

Answer 1) →

BFS

- 1) uses queue data structure.
- 2) Starts from breadth first search.
- 3) Can be used to find single source shortest path in an unweighted graph and we search a vertex with max. no. of edges from a source vertex.
- 4) Allings are visited before the children.

Applications:

- 1) Shortest path and minimum spanning tree from unweighted graph.
- 2) Peer to Peer Networks.
- 3) Social Networking websites.
- 4) GPS Navigation Systems.

DFS

- 1) uses stack data structure.
- 2) Starts from Depth First search.
- 3) we might traverse through more edges to reach a destination vertex from a source.
- 4) children are visited before the siblings.

Applications:

- 1) Detecting Cycle in a graph.
- 2) Path finding.
- 3) Topological Sorting.
- 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 retrieve it from root to the farthest.

node as much as possible this is the same idea as LIFO

[used by stack].

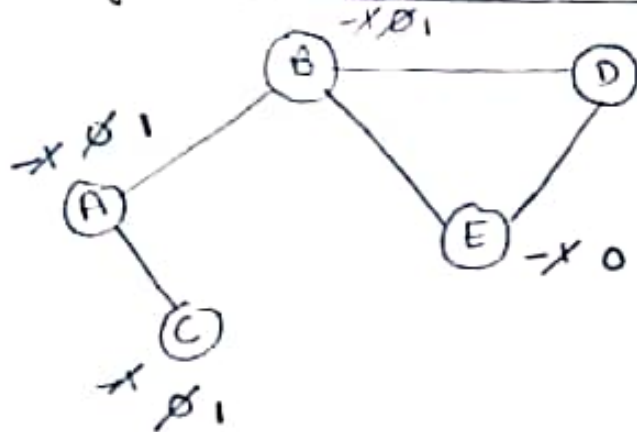
Answer 3) → Dense Graph is a graph in which the NO. of edges is close to the maximal NO. of edges.

Sparse Graph is a graph in which the NO. of edges is close to the minimal NO. of edges. It can be disconnected Graph.

* Adjacency lists are preferred for sparse Graph and

* Adjacency matrix are preferred for dense Graph.

Answer 4) → Cycle Detection in Undirected Graph (BFS):



-1 = Unvisited.

0 = into the queue (Node)

1 = Transversed.

Queue:

A	B	C	D	E
---	---	---	---	---

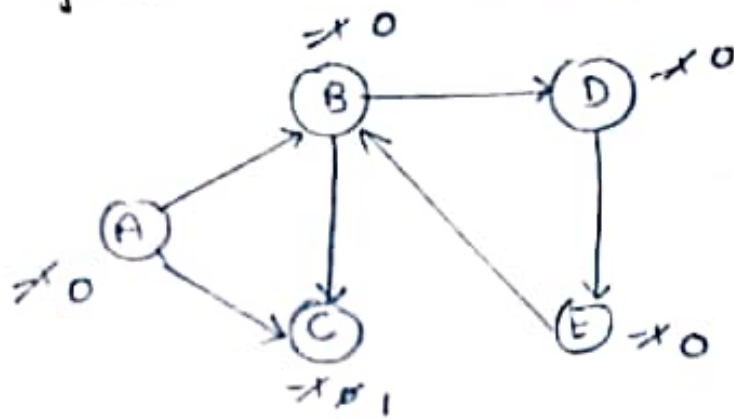
Visited set:

A	B	C	D	
---	---	---	---	--

When D checks its adjacent vertices it finds E with 0.

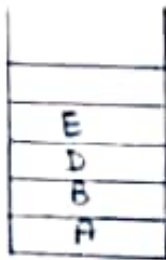
⇒ If any vertex finds the adjacent vertex with flag 0, then it contains cycle.

Cycle Detection in Directed Graph (DFS).



-1 = unvisited
 0 = visited and 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 collection of disjoint set means that when the set is partitioned into the disjoint subsets, various operation can be performed on it.

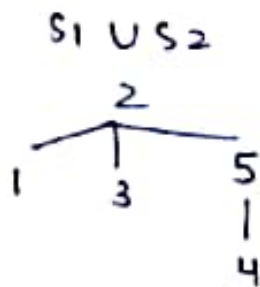
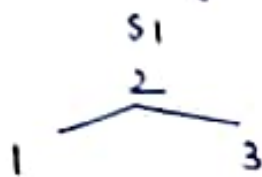
In this case, we can add new sets, we can merge the sets, and we can also find the representative member of a set. It also allows to find out whether the two elements are in the same set or not efficiently.

Operations on disjoint Set:

1.) union:

- a) If S_1 and S_2 are two disjoint sets, their union $S_1 \cup S_2$ is a set of all elements x such that x is in either S_1 or S_2 .
- b) As the sets should be disjoint $S_1 \cup S_2$ replaces S_1 and S_2 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 root.

Ex:

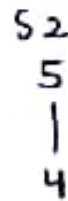
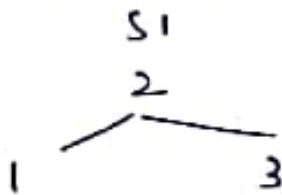


Merge the sets containing x and containing y into one.

2.) Find:

Given an element x , to find the set containing it.

Ex:



$\text{find}(3) \Rightarrow S_1$

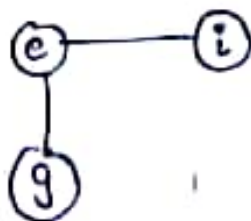
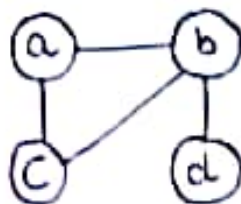
$\text{find}(5) \Rightarrow S_2$

return in which set x belongs

3.) Make-set (x):

Create a set containing x .

Answer 7)



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

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

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

we have ,

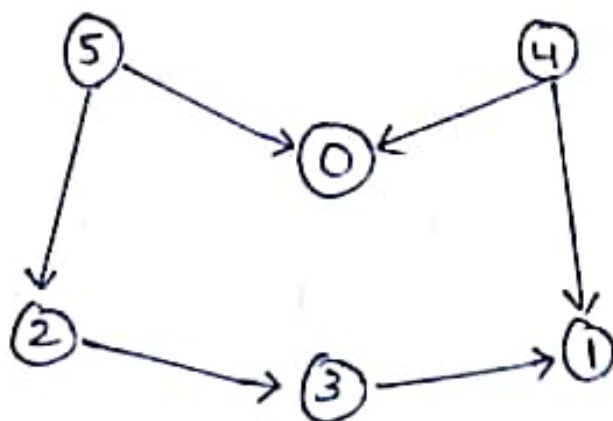
$$\{a, b, c, d\}$$

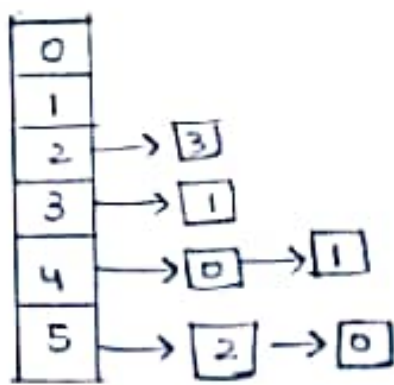
$$\{e, i, g\}$$

$$\{h, l\}$$

$$\{j\}.$$

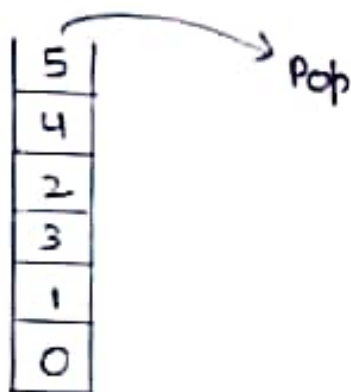
Answer 8)





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 ready and already visited so continuous with next Node.
- 5.) Go to node 4, all its adjacent Nodes are already visited so push nodes 4 into the stack and mark it visited.
- 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 9) → Heap is generally preferred for priority queue implementation because Heaps provide better performance compared to Arrays or linked List.

Algorithms 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 and extract minimum key node at every step.

Answer 10) →

Min Heap	Max Heap.
<ol style="list-style-type: none">1.) For every pair of the parent and descendant child node, the parent node always has lower value than descendant child node.2.) The value of nodes increase as we traverse from root to leaf node.3.) Root node has the lowest value.	<ol style="list-style-type: none">1.) For every pair of the parent and descendant child node, the parent node has greater value than descendant child node.2.) The value of nodes decreases as we traverse from root to leaf node.3.) The Root Node Has the Greatest Value.