorithm where we start from the selected node and traverse the graph level-wise or layer-wise, thus exploring the neighbor nodes (which are directly connected to the starting node), and then moving on to the next level neighbor nodes.

As the name suggests:

- We first move horizontally and visit all the nodes of the current layer.
- Then move to the next layer.



BFS Traversal: 0->1->2->3->4->5->6->7

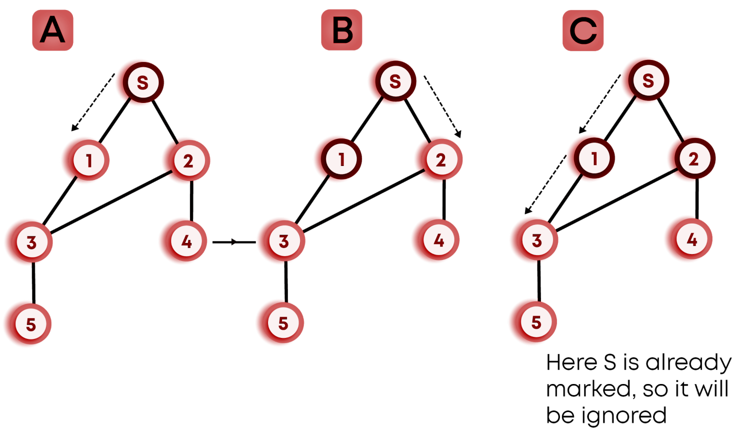
Other possible BFS traversals for the above graph can be:

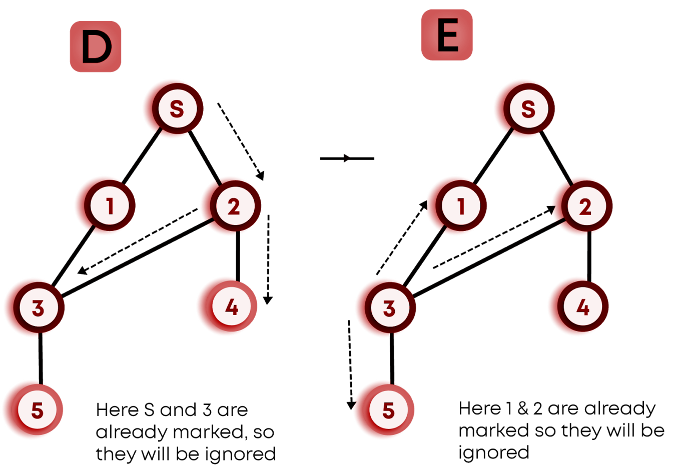
1. 0->3->2->1->7->6->5->4
2. 0->1->2->3->5->4->7->6

We can clearly observe that there can be more than one BFS traversals for the same graph, as shown for the above graph.

That is, we start examining say node A and then all the neighbors of A are examined. In the next step, we examine the neighbors of A, so on, and so forth. This means that we need to track the neighbors of the node and guarantee that every node in the graph is processed and no node is processed more than once. This is accomplished by using a queue that will hold the nodes that are waiting for further processing and a variable VISITED to represent the current state of the node.

For example: BFS for the below graph is:




D
E
S
1
2
3
4
5
Here S and 3 are already marked, so they will be ignored
Here 1 & 2 are already marked so they will be ignored

Implementation of BFS

```
function BFS(graph,source)

/*
Let Q be a queue, pushing source vertex in the queue.
        Q represents the vertices that have not been processed
        /visited so far, and their neighbors have not been processed.
*/

Q.enqueue(source)
visited[source] = true

//  Iterate through the vertices in the queue.
while Q is not empty
         //  Pop a vertex from the queue to visit its neighbors
                  cur = Q.front()
```

```
                    Q.dequeue()

                    /*
Push all the neighbors of the cur vertex that have not been visited yet, push them into the
queue and mark them as visited.
                    */

                    for all neighbors v of cur in graph:
                        if visited[v] is false
                            Q.enqueue(v)
                            visited[v] = true

          return
```

Features of Breadth-First Search Algorithm

- **Time Complexity:** The time complexity of a breadth-first search is proportional to the number of vertices plus the number of edges in the graphs that are traversed. The time complexity can be given as **(O(|V| + |E|))**, where **|V|** is the number of vertices in the graph and **|E|** is the number of edges in the graph, considering the graph is represented by adjacency list.
- **Completeness:** Breadth-first search is said to be a **complete** algorithm in the case of a finite graph because if there is a solution, the breadth-first search will find it regardless of the kind of graph. But in the case of an infinite graph where there is no possible solution, it will diverge.

Applications of Breadth-First Search algorithm

- Finding all connected components in a graph G.
- Finding all nodes within an individual connected component
- Finding the shortest path between two nodes, u and v, of an unweighted graph. (The shortest path is in terms of the minimum number of **moves** required to visit v from u or vice-versa).

NOTE

- The DFS and BFS algorithms work for both **directed** and **undirected** graphs, **weighted** and **unweighted** graphs, **connected** and **disconnected** graphs.
- For disconnected graphs, for traversing the whole graph, we need to call DFS/BFS for each **unvisited** vertex.

- For getting the number of connected components in a graph, the number of times we need to call DFS/BFS traversal for the graph on an **unvisited** vertex gives the number of connected components in the graph.