Tutorial-5

Q1

Ans

BFS	DFS
1. BFS Stands for Breadth First Search.	1. DFS Stands for Depth First Search.
2. BFS uses the Queue Data structure for finding the shortest path.	2. DFS uses Stack data structure.
3. In BFS Siblings are visited before the children	3. Here children are visited before the sibling.
4. Time Complexity of BFS:- for Adjacency list: $O(V+E)$ for Adjacency Matrix: $O(V^2)$	4. Time complexity of DFS is for Adjacency List: $O(V+E)$ for Adjacency matrix: $O(V^2)$
5. BFS considers all neighbours first and therefore not suitable for decision making trees used in games or puzzles.	5. DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
6. BFS is used in various Application such as Bipartite graph, and shortest path etc.	6. DFS is used in various application such as acyclic graph and topological order etc

Q2.

Ans BFS: It uses a Queue Data structure which follows FIFO.

In BFS, one vertex is selected at a time when it is visited & marked then its adjacent are visited and stored in Queue.

DFS: It uses the Stack Data Structure and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.

B.

Ans Dense Graph: A Graph in which the number of edges is close to the maximal no. of edges, or in other words, if every pair of vertices is connected by one edge.

Sparse Graph: Sparse Graph is a graph in which the number of edges is close to the minimal number of edges.

- Adjacency list is good for sparse graph representation.
- While ~~adja~~ Both Adjacency matrix and adjacency list can be used for Dense graph representation.

Or

Ans Algorithm to Detect a cycle in the Graph using DFS ~~BFS~~

(i) Create the graph using the given no. of edges and vertices.
(ii) Create a recursive function that initializes the current index or vertex, visited and recursion stack.
(iii) Mark the current node as visited and also mark the Index in recursion stack.
(iv) Find all vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices, If the recursive function returns true, return true.
(v) If the adjacent vertices are already marked in the recursion stack then return true.
(vi) Create a wrapper class, that calls the recursive function for all the vertices and if any function returns true return true. Else if for all vertices the function returns false return false.

• Algorithm for DFS cycle detection:

Step1: Compute in Degree for each of the vertex present in the graph and initialize the count of visited nodes as 0.

Step2: Pick all the vertices with in degree as 0 and add them into a queue.

Step3: Remove a vertex from the queue. and then.

(i) Increment count of visited nodes by 1.

(ii) Decrease in degree by 1 for all its neighbouring nodes.

(iii) If in-degree of a neighbouring nodes is reduced to zero, then add it to the queue.

Step4: Repeat step 3 untill the queue is empty.

Step5: If count of visited nodes is not equal to the no. of nodes in the graph has cycle, otherwise not.

Q5.

Ans Disjoint Set:- A Disjoint Set maintains a collection of disjoint dynamic sets.

$S = \{ S_1, S_2, \ldots, S_k \}$

$S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$

- Identify each set by a representative which is same number of the set.
- Ask for representative of a dynamic set twice without modifying the set:- you will get same answer.

Operations:-

(1) Make Set (x): Create a new set whose only member & thus representative is x.

S_1 = make-Set (1) = {1}, 1

S_2 = make-set (2) = {2}, 2

(2) Union (x, y): Unites the dynamic sets that contains x & y say S_x & S_y into a new set that is union of two sets.

- choose a new representative.

S = Union $(1, 2) = \{1, 2\}$

New representative $= 1$

(3) find-Set (x): Returns a pointer to the representative of the set containing x.

find set$(2) = 1$

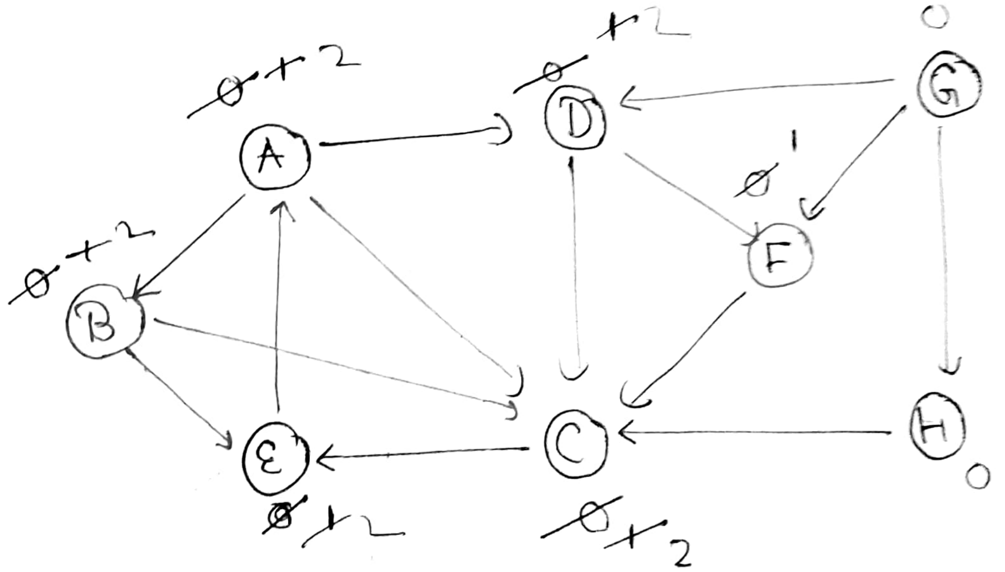
Ex: $S_1 = \{1, 2, 3\} \rightarrow 1$

$S_2 = \{4, 5, 6\} \rightarrow 4$

findset$(3) = 1$

union $(3, 6) = \{1, 2, 3, 4, 5, 6\} \rightarrow 1$

Q6 Ans BFS:-



⇒ queue:-

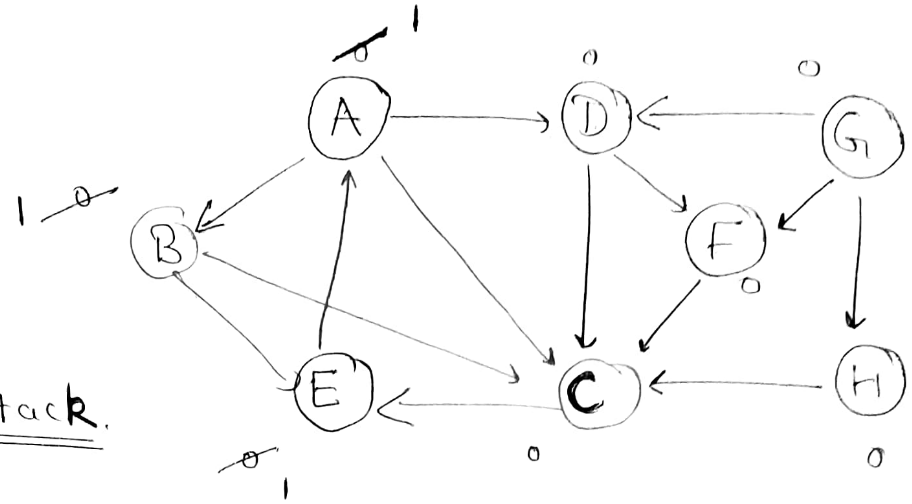
Node	~~A~~	~~B~~	~~C~~	~~D~~	~~E~~	~~F~~
Parent	–	A	A	A	B	D

Source Node: A.

~~A~~

Node processed A→B→C→D→E→F

DFS:-



Node Processed	Stack
	A
A	BCD
B	ECD
E	CD
C	D
D	F
F	

⇒ DFS: A → B → E → C → D → F.

Q.

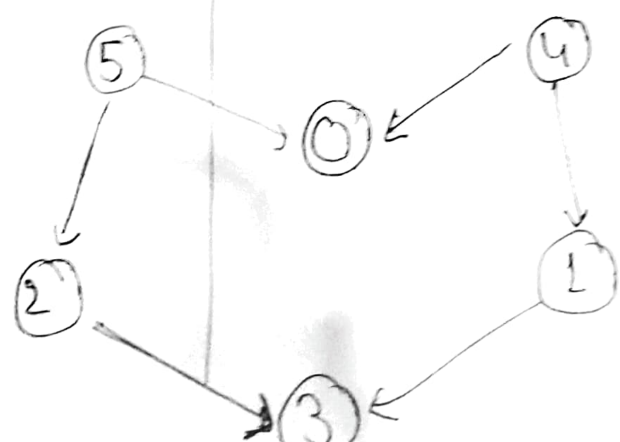
A.

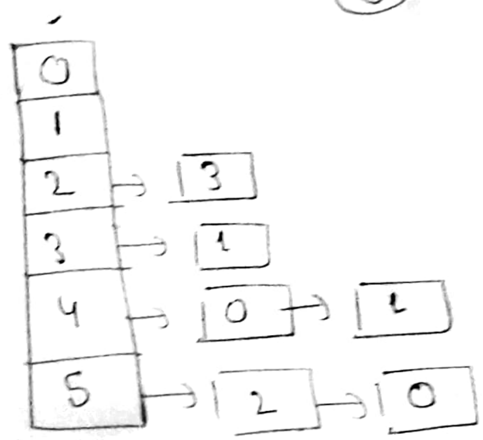
Edge Processed	Collection of Disjoint Sets.
initial Sets	{a} {b} {c} {d} {e} {f} {g} {h} {i} {j}
bd	{a} {b,d} {c} {e} {f} {g} {h} {i} {j}
eg	{a} {b,d} {c} {e,g} {f} {h} {i} {j}
ac	{a,c} {b,d} {e,g} {f} {h} {i} {j}
hi	{a,c} {b,d} {e,g} {f} {h,i} {j}
ab	{a,b,c,d} {e,g} {f} {h,i} {j}
ef	{a,b,c,d} {e,f,g} {h,i} {j}

There are 4 Connected Components

Q8

Ans





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 & marks it as visited.

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

=> Pop

5
4
2
3
1
0

5 4 2 3 1 0

(output)

Ans 7: Heap is generally preffered for priority queue implementation because heaps provide better performance compared to arrays or linked list.

Algorithms where priority queue is used:

1. Dijkstra 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 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 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.