TUTORIAL - 5

Q1 Differentiate b/w BFS & DFS

Ans 1

BFS	DFS
• Stands for Breadth first search.	Stands for Depth first search.
• BFS 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.	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.
- ~~The~~ Finding shortest path and minimal spanning tree.
- In network finding a route for packet transmission.

Ans 2

BFS used Queue data structure. BFS you mark any mode in graph as source mode, transfer 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 uses stack a graph in a depth world, motion and uses stack to remember to get the next vertex to start to a search, when a dead on occurs in any iteration.

Ans 3

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

Dense graph: A graph which the number of edges the maximal no. of edges.
If the graph is sparse, we should store it as list of edges.

Ans 4

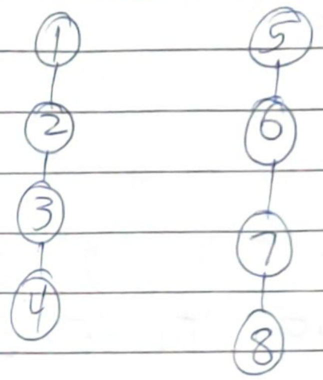
DFS can be used to detect cycle in a graph DFS. DFS for connected graph produces a tree. There is a cycle in graph. Only if graph. A backedge in an edge that is form a node to itself or out of its ancestor in the tree produces by DFS.

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 cycles.

Ans 5

Disjoint set Data structure:

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



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

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

Operation performed:

① find int find (V)

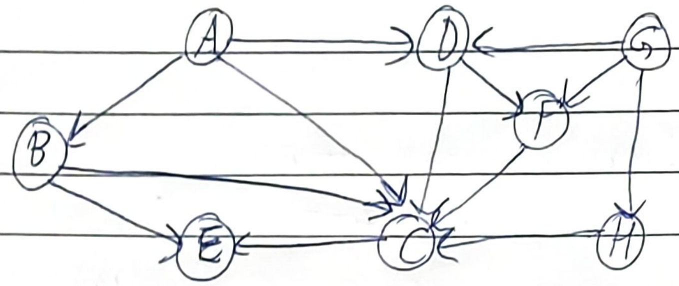
```
if (V == parent [V])
    return V;
return parent [V] = find (parent [V]);
```

Unions:

```
Void union (int a, int b) {
    a = find (a)
    b = find (b)
    if (a != b)
    if (Size [a] < size [b])
    { Swap (a, b) }
    parent [b] = a;
```

size [a] + = size [b];
}
}

Q 6



Ans 6 BFS: Node : B E C A D E

B B E A D

path: B → F → A → D → F

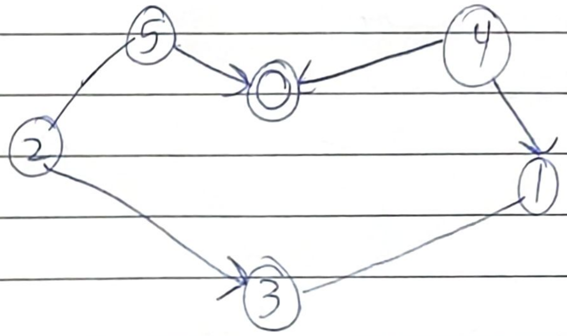
DFS:

Node: B B C E A D F

Stack: B CE EE AE DE FE E

path B → C → E → A → D → F

Q 8



V → Visited

f → false

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

stack (Empty)

Ans 8 Adjacency list:

0 →

1 →

2 → 3

3 → 1

5 → 2,0 4 → 0,1

Step 1: Topological Sort[0], visited [0]=true
Stack $\boxed{0}$

Step 2: Topological Sort[1], visited [1]= true
Stack $\boxed{0|1}$

Step 3: Topological Sort(2), Visited (2] = true;
Topological Sort (3), Visited [3]=true;
Stack $\boxed{0|1|3|2}$

Step 4:
Stack $\boxed{0|1|3|2|4}$.

Step 5:
Stack $\boxed{0|1|3|2|4|5}$

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.
- Used to ascending priority.	Uses descending priority.

· The minimum key present at the root node.

The max. key present at the root node.