Graphs:

$G = (V, E)$ V = vertices $n = |V|$

E = edges $m = |E|$

Undirected graphs: edge $(i, j) \in E$ means i & j connected by an edge

Directed graphs: $\vec{ij} \in E$ means edge from i to j

Representing graphs:

1) Adjacency matrix A size $O(n^2)$
2) Adjacency list size $O(n+m)$

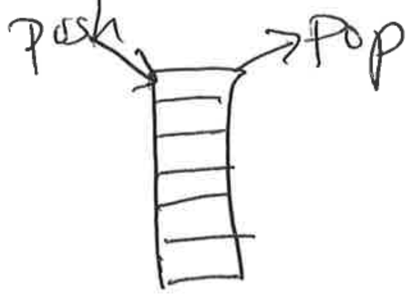
Exploring graphs:

DFS = depth first search

BFS = breadth first search.

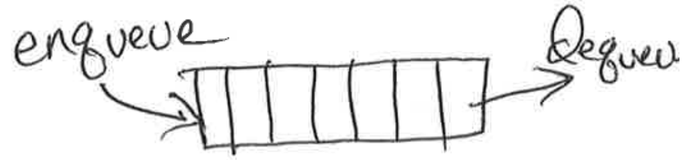
DFS:

stack = LIFO
= last-in, first-out

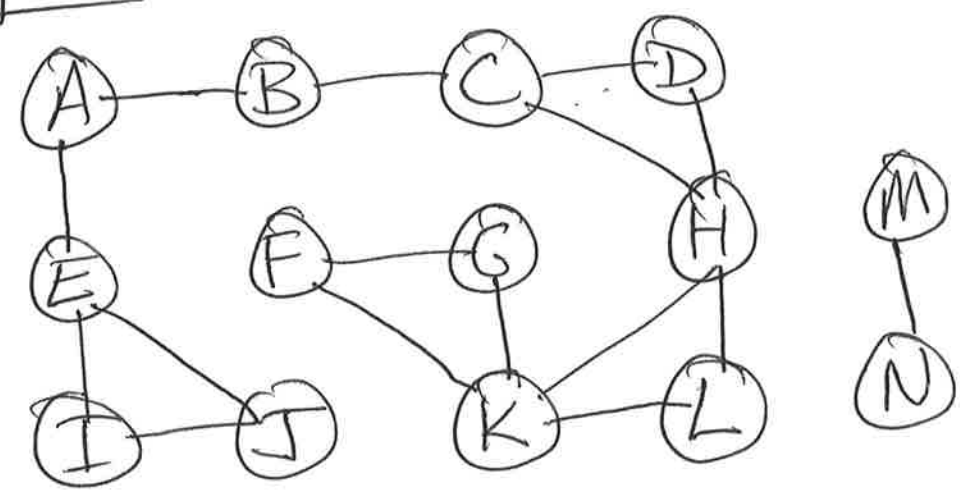


BFS:

queue = FIFO
= first-in, first-out

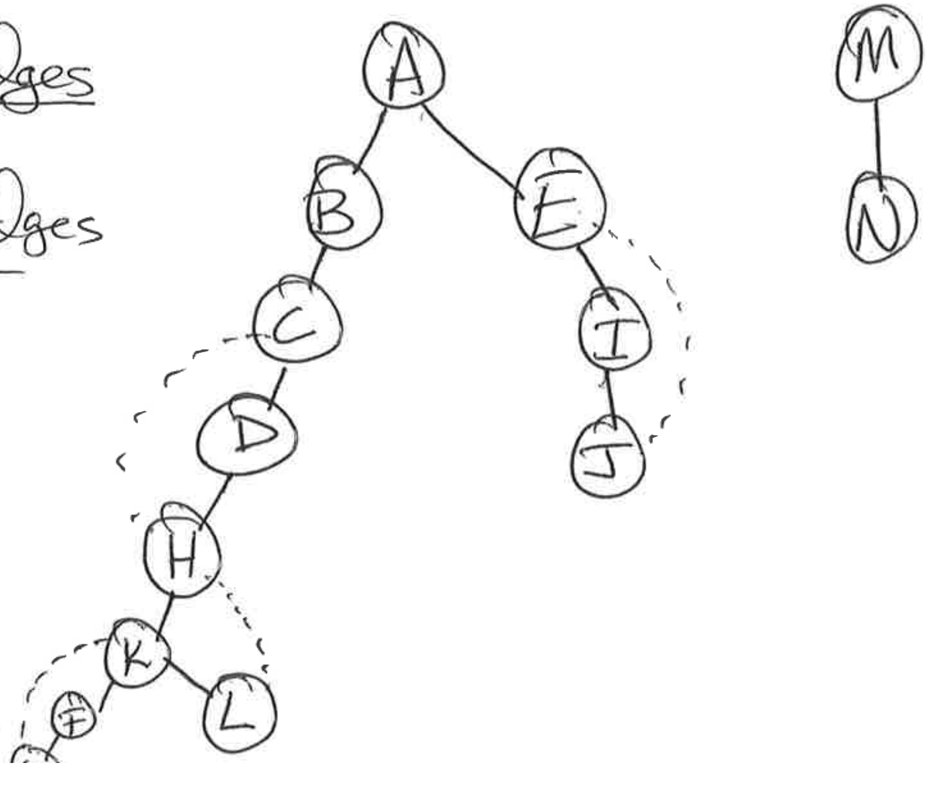


Example:



tree edges

back edges



DFS: Can implement a stack using recursion.

Pseudocode:

DFS(G):

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

~~Explore(~~

for all $v \in V$, if not visited(v)

then Explore(v)

Explore(w):

visited(w) = TRUE

for all $(w, z) \in E$:

if not visited(z)

then Explore(z).

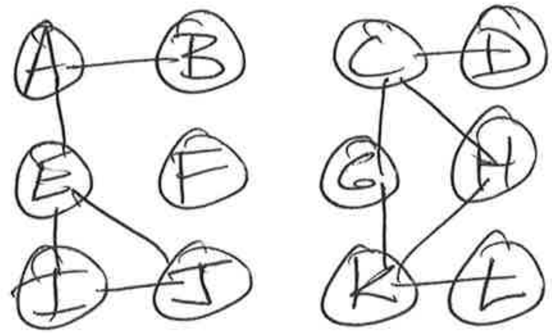
Explore(z): finds all vertices reachable from z.

Can use to find connected components.

For <u>undirected</u> G, vertices v & w are <u>connected</u>
if: there is a path between v & w.

Connected components: Maximal set of connected vertices.

<u>Example:</u>



3 components: {A, B, E, I, J}, {F}, {C, D, G, H, K, L}

To find connected components:

1) Choose arbitrary start vertex Z
2) Run ~~~~ Explore (Z) to find component containing Z.
3) Choose an unexplored Z & repeat.

Same pseudocode as DFS just keep track of connected component #.

Here's the pseudocode:

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

```
for all v∈V, visited(v) = FALSE
cc = 0
for all v∈V,
    if not visited(v)
        then [ cc++
               Explore(v)
```

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

```
visited(z) = TRUE
ccnum(z) = cc
for each (z,w)∈E:
    if not visited(w)
        then Explore(w)
```

<u>Running time:</u> $O(n+m)$, for G in adjacency list representation.

How about connectivity in directed graphs?

- Need more info from DFS.
  - add a clock
  - keep track of preorder # & postorder #

    $Pre(v)$ = time start exploring v

    $Post(v)$ = time finish exploring all neighbors of v.

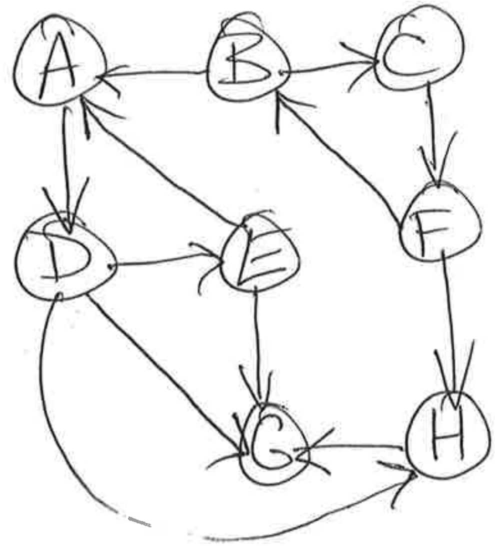
DFS(G):

```
for all v∈V, visited(v) = FALSE
clock = 1
for all v∈V,
    if not visited(v)
        then Explore(v)
```

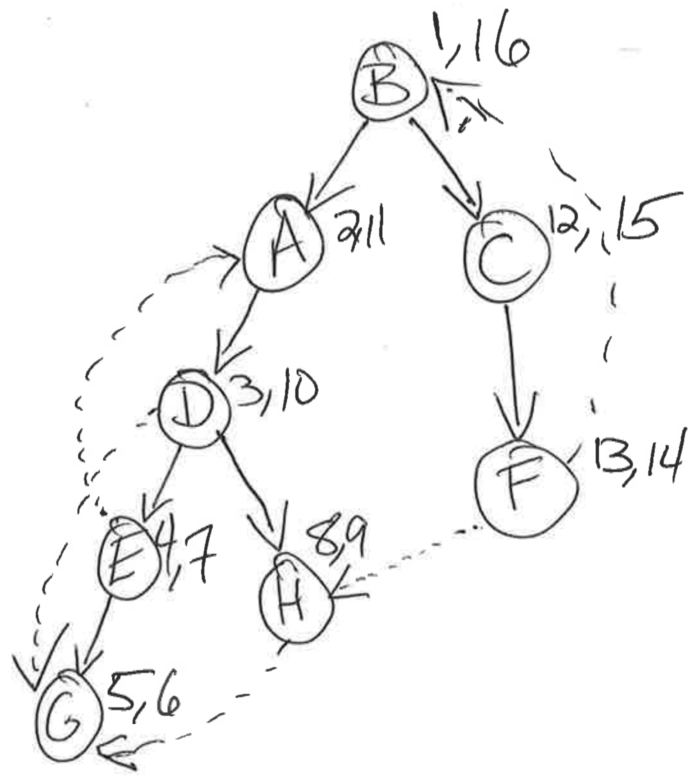
Explore(z):

```
Visited(z) = TRUE
Pre(z) = clock
clock++
for every (z,w)∈E
    if not visited(w) then Explore(w)
Post(w) = clock
clock++
```

Example:



Let's run DFS starting at B



3 types of nontree (unexplored) edges

Forward: $D \to G$: vertex to descendant

$$\underbrace{pre(D) < \underbrace{pre(G) < post(G)}_{G} < post(D)}_{D}$$

b/c G is in subtree of G

Back: $E \to A$, $F \to B$: vertex to ancestor

$$\underbrace{pre(B) < \underbrace{pre(F) < post(F)}_{F} < \overset{post}{\cancel{pre}}(B)}_{B}$$

F is in B's subtree

Cross: $F \to H$, $H \to G$

$$\underbrace{pre(H) < post(H)}_{H} < \underbrace{pre(F) < post(F)}_{F}$$

start & finish H before start/finish F.

<u>Property</u>: Directed G has a cycle iff its DFS tree contains a back edge.

<u>Proof</u>:

$\Rightarrow$ if G has a cycle C, let v be 1st vertex in C visited in the DFS. Then $C \setminus v$ is reachable from v so all are in v's subtree of the DFS tree.
& ≥ 1 edge from $C \setminus v$ to v which is a back edge.

$\Leftarrow$ Say the DFS tree has a back edge $w \to v$. Then w is a descendant of v so there is a path P from v to w, & taking $P \cup (w, v)$ gives a cycle.

$\boxtimes$

So if no cycles in directed $\vec{G}$ then
no back edges.

For back edge $w \to v$, $\text{Post}(w) < \text{post}(v)$

For all other edges $w \to v$, $\text{Post}(v) < \text{post}(w)$.

So can detect if G has a cycle
by checking post #'s to see if
we find a back edge.

DAG = Directed acyclic graph.

no cycles
hence no back edges

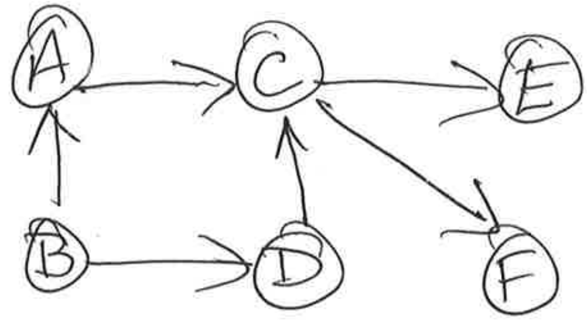
Topologically sorting a DAG
= order the vertices so that
all edges go left to right
(lower #) (higher #)

Easy: sort by decreasing post #

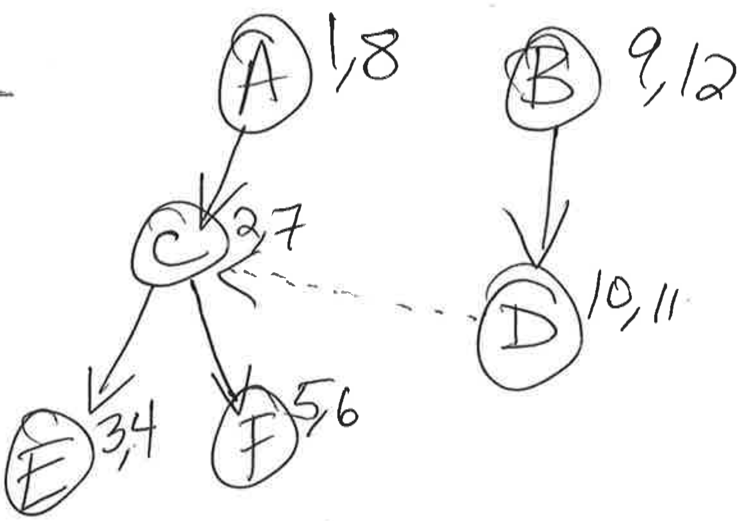
highest post # $\longrightarrow$ lowest post #

Since no back edges,
every edge $w \to v$
has $post(w) > post(v)$
so left $\to$ right.

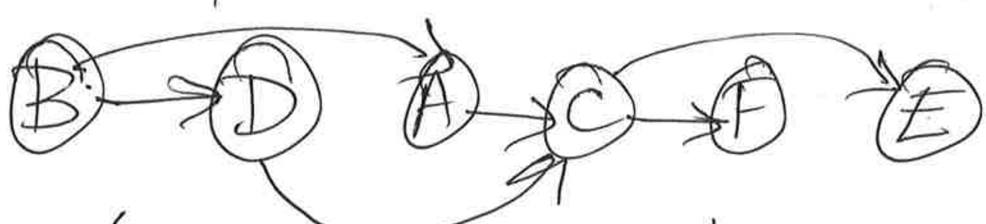
Example:



Run DFS:



Topologically sorted as:



(there are other topological orderings)