Q What is a graph ??

Network

Can be a physical (router, computer), countries flights

⇒ Network always has 2 entities.

1) Nodes
2) Edges

⇒ Graph is also a collection of nodes & edges.

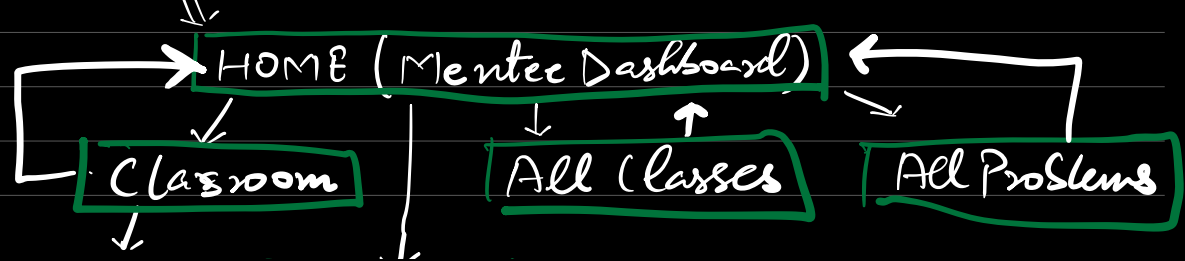
Q Why is graph important ??

⇒ Graph is the most relevant & versatile data structure.

Eg 1) Cloud Computing (network of computers)

2) Website

scaler.com



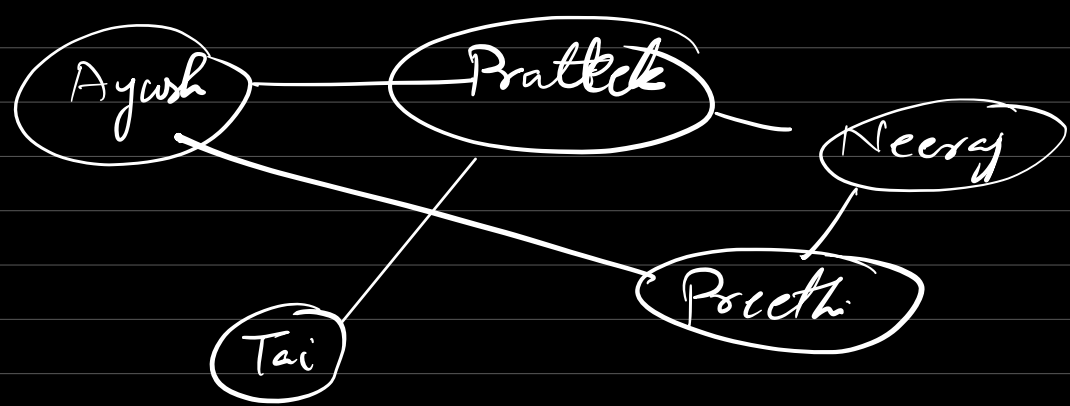
Live Class | Mentor

3) Social Media

- Facebook
- LinkedIn
- Instagram
- Twitter

Q What is node here ??

(Users)

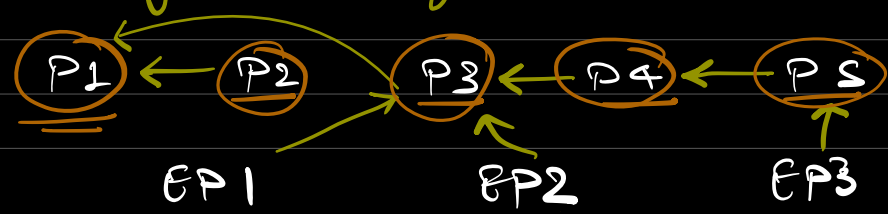


4) GPS

Most Common ⇒ Google Maps

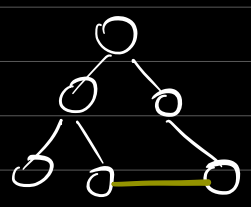
Node ⇒ All landmarks
Edge ⇒ Path b/w any 2 landmarks.

6) Project Management Tool (Maven)



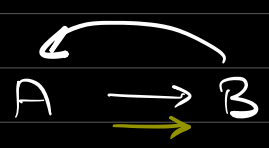
⇒ Perform dependency resolution

Diff b/w graphs & tree



1) scaler.com.

HOME ⟺ CLASSROOM ⟶ LIVECLASS



(Direction)

Directed graph

2) facebook

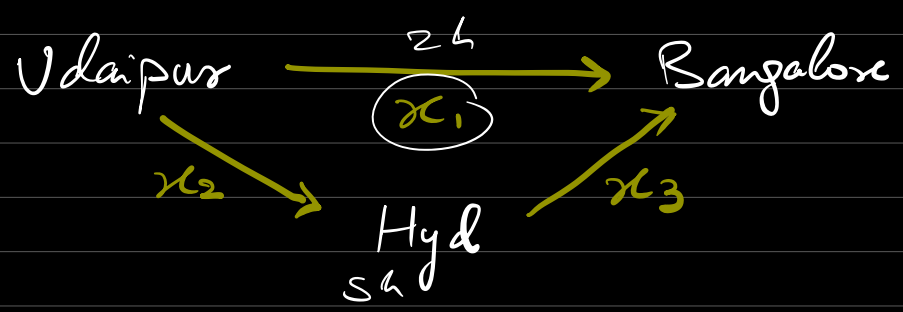
If Ram is a friend of Shyam.
Then Shyam is also a friend of Ram

(Undirectional)

Undirected graph.

Q Does Instagram have undirectional edges??

⇒ Flight Booking System.



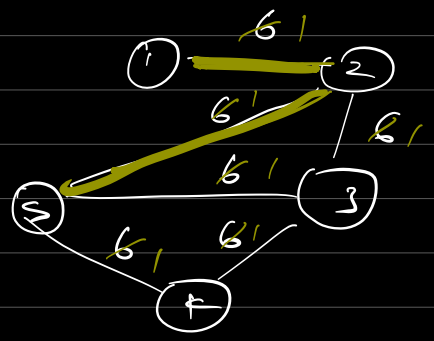
$x_1 >> x_2 + x_3$

(20K) (4K)

1) Price
2) Time

⇒ Weights
(Weighted graphs)

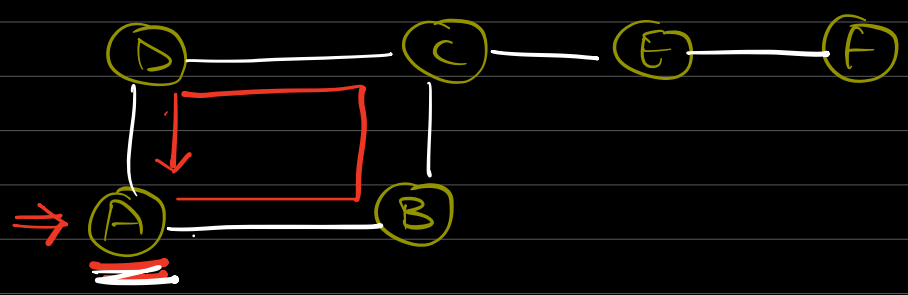
6×2

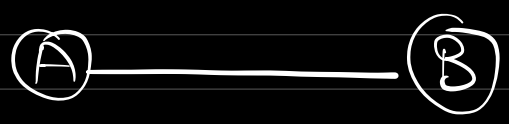


$\Rightarrow$ Weighted

Classification of Graphs

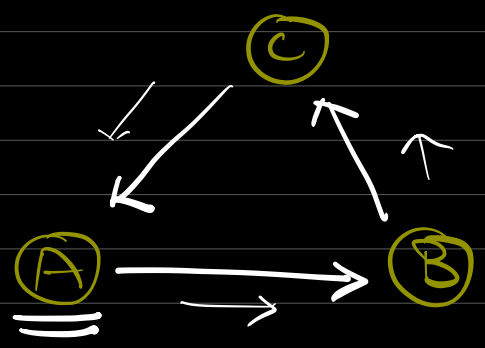
- Directed / Undirected
- Weighted / Unweighted



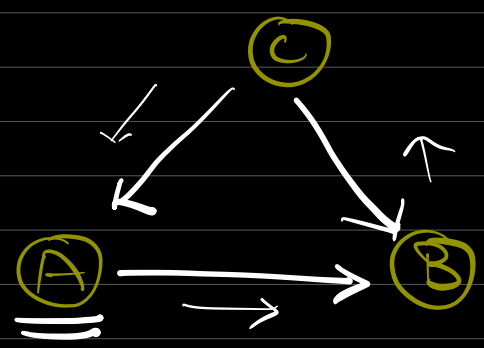


Undirectional.

~~Cyclic~~

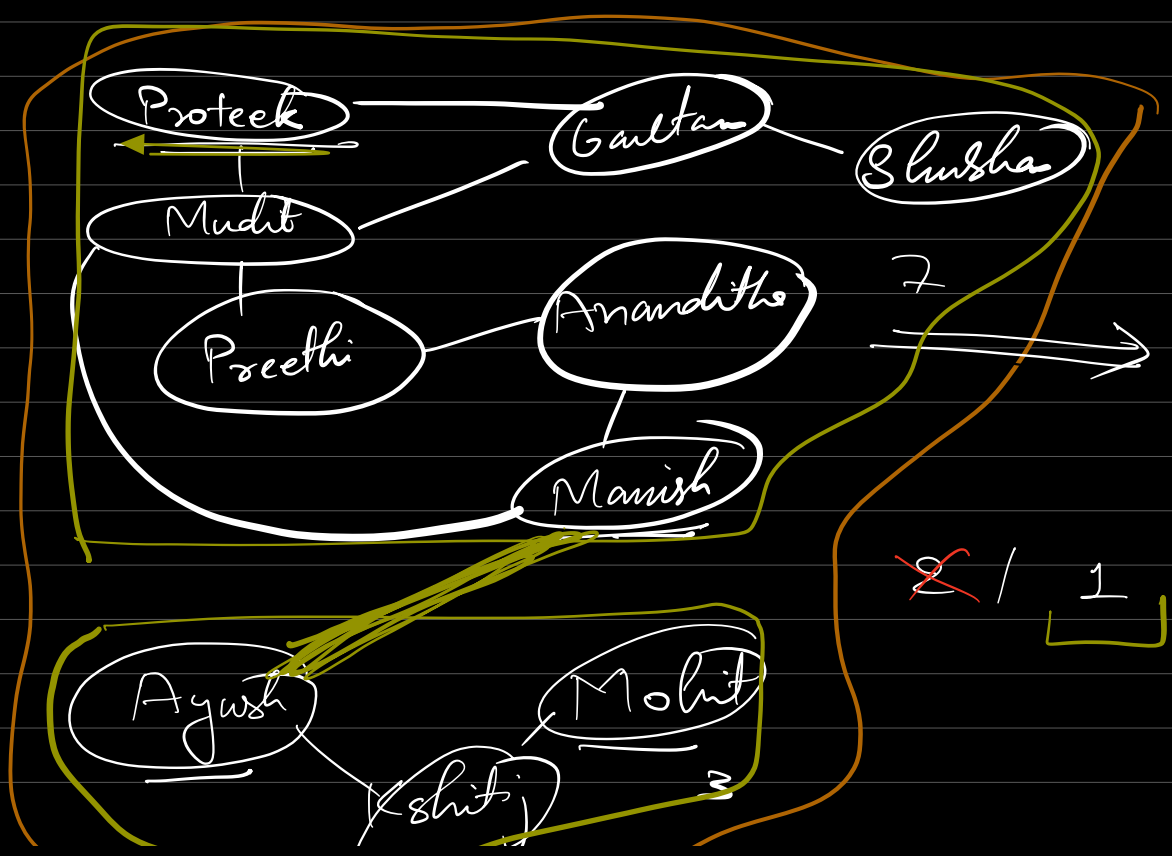


Yes



No

facebook



Graph g1 = new Graph

g1.add(7 nodes)

g2 = new Graph

g2.add(Arjun, Mahesh Bhat)

g3.

Connected Component

⇒ Every vertex should be reachable from every other vertex.

N ⇒ N CC

Representation of Graph in Code

1) Adjacency matrix $[V, E]$

$V \Rightarrow \{A, B, C, D, E\}$

1 2 3 4 5

5×5 matrix $[M]$

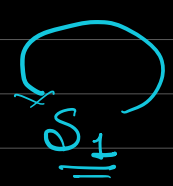
	1	2	3	4	5
	A	B	C	D	E
1 A	0				
2 B		0			
3 C			0		
4 D				0	
5 E					0

mirror image of upper quadrant

$M[i][j] = 1$ (If edge from node i to node j)

$1 \Rightarrow \omega$

Undirected.



$M[i][j] = 1$

$M[j][i]$?? $= 1$

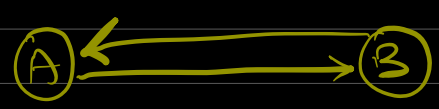
Q S.C. of AM $\Rightarrow O(V^2)$

0

Undirected Max no. of edges. (V vertices)

$$^{V}C_2 = \frac{(V)(V-1)}{2} = O(V^2)$$

Directed



$$^{V}C_2 \times 2 = (V)(V-1) = O(V^2)$$

* Pros of AM

1) $u, v \Rightarrow \underbrace{M[u][v]}_{O(1)}$ & $\underbrace{M[v][u]}_{O(1)}$

2) Add new edge $\Rightarrow O(1)$

3) Delete an edge $\Rightarrow O(1)$

Cons

1) N nodes $\simeq 10^7$

100 edges $\rightarrow (10^7) \times (10^7)$

$\Rightarrow$ When the no. of edges are less, sparse matrix is formed.

(not optimal by space)

2) Add a new node ??

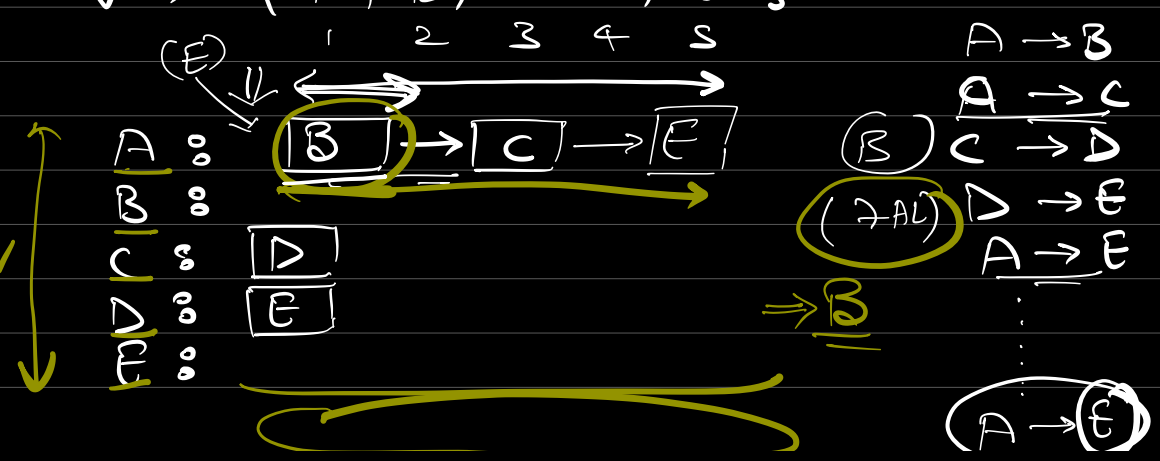
(i) new matrix $[V+1][V+1]$

(ii) Copy from old to new

$\Rightarrow O(V^2)$

2) Adjacency list (LL of edges)

$V \Rightarrow \{A, B, C, D, E\}$



$O(1)$ ✗ $\quad$ $O(V)$ ✓

Graph ⇒ V vertices & E edges

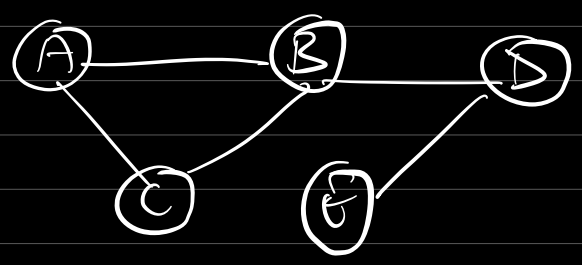
S.C. = $O(V+E)$

1) Object ⇒ 1) Reference to node
2) Weight.
3) Name

⇒ Pair

3) Pairs of nodes having an edge

N edges ⇒ $\langle u, v \rangle$

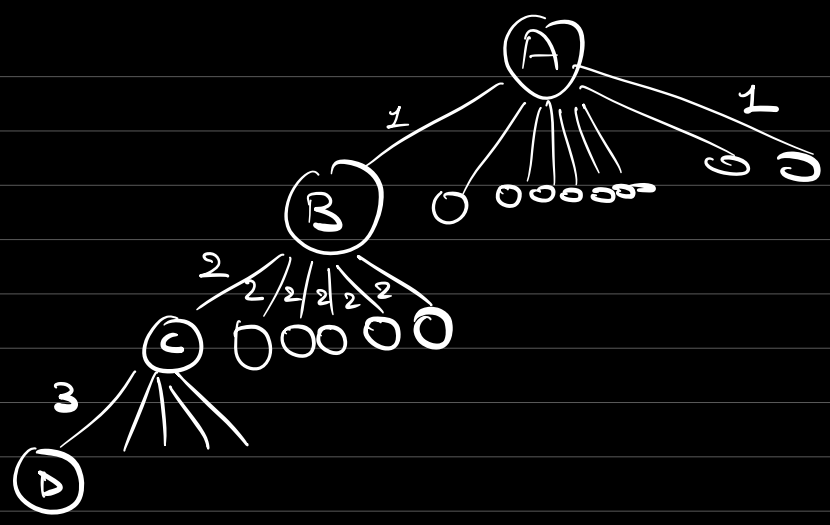


$\langle A, B \rangle$
$\langle B, D \rangle$
$\langle C, B \rangle$
$\langle C, A \rangle$
$\langle D, E \rangle$

Degree of Connection

linked In

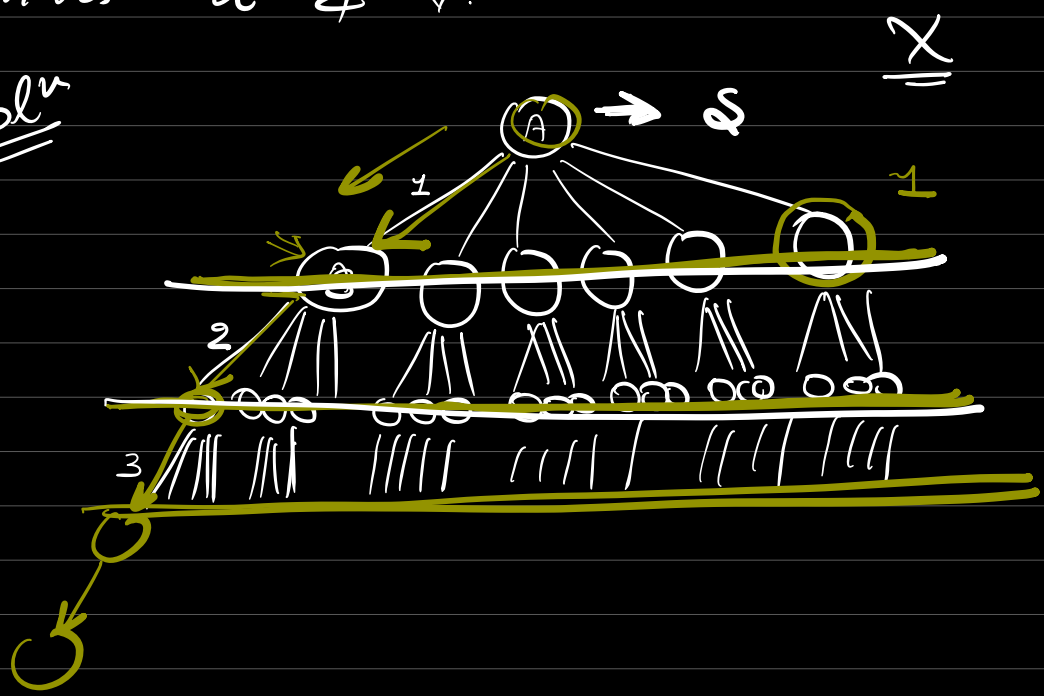
Profile (Degree of your relation to that person)



H.W: Given a graph of LinkedIn profiles.

⇒ find the degree of connection b/w 2 given nodes u & v.

Solⁿ



1) BFS (Breadth first Search)

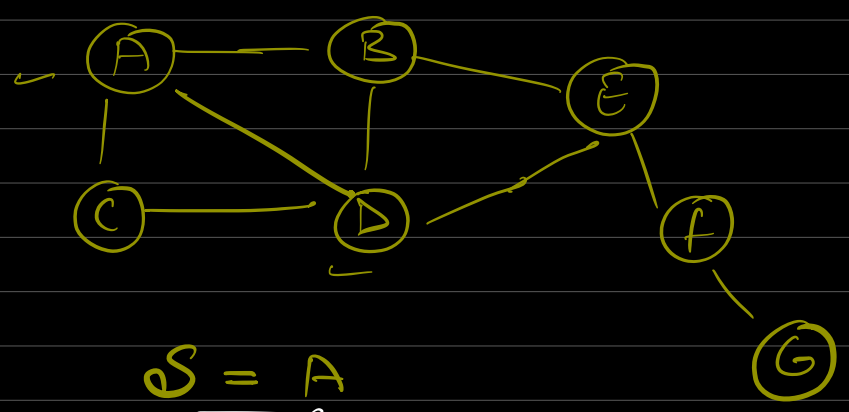
Queue

1) Start from source (Add source to queue) & mark as visited

2) While the queue is not empty

(i) Remove the first element

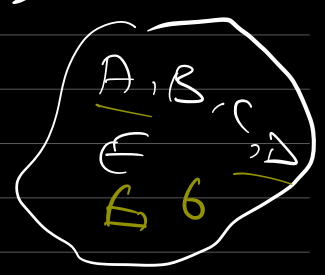
(ii) Add all the unvisited neighbours of the removed element to the queue. (also mark them as visited)



Visited Nodes

1) Visited Array

2) HashSet



S = A

Q ⇒

G ⇒ ~~E~~

Pseudo Code

```
bfs (root) {

    q ⇒ queue;
    q.push (root);
```

```
visited.add(root);
while(!q.isEmpty()){
    v = q.dequeue();
    for(all u connected to v){
        if(!visited.contains(u)){
            visited.add(u);
            q.push(u);
        }
    }
}
}
```

V, E

$T.C. = O(V+E)$

2) DFS (Depth first Search)

⇒ Recursion

⇒ Select one neighbour, & explore all paths

& then backtrack & explore next neighbour.

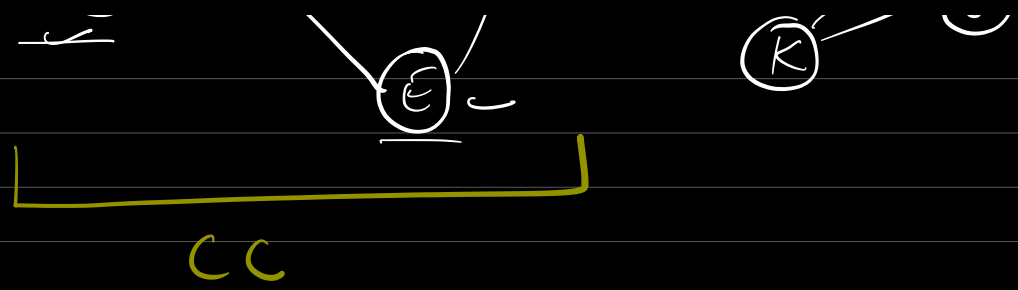
Dfs pseudo Code

```
Dfs (v) {
    mark v as visited;
    for (all u connected to v) {
                (AL of v)
        if (! visited.contains(u)) {
            Dfs(u);
        }
    }
    process (v);
}
```

Q If we call Bfs or Dfs on one node. will we visit all Nodes of the graph.

NO

⇒ They might not be connected.



Q How to solve??

⇒ Call for all nodes. (loop)

```
for all vertices v :
        if (! visited.contains(v)) {
                dfs(v);
        }
```

H.W. 1) Return the total no. of connected components of a graph.

2) Return the no. of elements in each CC.