Avantika Singh
CST SEC 1
Roll No. 01.

PAGE No.
DATE

TUTORIAL - 5

Q1.

BFS (Breadth First Search)	DFS (Depth First Search)
i) It is used to find one of the possible paths from source to destination	i) It finds the shortest path in a graph from source to distance.
ii) It uses stack data structure.	ii) It uses queue data structure.
iii) In BFS, there is no concept of backtracking.	iii) It is a recursive algorithm that uses the idea of backtracking.

Q2. **BFS:-**

In this algorithm, we find the shortest path from source to destination. for this, it uses queue data structure i.e. First In First Out, to pick up the adjacent node.

DFS:-

In this algorithm, we find one of the possible paths. We retrieve it from root to the farthest node as much as possible, this is the same idea as LIFO. Hence, stack is used.

Q3. **Dense Graphs:**

If the no of edges is close to the maximum number of edges in a graph, then the graph is Dense Graph. In this, every pair of vertex is connected by one edge.

for dense graphs, adjacency matrix is preferred.

PAGE No.
DATE

★ **Sparse Graph :-**
It is completely opposite of Dense Graph. If graph has few edges i.e. the no. of edges is close to minimal no. of edges. Adjacency list is better representation of sparse graphs.

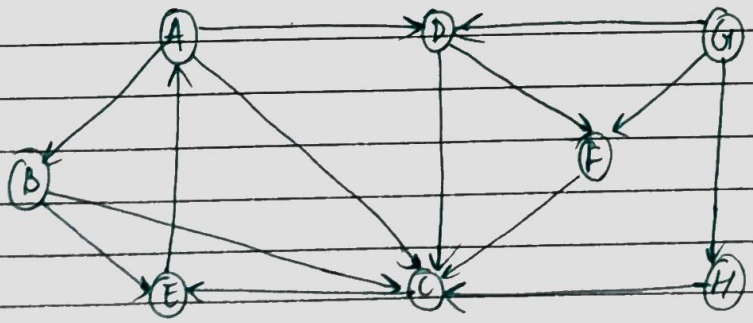
Q4. In BFS, we maintain a "visited" nodes list, when traversing a level, if we encounter a node that was already in this list, we come across a cycle.

In DFS, while traversing, if we find a node that is already in the recursion stack, a cycle is detected.

Q5. **Disjoint set Data structure :**
A disjoint set data structure is also known as union-find data structure and merge-find set. It contains collection of disjoint or non-overlapping sets.
Operations that can be performed on this data structure are :-

1. Intersection
2. Union
3. Delete an element

Q6.



PAGE No.
DATE

4 BFS.

Start Node → G Dest → B

Queue	Action	Visited
[G]	Insert D, F, H Remove G	G
[D | F | H]	Insert C Remove D	G, D, F, H
[F | H | C]	Remove F	G, D, F, H, C
[H | C]	Remove H	G, D, F, H, C
[C]	Insert E Remove C	G, D, F, H, C, E
[E]	Remove E Insert A	G, D, F, H, C, E
[A]	Remove A Insert B	G, D, F, H, C, E, A

Path: G → D → F → H → C → E → A

PAGE No.
DATE

4 DFS :

Stack	Node	Visited
G	Insert D	G
D G	Insert C	G, D
C D G	Insert E	G, D, C
E C D G	Insert A	G, D, C, E
A E C D G	Insert B	G, D, C, E, A

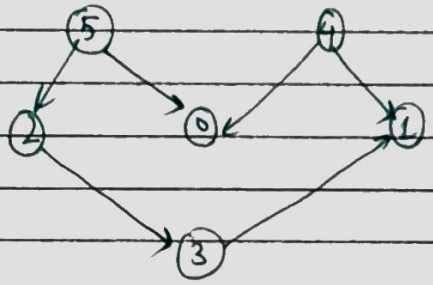
Path → G → D → C → E → A → B

Q7. 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 Component = 3.

Q8.



Top-sort (0) (No neighbour)
Top-sort (1) (No neighbour)
Top-sort (2)
↓
Top-sort (3) (No neighbour)
Top-sort (4)
Top-sort (5)

Stack
5
4
2
3
1
0

Linear Order: 5, 4, 2, 3, 1, 0

Q9. Binary heap is generally preferred over priority queue implementation because heap takes $O(\log n)$ for insertion, deletion etc. and heaps take constant time for peek() algorithm.

Algorithm where we need to use Priority Queue :-

i) Dijkstra's Algorithm
ii) Prim's Algorithm

In both the algorithms, we have to find the shortest path from source to destination.
To ensure that we explore the vertex, one after another must have smaller weight.

Q10. Difference b/w Min heap & Max heap.

	MIN HEAP	MAX HEAP
i)	In min heap, key at root nodes is <= child nodes.	i) In max heap, key at parent node is >= child nodes.
ii)	The min key element is present at the root.	ii) The max. key element is present at the root.
iii)	It uses ascending property.	iii) It uses descending property.
iv)	The smallest element has priority in min heap.	iv) The largest element has priority in max heap.