Tutorial - 5

Sol :-

BFS	DFS
• BFS stands for Breadth first search	• DFS stands for Depth first search
• BFS uses queue to find the shortest path.	• DFS uses stack to find shortest path.
• BFS is better when target is closer to source.	• DFS is better when target is far from source.
• slower than DFS.	• faster than BFS.
• T.C = O(V+E)	• T.C = O(V+E)

- Applications of DFS -
- use to find path b/w two vertices
- We can detect cycles in graph using DFS. If we get one back-edge during BFS.
- using DFS, we can find strongly connected components of graph. If there is a path from each vertex to every other vertex, that is strongly connected.

- Applications of BFS -
- BFS can used to find cycle in a graph
- finding shortest path and minimal ~~spanning tree~~ in unweighted graph.
- In peer-to-peer networking, BFS is to find neighbouring node.

Sol 2 - • BFS (Breadth first search) uses queue data structure for finding shortest path.
• DFS (Dept first search) uses stack data structure.

- A queue (FIFO) data structure is used by BFS. You make any node in the graph as root and start traversing the data from it. BFS traverse all nodes in graph and keeps dropping them as completed. BFS visits an adjacent unvisited nodes marks it as done, and inserts it into a queue.
- DFS algorithm traverse a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Sol 3 - Sparse graph - a graph in which number of edges is much less than possible number of edges.

Dense graph - a graph in which number of edges is close to maximal no. of edges possible.

Sol 4 - The existence of a cycle in directed and undirected graph can be determined by whether depth first search (DFS) finds an edge that points to an ancestors of the current vertex. All the back-edges which DFS skips over are part of cycles.

Page No.
Date

- **Detect cycle in a directed graph –**

DFS can be used to detect a 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 that is from a node to itself or one of its ancestors in the tree produced by DFS.

- **Detect cycle in undirected graph –**

Run a DFS from every unvisited node. DFS can be used to detect a cycle in a graph. DFS for connected graph produce a tree. There is a cycle in graph only if there is a back edge present in the graph.

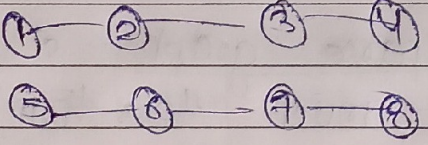
9/5 – **Disjoint set data structure –**

It allows to find out whether the two elements are in same set or no efficiently.

– The disjoint set can be defined as the subsets where there is no common element b/w two sets.

Eg – $S_1 = \{1, 2, 3, 4\}$

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



Operations –

1) **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]);
}
```

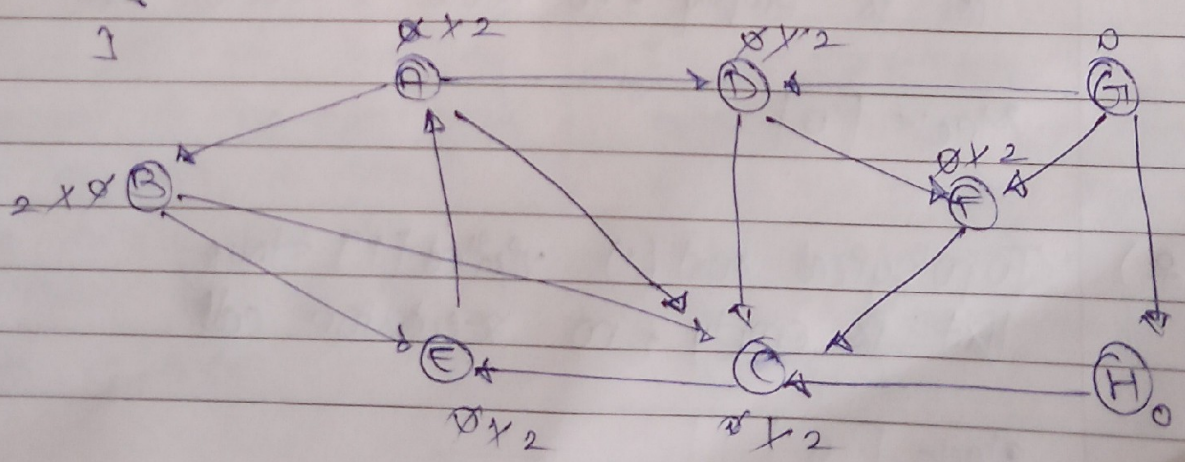
2) Union - It takes, as inputs, two elements and finds the representatives of their sets using the find operation and finally puts either one of the trees under the root node of the other tree, effectively merging the tree and the sets

```
void union (int i, int j)
{
  int iRep = this.find(i);
  int jRep = this.find(j);
  this.parent[iRep] = jRep;
}
```

3) Path compression - It speeds up the data structure by compressing the height of tree. It can be achieved by inserting a small caching mechanism into Find operation.

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

Sol:-



Page No.
Date

BFS -

node	B	E	C	A	D	F
Parent	-1	B	B	E	A	D

unvisited nodes Gard H

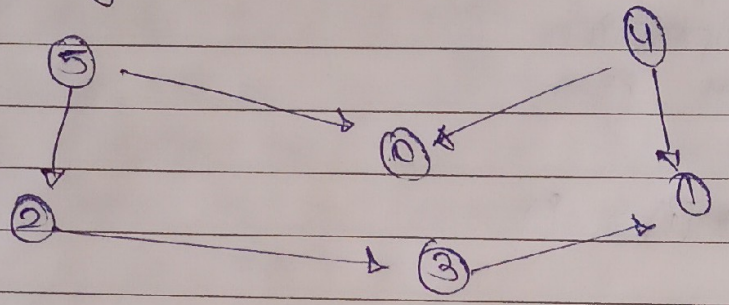
Path : B → E → A → D → F.

DFS -

Node processed	B	B	C	E	A	D	F
stack	B	CE	EE	AE	DEFE	E	

Path : B → C → E → A → D → F.

Sol8 - topological sort -



Adjacent list.

0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

	0	1	2	3	4	5
Visited	false	false	false	false	false	false

stack (empty)

1) Topological sort(0) visited[0] = true
list is empty, no more recursion call

stack: | 0 |

2) Topological sort(1) visited[1] = true
list is empty, no recursion call

stack: | 0 | 1 |

3) Topological sort (2), visited [2] = true

↓

topological sort (3), visited [3] = true.

'1' is already visited. No more recursion call

stack | 0 | 1 | 3 | 2 |

4) Topological sort (4), visited [4] = true.

'0', '4' already visited. no more recursion call

stack | 0 | 1 | 3 | 2 | 4 |

5) topological sort (5), visited [5] = true.

↓

'2', '0' already visited. No more recursion call

stack | 0 | 1 | 3 | 2 | 4 | 5 |

6) Print all element.

5, 4, 2, 3, 1, 0.

Sol 10.

Min-Heap	Max-Heap
• In a min-heap the key present at the root must be less than or equal to among the key present at all of its children.	• In a max-heap the key present at the root node must be greater than or equal to among the keys presented at all of its children.

• the minimum key element present at the root • uses the ascending priority	• the maximum key element present at root. • uses descending priority.