**DFSAll(G):**

- clock ← 0
- for all vertices v
- unmark v
- for all vertices v
- if v is unmarked
  - clock ← DFS(v, clock)

**DFS(v,clock):**

- mark v
- clock ← clock + 1; v.pre ← clock
- for each edge v→w
  - if w is unmarked
    - w.parent ← v
    - clock ← DFS(w, clock)
- clock ← clock + 1; v.post ← clock
- return clock

---

**forward edge** u→v

- u.pre < v.pre < v.post < u.post

**it isn’t a tree edge**

**cross edge** u→v

- v.pre < v.post < u.pre < u.post
Back edge $u \rightarrow v$

$\forall v, \text{pre} \leq u, \text{pre} < u, \text{post} < v, \text{post}$

$\Rightarrow u \rightarrow v$ is a back edge

iff $u, \text{post} < v, \text{post}$
Detecting Cycles

Lemma: Directed graph $G$ has a cycle if $\text{DFSAll}(G)$ yields at least one back edge.

Proof: Suppose $u \rightarrow v$ is a back edge. There is a path $v$ of tree edges from $v$ to $u$.
Suppose $G$ has a cycle $C$. Let $v$ be the first vertex of $C$ we explore during DFS. If $u$ is a back edge, then $u \rightarrow v$ is a back edge immediately before $v$ on $C$. During call $DFS(v)$, we will reach every unmarked vertex reachable from $v$. We call $DFS(u)$. 
Cycle Detection Algo:
Run DFSAll(G) to find a postorder.
Return true if there exists an edge \( \text{upost}_v \rightarrow \text{post}_v \) in \( E \).
Topological Sort

Given directed graph $G = (V,E)$, a topological ordering of its vertices is a total ordering of the vertices where $u < v$ if $u < v E$, i.e., you can draw the graph so edges only go left to right.

$\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c}
\end{array}$
Cycle $\Rightarrow$ no topological ordering.

Directed acyclic graph (DAG) if directed $\Rightarrow$ no cycles.

Every DAG has a top. order!

Proof: Run DFSAll, No back edges $\Rightarrow$ for all edges $u \rightarrow v$, 
Consider the reversed postorder. $w = v$ for each edge $u \rightarrow v$, so it is a top order.

Algorithm: Run DFSAI.

Return reversed post order.

$O(V+E)$ time

\begin{verbatim}
TopologicalSort(G):
    for all vertices $v$
        $v.status$ $\leftarrow$ New
    clock $\leftarrow V$
    for all vertices $v$
        if $v.status$ = New
            clock $\leftarrow$ TopSortDFS($v$, clock)
    return $S[1..V]$
\end{verbatim}

\begin{verbatim}
TopSortDFS($v$, clock):
    $v.status$ $\leftarrow$ Active
    for each edge $v \rightarrow w$
        if $w.status$ = New
            clock $\leftarrow$ TopSortDFS($w$, clock)
    else if $w.status$ = Active
        fail gracefully
    $v.status$ $\leftarrow$ Finished
    $S[clock] \leftarrow v$
    clock $\leftarrow$ clock $-$ 1
    return clock
\end{verbatim}
Dynamic Programming

Suppose we have a recurrence. The dependency graph has the subproblems as its vertices and edges \( x \rightarrow y \) for each direct call for a subproblem \( y \) from a subproblem \( x \).

Cycle \( \Rightarrow \) infinite loop of recursive calls.
So good recurrences have DAGs as dependency graphs.

If we do recursion with basic memorization, we’re doing a depth-first search.

So the subproblems are solved in post order.
The dynamic programming algorithms are really solving each subproblem in postorder. (usually we just say what the postorder is)
**Longest Path problem**

Given: A DAG $G = (V, E)$ with edge weights $l : E \to \mathbb{R}$,

Want length of longest path from given $s$ to given $t$.

$LLP(v):$ length of longest path from $v$ to $t$ or $-\infty$ if no $v$-$t$ path exists.
If $v = t$, $LLP(v) = 0$.

0.w.

$$L_{LLP}(v) = \begin{cases} 0 & \text{if } v = t \\ \max_{v \to w \in E} \{ \ell(v \to w) + LLP(w) \} & \text{otherwise} \end{cases}$$

(max over nothing $= -\infty$)

$G$ is the dependency graph for $LLP$.

\begin{algorithm}
\textsc{LongestPath}(s, t):
  for each node $v$ in postorder
    if $v = t$
      $v.LLP \leftarrow 0$
    else
      $v.LLP \leftarrow -\infty$
      for each edge $v \to w$
        $v.LLP \leftarrow \max\{v.LLP, \ell(v \to w) + w.LLP\}$
  return $s.LLP$
\end{algorithm}

$O(V + E)$ time
Shortest Path in DAG.

\[
\begin{align*}
\text{max} & \rightarrow \text{min} \\
-\infty & \rightarrow +\infty \\
O(V + E) & \text{ time}
\end{align*}
\]