TUTORIAL - 05

Answer - 1.

BFS	DFS
i) uses queue data structure	→ uses Stack data Structure
ii) Stands for Breadth first search.	→ stands for Depth first search.
iii) Can be used to find single source Shortest path in an unweighted graph. & we reach a vertex with max. no. of edges from a Source vertex.	→ we might traverse through more edges to reach a destination vertex from a source.
iv) Siblings are visited before the Children.	→ Children are visited before the siblings.
Applications:-	Applications:-
1. shortest path & minimum Spanning tree for unweighted graph.	1. Detecting cycle in a graph.
. peer to peer network.	2. path finding.
Social Networking websites.	3. Topological sorting.
GPS Navigation systems.	4. solving puzzles with only one solution.

Answer-2:- In BFS, we use Queue data structure as queue is used when things don't have to be processed immediately, but have to be processed in FIFO order like BFS.

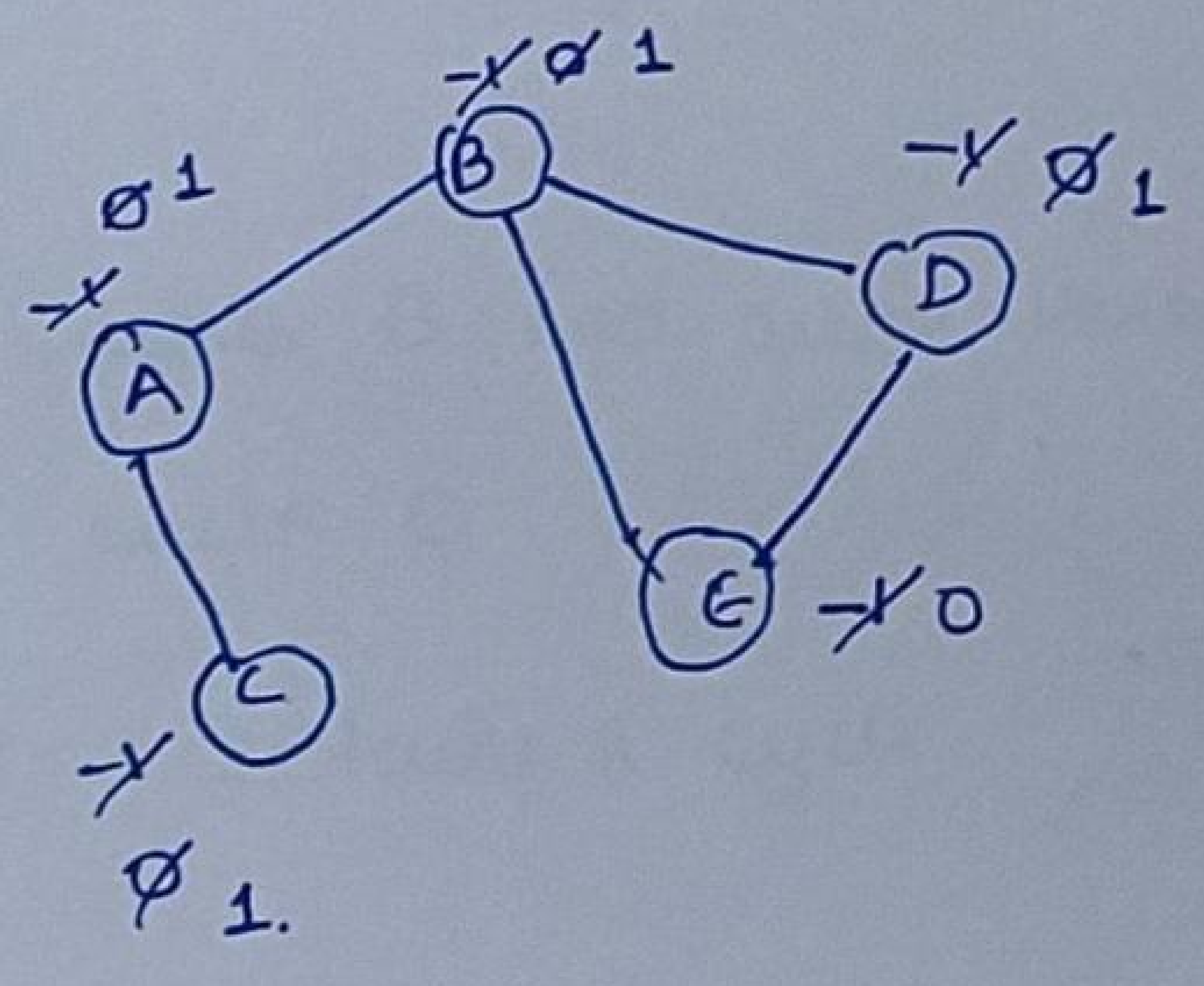
In DFS, Stack is used, as DFS uses backtracking. For DFS, we retrive it from root to the farthest node as much as possible, this is the same idea as LIFO [used by Stack].

Answer-3:- Dense graph is a graph in which the number of edges is close to the maximal number of edges.

Sparse graph is a graph in which the number of edges is close to the minimal number of edges. It can be disconnected graph.

* Adjacency lists are prefered for Sparse graph. and, adjacency matrix for dense graph.

wer-4:- Cycle detection in undirected graph (BFS)



-1 = unvisited

0 = into the queue. (node)

1 = traversed.

Queue:

A	B	C	D	E

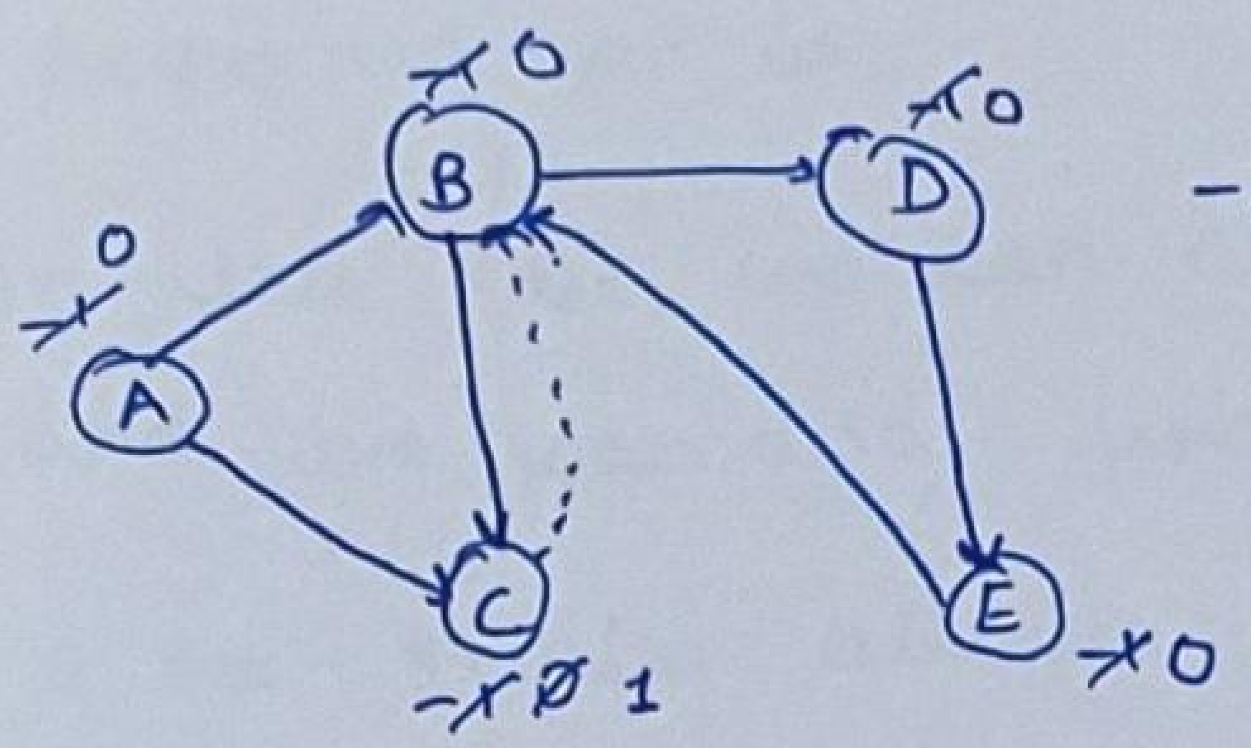
visited set :

A	B	C	D

when D checks it adjacent vertices it finds E with 0.

→ If any vertex finds the adjacent vertex with flag 0, then it contains a cycle.

Cycle detection in Directed Graph (DFS).

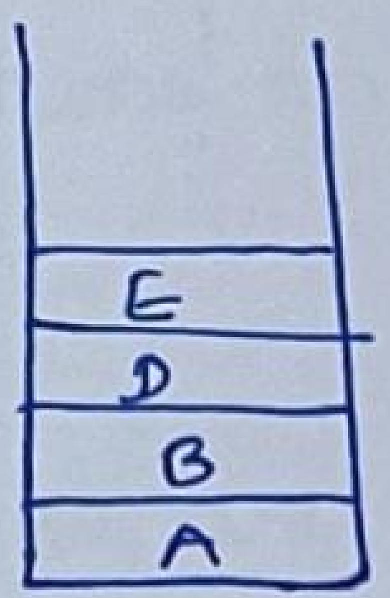


-1 = unvisited

0 = visited & in stack

1 = visited & popped out from stack

Stack:



visited set:

ABCDE

⇒ B → D → E → B

Parent Map

Vertex	parent
A	—
B	A
C	B
D	B
E	D

here, E finds B (adjacent vertex of E) with 0.

⇒ it contains a cycle.

Answer 5 → The disjoint set data structure is also known as union-find data structure and merge-find set. It is a data structure that contains a colln of disjoint or non-overlapping sets.

The disjoint set means that when the set is partitioned into the disjoint subsets, various operation can be performed on it.

In this case, we can add new sets, we can merge the sets, and we can also finds the representative member of a set. It also allows to find out whether the two elements are in the same set or not efficiently.

Operations on disjoint sets :-

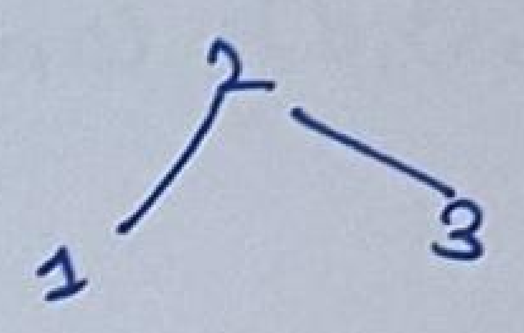
1. <u>Union</u>

a) If S1 & S2 are two disjoint sets, their union $S_1 \cup S_2$ is a set of all elements x such that x is in either S1 or S2.

b) As the sets should be disjoint $S_1 \cup S_2$ replaces S1 & S2 which no longer exists.

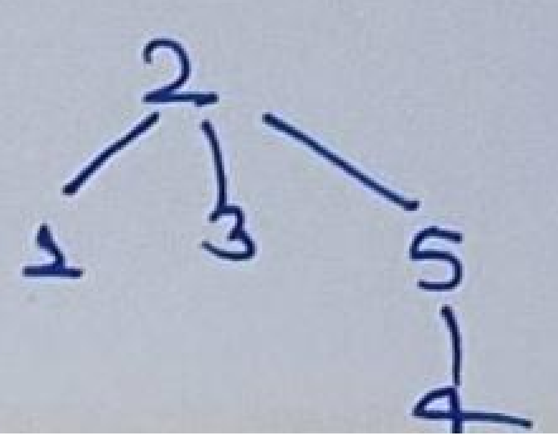
c) Union is achieved by simply making one of the trees as a subtree of other i.e. to set parent field of one of the roots of the trees to other tree.

ex- S_1 S_2



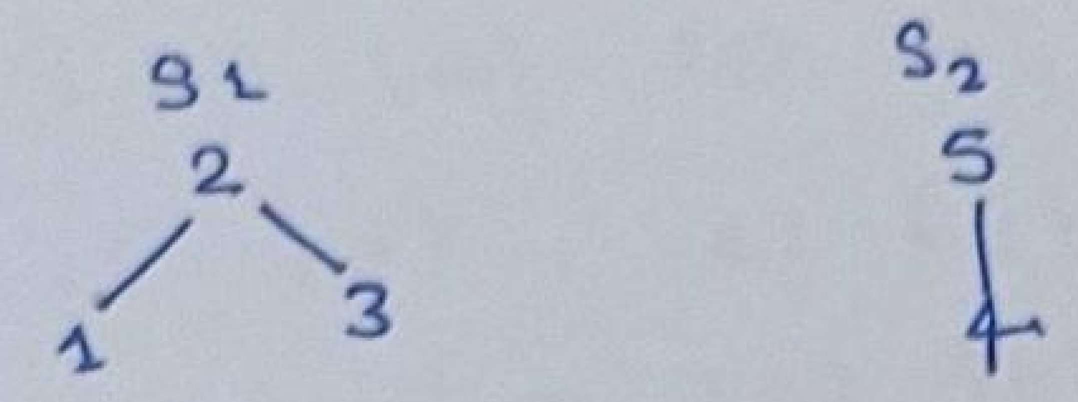
Merge the sets containing x and containing y into one.

$S_1 \cup S_2$



2. Find :- Given an element x, to find the set containing it :

ex -

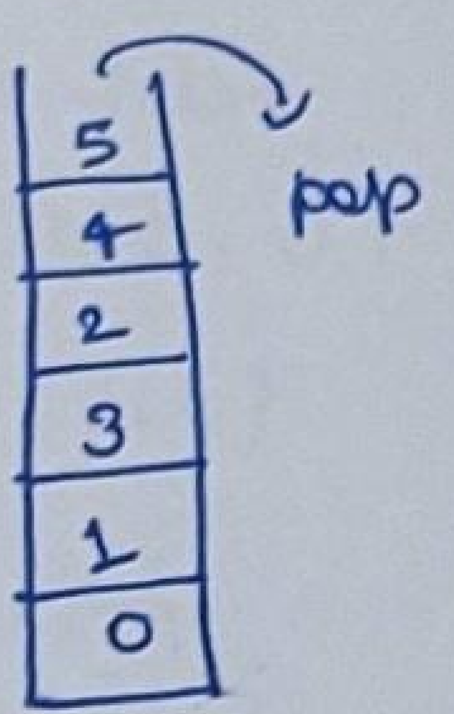


find (3) = S1
find (5) = S2

return in which set x belongs.

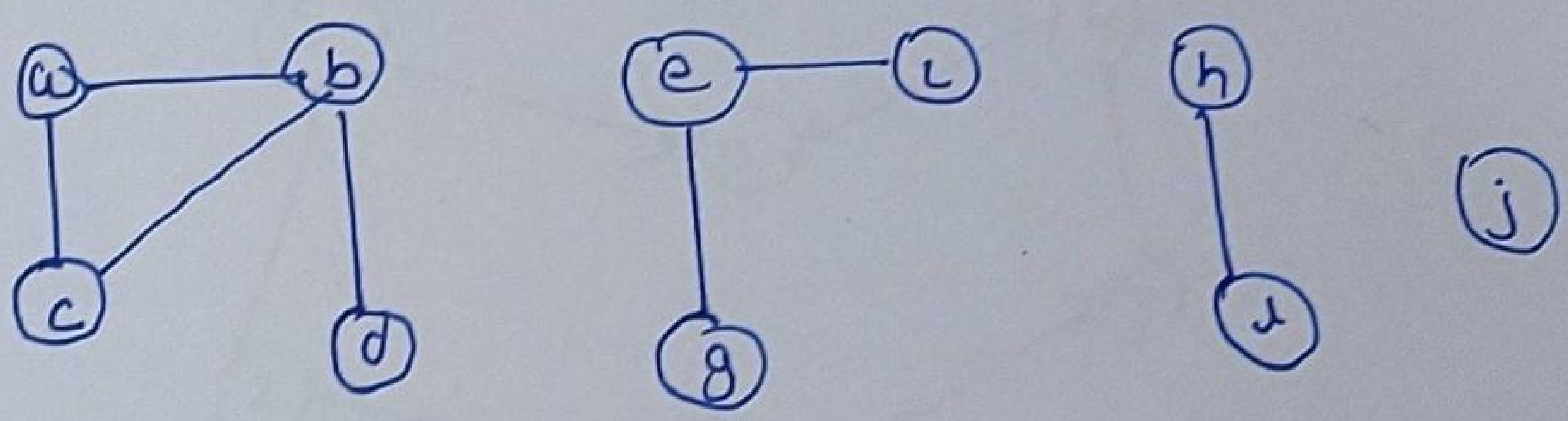
3. Make-set (x): Create a set containing x.

Answer-6 Go to node 5, all its adjacent nodes are already visited so push node 5 into the stack and mark it visited.



5 4 2 3 1 0
(output)

Answer-7:



$V = \{a, b, c, d, e, g, h, i, j, l\}$

$E = \{(a,b), (a,c), (b,c), (b,d), (e,i), (e,g), (h,l), (j)\}$

	{a}	{b}	{c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}
(a,b)	{a,b}	{c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}	
(a,c)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}		
(b,c)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{j}	{l}		
(b,d)	{a,b,c,d}	{e}	{g}	{h}	{i}	{j}	{l}			
(e,i)	{a,b,c,d}	{e,i}	{g}	{h}	{j}	{l}				
(e,g)	{a,b,c,d}	{e,i,g}	{h}	{j}	{l}					
(h,l)	{a,b,c,d}	{e,i,g}	{h,l}	{j}						
(j)	{a,b,c,d}	{e,i,g}	{h,l}	{j}						

We have.

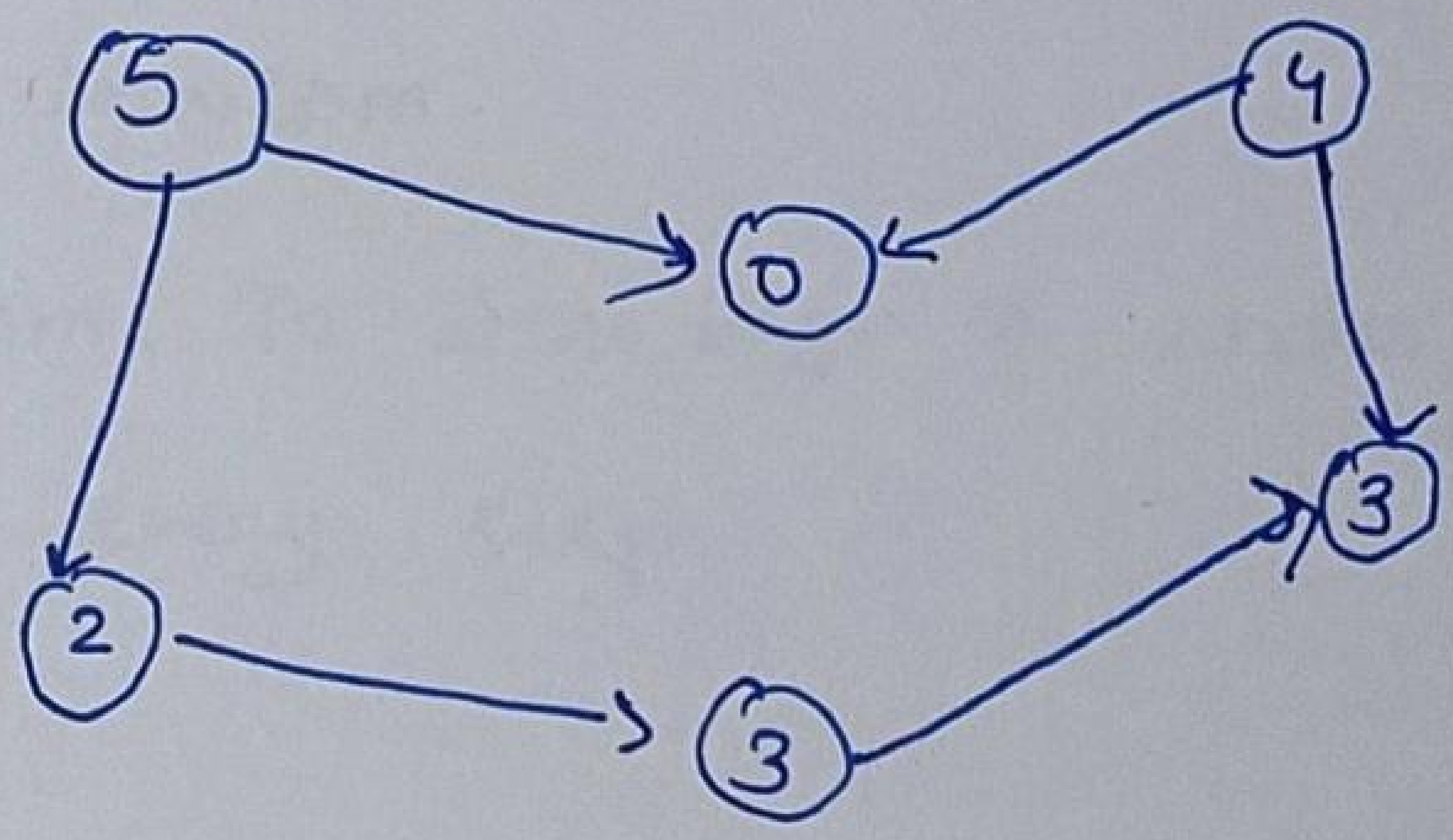
{a,b,c,d}

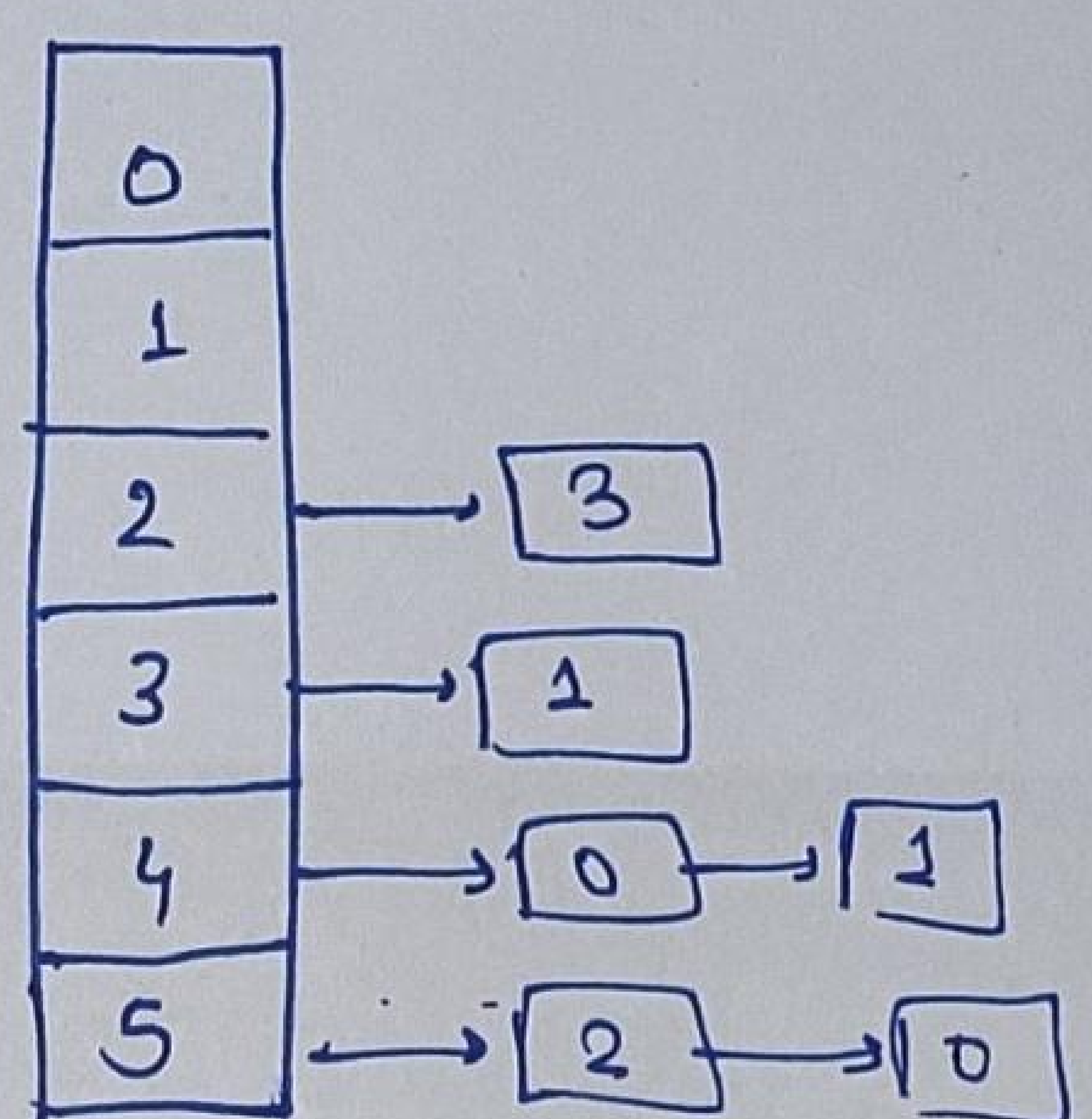
{e,i,g}

{h,l}

{j}

nswer-8 -





Algo -

1. Go to node 0, it has no outgoing edges so push node 0 into the stack and mark it visited

2. Go to node 1, again it has no outgoing edges, so push node 1 into the stack and mark it visited

3. Go to node 2, process all the adjacent nodes and mark node 2 visited.

4. Node 3 is already visited so continue with next node.

5. Go to node 4, all its adjacent nodes are already visited, so push node 4 into the stack and mark it visited.

Answer-9 — Heap is generally preferred for priority queue implementation because heap provide better performance compared to arrays or linked list.

Algorithm's where priority queue is used.

1 - Dijkstra's shortest path algorithm: when the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.

2. Prim's algorithm: To store keys of nodes, & extract minimum key node at every step

Answer-10

Min heap	Max heap.
(1) For every pair of the parent & descendant child node, the parent node always has lower value than descended child node.	(1) For every pair of the parent and descendant child node, the parent node has greatest value than descended child node.
(2) The value of nodes inc. as we traverse from root to leaf node.	(2) The value of nodes decreases as we traverse from root to leaf node.
(3) Root node has the lowest value.	(3) The root node has the greatest value.

2