DFS = depth first search

Run DFS on undirected graphs to get connected components.

<u>DFS(G):</u>

input: $G=(V,E)$ in adjacency list representation
output: vertices labelled by connected components

$CC = 0$
for all $v \in V$, visited(v) = FALSE

for all $v \in V$,
    if not visited(v) then
        $CC{+}{+}$
        Explore(v)

<u>Explore(z):</u>

ccnum$(z) = CC$
visited(z) = TRUE
for all $(z,w) \in E$,
    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

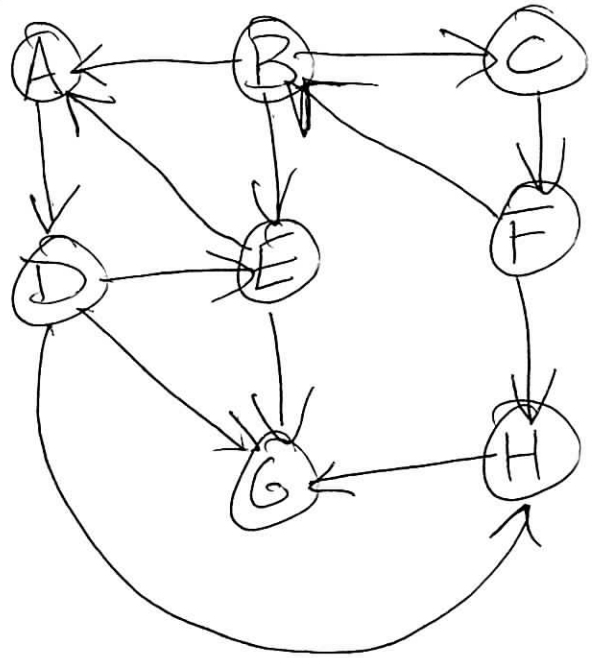
<u>DFS(G):</u>
input: directed $G=(V,E)$ in adj. list format

for all $v \in V$, visited(v) = FALSE
clock = 1
for all $v \in V$, if not visited(v) then Explore(v)

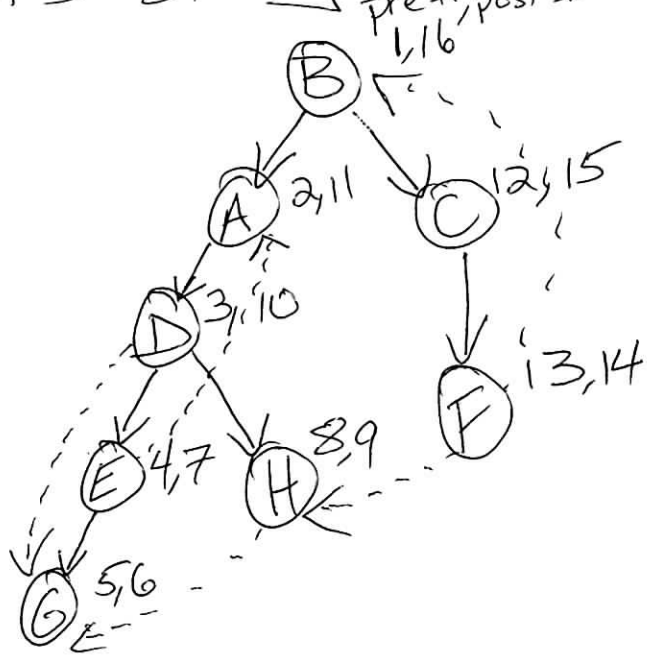
<u>Explore(z):</u>
visited(z) = TRUE
Pre(z) = clock
clock++
for all $(z,w) \in E$,
if not visited(w) then Explore(w)

Post(z) = clock
clock++

Example:



Run DFS starting at B:



Types of edges: $z \to w$

explored edges → tree edge: $post(z) > post(w)$

back edges: examples $E \to A$, $F \to B$
$post(z) < post(w)$

forward edges: $D \to G$
cross edges: $F \to H$, $H \to G$
] $post(z) > post(w)$

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 $z \to w$,
$\text{post}(z) > \text{post}(w)$

topological sorting algorithm = Sort vertices by
↓ postorder #

Source vertex = no incoming edges
Sink = no outgoing edges

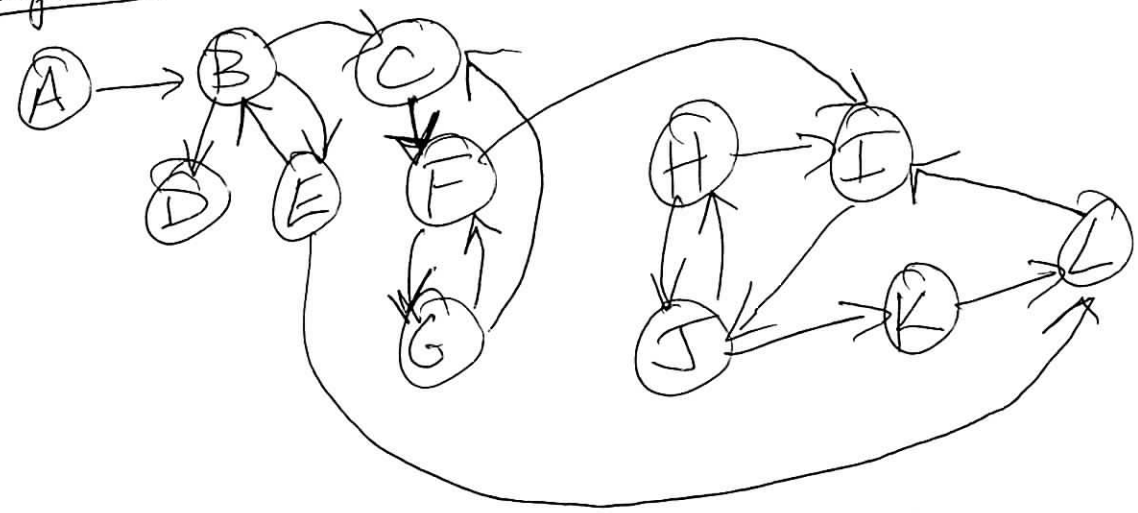
In a DAG,
lowest post # is a sink (might be other sinks)
highest post # is a source (might be others)

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

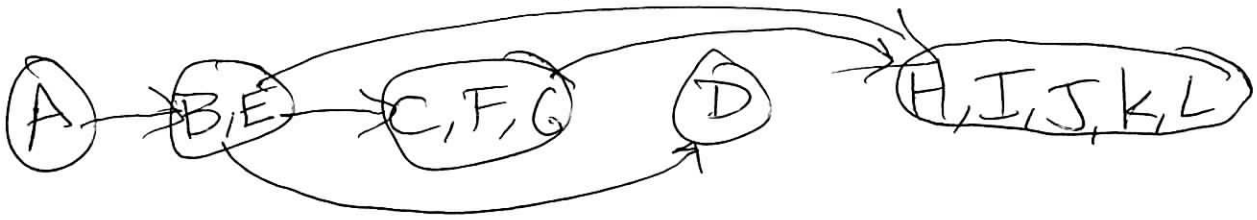
SCC = strongly connected component
= maximal set of strongly connected vertices.

Example:



SCCs: $\{A\}, \{B,E\}, \{C,F,G\}, \{D\}, \{H,I,J,K,L\}$

New metagraph: vertex for each SCC
edge from SCC S to S'
if some $v \in S$ & $w \in S'$ has $v \to w$



The metagraph is a DAG.

Every directed graph is a DAG of its SCCs.

Let's find the SCCs & topologically sort these SCC's.

<u>Approach</u>: Find sink SCC, output it, remove it, & repeat

How to find a sink SCC?

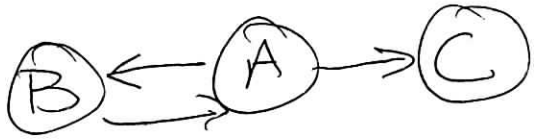
Take any vertex v in a sink SCC S.

Run Explore(v) — only visit S (no more).

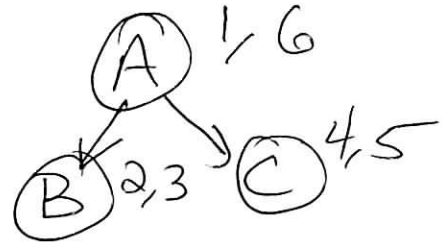
How to find v in a sink SCC?

Maybe v with lowest post #?

NO:



DFS from A:



B has lowest post # but $\{A, B\}$ is not a sink SCC.

<u>But</u>: 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 $E^R = \{w \to v : v \to w \in E\}$

= reverse of every edge in G

Source SCC in G = sink SCC in G^R

Sink SCC in G = source SCC in G^R

<u>SCC algorithm:</u>

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 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 $v \in S$ to $w \in S'$

then max post # in S > max post # 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 $S \to S'$ (since $v \in S \to w \in S'$)

So no path $S' \to S$ otherwise $S \cup S'$ is a SCC.

Let z be the 1st vertex in $S \cup S'$ visited by DFS.

<u>Case 1:</u> $z \in S'$

We see all of S' before visiting any of S & finish them

thus all post #'s in S' < all post #'s in S.

<u>Case 2:</u> $z \in S$.

When we Explore(z) we see all of $S \cup S'$ before finishing z

Post # of z > Post # of all of $S \cup S' - \{z\}$.

Since $z \in S$ we are done.