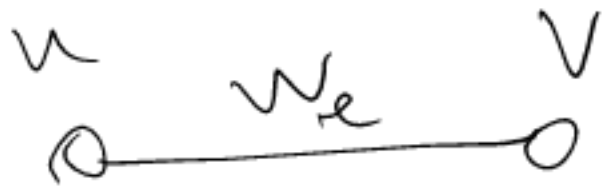
Chapter 3 Review

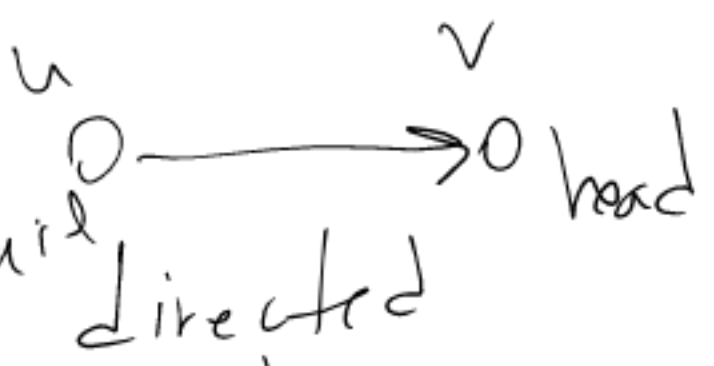
Graph theory $G = (V, E, w)$

V set of vertices / nodes

E set of links/edges between pair of nodes.



edge $e(u,v)$



directed edge

graph is called Directed graph

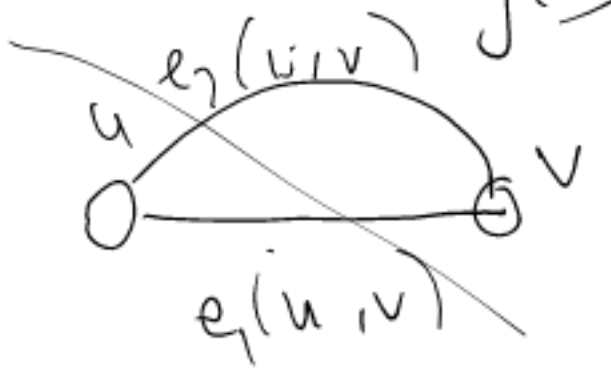
$G = (V, E)$

edge
Arc

Simple graph:

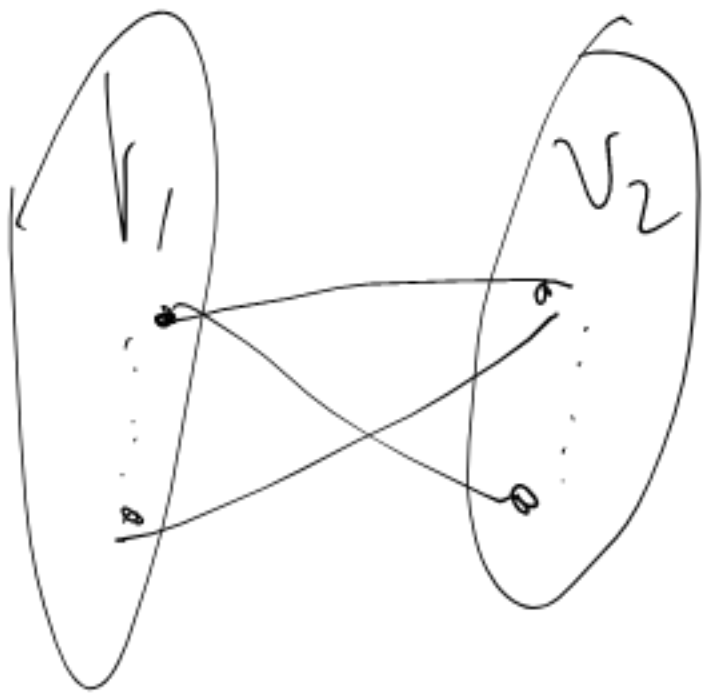
no self loops

no multiedges



Special types of graphs

- **Bipartite** $G = (V_1, V_2, E)$



$E = \{ (u,v) \mid u \in V_1$ and $v \in V_2$ and $\nexists$ edge (u,v) s.t. $u \in V_1$ and $v \in V_1$ or vice versa $\}$

(u,v): notation for edge between u and v

Tree: graph has no cycles.

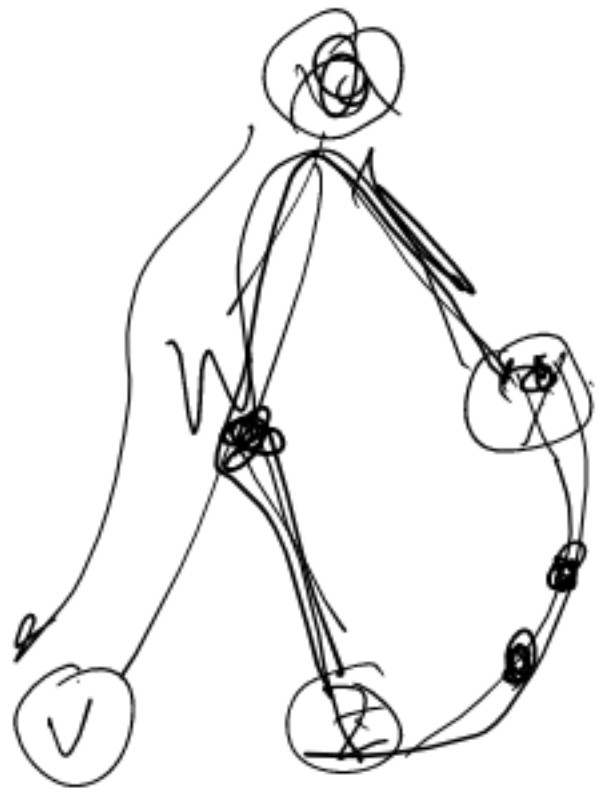
consider in this exam how many different ways u can reach v.

(1) $u - x_1 - x_2 - w - x_3 - v$

$u - x_1 - x_2 - w - v$

connectivity of a graph.

~~Circle~~ cycle is a 2-connected subgraph.



tree is a

<u>1-connected graph</u>.

$\exists$ one unique "path" between any pair of nodes

path:

walk: any alternating (vertex-edge) pairs in a graph.

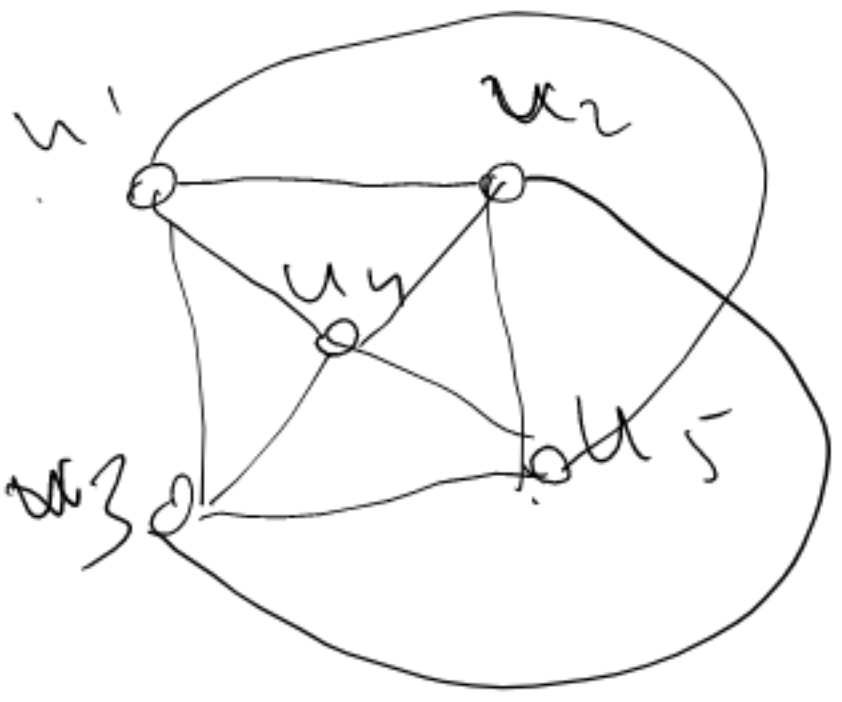
<u>path</u>: is walk without any duplicate vertices

$\Rightarrow$ cannot have any cycle

<u>How do we represent a graph</u>

data structure?

$G = (V, E)$

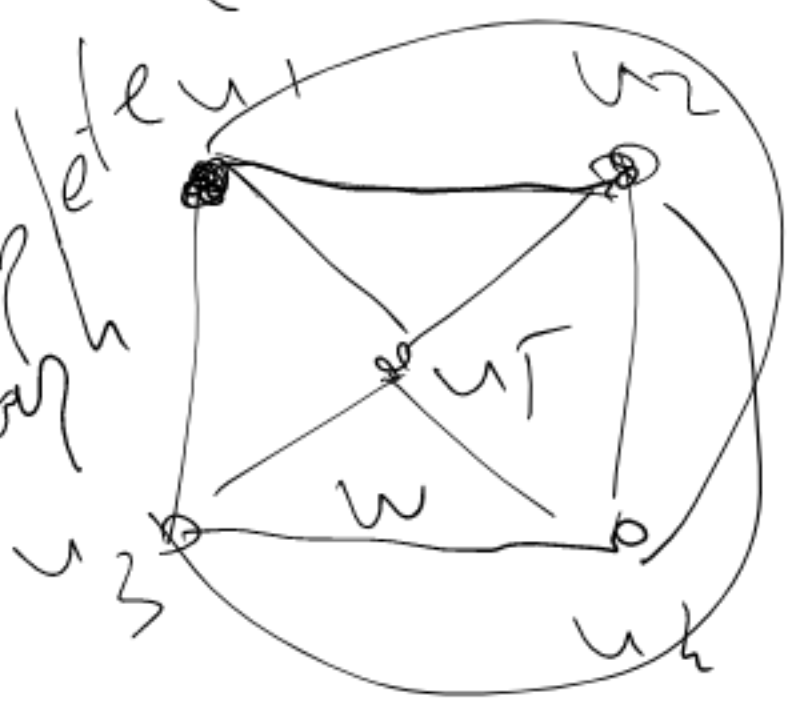


$n = |V|$ # of nodes

how many pairs one can create with n objects $\binom{n}{2}$

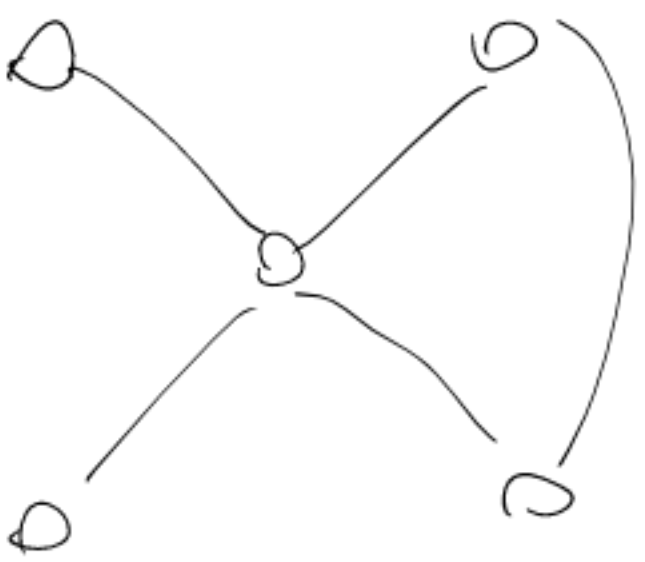
$n(n-1)$

complete graph

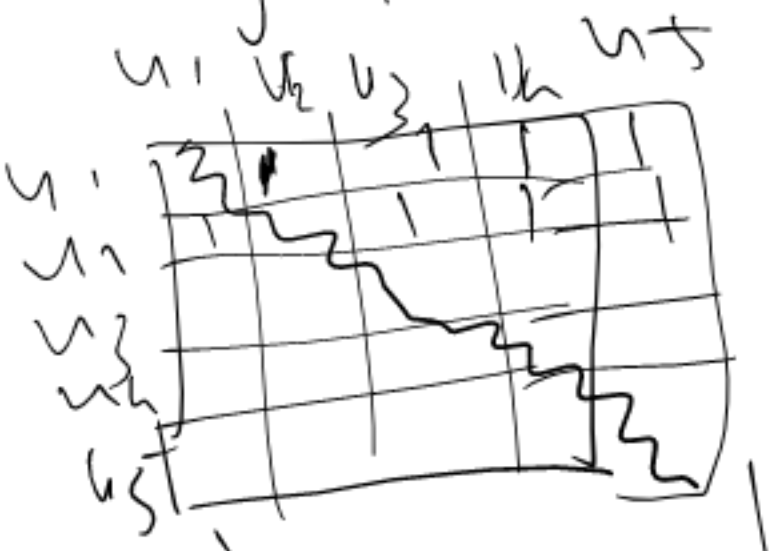


actual links $\Longrightarrow$ max possible link

how "dense" the graph is!



$-$



A

Adjacency Matrix

Question 1 is vertex u_2 a neighbour of vertex u_3?

$G = (V, E, w(e))$

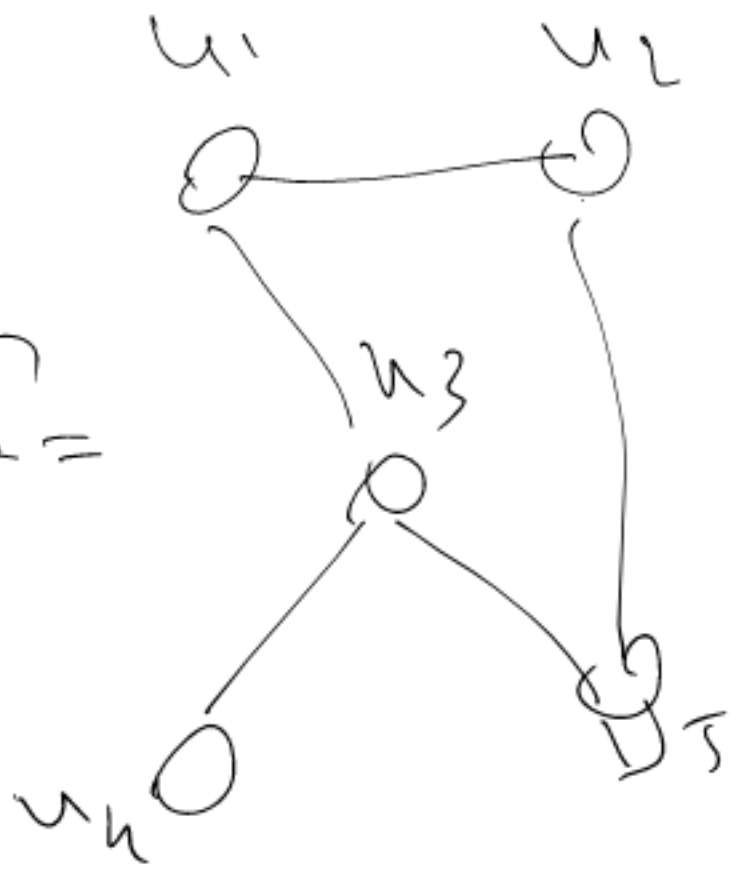
$A(u_2, u_3) = 1$ Yes

else 0 No.

$\Rightarrow$ if the graph is dense then Adj. Matrix representation is optimum to answer Q1:

Sparse graph

$G =$



u_4

0	0	1	0	0

$n \times n = O(n^2)$

Adjacency List

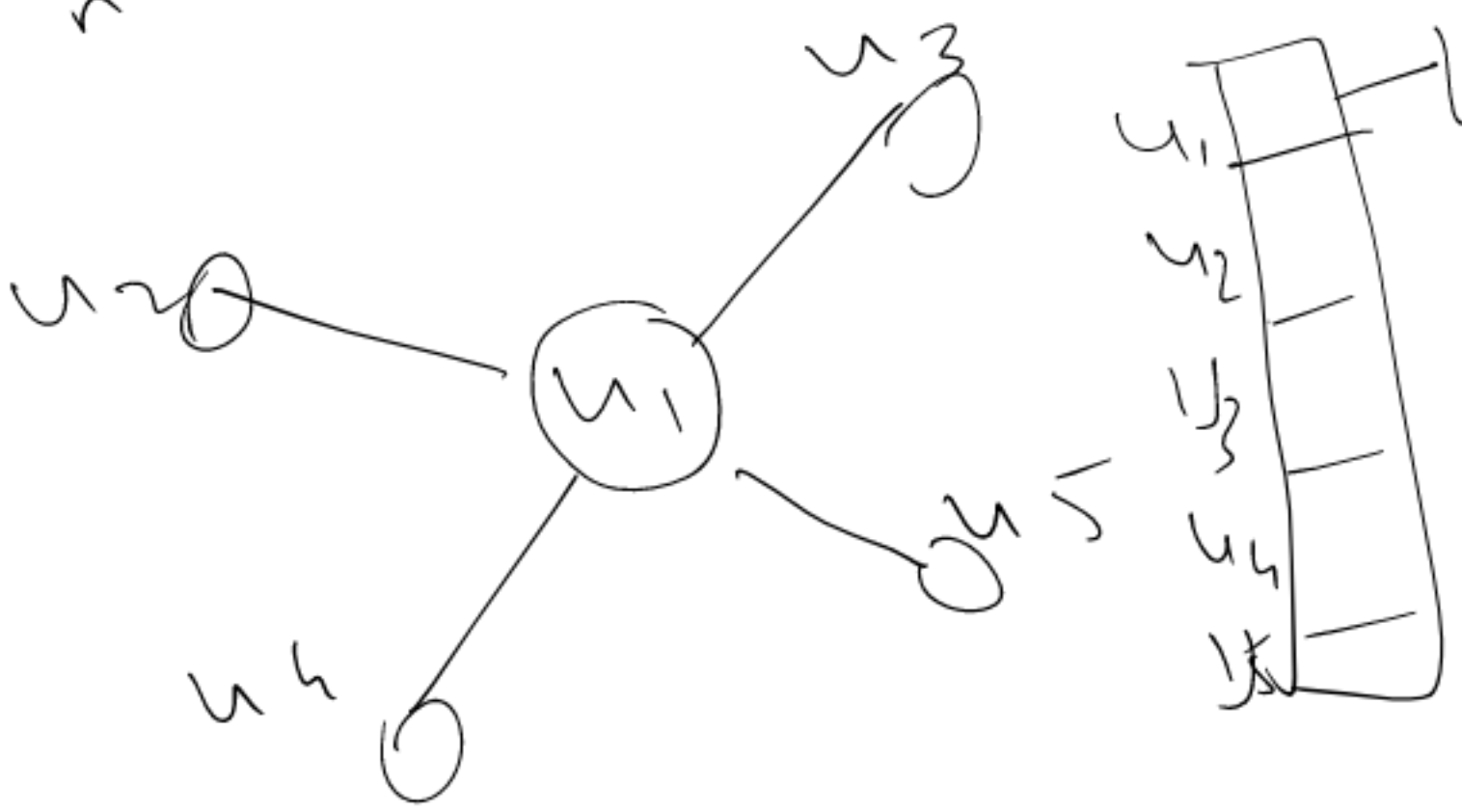
$O(1)$

linear search

$O(n)$

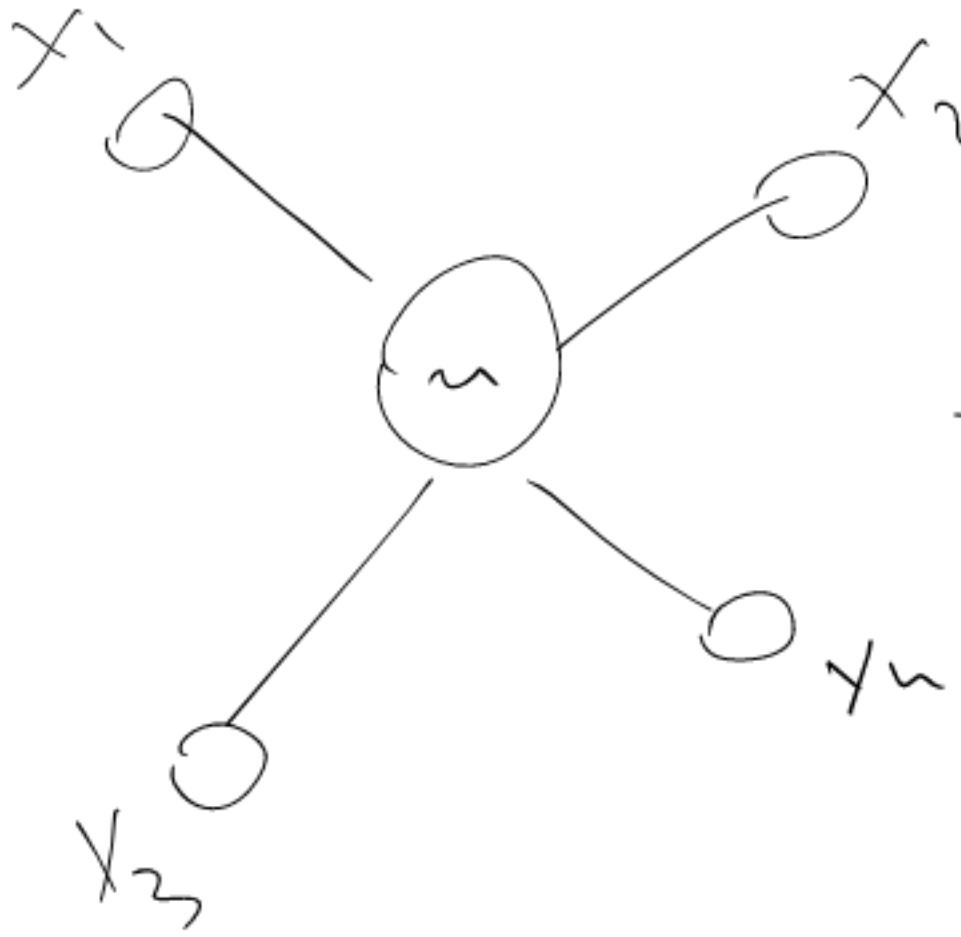
answering Q1

if $\exists$ link (u_i, u_j)?



$\Rightarrow$ sparse graphs use Adj. List

dense graphs use Adj. matrix



vertex
node

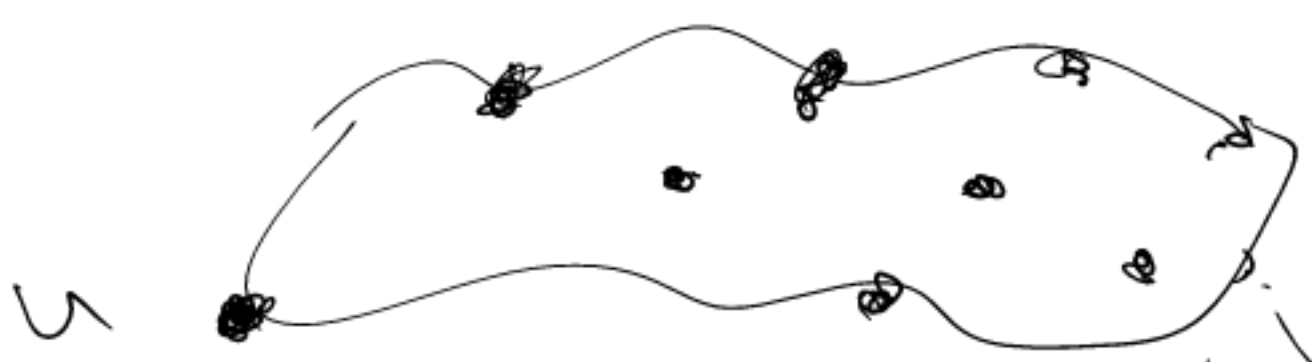
degree of u is the # of its neighbors

u and x_i are neighbors if $\exists$ edge (u, x_i) in the graph.

---

how to discover/explore the properties of a graph.

Q1: given a node $u \in V$ what other vertices/nodes are reachable from u?



G is connected if all other vertices are reachable from u.

Connected if All are reachable

Connected components

the Alg. to answer $\boxed{Q1}$ is

<u>Depth first Search</u>

Alg. DFS (G)

$\Theta(|V|) = n$ { for all $v \in V$
visited (v) = false

n { for all $v \in V$
if ¬ visited(v): call
explore (v)

Alg. Explore (G, v)

visited(v) = true

$\Theta(|E|) = m$ [for each edge (v, u) $\in E$
if not visited (u) call explore(u)

$O(|V| + |E|) = O(n + m)$

if graph is dense $m = O(n^2)$

sparse $m = O(n)$

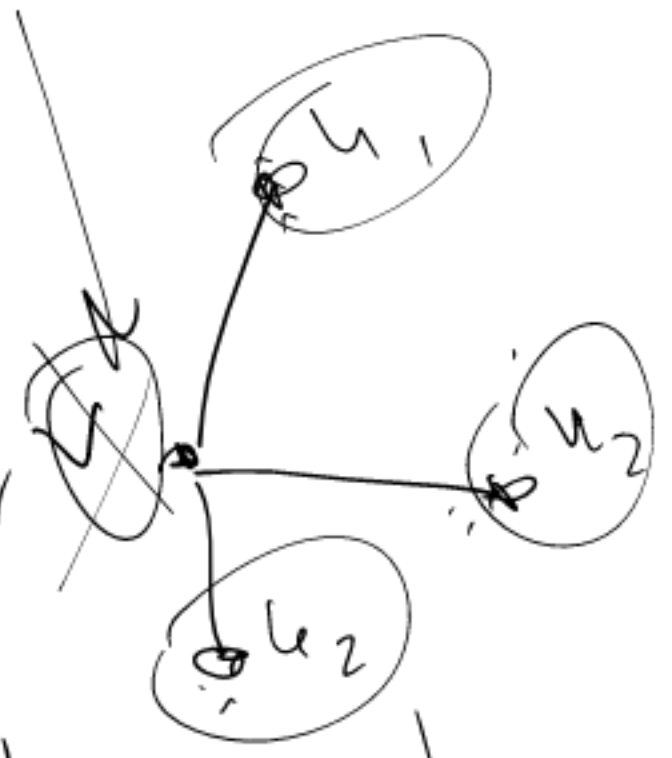
explore(G, v)

visited(v) = true

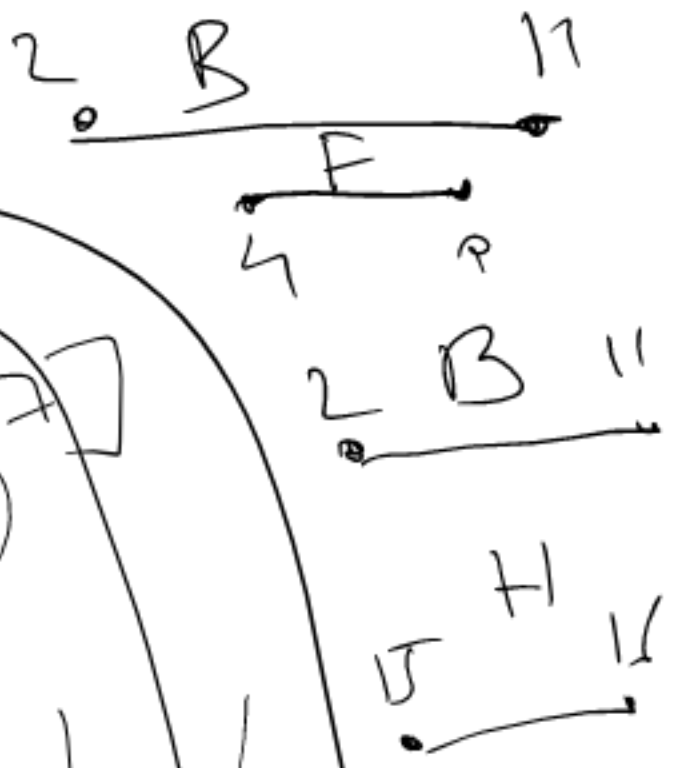
→ counter 1 preorder #

∀ edge (v, u)

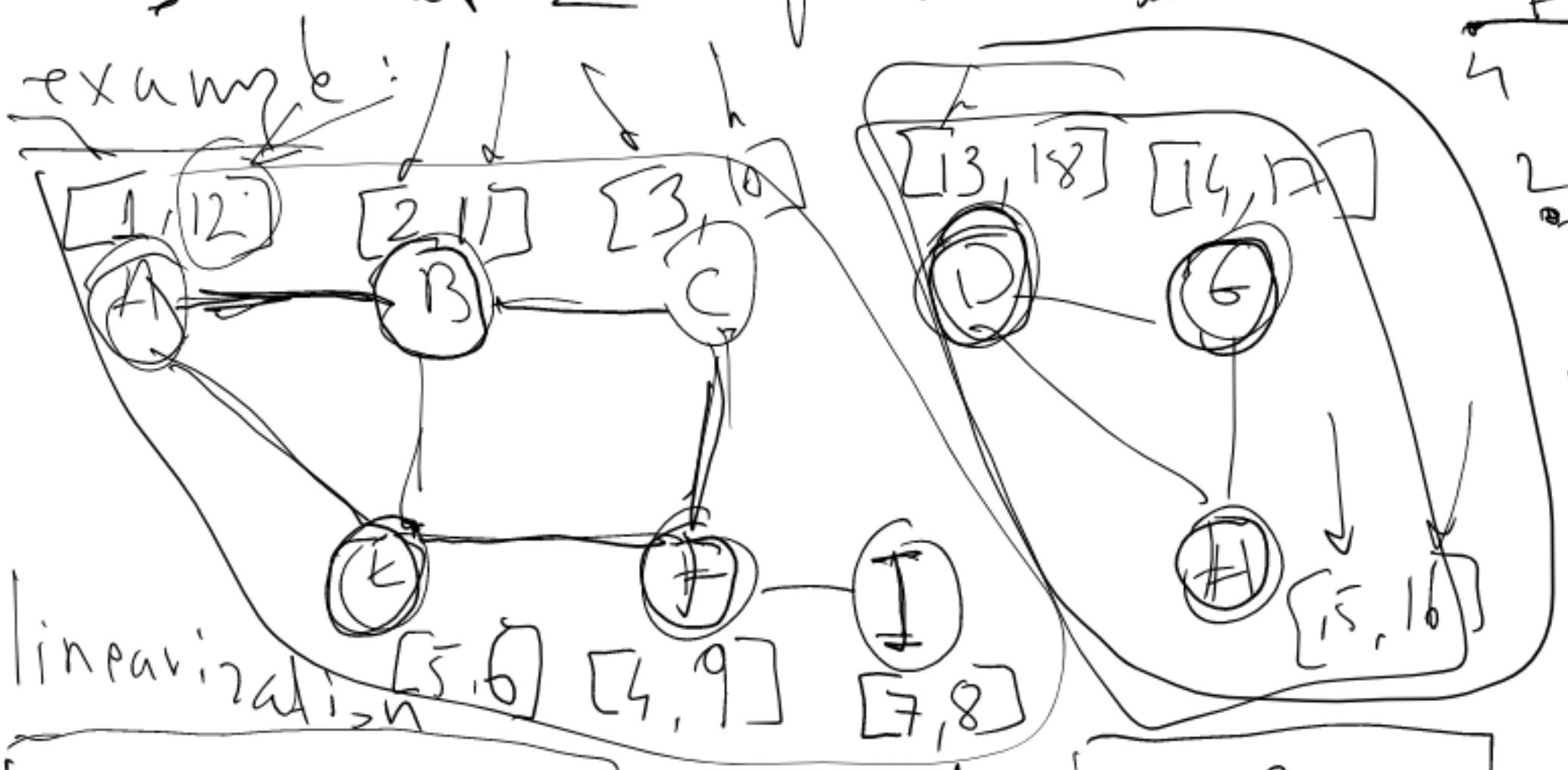
if u is not visited explore(u)



→ counter 2 postorder #



example:



linearization

A, B, C, F, I, E

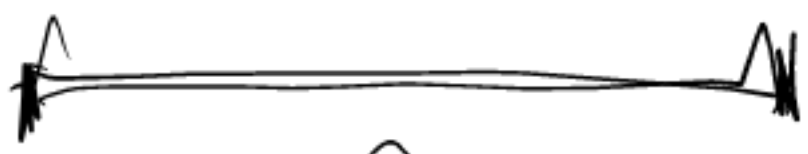
topological sorting

D, G, H

Topological Sort or Linearization is to print the node IDs in decreasing order of their post order #

Consider 2 nodes (u) and (v)

[pre(u), post(u)] [pre(v), post(v)]



interval for u

interval for v

these intervals are either disjoint or one is contained within the other

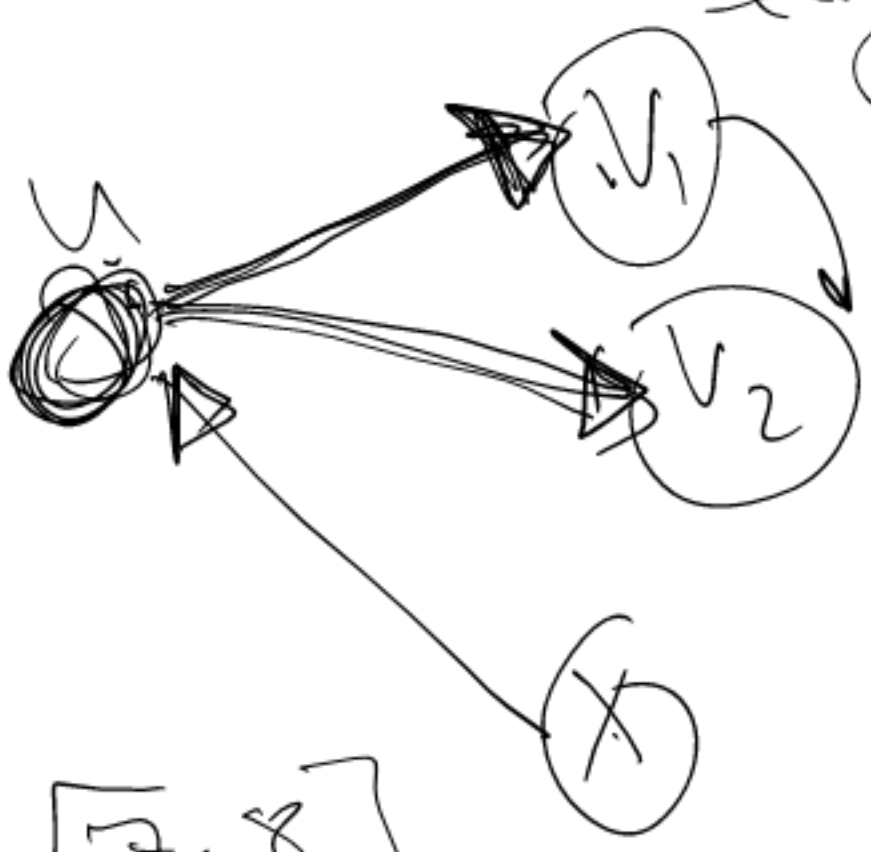
2 Cases — within ⇒ they reside in the same connected component
— disjoint

different C.C.

Directed graphs

edges have orientations

DFS works as before but only in the direction of the edges

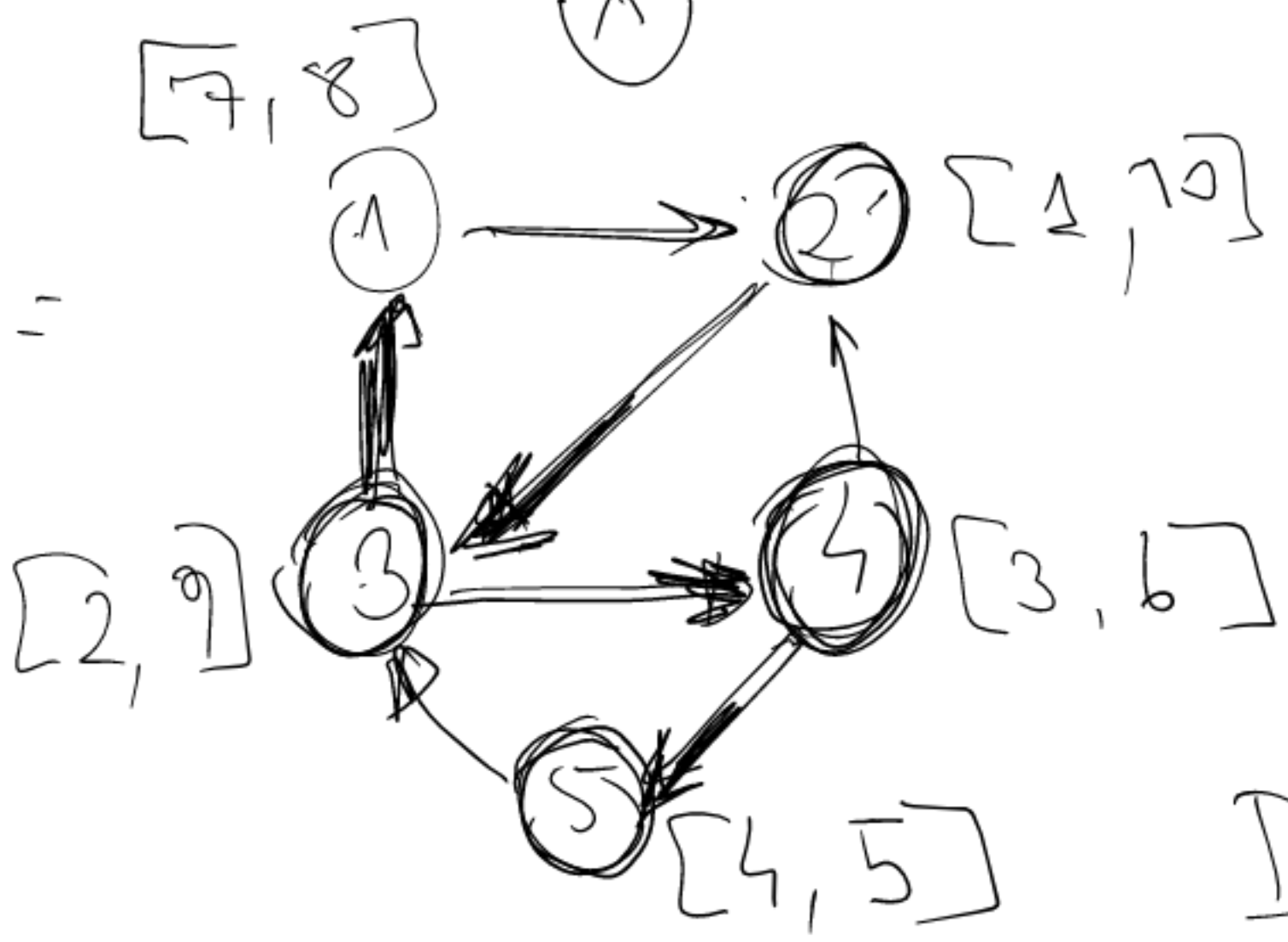


(v, u)

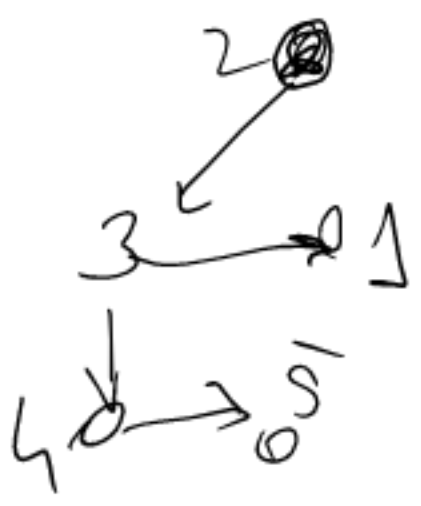
$\overrightarrow{\langle v, u \rangle}$

$\overrightarrow{(v, u)}$

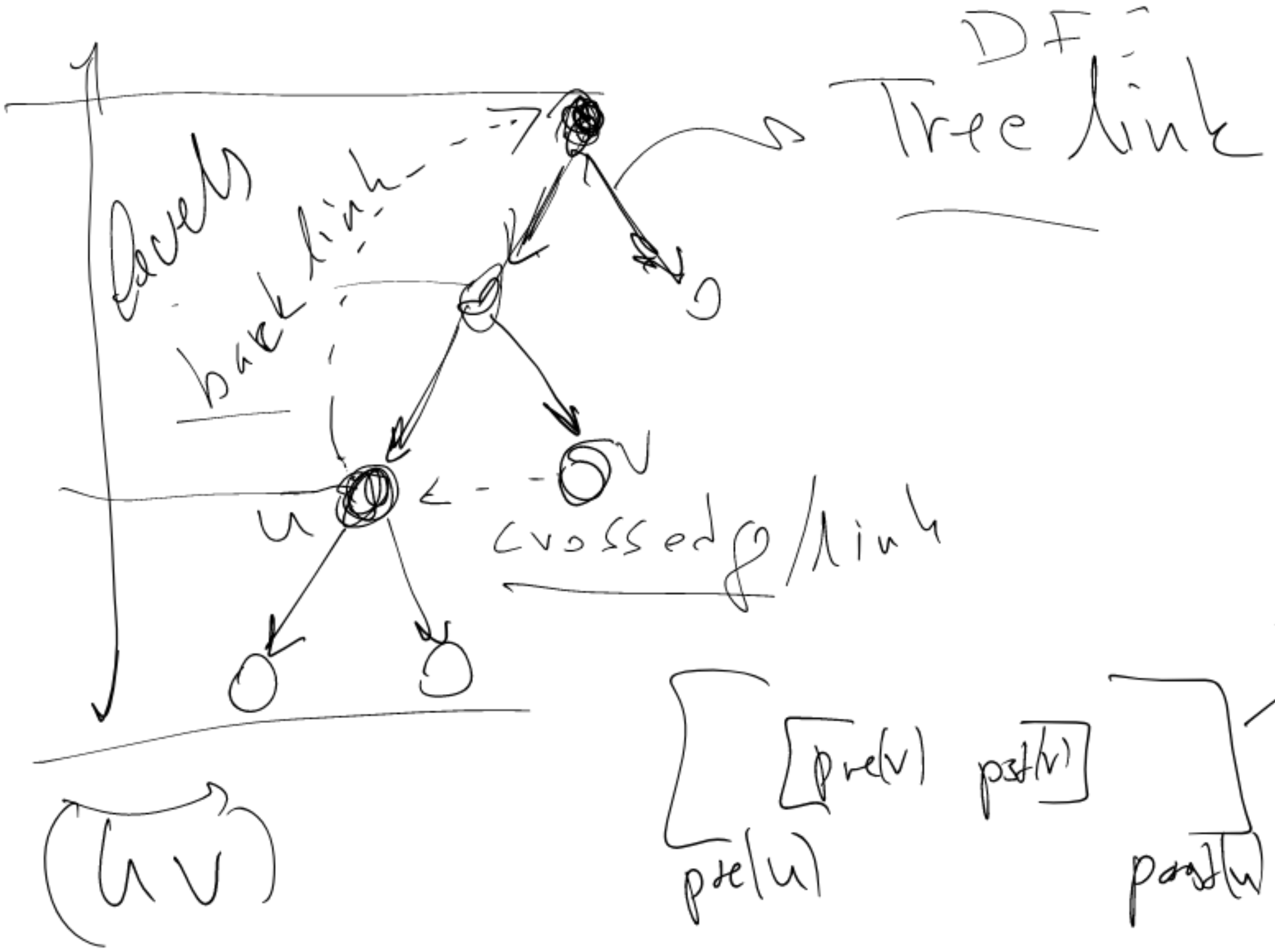
G =



TREE!



DFS



Cross edge [pre(u) post(u)] [pre(v) post(v)]

back edge [[u]]v

Given Directed Graph: how do we decide if ∃ cycle in this graph

Directed graphs that are cycle free

DAGs

DAGs we can also do topological sorting

1) Do DFS and assign post-order #s

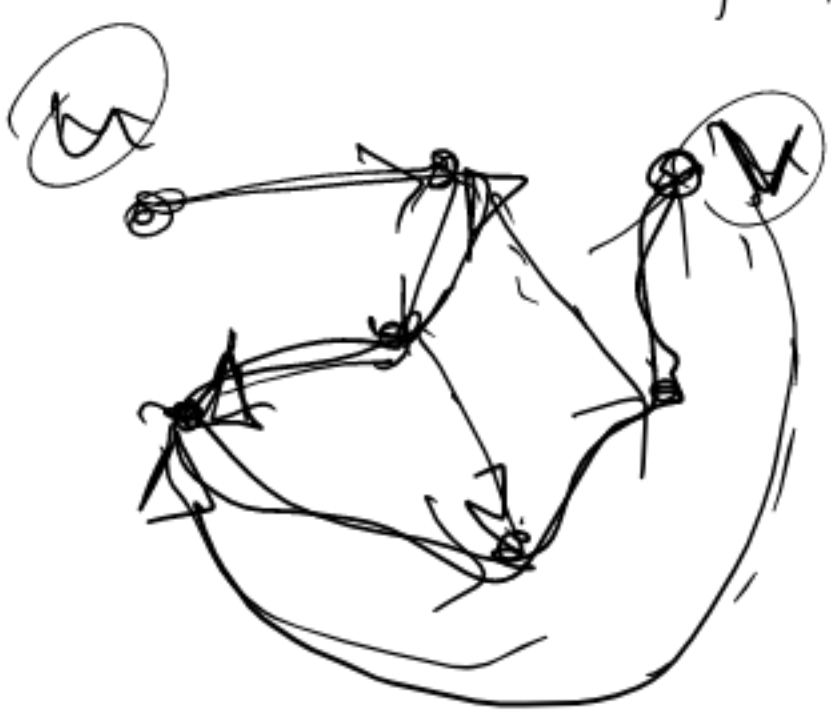
2) output node IDs in reverse order of the post order list

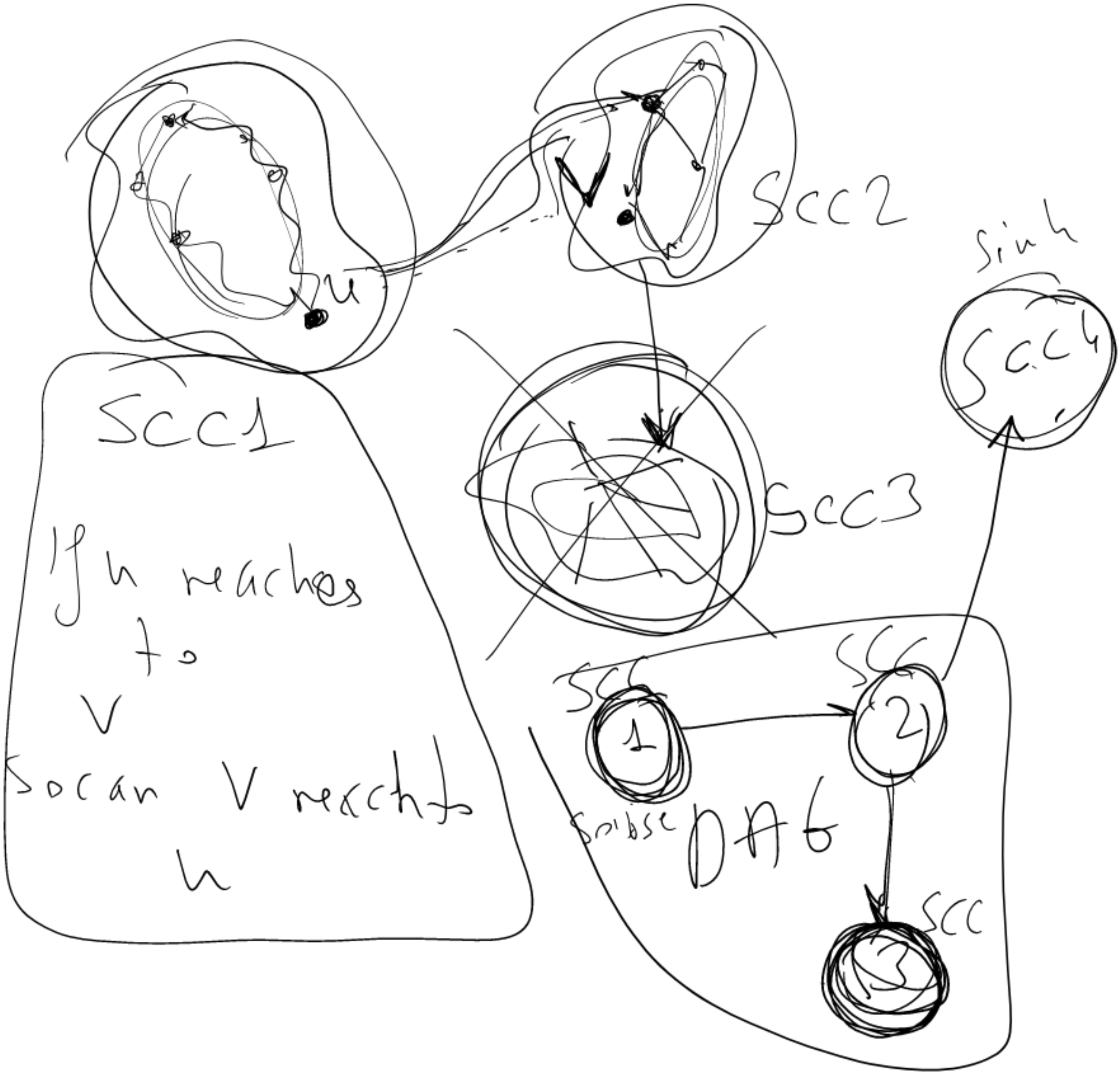
$O(|V| + |E|)$

In Directed graphs we have

$\boxed{\text{Strongly Connected Component}}$

(as supposed to Connected Comp in undirected graphs




SCC1
u
SCC2
v
SCC3
sink
SCC4
If u reaches to v
so can v reach u
SCC 1
SCC 2
SCC 3
DAG

How do we find SCC.

Given a directed graph.

① Identify all SINK nodes

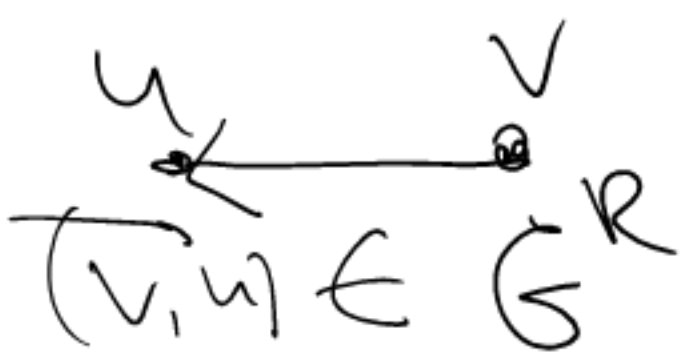
② From each sink node.
Do DFS ⟹ SCC

remove it and mark all the vertices in it as "visited"

How do we find the sink/source nodes

a source vertex in a directed graph has the highest post order #

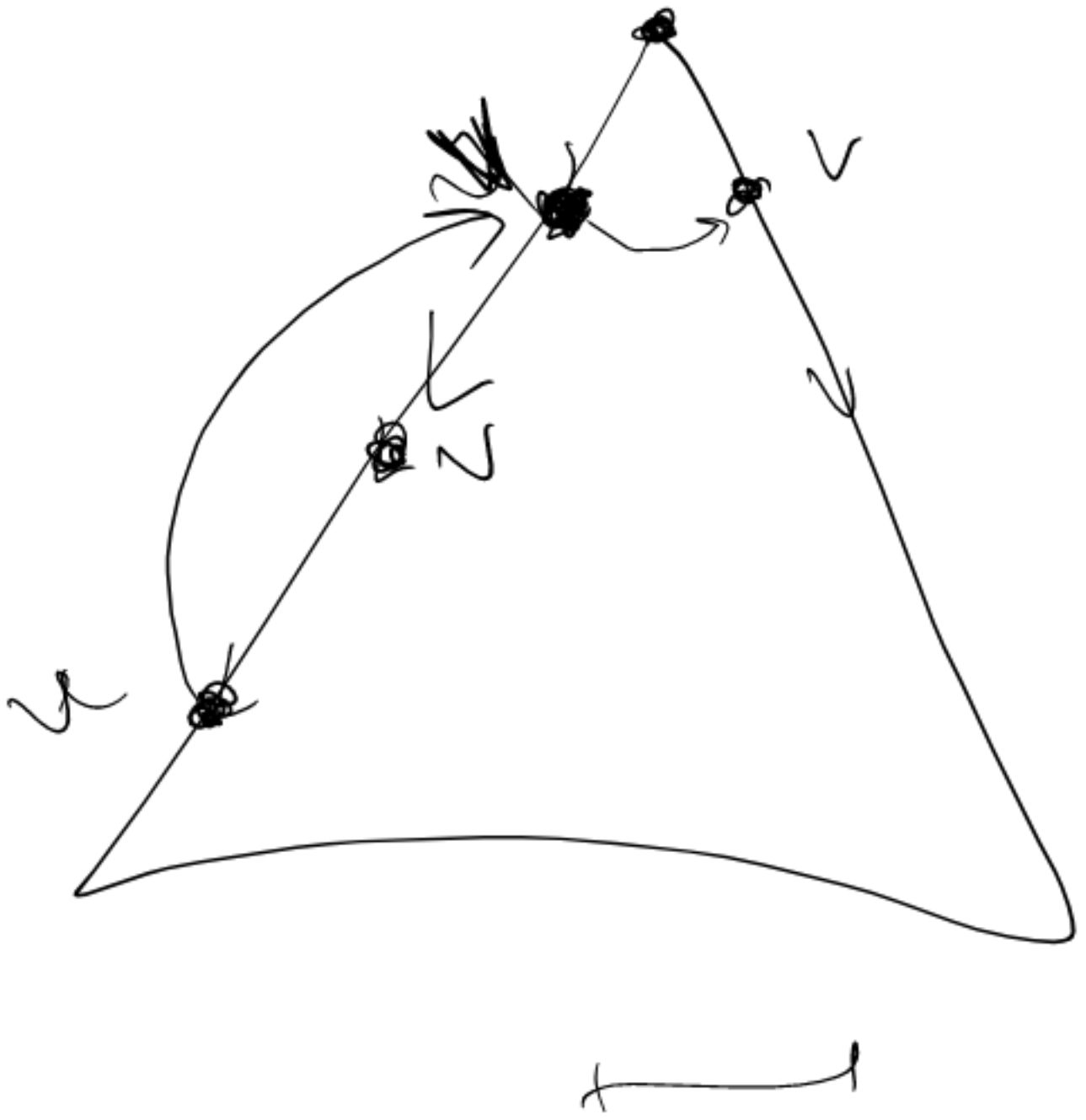
let's reverse all the links in G to opposite direction



Revised Alg:

1. Build G^R reverse graph
2. Run DFS on G^R
3. Traverse in decreasing order of post(v) in G and find all reachable vertices for v

every such set is a SCC.



DFS

u v

(u, v) ?

tree [v] = u

back [w] = v

cross

pre

post

pre(u) post u