Few Basic Terminologies in Graph

(a) Degree of a node.
- → In degree
- → Out degree

(b) Loops

(c) Connected and Disjoint Graph.
- → Complete graph
- → weighted / unweighted
- → Directed / undirected

(d) Graphs represented as
- → Adjacency list
- → Adjacency matrix

(e) Graph Vs Tree (Connected graph w/o cycle)

(f) Spanning Tree

(G) Multigraph - (have multiple edges b/w 2 nodes.)

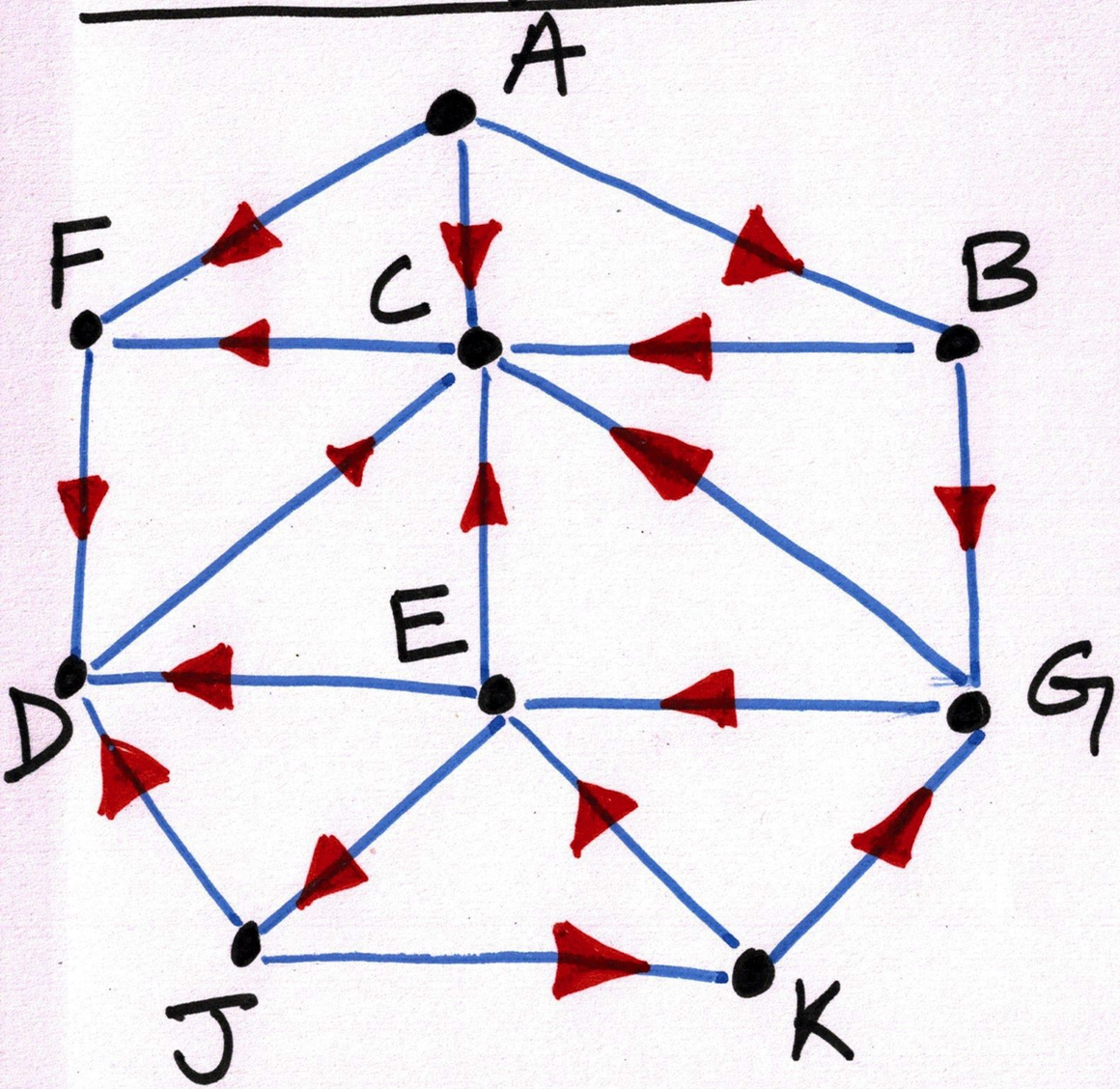
(H) DAG (Directed Acyclic Graphs)

(I) Bi-Connected graph - Connected graph that cannot be broken into disconnected pieces by deleting any single vertex (along with incident edges.

(J) Graph Traversal - Visiting each node exactly once.

- → Depth first Search (DFS)
- → Breadth first Search (BFS)

Traversing a Graph :



DFS, starting at node A.

- examine node A.
- Then one by one examine all the paths originating from A to their dead end.
- (Backtrack and go for some other path) (Similar to Maze traversal)

Adj. List

A → F, C, B
B → G, C
C → F
D → C
E → D, C, J
F → D
G → C, E
J → D, K
K → E, G

DFS - in action: Print all nodes reachable from the node (J).

(a) STACK: [J] (Push J)

(b) Pop and Print (J) (a) then push all neighbors of (J). (Set POP element visited) (only those whose are presently unvisited)

(c) STACK: [D, K], Pop and Print (K) (b) Push all its unvisited neighbors

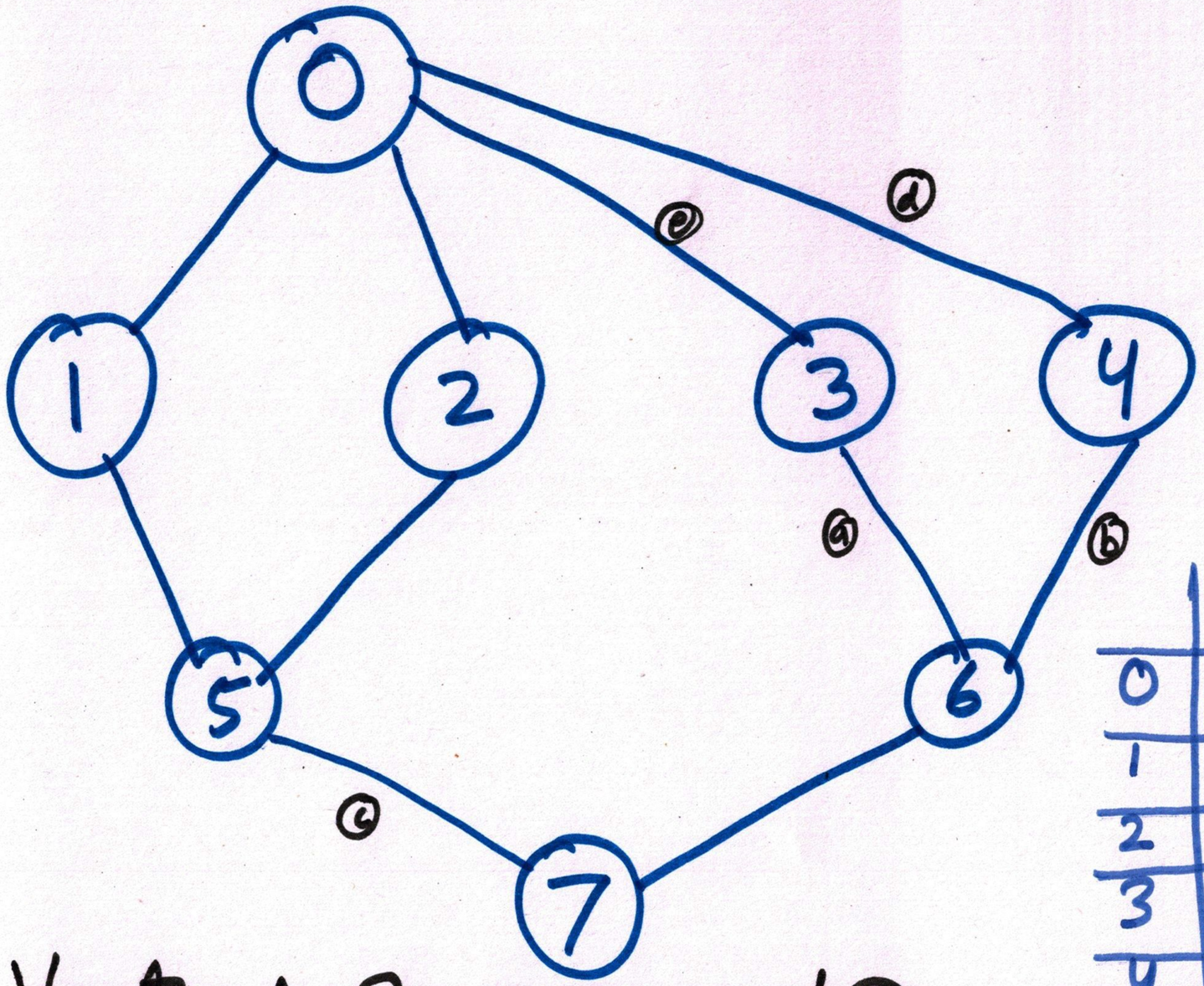
(d) STACK: [D, E, G], Pop and Print (G) (c) Then Push

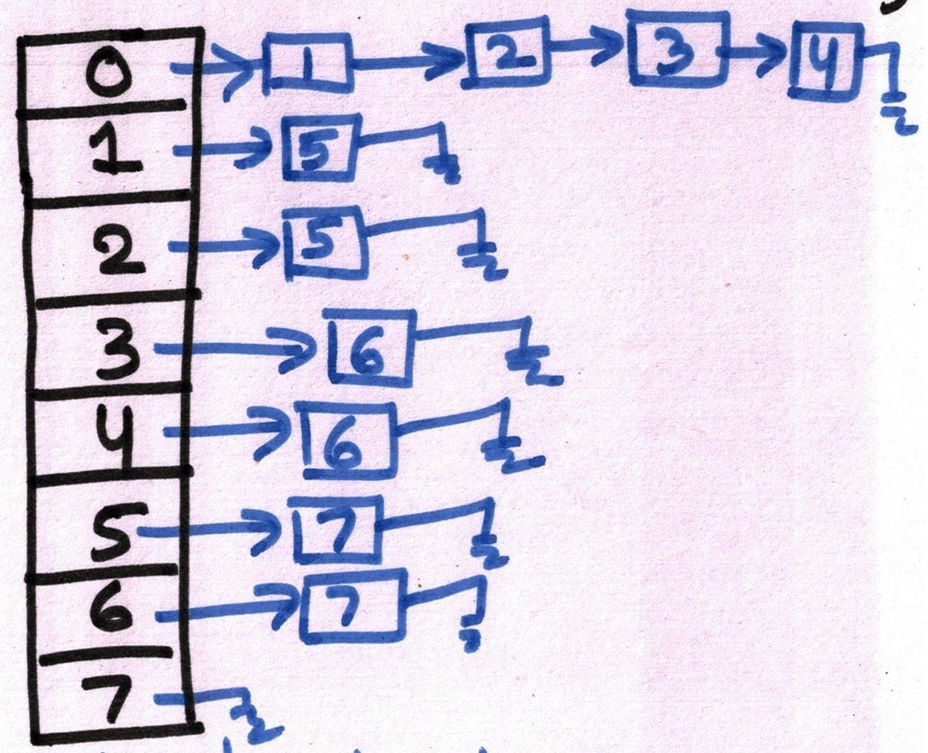
(e) STACK: [D, E, C], Pop and print (C) (d)

(f) STACK: [D, E, F] → (F) (e)

(g) STACK: [D, E] → (E) (f)

(H) STACK: [D] → (D) (g)





	0	1	2	3	4	5	6	7
0	0	1	1	1	1	0	0	0
1	1	0	0	0	0	1	0	0
2	1	0	0	0	0	1	0	0
3	1	0	0	0	0	0	1	0
4	1	0	0	0	0	0	1	0
5	0	1	1	0	0	0	0	1
6	0	0	0	1	1	0	0	1
7	0	0	0	0	0	1	1	0

Vertex : 8 , Edges = 10

Basic DFS logic:

n = no. of nodes

Initialize visited[] array to false.

```
Void DFS(Vertex i)          // DFS starting from i
{
    Visited[i] = 1;

    for each w adjacent to i
        if (! visited[w])
            DFS(w);
}
```

recursive call.

Breadth-First Search (BFS)

① This uses FIFO hence queue is used

② DFS uses LIFO hence (stack) was involved.

③ Each node (N) can be in one of these three states.

- → STATUS = 1 (Ready state)/Initial (Initial State)
- → STATUS = 2 (Waiting) (on Queue waiting to be processed)
- → STATUS = 3 (Processed State)/Visited (The node is already processed.

④ First examine entry node (A). Then examine all of its neighbours (make sure <u>not</u> to examine any node twice.).
Then examine the neighbors of neighbours.

* How to ensure that <u>no</u> node is processed <u>twice</u>.

↳ Use a Queue holding nodes ~~xxxx~~ processes.

That are waiting to be

1. Initialize the (STATUS = 1) for all nodes.
2. Put Starting node A in Q, Change its status to waiting state. (STATUS = 2)
3. Repeat Steps 4 and 5 untill Q is empty.
4. Dequeue an element (N). Process N and Changed its (STATUS = 3)
5. En queue all the <u>neighbours of (N)</u>, with <u>STATUS=1</u>
   Also change their STATUS = 2

Loop ends

6. EXIT.

BFS of previous Graph: from A

(Graph of slide 3)

A at distance (0)

B C F at distance (1)

G D at distance (2)

E at distance (3)

J at distance (4)

---

from O

(Graph of slide 5)

O at distance (0)

1 2 3 4 at distance (1)

5 6 at distance (2)

7 at distance (3)