Tutorial-5

Q1.

BFS (Breadth-first search)

* Uses queue data structure
* Can be used to single source shortest path in an unweighted graph & we reach a vertex with min. no. of edges from source vertex.
* Siblings are visited before children
* Applications:
  - Shortest path & minimum spanning tree for unweighted graph.
  - Peer to peer networks.
  - Social Networking Website
  - GPS Navigation System

DFS (Depth-First Search)

* Uses stack data structure
* We might traverse through more edges to reach a destination vertex from a source
* Children are visited before sibling.
* Applications:
  - Detecting cycles in a graph
  - Path findings.
  - Solving puzzles with only one solution.

Q2. In BFS, we use queue as queue is used when things don't have to be processed immediately, but have to be processed in FIFO order like BFS.

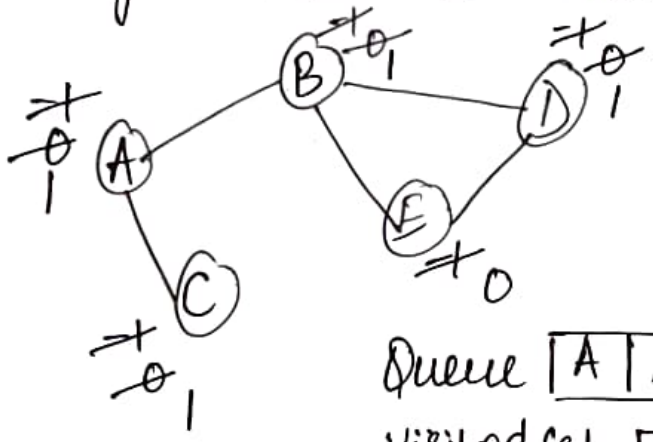
In DFS, stack is used as DFS using backtracking. For DFS, we retrieve it from root to the farthest node as much as possible, this is the same idea as LIFO.

Q3. Dense graph is a graph in which the no. of edges is close to maximal no. of edges.

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

* Adjacency lists are preferred for sparse graph & adjacency matrix for dense graph.

Q4. Cycle Detection in undirected graph (BFS)



-1 = unvisited
0 = into queue (node)
1 = traversed

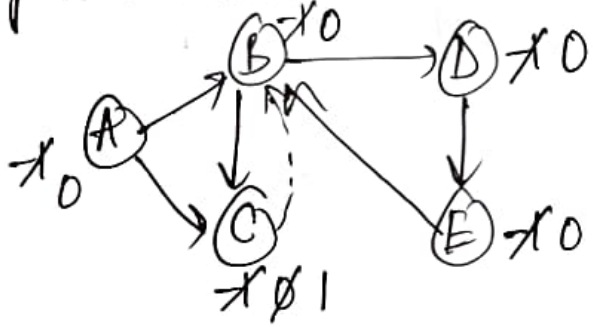
Queue | A | B | C | D | E |

Visited set | A | B | C | D |

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

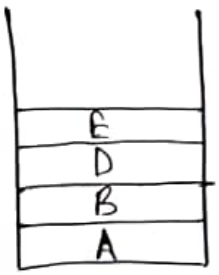
⇒ if any vertex finds adjacent vertex with flag = 0, then it contains cycle.

Cycle Detection in Directed Graph (DFS)



-1 = unvisited
0 = visited & in stack.
1 = visited & popped out of stack.

Stack:



Visited set : A B C D E.

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 cycle

Q5. Disjoint set data structure is also known as union-find data structure & merge-find set. It is a data structure that contains a collection of disjoint or non-overlapping sets. The disjoint sets means that when set is partioned into disjoints subsets various operation can be performed on it.

Operations on Disjoint Set

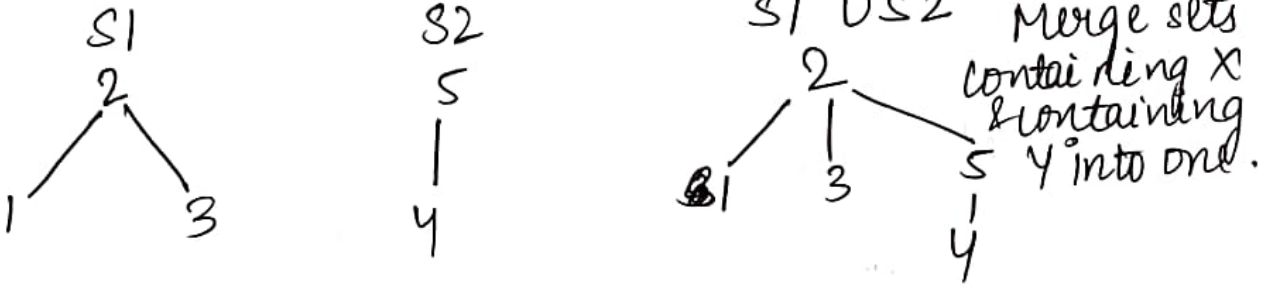
1.] Union

a.] If S1 & S2 are 2 disjoint sets, their union S1 ∪ S2 is a set of all elements 'x' such that x is in either S1 or S2.

b.] As sets should be disjoint S1 ∪ S2 replaces S1 & S2 which no longer exists.

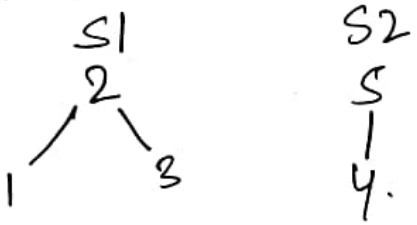
c.] Union is achieved simply making one of trees as subtree of other, i.e, to set parent field of one of roots of trees to other root.

ex:



2. Find.

Given an element x, to find set containing it.



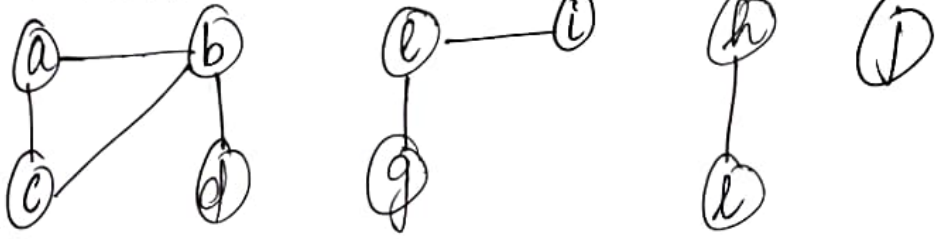
find (3) ⇒ S1

find (5) ⇒ S2

Return in which set x belongs.

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

Q7.



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}				
(g,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,

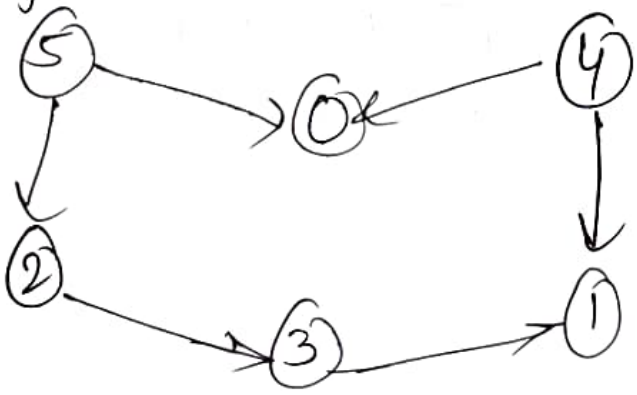
{a,b,c,d}

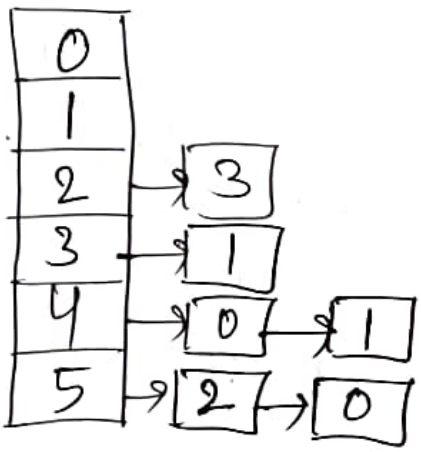
{e,i,g}

{h,l}

{j}

Q8





Algo:

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

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

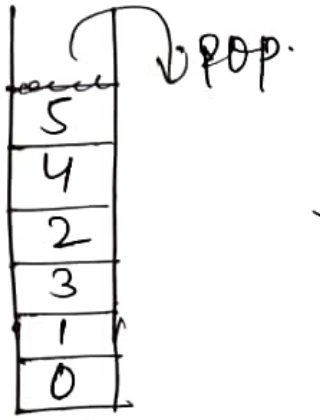
3.] Go to node 2, process all adjacent nodes & mark node 2 visited

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 stack & mark it visited.

6.] Go to node 5, all its adjacent nodes are already visited so push node 5 into stack & mark it visited.

⇒



5 4 2 3 1 0

(output).

Q9. Heap is generally preferred for priority queue implementation because heaps provide better performance compared to arrays or linked list.

Algorithms where priority queues is used:-

1.] Dijkstra's Shortest Path Algorithm: When graph is stored in form of adjacency list or matrix, priority queues can be used to extract minimum efficiency when implementing Dijkstra's algorithm.

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

Q10.

Min Heap.

1.] For every pair of parent & descendant child node, the parent node always has lower value than descendent child node.

2.] Value of node increase as we traverse from root to leaf node.

3.] Root node has lowest value.

Max. Heap

1.] For every pair of parent & descendent child node. the parent node has greater value than descended child node.

2.] Value of nodes decrease as we traverse from root to leaf node.

3.] Root node has greatest value.