TUTORIAL-5

Name:- Kartikey Rana
Section:- F
Roll no:- 61
University Roll no:- 2016812

1) BFS	DFS
(i) It stands for Breadth First Search.	(i) It stands for Depth First Search.
(ii) It uses queue data structure.	(ii) It uses stack data structure
(iii) It is more suitable for searching vertices which are closer to given source.	(iii) It is more suitable when there are solutions away from source.
(iv) BFS considers all neighbours first & therefore not suitable for decision making trees used in games & puzzles.	(iv) DFS is more suitable for game or puzzles. We make a decision then explore all paths through this decision. And if decision leads to win situation, we stop.
(v) Here siblings are visited before children.	(v) Here children are visited before siblings.
(vi) There is no concept of backtracking.	(vi) It is a recursive algorithm that uses backtracking.
(vii) It requires more memory.	(vii) It requires less memory.

Applications of:-

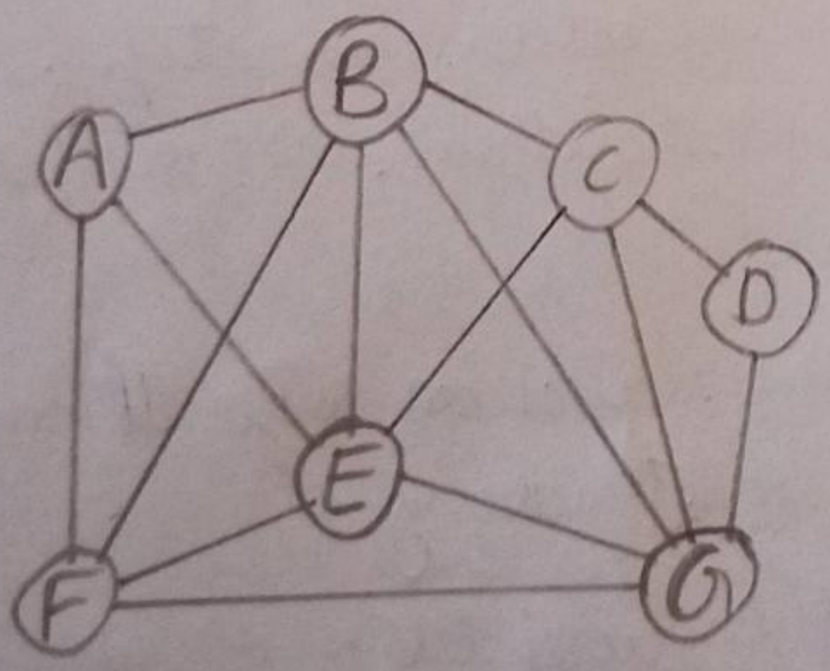
2) BFS:- Bipartite graph and shortest path, peer to peer networking, crawlers in search engines & GPS navigation.

3) DFS:- Acyclic graph, topological order, scheduling problems, sudoku puzzle.

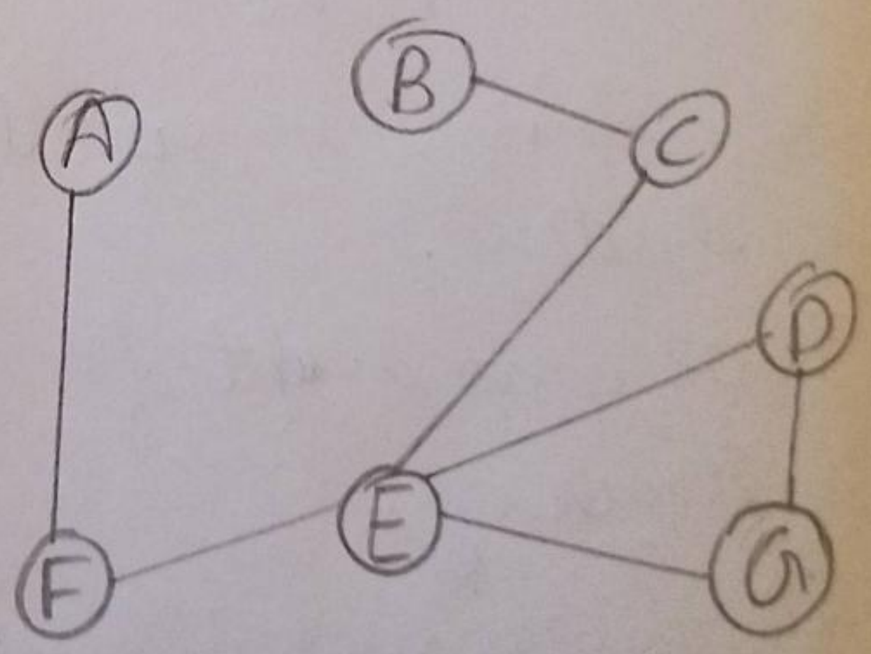
2) For implementing BFS we need a queue data structure for finding shortest path between any node. We use queue because things dont have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes level wise, i.e. it searches nodes with respect to their distance from root (source). For this queue is better to use in BFS.

For implementing DFS we need a stack data structure as it traverses a graph in depthward motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

3) Dense graph is a graph in which no. of edges is close to maximum edges. Sparse graph is a graph in which no. of edges is very less.



Dense Graph



Sparse Graph

- For sparse graph it is preferred to use adjacency list.
- For dense graph it is preferred to use adjacency matrix

4) For detecting cycle in a graph using BFS we need to use Kahn's algorithm for topological sorting :-

(i) Compute in-degree (no. of incoming edges) for each of vertex present in graph & initialise count of visited nodes as 0.

(ii) Pick all vertices with in-degree as 0 and add them in queue.

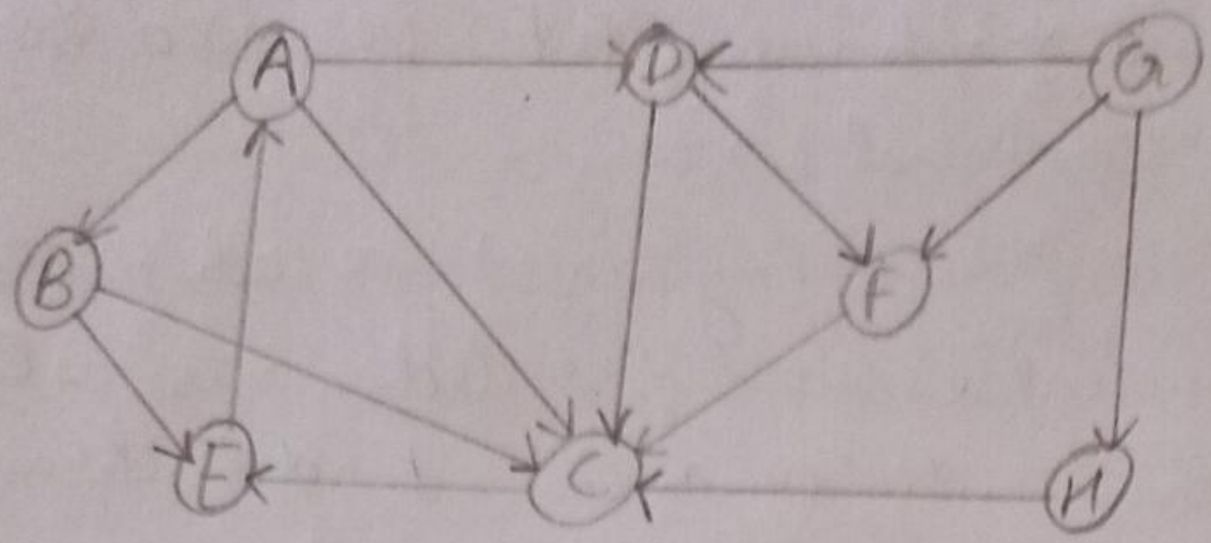
(iii) Remove a vertex from queue and then
- Increment count of visited nodes by 1.
- Decrease in-degree by 1 for all its neighbouring nodes.
- If in-degree of neighbouring nodes is reduced to zero then add to queue.

(iv) Repeat steps (iii) until queue is empty.

(v) If count of visited nodes is not equal to no. of nodes in graph has cycle, otherwise not.

For detecting cycle in graph using DFS we use the following steps :- DFS for a connected graph produces a tree. There is cycle in graph if there is a back edge present in the graph. A back edge is an edge that is from a node to itself or one of its ancestors in the tree produced by DFS. For a disconnected graph, get DFS as output. To detect cycle, check for a cycle in individual trees by checking back edges. To detect a back edge, keep a track of vertices currently in recursion track for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

6)



BFS →

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

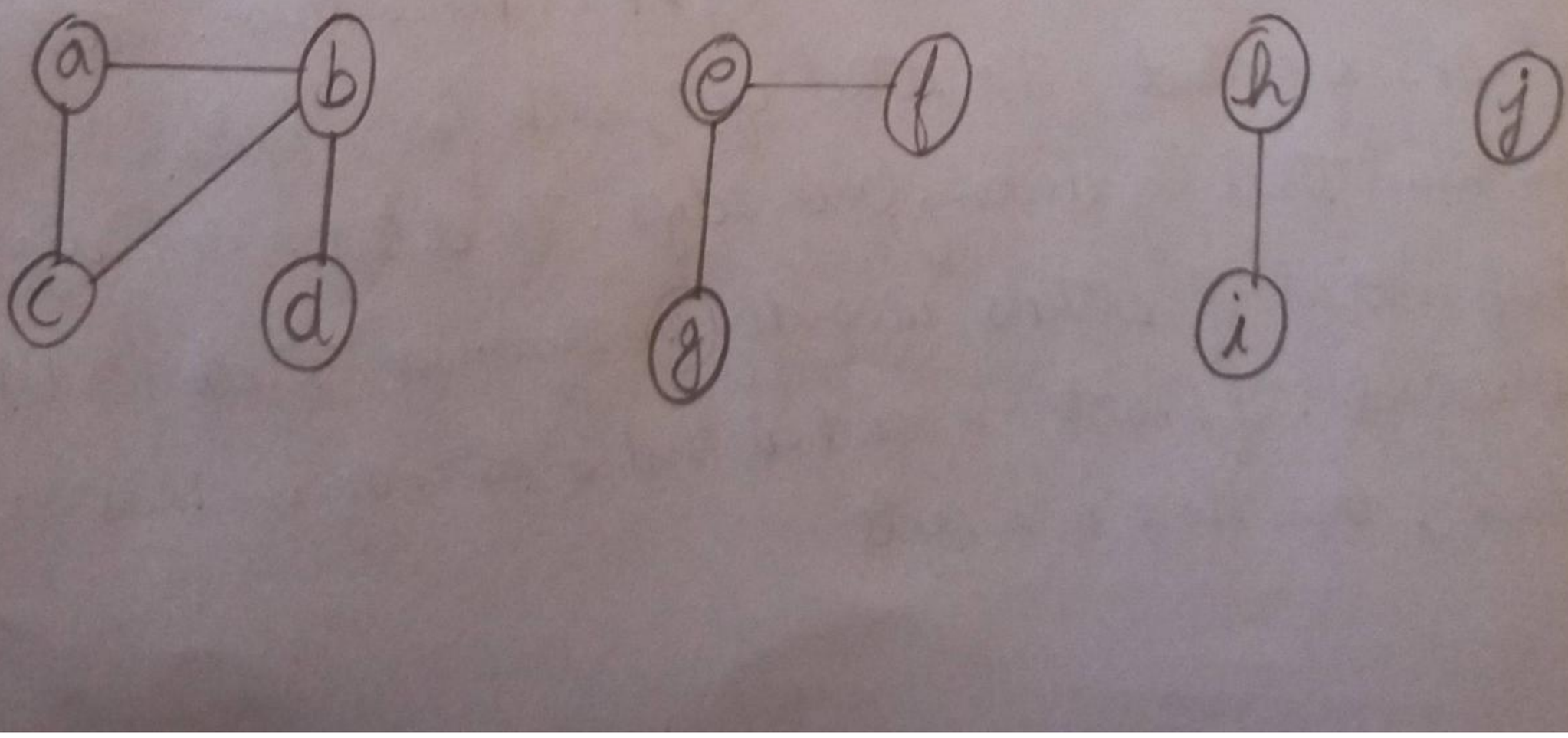
Path :- G → H → C → E → A → B

DFS →

Nodes visited	Stack
G	G
D	F
H	C
F	E
C	A
E	B
A	
B	

Path :- G → F → C → E → A → B

7)



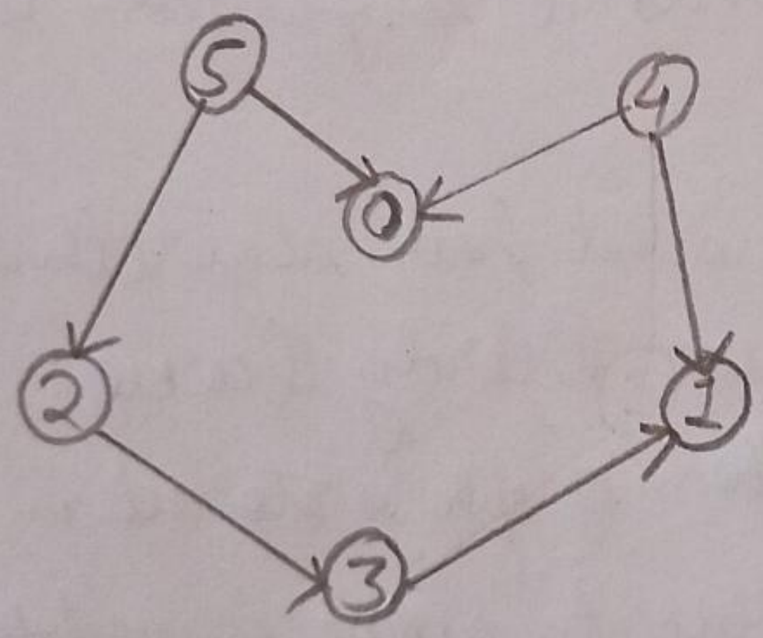
$V = \{a\}\{b\}\{c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$

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

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

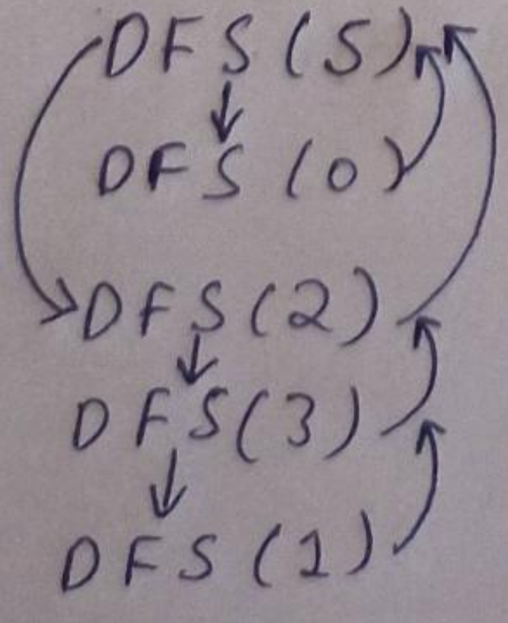
No. of connected components = 3

8)



We need source node as 5

Applying Topological sort

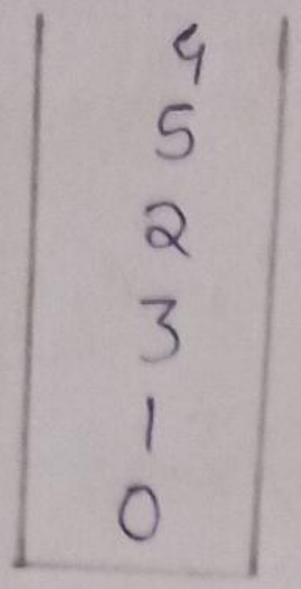


DFS(4)

↓

Not possible

DFS



Stack

4 → 5 → 2 → 3 → 1 → 0

9) Yes, heap data structure can lee used to implemant priority queue. It will take O(log N) time to invest and delete each element in priority queue. Based on heap structure, priority queue has two types: max-priority queue based on max heap and min-priority queue based on min-heap. Heaps provide better performance comparision to array & linked list.

The graphs like Dijkstra's shortest path algorithm, Prim's Minimum Spanning Tree use Priority Queue.

- Dijkstra's Algorithm → When graph is stored in form of adjacency list or matrix, priority queue is used to extract minimum efficiently when implementing the algorithm.
- Prim's Algorithm → It is used to store keys of nodes and extract minimum key node at every step.

10) Min Heap	Max Heap
(i) In min-heap, key present at root node must be less than or equal to among keys present at all of its children.	(i) In max-heap the key present at root node must be greater than or equal to among keys present at all of its children.
(ii) The minimum key element is present at the root.	(ii) The maximum key element is present at the root.
(iii) It uses ascending priority.	(iii) It uses descending priority.
(iv) The smallest element has priority while constructing min-heap.	(iv) The largest element has priority while constructing the max-heap.
(v) The smallest element is the first to be popped from the heap.	(v) The largest element is the first to be popped from the heap.