Tutorial - 5.

Ques 1
Soln

BFS	DFS
• Stands for Breath first Search	• Stands for Depth fer Search.
• DFS uses queue to find the shortest path	• It uses stack to find shortest path.
• BFS is better when target is closer to Source	• DFS is better when target is far from Source.
• As BFS consider all neighbour so it is not Suitable for decision, trees used in puzzle games	• DFS is more suitable for decision tree.
• BFS is slower than DFS	• DFS is faster than BFS.

Application of DFS
- Using DFS we can find path between two vertices.
- We Can which is used to scheduling jobs.
- we can use DFS to detect cycles.

Application of BFS:
- BFS may also used to detect cycles.
- Finding Shortest Path and minimal spanning tree
- In network finding a route for packet transmission

Ques 2 BFS uses Queue data structure. BFS you mark any node in graph as source node, transverse all the nodes in graph and keeps as completed. BFS visited an adjacent unvisited node, marks it as done and insert it into Queue.

DFS us stack a graph in a depthwards beto motion and uses a stack to remember to get the next vertex to start to a search, when a dead an occurs in any iteration.

Ques3

Sparse graph :- A graph in which the number of edges number of edge.

Dense graph :- A graph which the number of edges the maximal nos of edges of edges.

→ If the graph is sparse, we should store it as list of edges.

Ques4

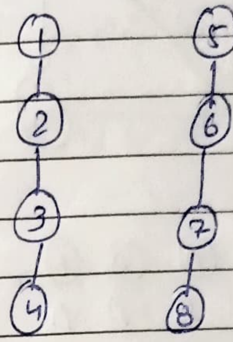
DFS can be used to detect cycle in a graph DFS for connected graph produces a tree. There is a cycle in graph. only is graph. A backedge In an edge that is form a node to it self or one of its ancestor in the tree produced by DFS.

BFS can also be used cycles. perform BFS while keeping a list of previous nodes at each node visited or else that is already marked by BFS I founded a cycle.

Ques
Sol^n

Disjoint Set Data structure:-

- It allows to find out whether the two elements are in the same set or not efficiently.



$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{5, 6, 7, 8\}$

Operation Performed:-

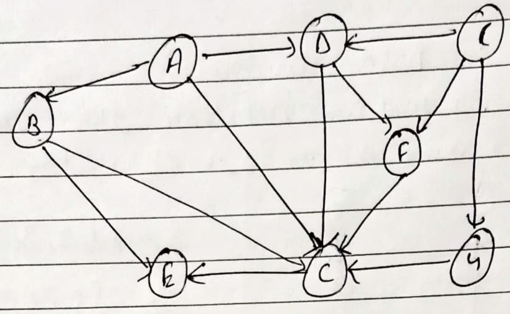
① find

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

Unions:-

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

Qus 6:-



BFS:- Node : B E C A D F

~~B~~ B B E A D

path : B → E → A → D → F

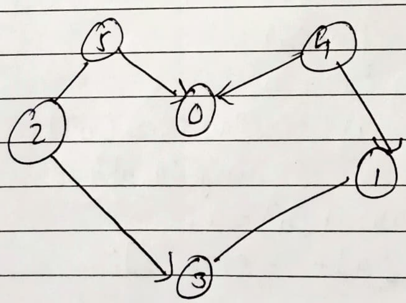
DFS :

Node : B B C E A D F

Stack: B CE EE AE DE FE F

path B → C → E → A → D → F

Qus 8



V → Visted.
f → false

Adjacency list :

0 →
1 →
2 → 3
3 → 1
5 → 2,0   4 → 0,1

0	1	2	3	4	5
F	F	F	F	F	F

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;

↓

Topological Sort (3), Visited [3] = true;

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, 2, 3, 1, 0;

Ans 10

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