DESIGN AND ANALYSIS OF ALGORITHM

1. What is difference between DFS and BFS. Please write their applications.

BFS	DFS
BFS stands for Breadth first search.	1. DFS stands for depth first search.
It uses queue data structure for finding the shortest path.	2. It uses stack data structure.
BFS can be used to find single source shortest path in an unweighted graph, because in BFS we reach a vertex with minimum number of edges from a source vertex.	3. In DFS, we might traverse more edges to reach a destination vertex from a source.
4. BFS is more suitable for searching vertices which are closer to the given source	4. DFS is more suitable when there are solutions away from source.
5. BFS considers all neighbours first and therefore not suitable decision making trees used in games.	5 DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision.
6. The time complexity is $O(V+E)$ when adjacency list is used and $O(V^2)$ when adjacency matrix is used. V = vertices, E = edges	6. The time complexity is $O(V+E)$ when adjacency list is used and $O(V^2)$ when adjacency matrix is used.

7. Siblings are visited before children.	7. Children are visited before siblings.
8. There is no concept of back-tracking.	8. DFS is a recursive algorithm that uses backtr
9. BFS requires more memory.	9. Requires less memory.
APPLICATION	APPLICATION
We can detect cycles in a graph using DFS.	1. We can detect cycles.
In peer-to-peer network like bit-torrent.	2. We can perform topologi sorting schedule jobs from given dependencies.
Search engine crawlers are used in BFS to build index	3. We can find paths betw two vertices.
Used in Ford-Fulkerson algorithm.	

Q2 Which data structures are used to implement BFS and DFS and why?

BFS

BFS uses queue data structure.
It acts as a container
Queue ensures that the items which are discovered first will be explored first.

DFS

DFS uses stack data structure.
Stack always expands the deepest node in the current frontier of the search tree.
The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those node are expanded, they are dropped

from the frontier, so then the search "backs up" to the next deepest node that still has unexplored successors.

Q3. What do you mean by sparse and dense graphs? Which representation of graph is better for sparse and dense graphs?

If most of the elements of the matrix have 0 value then it is called sparse matrix.

Similarly, sparse graph is a graph in which the number of edges is much less than the possible number of edges.

Dense graph is a graph in which the number of edges is close to the maximal number of edges.

If the graph is sparse, we should store it as a list of edges i.e using adjacency list
If the graph is dense we should represent it using adjacency matrix.

Q4 How can you detect cycle using BFS and DFS?

BFS

1. Find no. of incoming edges for each of the vertex present in the graph and initialize the count of visited nodes as 0.
2. Pick all the vertices with in-degree as 0 and add them into a queue.
3. Remove a vertex from the queue (dequeue operation) and then
   (a) Increment count of visited nodes by 1.

(b) Decrease in-degree by 1 for all its neighboring nodes.

(c) If in degree of a neighboring node is reduced to zero, then add it to the queue.

4. Repeat step 3 until queue is empty.

5. If count of visited nodes is not equal to the number of nodes in the graph has cycle, otherwise not

DFS

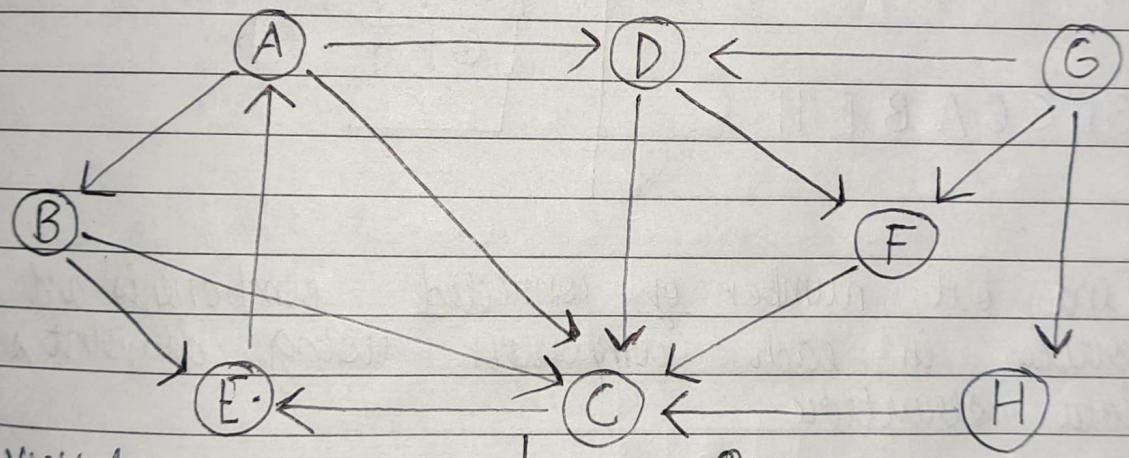
1. Create the graph using the given number of edges and vertices.
2. Create a recursive function that have current index or vertex, visited array and parent node
3. Mark the current node as visited.
4. Find all the vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices. If the recursive function returns true return true.
5. If the adjacent node is not parent and already visited then return true.
6. Create a wrapper class, that calls the recursive function for all the vertices and if any function returns true, return true.
7. Else if for all vertices the function returns false return false.

Q5 What do you mean by disjoint set data structure? Explain 3 operations along with examples, which can be performed on disjoint sets.

A disjoint set - data structure, also called a union - find data structure, is a data structure that stores a collection of disjoint sets. equivalently it stores a partition of a set into disjoint subsets. It provides operation for adding new sets, merging sets and finding a representative member of a set. The last operation makes it possible to find out efficiently if any two elements are in the same or different sets.

Q6 Run BFS and DFS on graph shown on right side (Graph with 8 vertices).

1. BFS



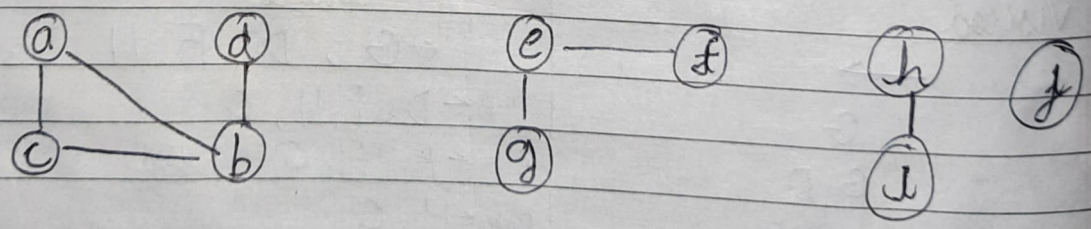
Visited	Queue
→	Pop ← G, D F H
G	Pop ← D F H, C
G D	Pop ← F H C
G D F	Pop ← H C
G D F H	pop ← C, E
G D F H C	Pop ← E, A
G D F H C E	Pop ← A, B
G D F H C E A	pop ← B
G D F H C E A B	—

PAGE No
DATE : / 202

DFS

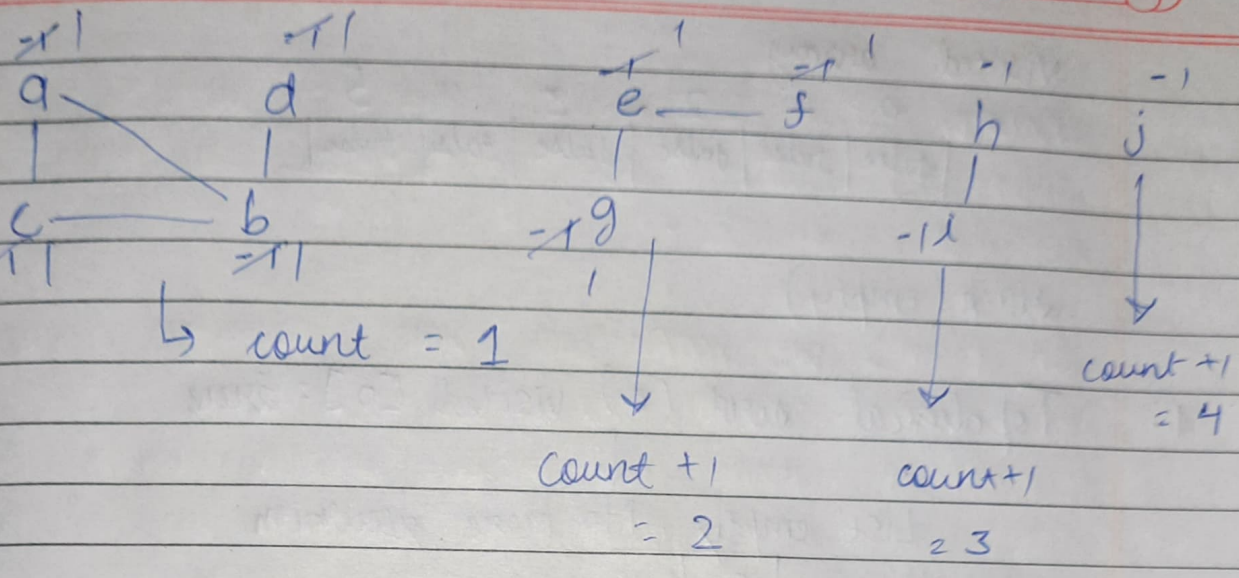
Visited	Stack
-	G
G	G D
G D	G D C
G D C	G D C E
G D C E	G D C E A
G D C E A	G D C E A B → Pop
G D C E A B	G D C E A → Pop
	G D C E → Pop
	G D C → Pop
G D C E A B F	G D F → Pop
G D C E A B F H	G D → Pop
	G H → Pop
	G → Pop
G D C E A B F H	

Q7 Find out number of connected components and vertices in each component using disjoint set data structure.



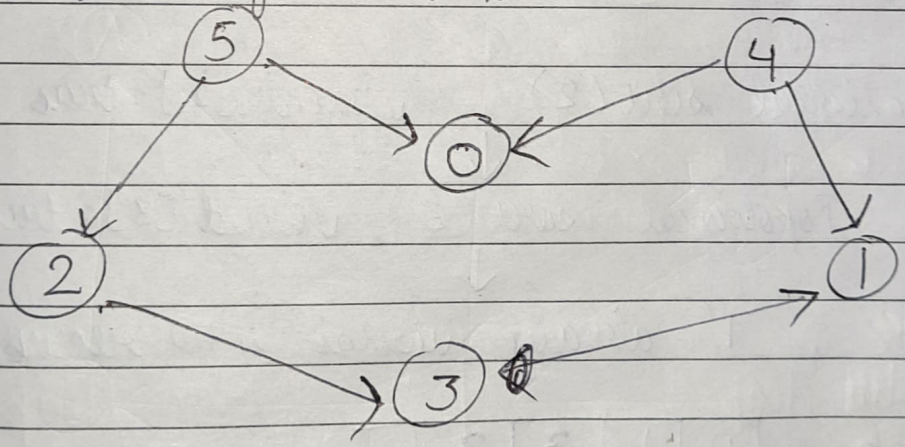
V = { a, b, c, d, e, f, g, h, i, j }

E = { (a,c), (a,b), (c,b), (d,b), (e,g), (e,f), (h,i) }



∴ total components = 4.

Q8 Apply topological sorting and DFS on graph having vertices from 0 to 5.



Adjacency list (G)

0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

visited array

0	1	2	3	4	5
false	false	false	false	false	false

stack (empty)

1. Topological sort (0), visited [0] = true
   ↓
   List empty, No more recursion
   stack [0]

2. Topological sort (1), visited [1] = true
   ↓
   List empty, No recursion
   stack [0 | 1]

3. Topological sort (2), visited [2] = true
   ↓
   Topological sort (3), visited [3] = true
   ↓
   '1' already visited ∴ no recursion
   stack [0 | 1 | 3 | 2]

4. Topological sort (4), visited [4] = true
   '0', '1' are already visited,
   stack [0 | 1 | 3 | 2 | 4]

5. Topological sort (5), visited (5) = true
   '2', '0' are already visited

stack	0	1	3	2	4	5

6. print the stack from top to bottom

i.e

5	4	2	3	1	0

DFS

visited	Stack
5	5
5, 2	5 | 2
5, 2, 3	5 | 2 | 3
5, 2, 3, 1	5 | 2 | 3 | 1 → Pop
	5 | 2 | 3 → Pop
	5 | 2 → Pop
5, 2, 3, 1, 0	5 | 0 → Pop
	5 → Pop
5, 2, 3, 1, 0	

Q) Heap data structure uses priority queue. Name few graph algorithms where you need to use priority queue and why?

Priority Queue is used in

1. Dijkstra's Shortest path algorithm
2. Prim's Algorithm
3. Data compression algorithm like Huffman coding

Dijkstra's algorithm uses priority queue because the value used to determine the order of the objects in the priority queue is the distance from our starting vertex. Priority queue

ensures that as we explore one vertex after another, we are always exploring the one with the smallest distance.

In Prim's we make our own priority queue in order to perform the operations

1. ExtractMin from all those vertices which have not yet been included in MST, we need to extract vertex with min key value.
2. DecreaseKey After extracting vertex we need to update keys of its adjacent vertices and if a new key is smaller then update that in data struct

In Huffman coding, we assign two popped node from priority queue as left and right child of new node. We push the new node formed in priority queue. We repeat all above steps until size of priority queue becomes 1.

Q10 What is the difference between Max and Min heap?

Min Heap	Max Heap
1. In a min-heap the key present at the node must be less than or equal to among the keys present at all of its children	1. In a max heap the key present at the root node must be greater than or equal to among the keys present at all of its childre
2. In a min-heap the minimum key element present at the root.	2. In a max-heap key element present at the root.

3. A min-heap uses the ascending priority In the construction of min-heap the smallest element has priority.	3. A max heap uses the descending priority. In the construction of a max heap, the largest element has priority.
4. In the construction of a min-heap, the smallest element has priority.	4. In a max heap the largest element is the first to be popped from the heap.