Tutorial - 5.

Name :- Gaurang Srivastava
Section :- F
Roll No. :- 57
Univ. Roll. No. :- 2016746.

Q1. Ans

BFS	DFS
• It stands for Breadth First Search	It stands for Depth first search
• It uses queue data structure	It uses stack data structure.
• It is more suitable for searching vertices which are closer to given source.	It is more suitable when there are solutions away from source.
• BFS considers all neighbours first & therefore not suitable for decision making trees used in games & puzzles.	DFS is more suitable for game or puzzle problems. We make a decision then explore all paths through this decision. And if decision leads to win ~~sad~~ situations, we stop.
• Here siblings are visited befor children.	Here children are visited before siblings.
• There is no concept of backtracking	It is a recursive algo that uses backtracking.

Applications :-

- BFS ⇒ Bipartite graph & shortest path, peer to peer networking, crawlers in search engines & GPS navigation system.
- DFS :- Acyclic graph, tropical order, scheduling problems, sudoku puzzle.

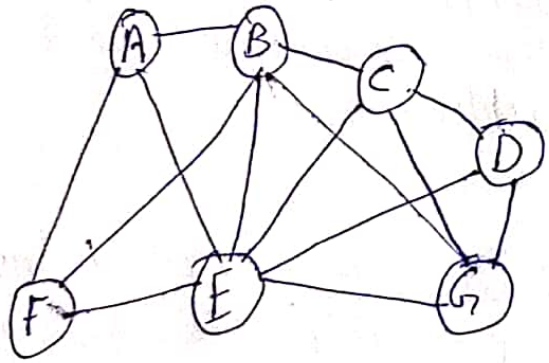
Q2. Which data structure are used to implement BFS & DFS & why?

→ For implementing BFS we need a queue data structure for finding shortest path between any node, We use queue because things don't have to be processed immediately, but have to be processed in FIFO order like BFS searche for nodes level wise, i.e it searches nodes w.r.t there distance from root (source). So For this queue is better to use in BFS.

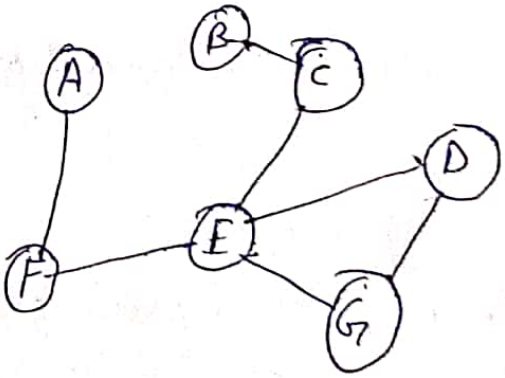
For implementing DFS we need a stack data structure as it traverses a graph in depth ward motion & uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Q3. Dense graph is a graph in which no. of edges is close to maximal of no. of edges.

Sparse graph is 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.

Q4. For detecting cycle in a graph using BFS we need to use Kahn's algorithm for Topological Sorting.

The steps involved are :

1) Compute in-degree (no. of incoming edges) for each of vertices present in graph & initialize count of visited nodes as 0.
2) Pick all vertices with in-degree as 0 and add them in queue.
3) Remove a vertex from queue then 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.
4) Repeat 3) until queue is empty
5) If count of visited nodes is not equal to no. of nodes in graph, has cycle otherwise not.

For detecting cyclic in graph using DFS we need to do following:-

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 (self-loop) or one of the its ancestor in the tree produced by DFS. For a disconnected graph get DFS forest as output. To detect cycle, check for a cycle in individual tree by checking back edges. To detect a cycle, check for a cycle in individual tree by checking back edges.

Q5. A disjoint set is a data structure - that keeps track of set of elements partioned into several disjoint subsets.

3 Operations:-

- Find:- can be implemented by recursively traversing the parent array until we hit a node who is parent to itself.

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

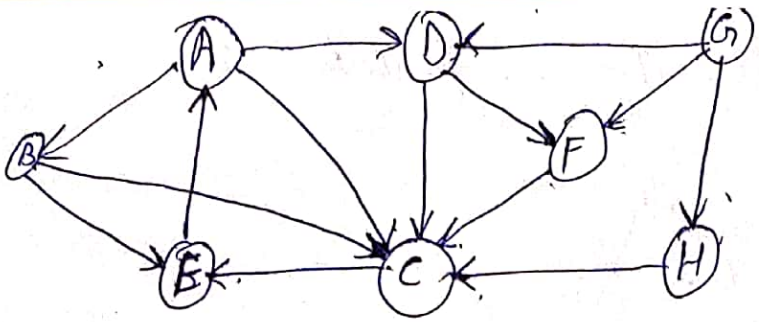
*) Union: It takes 2 or elements as input & find representative of this so is using the find operation & finally puts either one of the tree under root node of other tree, effectively merging the trees & sets.

```
eg.  void union (int i, int j) {
        int irep = this.Find(i);
        int jrep = this.Find(j);
        this.parent[irep] = jrep;
     }
```

-) Union by Rank: We need a new array. rank[]. ~~Since~~ Size of array come as parent array. If i is representable of set, rank[i] is height of tree.

```
eg.  void union (int i, int j) {
        int irep = this.Find(i);
        int jrep = this.Find(j);
        if (irep == jrep) return;
        irank = Rank[irep];
        jrank = Rank[jrep];
        if (irank < jrank)
            this.present[irep] = jrep;
        else if (jrank < irank)
            this.parent[jrep] = irep;
        else {
            this.parent[irep] = jrep;
            Rank[jrep]++;
        }
    }
```

Q6.



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

G
D
H
F
C
E
A
B

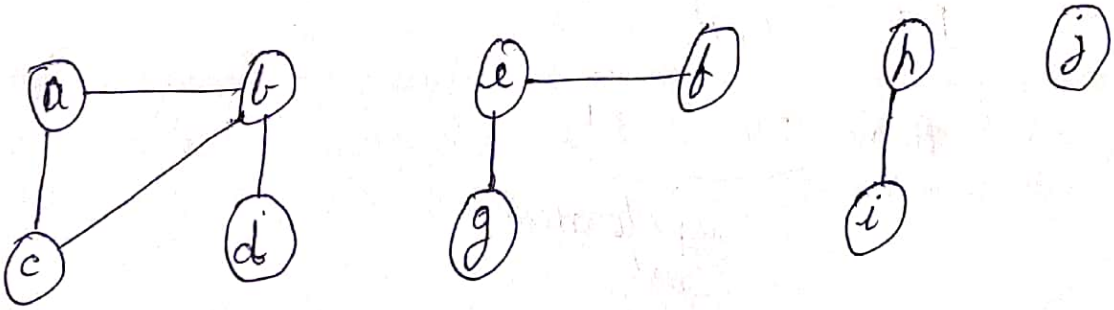
Nodes visited

G
F
C
E
A
B

Stack.

Path → G → F → C → E → A → B

Q7.



Q.

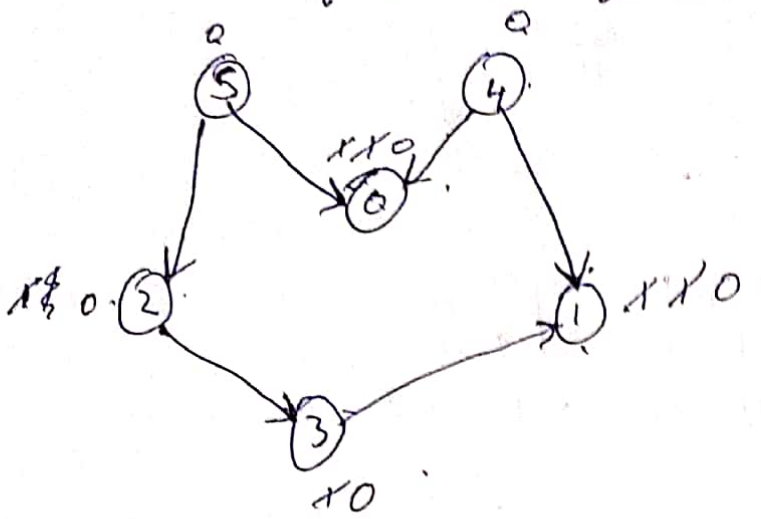
V = {a} {b} {c} {d} {e} {f} {g} {h} {i} {j}

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}

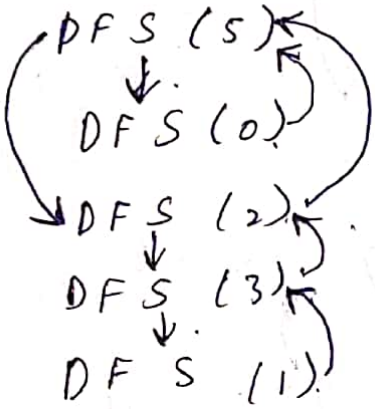
No. of connected components = 3.

Q 8.



We take service node as 5.

Applying Topological Sort.



DFS (4)
↓
Not possible

Ans:- 5 4 2 0 3 1.
Topological sort

q: 5/4 ; Pop 5 & undergo of it by 1

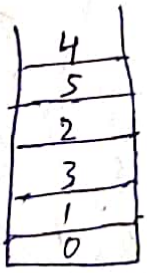
q: 4/2

q: 2/0

q: 0/3

q: 1 ; Pop 1.

DFS.



Stack

4 → 5 → 2 → 3 → 1 → 0.

Q9. Yes, heap data structure can be used to implement priority queue. It will take O(log N) time to insert & delete each element in priority queue. Based on heap structure priority queue has two types max-priority queue based on max heap & min priority queue based on min-heap. Heaps provide better performance comparision to array & L.L.

The graphs like Dijkstra's shortest path algo., Prism's minimum spanning tree use Priority queue.

Q10.

Min - Heap	Max - Heap.
1.) In min-heap, key present at root node.	In max-heap the key present at root node must be greater than or equal to among keys present at all of its children.
2.) The minimum key element is present at the root	The maximum key element is present at the root.
3.) It uses ascending priority	It uses descending priority
4.) The smallest element has priority while construction of Min-heap.	The largest element has priority while construction of Max-heap
5.) The smallest element is the first to be popped from the heap.	The largest element is the first to be popped from the heap.