Tutorial-5

Ans-1

BFS	DFS
1) Uses queue data structure.	1) Uses stack data structure.
2) Stands for breadth first search	2) Stands for depth first search.
3) Can be used to find single source shortest path in an unweighted graph, as we reach a vertex with min no. of edges from a source vertex.	3) We might transverse through more edges to reach a destination vertex from a source.
4) Siblings are visited before the children.	4) Children are visited before the siblings.
Application :-	Applications.
1) Shortest path & Minimum spanning tree for unweighted graph.	1) Detecting cycle in a graph.
2) Peer to Peer Network.	2) Path finding
3) Social Networking websites	3) Topological sorting
4) GPS Navigation systems.	4) Solving puzzles with only one soln.

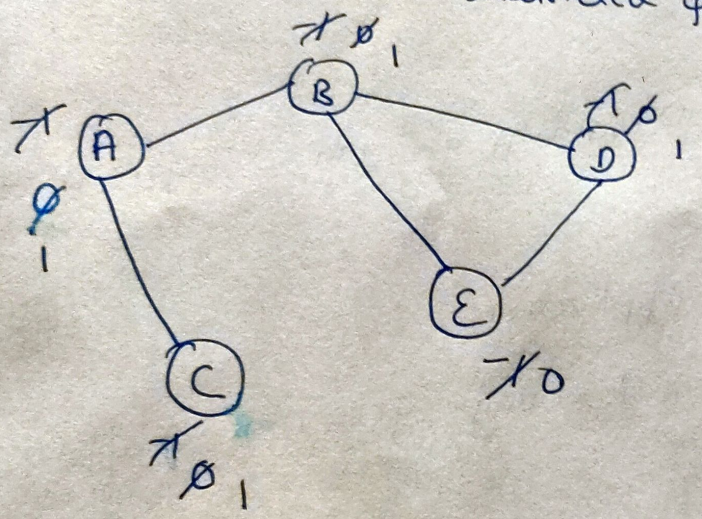
Ans-2 In BFS we use queue data structure as queue is used when things don't have to be processed immediately, but have to be processed in FIFO order like BFS.

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 disconnected graph.

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

Ans-4 cycle detection in undirected graph (BFS)



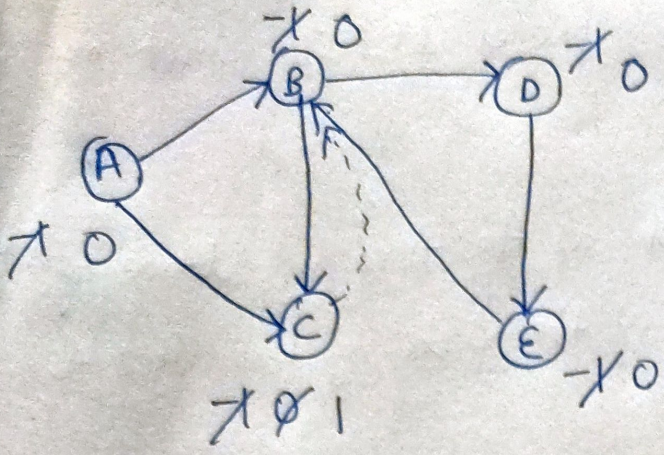
Queue :- | A | B | C | D | E |

Visited set :- | A | B | C | D |

when D checks it adjacent vertics it finds E with 0

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

cycle detection in directed graph (DFS)



stack :-

E
D
B
A

visited set
A B C D E

Parent Map

Vertex	Parent
A	—
B	A
C	B
D	B
E	D

$\Rightarrow B \rightarrow D \rightarrow E \rightarrow B$

there 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 soln. of disjoint or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets, various op". can be performed on it.

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

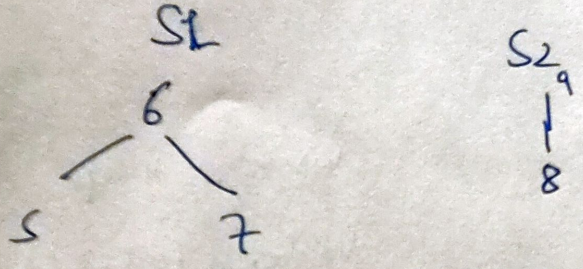
operations on disjoint set :-

1) Union

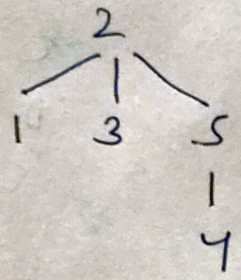
as of S1 & S2 are two disjoint sets, their union S1 ∪ S2 is a set of all elements x each that x is in either S1 or S2.

- As the sets should be disjoint S1 ∪ S2 replaces S1 & S2 which no longer exists.
- Union is achieved by simply making one of the trees as subtree of other ie.. to set parent field of one of the roots of the trees to other not.

Ex.

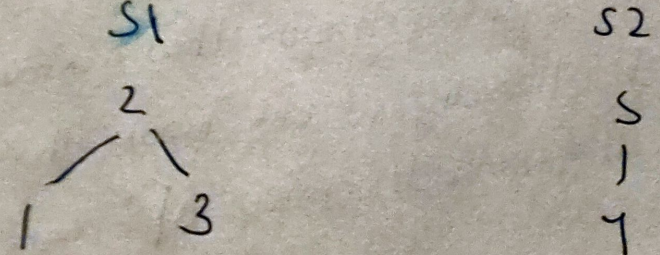


S1 ∪ S2



2) Find

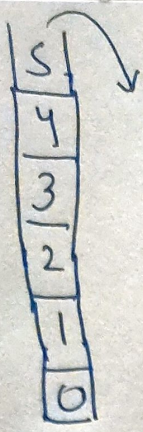
Given an element x, to find the set containing it Ex:-



find(3) ⇒ S1

find(5) -> S2

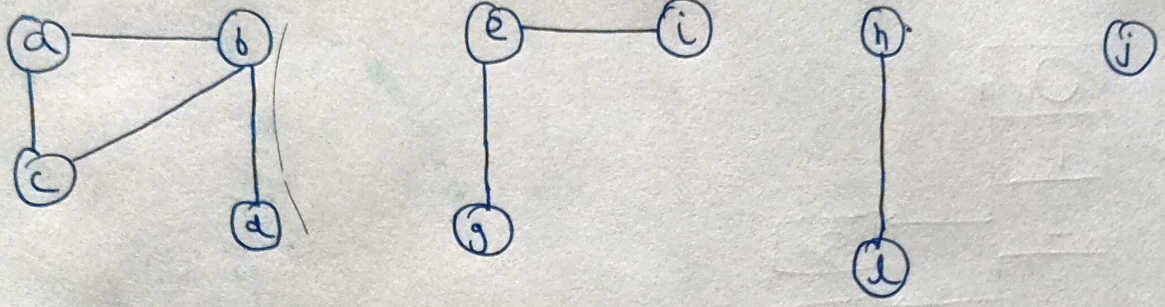
Ans 6 No to node S, all the adjacent node are already visited so push node S into the stack & mark it visit.



5 4 2 3 1 0

output.

Ans 7



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

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

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

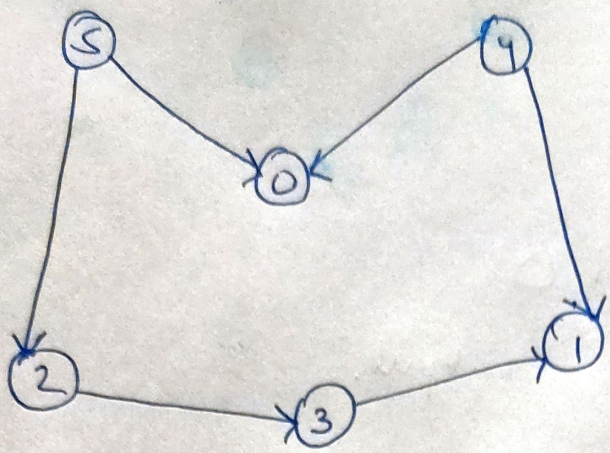
{a,b,c,d}

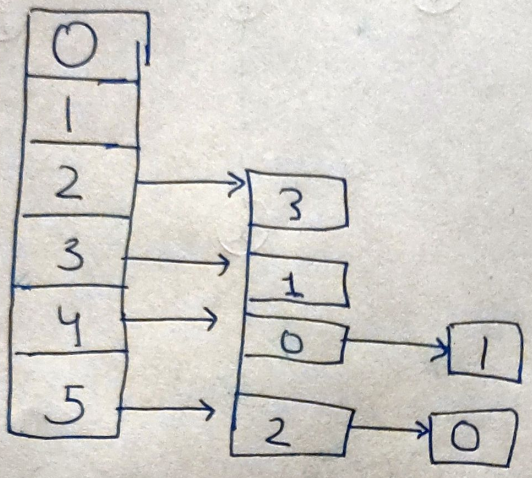
{e,i,g}

{h,l}

{j}

Ans 9





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

Algorithm 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 Algo.

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 descendant child node.	1) for every pair of the parent & descendant child node, the parent node has greater value than descendant child node.
2) The value of nodes increase as we traverse from root to leaf node.	2) The value of nodes decrease as we traverse from root to leaf node
3) Root node has the lowest value	3) Root node has the greatest value.