Decompositions of Graphs

$G = (V, E)$

vertex set

edge set

$E \subseteq V \times V$

(u, v)

$|V| = n$

$|E| = m.$

Two types of graphs

- undirected
- directed

$(u,v) \in E$
$(v,u) \in E$

Representation

- Adjacency Matrix
- Adjacency List

n

A n $\boxed{1}$

$A_{i,j} = 1$ iff $(v_i, v_j) \in E$

1: 3, 7, 10

2: 3, 4, 8

3: 1, 2, 9

...

Tradeoffs between Adj. List & Adj. Matrix

Operation	List	Matrix
Check $(u,v) \overset{?}{\in} E$	$O(d_u)$	$O(1)$
List neighbors of u	$O(d_u)$	$O(n)$
Space	$O(n+m)$	$O(n^2)$

$n = |V|$

$m = |E|$.

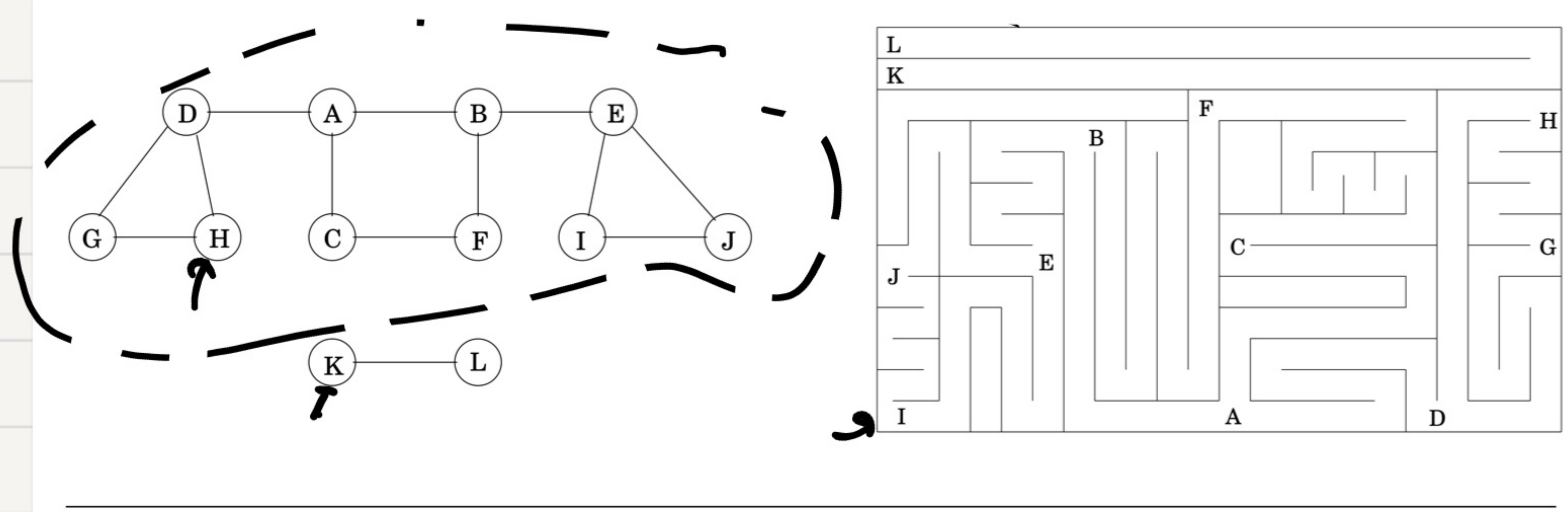
Connectivity in Graphs

Two Algorithms to explore graphs:

DFS = Depth First Search (Today)

BFS = Breadth First Search (Thursday)

Figure 3.2 Exploring a graph is rather like navigating a maze.



Explore (G, v):

- visited[v] = true.

pre

- for all (v,u) ∈ E:
  - if not visited[u]:
    - Explore(G,u)

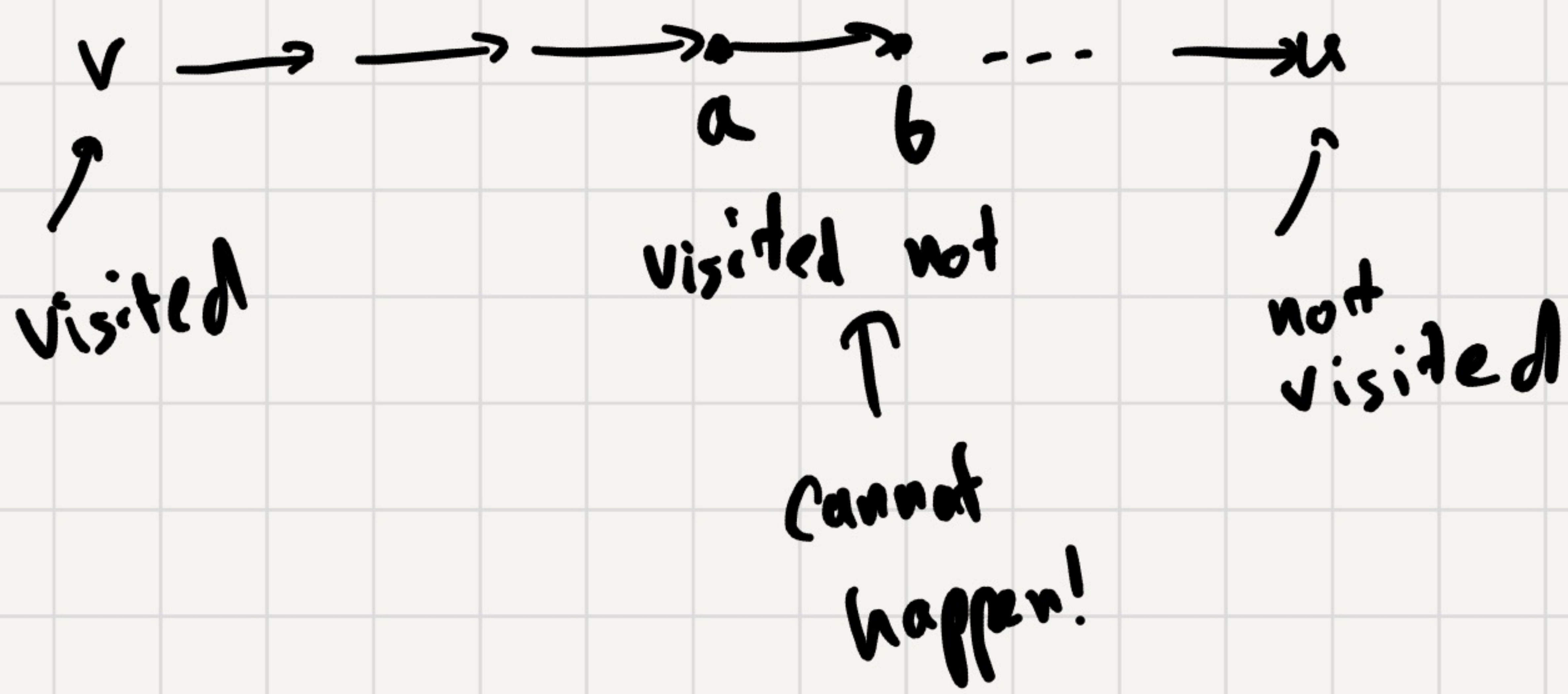
post

DFS(G):

- for all v∈V: visited[v] = false.
- for all v∈V:
  - if not visited[v]
    - Explore(G,v).

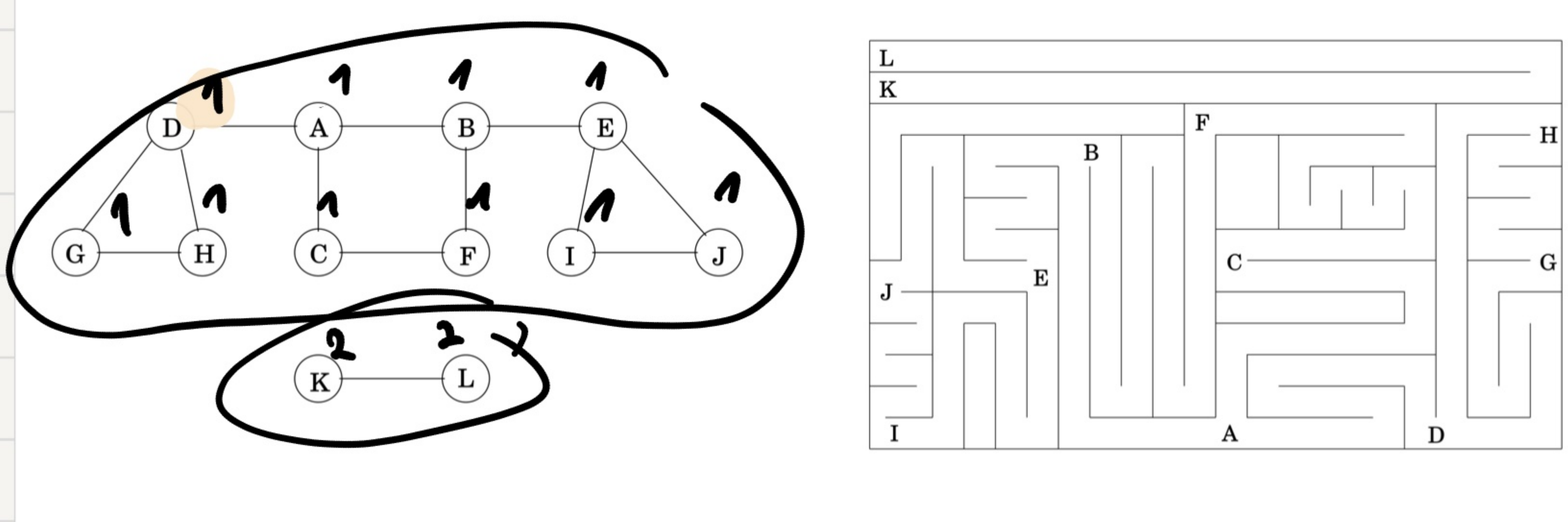
Claim: Explore (G, v) visits every node reachable from v.

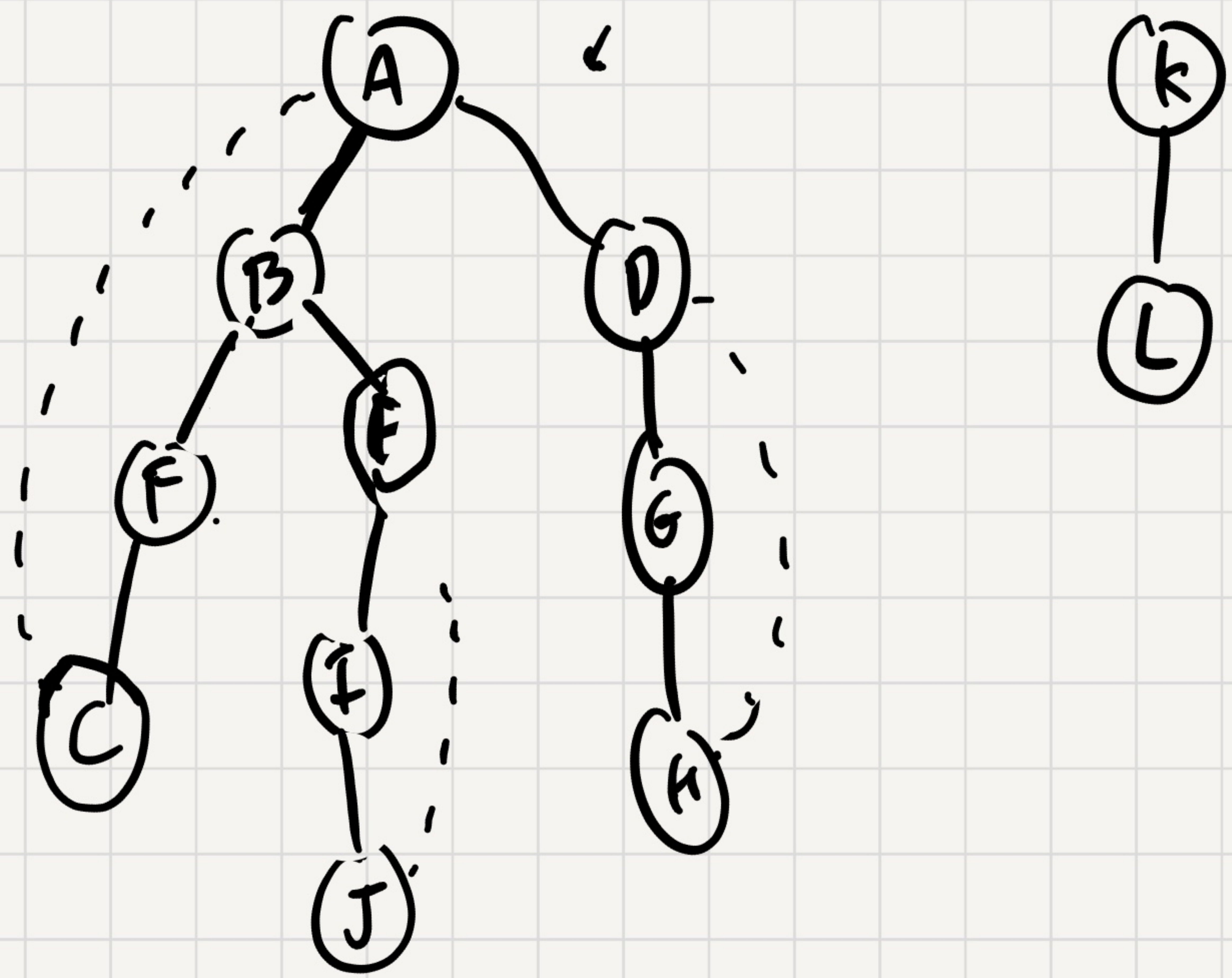
Proof: By contradiction. Assume $\exists u \in V$ that is reachable & not visited



The DFS Search Tree

Figure 3.2 Exploring a graph is rather like navigating a maze.

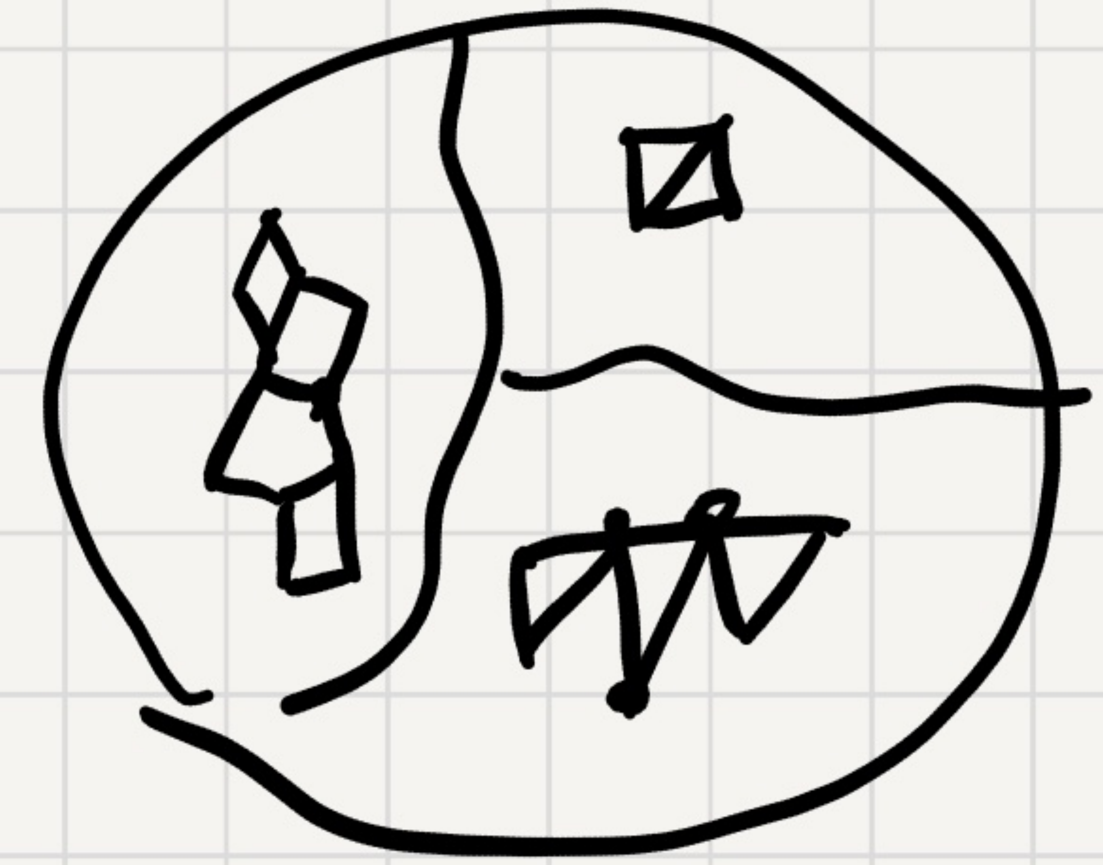




Connected Components in Undirected Graphs

$u \sim v$ if there's a path in G from u to v.

$u - a - b - c - v$



$v \sim v$ & $v \sim w \Rightarrow u \sim w$

Explore(G,v):

- visited[v] = true
- comp[v] = cc
- for all $(v,u) \in E$:
  - if not visited[u]:
    - Explore(G,u)

DFS(G):

- for all $v \in V$: visited[v] = false.
  comp[v] = null
- cc = 1.
- for all $v \in V$:
  - if not visited[v]:
    - Explore(G,v)
    - cc += 1.

Running Time of DFS

We consider every node exactly once

We consider every edge exactly once or twice

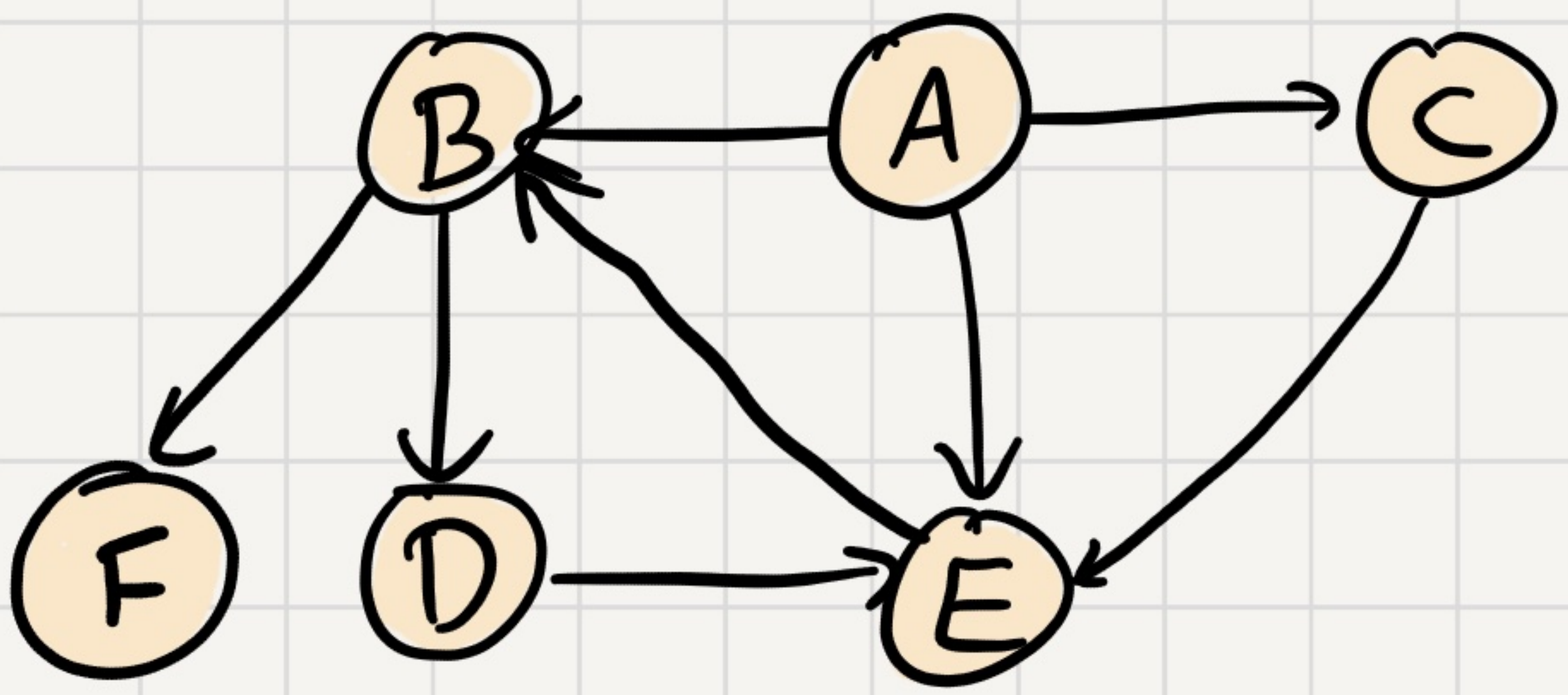
directed undirected

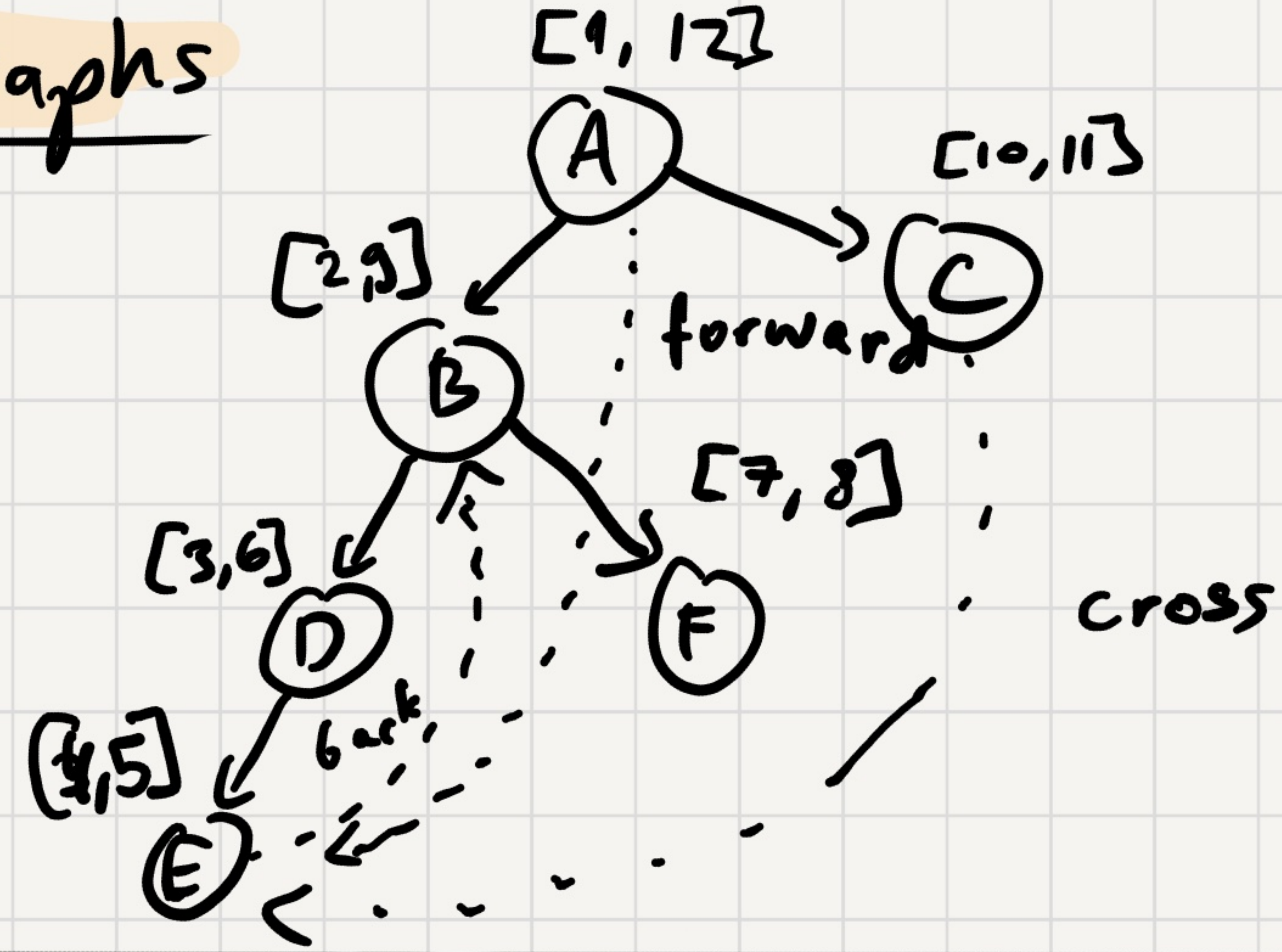
→ $O(n+m)$ time.

$n = |V|$

$m = |E|$.

DFS in Directed Graphs





Explore (G, v):

- visited[v] = true
- pre[v] = clock++
- for all $(v,u) \in E$:

  if not visited[u]:

  Explore(G,u)

- post[v] = clock++

DFS(G):

- for all $v \in V$: visited[v] = false.

  pre[v] = null

  post[v] = null

- clock = 1
- for all $v \in V$:

  if not visited[v]:

  Explore(G,v)

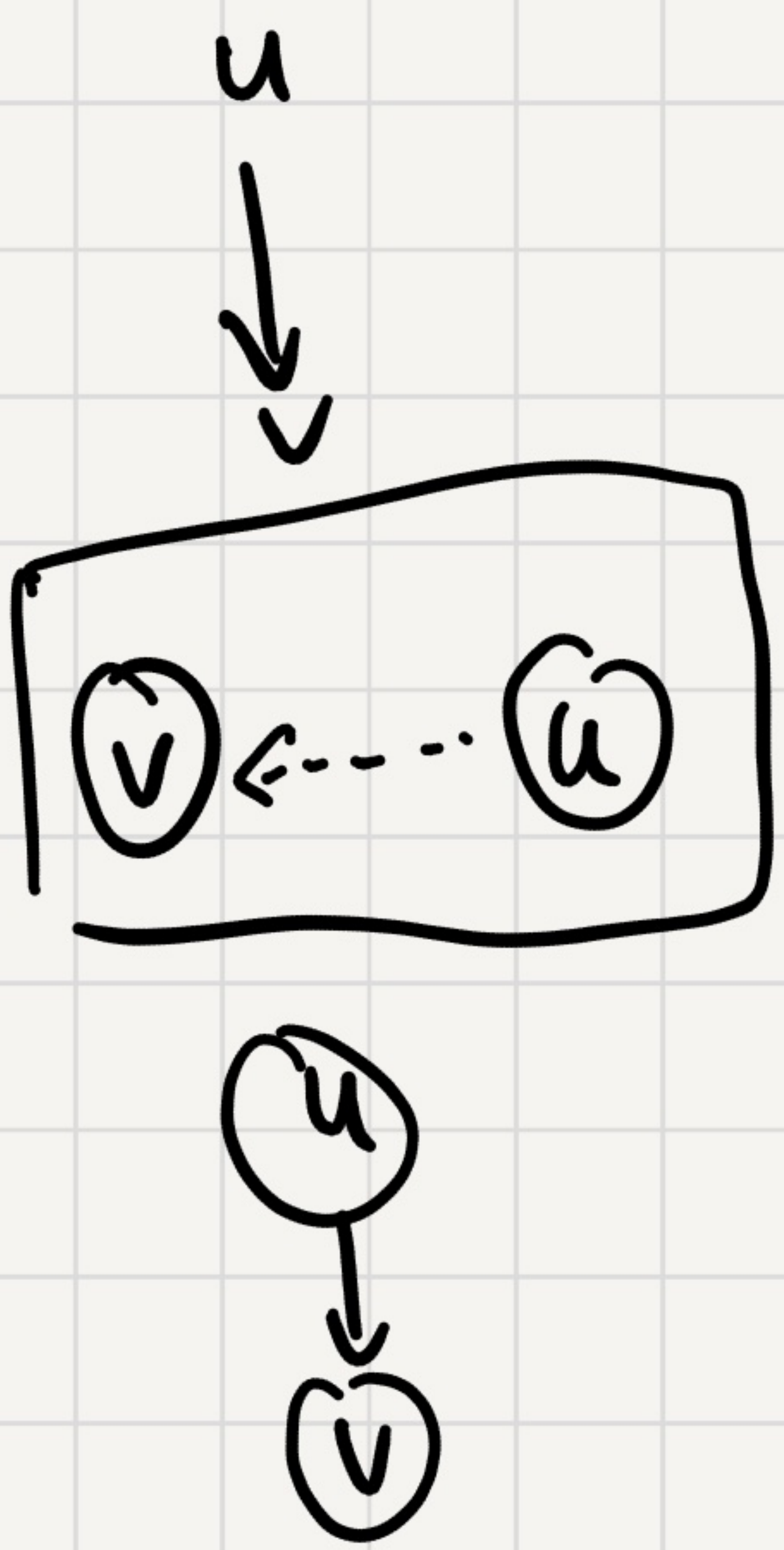
Types of edges $(u,v) \in E$

$[_u \;\; [_v \;]_v \;]_u$ Tree edge / Forward edge

$[_u \;\; [_v \;]_u \;\;]_v$ impossible

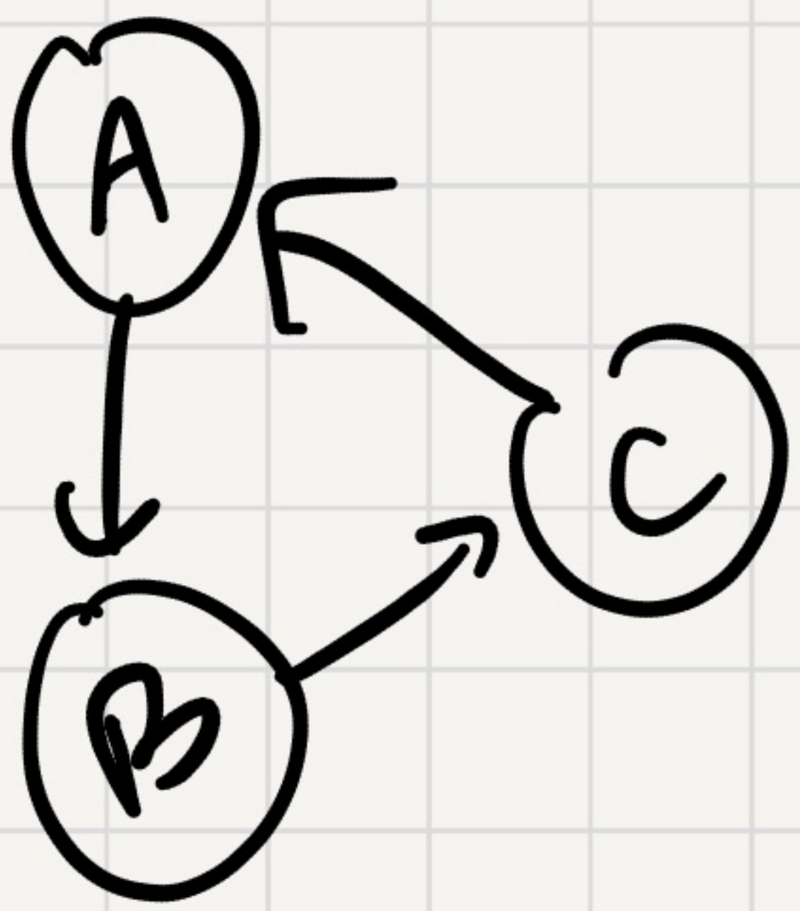
$[_u \;]_u \;\; [_v \;]_v$? not possible

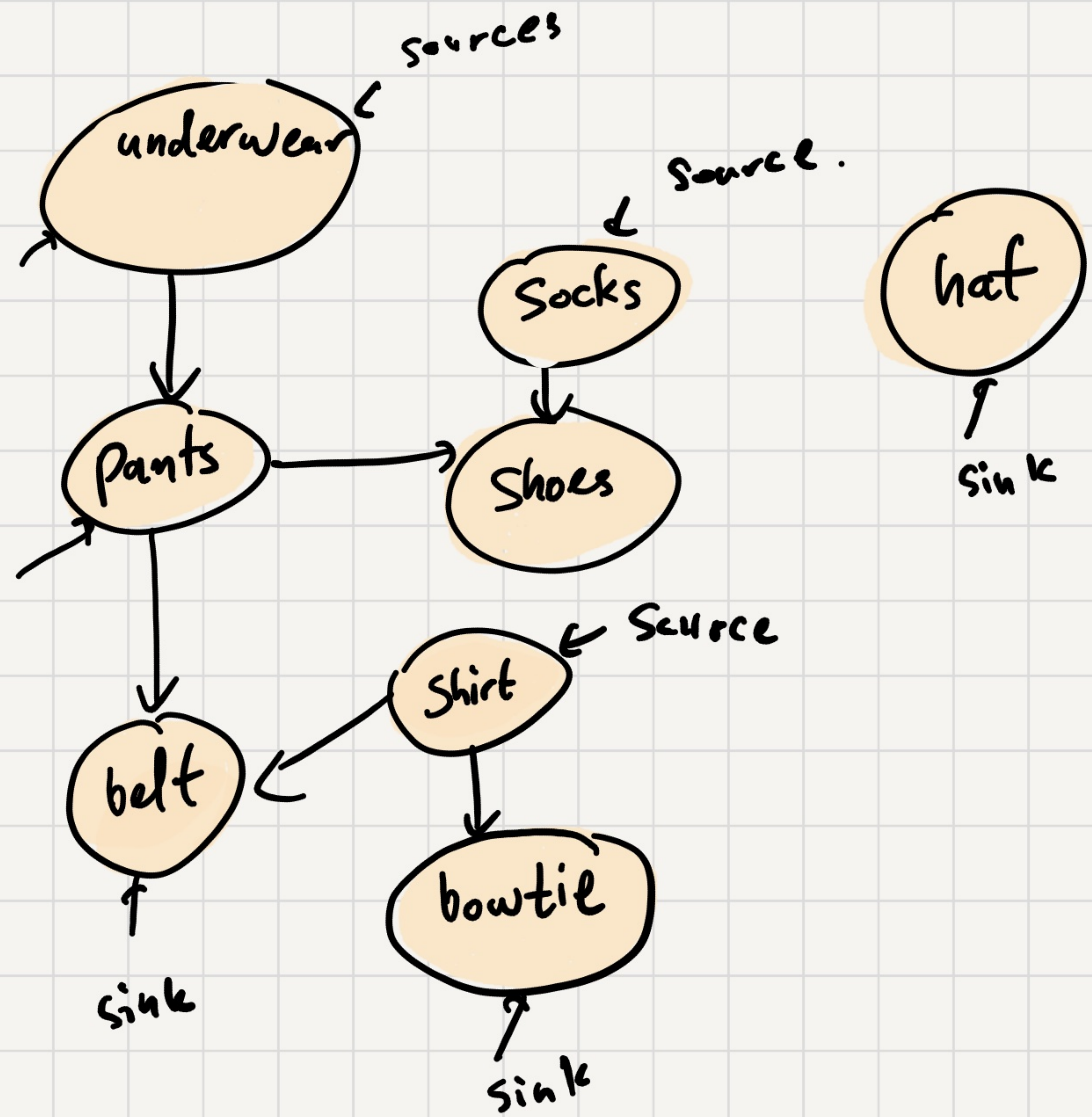
$[_v \;]_v \;\; [_u \;]_u$ cross edge ← impossible if G is undirected.

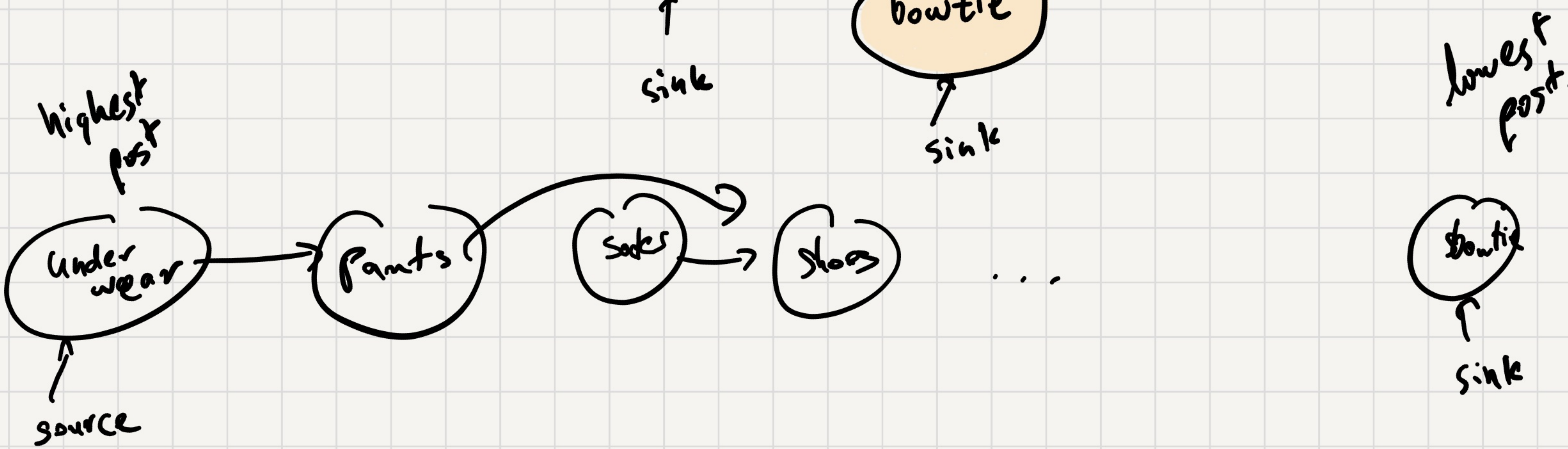


$\cancel{[_v \;\; [_u \;]_u \;\;]_v}$ back.

Topological Sort







Topological Sort

Def'n: A directed acyclic graph (DAG) is a digraph with no cycles.

Claim: A digraph G is a DAG iff in DFS(G) there are no back edges.

Proof: If there's a back edge $\Rightarrow$ cycle in the original graph.

Assume G contains a cycle $v_1 \to v_2 \to \cdots \to v_t$

WLOG assume v_1 is the first vertex explored in DFS(G).

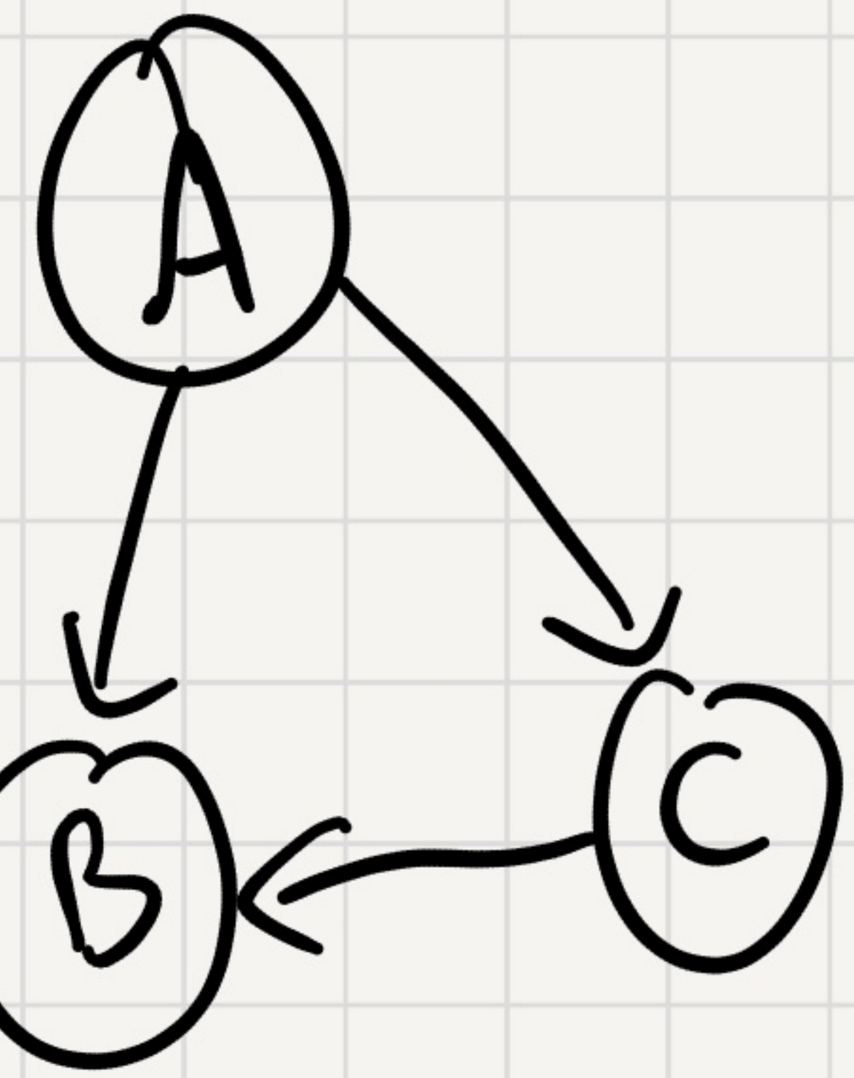
Claim: In a DAG $(u, v) \in E$ $\quad$ $post(u) > post(v)$.

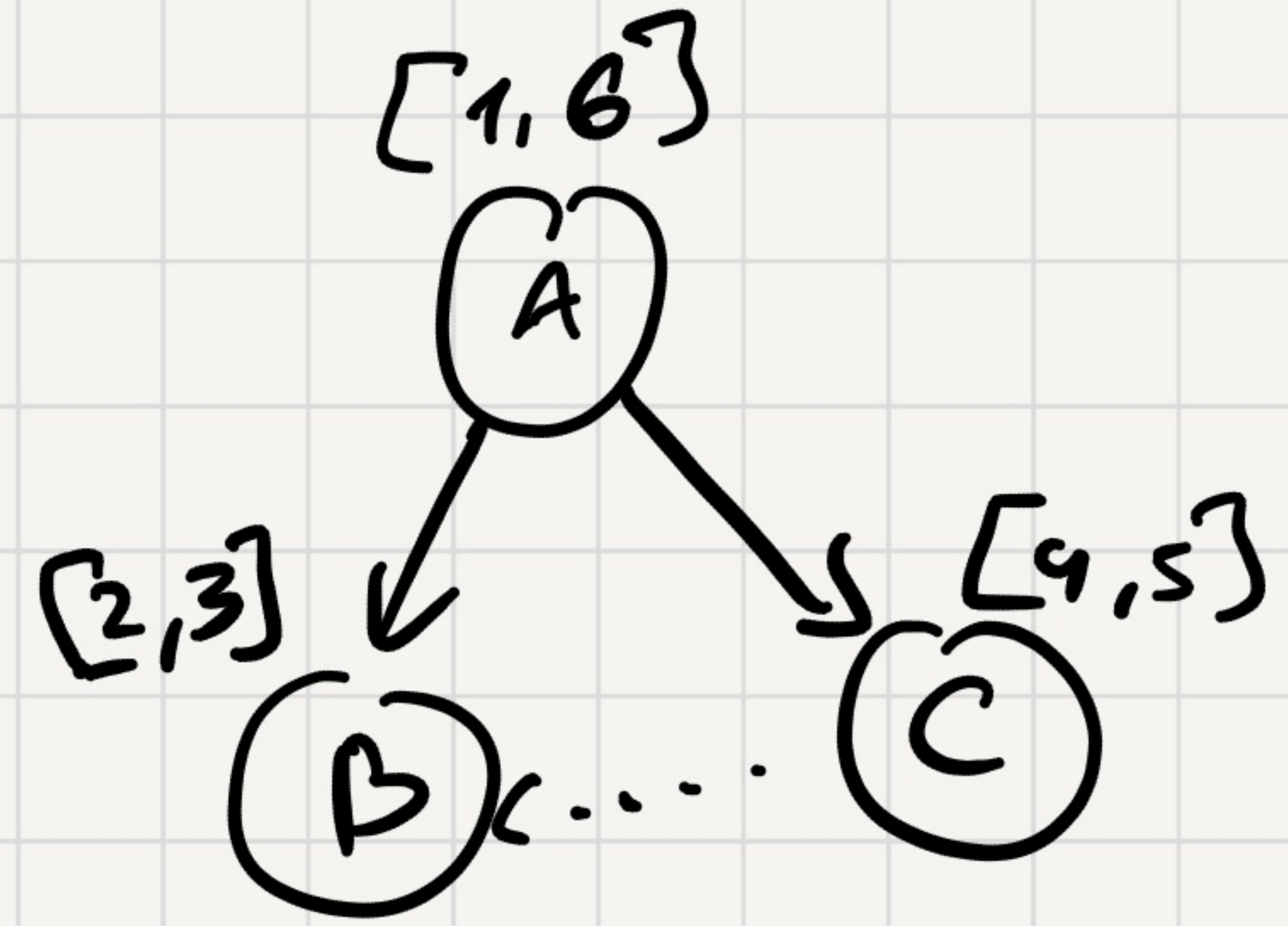
check all cases: forward ✓
cross ✓
tree edges ✓

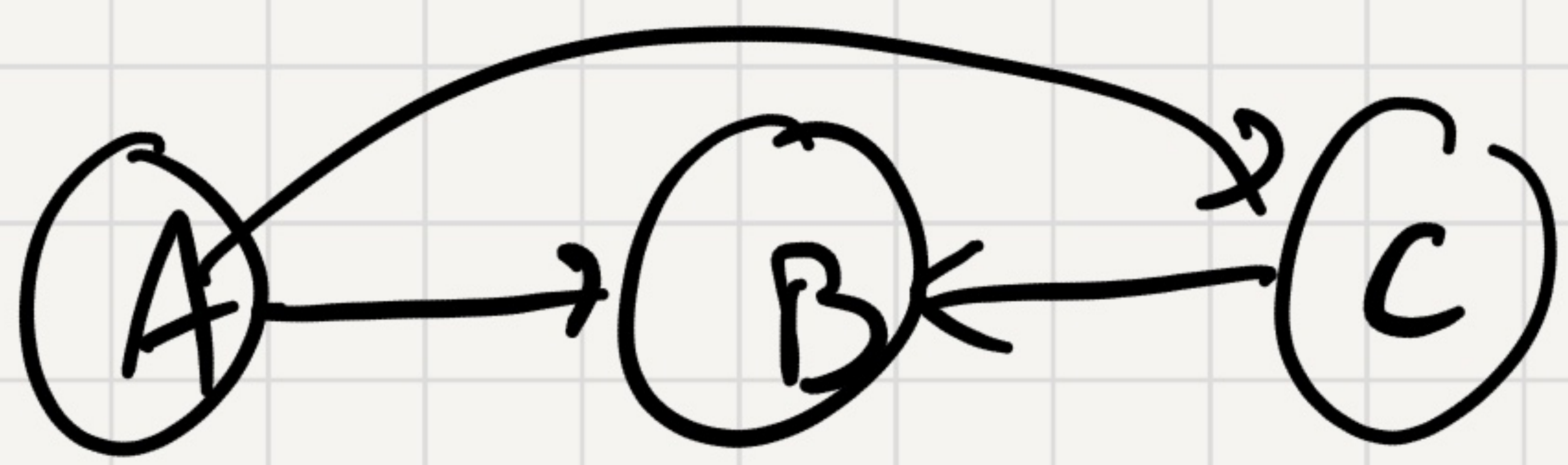
post $\quad$ post

12 $\quad$ 11

$u \longrightarrow v$


A
B
C


[1, 6]
A
[2, 3]
B
[4, 5]
C

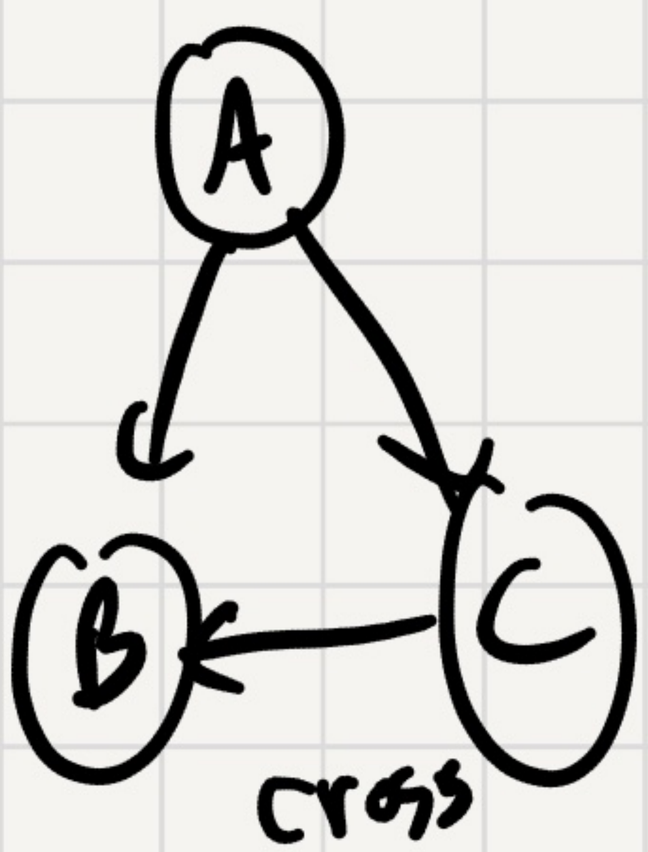

A
B
C

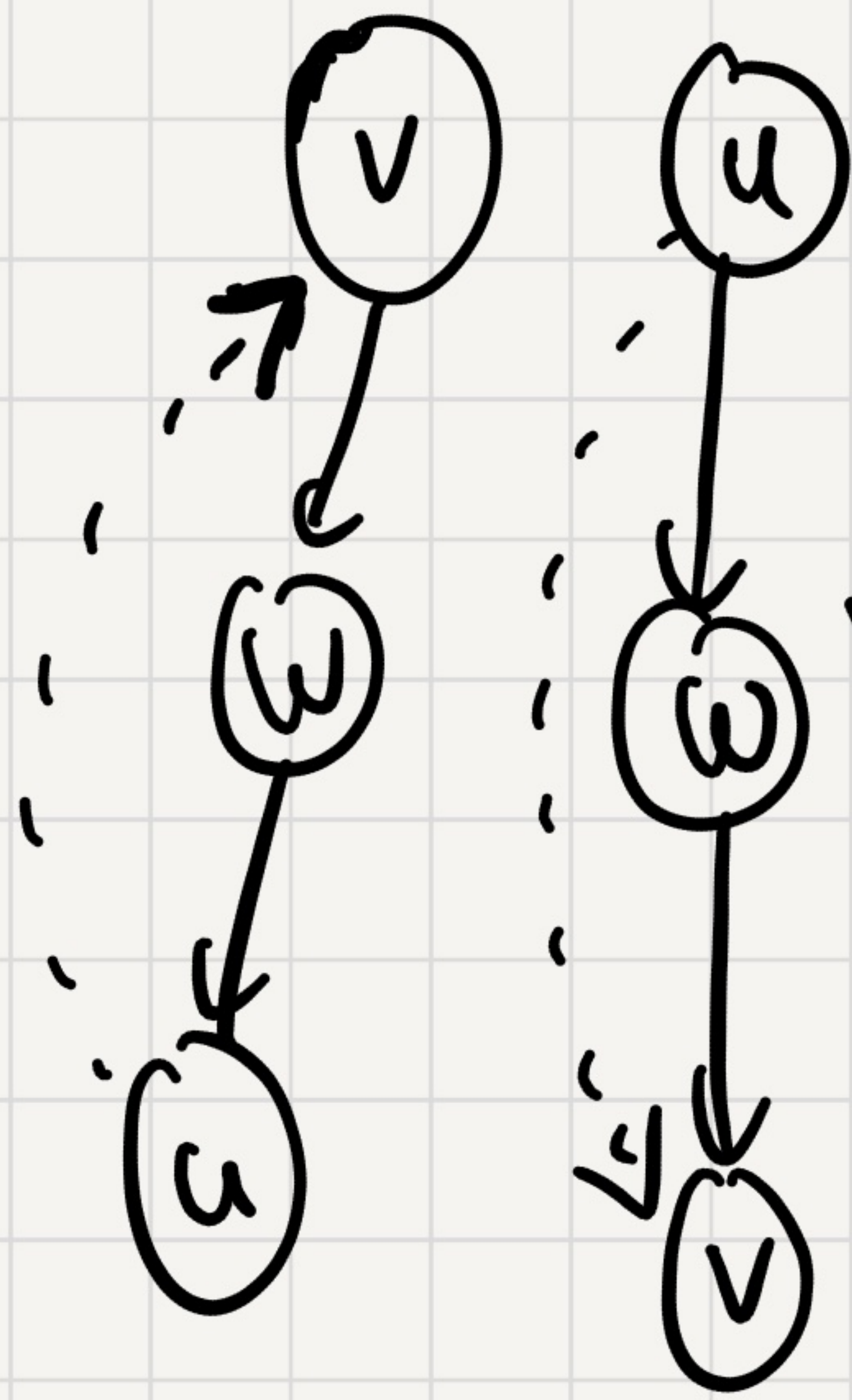
SCC : Strongly Connected Components

$u \sim v$ if there's a path from u to v and a path from v to u.

Distances in Graphs

$(u,v) \in E$





$\underset{v}{[}\ \ \underset{v}{]}\ \ \underset{u}{[}\ \ \underset{u}{]}$

$\underset{v}{[}\ \ \underset{u}{[}\ \ \underset{u}{]}\ \ \underset{v}{]}$ - back edge

$\underset{u}{[}\ \ \underset{v}{[}\ \ \underset{v}{]}\ \ \underset{u}{]}$ - forward edge