Tutorial - 5

Q. Diff b/w DFS and BFS. Application.

DFS	BFS
1. DFS uses stack data structure	1. BFS uses queue data structure for finding shortest path.
2. DFS, we might traverse through more edges to reach a destination vertex from a source.	2. BFS can be used to find single source shortest path in an unweighted graph, bcs in BFS we reach vertex with min no. of edges from source vertex
3 DFS is more suitable when there are soln away from source.	3. BFS is suitable for searching vertices which are closer to given source
4. Here, children are visited before siblings	4. Here siblings were visited before children.
5. DFS algo is recursive algo that uses idea of backtracking	5. In BFS, there is no concept of backtracking.
Application:	Application:
Acyclic Graph.	Bipartite Graph
Topological Order	Shortest path.

Date.
Page No.

Q Which data struct are used to implement BFS & DFS & why?

⇒ DFS (Depth First Search) algo traverses a graph in a depthward motion and uses a stack data structure to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

BFS (Breadth First Search) algo traverses graph breadthwise. Hence, uses queue data structure.
As By enqueuing the neighbour of start node, we force traversal process to visit all neighbours before moving to next level.

Q3 Sparse and Dense Graph? Which rep. of graph is better for each.

Sparse Graph is a graph in which the no. of edges is close to minimal no. of edges. Sparse graph can be a disconnected graph.
No. of edges = $O(n)$
where n = no. of vertices.

Dense Graph is a graph in which the no. of edges is close to the maximal no. of edges.
No. of edges = $O(n^2)$

Date.
Page No.

⇒ In Sparse Graph, adjacency lists are preferred since they require constant space for every edge.

⇒ Alternatively, in Dense Graph adjacency matrix is preferred as graph is dense to rep.

Q4 How can you detect a cycle in graph using BFS and DFS?

* Detect cycle using BFS:

Step 1: Compute in-degree (no. of incoming edges) for each of the vertex present in graph & initialize the count of visited nodes as 0.

Step 2: Pick all vertices with in degree as 0 & add them into queue

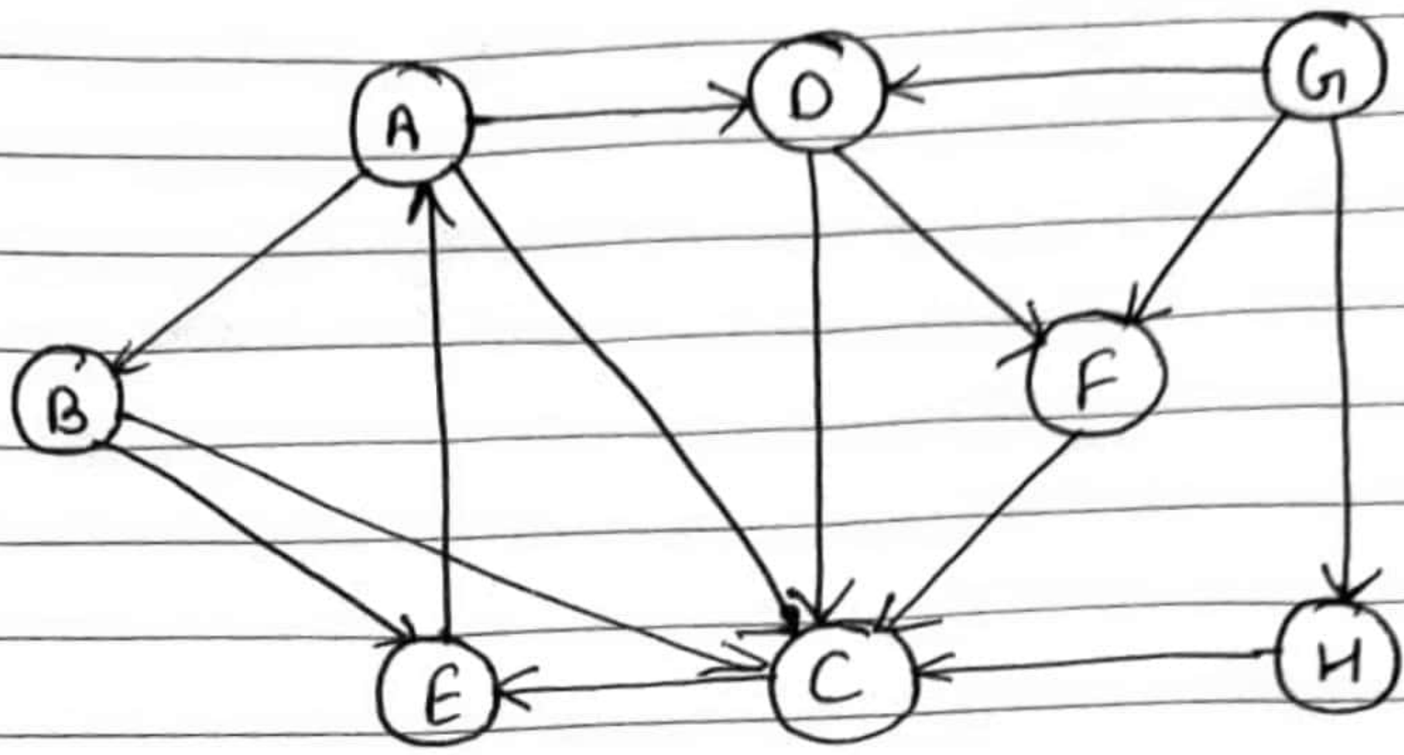
Step 3: Remove vertex from queue & then.
1. Increment count of visited nodes by 1.
2. Decrease in-degree for all its neighbouring nodes
3 If in-degree of a neighbouring nodes is reduced to zero, then add it to the queue.

Step 4: Repeat step (3) until the queue is empty.

Step 5: If count of visited nodes is not equal to the no. of nodes in graph has cycle, otherwise not.

Date.
Page No.

Q6 Run BFS and DFS

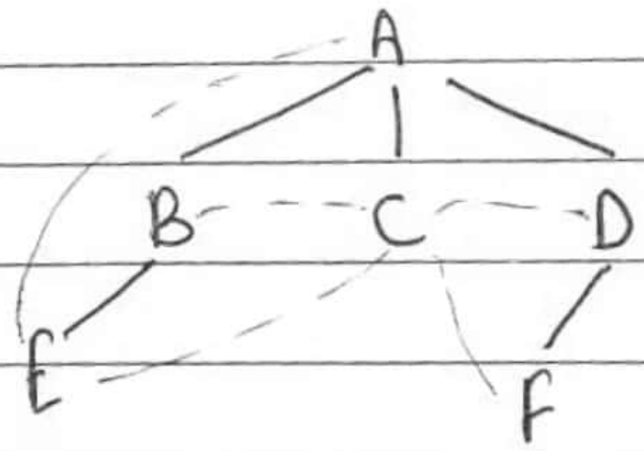


Assume Start node : A

BFS.

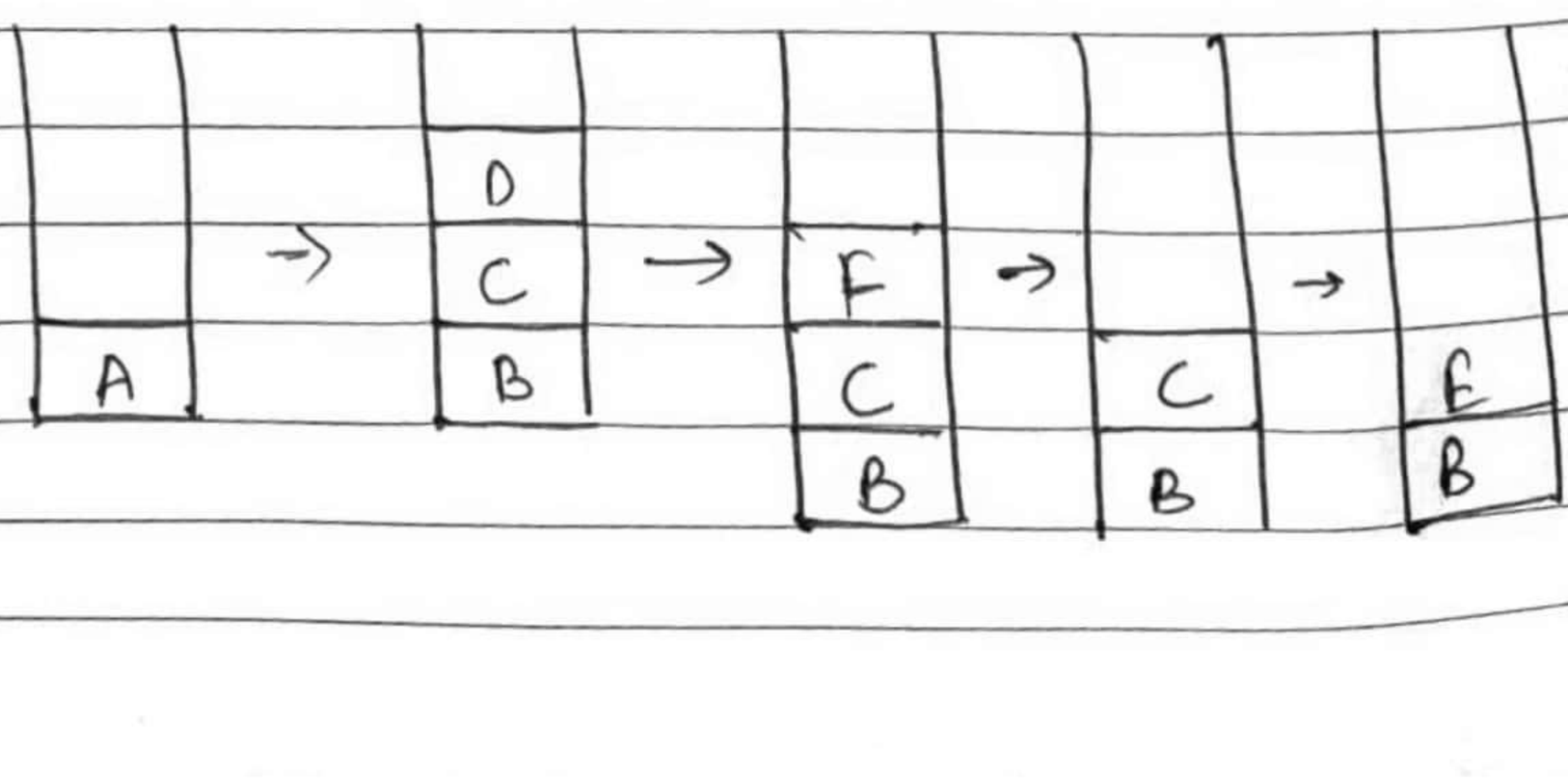
Queue

| ~~A~~ | ~~B~~ | ~~C~~ | ~~D~~ | ~~E~~ | ~~F~~ |

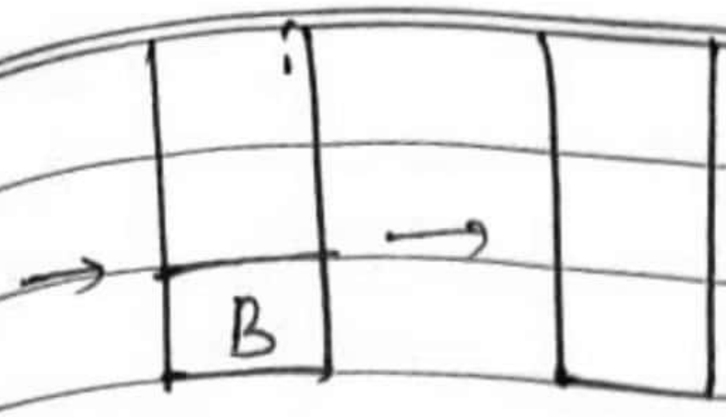


A → B → C → D → E → f

DFS



Date
Page No.



A → D → f → C → £ → B