Depth-first Search

```
DFSALL(G):
    clock ← 0
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            clock ← DFS(v, clock)
```

starting time of v

```
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
```

finishing time of v

preorder: sort vertices by start time

postorder: sort vertices by finish time

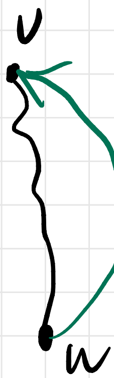
Edge $u \to v$:

$u.pre < v.pre < v.post < u.post$

If DFS(u) calls DFS(v) directly, $u \to v$ is a <u>tree edge</u>

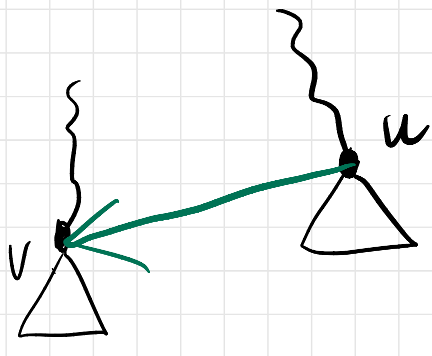
o.w., $u \to v$ is a <u>forward edge</u>

$v.pre < u.pre < u.post < v.post$



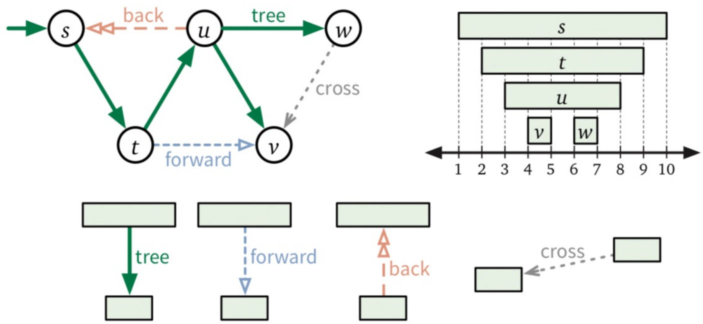
$u \to v$ is a <u>back edge</u>

$v.post < u.pre$



$u \to v$ is a cross edge

$u.post < v.pre$ cannot happen!



back
tree
cross
forward
s
t
u
v
w
1 2 3 4 5 6 7 8 9 10
tree
forward
back
cross

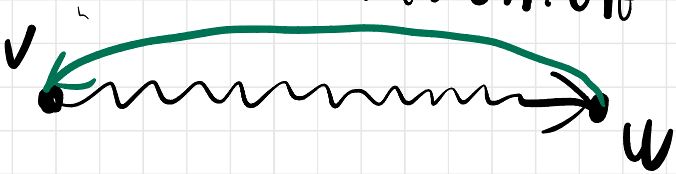
Given directed graph G.

Are there any directed cycles?

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

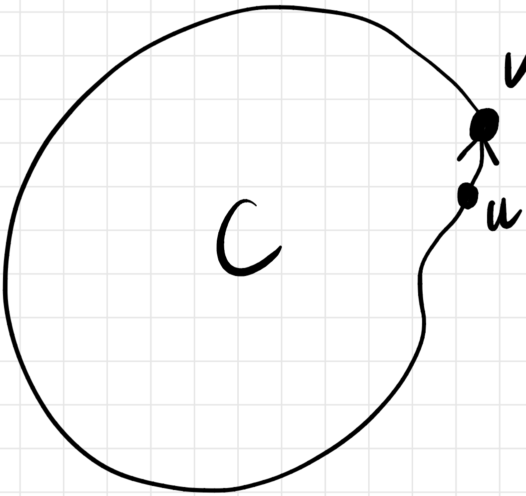
- Suppose there is a back edge $u \to v$. $v.pre < u.pre < u.post < v.post$

so u is reachible from v



← 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.



DFS(v) visits all reachible vertices where nothing on the path is marked (or by time v is finished all reachible vertices are marked) so u gets marked during DFS(v)

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

$u \to v$ is a back edge

Edge $u \to v$ is a back edge iff $u.post < v.post$. So

1) Compute finishing times in $O(V+E)$ time via DFSAll(G).

2) Check if $u.post < v.post$ for any edge $u \to v$. $O(E)$

3) If so $\Rightarrow$ cycle

If not $\Rightarrow$ no cycle

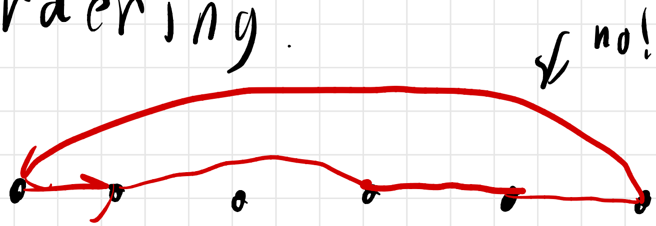
Total: $O(V+E)$

Directed acyclic graphs (DAGs) are those without cycles.

Topological ordering:

Find a total order on vertices such $u < v$ if there exists an edge $u \to v$.

If G has a directed cycle, there is no topological ordering.



But if there are no cycles...
then no back edges...
so u.post > v.post for
every edge u → v
⇒ reverse postorder is
a topological ordering
⇒ every DAG has a
topological ordering

```
TopologicalSort(G):
    for all vertices v
        v.status ← New
    clock ← V
    for all vertices v
        if v.status = New
            clock ← TopSortDFS(v, clock)
    return S[1..V]
```

```
TopSortDFS(v, clock):
    v.status ← Active
    for each edge v→w
        if w.status = New
            clock ← TopSortDFS(w, clock)
        else if w.status = Active
            fail gracefully
    v.status ← Finished
    S[clock] ← v
    clock ← clock − 1
    return clock
```

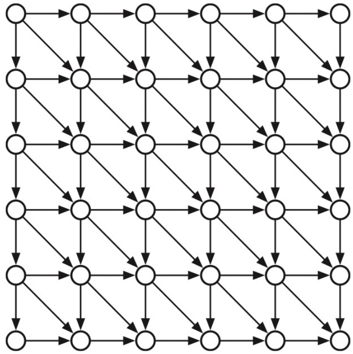
$O(V+E)$ time

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 $x \to y$ means you make a call to y while handling the call to x



(Edit(i, j)'s dependency graph)

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.

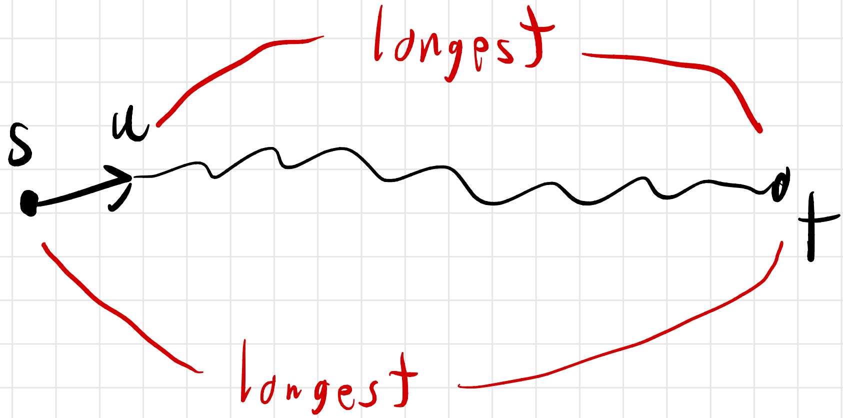
Longest Path: Given directed graph $G=(V,E)$ with weights on edges $\ell: E \to \mathbb{R}$.

Also given s & t.

What is the max length of a simple s,t-path.

(no repeated vertices)

Will assume G is a DAG.



LLP(v): longest path length from v to t, ($-\infty$ if none exists)

$$LLP(v) = \begin{cases} 0 & v = t \\ \max\{\ell(v \to w) + LLP(w) \mid v \to w \in E\} \end{cases}$$

($\max = -\infty$ if no edges $v \to w$)

Want to know LLP(s).

Dependency graph is G itself.

So solve subproblems in postorder for G.

```
LONGESTPATH(s, t):
  for each node v in postorder
    if v = t
      v.LLP ← 0
    else
      v.LLP ← −∞
      for each edge v→w
        v.LLP ← max{v.LLP, ℓ(v→w) + w.LLP}
  return s.LLP
```

$O(V+E)$ time

$O(1)$ per edge across all loops