BREADTH FIRST SEARCH (BFS)

BFS for a graph is similar to the BFS of a tree.

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
-> Not visited.

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.

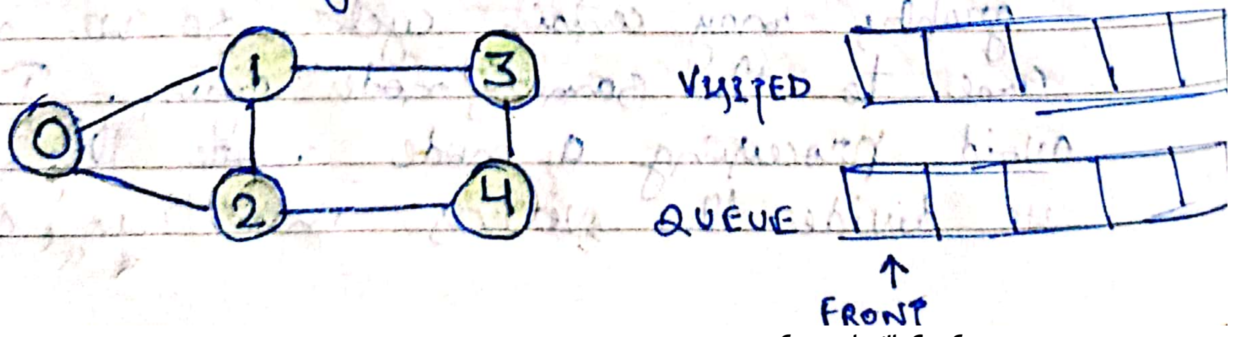
How does BFS work?

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.

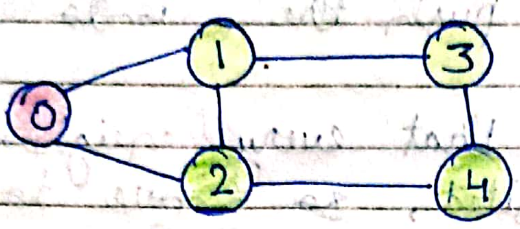
To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

e.g:

STEP: 1 Initially queue and visited arrays are empty!!



STEP: 2 Push node 0 into queue and mark it visited.



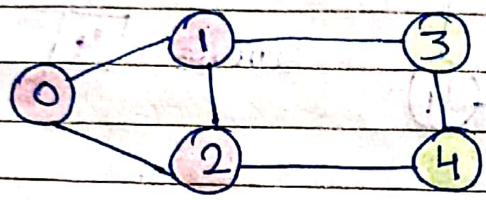
VISITED

0				

QUEUE

0				

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



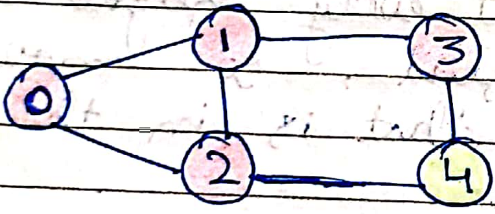
VISITED

0	1	2		

QUEUE

1	2			

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



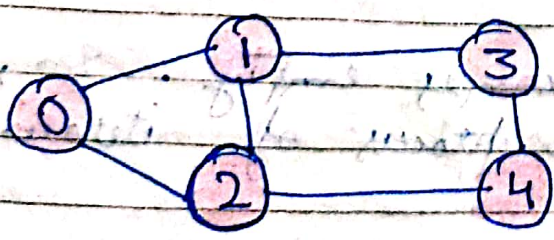
VISITED

0	1	2	3	

QUEUE

2	3			

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



VISITED

0	1	2	3	4

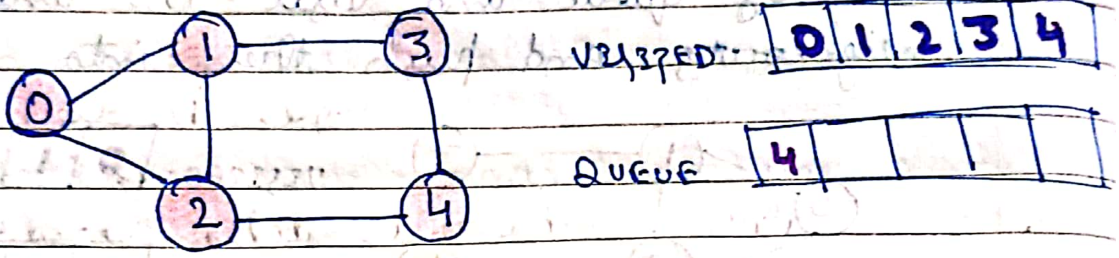
QUEUE

3	4			

Page No
Date

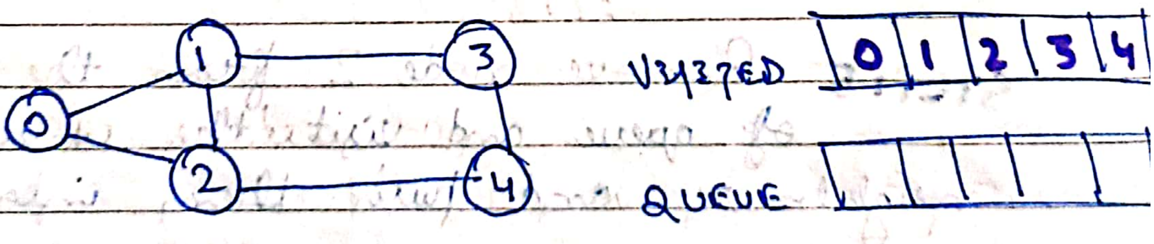
STEP: 6 Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.

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



STEP: 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.



Now, Queue becomes empty. So, terminate these process of iteration.

Scanned with CamScanner

Page No.
Date

CODE

```
CONST INT N = 1e5 + 10;
VECTOR <INT> G[N];
-> Visited array
INT VIS [N];
-> Level array
INT LEVEL[N];

VOID BFS (INT SOURCE)
{
        QUEUE <INT> Q;  // queue to store current
        Q.PUSH (SOURCE);            vertex
        -> Marking current vertex visited. So,
           we don't come back to it again.
        VIS [SOURCE] = 1;

        WHILE (!Q.EMPTY())
        {
              INT CUR_V = Q.FRONT ();
              -> Popping out the current vertex
              Q.POP ();
              -> Printing order of popping (just to
                 visualize)
              COUT << CUR_V << " ";
              -> Visiting all children of current
                 vertex
              FOR (INT CHILD : G[CUR_V])
              {
                    -> If that child is not already
                       in visited array
                    IF (!VIS[CHILD])
```

Page No.
Date

```
                {
                    Q.PUSH (CHILD);
                    VIS [CHILD] = 1;
                    LEVEL [CHILD] = LEVEL [CUR_V]+1
                }
            }
        }
        -> V = vertex , E = Edges
        -> Time complexity = O (V+E)
}


INT MAIN ( )
{
    INT n;
    CIN >> n;

    FOR ( INT i=0; i< n-1 ; i++ )
    {
        INT V1, V2;
        CIN >> V1 >> V2;

        G[V1]. PB (V2);
        G[V2]. PB (V1);
    }

    BFS (1);

    -> Printing level (Just to visualize)
    FOR (INT i = 1; i <= n; i++ )
    {
        COUT << i << ": " << LEVEL[i] << "\n";
```

Scanned with CamScanner

Page No. ______
Date

```
    }

    RETURN 0;
}
```

INPUT:

```
13
1  2
1  3
1  13
2  5
5  6
5  7
5  8
8  12
3  4
4  9
4  10
10 11
```

OUTPUT:

```
1 2 3 13 5 4 6 7
8 9 10 12 11
1 : 0
2 : 1
3 : 1
4 : 2
5 : 2
6 : 3
7 : 3
8 : 3
9 : 3
10 : 3
11 : 4
12 : 4
13 : 1
```