Tutorial-5

Ans 1

	BFS	DFS
i)	Uses queue data structrue	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 min. no. of edges from a source vertext	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.

BFS Applications:

1. Shortest Path & Minimum Spanning Tree for unweighted graph.
2. Peer to Peer Networks
3. Social Networking Websites
4. GPS Navigation Systems

DFS Applications:

1. Detecting cycle in a graph
2. Path finding
3. Topological Sorting
4. Solving puzzles with only one solⁿ

Ans 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.

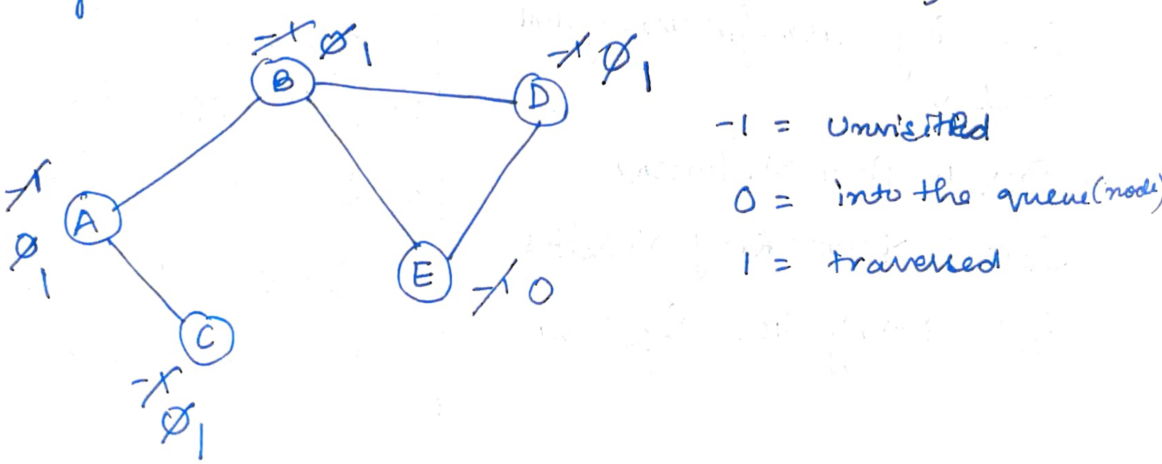
In DFS stack is used as DFS uses backtracking. For DFS, we retrieve it from root to the farthest node as much as possible, this is the same idea as LIFO [used by stack].

Ans 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 dis connected graph.

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

Ans 4 Cycle Detection in Undirected Graph (BFS)



Queue :

A	B	C	D	E

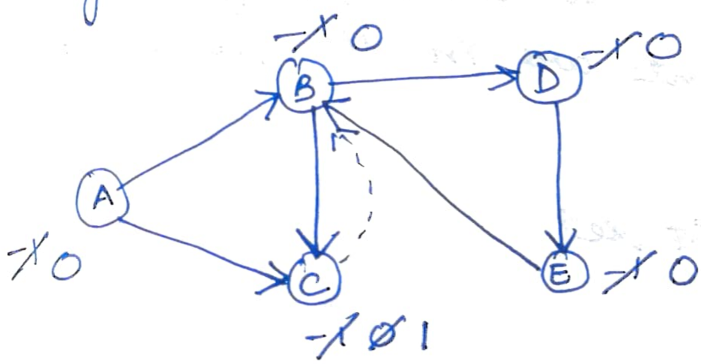
Visited Set :

A	B	C	D

When D checks it adjacent vertices it finds E with 0

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

Cycle Detection in Directed Graph (DFS)



-1 = unvisited

0 = visited & in stack

1 = visited & popped out from stack

Stack :

E
D
B
A

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

Ans 5

The disjoint set data structure is also known as union-find data structure & merge-find set. It is a data structure that contains a coll^n of disjoint or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets, various op^n can be performed on it.

In this case, we can add new sets, we can merge the sets, & 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

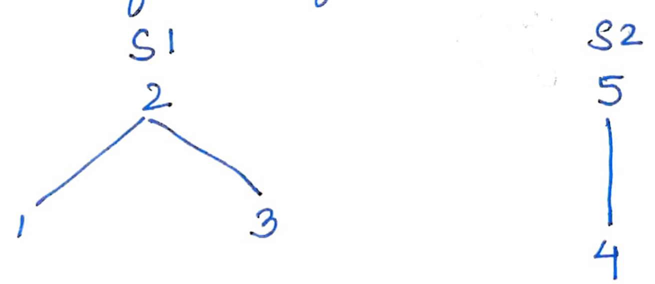
1. Union

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

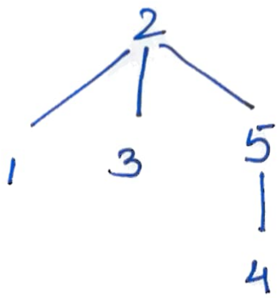
b) As the sets should be disjoint S1 U S2 replaces S1 & S2 which no longer exists.

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 root.

Ex:



$S1 \cup S2$

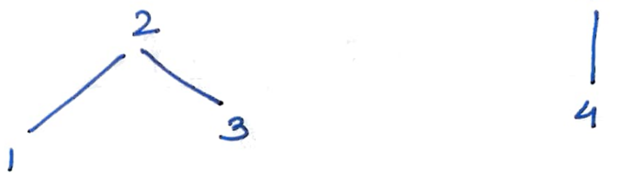


Merg the sets containing X & containing Y into one

2. Find

Given an element X, to find the set containing it

Ex: S1 S2



find(3) ⇒ S1

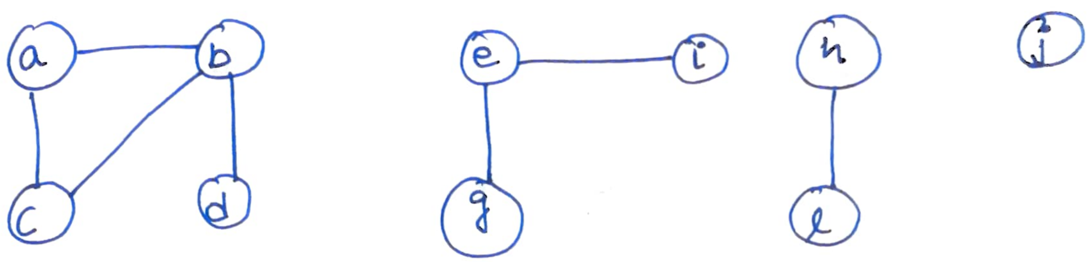
find(5) ⇒ S2

return in which set X belongs

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

~~Ans 6~~

Ans 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

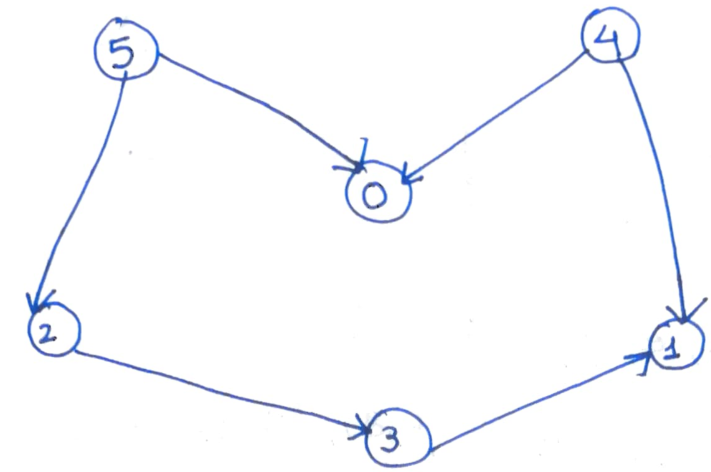
{a, b, c, d}

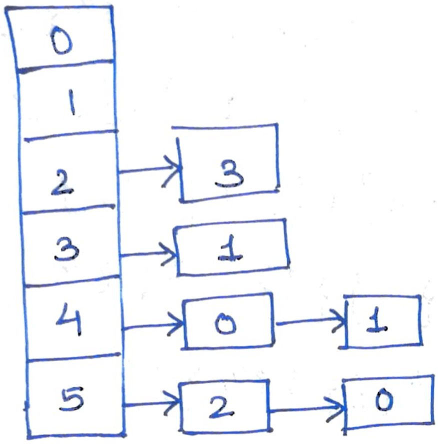
{e, i, g}

{h, l}

{j}

Ans 8

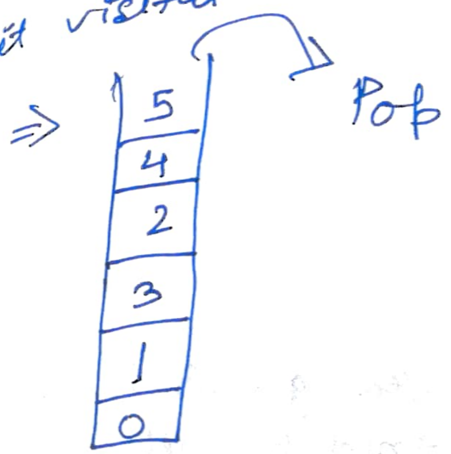




Algo:

1. Go to node 0, it has no outgoing edges so push node 0 into the stack & mark it visited.
2. Go to node 1, again it has no outgoing edges, so push node 1 into the stack & mark it visited.
3. Go to node 2, process all the 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 the stack & mark it visited.

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



5 4 2 3 1 0

(output)

Ans 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 & extract minimum key node at every step.

Ans 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 & descendant child node, the parent node has greater 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.