Depth-first search (DFS)

```
DFS(v):
    mark v
    PreVisit(v)
    for each edge vw
        if w is unmarked
            parent(w) ← v
            DFS(w)
    PostVisit(v)
```

```
DFSAll(G):
    Preprocess(G)
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            DFS(v)
```

$O(V+E)$

```
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
```

- add a clock to learn about visitation order

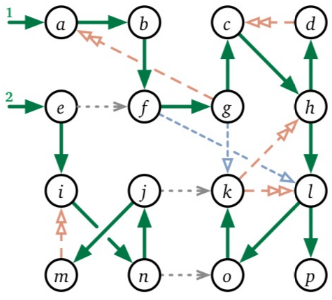
v.pre: starting time of v

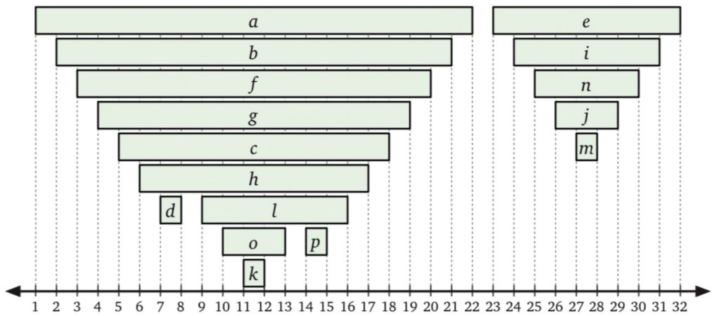
v.post: finishing time of v

[v.pre, v.post]

active interval

all pairs are nested or disjoint





If DFS(v) called while u is active => there is a u,v-path.

Sort by x.pre for a <u>preordering</u>

Sort by x.post for a <u>postordering</u>.

We're midway through running DFS all. We have a current clock value.

Will compare to final pre & post values.

vertex v is:

new if $clock < v.pre$

active if $v.pre \le clock < v.post$

finished if $v.post \le clock$

active vertices form a directed path in G

Partioning edges:

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

(clock = u.pre)

If v is new,

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

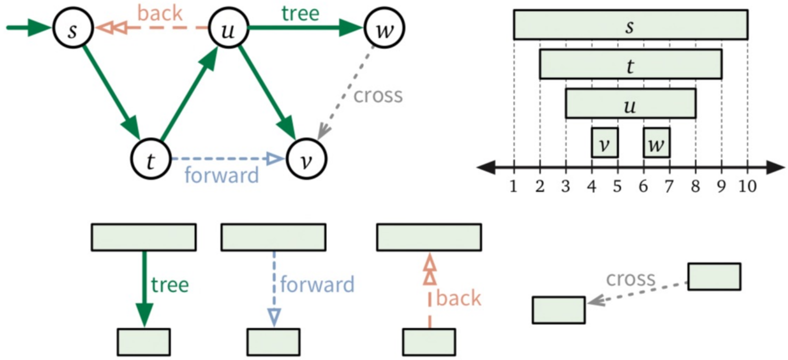
If DFS(u) directly calls DFS(v), $u \to v$ if a <u>tree edge</u>

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

If v is active, v is on stack

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

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



If v is finished,

$$v.post < u.pre$$

$u \to v$ is a cross edge

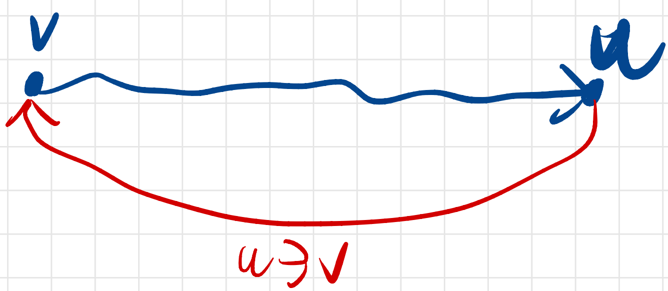
The classification depends on how the DFS runs!!

Detecting Cycles in Directed Graphs

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

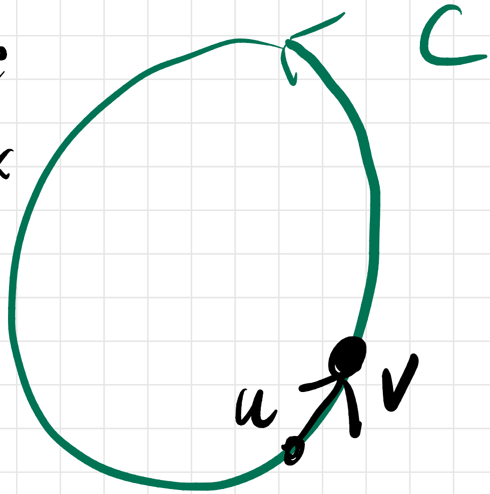
Proof: Suppose $u \to v$ is a back edge. v was active when we called DFS(u).

$\Rightarrow$ There is a v,u-path



Suppose there is a cycle C.

Let v be first vertex of C marked by DFSAll.



u: predecessor of v on C

DFS(v) eventually calls DFS(u)

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

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

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

$O(V+E)$ time!

Topological Sort

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

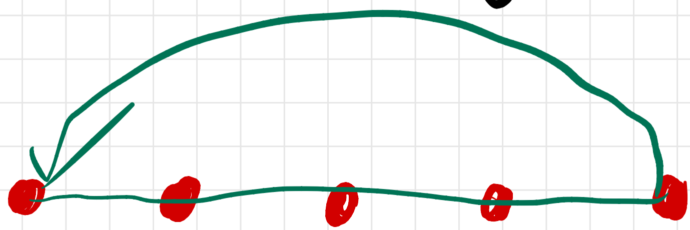
topological ordering of G:

a total ordering of vertices where $u < v$ if $u \to v \in E$

-or-

can write all vertex names from left to right so all edges go left to right.

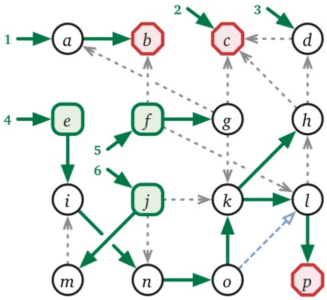
directed cycle $\Rightarrow$ no topological ordering

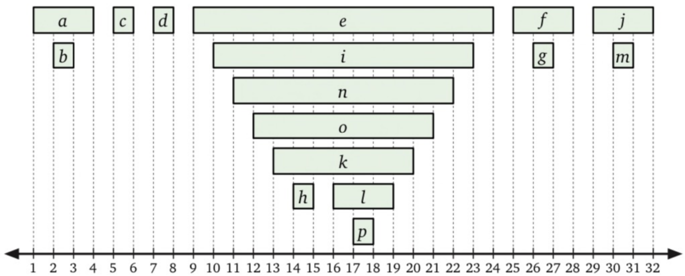


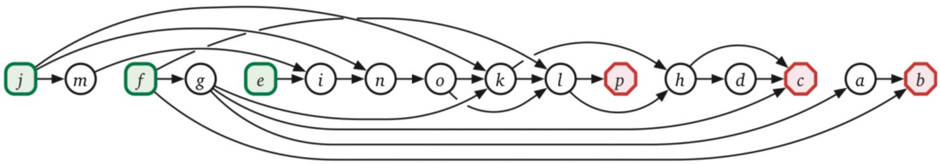
o.w., ... no back edges, ...

$\Rightarrow$ u.post > v.post for all $u \to v \in E$

$\Rightarrow$ order by <u>decreasing</u> 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 ← 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
```

Dynamic Programming

Given a recurrence, the <u>dependency graph</u> has one vertex per subproblem & an edge $x \to y$ for every direct call of a subproblem y from a subproblem x.

Must be acyclic!

If you use basic memoization, you solve problems in <u>post order</u>.

Iterative dynamic prog. algs, solve the problems in some reverse topological order.

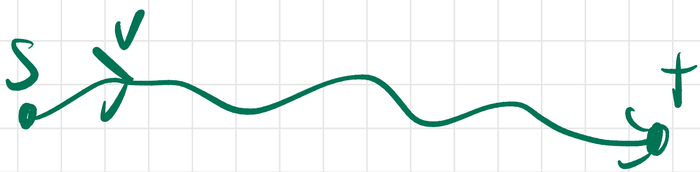
Longest Path:

Given 1) $G=(V,E)$ with

2) edge weights $\ell : E \to \mathbb{R}$,

3) two vertices s + t.

Assume G is a DAG...



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

Want $LLP(s)$.

rest of path

$$LLP(v) = \begin{cases} 0 & \text{if } v = t, \\ \max\left\{\ell(v\to w) + LLP(w) \;\middle|\; v\to w \in E\right\} & \text{otherwise,} \end{cases}$$

$-\infty$ is no $(v \to w)$'s

first edge?

- use min instead for shortest paths in a DAG!

recurrence is ill-defined if G has a cycle

The dependency graph is G itself + 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 ← 0
        else
            v.LLP ← −∞
            for each edge v→w
                v.LLP ← max{v.LLP, ℓ(v→w) + w.LLP}
    return s.LLP
```