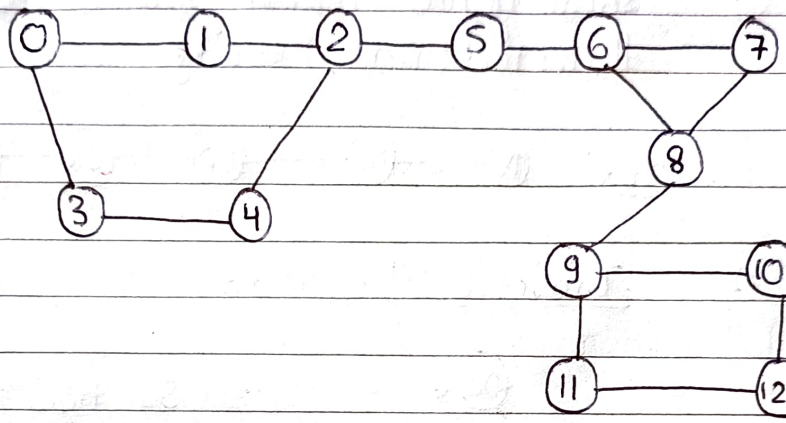
Page:
Date: / /

# Day-218

## Graph - 22

* **Bridge in a Graph:**

⇒




⇒ Bridge is that edge if we remove that edge then our graph divides into two component.

⇒ So here edge b/w → (2)—(5), (5)—(6), (8)—(9)

⇒ Brute force approach will be like we apply BFS & DFS and check we can reach that node or not after removing that edge.

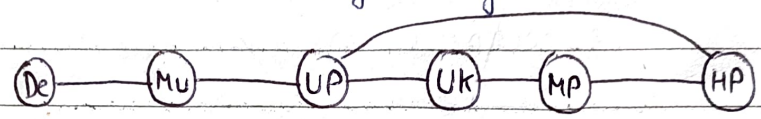
⇒ But this will increase our T.C.

Page:
Date: / /

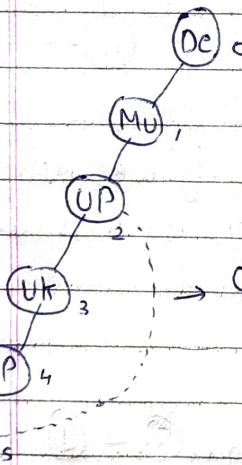
=> So, if we observe that edges other bridges edges all are the part of cycle.

=> If the node is not the part of same cycle then also edge b/w them is a bridge edge.

Ex:




=> Traverse the Graph




=> So, how to identify bridge edges now.

=> Edge b/w UP - HP is not bridge is confirmed.

=> So, how about MP - HP.

=> For this, we will ask HP that it can visit any node that come before me (MP).

=> So, it will say yes because it can visit UP.

=> So, MP - HP also the part of cycle.

Page:
Date / /

⇒ Data will be stored like this —

HP: HP, UP

MP: MP HP UP

4 5 2

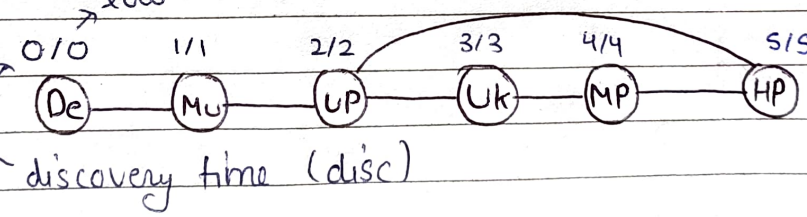
⇒ If any value here, is less than or equal to parent value, then the edge is not a bridge.

⇒ But, we have to traverse all the values of array for checking.

⇒ For that, we can just store the min. value.

⇒ This time or value, we called it as discovery time.

⇒




⇒ We will update the value of low with min of parent-child low value.

For bridge

disc[parent] < low[child]

Page:
Date: / /

⇒ Three conditions:

① if( ~~#~~ ∃ neighbour == parent )
ignore

② else if ( visited [neighbour] )
low [node] = min(low [node],
low[neighbour])

③ else {
DFS ( )
if ( dis[node] < low[neigh] )
bridge ++ ;