NAME : Aastha Verma
Section : I
Roll No : 12

[Tutorial-5]

Q1:
⇒

BFS	DFS
• Stands for Breadth first search	• Stands for Depth for 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 considers all neighbours so it is not suitable for decision tree used in puzzle games	• DFS is more suitable for Decision tree. As with one decision we need to traverse further to argument the decision If we search the conclusion.
• BFS is slower than DFS	

Application of DFS
- Using DFS we can find path between two vertices.
- We can perform topological sorting which is used to scheduling 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 graph.
- In networking finding a route for packet transmission.
- Finding a route through GPS navigation system.

Q5.
⇒ Breadth first search (BFS) uses Queue data structure. In BFS you mark any node in the graph as source node and start traversing from it. BFS traverses all the nodes in the graph and keeps dropping them as completed. BFS visited an adjacent unvisited node, marks it as done and insert it into Queue.

DFS uses stack data structure because DFS traverse a graph in a depthward motion and uses a stack to remember to get get the next vertext to start a search, when a dead ~~each~~ end occurs in any iteration.

Q3:

→ **Sparse graph**: A graph in which the number of edges is much less than the possible number of edges.

Dense graph: A dense graph is a graph in which the number of edges. of edges.

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

Alternatively if a graph is dense, we should store it as a adjacency matrix.

Q4:

→ 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 ~~aa~~ ancestor in the tree produced by DFS.

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

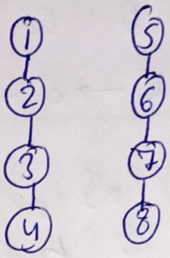
Q5:

Soln:

Disjoint Set Data structure:

- It allows to find out whether the two elements are in the same set or efficiently.
- A disjoint set can be defined as the subsets when there is no common element between the two sets.

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



Operation performed:

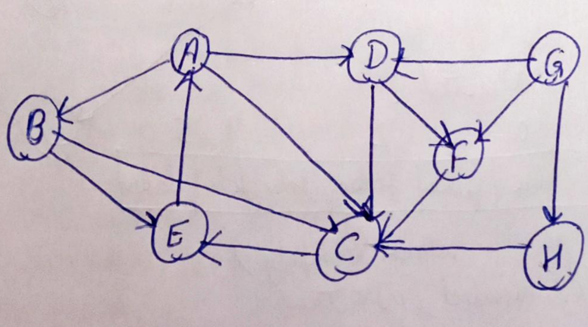
(i) find:

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

(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];
  }
}
```

Q6.



BFS: Node: B E C A D F

parent: – B B E A D

path: $B \rightarrow E \rightarrow A \rightarrow D \rightarrow F$

DFS: Node processed B B C E A D F

stack B CE EE AE DE FE E

path: $B \rightarrow C \rightarrow E \rightarrow A \rightarrow D \rightarrow F$

Q7:

Ans: 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

Q8:

⇒ Adjacency list

$0 \rightarrow$

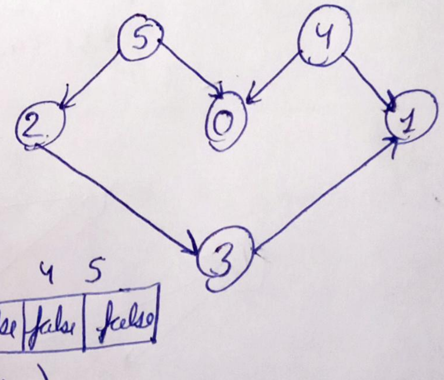
$1 \rightarrow$

$2 \rightarrow 3$

$3 \rightarrow 1$

$4 \rightarrow 0, 1$

$5 \rightarrow 2, 0$



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

0	1

Step 3: Topological sort (2) visited [2] = true

Talogical 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

Q 9:

⇒ **Algorithms that use Priority Queue:**

(i) **Dijkstra's shortest path Algorithm using priority Queue**

When graph is sorted in the form of list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijsktra's Algo.

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

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

Q 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 at all its children.	• In max-head the key present at root node must be greater or equal to the key present at all its childrens.
• Uses the ascending priority	• Uses descending priority
• The minimum key present at the root node.	• The max. key present at the root node.