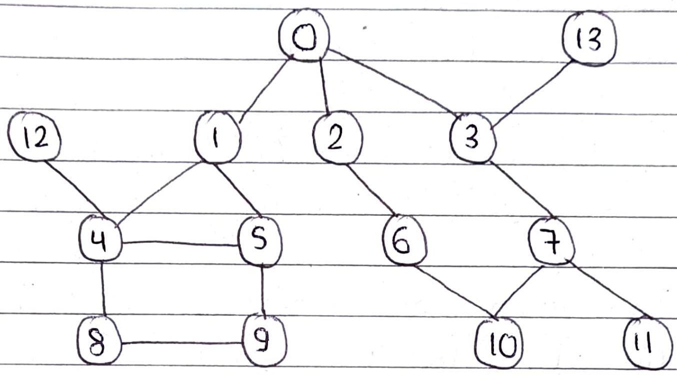
Day - 199

Graph - 3

* **BFS Traversal:**



⇉ BFS Traversal is like starting from one node and then traverse the connected nodes and again do the same process.

⇉ BFS Traversal of the above Graph —
(we are starting from 0)

→ 0

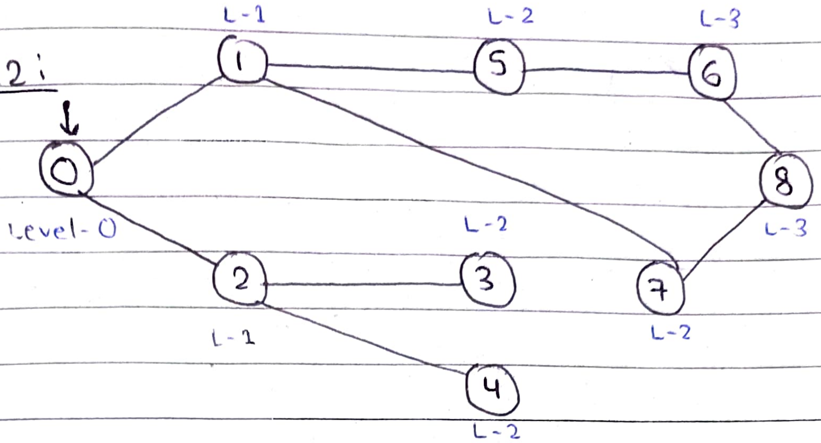
→ 1 2 3

→ 4 5 6 7 13

→ 8 12 9 10 11

⇉ BFS → Breadth First Search.

Ex2 :



⇒ 0 1 2 5 7 3 4 6 8
(we will only visit non-visited nodes)

⇒ So, we will get to know that we have to use queue here.

⇒ So, neighbours info. get from the adjacency list.

⇒ Also, we will take an array of size equals to no. of nodes in the graph so that we will know which node is visited or not.

⇒ So, the flow will be like this —

→ Put the starting node in a queue.

→ Make a visited array.

→ Then pop the element from the queue & push their neighbours in the queue.

→ And only non-visited ~~will~~ nodes will come in queue.

⇒ Also, make the visited of that node to 1.

Code

⇉

```
vector <int> BFS Graph (int V, vector<int>
adj[ ]){
    queue <int> q;
    vector <bool> visited (v, 0);
    q.push (0);
    visited [0] = 1;
    vector <int> ans;

    while( !q.empty() ){
        node = q.front ();
        q.pop();
        ans. push_back (node);
        for(int j = 0; j < adj[node].size(); j++){
            if( !visited [adj [node][j] ]){
                q.push_back (adj[node][j]);
                visited[adj[node][j]] = 1;
            }
        }
    }
    return ans;
}
```

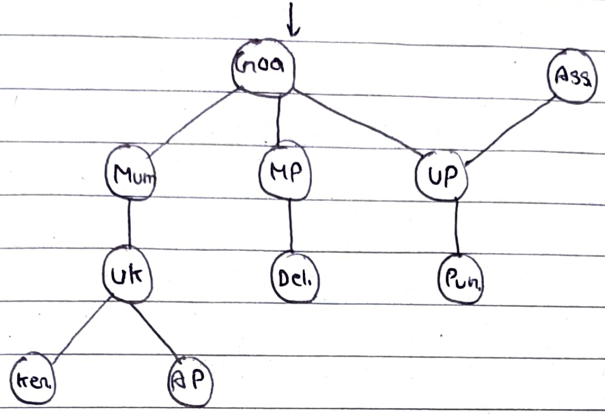
S.C. → O(V).

T.C. → O(V + 2E) → O(V+E)

⇉ BFS is used when we search ~~our~~ any anyone on the social medias.

* D.FS Traversal:

⇒



⇒ We will start from a node and go to the depth of that node and after we reached the end of that side, we will go back to the previous node & explore other paths.

⇒ Goa, Mumbai, Uk, Kerala, AP, MP, Delhi, UP, Punjab, Assam.

⇒ So, we will given a adj. list.

⇒ We can solve DFS by using recursion and stack.

⇒ Also,. we will need a visited array.

⇒ So, we will visit all the neighbours of a node one by one.

⇒ If all the ~~nodes~~ neighbours visited then back to the previous node.

```
void DFS( int node , vector <int> adj[ ],
    vector < bool > &visited , vector<int>&ans){
    visited [node ]= 1;
    ans. push_back (node);

    for(int j = 0 ; j < adj [node ].size() ; j++){
        if(!visited [adj [node ][j ]])
            DFS(adj[node][j], adj, visited,
                ans);
    }
}
```

T.C. → O(V+E)

S.C → O(V)