Page No.
Date

METHOD → 1

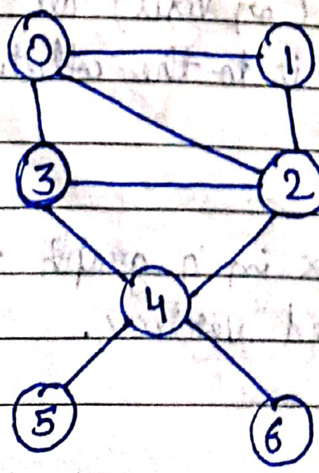
Breadth First Search (BFS)

→ Graph traversal refers to the process of visiting (checking and / or updating) each vertex (node) in a graph.

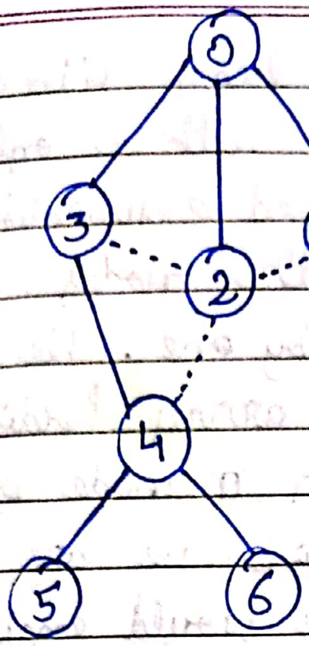
→ Two Algorithms of Graph Traversal are:
- → Breadth First Search (BFS)
- → Depth First Search (DFS)

→ In BFS, we start with a node and start exploring its connected nodes. The same process is repeated with all the connecting nodes until all the nodes are visited.

→ BFS spanning tree:
- → Consider the graph shown.
- → We can start with any source node.
- → Lets start with 0.
- → Try to contruct a tree with 0 as the root.



GIVEN



MODIFIED

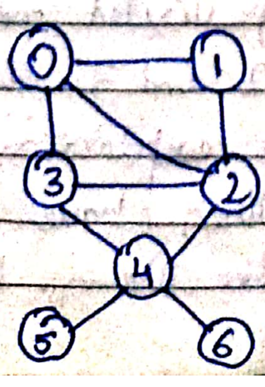
Think 0 as ROOT node, and think of tyeing it with the ceiling. Then everynode will align something like this.

→ Mark all the sideways or duplicate edges (above a node) as dashed.
→ This constructed tree is called as BFS Spanning Tree.
→ Level order Traversal of a BFS spanning tree is a valid BFS traversal of a graph!

LOT ⇒ 0 3 2 1 4 5 6
↳ ~~a~~ this is BFS traversal as Well.

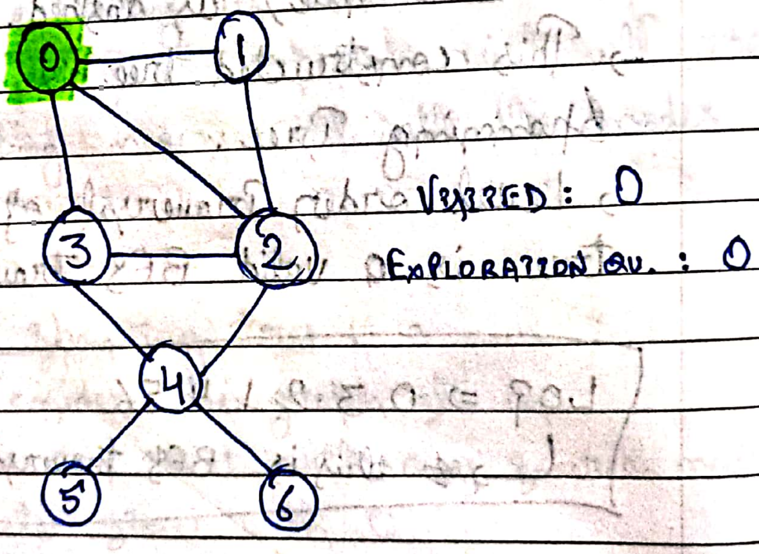
METHOD → 2

Conventional Breadth-First Search Traversal Algorithm:



Page No.
Date

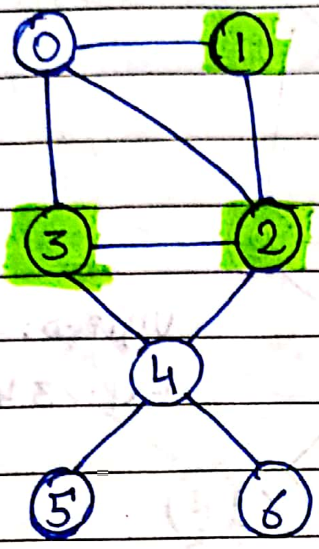
→ Considering we could begin with any source node; we'll start with 0 only. Let's define a queue named exploration queue, which would hold the nodes we'll be exploring one by one. We would maintain another array holding the status of whether a node is already visited or not. Since we are starting with node 0, we would enqueue 0 into our exploration queue and mark it visited.



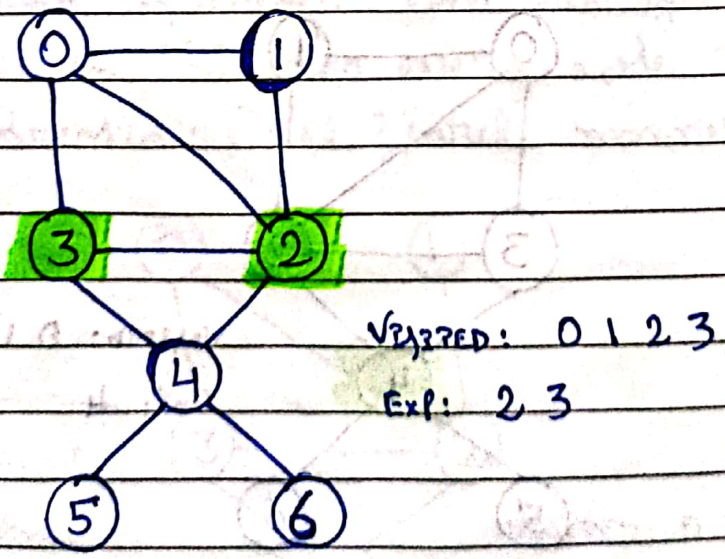
→ Now, we'll start visiting all the nodes connected to node 0, and remove node 0 from the exploration queue, enqueuing all the currently visited nodes which were nodes 1, 2 and 3. We are pushing them inside the exploration queue because these might further have some unvisited nodes connected to them. Mark these

Scanned with CamScanner

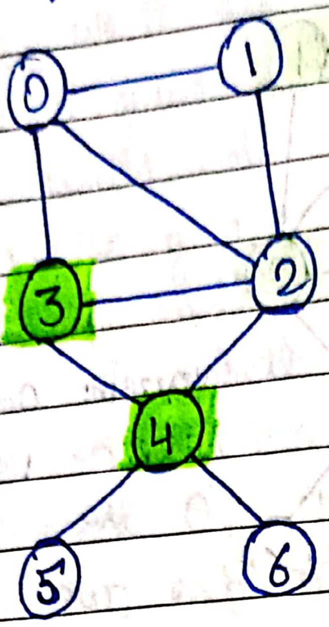
nodes visited as well.



→ After this we have node 1 at the top in the exploration queue, so, we'll take it out and visit all unvisited nodes connected to it. Unfortunately, there aren't any. Therefore, we'll continue exploring further.



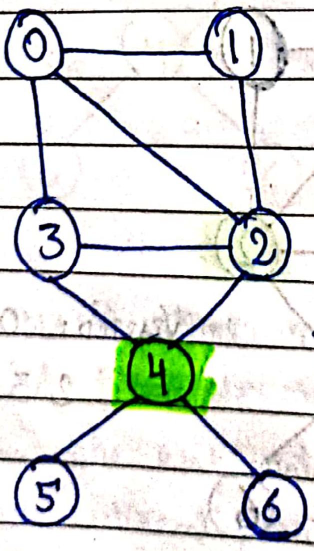
→ Next, we have node 2. And the only unvisited node connected to node 2 is node 4. So, we'll mark it visited and will also enqueue it in our exploration queue.



VISITED: 0 1 2 3 4

EXP: 3 4

→ Node 3 is the next in line. Since, all nodes 1, 2 and 4 which are the nodes connected to it are already visited, we have nothing to do here while we are on node 3.



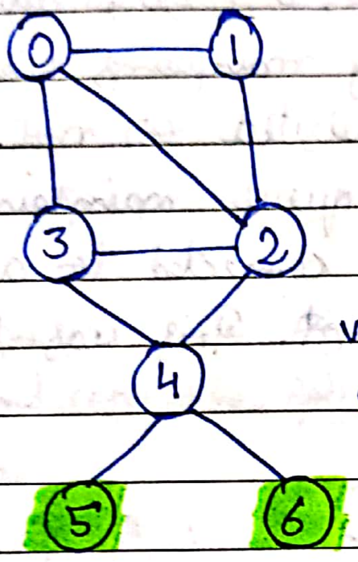
VISITED: 0 1 2 3 4

EXP: 4

→ Next, we have node 4 on the top in the exploration queue. Let's get it out and see what nodes are connected and

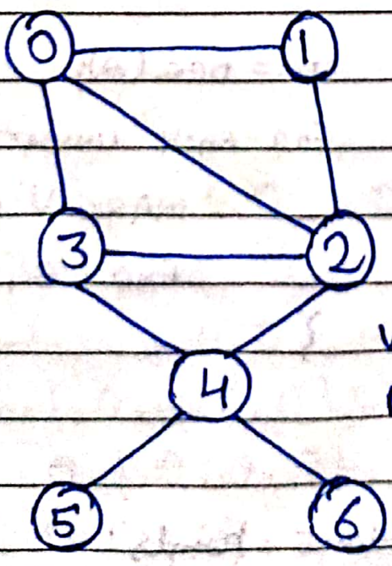
Page No.
Date

unvisited to it,.. So, we got nodes 5 and 6. Mark them visited and push them inside the exploration queue.



VISITED: 0 1 2 3 4 5 6
EXP: 5 6

And now we can explore the other two nodes left in the queue, and since all nodes are already visited, we'll get nothing in them. And this got our queue emptied and every node traversed in Breadth-First Search manner.



VISITED: 0 1 2 3 4 5 6
EXP:

Scanned with CamScanner

→ And the order in which we marked our nodes visited is the Breadth-First Search traversal order. Here, it is 0, 1, 2, 3, 4, 5, 6. So, basically, the visited array maintains whether the node itself is visited or not, and the exploration queue maintains whether the nodes connected to a node are visited or not. This was the difference.

<u>PSEUDOCODE</u>

```
→ INPUT: A graph G = (V, E) and source code s in V
→ ALGORITHM:

→ MARK ALL NODES V in V as unvisited.
→ MARK SOURCE NODE S AS VISITED.
→ ENQ(Q, S)      // First-in-first-out Queue
→ WHILE (Q IS NOT EMPTY)
      {
          U := DEQ(Q);
          FOR EACH UNVISITED NEIGHBOUR V of U {
                  MARK V AS VISITED;
                  ENQ(Q, V);
          }
      }
```

→ Few important points:

Page No. ______

Date

1) We can start with any vertex.
2) There can be multiple Breadth - First Search results for a given graph.
3) The order of visiting the vertices can be anything.