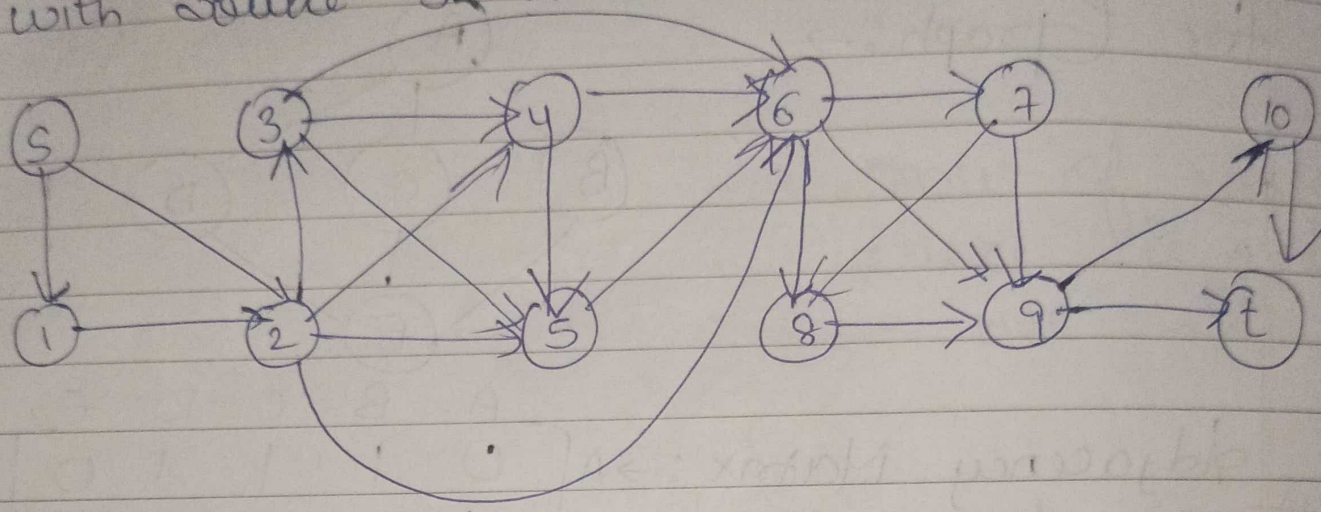
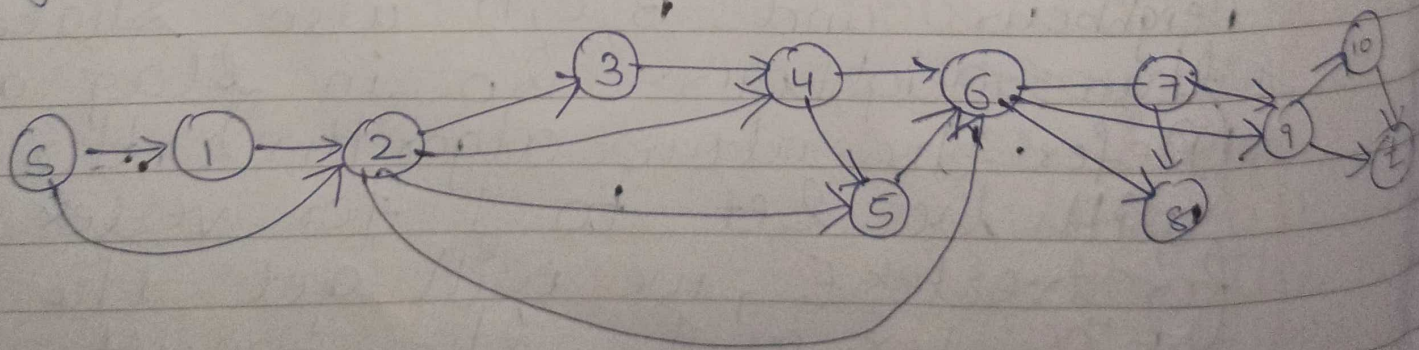
4) G is a directed Acyclic Graph (DAG) with source s and sink t.

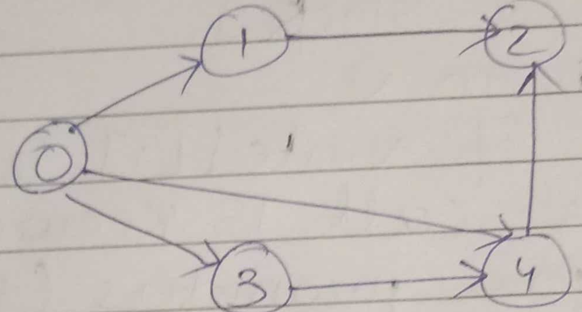


for the given above problem we will compute pre and post of all node and then sort them using topological sort. (by post)

After applying topological sorting the graph looks like this :->

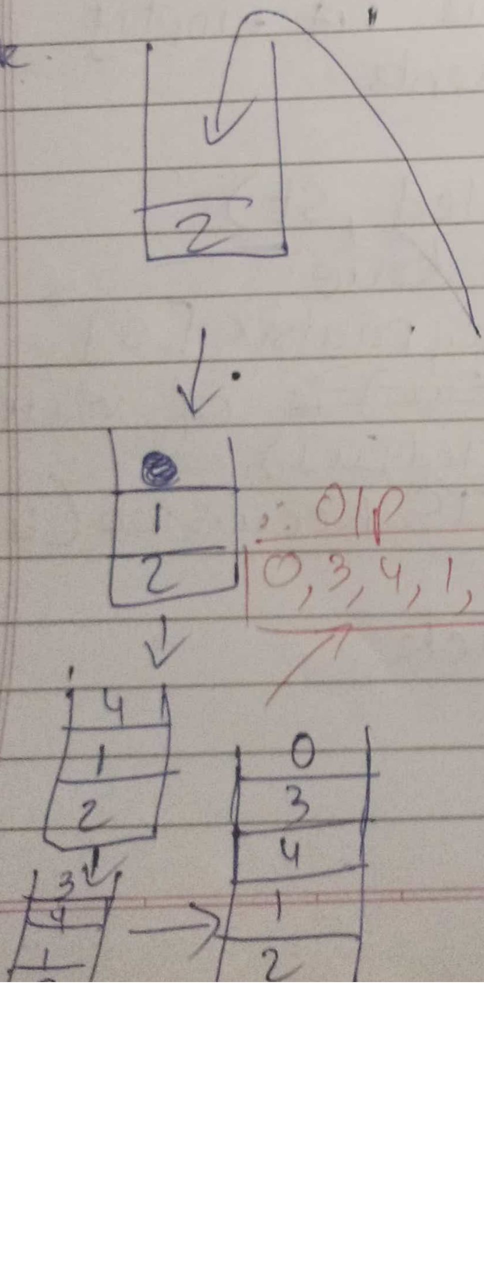


Eg:



* 'u should come before v.

Stack



∴ O/P
0, 3, 4, 1, 2

~~→ Once a~~

→ ① Lets start with ① node. we reach a dead end when we reach ②.

→ Now push ② to stack.

→ Back track to the parent node and push the child elements in that order

→ Theres another child from 0 ie 4 (0 → 4) ∴ Push 4

→ and there is one more child which goes from 0 → 3 → 4 ∴ 4 comes before 3

→ Since 4 is already present, push 3

Push root node 0 → and then pop in same order

Algorithm:→

adjL → Adjacency Matrix of each vertices
MaxVertex → Stores the farthest edge of the current Vertex

CutVertex ()
try all vertex u, if u hasn't visited, topo(u)
topo(u), initiate num[u] = low[u] = topoCount
try all neighbour v of u
if v is free, topo(v)
low[u] = min(low[u], low[v])
check the Condition
else low[u] = min(low[u], num[v])

topo(u)

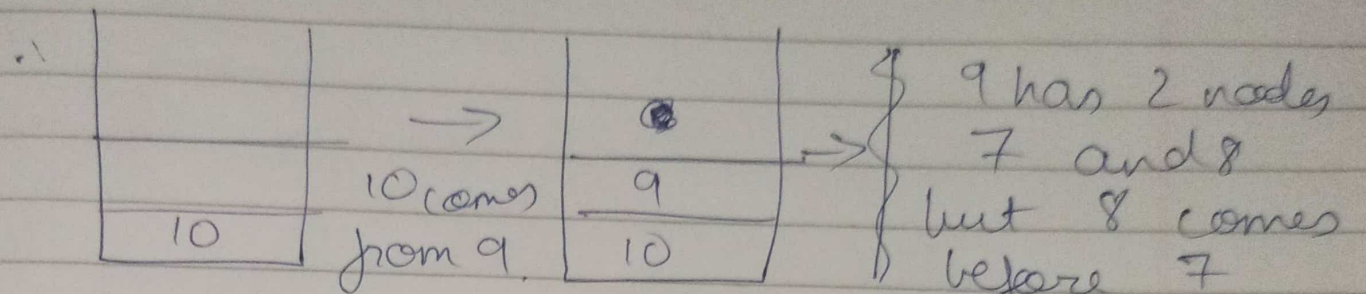
1. Create visited array and intialize it to false
2. Create a stack S
3. loop i to V
4. Check if visited[i] is false
5. If yes, call TS_Recursion function and pass parameters (i, visited, S)
6. Loop till stack is not empty and pop the elements

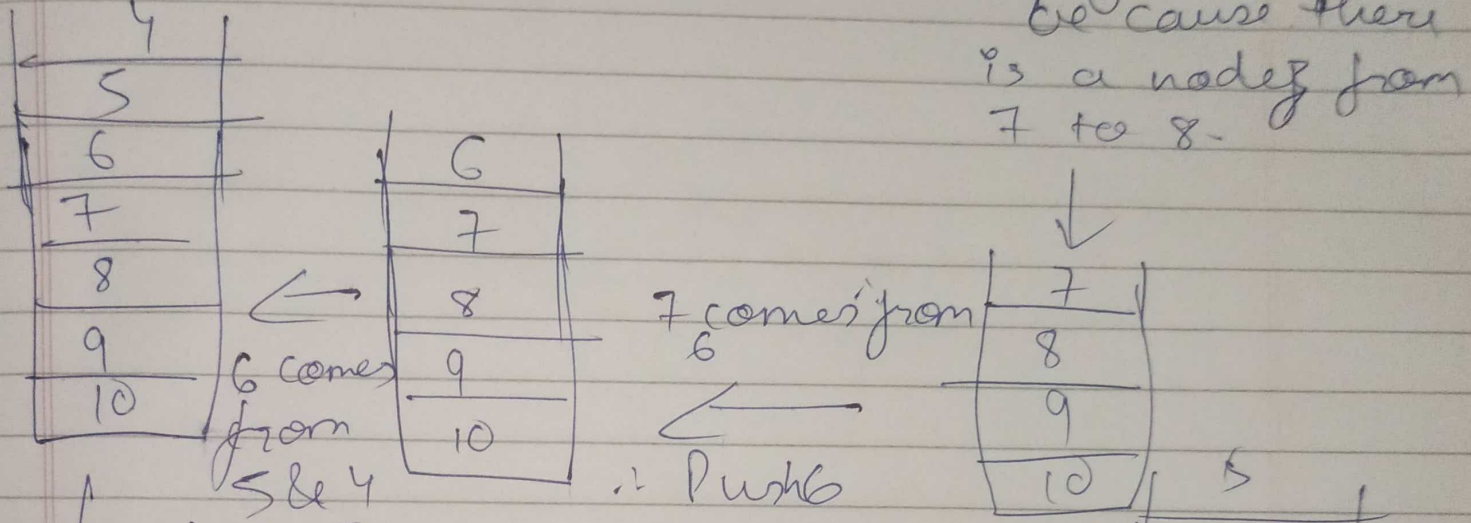
TS_Recursion (S, visited, St)

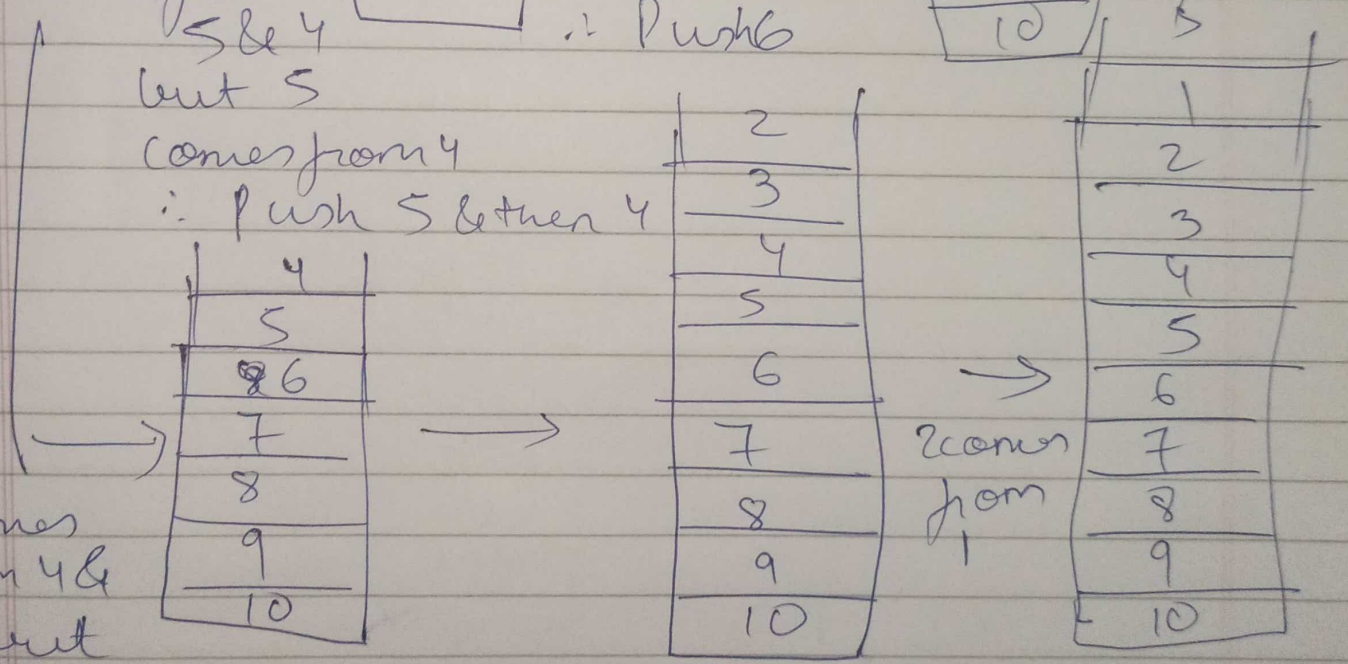
1. set visited[S] to true
2. Loop till adjacency matrix [S]
3. check if visited ~~[u] is~~ of vertex is false (i.e. !visited[u])
4. Recursively call TS_Recursion(u, visited, St)
5. push s into stack

Topological Sort of example given

5 → Dead end at 10.







Pop everything in this order

∴ O/p

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Time Complexity :->

$|V|$ * Time to remove vertex from G and adding back the vertex + Time to perform topological sort

$= O(|V| * |V| + |E| + |V| + |E|)$

$= O(|V|^2 + |V||E|)$