## Tutorial 5

DFS-) DFS means depth first search uses a stack to heap track of the next location to visit.

→ Children are visited before siblings.

Application!

(1) Detecting cycle in Graph.

(2) Path ~~p~~ finding

(3) Topological Sorting

(4) Solving puzzles with only one solution.

BFS:

→ uses Queue Data structure,

Stands For Breadth First Search,

Sibling are visited before the children.

Application:

1. Shortest path & minimum spanning tree For unweighted Graph
2. peer to peer network
3. Social Networking website.
4. GPS Navigation System.

Ans 2) In BFs we use Queue data structure as queue used when things don't have to be processed immediately but have to be processed in fifo. like BFs

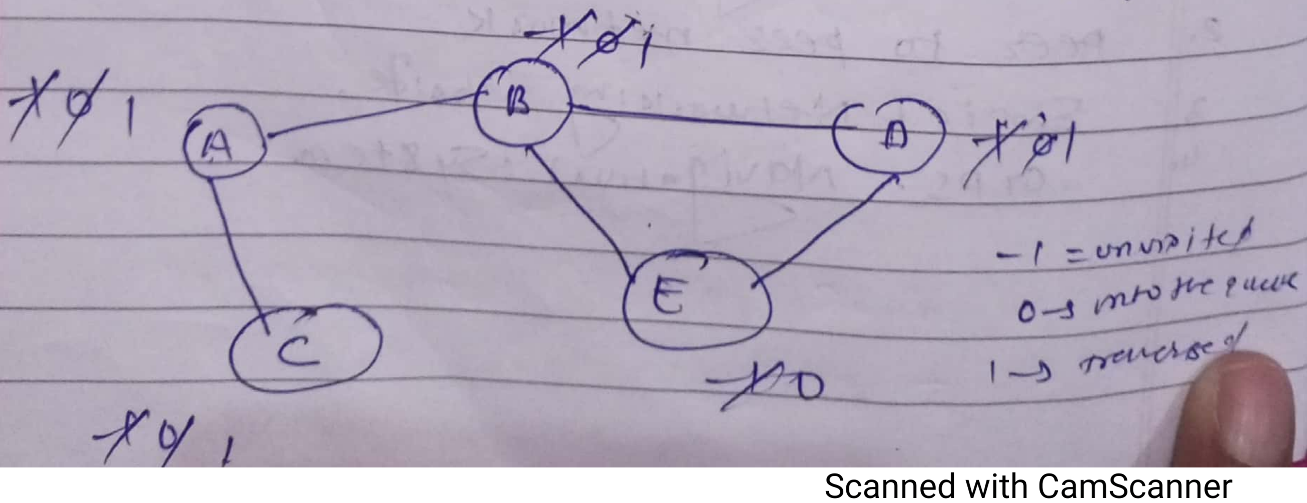
In dFs stack is used as DFs use back tracking For DFs we retrive it From to the farthest node as much as possible this is the same ide as LIFO.

3 Ans:- Dense graph is a graph which the no of edges is close to the maximal no of edges.

Sparse Graph is a graph in which the no of edge is close to minimal no of edge it can be disconnected graph

Adjacency list are prefered for sparse graph & Adjacency matrix for dense graph

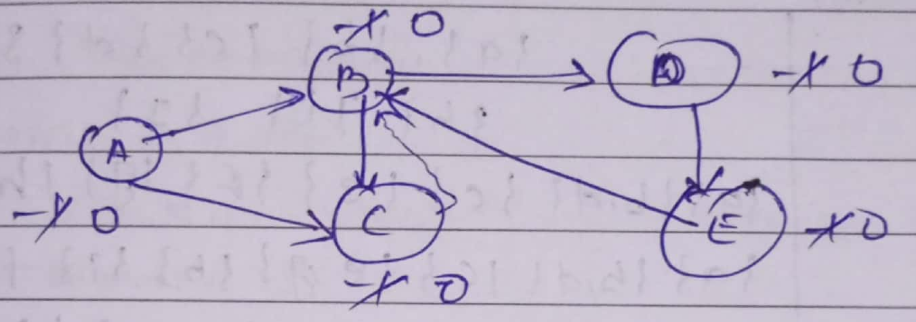
4 Ans:- Cycle detection in Undirected Graph (BFs)



Queue | A | B | C | D | E |

visited set | A | B | C | D

cycle dection in Directed Graph (DFS)



-1 = unvisited
0 = visited & in stack
1 = visited & popped from stack

5 Ans:- The disjoint set can be defined as the subset where the is no common element b/w the two sets Ex — S1 = {1, 2, 3, 4}
S2 = {5, 6, 7, 8}

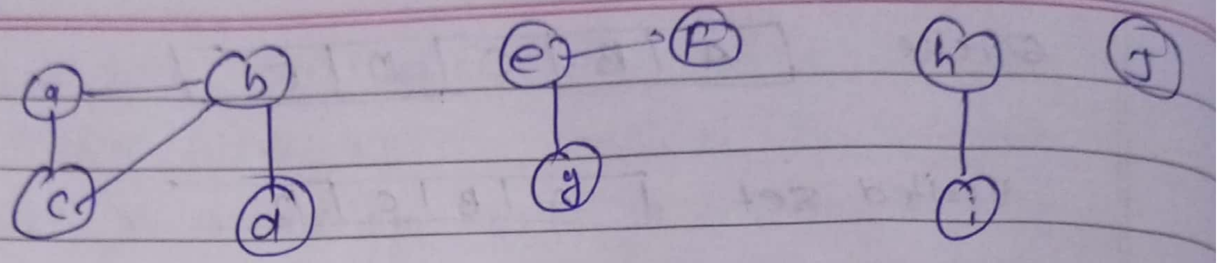
There are 3 operations which is performed a new element set contaning a new element

Finding the ~~o~~ repreesentive of the set containing a given element merging two set

10 individuals:
a, b, c, d, e, f, g, h, i, j,

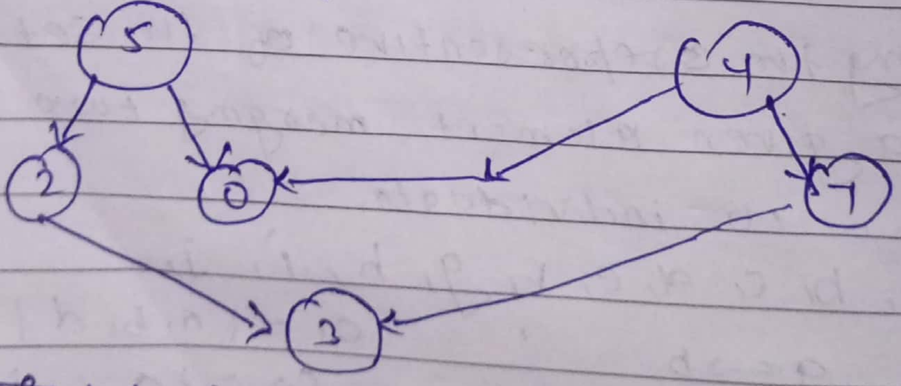
a ↔ b    G1 = {a, b, d}
b ↔ d    G2 = {c, f, i}
c ↔ f    G3 = {e, g, j}
c ↔ i    G4 = {h}
j ↔ e

7.



| Edge processed | Collection of disjoint sets |
|---|---|
| initial | {a}, {b} {c} {d} {e} {f} {g} {h} {i} {J}. |
| (b, d) | {a} {b,d} {c} {e} {f} {g} {h} {i} {J} |
| (e, g) | {a} {b,d} {c} {e,g} {h} {i} {J} {f} |
| (a, c) | {a,c} {b,d} {e g} {h} {i} {J} {f} |
| (h, i) | {a,c} {b,d} {e,g} {h, i} {J} {f} |
| (a, b) | {a,b,c,d} {e,g} {h,i} {J} {f} |
| (e, f) | {a,b,c,d} {e,f,g} {h,i} {J} |
| (b, c) | {a,b,c,d} {e,f,g} {h,i} {J} |

(8) Topological sorting:



Topological sort: 5, 4, 2, 3, 1, 0

4, 5, 2, 3, 1, 0

10

|  | Min heap | Max heap |
|---|---|---|
| 1 | The ascending priority | (1) The desecending priority |
| 2. | The smallest ele is to be papped from heap | The largest element is to be popped from the heap. |
| 3. | The smallest element has priority | The largest element has priority. |
| 4. | The minimum key elen present at the root | 4. The maximum key element is present at the root |