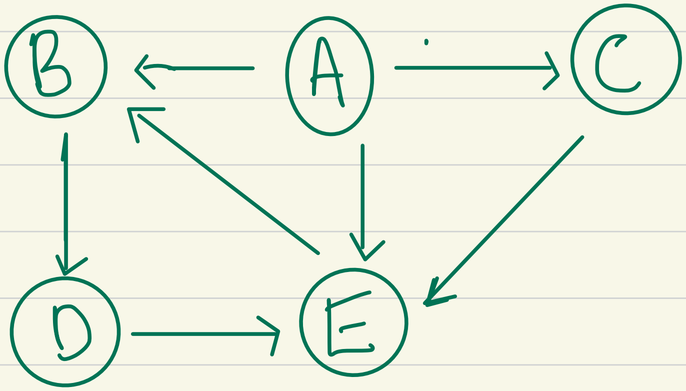
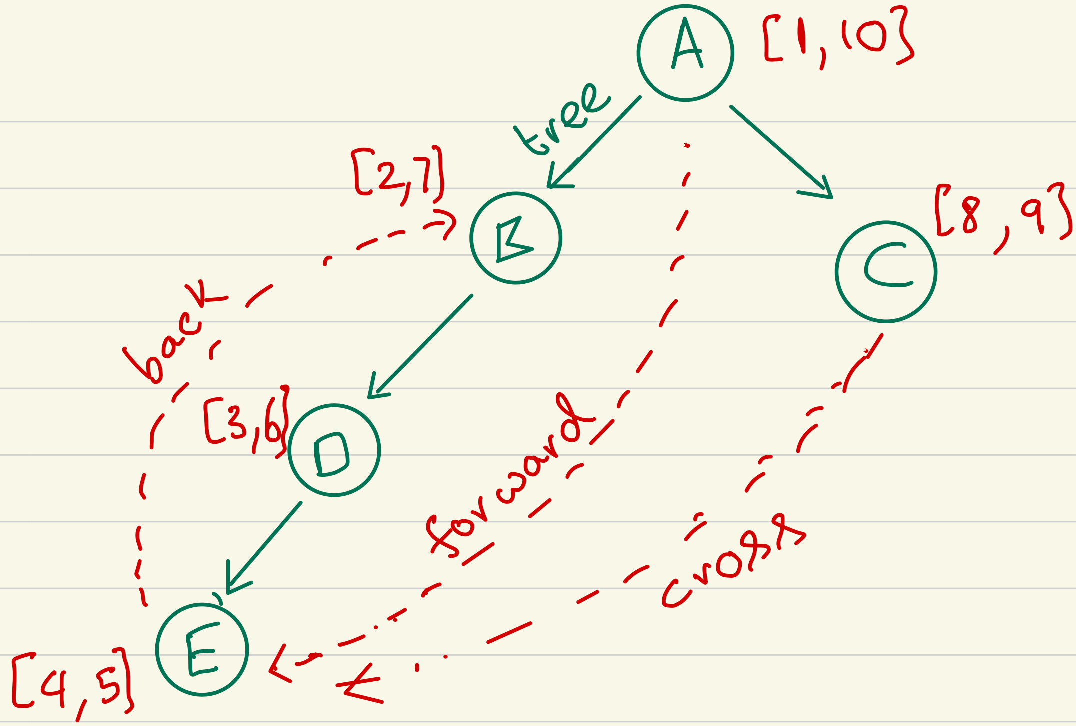
LECTURE 6

DFS in DIRECTED GRAPHS

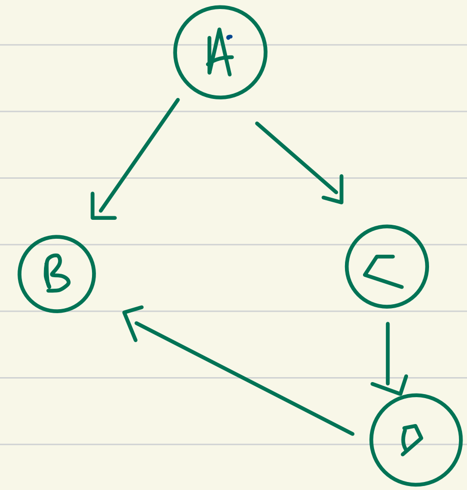


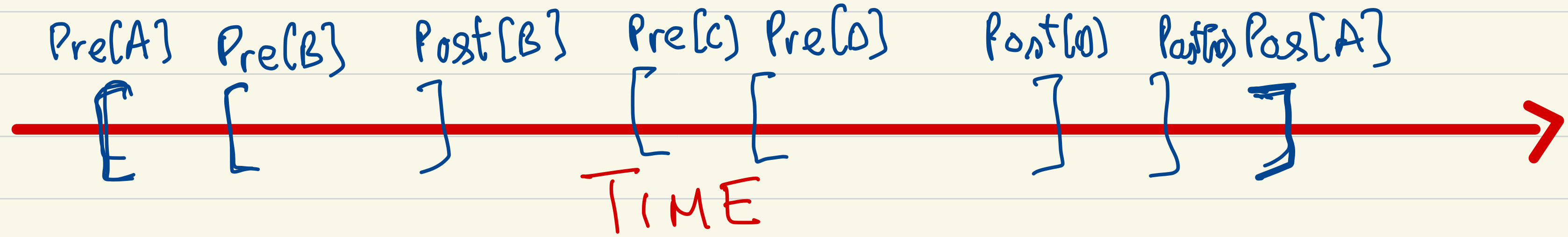



```
EXPLORE(vertex v)

    visited[v] = TRUE
    pre[v] = clock
    clock = clock + 1

    for each edge (v,w) ∈ E
        if NOT visited[w]
            EXPLORE(w)
    post[v] = clock
    clock = clock + 1
```

```
DFS(Graph G)

    visited[u] = FALSE ∀u ∈ V
    clock = 0
    int array pre[n]  post[n]

    for v ∈ V
        if NOT visited[v]
            EXPLORE(v)
```

PRE & POST NUMBERS





Edge: $u \longrightarrow v$

$[_u \ [_v \quad]_v \ ]_u$ = Tree OR Forward Edge

$[_v \ [_u \quad]_u \ ]_v$ = Back Edge

$[_u \ ]_u \ [_v \ ]_v$ = IMPOSSIBLE

$[_v \quad]_v \; [_u \quad]_u$ = CROSS EDGE

$[_u \quad [_v \quad]_u \quad]_v$ = IMPOSSIBLE

Observation: For all edges $u \to v$

$post[u] < post[v]$ if and only if

$u \to v$ is a back edge

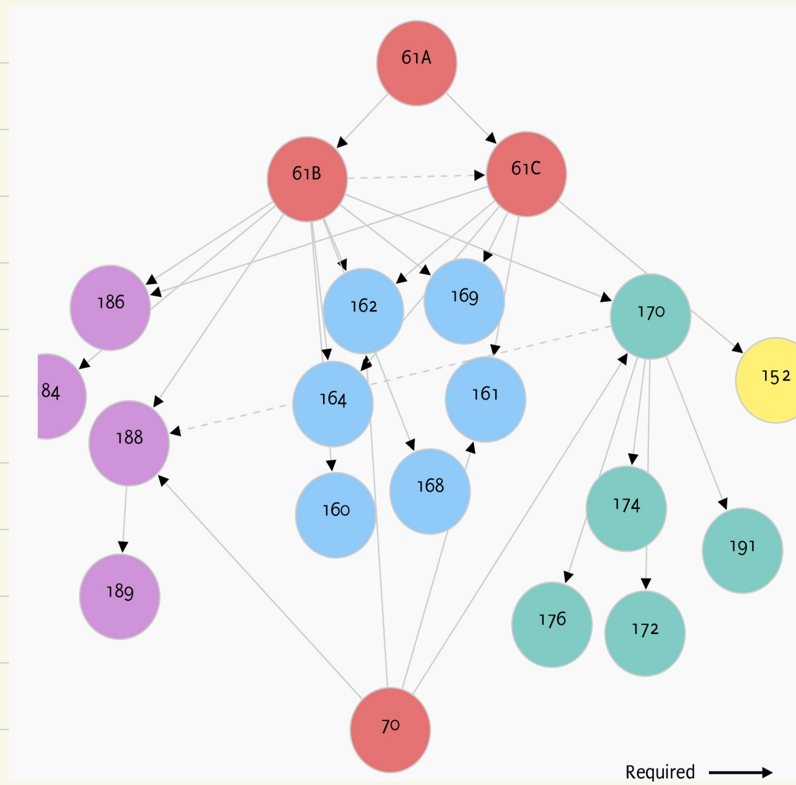
DIRECTED ACYCLIC GRAPHS

"Directed graphs with NO directed cycles"

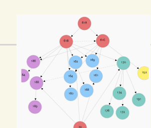
Application:

1) Modelling Dependencies / Pre-requisites

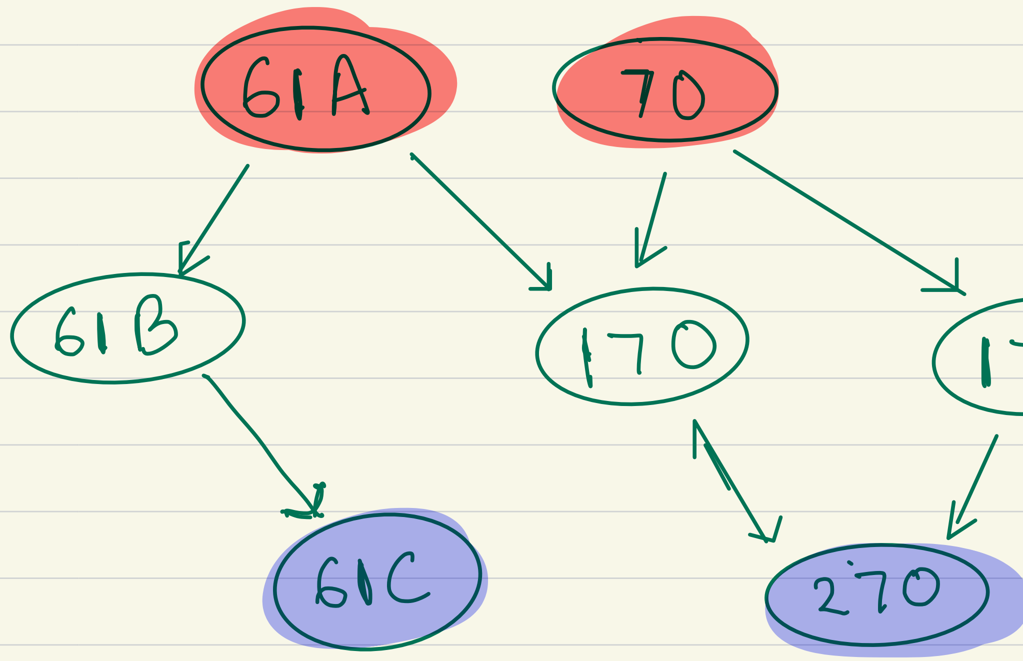
Example 1: Course prerequisites



Example 2: Source files for Compilation



EXAMPLE:



SOURCE NODE:

No incoming edges

SINK Node:

No Outgoing edges

FACT: Every DAG has at least ONE SOURCE AND at least ONE SINK.

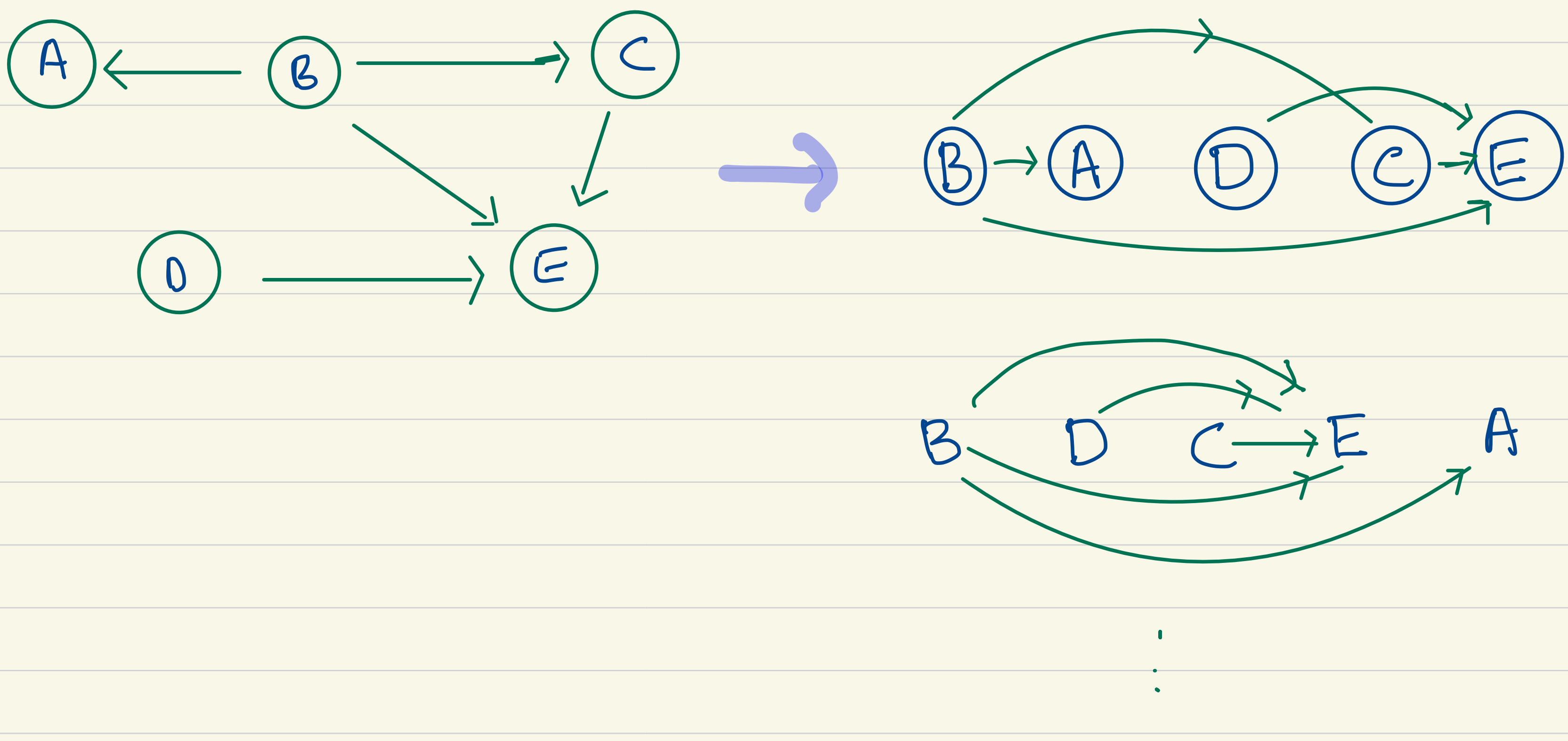
[Excercise: Convince yourself by proof]

TOPOLOGICAL SORT [LINEARIZATION]

INPUT : A directed acyclic graph $G = (V, E)$

OUTPUT : An ordering of vertices so ALL edges go from LEFT to RIGHT

[a.k.a. "LINEARIZE" the DAG]



TOPOLOGICAL SORT

INPUT : A directed acyclic graph $G = (V, E)$

OUTPUT : An ordering of vertices so ALL edges go from LEFT to RIGHT

ALGORITHM:

1) Run DFS to compute Pre & Post values

2) Sort vertices in decreasing post values

⋮

PROOF OF CORRECTNESS:

1) By Observation earlier

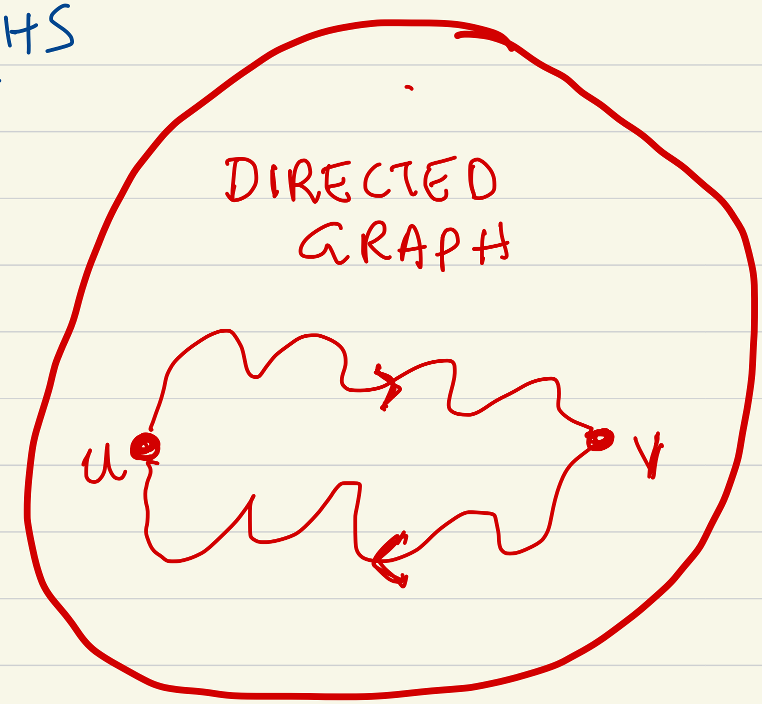
$post[u] < post[v]$ if and only if $u \to v$ is a back edge

2) DAG has NO back edges

$\Rightarrow$ $\forall$ edges $u \to v$ $post[u] > post[v]$

$\Rightarrow$ All edges are from LEFT to RIGHT when vertices are ordered in decreasing post values

CONNECTIVITY IN DIRECTED GRAPHS



DEFINITION:

u is STRONGLY-CONNECTED to v

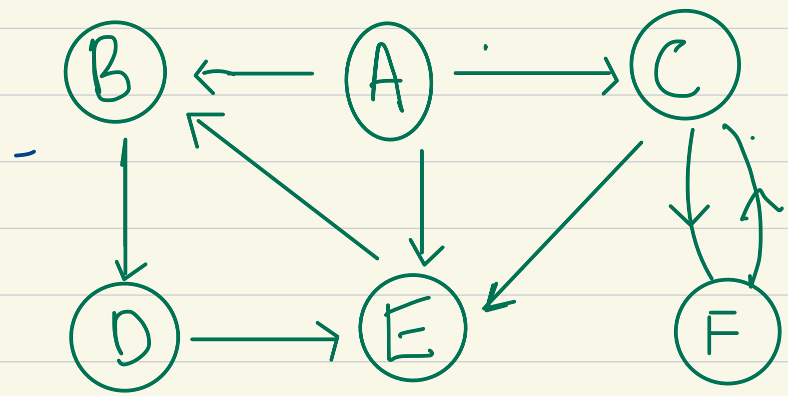
$\iff \exists$ a path $u \rightsquigarrow v$

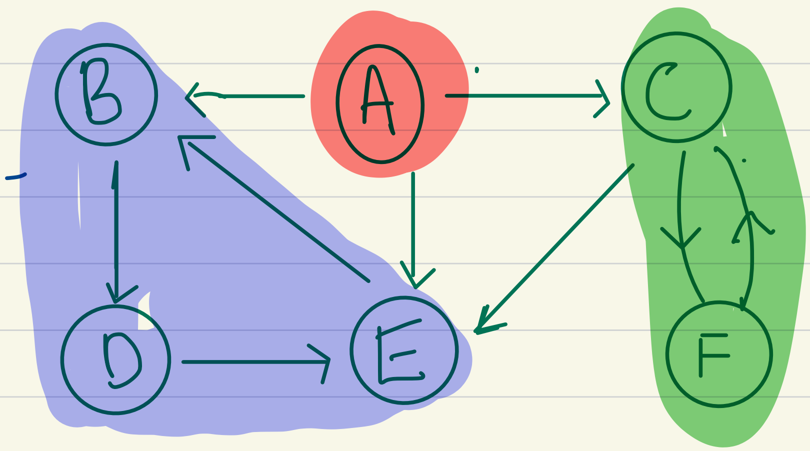
AND

$\exists$ a path $v \rightsquigarrow u$

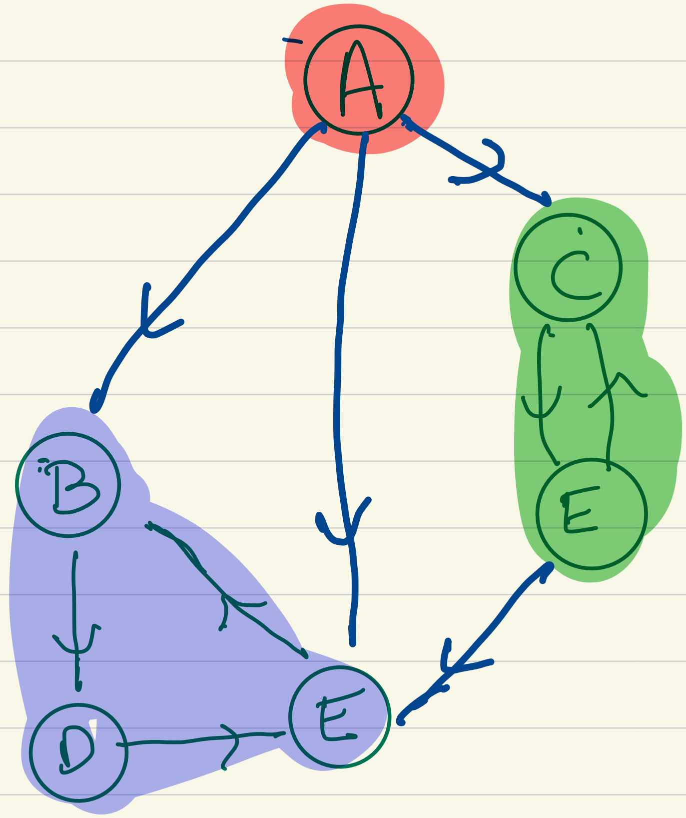
EXAMPLE:

	STRONGLY CONNECTED?
B, E	YES
A, E	NO
B, D	YES





$\iff$



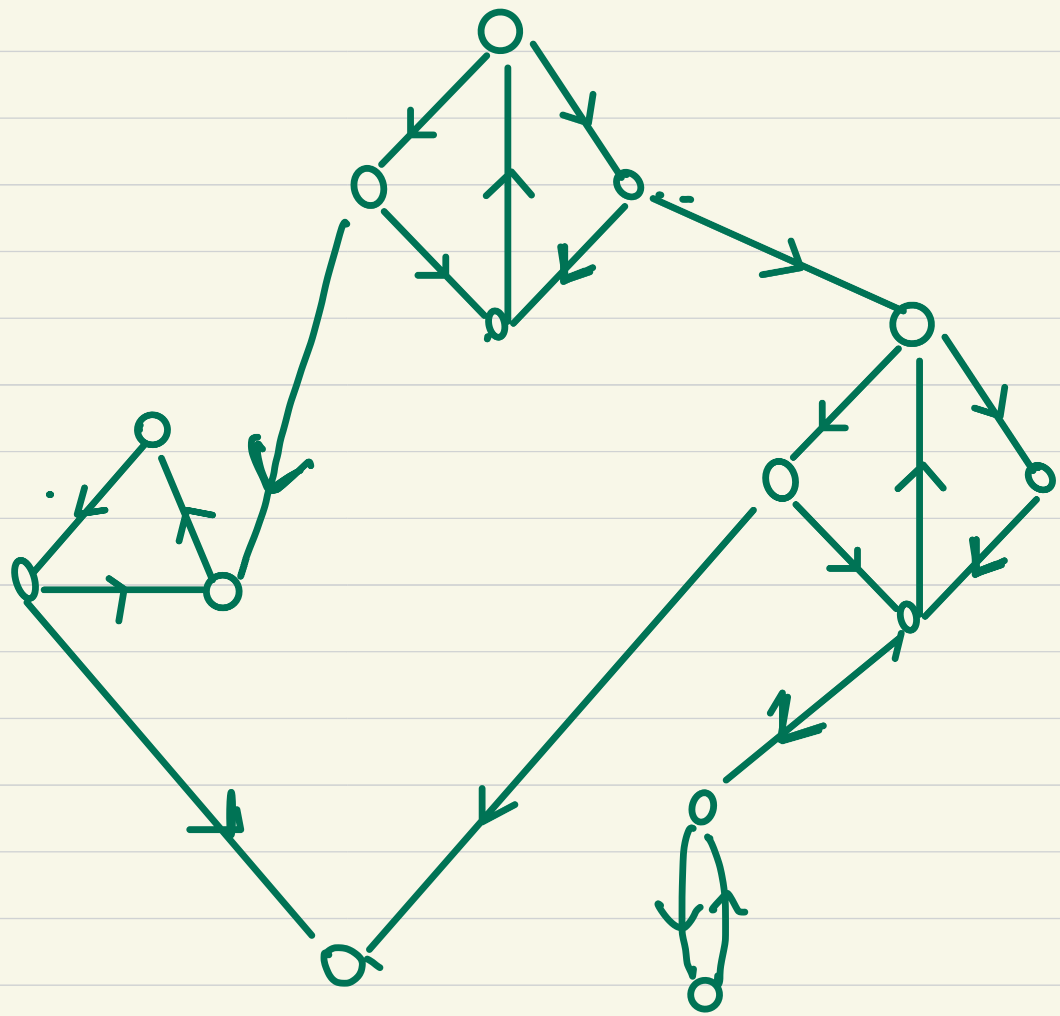
FACT:

EVERY DIRECTED GRAPH can be DECOMPOSED as a DAG of Strongly Connected Components (SCC).

DECOMPOSING DIRECTED GRAPHS

Input: Directed Graph $G = (V, E)$

GOAL: Decompose G into DAG of SCCs.

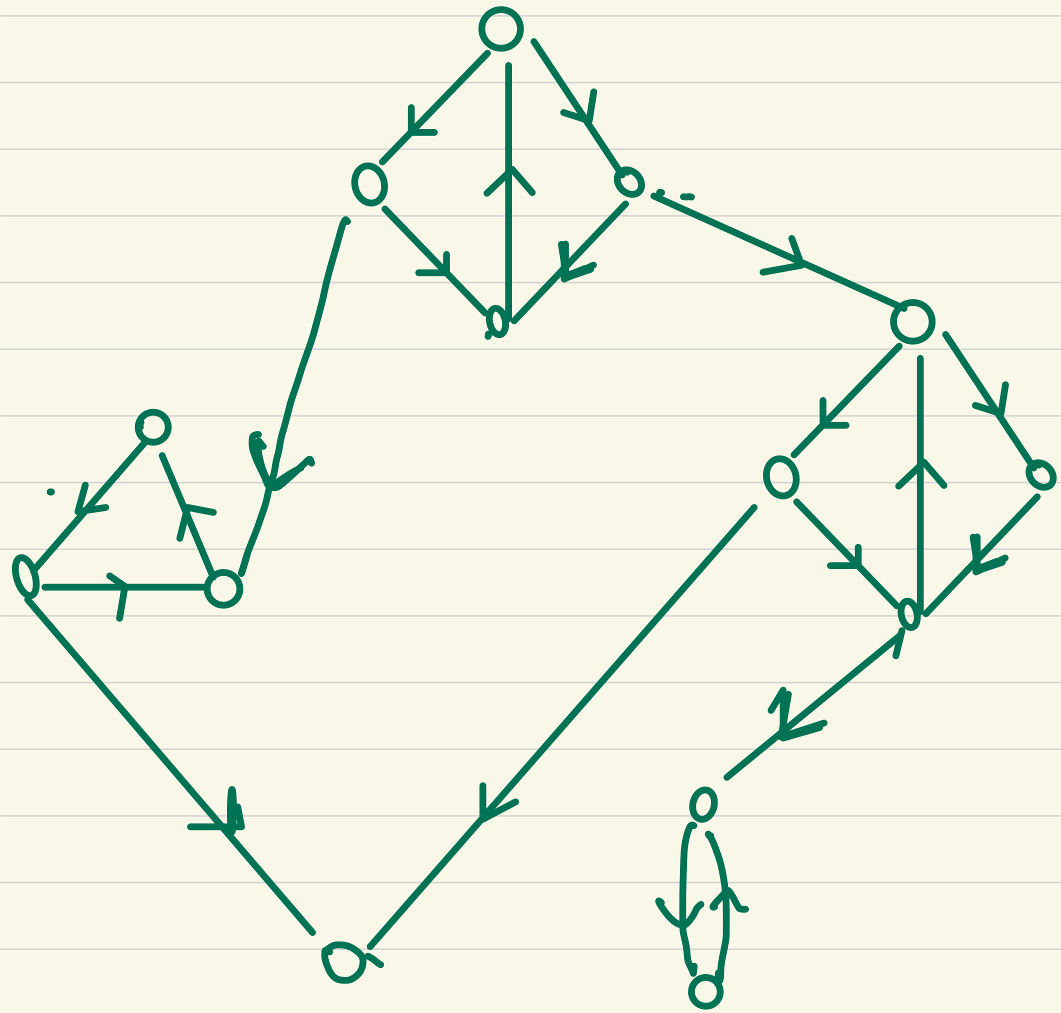


INTUITION:

IDEA:

1) RUN explore(v) for SOME vertex v in SOME SINK SCC

2) REMOVE SINK SCC & REPEAT



PROBLEM:

How do we find a vertex in a SINK SCC?

FACT: In a DFS traversal, vertex with highest POST value is in a SOURCE SCC.

BUT WE WANT: A NODE IN SINK SCC ??

IDEA: 1) RUN DFS on G_R = G with edges reversed.

2) Output vertex v with HIGHEST POST VALUE

$v \in$ SOURCE SCC in G_R $\equiv$ SINK SCC in G !!

KOSARAJU'S ALGORITHM

INPUT: Directed Graph $G = (V, E)$
OUTPUT: Decompose G into DAG of SCCs.

1) $G_R \leftarrow G$ with edges reversed.
2) $pre_R[v]$, $post_R[v]$ for all $v \in V \Longleftarrow DFS(G_R)$
3) DFS on G exploring in decreasing $post_R$ order
   a) $visited[v] = FALSE \quad \forall v \in V$
   b) $count = 0$, $ccnum[1..n]$: int array.
   b) for vertices v in decreasing $post_R$ order
      if NOT $visited[v]$
         $explore(v)$
         $count = count + 1$

Explore(v):

visited[v] = TRUE

cc num[v] = count

for each edge $v \to w$ do

if NOT visited[w] explore(w)

BREADTH-FIRST SEARCH [BFS]

INPUT: Graph $G = (V, E)$, $s \in V$

OUTPUT: $\forall v \in V$ $dist[v]$ = distance from s to v

$dist[v] \leftarrow \infty \quad \forall v \in V$

$dist[s] = 0$

$Q \leftarrow$ queue with $\{s\}$

while Q NOT EMPTY

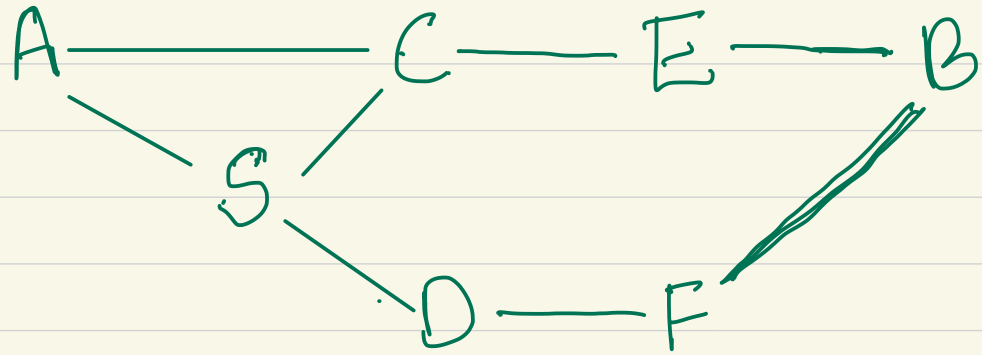
  $v \leftarrow eject(Q)$

  for all edges $v \to w$

    if $(dist[w] = \infty)$

      $dist[w] = dist[v] + 1$

      $Q.add(w)$



QUEUE dist.	S	A	B	C	D	E	F
[S]	0	∞	∞	∞	∞	∞	∞
[A, C, D]	0	1	∞	1	1	∞	∞
[C, D]	0	1	∞	1	1	∞	∞
[D, E]	0	1	∞	1	1	2	∞
[E, F]	0	1	∞	1	1	2	2
[F, B]	0	1	3	1	1	2	2
[F]							

[B]