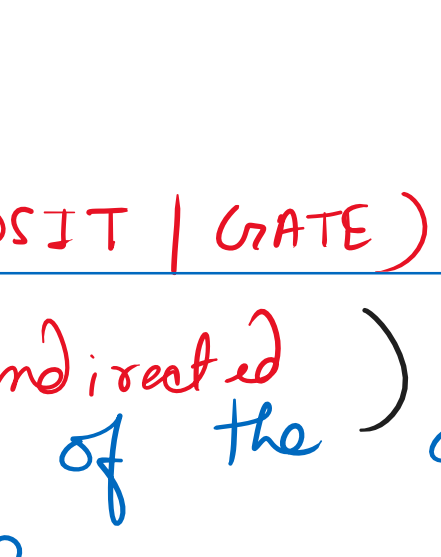
Graph Interview Questions (Frequently Repeated) :→

* Convert A Given Adjacency Matrix To AdjacencyList

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



function (int[][] mat, int V)
{

} → (list)

{ int V = 5 }
Number of Nodes

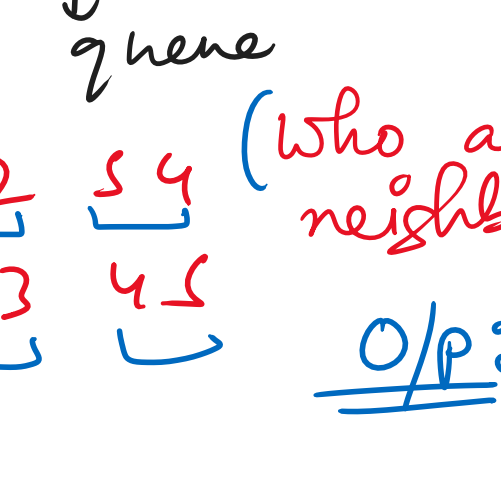
u → v
v → u
a[u][v] == 1

Node		Neighbours
1	→	2, 3, 4
2	→	1, 4
3	→	1, 4, 5
4	→	1, 2, 3, 5
5	→	3, 4

(JSSATE | RNS | SJBIT | DSIT | GATE) (2024/2025) (Synamedia OTT)

* Given a graph (undirected) calculate the number of nodes & edges of the given graph :→

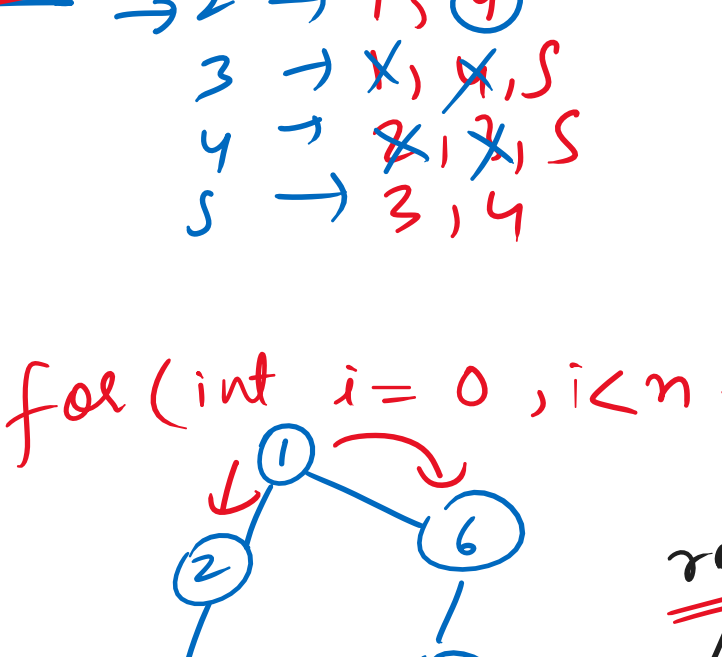
Node 0 → {1,2}
Node 1 → {0,2,3}
Node 2 → {0,1,4}
Node 3 → {1,4}
Node 4 → {2,3}



Graph Traversals (DFS & BFS) ⇒ Cycle Detection

Src node, start node

DFS ↓ recursion; BFS ↓ queue



1 | 3 2 | 5 4 (Who are your neighbours :)
1 | 2 3 | 4 5

O/P: 1, 2, 3, 4, 5 (L0, L1, L2)

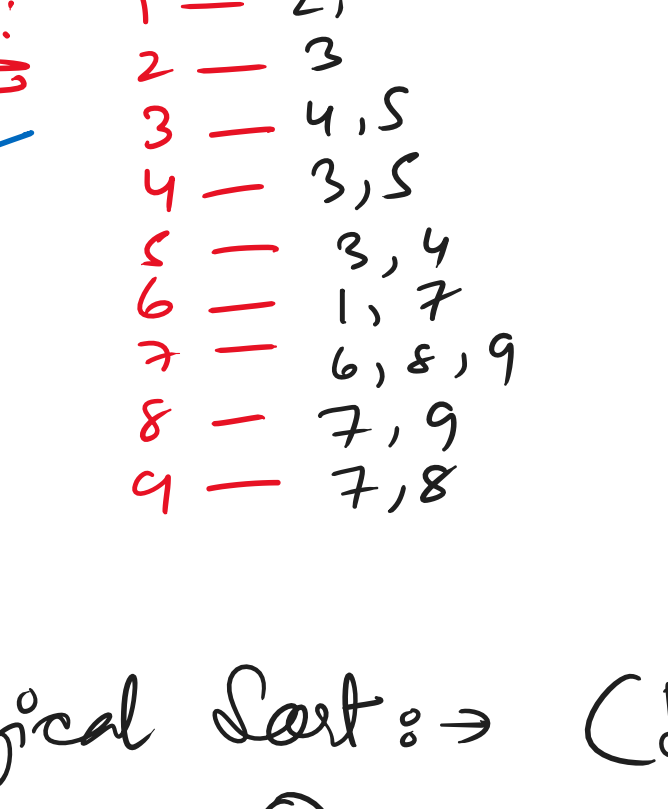
adj list:
1 → 2, 3
2 → 1, 4
3 → 1, 4, 5
4 → 2, 3, 5
5 → 3, 4

Vis[ ]

1	F
2	F
3	F
4	F
5	F

queue

for (int i = 0, i<n, i++)



dfs(1) ... dfs(6) → dfs(7) → dfs(8) → dfs(9)
recursion
dfs(2) → dfs(3) — dfs(4) — dfs(5)

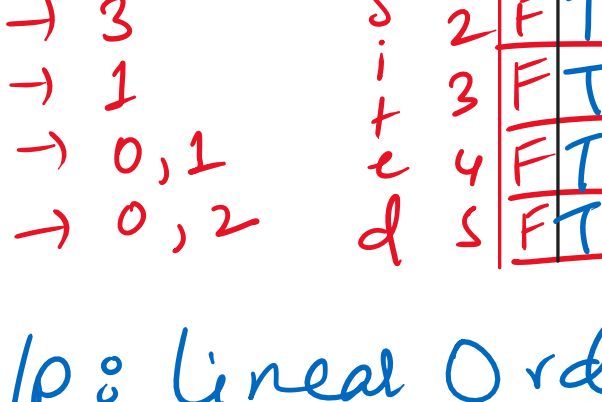
O/p — 1, 2, 3, 4, 5, 6, 7, 8, 9

Adj List:
1 — 2, 6
2 — 3
3 — 4, 5
4 — 3, 5
5 — 3, 4
6 — 1, 7
7 — 6, 8, 9
8 — 7, 9
9 — 7, 8

Visited

1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Topological Sort :→ (Directed Acyclic Graph) (DAG) (u → v)



Definition :→ Linear Ordering of nodes such that if there is an edge from "u" to "v", in the linear ordering "u" always comes before "v".

Adjacency List: (We do a component-wise traversal) :→

0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 0, 2

visited

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

for (int i=0; i<n; i++) {
dfs (i)
↳ push(i)
}

dfs(0), dfs(1), dfs(2) → dfs(3), dfs(4), dfs(5)

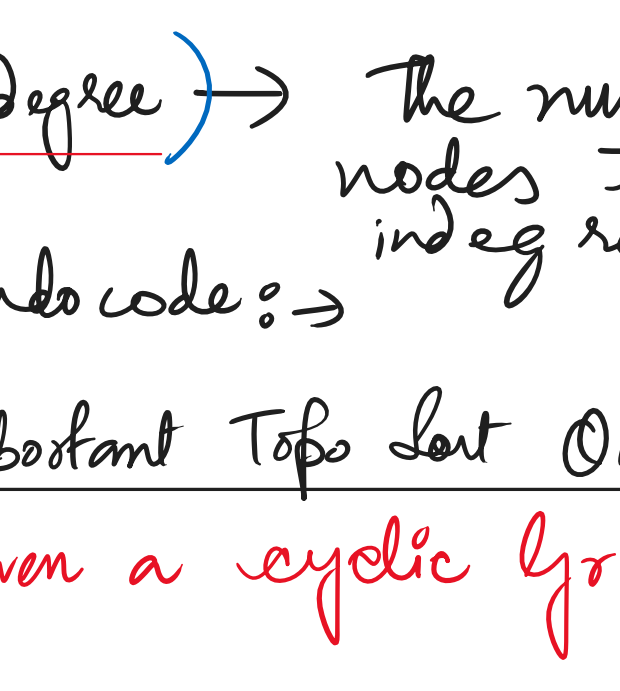
Stack: 5, 4, 2, 3, 1, 0 — After dfs(i) finishes ⇓ Stack

O/p: Linear Ordering : (5 4 2 3 1 0) [TS == N]
5 4 2 3 0 1

u → v
2 → 3
3 → 1
4 → 1
5 → 2
5 → 0
4 → 0

Topological Sort :→ (BFS) Kahn's Algorithm :→

5 0 2 3 1



Step 1: Insert all nodes with indegree 0 into the queue
Step 2: Take out the nodes from the queue & reduce the indegree of neighbours.

Queue: 4, 5, 0, 2, 3, 1 (queue for BFS)

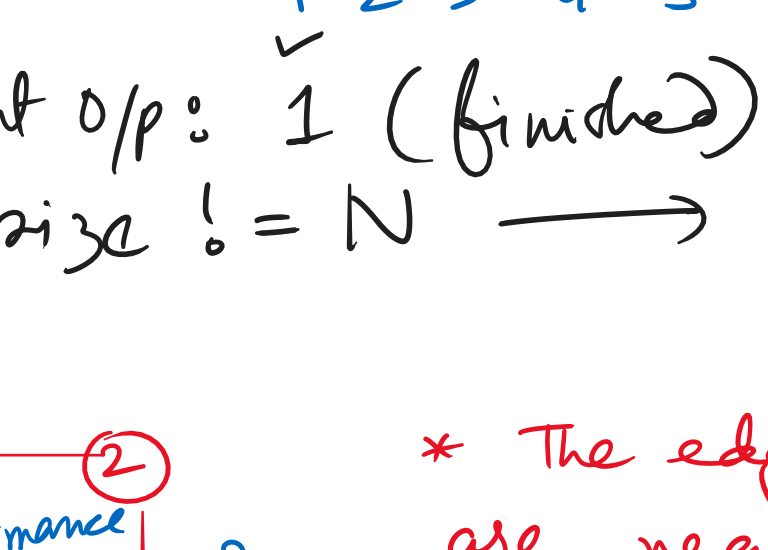
Adj List
0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 0, 2

(Indegree) → The number of incoming nodes to a particular node is it's indegree.

pseudocode :→ if (node u → v) indegree[v] ++;

Important Topo Sort Question :→

Given a cyclic Graph, detect cycle using Topo Sort :→



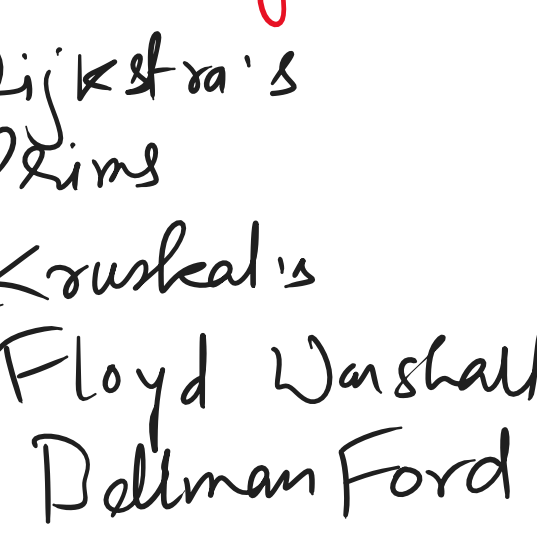
Adjacency List
1 → 2
2 → 3
3 → 4, 5
4 → 2
5 →

Indegree:

1	2	3	4	5
0	2 (1)	1	1	1

empty — queue for BFS

Topo Sort O/p: 1 (finished)
if TS.size != N → cycle



* The edge weight for 1→2 & 2→4 are negative :→

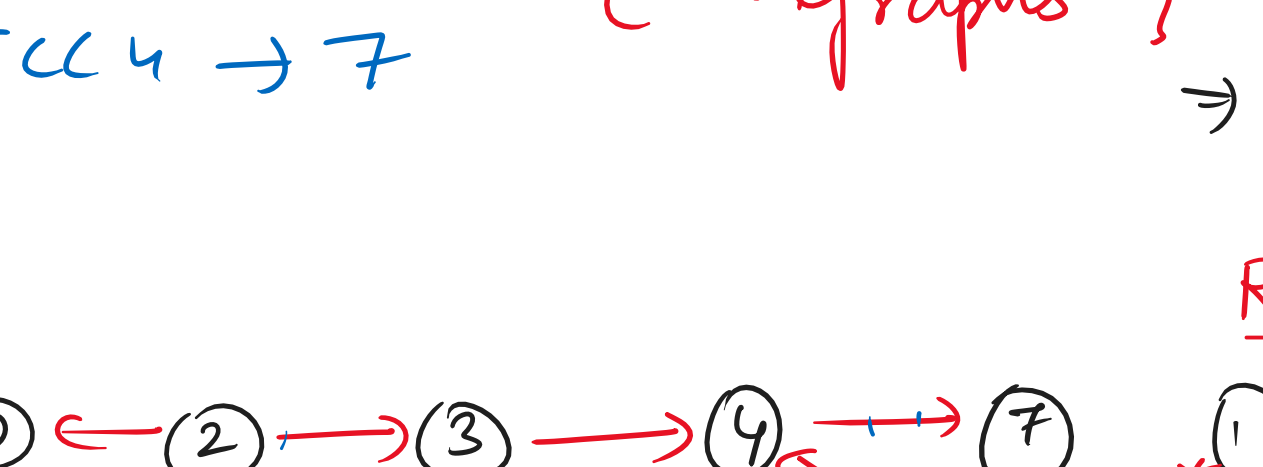
{In case of sales, marks, performance a graph can be negative}

Single Source Distance Algos {
1. Dijkstra's
2. Prims
3. Kruskal's
4. Floyd Warshall
5. Bellman Ford → (-ve)

Problems :→
1. Number of Provinces
2. Number of Islands

Strongly Connected Components (SCCs) :→

[Kosaraju's Algo]

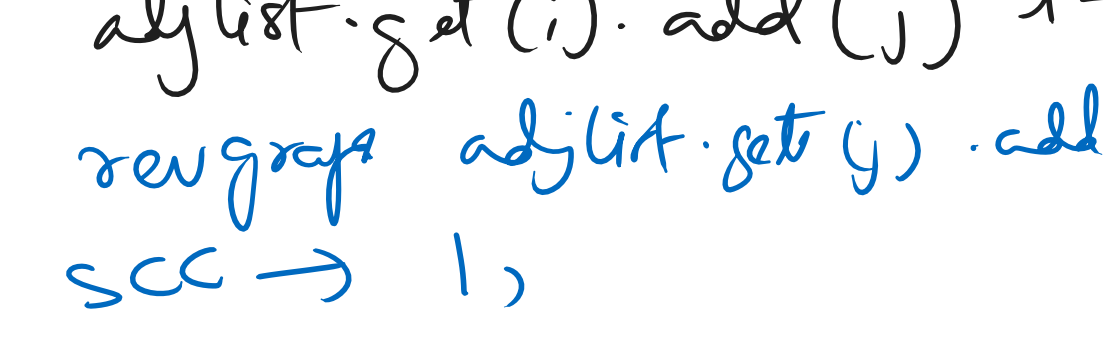


⇒ An scc is a component in a graph where we can travel from each node to every other node in any order.
→ A single node is also an SCC

SCC1 → 0, 1, 2
SCC2 → 3
SCC3 → 4, 5, 6
SCC4 → 7

{ only applicable for directed graphs }

Rules / Steps :→



1. Sort all the edges according to their finish time. (dfs)
2. Reverse the Graph

3. Do a dfs call

Stack (top): 0 ✓, 1 ✗, 2 ✗, 3 ✓, 4 ✓, 5 ✗, 6 ✗, 7 — (RG)

dfs(0), dfs(3), dfs(4)

adjlist.get(i).add(j) i → j
revgraph adjlist.get(j).add(i);
scc → 1,