Tutorial - 5.

Samyu Jain
40 (I)

Ques-1.

BFS	DFS
→ Stands for breadth first search	→ Stands for depth first search,
→ It uses queue to find shortest path.	→ It uses stack to find shortest path.
→ It is better when target is closer to source.	→ It is better when target is far away from source
→ As BFS consider all neighbours so it is not suitable for decision.	→ It is more suitable for decision tree. As with one decision we need to traverse further to argument the decision If we search the conclusion.

Application of DFS-

→ Using DFS, we can find the b/w two vertices.

→ we can perform topological sorting which is used to schedule jobs.

→ We can use DFS to detect cycles.

→ Using DFS, we can find strongly connected components of a graph.

Application of BFS-

→ BFS may also used to detect cycles.

→ Finding shortest path and minimal spanning tree in unweighted graphs.

→ In networking, finding a router for peak to transmission

→ finding a router through GPS navigation system.

Stack

0	1	3	2	

Step 4 :- Stack

0	1	3	2	4	

Step 5 :- Stack

0	1	3	2	4	5	

Step 6 :- Print all elements of stack from top to bottom
5, 4, 3, 2, 1, 0.

Ans 9→ Algorithm that uses Priority Queue

(i). Dijkstra's Algorithm
When graph is sorted in the form of list or matrix, priority queue can be used to extract minimum efficiency when implementing Dijkstra's Algorithm.

(ii). Prim's Algorithm
It is used to implement prims algo to store key of nodes to extract minimum key node at every step.

(iii). Data compression
It is used in Huffman's code which is used to compress data.

Ans 10→ Difference between Max and Min Heap :-

Min Heap	Max Heap
• In min heap, the key present at root node must be less than or equal to among the key present all its children.	In max heap, the key present at root node must be greater than or equal to the key present at all its children.
• Uses the ascending priority.	Uses the descending priority.
• The minimum key is present at the root node.	The maximum key is present at the root node.

Ans 3→ Sparse Graph :-

A graph in which the number of edges is much low then the possible number of edges.

Dense graph :-

A dense graph is a graph in which the number of edges is close to the maximum number of edges.

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

Ans 4→ DFS can be used to detect cycle in a graph.

DFS for a connected graph produces a tree. There is a cycle in a graph only 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 ancestor in the tree produced by DFS.

BFS can be used to detect cycles. Just perform BFS, which keeping a list of previous nodes at each node visited or else constructing a tree from the starting node. If I visit a node that is already marked by BFS, I found cycle.

Ans 5→ Disjoint Set Data Structure :-

- It allows you to find out whether the two element are in the same set or not efficiently.
- A disjoint set can be defined as the subset when there is no common elements between the two nodes.

Eg - $S_1 = \{1, 2, 3, 4\}$
$S_2 = \{5, 6, 7, 8\}$

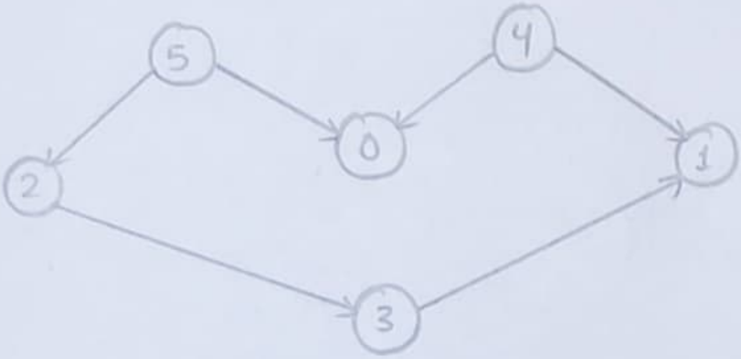
Operation Performed :-

(i). find :

```
int find (int v)
{ if (v == parent[v])
      return v;
   return parent[v] = find(parent[v]);
}
```

(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}

Ans 8 →



Adjacent List :-

0 →

1 →

2 → 3

3 → 1

4 → 0, 1

5 → 2, 0

Visited

0	1	2	3	4	5
False	False	False	False	False	False

Stack (Empty)

Step 1 :- Topological Sort (0), Visited [0] = True

stack	0	

Step 2 :- Topological Sort (1), Visited [1] = True

Stack	0	1	

Step 3 :- Topological Sort (2), Visited [2] = True

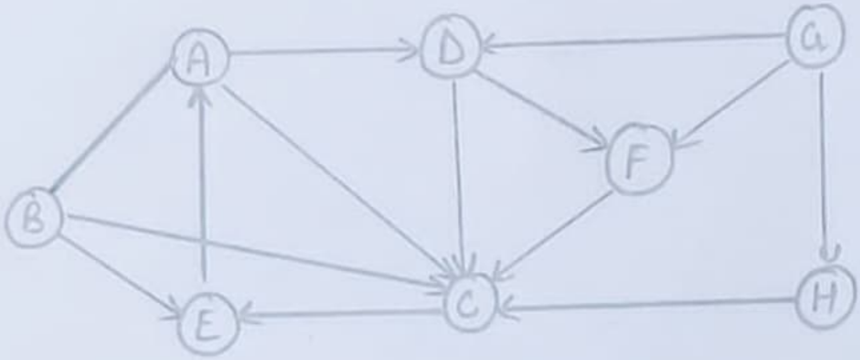
↓

Topologicalsort (3), Visited [3] = True

(ii). Union :

```
void union (int a, int b)
{
      a = find (a)
      b = find (b)
      if (a != b)
      {   if (size[a] < size[b])
              swap (a,b)
          parent [b] = a;
          size [a] += size[b];
      }
}
```

Ans 6 →



BFS :-

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

Path : B → E → A → D → F

DFS:-

Node processed :	B	B	C	E	A	D	F
stack :	B	CE	EE	AE	DE	FE	E

Path : B → C → E → A → D → F

Ans 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}