Depth-first Search

DFSALL(G):
\[
clock \leftarrow 0
\]
for all vertices \( v \)
unmark \( v \)
for all vertices \( v \)
if \( v \) is unmarked
\[
clock \leftarrow \text{DFS}(v, clock)
\]

DFS(\( v, \text{clock} \)):
mark \( v \)
\[
clock \leftarrow \text{clock} + 1, \ v.pre \leftarrow \text{clock}
\]
for each edge \( v \rightarrow w \)
if \( w \) is unmarked
\[
w.parent \leftarrow v
\]
\[
clock \leftarrow \text{DFS}(w, \text{clock})
\]
\[
clock \leftarrow \text{clock} + 1, \ v.post \leftarrow \text{clock}
\]
return \( \text{clock} \)

preorder: sort vertices by start time

postorder: sort vertices by finish time
Edge $u \rightarrow v$:

- $u_{pre} < v_{pre} = v_{post} < u_{post}$

If $DFS(u)$ calls $DFS(v)$ directly, $u \rightarrow v$ is a tree edge.

Otherwise, $u \rightarrow v$ is a forward edge.
v.pre = u.pre ≤ u.post = v.post

\[ u \rightarrow v \] is a back edge

v.post ≤ u.pre

\[ u \rightarrow v \] is a cross edge

u.post < v.pre cannot happen!
Given directed graph $G$.
Are there any directed cycles?

**Lemma:** Directed graph $G$ has a (directed) cycle iff $\text{DFSAll}(G)$ yields a back edge.

- Suppose there is a back edge $u \rightarrow v$. $v.\text{pre} < u.\text{pre} = u.\text{post} < v.\text{post}$

  so $u$ is reachable from $v$

$a \leftarrow u$ ∈ a cycle!
Suppose $G$ has a cycle $C$. Let $v$ be the first vertex we visit of $C$. Let $u$ be vertex immediately before $v$ in $C$.

$\text{DFS}(v)$ visits all reachable vertices where nothing on the path is marked (or, by time $v$ is finished all reachable vertices are marked).
so u gets marked during DFS(v)

so v.pre < u.pre < u.post < v.post.

$u \rightarrow v$ is a back edge

Edge $u \rightarrow v$ is a back edge

iff $u.post < v.post$. So

1) Compute finishing times in $O(V + E)$ time via DFS ALL(G).
2) Check if \( u \text{.post} < v \text{.post} \) for any edge \( u \rightarrow v \). \( O(E) \)

3) If so \( \Rightarrow \) cycle

If not \( \Rightarrow \) no cycle

Total: \( O(V+E) \)
Directed acyclic graphs (DAGs) are those without cycles.

Topological ordering:
Find a total order on vertices such that \( u < v \) if there exists an edge \( u \rightarrow v \).

If \( G \) has a directed cycle, there is no topological ordering.
But if there are no cycles...
Then no back edges...
so \( u \cdot \text{post} \geq v \cdot \text{post} \) for every edge \( u \rightarrow v \)
\( \Rightarrow \) reverse postorder is a topological ordering
\( \Rightarrow \) every DAG has a topological ordering

\[
\text{TopologicalSort}(G):
\begin{align*}
&\text{for all vertices } v \\
&v\cdot\text{status} \leftarrow \text{New} \\
&\text{clock} \leftarrow V \\
&\text{for all vertices } v \\
&\quad \text{if } v\cdot\text{status} = \text{New} \\
&\quad \quad \text{clock} \leftarrow \text{TopSortDFS}(v, \text{clock}) \\
&\text{return } S[1..V]
\end{align*}
\]

\[
\text{TopSortDFS}(v, \text{clock}):
\begin{align*}
&v\cdot\text{status} \leftarrow \text{Active} \\
&\text{for each edge } v \rightarrow w \\
&\quad \text{if } w\cdot\text{status} = \text{New} \\
&\quad \quad \text{clock} \leftarrow \text{TopSortDFS}(w, \text{clock}) \\
&\quad \text{else if } w\cdot\text{status} = \text{Active} \\
&\quad \quad \text{fail gracefully} \\
&v\cdot\text{status} \leftarrow \text{Finished} \\
&S[\text{clock}] \leftarrow v \\
&\text{clock} \leftarrow \text{clock} - 1 \\
&\text{return } \text{clock}
\end{align*}
\]
Dynamic Programming

Suppose you have a recurrence to evaluate for a dynamic programming algorithm...

Dependency graph: each subproblem (choice of parameters) is a vertex. Each edge $x \Rightarrow y$ means you make a call to $y$ while handling the call to $x$. 
Dependency graphs must be acyclic.

Basic memoization is like a depth-first search of the dependency graph. Stores answers in postorder.
The iterative DP algs are like "hardwiring" in a particularly clean postorder.
Longest Path: Given directed graph \( G = (V,E) \) with weights on edges \( l:E\rightarrow \mathbb{R} \). Also given \( s,t \).

What is the max length of a simple \( s,t \)-path, (no repeated vertices)

Will assume \( G \) is a DAG.
$\text{LLP}(v)$: longest path length from $v$ to $t$, (\(\infty\) if none exists)

$$\text{LLP}(v) = \begin{cases} 0 & v = t \\ \max \left\{ \ell (v \rightarrow w) + \text{LLP}(w) \mid v \rightarrow w \in E \right\} \\ \left( \max = -\infty \text{ if no edges } v \rightarrow w \right) \end{cases}$$

Want to know $\text{LLP}(s)$. 
Dependency graph is \( G \) itself.

So solve subproblems in postorder for \( G \).

\[
\text{LONGESTPATH}(s, t): \\
\text{for each node } v \text{ in postorder} \\
\quad \text{if } v = t \\
\quad \quad v.LLP \leftarrow 0 \\
\quad \text{else} \\
\quad \quad v.LLP \leftarrow -\infty \\
\quad \quad \text{for each edge } v \rightarrow w \\
\quad \quad \quad v.LLP \leftarrow \max \{v.LLP, \ell(v \rightarrow w) + w.LLP\} \\
\text{return } s.LLP
\]

\( O(V+E) \) time \( \O(1) \) per edge across all loops.