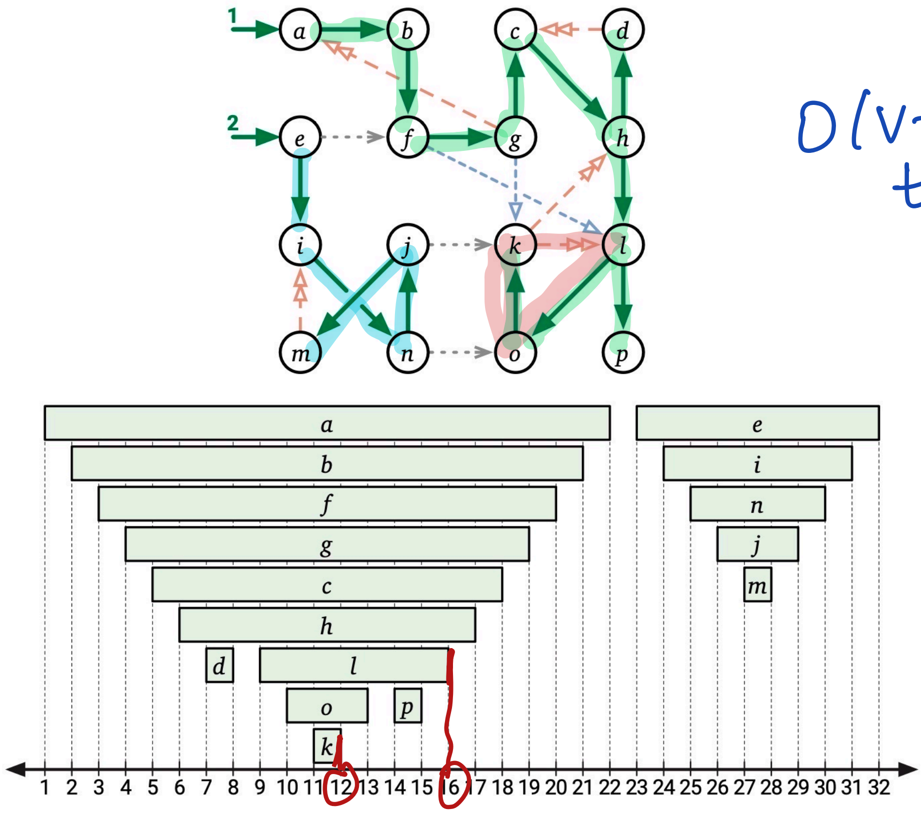
HW 8.2 is due Wed Oct 25 at 9pm
HW 9 out — last HW before Midterm 2 (two weeks)

Depth-First search in directed graphs

```
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
```



O(V+E) time

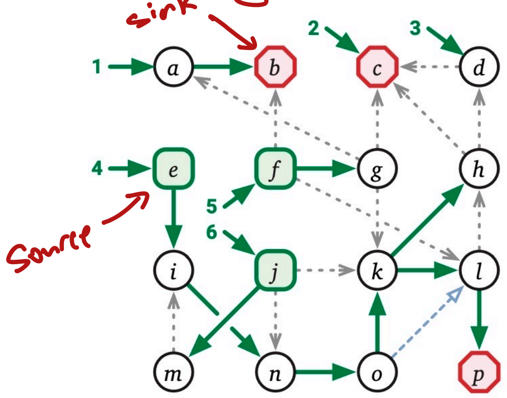
Pre: a b f g c h d l o k p e i n j m

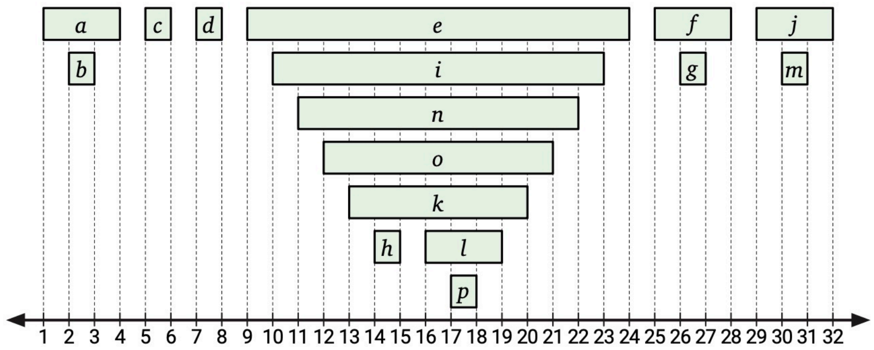
Post: d k o p l h c g f b a m j n i e

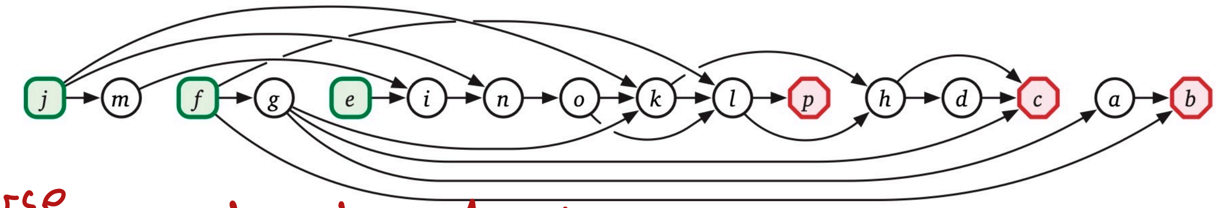
Detect dir. cycles in O(V+E) time

Lemma: G has no cycle iff For every edge v→w we have v.post > w.post

dag = directed acyclic graph





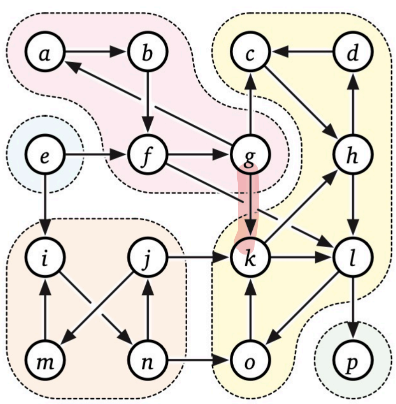


reverse postorder = topological order

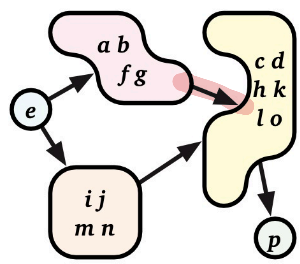
TopologicalSort(G):
    $clock \leftarrow V$
    for all vertices v in postorder
        $S[clock] \leftarrow v$
        $clock \leftarrow clock - 1$
    return $S[1..V]$

$O(V+E)$

$\boxed{O(V+E) \text{ time}}$



strong components



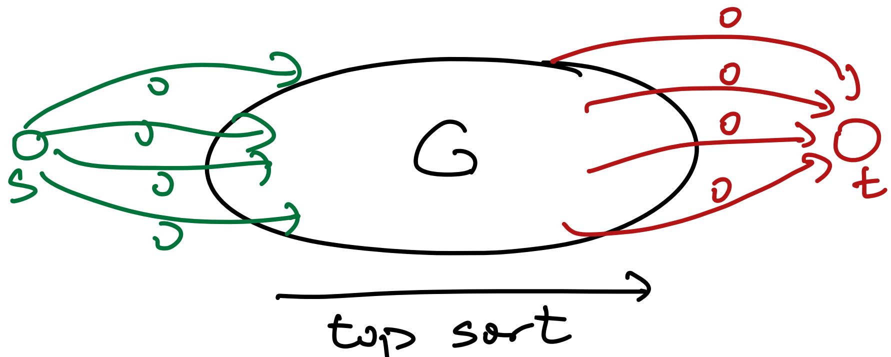
metagraph $\langle G \rangle$
directed acyclic graph

Longest Path:

Given a dag $G=(V,E)$

$w: E \to \mathbb{R}$ edge ~~weights~~ lengths

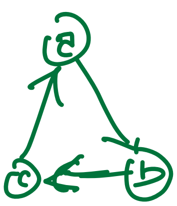
Find a path in G with max total ~~weight~~ length



We want LLP(s)

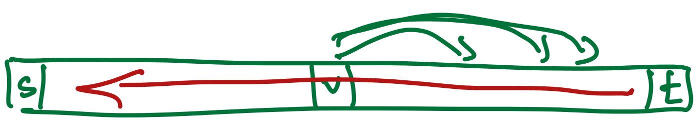
LLP(v) = length of longest path in G from v to t.

$$LLP(v) = \begin{cases} 0 & \text{if } v = t \\ \max\limits_{v \to w} \left(w(v \to w) + LLP(w) \right) & \text{else} \end{cases}$$




```
LongestPath(v, t):
  if v = t
      return 0
  if v.LLP is undefined
      v.LLP ← −∞
      for each edge v→w
          v.LLP ← max{v.LLP, ℓ(v→w) + LongestPath(w, t)}
      return v.LLP
```

verts in top order



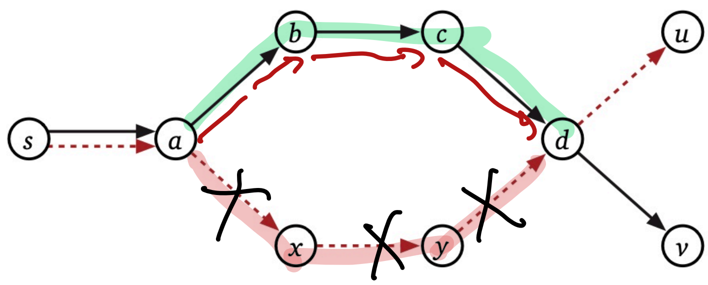

```
LongestPath(s, t):
  for each node v in postorder
      if v = t
          v.LLP ← 0
      else
          v.LLP ← −∞
          for each edge v→w
              v.LLP ← max{v.LLP, ℓ(v→w) + w.LLP}
  return s.LLP
```

$\underline{O(V+E)}$

Shortest paths

Single-source shortest paths:
Given $G=(V,E)$, $w: E \to \mathbb{R}^+$, $s \in V$
Find shortest path from s to every other vertex of G

These paths define a tree rooted at s.



if green path $a \to b \to c \to d$ is shorter than red path $a \to x \to y \to d$
then shortest path from s to u is incorrect

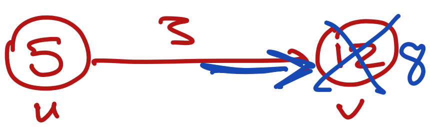
$v.dist$ = over-estimate of shortest path distance from s to v

$v.pred$ = predecessor of v on est. shortest path

INITSSSP(s):
  $s.dist \leftarrow 0$
  $s.pred \leftarrow$ NULL
  for all vertices $v \neq s$
    $v.dist \leftarrow \infty$
    $v.pred \leftarrow$ NULL

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

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



FORDSSSP(s):
  INITSSSP(s)
  while there is at least one tense edge
    RELAX any tense edge

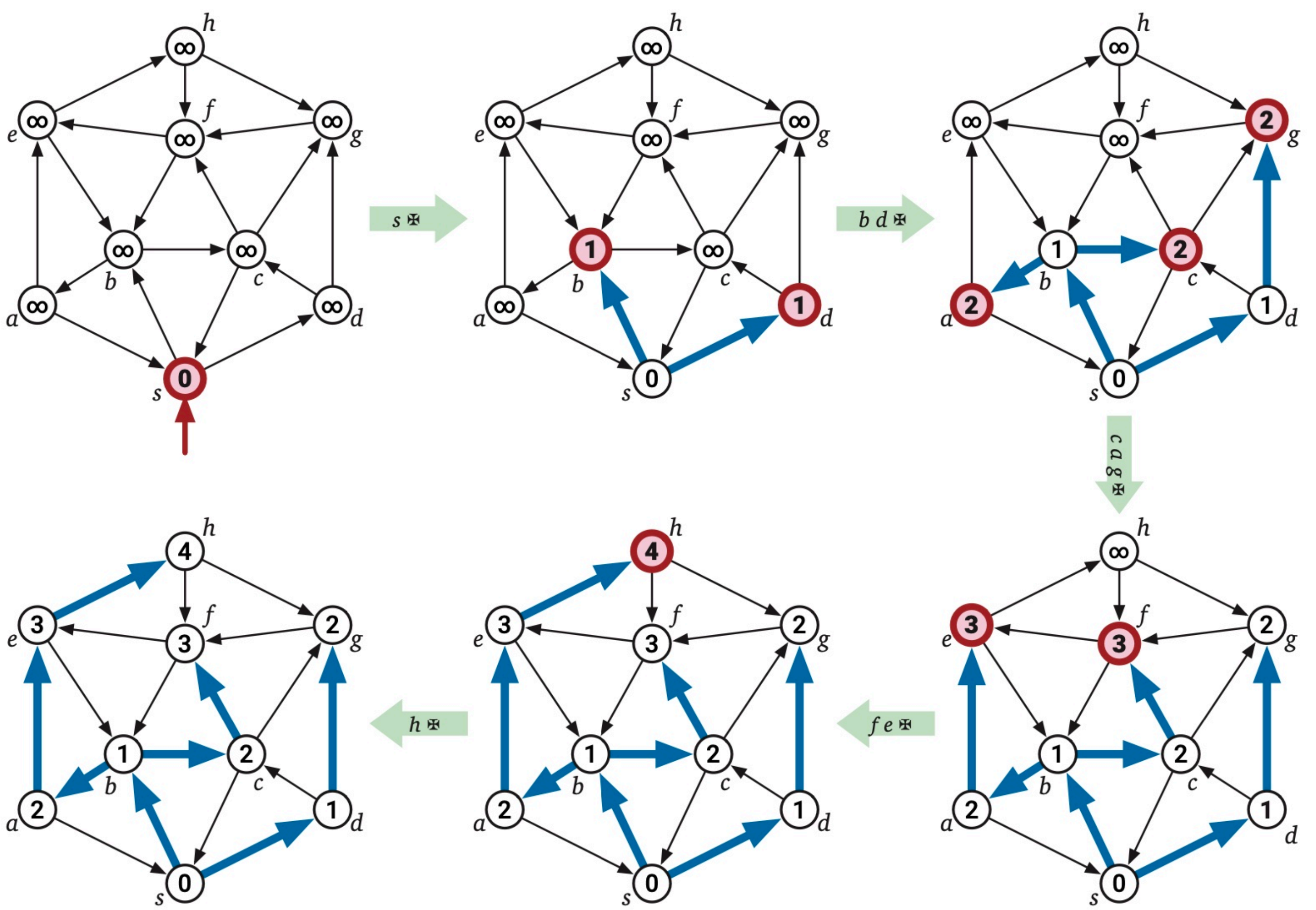
Lester Ford 1953

Breath-First search

unweighted edges
w(e) = 1

```
BFS(s):
  InitSSSP(s)
  Push(s)
  while the queue is not empty
      u ← Pull()
      for all edges u→v
          if v.dist > v.dist + 1            ⟨⟨if u→v is tense⟩⟩
              v.dist ← u.dist + 1
              v.pred ← u                    ⟨⟨relax u→v⟩⟩
              Push(v)
```

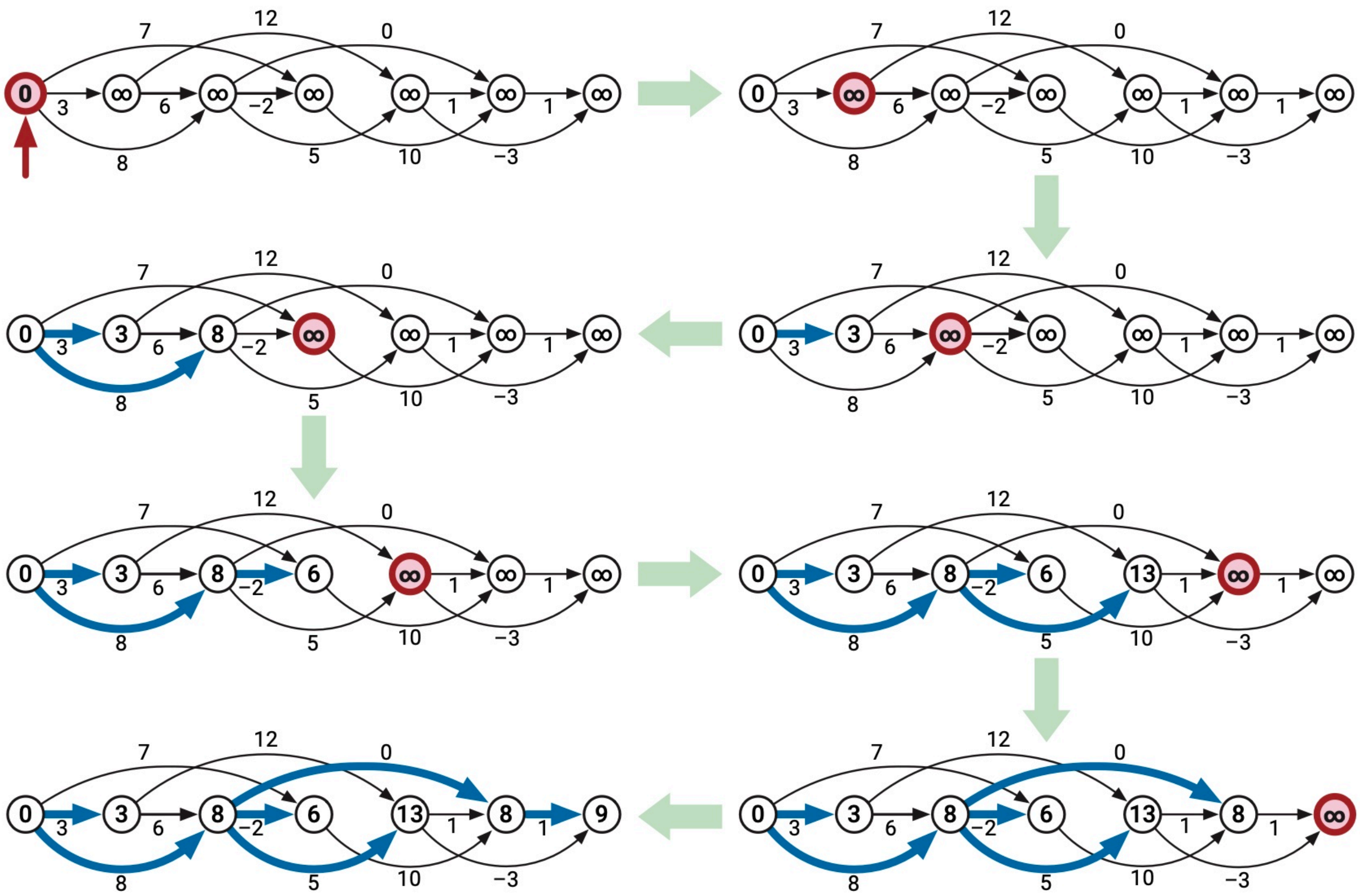
O(V+E) time



$$dist(v) = \begin{cases} 0 & \text{if } v = s \\ \min\limits_{u \to v} \left(dist(u) + w(u \to v) \right) & \text{otherwise} \end{cases}$$

```
DagSSSP(s):
  for all vertices v in topological order
      if v = s
          v.dist ← 0
      else
          v.dist ← ∞
          for all edges u→v
              if v.dist > u.dist + w(u→v)          ⟨⟨if u→v is tense⟩⟩
                  v.dist ← u.dist + w(u→v)          ⟨⟨relax u→v⟩⟩
```

```
DagSSSP(s):
  InitSSSP(s)
  for all vertices v in topological order
      for all edges u→v
          if u→v is tense
              Relax(u→v)
```



```
DIJKSTRA(s):
  INITSSSP(s)
  INSERT(s, 0)
  while the priority queue is not empty
      u ← EXTRACTMIN()
      for all edges u→v
          if u→v is tense
              RELAX(u→v)
              if v is in the priority queue
                  DECREASEKEY(v, v.dist)
              else
                  INSERT(v, v.dist)
```

Best-first search

priority queue

priority(v) = v.dist

positive edge weights:

$O(E \log V)$ time

in general

$2^{O(V)}$ worst case

fast in practice