DFS = Depth first search
Run DFS on undirected graphs to get connected components.

DFS(G):
  input: G = (V, E) in adjacency list representation
  output: vertices labelled by connected components
  CC = Ø
  for all v ∈ V, visited(v) = FALSE
  for all v ∈ V,
    if not visited(v) then
      CC++
      Explore(v)

Explore(z):
  ccnum(z) = CC
  visited(z) = TRUE
  for all (z, w) ∈ E,
    if not visited(w) then Explore(w)

Running time: O(n+m)
  n = |V|
  m = |E|
Directed graphs: we need more information
So add pre/postorder numbers

DFS(G):
input: directed G = (V, E) in adjacency list format
for all \( v \in V \), \( \text{visited}(v) = \text{FALSE} \)
clock = 1
for all \( v \in V \), if not visited \( (v) \) then Explore \( (v) \)

Explore \( (z) \):
\( \text{visited}(z) = \text{TRUE} \)
\( \text{Pre}(z) = \text{clock} \)
clock++
for all \( (z, w) \in E \), if not visited \( (w) \) then Explore \( (w) \)
\( \text{Post}(z) = \text{clock} \)
clock++
Example:

Run DFS starting at B:

Types of edges:
- $z \rightarrow w$: explored edges
- tree edge: $\text{post}(z) > \text{post}(w)$
- back edges: examples $E \rightarrow A, F \rightarrow B$
  $\text{post}(z) < \text{post}(w)$
- forward edges: $D \rightarrow G$
- cross edges: $F \rightarrow H, H \rightarrow G$ $\text{post}(z) > \text{post}(w)$
G has a cycle iff its DFS tree has a back edge.

DAG = directed acyclic graph

- no cycles

Topologically sorting a DAG

= order vertices so that all edges go left → right
  (or lower → higher)

Run DFS on a DAG.

No back edges, so for all edges $z \rightarrow w$,

$\text{Post}(z) > \text{Post}(w)$

Topological sorting algorithm = Sort vertices by $\text{Postorder}$
Source vertex = no incoming edges
Sink = no outgoing edges

In a DAG,
 lowest post # is a sink (might be other sinks)
 highest post # is a source (might be others)

Alternative topological sorting algorithm:
1) Find a sink, output it, and delete it.
2) Repeat (1) until the graph is empty.

General directed graph $G=(V,E)$,
vertices $v$ & $w$ are strongly connected if
there is a path $v$ to $w$
& a path $w$ to $v$

$SCC =$ strongly connected component
= maximal set of strongly connected vertices.
Example:

SCCs: 5A5, 3BEJ, 7C, F, G, 7D, 9H, I, J, K, L

New metagraph: vertex for each SCC, edge from SCC S to S' if some vεS & wεS' has v→w

The metagraph is a DAG.
Every directed graph is a DAG of its SCCs.
Let's find the SCCs & topologically sort these SCCs.
Approach: Find sink SCC, output it, remove it & repeat

How to find a sink SCC?
Take any vertex \( v \) in a sink SCC \( S \).
Run Explore(\( v \)) — only visit \( S \) (no more).

How to find \( v \) in a sink SCC?
Maybe \( v \) with lowest post #?
No:

\[
\text{DFS from A: } \\
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C}
\end{array}
\]

B has lowest post # but \( FA,B \) is not a sink SCC.

But: Vertex with highest post # lies in a source SCC.

We'll prove it later. Now we can do our algorithm.
How to get a vertex in a sink SCC?

Look at $G^R$ = reverse of $G$.

For $G = (V,E)$, let $G^R = (V,E^R)$
where $E^R = \{ w \rightarrow v : v \rightarrow w \in E \}$

= reverse of every edge in $G$

Source SCC in $G$ = Sink SCC in $G^R$
Sink SCC in $G$ = Source SCC in $G^R$

**SCC algorithm:**

For input $G = (V,E)$,
1. Construct $G^R$
2. Run directed DFS on $G^R$
3. Order vertices $V$ by $\downarrow$ Post #
4. Run undirected connected components algorithm on $G$
   (this is just DFS with $\text{ccnum}(v)$)

Running time: $O(n+m)$. 
Proof of key fact:

Vertex with highest post # lies in a source SCC.

First simpler claim:
if $S$ & $S'$ are SCC's, and if there is an edge from $V(S)$ to $V(S')$
then $\max_{S} \text{post #} > \max_{S'} \text{post #}$
in $S$
in $S'$

Assuming the claim,
topological sort the SCC's by the max post # in each.
The first SCC is a source SCC since it's a topological sorting and it has the vertex with max post #.
So that proves the key fact.
Just need to prove the claim.
There is a path $S \rightarrow S'$ (since $\text{yes} \rightarrow \text{west}$)
So no path $S' \rightarrow S$ otherwise $SUS'$ is a SCC
Let $z$ be the 1st vertex in $SUS'$ visited by DFS.

Case 1: $z \in S'$
We see all of $S'$ before visiting any of $S$
& finish them
Thus all post # of $S'$ in $S' < all post # of $S$.

Case 2: $z \in S$
When we Explore($z$) we see all of $SUS'$
before finishing $z$
Post # of $z > Post # of all of $SUS'$

Since $z \in S$ we are done.