------------------------------ 24 ----------------------------

Single Source Shortest Paths

Single-source shortest paths problem

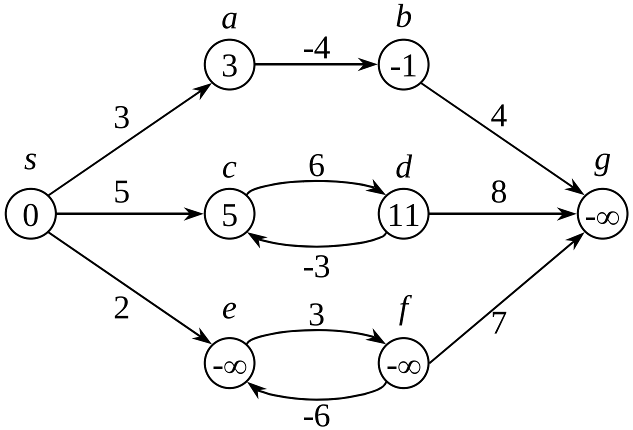
Input: A weighted directed graph $G=(V, E)$ and a source vertex s.

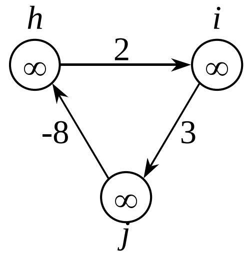
Output: for every $v \in V$, find a shortest path from s to v.

Negative-weight edges:

$\delta(u, v)$: the shortest-path weight from u to v.

If G contains no negative-weight cycles reachable from s, then $\delta(s, v)$ is well-defined for all $v \in V$. Otherwise, if there is a negative-weight cycle on some path from s to v, we define $\delta(s, v) = -\infty$.





Variants (Applications):

Single-destination shortest paths problem
(Transform to single-source by reversing edges.)
Single-pair shortest paths problem
(No faster algorithms exist.)
All-pairs shortest paths problem
(Solve single-source problem V times.
Faster algorithms exist: Chapter 25.)

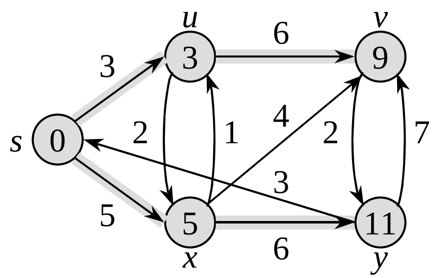
Representing shortest paths:
for every $v \in V$, maintain a predecessor $\pi(v)$

***Predecessor subgraph*:** $G_\pi = (V_\pi, E_\pi)$, where

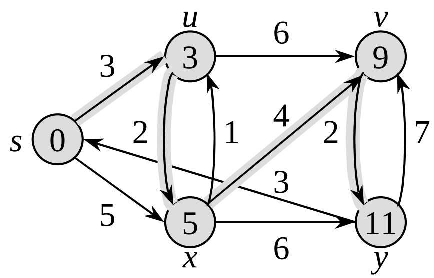
$V_\pi=\{v \mid v \in V \text{ and } \pi(v) \neq \text{NIL}\} \cup s$

$E_\pi=\{(\pi(v), v) \mid v \in V_\pi - \{s\}\}$

G_π is a *shortest-paths tree* rooted at s.
(not unique)



(b)



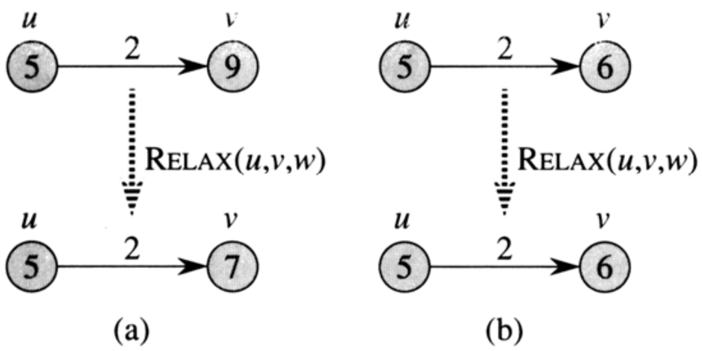
(c)

Relaxation:

Repeatedly decrease $d[v]$ until $d[v]=\delta(s, v)$.
Two procedures:

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ V[G]
      do d[v] ← ∞
         π[v] ← NIL
d[s] ← 0
```

```
RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
   then d[v] ← d[u] + w(u, v)
        π[v] ← u
```



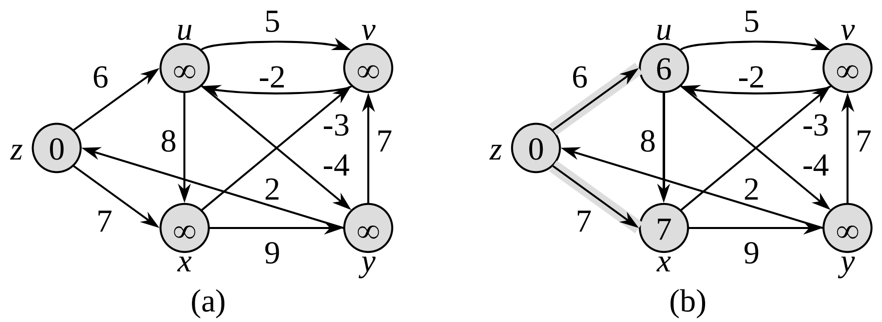
24.1 The Bellman-Ford algorithm

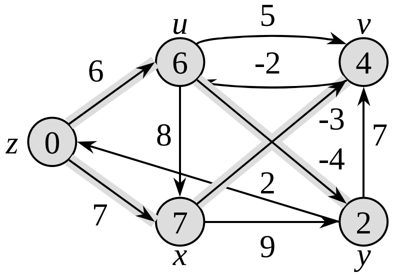
* Weights can be negative. If there is a negative cycle reachable from s, it returns FALSE.

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| − 1
     do for each edge (u, v) ∈ E[G]
            do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
     do if d[v] > d[u] + w(u, v)
           then return FALSE
return TRUE
```

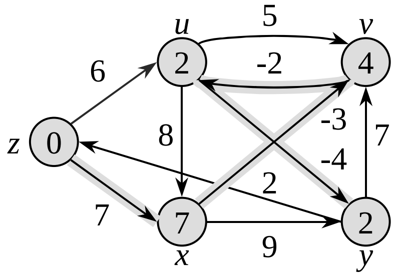
* $O(VE)$ time

Example: (Each phase relaxes edges in lexicographic order: (u,v), (u,x), (u,y), (v,u), (x,v), (x,y), (y,v), (y,z), (z,u), (z,x). Shaded edges indicate π.)

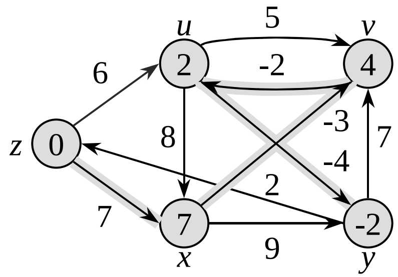




(c)



(d)



(e)

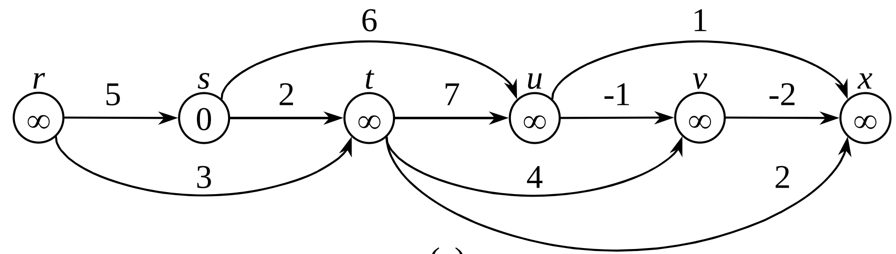
Lemma 24.2: If *G* contains no negative-weight cycles reachable from *s*. Then, Bellman-Ford algorithm computes $d[v]=\delta(s, v)$ for all vertices *v* that are reachable from *s*.

Proof: By induction, we can easily show that if the shortest path from *s* to *v* contains *i* edges, $d[v]$ will be correctly computed after phase *i*. Since a simple path contains at most *V*-1 edges, The lemma holds.

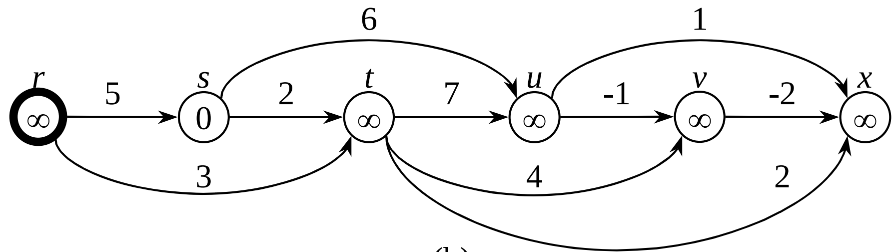
Theorem 24.4: Bellman-Ford algorithm is correct. (See textbook for its proof.)

24.2 Single-source shortest path in DAGs

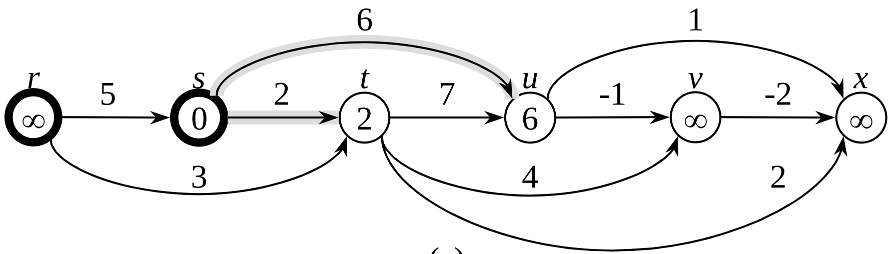
```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    do for each vertex v ∈ Adj[u]
        do RELAX(u, v, w)
```



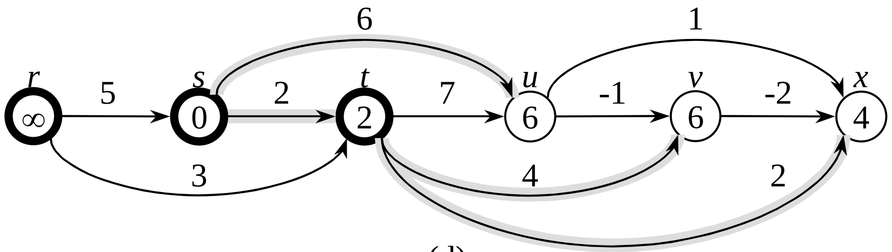
(a)



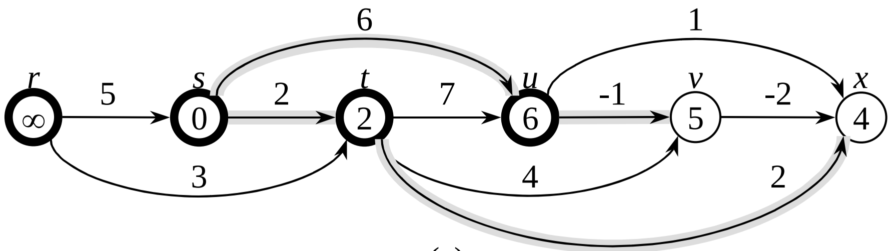
(b)



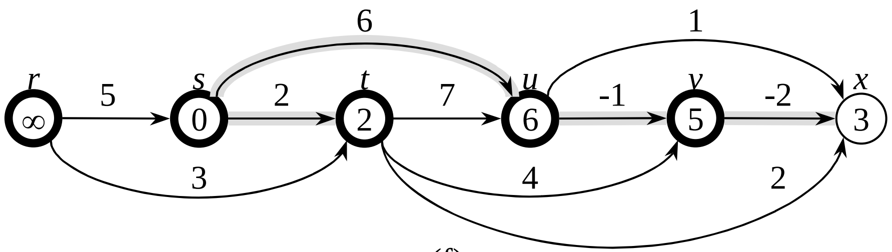
(c)



(d)



(e)



(f)

* Correctness: By Induction

***The critical path problem*:** Finding a longest path through a DAG.

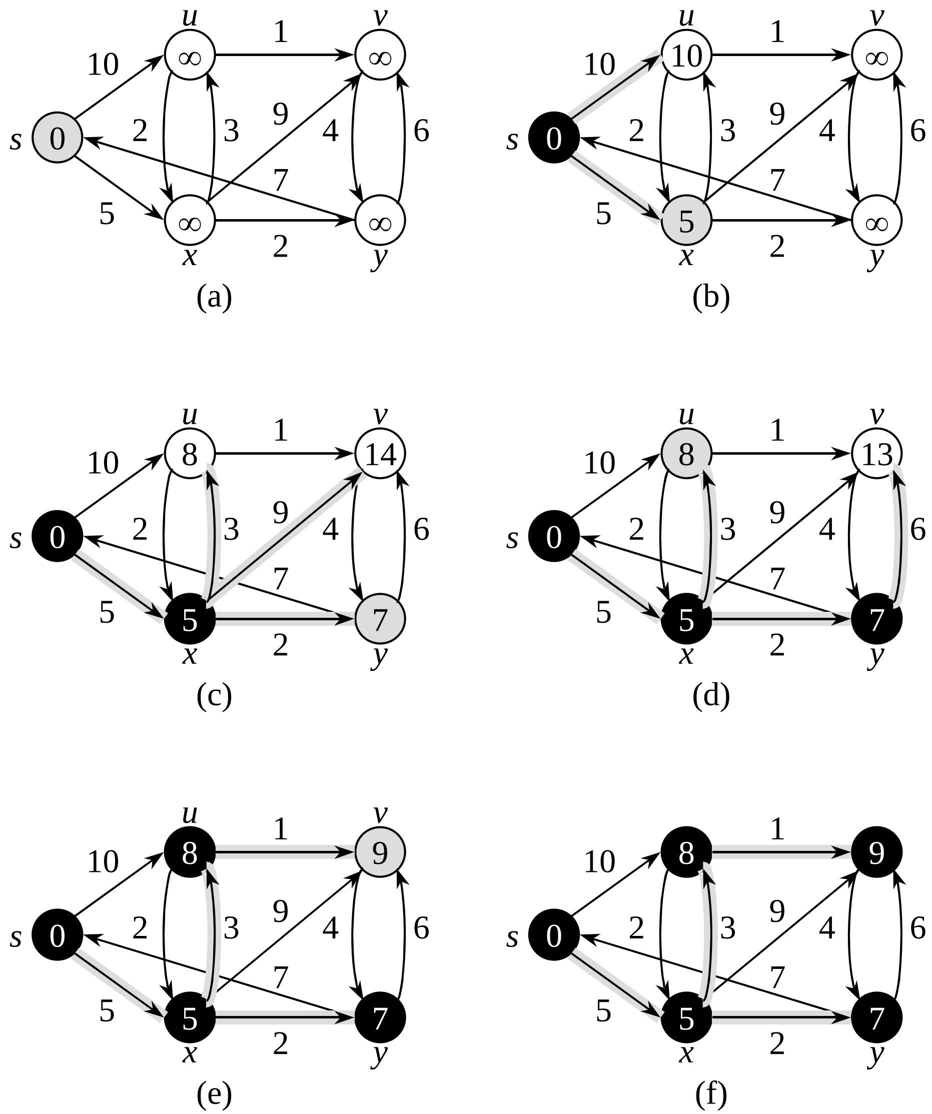
Since a DAG contains no cycles, this problem can be solved by negating the weights and then running DAG-SHORTEST-PATHS.

24.3 *Dijkstra's algorithm*:

(All weights should be nonnegative.)

It maintains a set *S* of vertices *v* whose $\delta(s, v)$ is computed. And it repeatedly selects a vertex $u \in V\text{-}S$ with minimum $d[u]$, inserts it into *S*, and relaxes all edges leaving *u*.

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S ← Ø
Q ← V[G]
while Q ≠ Ø
    do u ← EXTRACT-MIN(Q)
        S ← S ∪ {u}
        for each vertex v ∈ Adj[u]
            do RELAX(u, v, w)
```



Time Complexity:

* Similar to Prim's MST algorithm.

(a) Implement priority queue *Q* as an array

Steps 1~3: $O(V)$ (Build Q)
Step 5: $O(V^2)$ (V times Extract-Min)
Step 6: $O(V)$
Steps 7~8: $O(E)$ ($2E$ times Decrease-key)
Total: $O(V^2+E) = O(V^2)$ (for dense *G*)

(b) Implement *Q* as a binary heap

Step 5: $O(V \lg V)$ (V times Extract-Min)
Steps 7~8: $O(E \lg V)$ ($2E$ times Decrease-key)
Total: $O(E \lg V)$ (for sparse *G*)

(c) Implement *Q* as a Fibonacci heap

Step 5: $O(V \lg V)$ (V times Extract-Min)
Steps 7~8: $O(E)$ ($2E$ times Decrease-key)
Total: $O(E+V \lg V)$ (for sparse *G*)

Homework: Ex. 24.1-4, 24.2-1, 24.3-2, Pro. 24-2, 24-3, 24-5 (mean-weight cycle).