TUTORIAL 5

Ans 1)

BFS	DFS
1. Uses queue data structure	Uses stack data structure
2. Stands for Breadth First Search	Stands for Depth First Search
3. Can be used to find single source shortest path in an unweighted graph and we reach a vetex with min. no. of edges from a source vertex	We might traverse through more edges to reach a destination vertex from source
4. ~~Simblings~~ Siblings are visited before the children	Children are visited before the siblings
Applications	**Applications**
• Shortest path and minimum spanning tree are unweighted graph	• Detecting cycle in graph
• Peer to Peer Networks	• Path finding
• Social Networking Websites	• Topological sorting
• GPS Navigation Systems.	• Solving puzzle with only one solution

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.

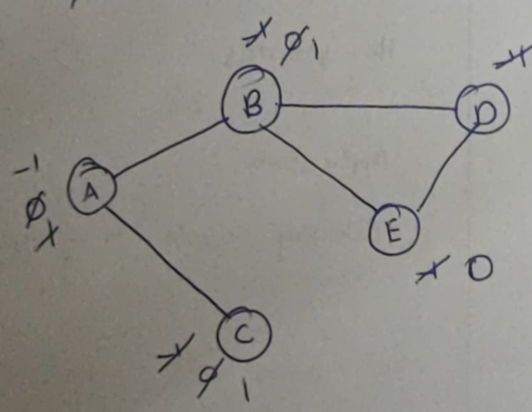
In DFS stack is used as DFS uses backtracking. For DFS, we retrieve it from root to the farthest node as much as possible, this is the same idea as LIFO [used by stack]

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 preffered for sparse graph and Adjacency matrix for dense graph.

Ans 4) Cycle detection in undirected graph (BFS)



-1 = unvisited

0 = into the queue (node)

1 = traversed

Queue :

A	B	C	D	E

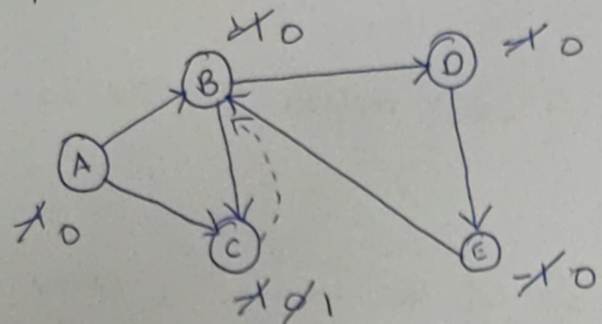
Visited Set :

A	B	C	D	

When D checks 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

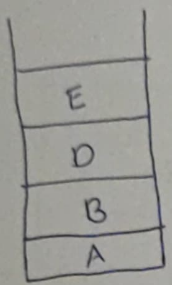
0 = visited and in stack

1 = visited and popped out from stack

Parent Map

Vertex	Parent
A	–
B	A
C	B
D	B
E	D

Stack :



Visited set :

AB CD E

⇒ B → D → E → B

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 and merge-find set. It is a data structure that contains a collection of disjoint or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets, various operations 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 :

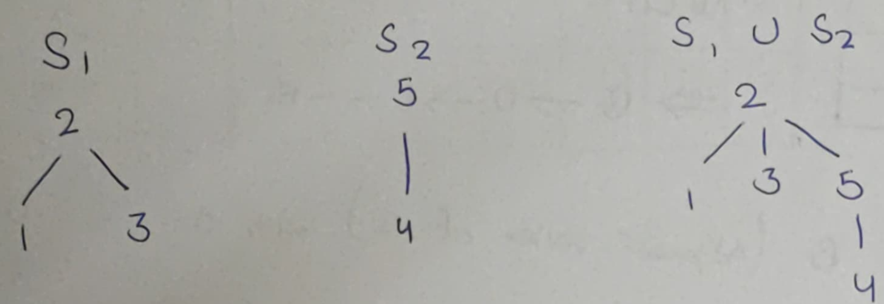
(a) If S_1 and S_2 are two disjoint sets, their union $S_1 \cup S_2$

is a set of all elements X such that X is in either S1 or S2

(b) As the sets should be disjoint S1 ∪ S2 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 roots.

Ex:

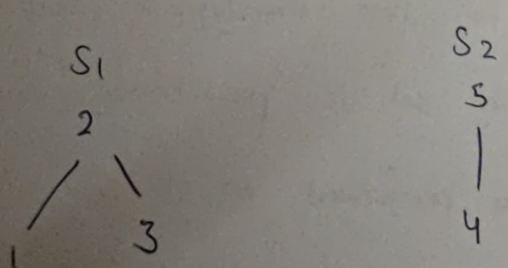


Merge the sets containing X and containing Y into one

2. Find

Given an element X, to find the set containing it

Ex:



find (3) ⇒ S_1 return in which set x belongs

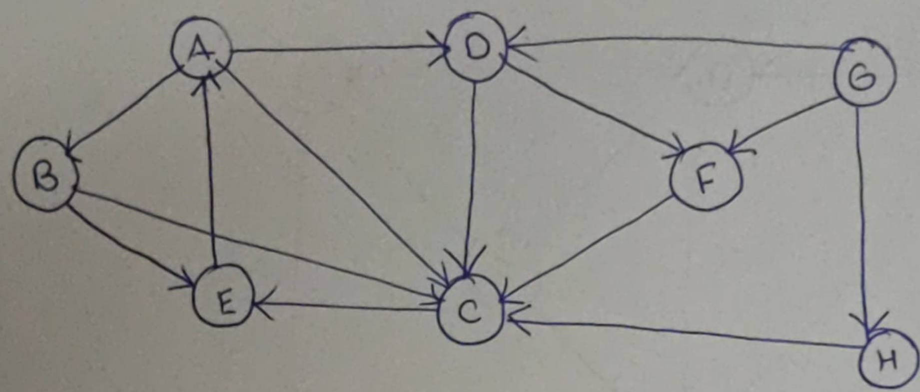
find (5) ⇒ S_2

3. Make-Set (x):

Create a set containing X

makeset (1) = {1}

Ans 6)



BFS:

Child	G	D	F	H	C	E	A	B
Parent		G	G	G	H	C	E	A

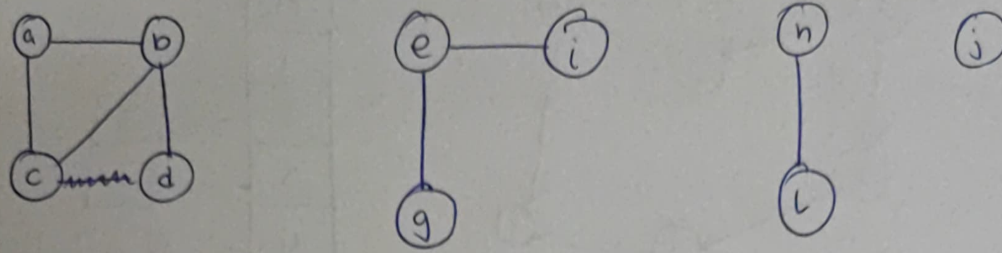
Path: G → H → C → E → A → B

DFS:

Node visited	Stack
~~G~~	G
D	F
~~E~~ H	C
~~H~~ ~~F~~	E
C	A
~~F~~	B
~~A~~	
~~B~~	

Path: G → F → C → E → A → B

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

We have

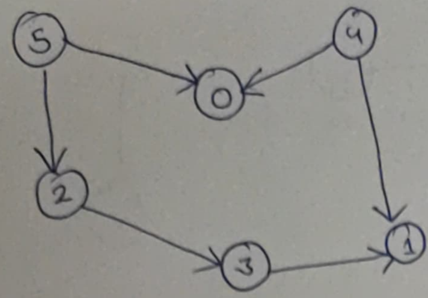
{a, b, c, d}

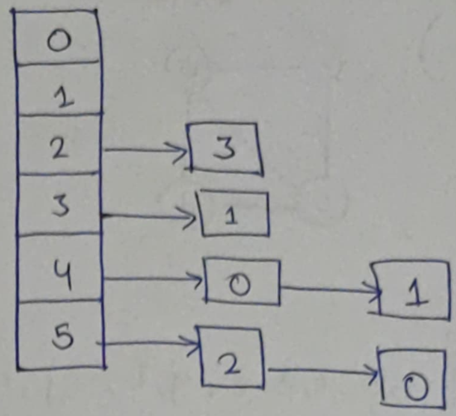
{e, i, g}

{h, l}

{j}

Ans 8)





Algo:

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

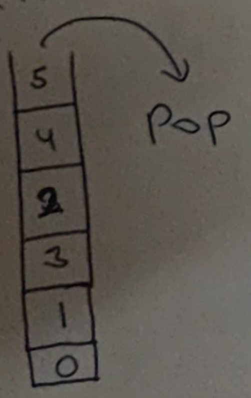
2. Go to node 1, again it has no outgoing edges so push node 1 into the stack and mark it visited

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

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 3 2~~

5 4 2 3 1 0

(output

Ans 9) Heap is generally preffered for priority queue implementation because heaps provide better performance compared to arrays or linked lists.

Algorithms where priority queue is used:

1. Djikstra's ~~algorithm~~ 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 Djikstra's algorithm

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

Ans 10)

Min Heap	Max Heap
1. For every path of the parent and descendant child node, the parent node always has lower value than descendant child node.	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 increases as we traverse from root to leaf node	The value of node decreases as we traverse from root to leaf node
3. Root node has the lowest value	Root node has the greatest value