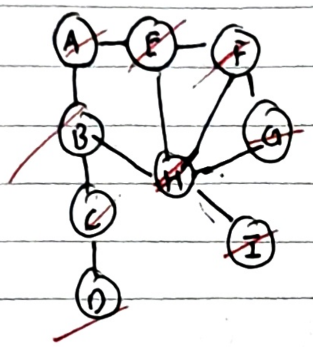
Wk 7 Tutorial

Q1)



a) breadth-first search starting from A

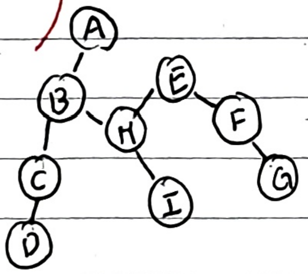
l_0: {A}

l_1: {B, E}

l_2: {C, H, F}

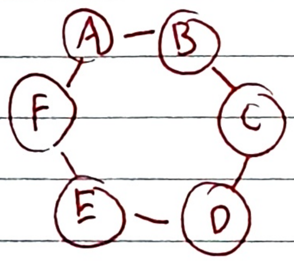
l_4: {D, G, I}

b) depth-first search (from A)



Q2) G = (V, E) said to be biparlile if its vertex set V can be partitioned into two sets A and B such that E ⊆ A × B

a.

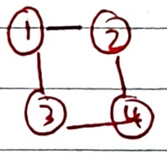


l_0: {A}

l_1: {B, F}

l_2: {C, E}

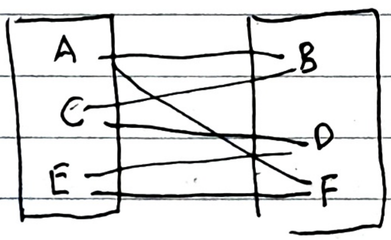
l_3: {D}

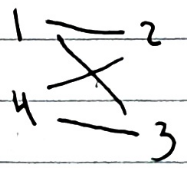


l_0: {1}

l_2: {2, 3}

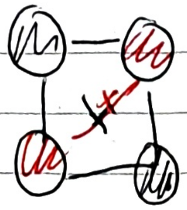
l_4: {4)





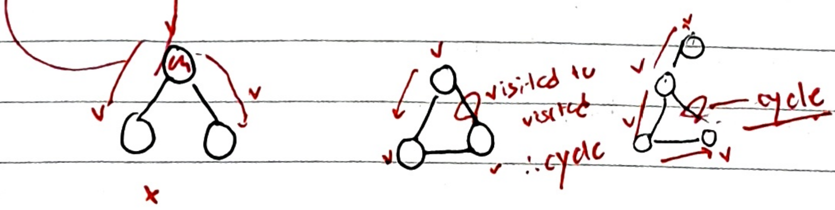
Testing bipartiteness in O(n+m)

- During DFS, create even/odd layers. (color 'em / hash ~~them~~)
- Check there is no intra-layer ~~traversal~~ edge.



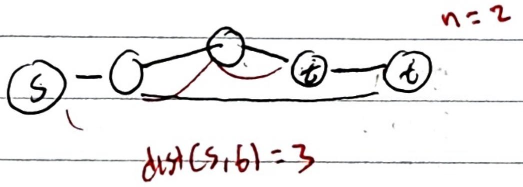
다른 색깔끼리만 이어져 있으면 됨.

Q3) Undired graph, have cycle? O(n)

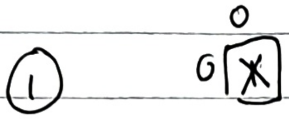


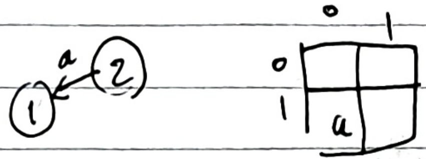
Do DFS. Note that in DFS, nodes are marked as visited upon visit.
If a ~~visited~~ node ~~to~~ is incitedent to other visited node, the graph has a cycle.

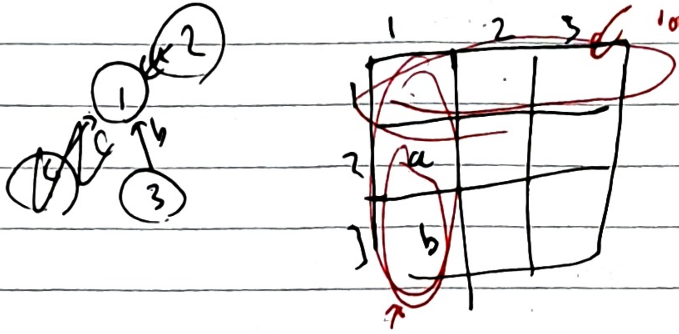
Q4) G=(V,E) be n-vertex graph.
Let s, t be two vertices.
Argue that if dist(s,t) > n/2, then there exists
a vertex u ≠ s such that every path from
s to t goes through u.



Q5) get-stuck vertex has n-1 incoming edges, 0 outgoing edges.





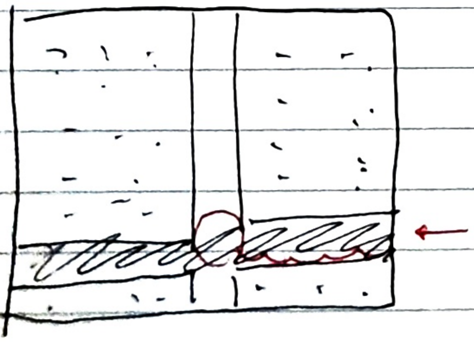


row should have no edges outgoing to other vertices.

Should have edges except one connecting to itself.

```
function is_get_stuck(G v):
  v_idx = G.index~~of~~(v)
   for i in (0, n-1) do
      if not i == v_idx then
   ~~assert~~if G[i][v_idx] ≠ null
           return False
        if G[v_idx][i] ≠ null: then
           return false
     return True
```

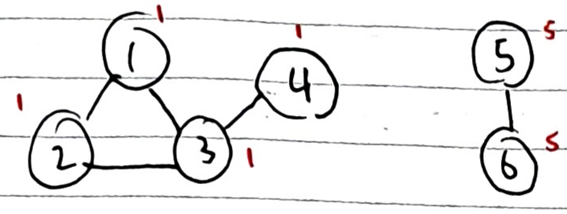
but ~~see~~ testing if graph has get-stuck vertex in O(n)...?



get-stuck !

0 0
0 0 0
1 0 0

Q6)



Do DFS, and for each component of connected trees, find the minincum.

Apply that minimcum for all vertices in that tree O(n+m)

~~Q7)~~ Algorithm that searches for cut vertices in O(n+m) time.