HW 8: Today Oct 27

GPS 9: Mon Nov 2

Midterm 2: Mon Nov 9

```
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
```

← start time of v

← finish time of v

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

DFS(u)



if not marked yet, we will find v during DFS(u) & call DFS(v)

order by v.post, you get a postorder

Problem: Given directed $G = (V, E)$.
Does G contain a cycle?

If DFS(v) indirectly calls DFS(u) &
there is an edge $u \to v$, we call $u \to v$
a <u>back edge</u>.

$\Rightarrow G$ has a cycle.



Lemma: If DFSAll(G) finds no back edge, then G is acyclic.

Proof: Suppose G has a cycle C.

Let v be the first vertex that gets marked in C.

Let u immediately precede v on cycle C. So there is an edge u → v.



There is an unmarked path from v to u, so DFS(v) indirectly

calls DFS(u).

$u \Rightarrow v$ is a back edge.

So, G has a cycle iff DFSALL finds a back edge!

iff there is a $u \Rightarrow v$ s.t.

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

iff $u.post < v.post$

Acyclic(G)

DFSALL(G) (with clocks)

for all edge $u \Rightarrow v$

if $u.post < v.post$

$O(|V| + |E|)$

return False

time

return True

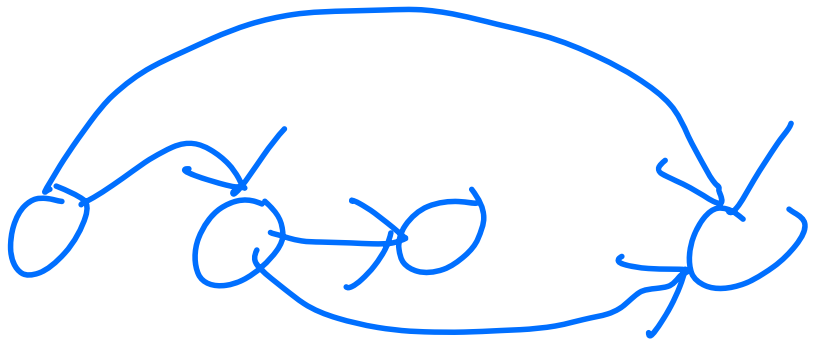
—

Topological ordering of a graph

$G = (V, E)$ is a total/linear order over the vertices

s.t. for all edges $u \to v$

$u < v$ in the order.



Does not exist if there is a cycle.

But no cycle iff no back edge

iff for all edges $u \to v$,

$v.post < u.post$

so... reversing a postorder gives a topological order.

So can compute a topological order in $O(|V| + |E|)$ time.

New problem:

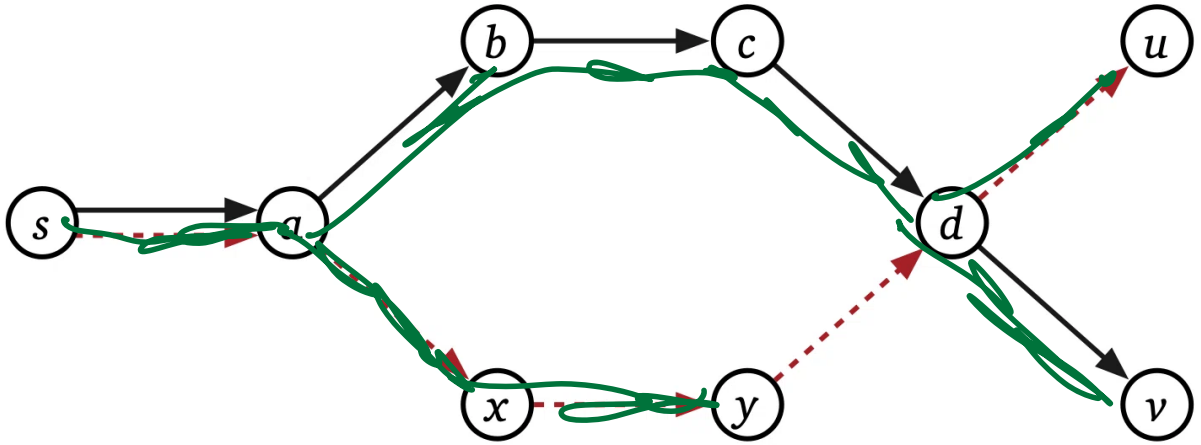
Given a directed graph $G=(V,E)$ with weights $w: E \Rightarrow \mathbb{R}$ on the edges.

Distance from s to t: minimum total weight over all (s,t)-paths.

Single source shortest paths (SSSP)

distances & shortest paths from some $s \in V$ to all other vertices in G.

Today: Assume positive weights.



If two subpaths of shortest paths disagree, can just pick one

⇒ we can look for shortest paths that follow a tree rooted at s

The shortest paths tree rooted at s.

Ford '56:

Maintain two values for each vertex v.

v.dist: ∞ or length of <u>same</u> path from s to v

v.pred: the predecessor vertex on the path of length v.dist (or Null)

InitSSSP(s):

s.dist ← 0

s.pred ← Null

for all vertices $v \neq s$

An edge $u \to v$ is <u>tense</u> if

$u.dist + w(u \to v) <$ v.dist

v.dist ← ∞
v.pred ← Null

Relax$(u \to v)$:

$v.dist \leftarrow u.dist + w(u \to v)$

$v.pred \leftarrow u$

FordSSSP(s):

InitSSSP(s)

while there is at least one tense edge $u \to v$

Relax$(u \to v)$

Lemma: (Assuming positive edge weights) FordSSSP will terminate & for all v, v.dist will be distance from s to v & v.pred will be v's predecessor on some shortest (s,v)-path

Dijkstra's algorithm.

- If $u \to v$ is <u>not</u> tense right now, it can only become tense by decreasing $u.dist$
- Let $u \to v$ be a tense edge minimizing $u.dist$. We should relax $u \to v$ first.

```
Dijkstra(s):
    InitSSSP(s)
    Insert(s, 0)
    while the priority queue is not empty
        u ← ExtractMin()
        for all edges u → v
```

Assumes a priority queue.

if $u \to v$ is tense
    $\text{Relax}(u \to v)$
    if v is in the priority queue
        $\text{DecreaseKey}(v, v.dist)$
    else
        $\text{Insert}(v, v.dist)$

← change priority of v to v.dist

↑ insert v into p.queue with priority v.dist

It's FordSSSP, so correct.

d_i: u.dist if u is ith vertex ExtractMined

$\Rightarrow d_{i+1} \geq d_i$

v.dist can only decrease

$\Rightarrow$ we cannot remove v twice

$\geq |V|$ Insert & ExtractMin

$\leq |E|$ Relaxations & DecreaseKeys

$O((|V| + |E|) \log |E|) = \underline{O(|E| \log |V|)}$

if graph connected & simple