Depth-first search (DFS)

\[
\text{DFS}(v):
\]
mark \( v \)
\[ \text{PREVISIT}(v) \]
for each edge \( vw \)
if \( w \) is unmarked
\[ \text{parent}(w) \leftarrow v \]
DFS(\( w \))
\[ \text{POSTVISIT}(v) \]

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

\[ O(V + E) \]
**DFS**

DFSALL($G$):

$\text{clock} \leftarrow 0$

for all vertices $v$

unmark $v$

for all vertices $v$

if $v$ is unmarked

$\text{clock} \leftarrow \text{DFS}(v, \text{clock})$

DFS($v, \text{clock}$):

mark $v$

$\text{clock} \leftarrow \text{clock} + 1$; $v.pre \leftarrow \text{clock}$

for each edge $v \rightarrow w$

if $w$ is unmarked

$w.parent \leftarrow v$

$\text{clock} \leftarrow \text{DFS}(w, \text{clock})$

$\text{clock} \leftarrow \text{clock} + 1$; $v.post \leftarrow \text{clock}$

return $\text{clock}$

---

- add a clock to learn about visitation order
  - $v.pre$: starting time of $v$
  - $v.post$: finishing time of $v$

$[v.pre, v.post]$

active interval

↑

all are nested or disjoint pairs
If DFS(v) called while u is active $\Rightarrow$ there is a $u,v$-path.

Sort by x.pre for a preorder.

Sort by x.post for a postordering.
We're midway through running DFS on. We have a current clock value. Will compare to final pre and post values.

vertex v is:

- new if clock = v.pre
- active if v.pre ≤ clock ≤ v.post
- finished if v.post ≤ clock

active vertices form a directed path in G
Partitioning edges:

Consider edge \( u \rightarrow v \) at moment \( DFS(u) \) begins...

(clock = \( u, \text{pre} \))

If \( v \) is new,

\[
\text{\( u, \text{pre} = v, \text{pre} < v, \text{post} = u, \text{post} \)}
\]

If \( DFS(w) \) directly calls \( DFS(v) \), \( u \rightarrow v \) is a \underline{tree edge}

\[ o.w. \text{ \( w \rightarrow v \) is a forward edge} \]

If \( v \) is active, \( v \) is on stack

\[
\text{\( v, \text{pre} = \text{pre} \rightarrow v, \text{pre} = u, \text{post} = v, \text{post} \)}
\]

\( u \rightarrow v \) is a \underline{back edges}
If \( v \) is finished,
\[
v \text{.post} < w \text{.pre}
\]
\( u \leftrightarrow v \) is a cross edge

The classification depends on how the DFS runs!!
Detecting Cycles in Directed Graphs

Lemma: Directed graph \( G \) has a cycle iff DFSAll (\( G \)) yields a back edge.

Proof: Suppose \( u \rightarrow v \) is a back edge, \( v \) was active when we called DFS (\( u \)).

\( \Rightarrow \) There is a \( uv \)-path
Suppose there is a cycle $C$. Let $v$ be the first vertex of $C$ marked by DFS after.

$u$: predecessor of $v$ on $C$.

$\Downarrow$ $\text{DFS}(v)$ eventually calls $\text{DFS}(u)$.

$\Rightarrow u \Rightarrow v$ is a back edge.
$u \Rightarrow v$ is a back edge iff $u.post < v.post$.

So compute a postordering. Return "cycle!!" iff $u.post < v.post$ for any $u \Rightarrow v \in E$.

$O(V+E)$ time!
Topological Sort

Given directed $G = (V, E)$,

topological ordering of $G$: a total ordering of vertices where \( u \prec v \) if \( u \to v \in E \)

or

can write all vertex names from left to right so all edges go left to right.
directed cycle \implies no topological ordering

0.w, ... no back edges, ...

\implies u.post = v.post for all \( u \rightarrow v \in E \)

\implies order by decreasing \( x.post \) (reverse postorder) to get a topological ordering
Just reverse order you finish DFS calls, so \(O(V+E)\) time!

**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 \(S[1..V]\)

**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 \(clock\)
Dynamic Programming

Given a recurrence, the dependency graph has one vertex per subproblem and an edge $x \Rightarrow y$ for every direct call of a subproblem $y$ from a subproblem $x$.

Must be acyclic!
If you use basic memoization, you solve problems in post order.

Iterative dynamic prog. algs. solve the problems in some reverse topological order.
Longest Path:
Given a $G = (V, E)$ with edge weights $l: E \rightarrow \mathbb{R}$, two vertices $s \neq t$.

Assume $G$ is a DAG...

$LLP(v)$: length longest path from $v$ to $t$.

Want $LLP(s)$.

$LLP(v) = \begin{cases} 0 & \text{if } v = t, \\ \max \{l(v \rightarrow w) + LLP(w) | v \rightarrow w \in E\} & \text{otherwise,} \end{cases}$
use min instead for shortest paths in a DAG!

recurrance is ill-defined if G has a cycle

The dependency graph is G itself and therefore a DAG

So compute LLP(x) in postorder

Constant time per edge so $O(V+E)$ time!

**LongestPath(s, t):**

for each node $v$ in postorder

if $v = t$

$v.LLP \leftarrow 0$

else

$v.LLP \leftarrow -\infty$

for each edge $v \rightarrow w$

$v.LLP \leftarrow \max \{v.LLP, \ell(v \rightarrow w) + w.LLP\}$

return $s.LLP$