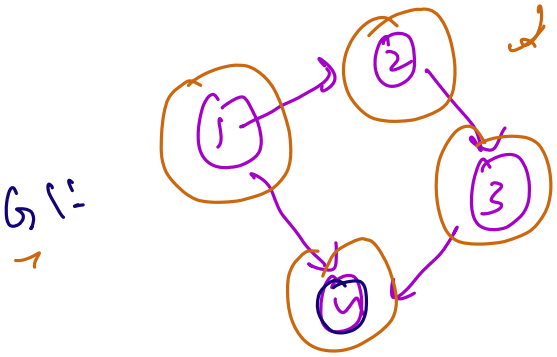
Graphs Followup

Agenda

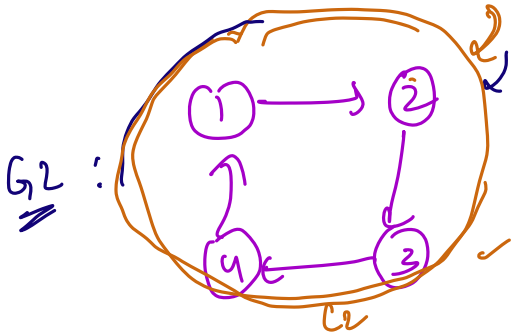
1) Kosaraju's Algorithm
2) Bellman Ford (SSSP, works for -ve edges)
3) Detect cycle in Directed
4) 1-2 problems

Question: Find no. of strongly connected components [Directed graphs]

Strongly connected Graph :



Ans = 4 ✗



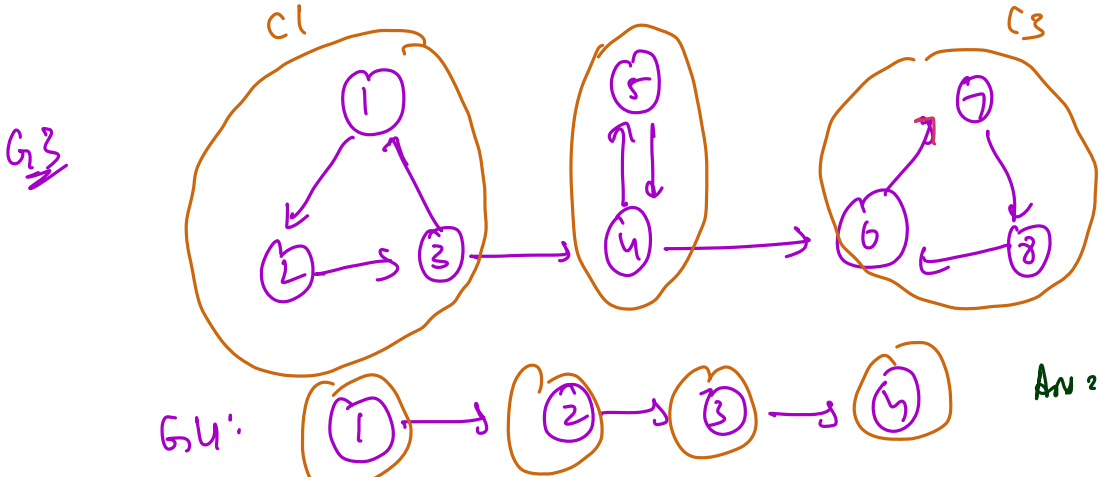
Ans = 1 ✓

(5, 1)

u → v
v → u

1 → 5
5 → 1

Ans = 3

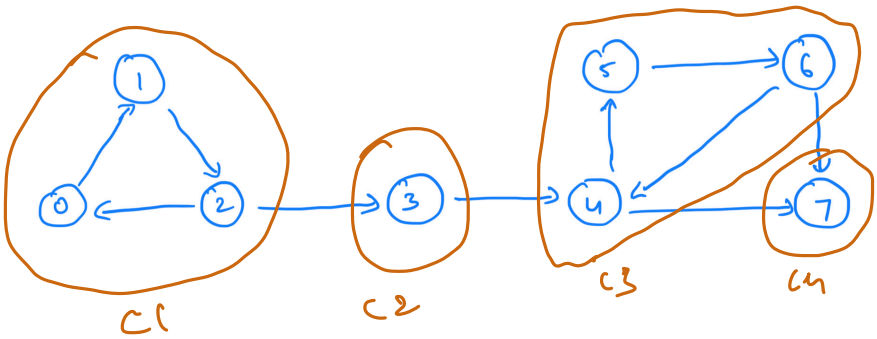


5 → 1 !

Ans = 4 ✓

Max No of S.C.C with 'V' vertices = V

Strongly Connected Graph: For every pair of vertices (u, v) there exists a path from $u \to v$ and $v \to u$.

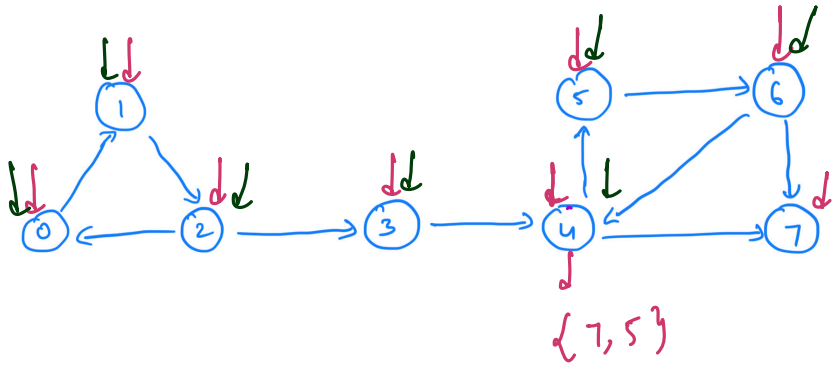


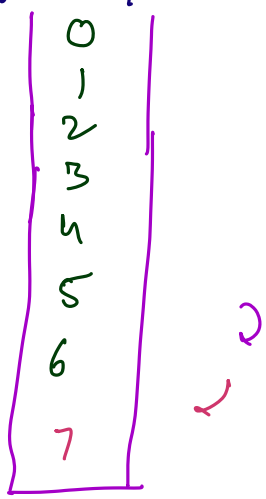
dfs = 4

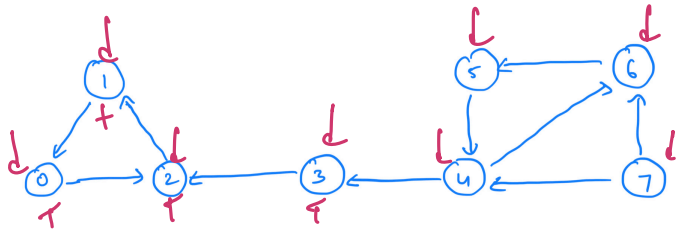
```
for(i=0; i<V; i++)
  if(!visited)
    DFS(i);
```

Algo

1) Apply DFS on every unvisited node
2) when a node is processed, then push it on top of stack. [Topo logical]
3) Reverse all the edges of the Graph
4) pop elements from the stack and apply DFS on it
5) No. of DFS calls = = No of SCCs







SCC1: 0 2 1

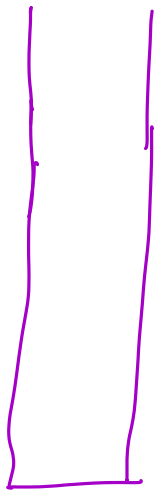
SCC2: 3

SCC3: 4 6 5

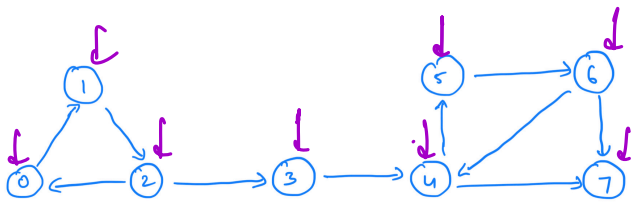
SCC4: 7

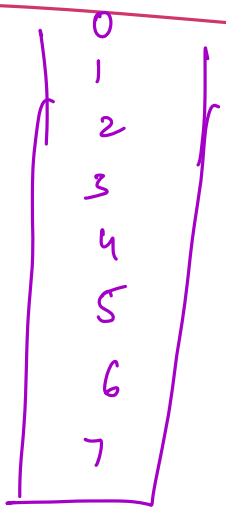
} 4 SCCs

DFS Calls



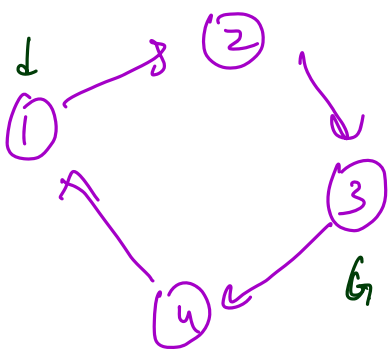
stack.



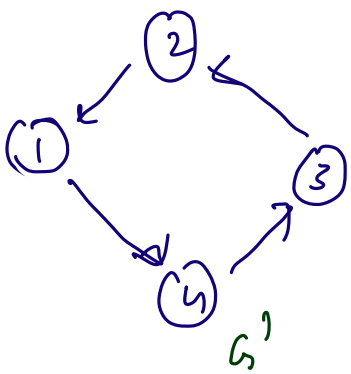


stack1

Observations:

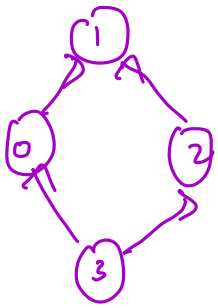


G

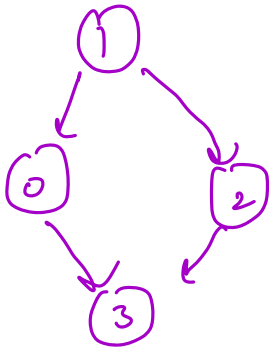


G'

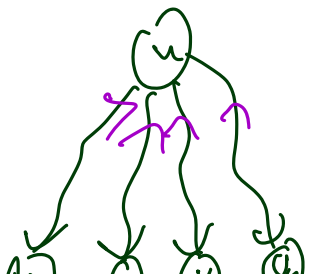
If I reverse all edges of a S.C.G, is the new graph still S.C.G?

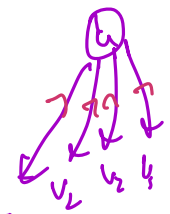


Reverse



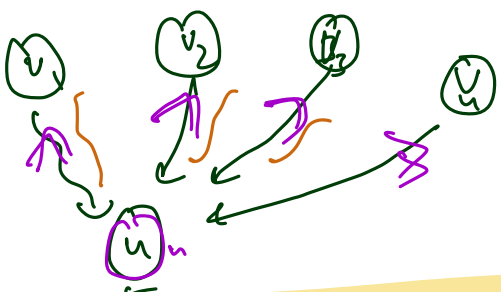
→ For every vertex 'u' in G_1, there is a path to every other vertex 'v' in the graph





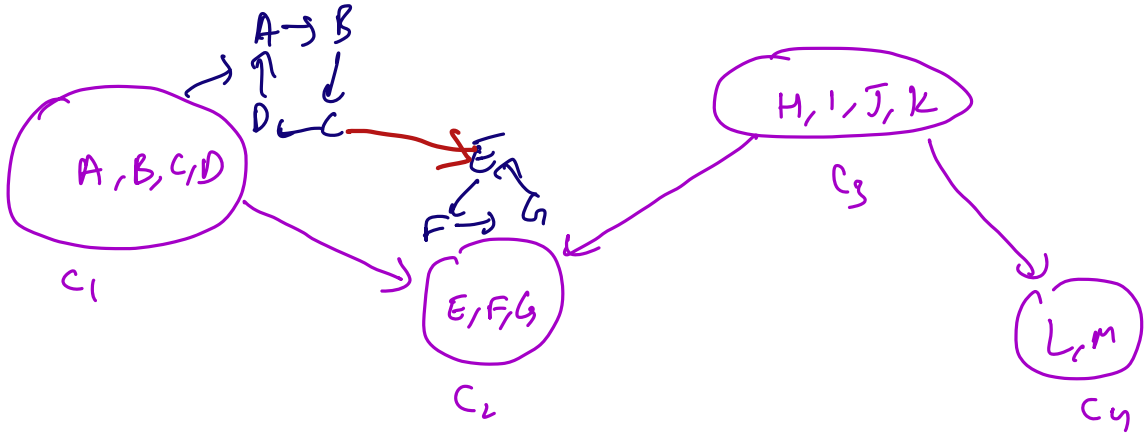
G^1

P1 → There is a path from every vertex 'u' in G^1 to vertex 'v' from vertex 'v' in the graph G.

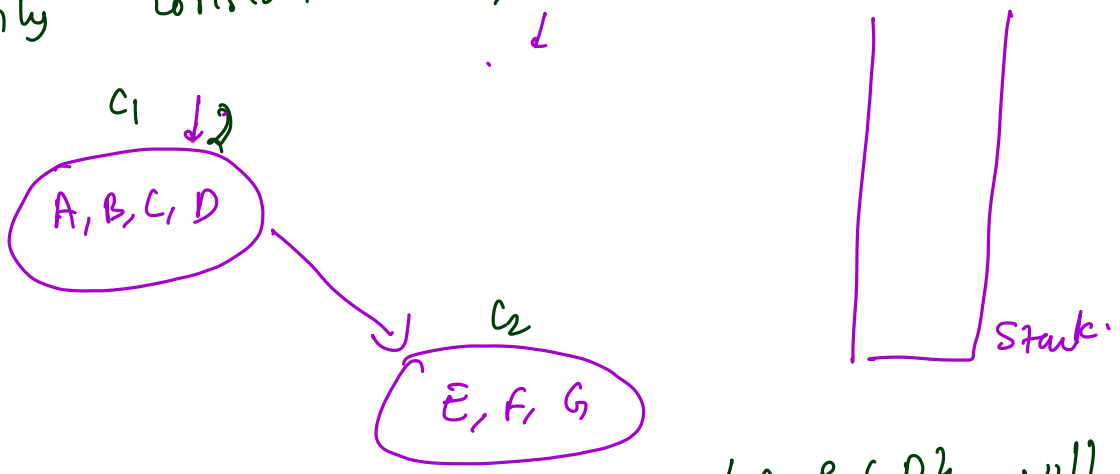


→ There is a path from u to all other vertices in G^1

G^1 is also Strongly Connected Graph

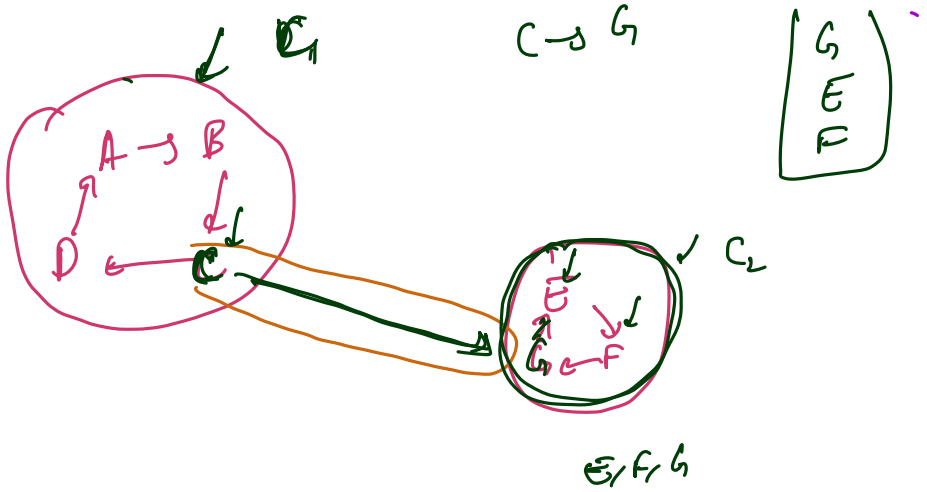


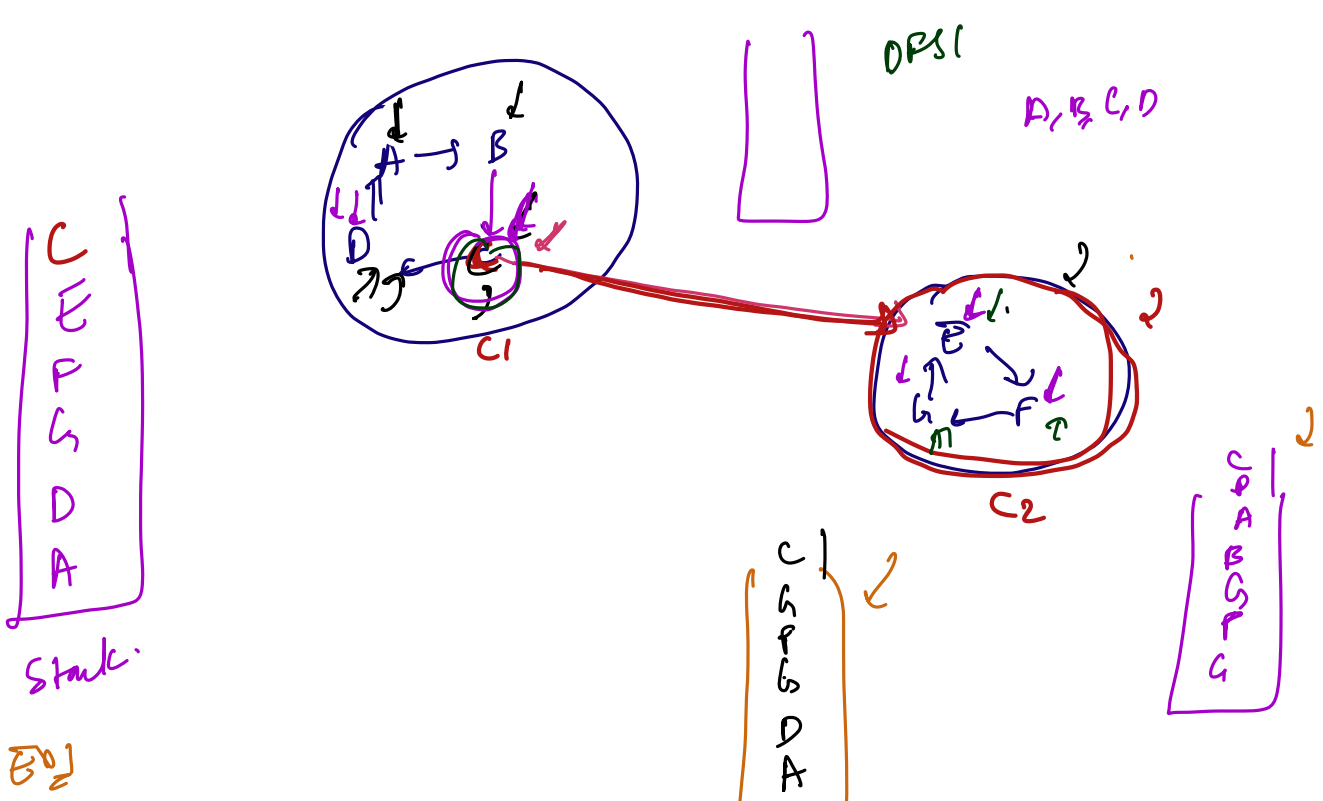
Lets only consider C_1, C_2



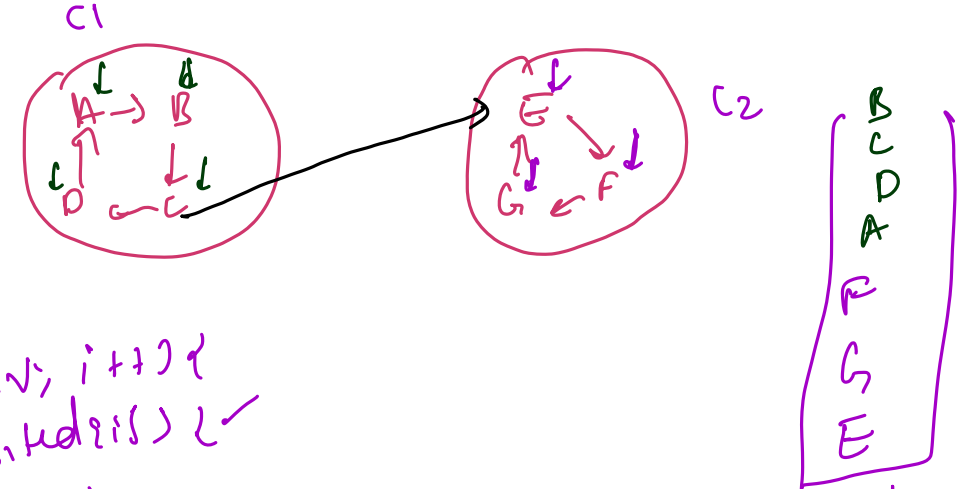
Claim: One of the vertices of C_1 $\{A, B, C, D\}$ will come on top of all E, F, G on the stack

Case 1: start DFS on a node from C_1



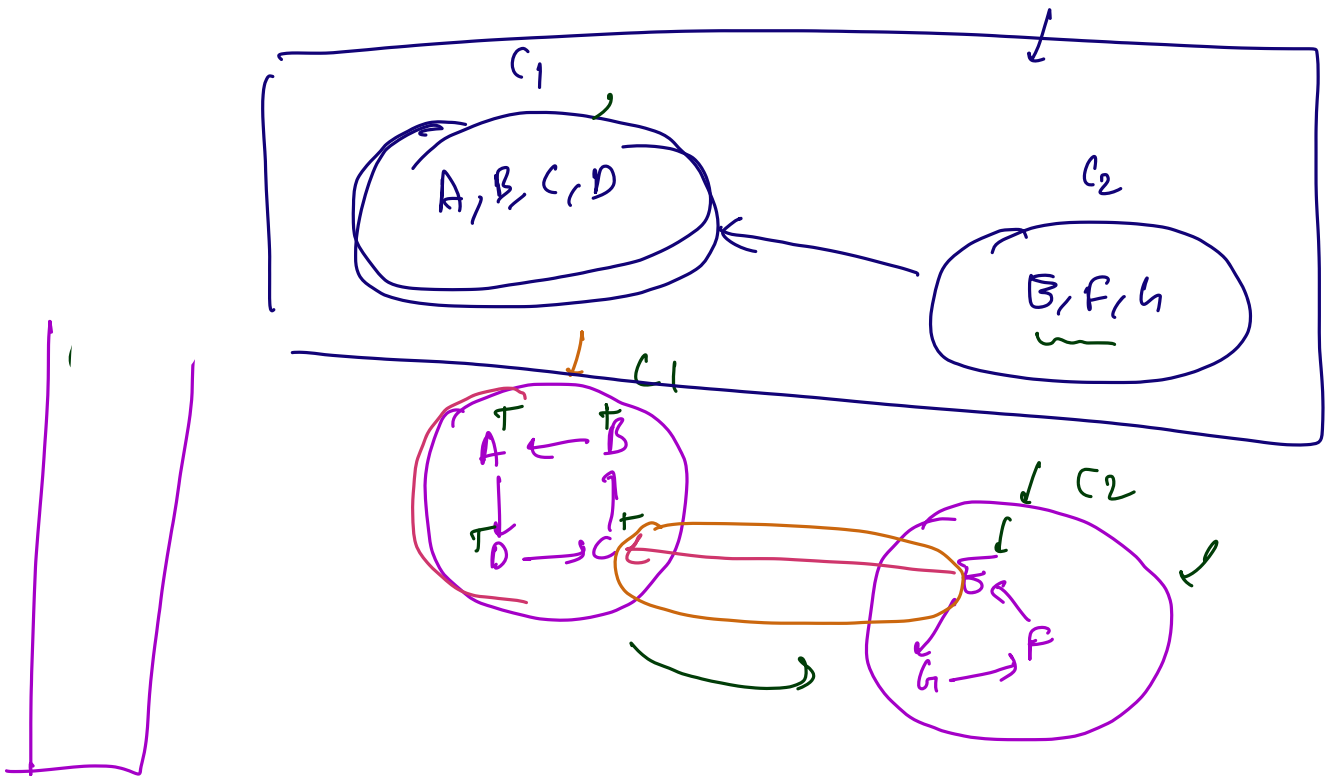


Case 2 : We call DFS on node of C2



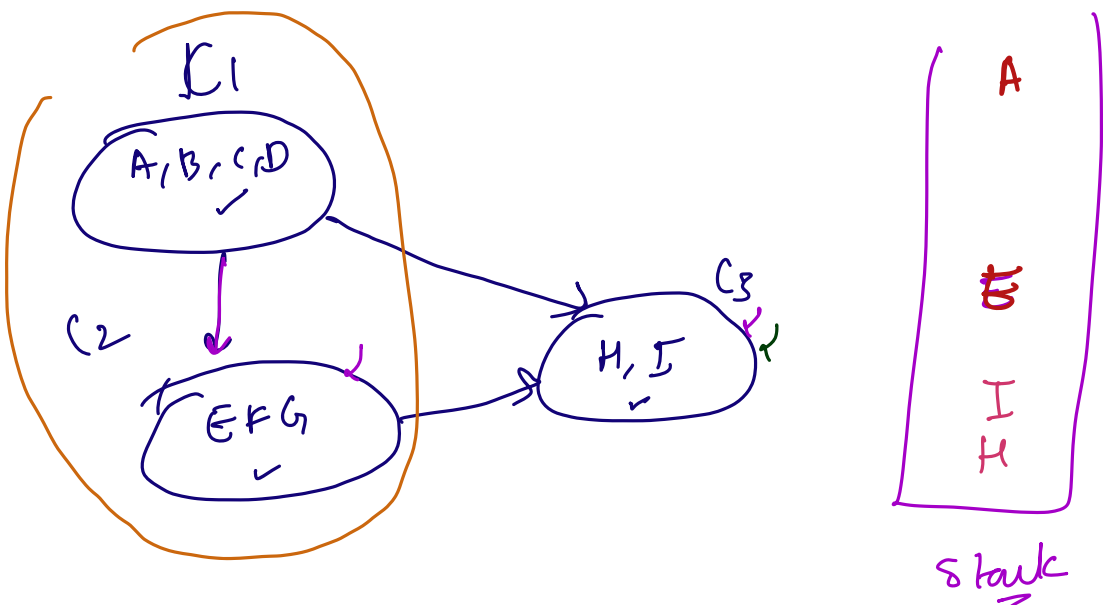

```
for(i=0; i<V; i++){
    if(!visited[i]){
        DFS(i);
    }
}
```

→ Reverse the Edges



C1: A B C D

C2: E F G



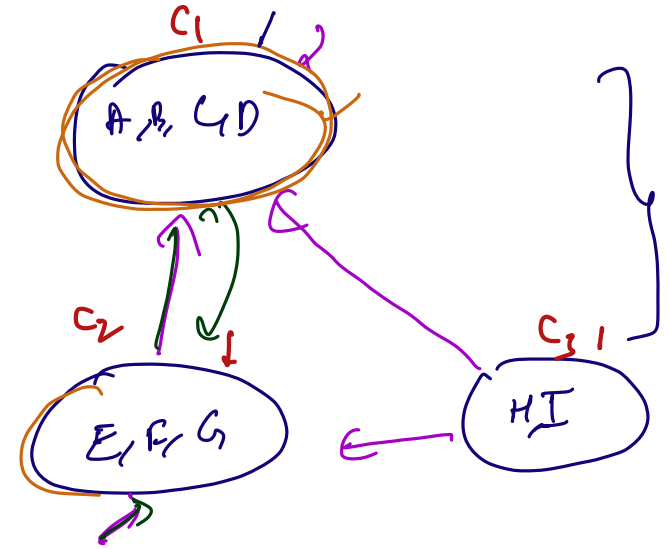
$C_1 \rightarrow C_2$,
$C_1 \rightarrow C_3$

★ → Top of the stack would be from C_1

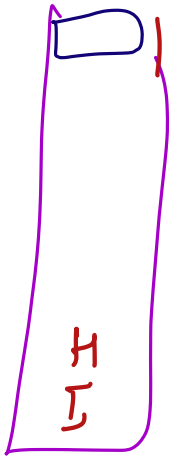
$C_2 \rightarrow C_3$

→ One of the vertices {E/F/G} will def be on top of H, I in the stack

for (i=



A, B, C, D

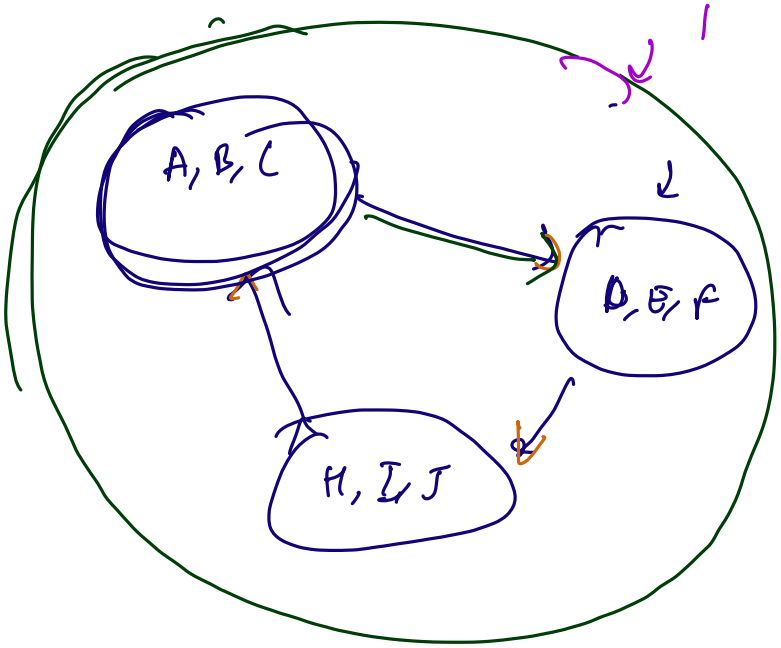


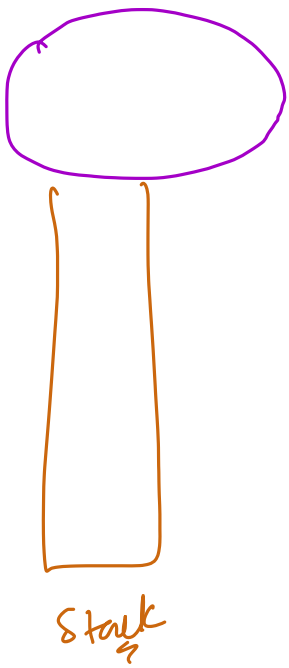
SCC1 : A, B, C, D

SCC2 : E, F, G

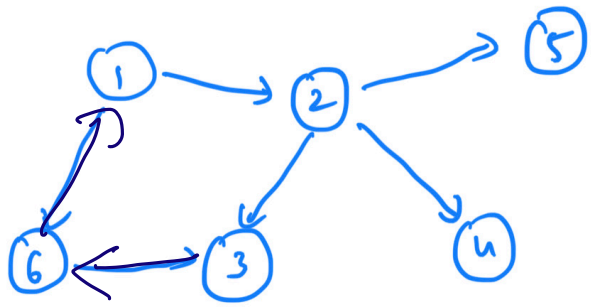
SCC3 : H, I

} 3 S-C-CS

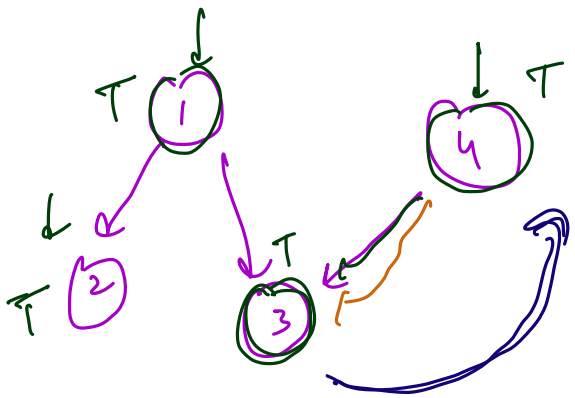




Question: Cycle in Directed graph



→ In undirected graph, if we visit any node which is already visited & not its parent ⇒ Cycle



Brute force:

If I start DFS on any node v, and come back to it ⇒ CYCLE

T.C: $O(V+E) \times V \Rightarrow O(V^2 + V \cdot E)$

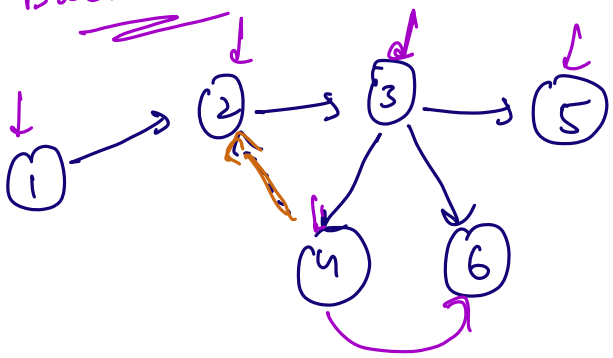
Approach: Indegree Approach

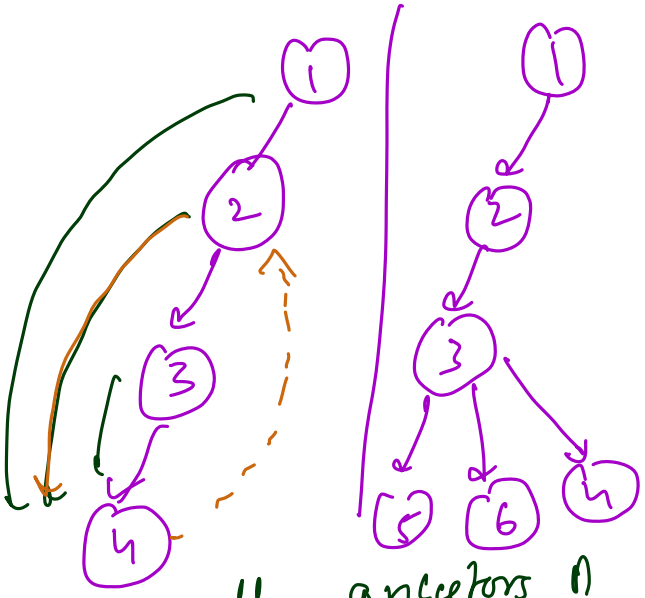
Kahn's Algo to find Topo Sort.

→ If all the vertex are not visited, then it is cyclic.

Approaches :

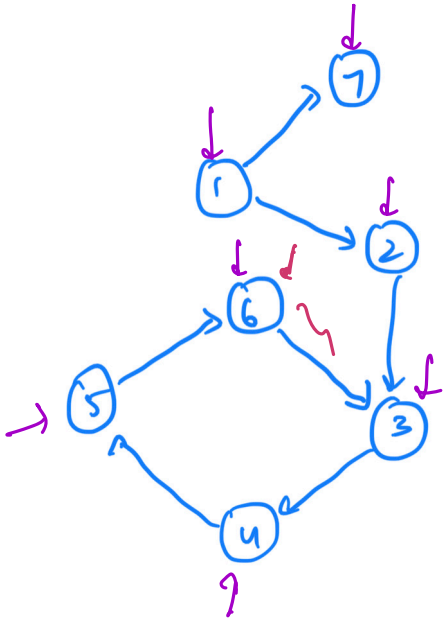
Back Edge :





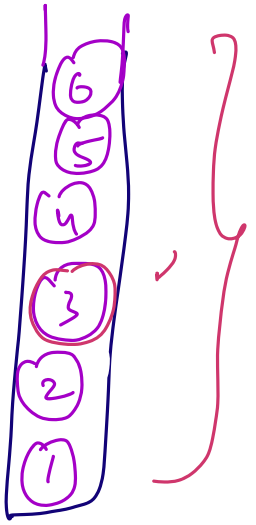
→ there exists a path from all ancestors of 4 to the node 4

Back Edge : If there is an edge from a node to one of its ancestors, its called Back edge.



3 → 6

6 → 3



Recursion Stack

How to keep track of a vertex is in Recursion stack or not

→ Set
→ Boolean array which denoty whether an element v in stack or not.

```
for(i=0; i<V; i++){
    if(!visited[i] && isCycle(i))
        return true;
}
return false;
    bool isPresentStack[] = {false};
bool isCycle(int v){
        visited[v] = true;
        isPresentStack[v] = true;
        for(i : adj[v]){
            if(!visited[i] && isCycle(i))
                return true;
            else if(isPresentStack[i] == true)
                return true;
        }
        isPresent[v] = false;
        return false;
}
```

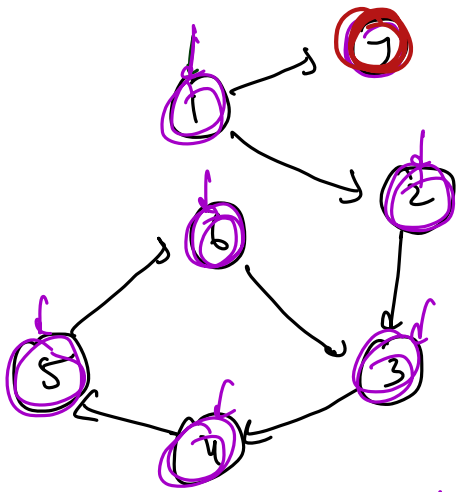
Implementation 2 [Using Colours]

Black : Unvisited Vertices

Red : Vertices whose DFS is completed

Purple : vertices whose DFS is in progress

0, 1, 2



If we have an edge from any colour vertex to a purple vertex ⇒ CYCLE

T.C : O(V+E)

Bellman Ford Algorithm

Single Source shortest Path.

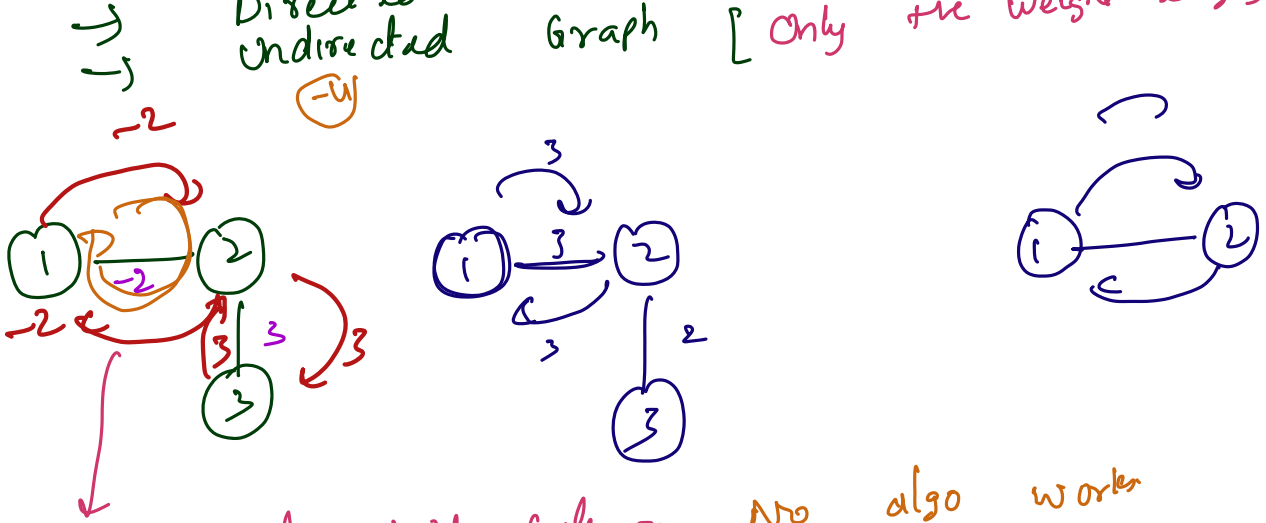
Dijkstra ⇒ Does work with -ve Edge

⇒ Bellman Ford ⇒ works with -ve edges

→ No -ve Edge weight cycle

Directed Graph [Works for all]

Undirected Graph [Only the weight edges]



Negative Edge weight Cycle ⇒ No algo works

Algorithm :

We will do in stages

Stage 1 : Find shortest from src to all other vertex with atmost 1 edge involved

Stage k : with atmost k edges involved

dist [V] = {INF}

dist [src] = 0 }

Algo:

1) Relax all the edges V-1 times

Relax: Edge (u, v)

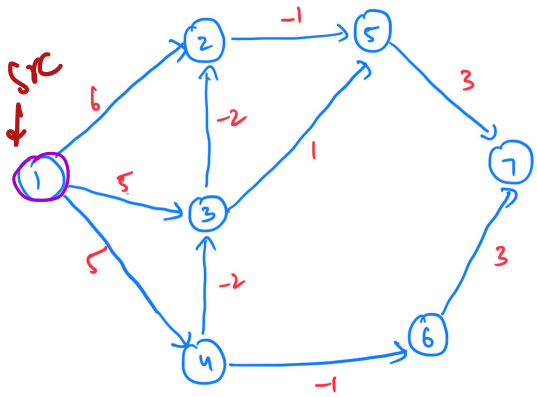
if (dist[u] + w(u,v) < dist[v])

dist[v] = dist[u] + w(u,v)

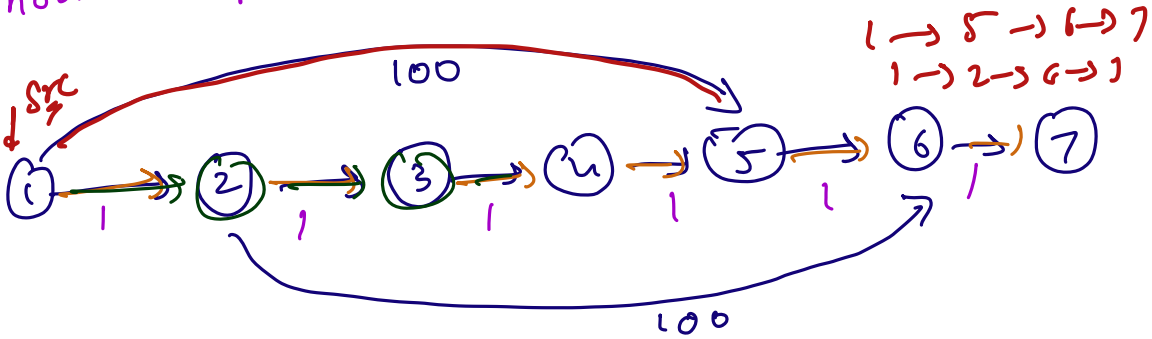
} Floyd warshall

2) dist array would contain the shortest distances from src to all other vertices

dist[i][v] + edge[v][j] < dist[i][j]



What is max. number of edges in the shortest path between any 2 vertices?



→ In worst case, the shortest path goes through all the other vertices → (V-1) edges

→ Will we visit the same vertex multiple times? NO!

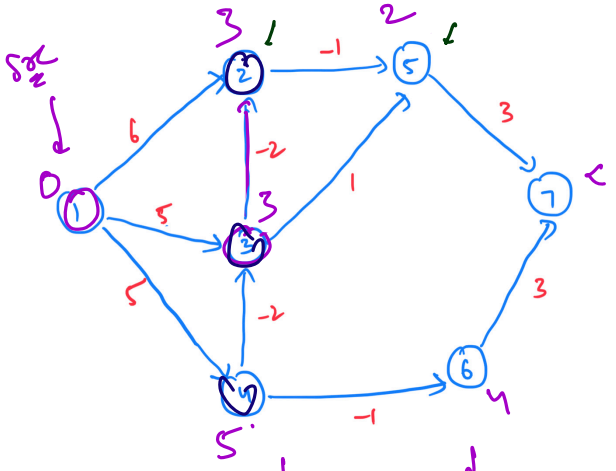
Shortest Path = 6

Observation:

Shortest path to any node will have atmax V-1 edges

Relax 6 times.

Iteration 1 : u-5 (5,7) (6,7) (4,6) (4,3) (3,5) (2,5)



$dist[u] + w(u,v) < dist[v]$

$\{ dist[v] = dist[u] + w(u,v)$

1 → 3 → 2 ⇒ 2 edges

1 → 4 → 7 → 2 ⇒ 3 edges

Edges: (5,7) (6,7) (4,6) (4,3) (3,5) (2,5) (1,4) (1,3) (1,2)

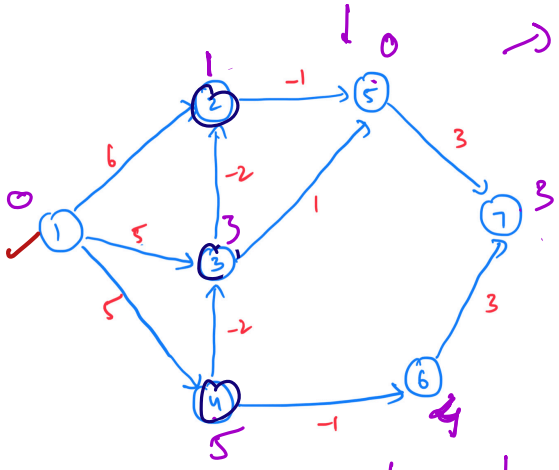
$5 + (-2) < 5$

(1,4)
dist[1] + 5
0 + 5 < dist[4]

After Iteration 2, we have shortest path to all vertices with atmost 2 edges involved

After k iterations, we will have the shortest path to all vertices with atmost k edges involved.

Just do V-1 iterations



→ 1 Iteration

Iter1: ✓

Iter2: ✓

Iter3: No value changed

↓ Terminate

#Iters = 6

Topo Sort: 1 4 3 2 5 6 7

O(V+E)

(1,2) (1,3) (3,2) (1,4) (4,3) (2,5) (3,5) (4,6) (5,7)

(6,7)

→ the order of Edges made the difference

→ If the graph is Directed Acyclic Graph

→ Do a Topological sort

→ Consider edge based on Topo-Sort

Relax (V-1) times {

O(E) { for(i=0; i<Edges..

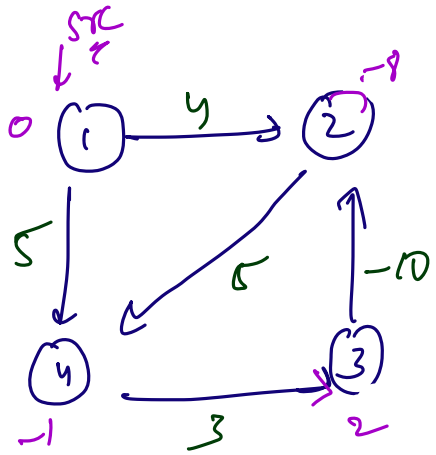
}

T.C: $O((V-1) \times E)$

$O(V \cdot E)$

*DAG: $O(V+E) + E \Rightarrow O(V+E)$

Bellman Ford — Detects Negative Edge Weight Cycle



V = 4

iterations = 3

iter1 : ✓

iter2 : ✓

✓ iter3 :

iter4 :

(1,2) (1,4) (2,4) (4,3) (3,2)

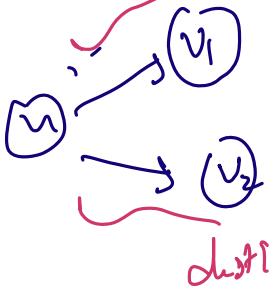
In V^{th} iteration, if dist of any vertex decreases, Negative Edge Weight

DP based

Optimal Substructure :

For stage K, I'm using the results of stage K-1 [smaller subproblems]

Overlapping subproblems : distinct



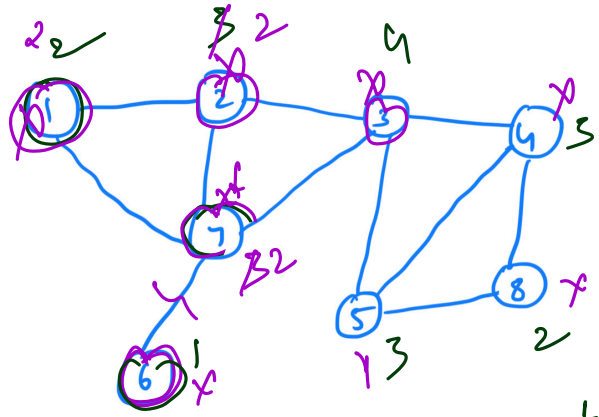
DP

8919159928

Linked In

Question: No. of people who can attend a party

→ there are 9 people



↙ Friendship

K = 1 {1, 2, 5, 4, 5, 6, 7, 9}

K = 2 except 6 & 9

K = 3 No one

→ A person can attend a party only if he gets K of his friends

K

{1, 6, 2, 5}

{1, 5, 7}

if (indegree[i] < K)

queue