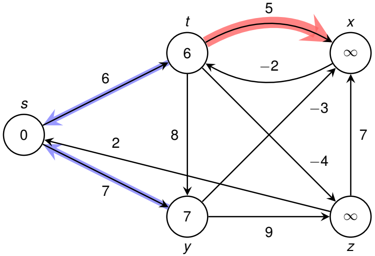


6.4: Single-Source Shortest Paths

Frank Stajano Thomas Sauerwald

Lent 2016

UNIVERSITY OF CAMBRIDGE

Outline

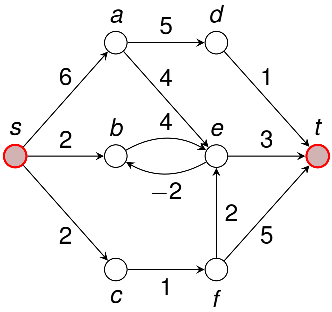
Introduction

Bellman-Ford Algorithm

Shortest Path Problem

Shortest Path Problem

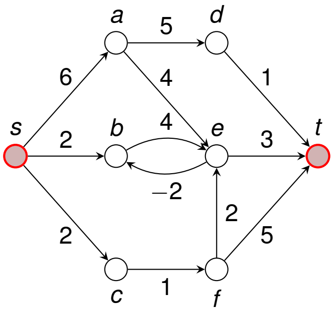
- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$



Shortest Path Problem

Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G

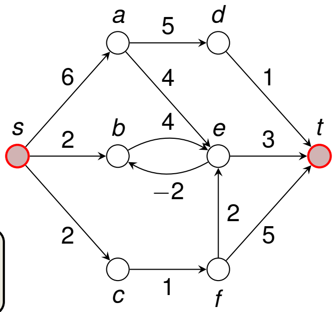


Shortest Path Problem

Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G

$p = (v_0 = s, v_1, \ldots, v_k = t)$ such that $w(p) = \sum_{i=1}^{k} w(v_{k-1}, v_k)$ is minimized.

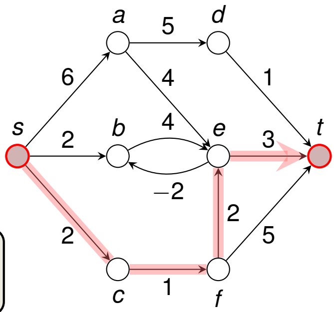


Shortest Path Problem

Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G

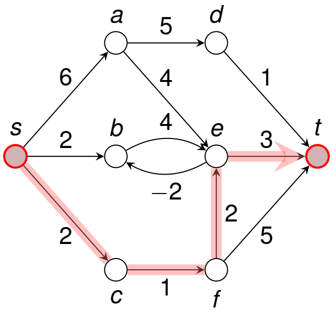
$p = (v_0 = s, v_1, \ldots, v_k = t)$ such that $w(p) = \sum_{i=1}^{k} w(v_{k-1}, v_k)$ is minimized.



Shortest Path Problem

Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G

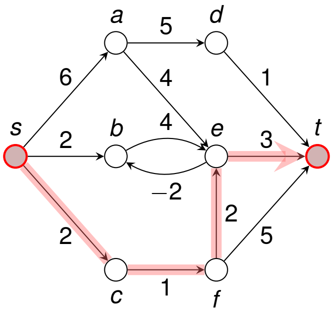


What if G is **unweighted**?

Shortest Path Problem

Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G



What if G is unweighted?

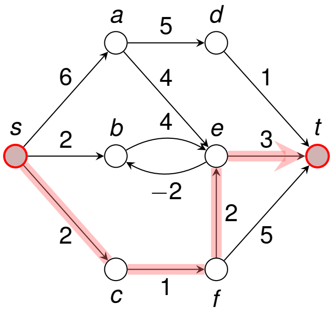
Two possible answers are:

1. Run BFS (computes shortest paths in unweighted graphs)
2. Set the weight of all edges to 1

Shortest Path Problem

Shortest Path Problem

- Given: directed graph $G = (V, E)$ with edge weights, pair of vertices $s, t \in V$
- Goal: Find a path of minimum weight from s to t in G



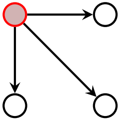
Applications

- Car Navigation, Internet Routing, Arbitrage in Concurrency Exchange

Variants of Shortest Path Problems

Single-source shortest-paths problem (SSSP)

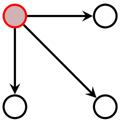
- Bellman-Ford Algorithm
- Dijsktra Algorithm



Variants of Shortest Path Problems

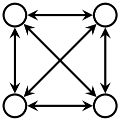
Single-source shortest-paths problem (SSSP)

- Bellman-Ford Algorithm
- Dijsktra Algorithm

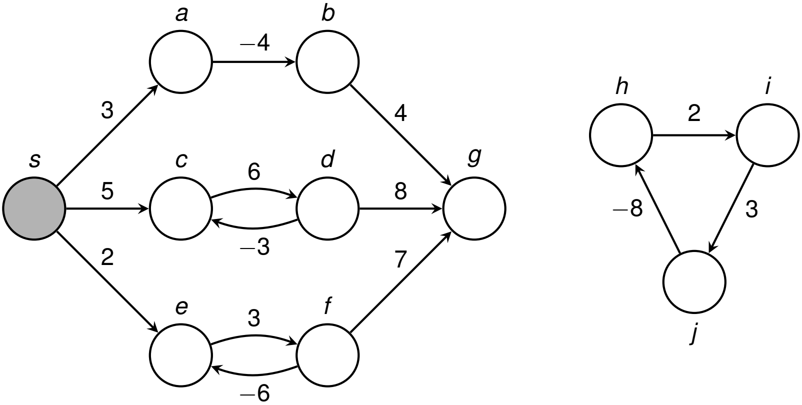


All-pairs shortest-paths problem (APSP)

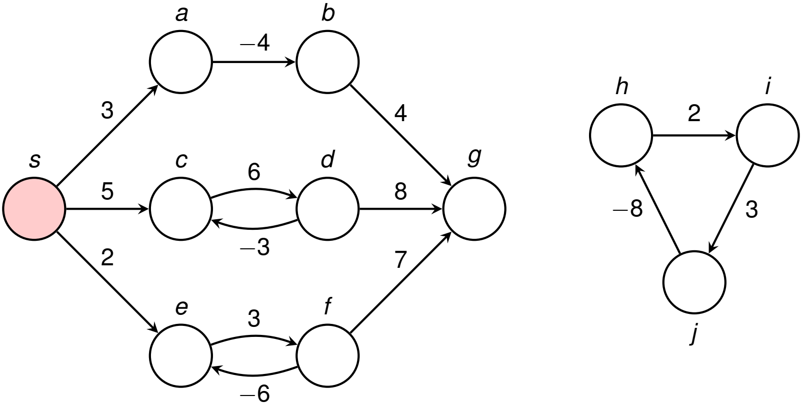
- Shortest Paths via Matrix Multiplication
- Johnson's Algorithm



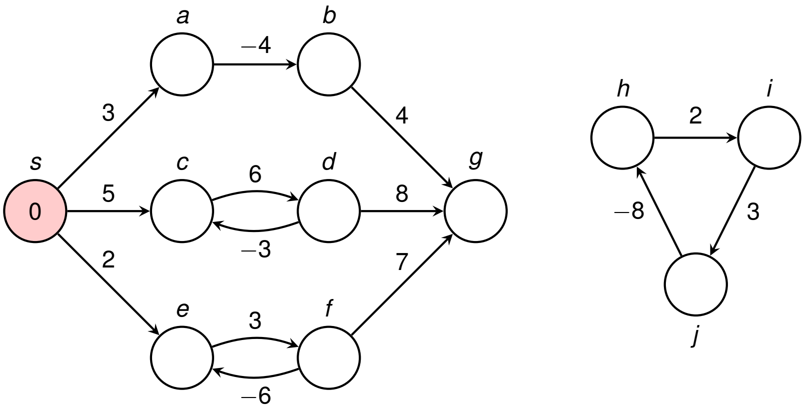
Distances and Negative-Weight Cycles (Figure 24.1)



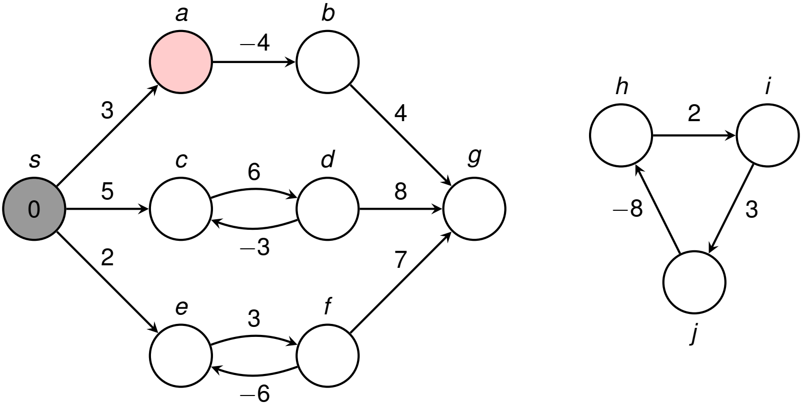
Distances and Negative-Weight Cycles (Figure 24.1)



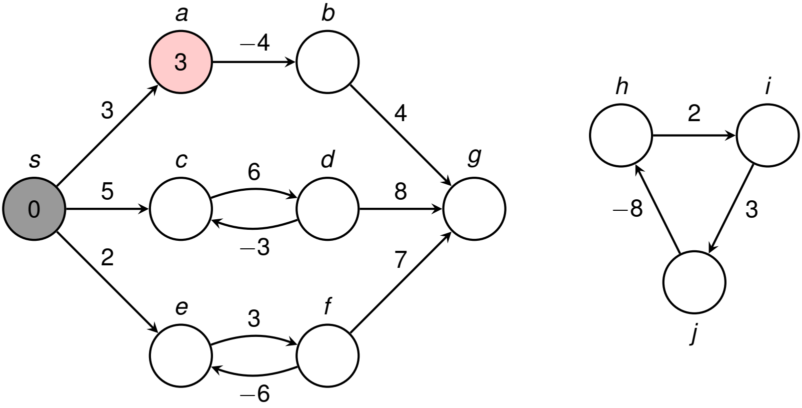
Distances and Negative-Weight Cycles (Figure 24.1)



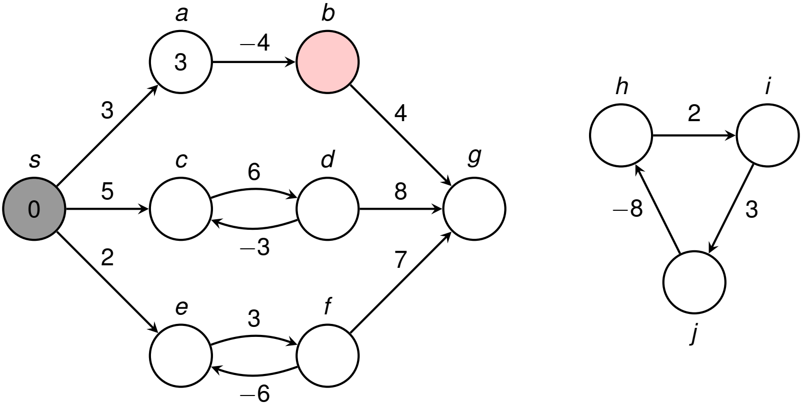
Distances and Negative-Weight Cycles (Figure 24.1)



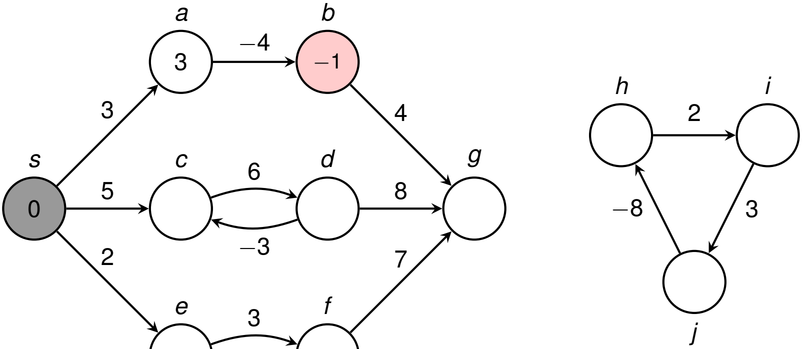
Distances and Negative-Weight Cycles (Figure 24.1)



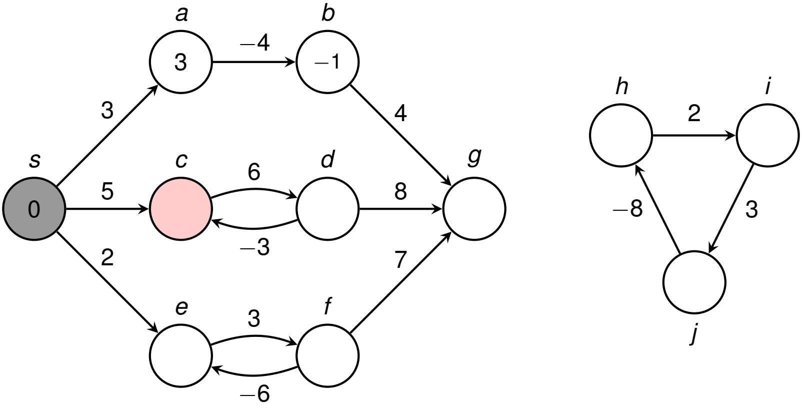
Distances and Negative-Weight Cycles (Figure 24.1)



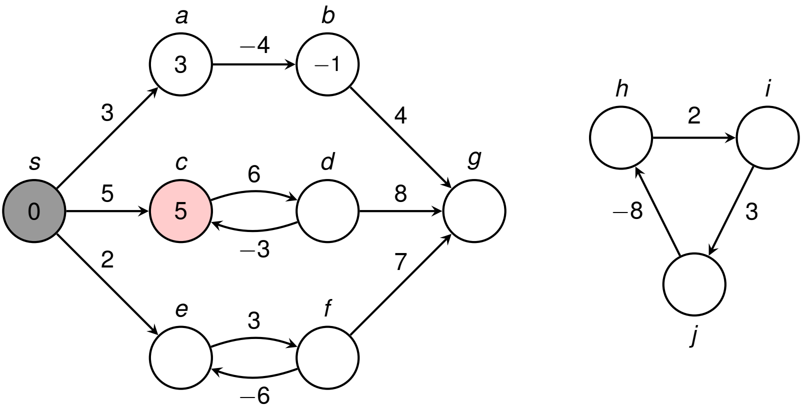
Distances and Negative-Weight Cycles (Figure 24.1)



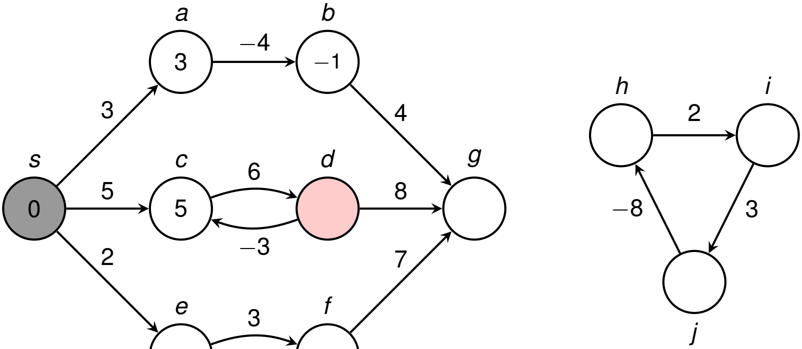
Distances and Negative-Weight Cycles (Figure 24.1)



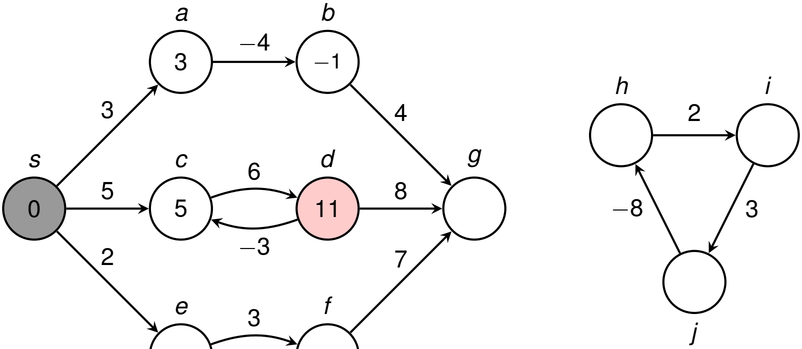
Distances and Negative-Weight Cycles (Figure 24.1)



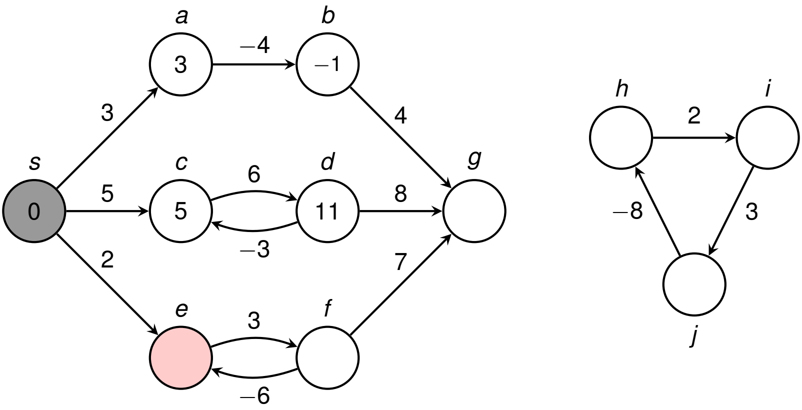
Distances and Negative-Weight Cycles (Figure 24.1)



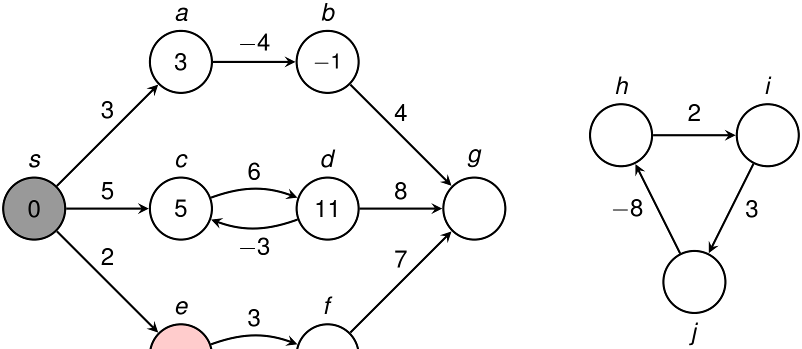
Distances and Negative-Weight Cycles (Figure 24.1)



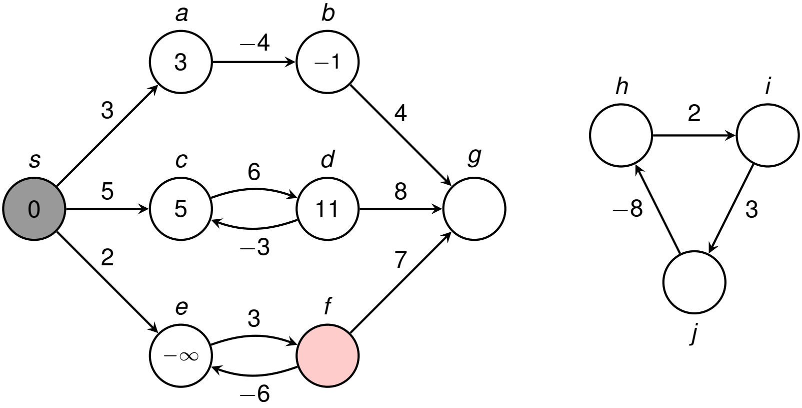
Distances and Negative-Weight Cycles (Figure 24.1)



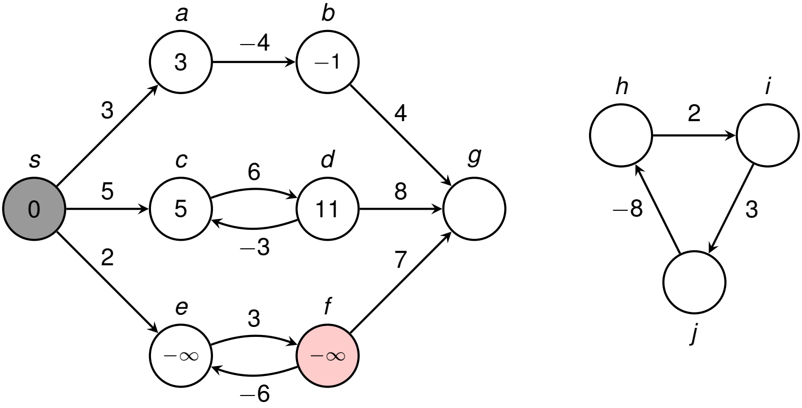
Distances and Negative-Weight Cycles (Figure 24.1)



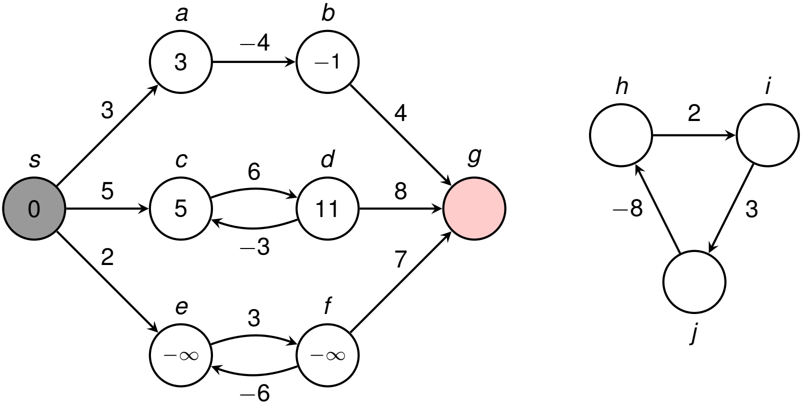
Distances and Negative-Weight Cycles (Figure 24.1)



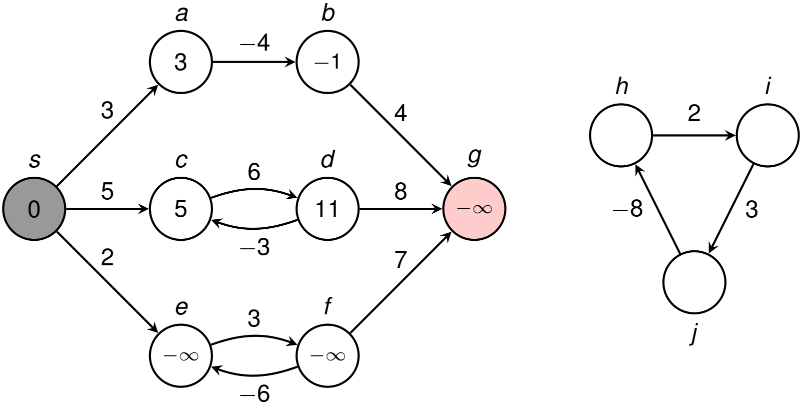
Distances and Negative-Weight Cycles (Figure 24.1)



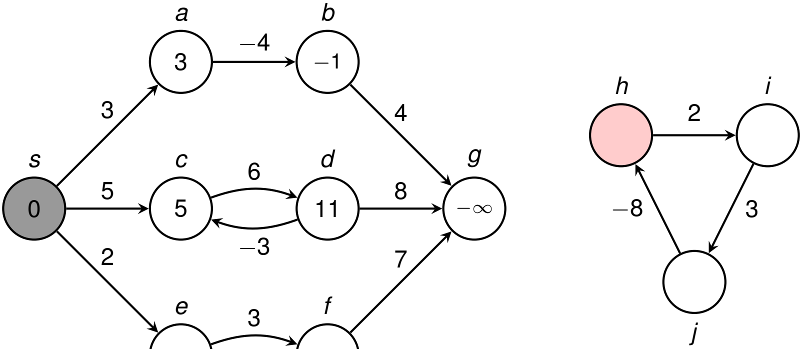
Distances and Negative-Weight Cycles (Figure 24.1)



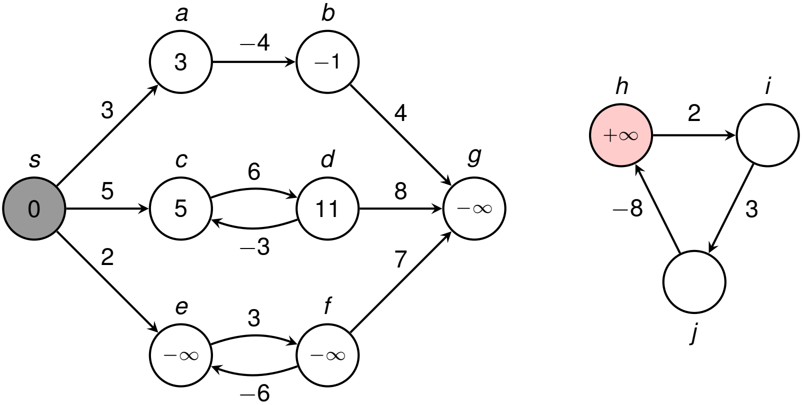
Distances and Negative-Weight Cycles (Figure 24.1)



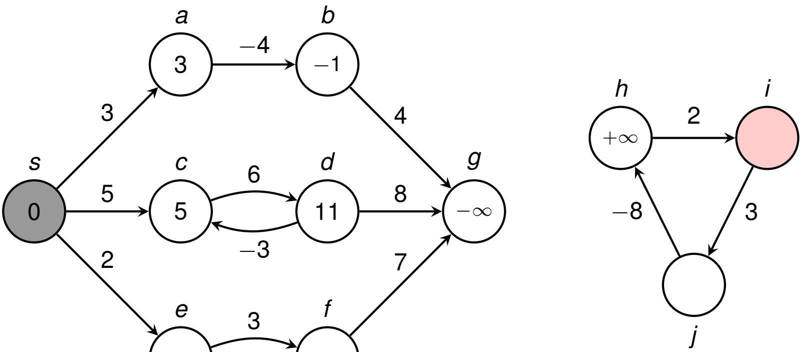
Distances and Negative-Weight Cycles (Figure 24.1)



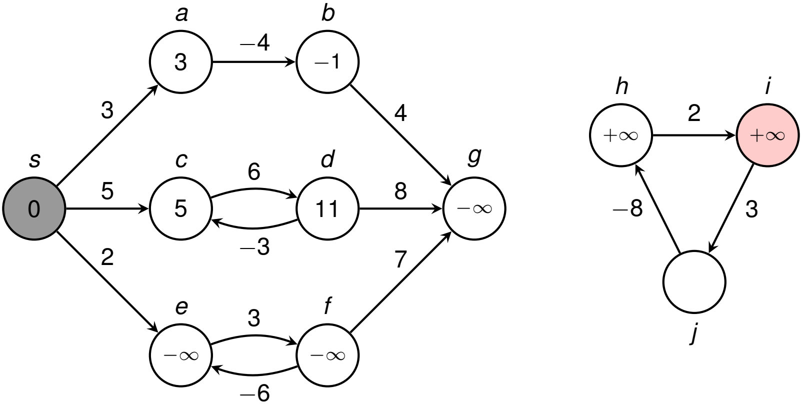
Distances and Negative-Weight Cycles (Figure 24.1)



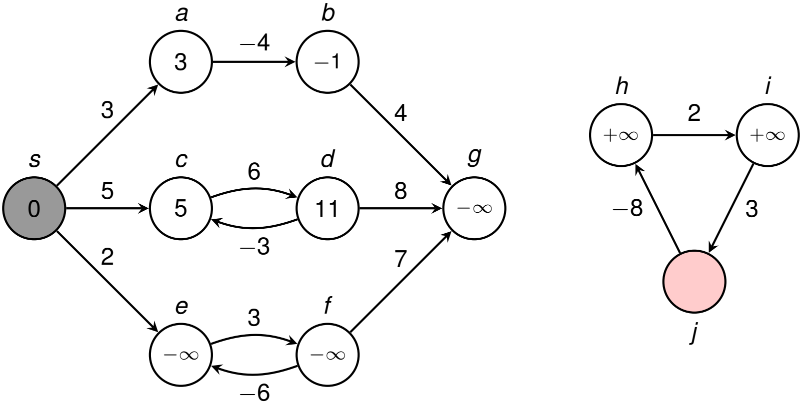
Distances and Negative-Weight Cycles (Figure 24.1)



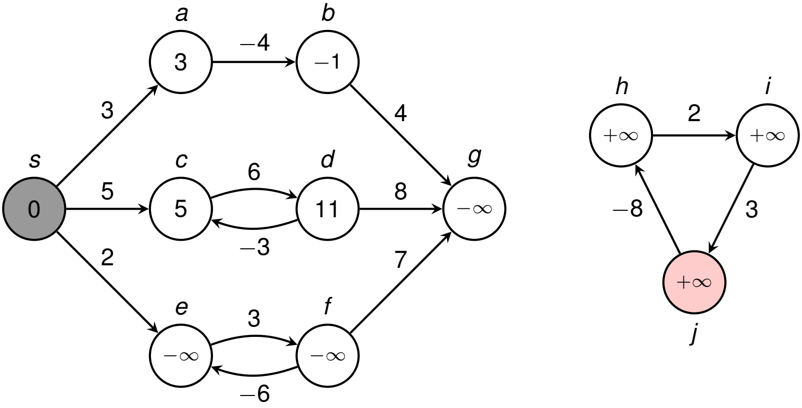
Distances and Negative-Weight Cycles (Figure 24.1)



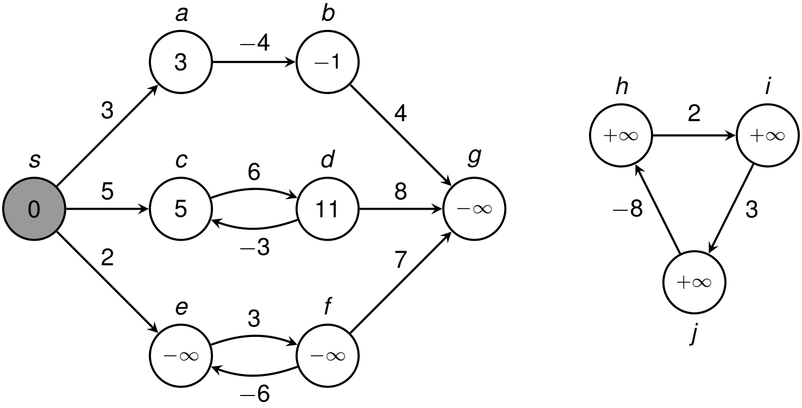
Distances and Negative-Weight Cycles (Figure 24.1)



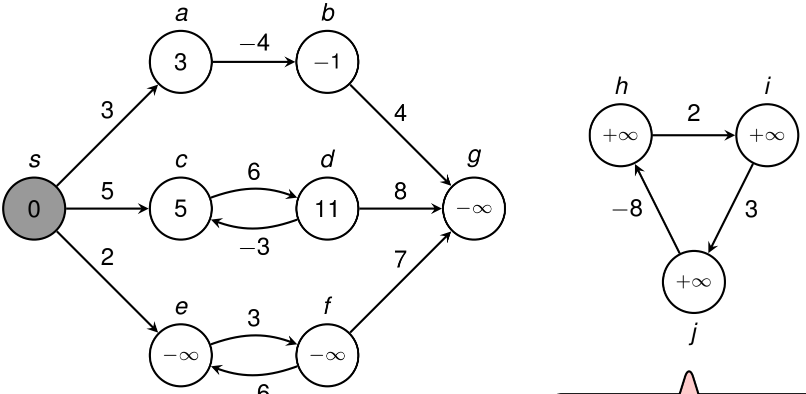
Distances and Negative-Weight Cycles (Figure 24.1)



Distances and Negative-Weight Cycles (Figure 24.1)



Distances and Negative-Weight Cycles (Figure 24.1)



Negative-Weight Cycle (reachable from *s*)

Negative-Weight Cycle (not reachable from *s*)

Outline

Introduction

Bellman-Ford Algorithm

Relaxing Edges

Definition

Fix the source vertex $s \in V$

- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

Relaxing Edges

Definition

Fix the source vertex $s \in V$

- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$

Relaxing Edges

Definition

Fix the source vertex $s \in V$

- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing Edges

Definition

Fix the source vertex $s \in V$

- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing an edge (u, v)

Given estimates $u.d$ and $v.d$, can we find a better path from v using the edge (u, v)?

Relaxing Edges

Definition

Fix the source vertex $s \in V$

- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing an edge (u, v)

Given estimates $u.d$ and $v.d$, can we find a better path from v using the edge (u, v)?

$$v.d \stackrel{?}{>} u.d + w(u, v)$$

Relaxing Edges

Definition

Fix the source vertex $s \in V$

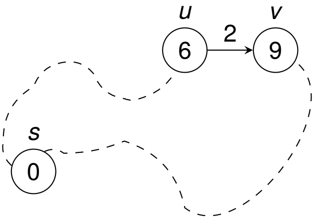
- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing an edge (u, v)

Given estimates $u.d$ and $v.d$, can we find a better path from v using the edge (u, v)?

$$v.d \stackrel{?}{>} u.d + w(u, v)$$



Relaxing Edges

Definition

Fix the source vertex $s \in V$

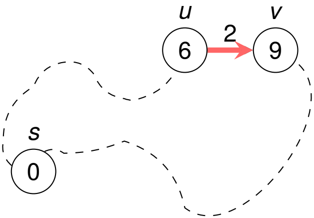
- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing an edge (u, v)

Given estimates $u.d$ and $v.d$, can we find a better path from v using the edge (u, v)?

$$v.d \stackrel{?}{>} u.d + w(u, v)$$



Relaxing Edges

Definition

Fix the source vertex $s \in V$

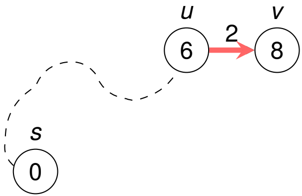
- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing an edge (u, v)

Given estimates $u.d$ and $v.d$, can we find a better path from v using the edge (u, v)?

$$v.d \overset{?}{>} u.d + w(u, v)$$



Relaxing Edges

Definition

Fix the source vertex $s \in V$

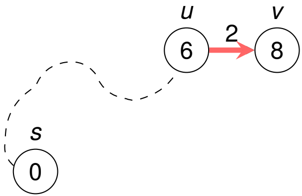
- $v.\delta$ is the length of the shortest path (distance) from s to v
- $v.d$ is the length of the shortest path discovered so far

- At the beginning: $s.d = s.\delta = 0$, $v.d = \infty$ for $v \neq s$
- At the end: $v.d = v.\delta$ for all $v \in V$

Relaxing an edge (u, v)

Given estimates $u.d$ and $v.d$, can we find a better path from v using the edge (u, v)?

$$v.d \overset{?}{>} u.d + w(u, v)$$



After relaxing (u, v), regardless of whether we found a shortcut: $v.d \leq u.d + w(u, v)$

Properties of Shortest Paths and Relaxations

Toolkit

Triangle inequality (Lemma 24.10)

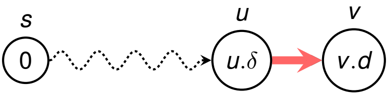
- For any edge $(u, v) \in E$, we have $v.\delta \leq u.\delta + w(u, v)$

Upper-bound Property (Lemma 24.11)

- We always have $v.d \geq v.\delta$ for all $v \in V$, and once $v.d$ achieves the value $v.\delta$, it never changes.

Convergence Property (Lemma 24.14)

- If $s \rightsquigarrow u \rightarrow v$ is a shortest path from s to v, and if $u.d = u.\delta$ prior to relaxing edge (u, v), then $v.d = v.\delta$ at all times afterward.



$v.d$

Properties of Shortest Paths and Relaxations

Toolkit

Triangle inequality (Lemma 24.10)

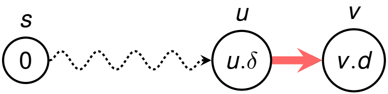
- For any edge $(u, v) \in E$, we have $v.\delta \leq u.\delta + w(u, v)$

Upper-bound Property (Lemma 24.11)

- We always have $v.d \geq v.\delta$ for all $v \in V$, and once $v.d$ achieves the value $v.\delta$, it never changes.

Convergence Property (Lemma 24.14)

- If $s \rightsquigarrow u \to v$ is a shortest path from s to v, and if $u.d = u.\delta$ prior to relaxing edge (u, v), then $v.d = v.\delta$ at all times afterward.



$$v.d \leq u.d + w(u, v)$$

Properties of Shortest Paths and Relaxations

Toolkit

Triangle inequality (Lemma 24.10)

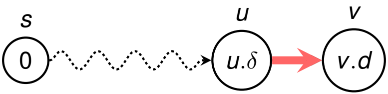
- For any edge $(u, v) \in E$, we have $v.\delta \leq u.\delta + w(u, v)$

Upper-bound Property (Lemma 24.11)

- We always have $v.d \geq v.\delta$ for all $v \in V$, and once $v.d$ achieves the value $v.\delta$, it never changes.

Convergence Property (Lemma 24.14)

- If $s \rightsquigarrow u \to v$ is a shortest path from s to v, and if $u.d = u.\delta$ prior to relaxing edge (u, v), then $v.d = v.\delta$ at all times afterward.



$$\begin{aligned} v.d &\leq u.d + w(u, v) \\ &= u.\delta + w(u, v) \end{aligned}$$

Properties of Shortest Paths and Relaxations

Toolkit

Triangle inequality (Lemma 24.10)

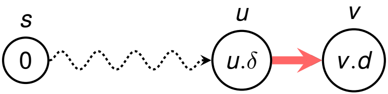
- For any edge $(u, v) \in E$, we have $v.\delta \leq u.\delta + w(u, v)$

Upper-bound Property (Lemma 24.11)

- We always have $v.d \geq v.\delta$ for all $v \in V$, and once $v.d$ achieves the value $v.\delta$, it never changes.

Convergence Property (Lemma 24.14)

- If $s \rightsquigarrow u \rightarrow v$ is a shortest path from s to v, and if $u.d = u.\delta$ prior to relaxing edge (u, v), then $v.d = v.\delta$ at all times afterward.



$$\begin{aligned} v.d &\leq u.d + w(u, v) \\ &= u.\delta + w(u, v) \\ &= v.\delta \end{aligned}$$

Properties of Shortest Paths and Relaxations

Toolkit

Triangle inequality (Lemma 24.10)

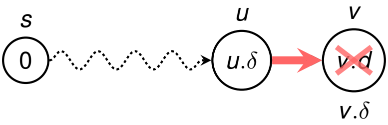
- For any edge $(u, v) \in E$, we have $v.\delta \leq u.\delta + w(u, v)$

Upper-bound Property (Lemma 24.11)

- We always have $v.d \geq v.\delta$ for all $v \in V$, and once $v.d$ achieves the value $v.\delta$, it never changes.

Convergence Property (Lemma 24.14)

- If $s \rightsquigarrow u \rightarrow v$ is a shortest path from s to v, and if $u.d = u.\delta$ prior to relaxing edge (u, v), then $v.d = v.\delta$ at all times afterward.



$$\begin{aligned} v.d &\leq u.d + w(u, v) \\ &= u.\delta + w(u, v) \\ &= v.\delta \end{aligned}$$

Since $v.d \geq v.\delta$, we have $v.d = v.\delta$. □

Path-Relaxation Property

Path-Relaxation Property (Lemma 24.15)

If $p = (v_0, v_1, \dots, v_k)$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$, then $v_k.d = v_k.\delta$ (regardless of the order of other relaxation steps).

Path-Relaxation Property

Path-Relaxation Property (Lemma 24.15)

If $p = (v_0, v_1, \ldots, v_k)$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = v_k.\delta$ (regardless of the order of other relaxation steps).

Proof:

- By induction on i, $0 \leq i \leq k$:
  After the ith edge of p is relaxed, we have $v_i.d = v_i.\delta$.

Path-Relaxation Property

Path-Relaxation Property (Lemma 24.15)

If $p = (v_0, v_1, \ldots, v_k)$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = v_k.\delta$ (regardless of the order of other relaxation steps).

Proof:

- By induction on i, $0 \leq i \leq k$:
  After the ith edge of p is relaxed, we have $v_i.d = v_i.\delta$.
- For $i = 0$, by the initialization $s.d = s.\delta = 0$.
  Upper-bound Property $\Rightarrow$ the value of $s.d$ never changes after that.

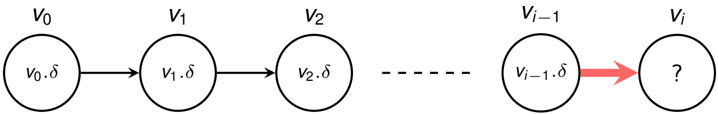
Path-Relaxation Property

Path-Relaxation Property (Lemma 24.15)

If $p = (v_0, v_1, \ldots, v_k)$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = v_k.\delta$ (regardless of the order of other relaxation steps).

Proof:

- By induction on i, $0 \leq i \leq k$:
  After the ith edge of p is relaxed, we have $v_i.d = v_i.\delta$.
- For $i = 0$, by the initialization $s.d = s.\delta = 0$.
  Upper-bound Property $\Rightarrow$ the value of $s.d$ never changes after that.
- Inductive Step ($i - 1 \to i$): Assume $v_{i-1}.d = v_{i-1}.\delta$ and relax (v_{i-1}, v_i).



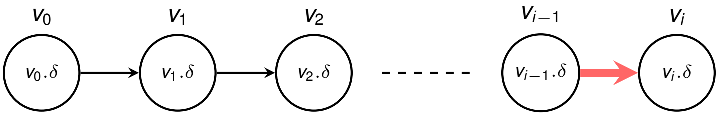
Path-Relaxation Property

Path-Relaxation Property (Lemma 24.15)

If $p = (v_0, v_1, \ldots, v_k)$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = v_k.\delta$ (regardless of the order of other relaxation steps).

Proof:

- By induction on i, $0 \leq i \leq k$:
  After the ith edge of p is relaxed, we have $v_i.d = v_i.\delta$.
- For $i = 0$, by the initialization $s.d = s.\delta = 0$.
  Upper-bound Property $\Rightarrow$ the value of $s.d$ never changes after that.
- Inductive Step ($i - 1 \rightarrow i$): Assume $v_{i-1}.d = v_{i-1}.\delta$ and relax (v_{i-1}, v_i).
  Convergence Property $\Rightarrow$ $v_i.d = v_i.\delta$ (now and at all later steps) $\square$



Path-Relaxation Property

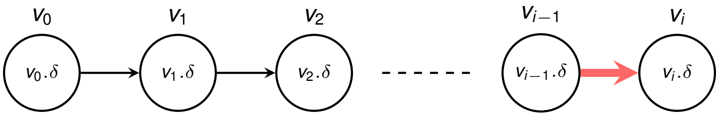
"Propagation": By relaxing proper edges, set of vertices with $v.\delta = v.d$ gets larger

Path-Relaxation Property (Lemma 24.15)

If $p = (v_0, v_1, \ldots, v_k)$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = v_k.\delta$ (regardless of the order of other relaxation steps).

Proof:

- By induction on i, $0 \leq i \leq k$:
  After the ith edge of p is relaxed, we have $v_i.d = v_i.\delta$.
- For $i = 0$, by the initialization $s.d = s.\delta = 0$.
  Upper-bound Property $\Rightarrow$ the value of $s.d$ never changes after that.
- Inductive Step ($i - 1 \rightarrow i$): Assume $v_{i-1}.d = v_{i-1}.\delta$ and relax (v_{i-1}, v_i).
  Convergence Property $\Rightarrow$ $v_i.d = v_i.\delta$ (now and at all later steps) □



The Bellman-Ford Algorithm

```
BELLMAN-FORD(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0
repeat |V|-1 times
   for e in G.edges()
   Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
      if e.start.d + e.weight.d < e.end.d:
         e.end.d = e.start.d + e.weight
         e.end.predecessor = e.start
for e in G.edges()
   if e.start.d + e.weight.d < e.end.d:
      return FALSE
return TRUE
```

The Bellman-Ford Algorithm

```
BELLMAN-FORD(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0

repeat |V|-1 times
   for e in G.edges()
   Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
      if e.start.d + e.weight.d < e.end.d:
         e.end.d = e.start.d + e.weight
         e.end.predecessor = e.start

for e in G.edges()
   if e.start.d + e.weight.d < e.end.d:
      return FALSE
return TRUE
```

Time Complexity

The Bellman-Ford Algorithm

```
 BELLMAN-FORD(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V|-1 times
7:    for e in G.edges()
8:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
9:       if e.start.d + e.weight.d < e.end.d:
10:         e.end.d = e.start.d + e.weight
11:         e.end.predecessor = e.start
12:
13: for e in G.edges()
14:    if e.start.d + e.weight.d < e.end.d:
15:       return FALSE
16: return TRUE
```

Time Complexity

- A single call of line 9-11 costs $\mathcal{O}(1)$

The Bellman-Ford Algorithm

```
 BELLMAN-FORD(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V|-1 times
7:    for e in G.edges()
8:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
9:       if e.start.d + e.weight.d < e.end.d:
10:         e.end.d = e.start.d + e.weight
11:         e.end.predecessor = e.start
12:
13: for e in G.edges()
14:    if e.start.d + e.weight.d < e.end.d:
15:       return FALSE
16: return TRUE
```

Time Complexity

- A single call of line 9-11 costs $\mathcal{O}(1)$
- In each pass every edge is relaxed $\Rightarrow \mathcal{O}(E)$ time per pass

The Bellman-Ford Algorithm

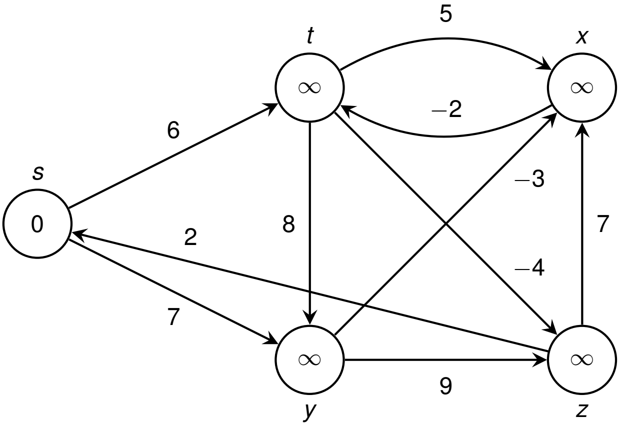
```
 BELLMAN-FORD(G,w,s)
0: assert(s in G.vertices())
1: for v in G.vertices()
2:    v.predecessor = None
3:    v.d = Infinity
4: s.d = 0
5:
6: repeat |V|-1 times
7:    for e in G.edges()
8:    Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
9:       if e.start.d + e.weight.d < e.end.d:
10:          e.end.d = e.start.d + e.weight
11:          e.end.predecessor = e.start
12:
13: for e in G.edges()
14:    if e.start.d + e.weight.d < e.end.d:
15:       return FALSE
16: return TRUE
```

Time Complexity

- A single call of line 9-11 costs $\mathcal{O}(1)$
- In each pass every edge is relaxed $\Rightarrow \mathcal{O}(E)$ time per pass
- Overall $(V-1)+1 = V$ passes $\Rightarrow \mathcal{O}(V \cdot E)$ time

Execution of Bellman-Ford (Figure 24.4)

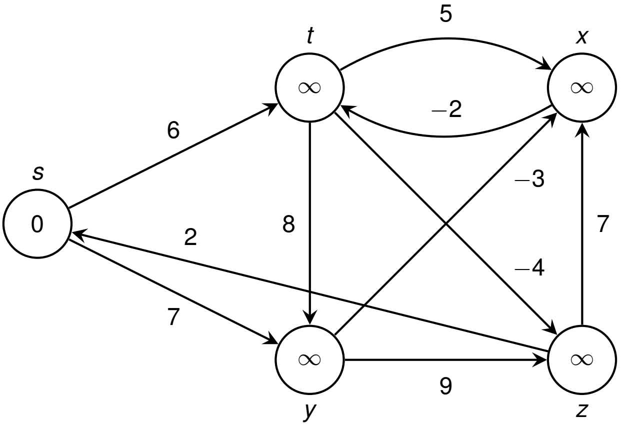
Pass: 1



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

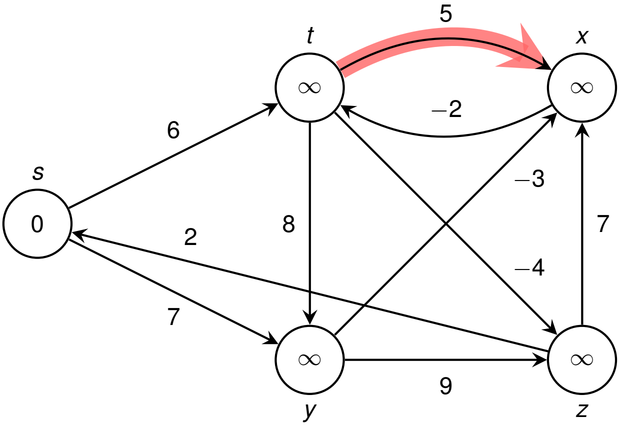
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

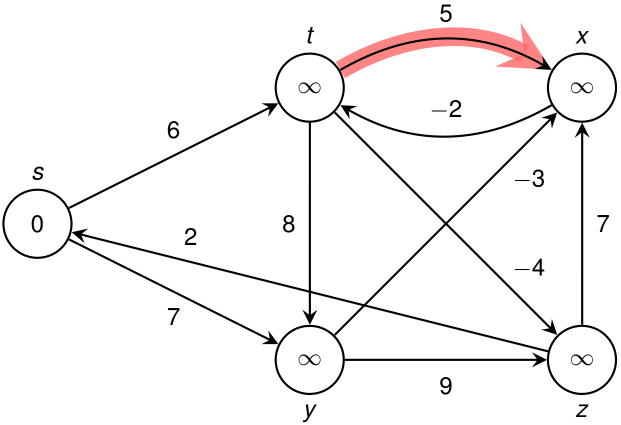
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

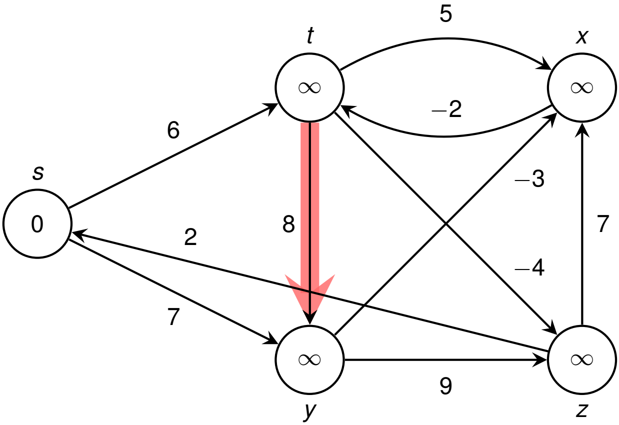
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

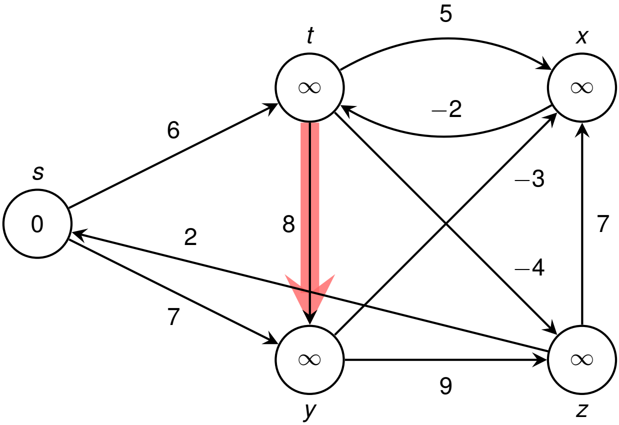
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

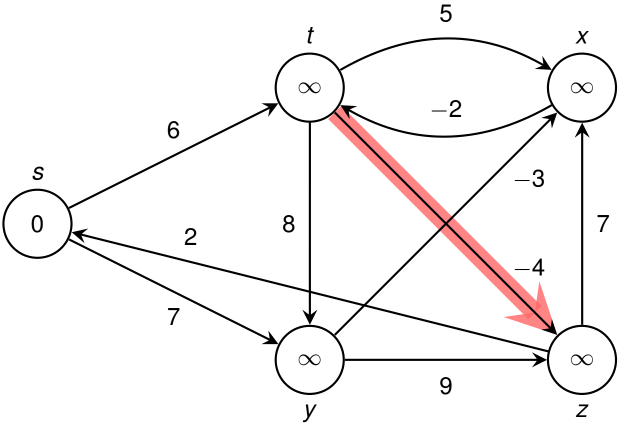
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

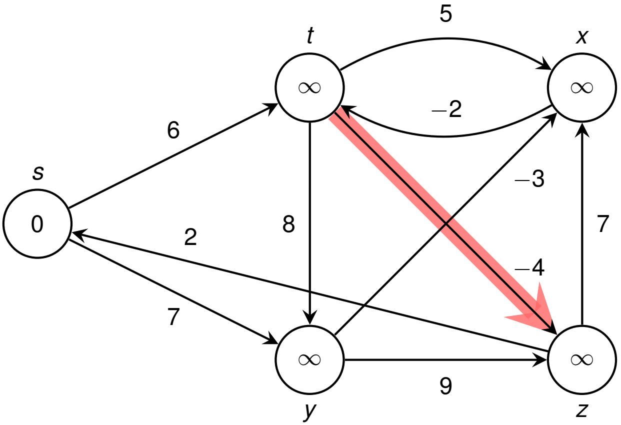
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

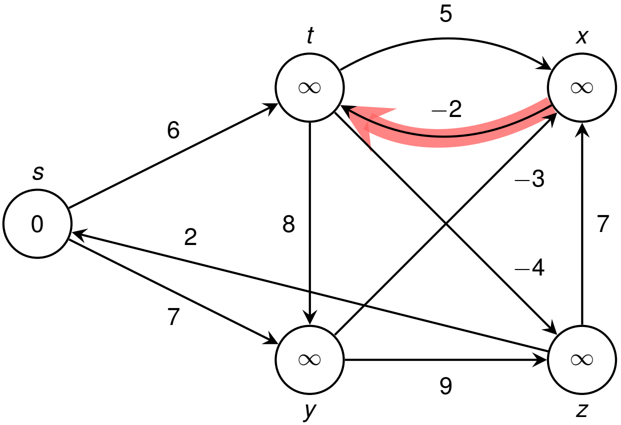
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

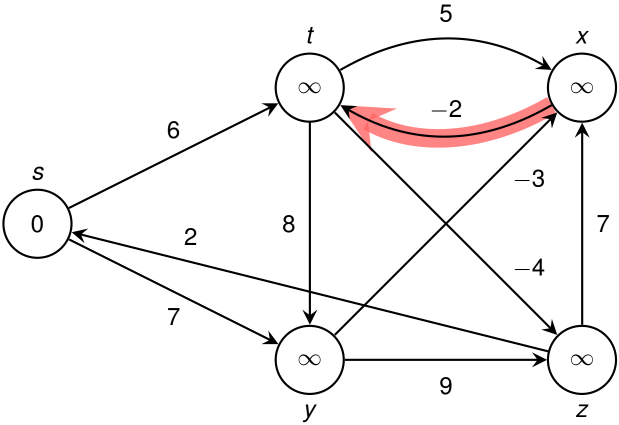
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

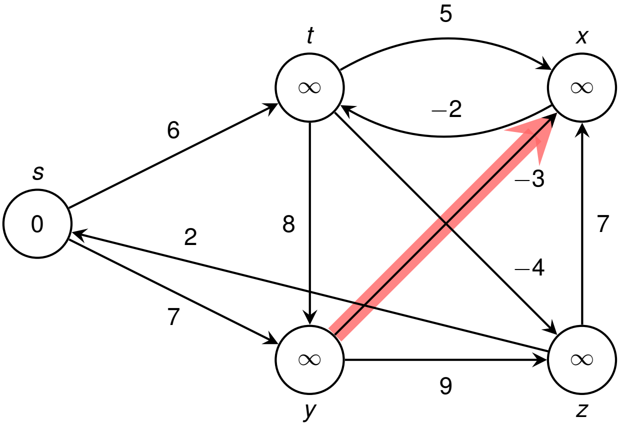
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

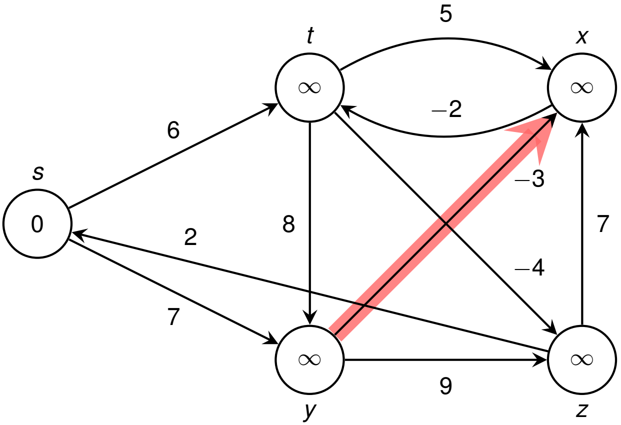
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

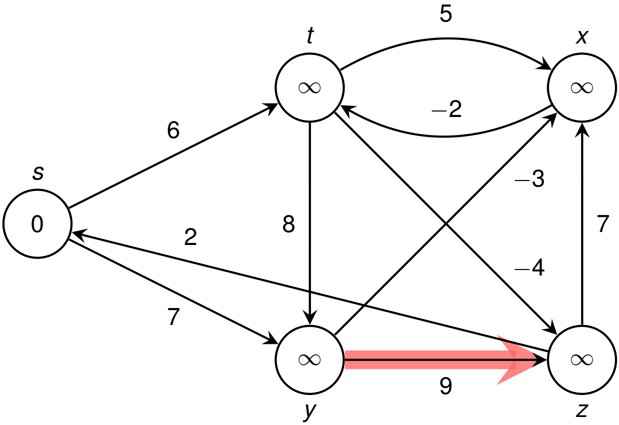
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

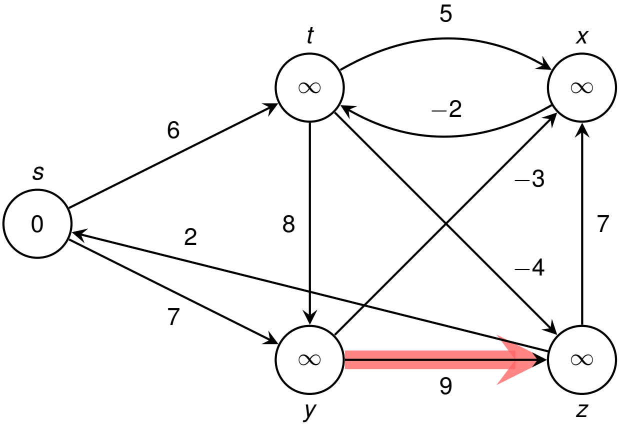
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

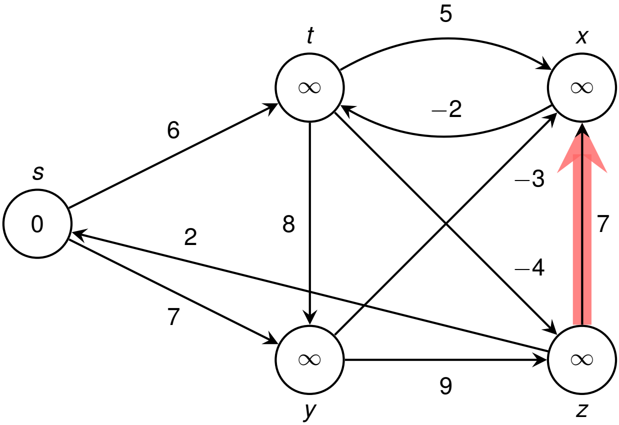
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

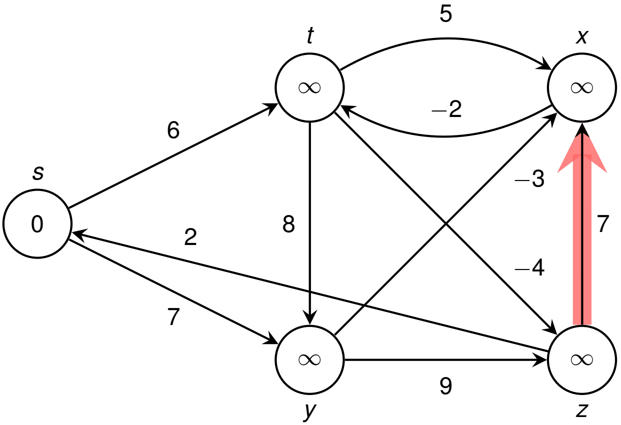
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

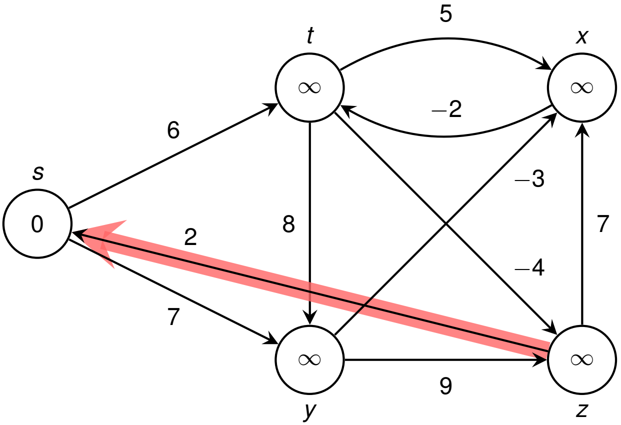
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

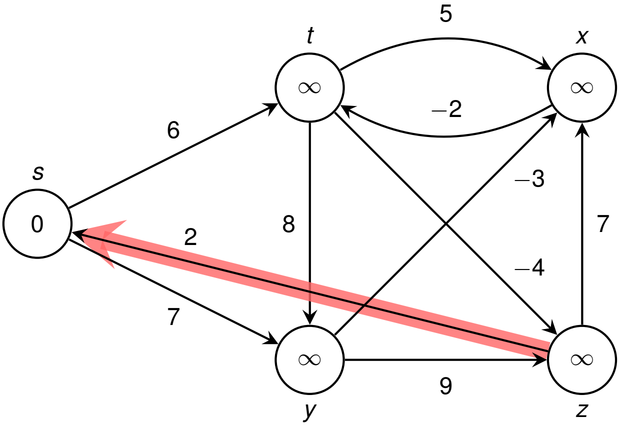
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

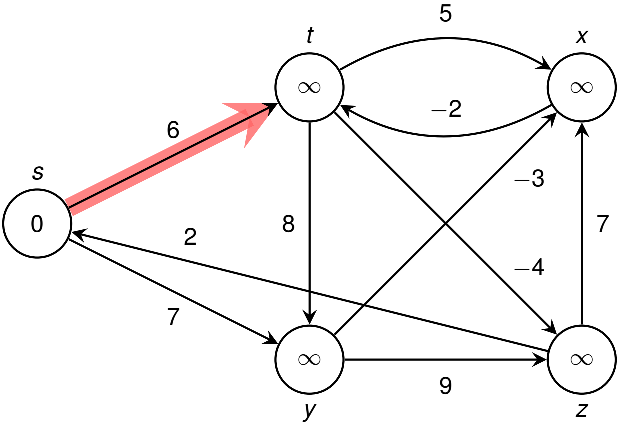
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

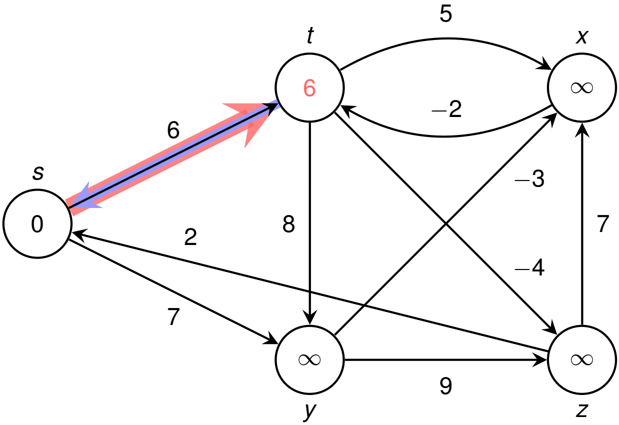
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

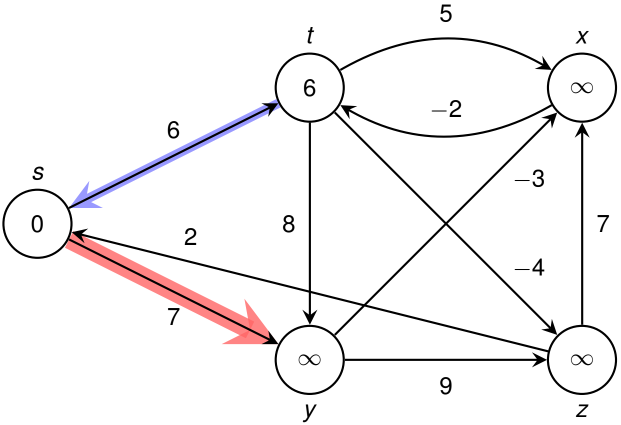
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

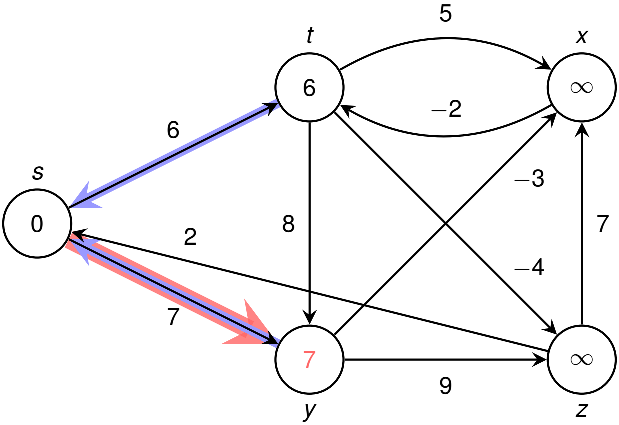
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 1

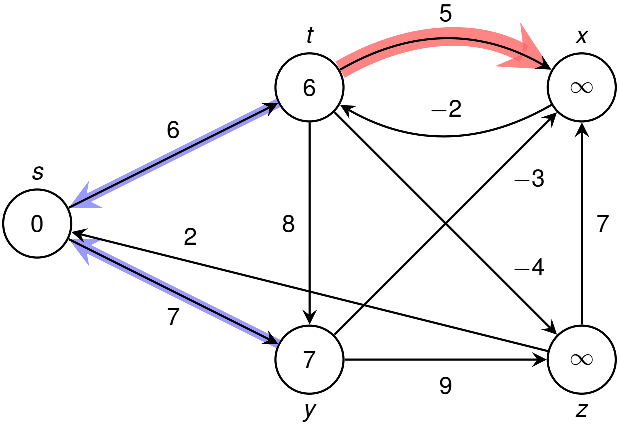
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

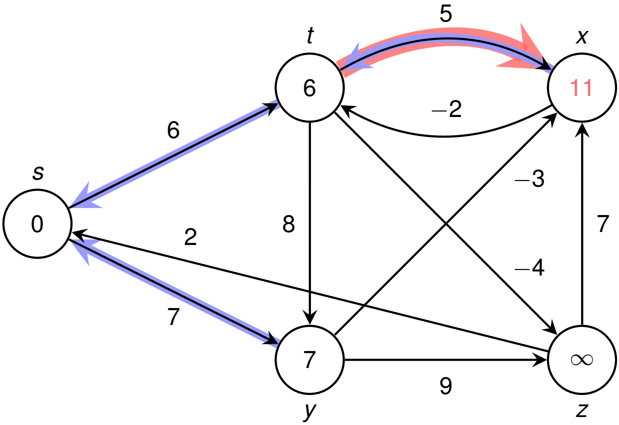
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

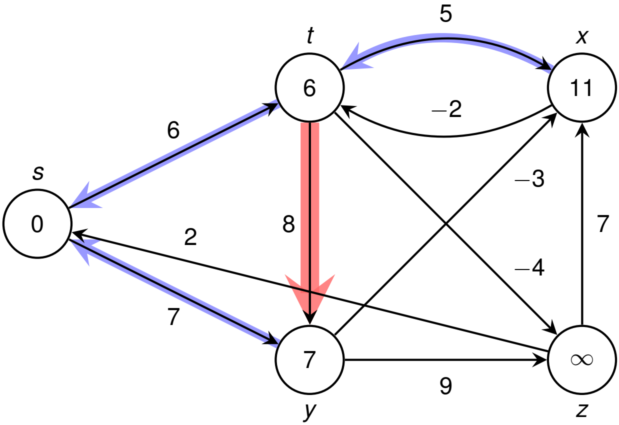
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

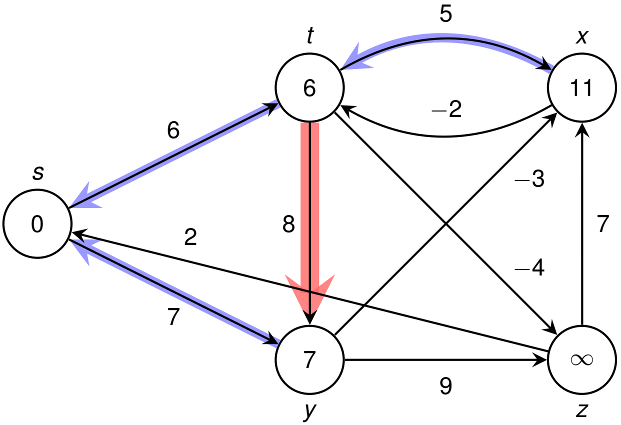
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

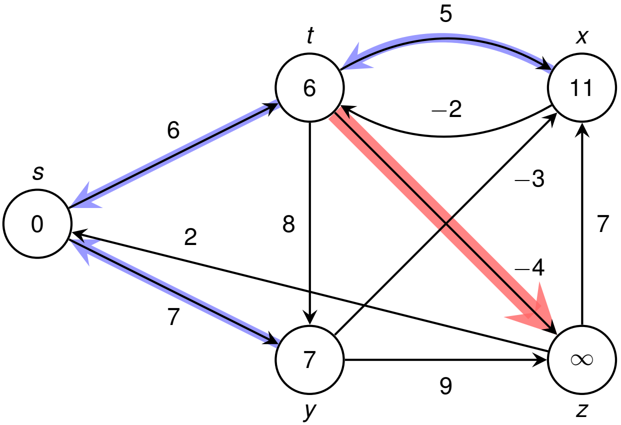
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

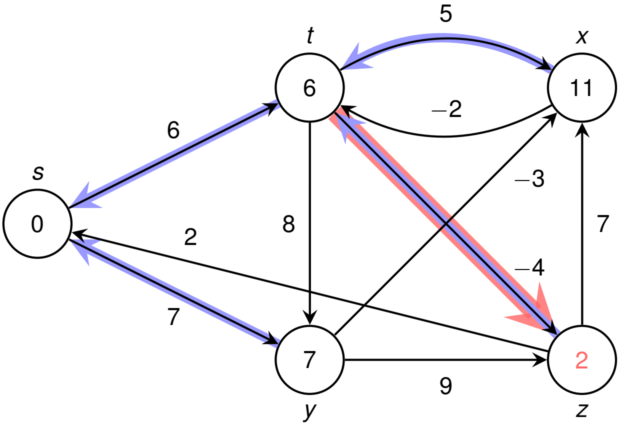
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

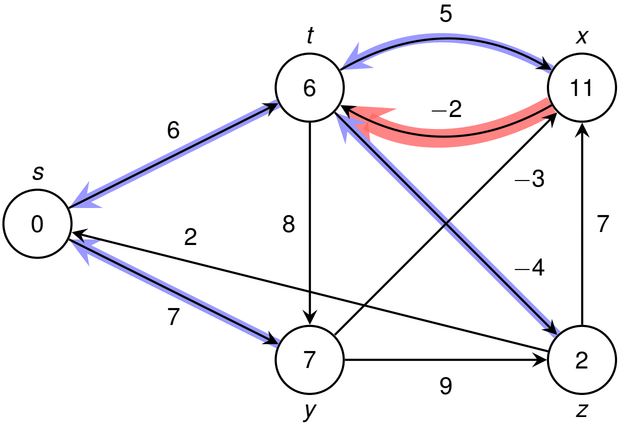
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

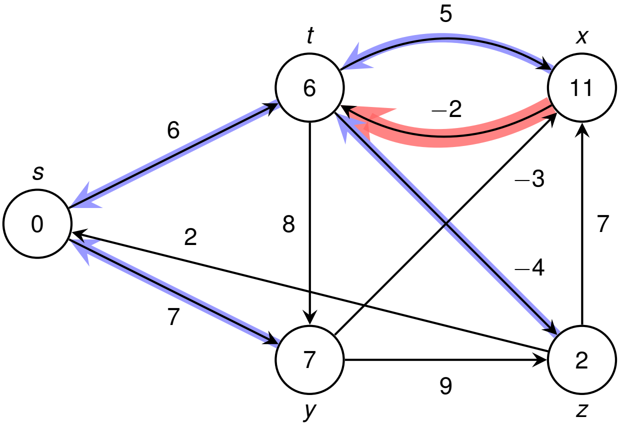
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

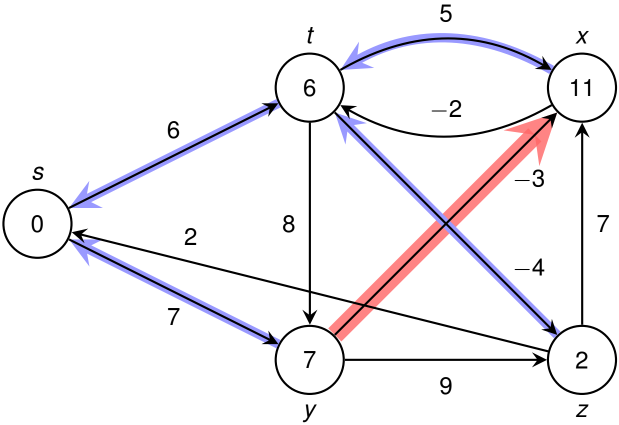
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

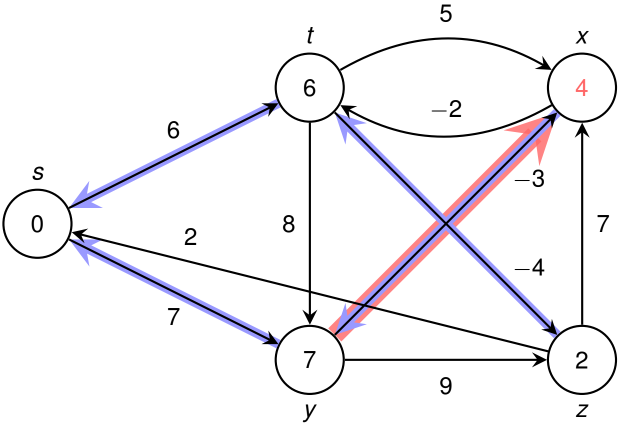
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

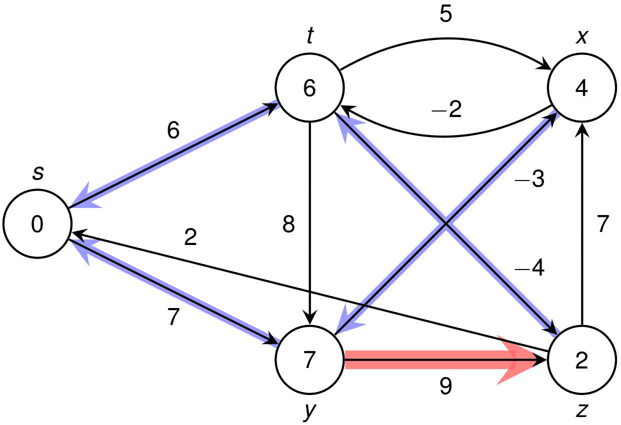
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

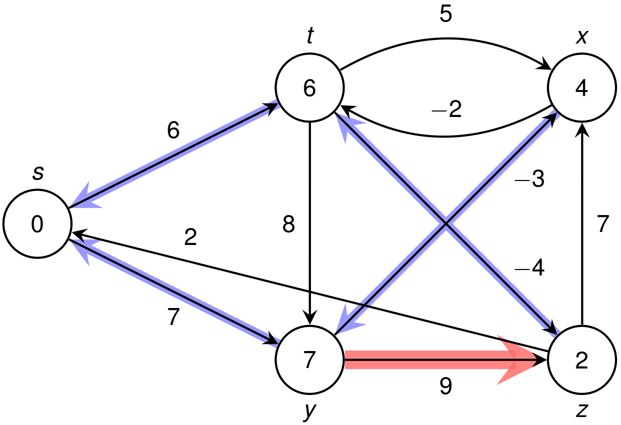
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

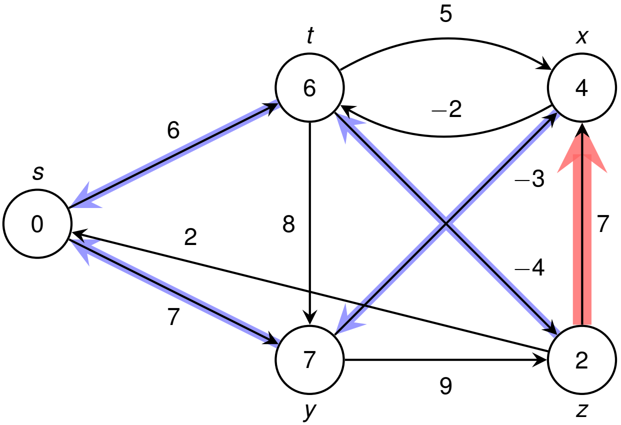
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

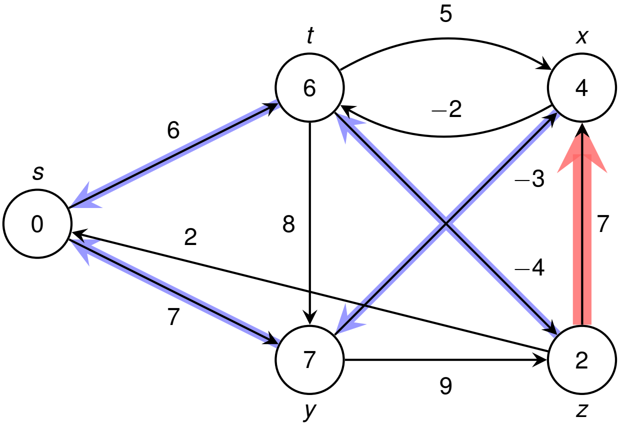
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

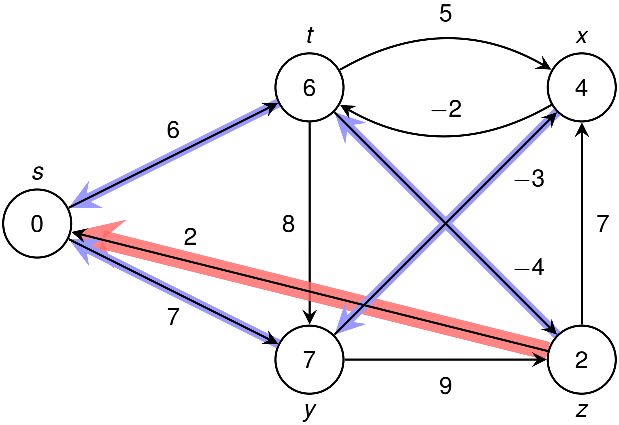
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

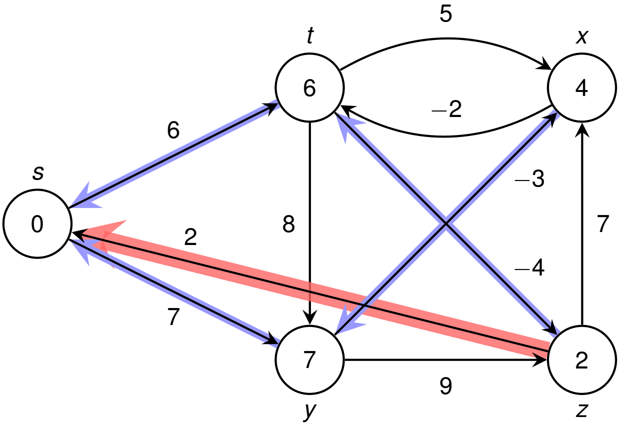
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

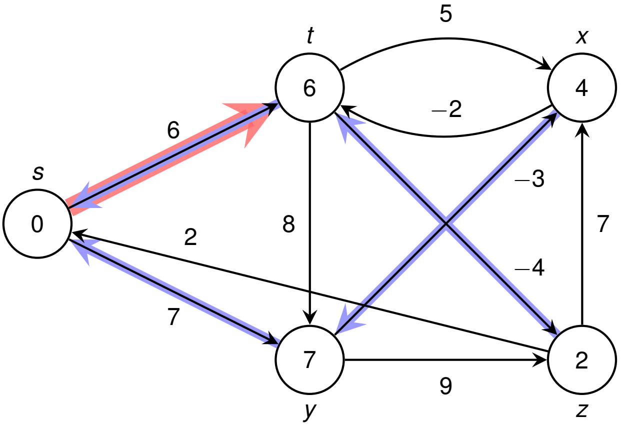
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

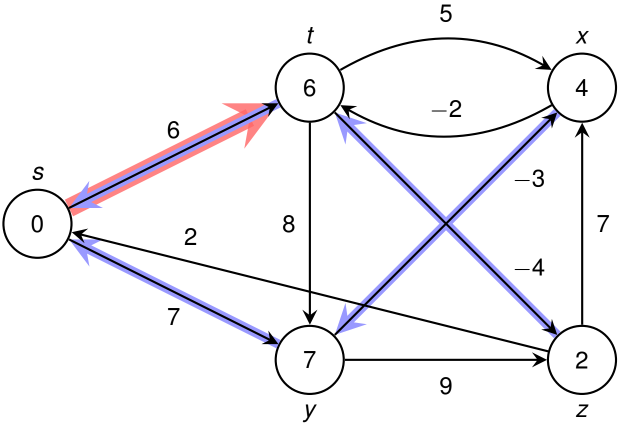
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

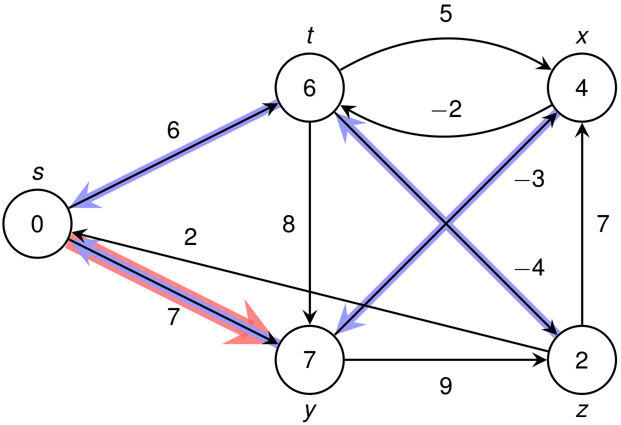
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

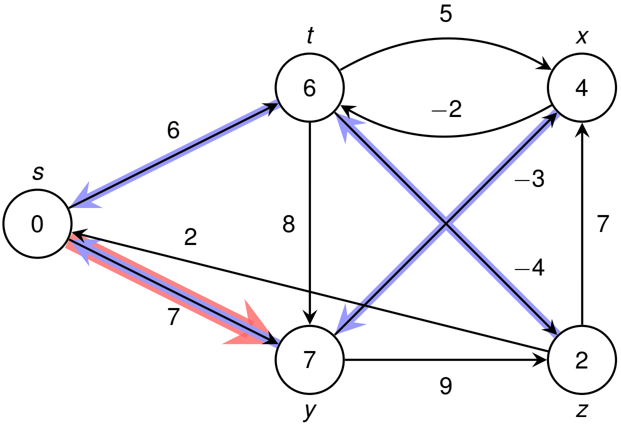
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 2

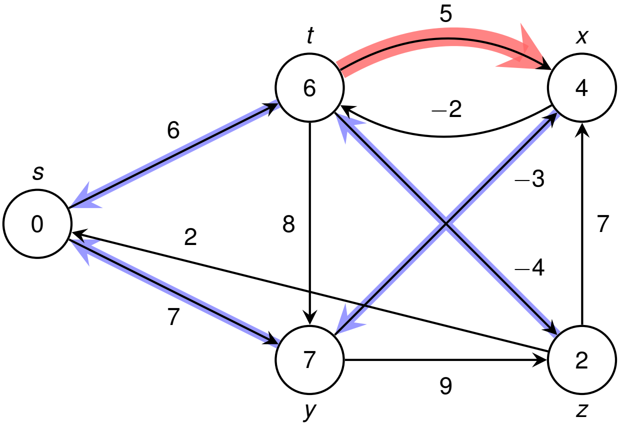
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

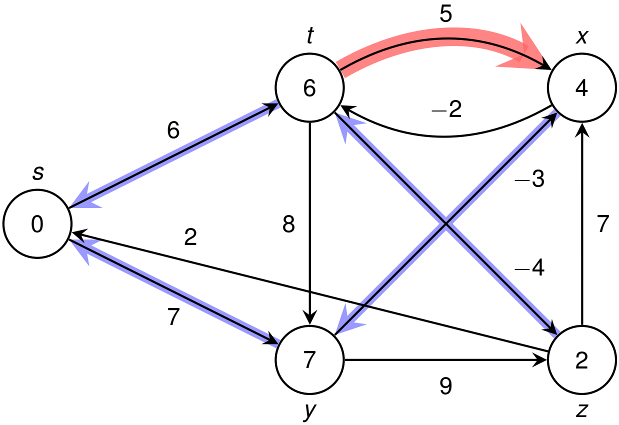
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

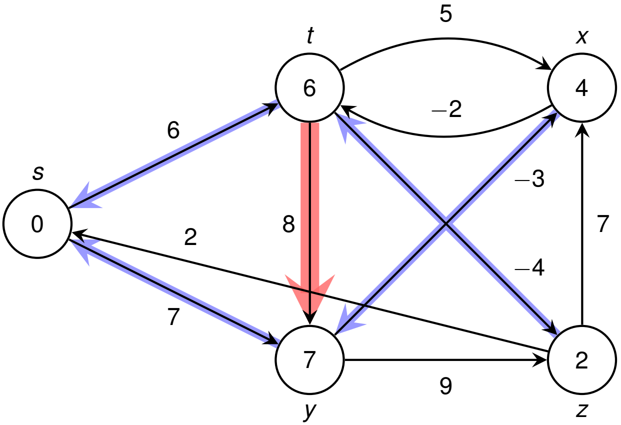
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

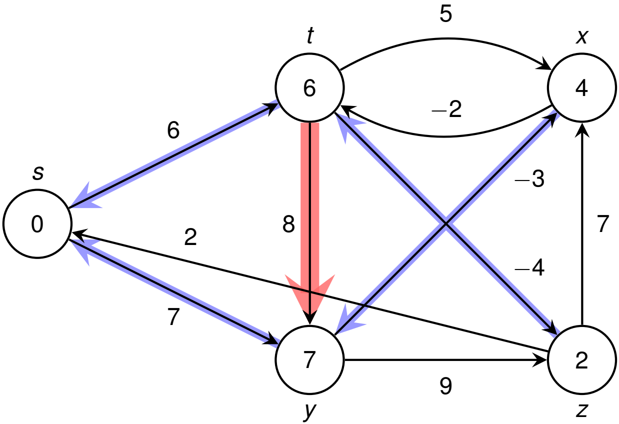
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

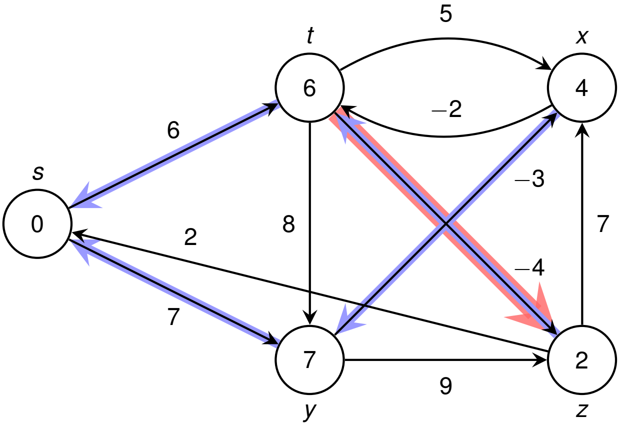
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

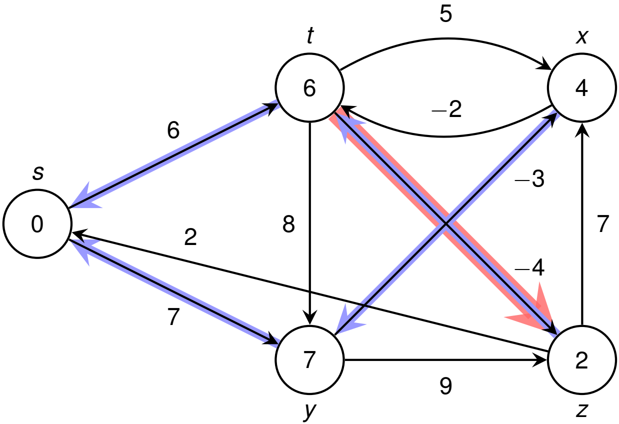
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

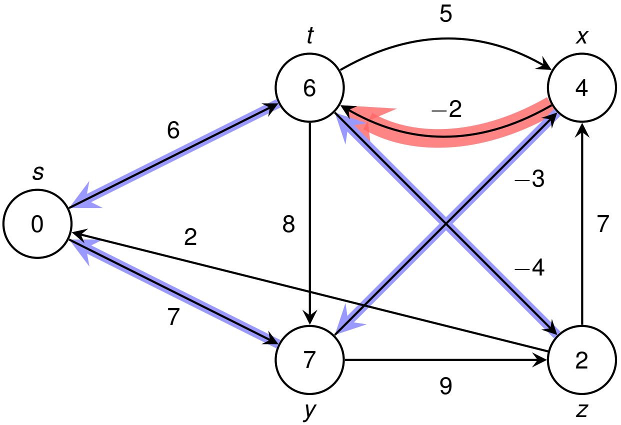
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

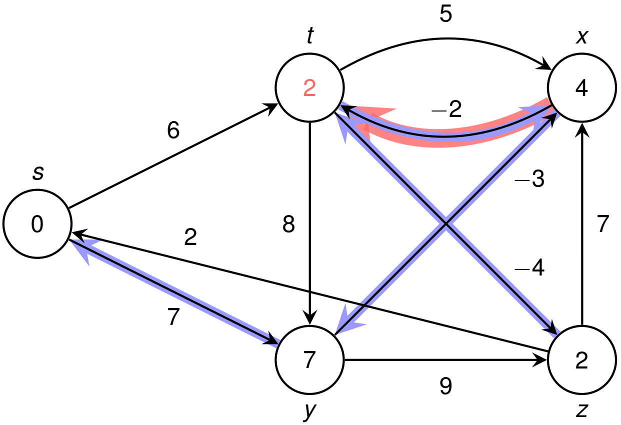
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

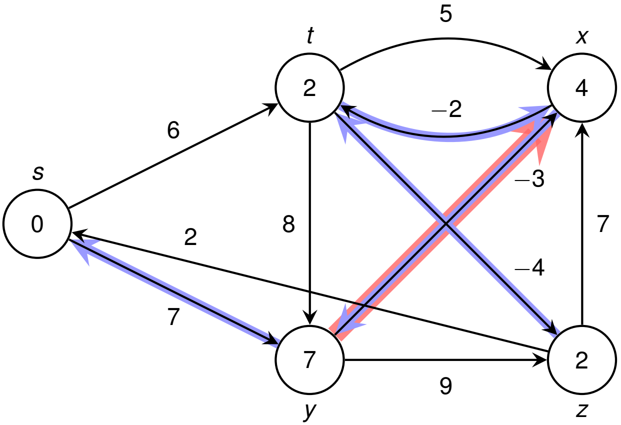
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

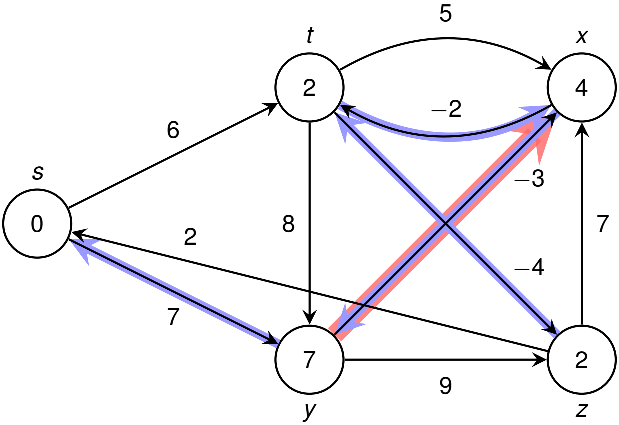
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

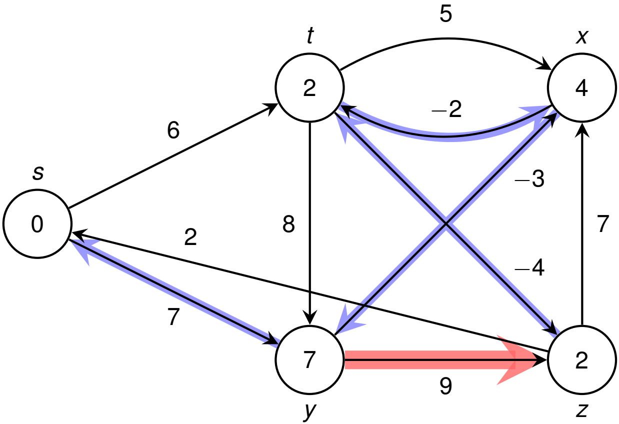
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

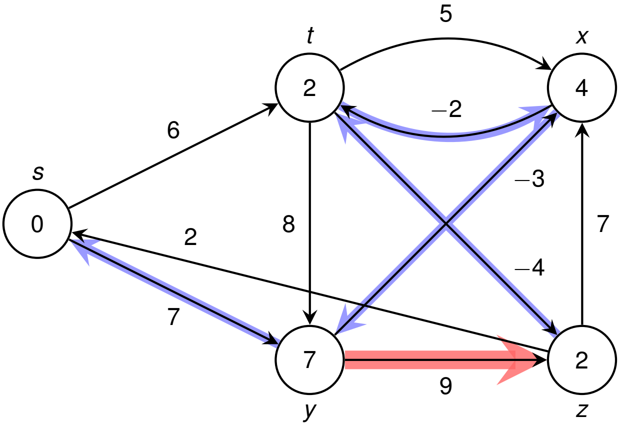
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

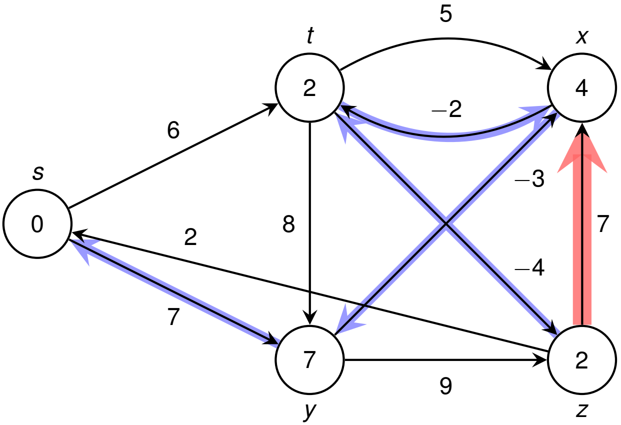
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

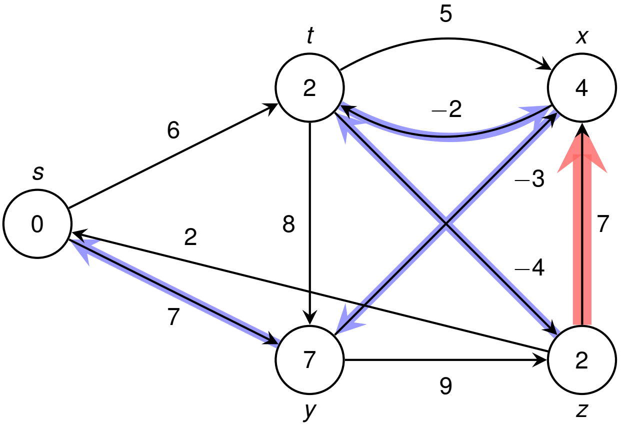
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

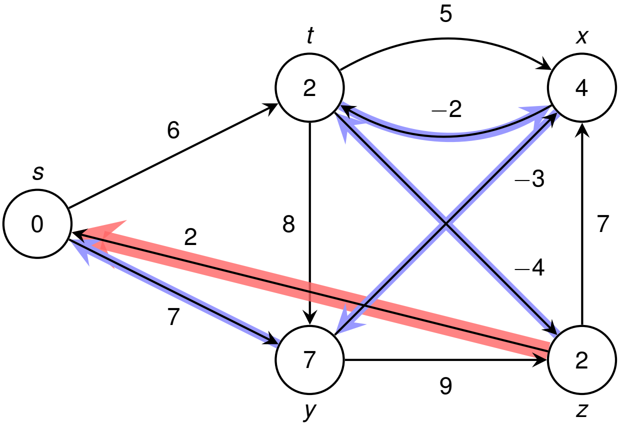
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

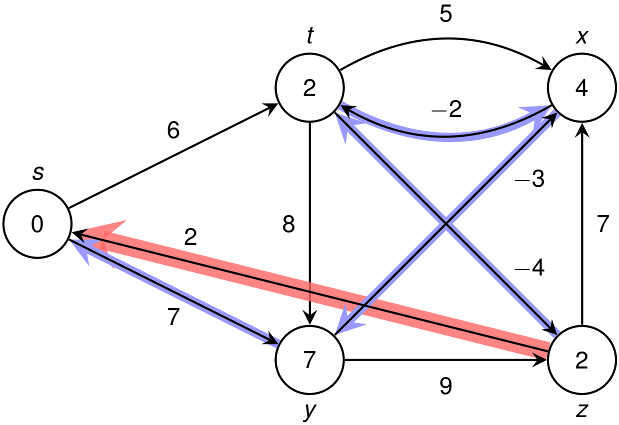
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

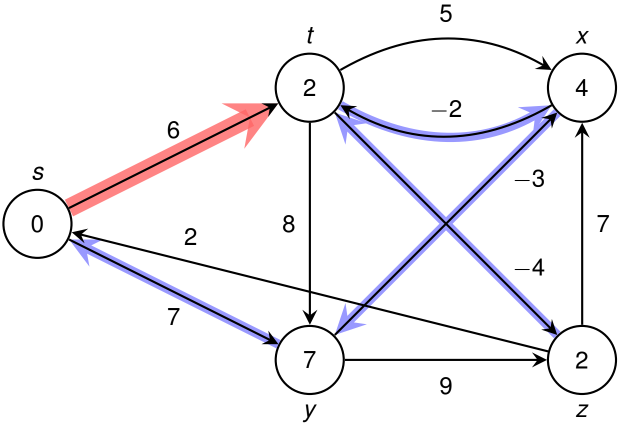
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

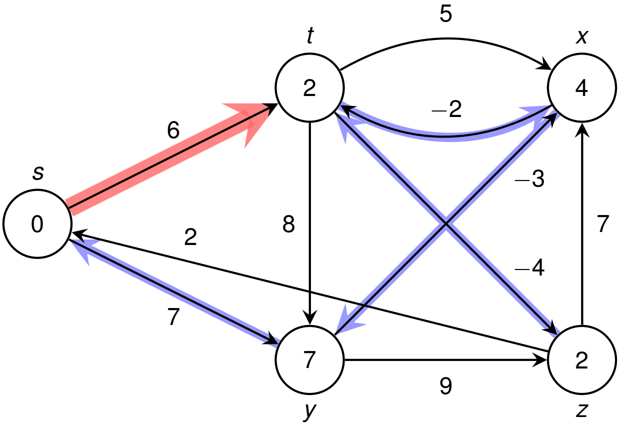
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

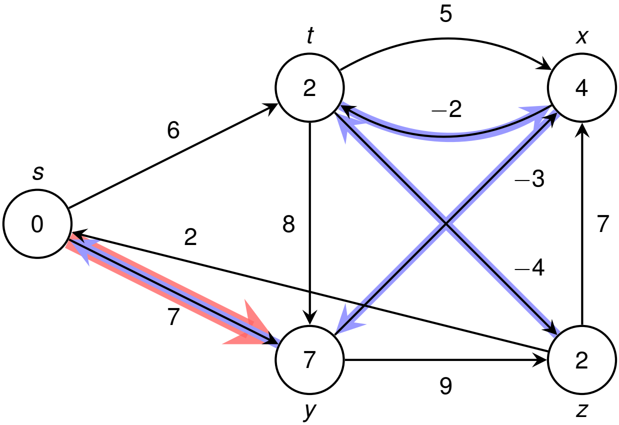
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

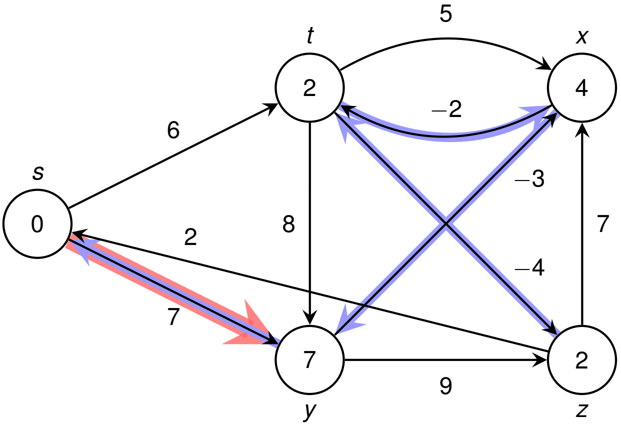
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 3

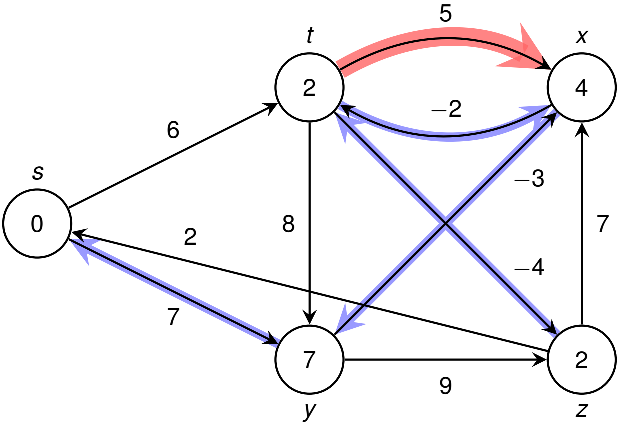
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

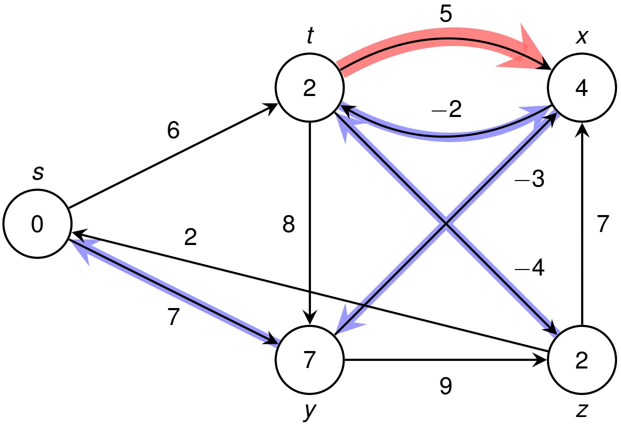
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

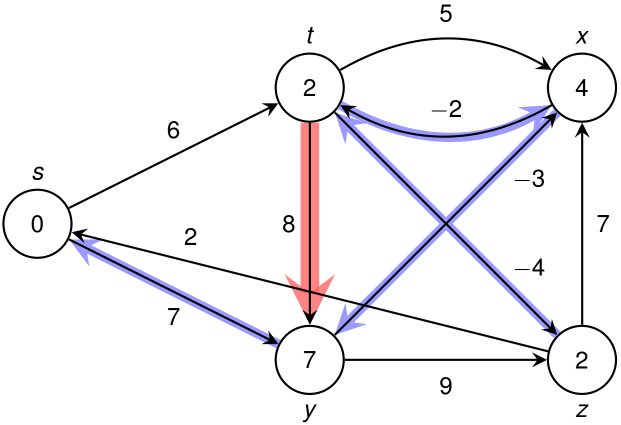
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

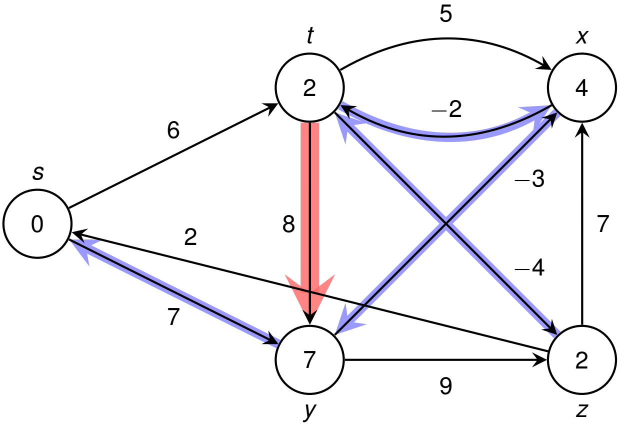
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

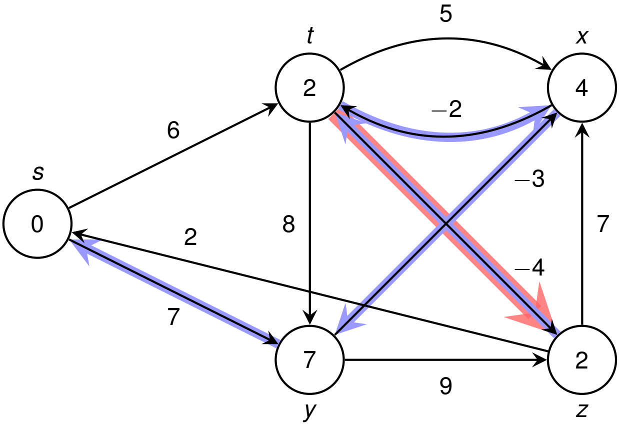
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

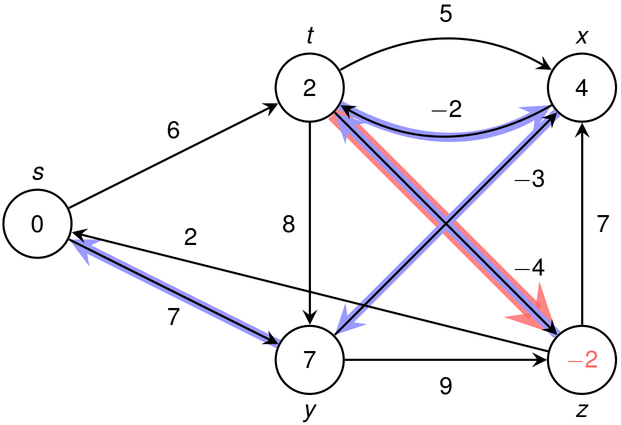
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

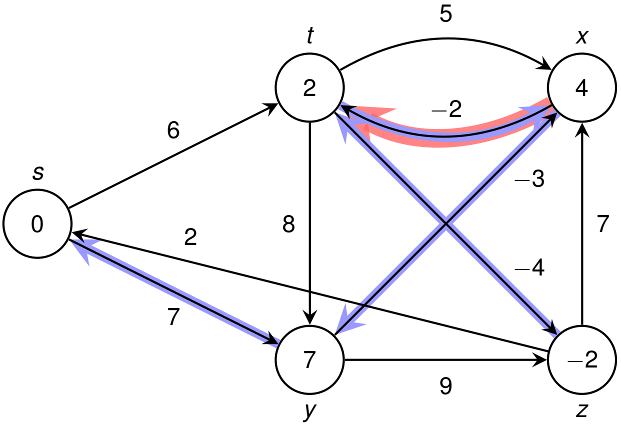
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

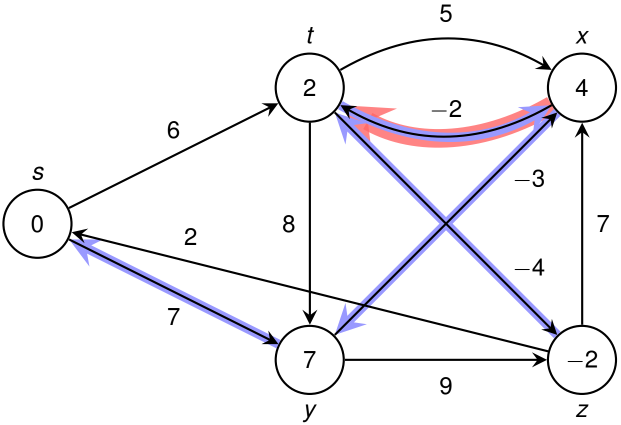
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

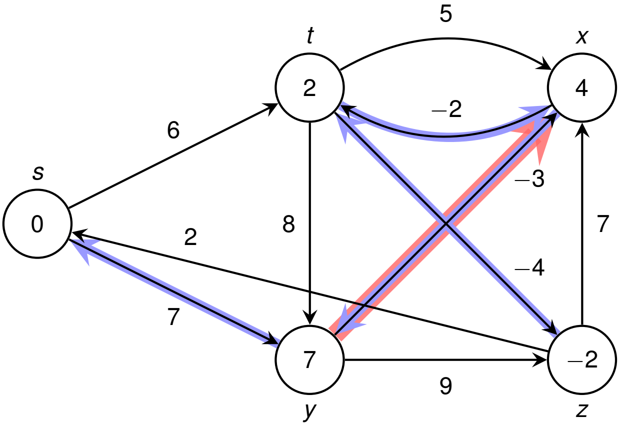
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

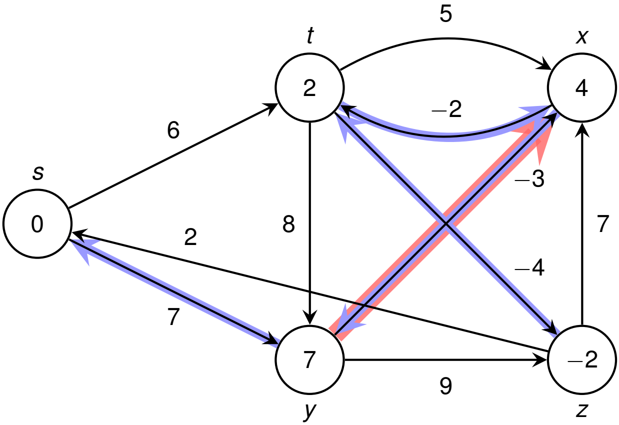
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

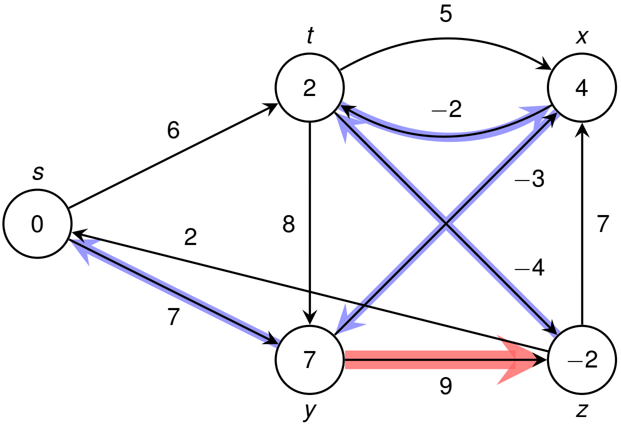
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

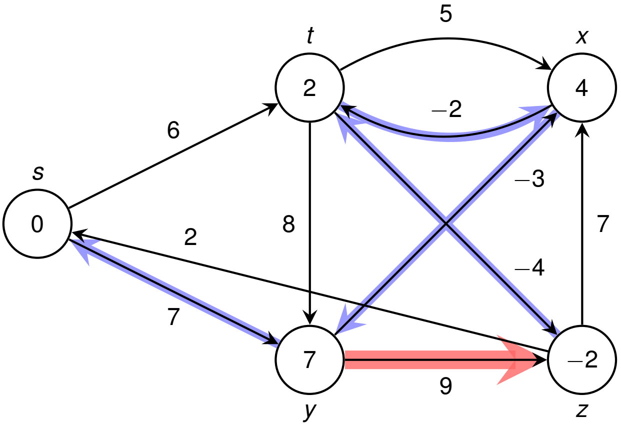
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

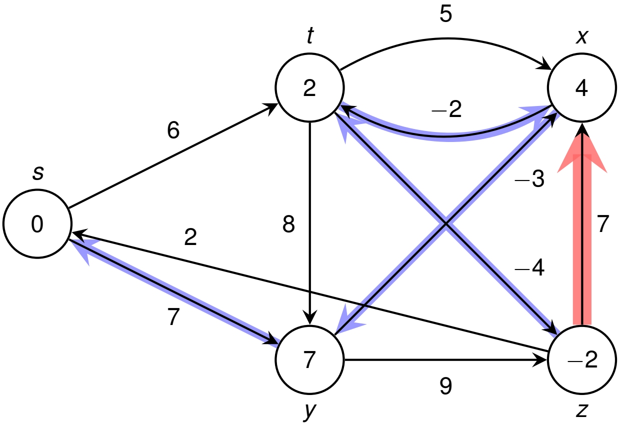
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

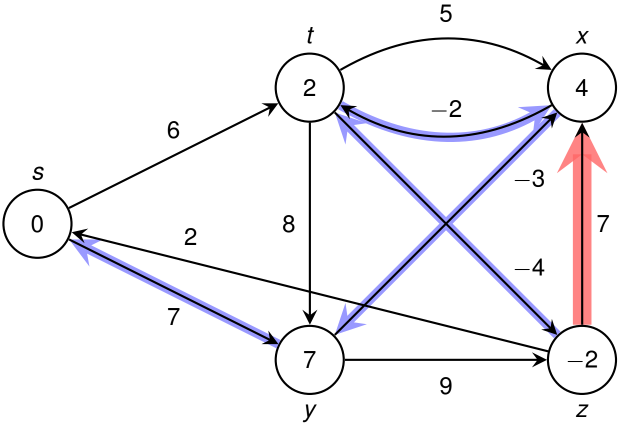
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

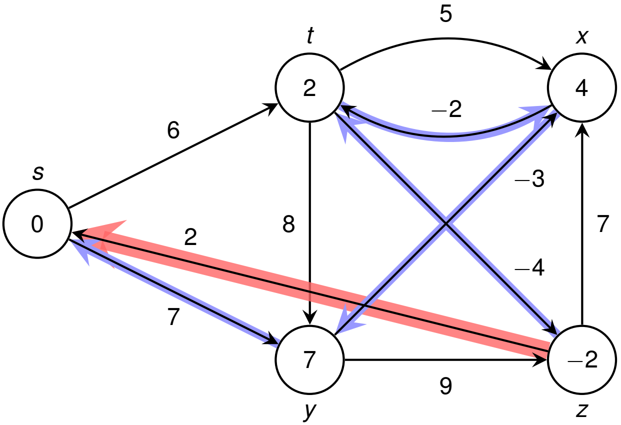
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

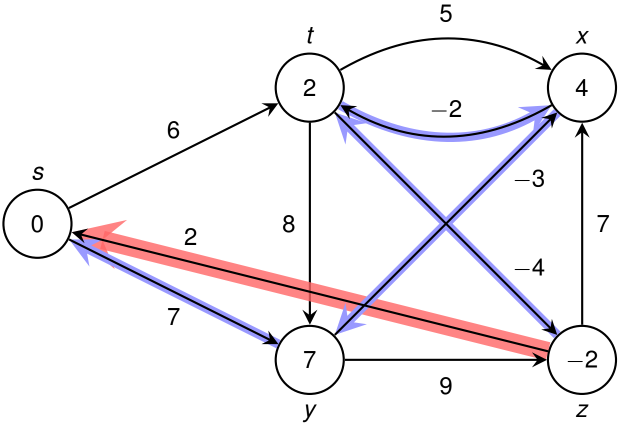
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

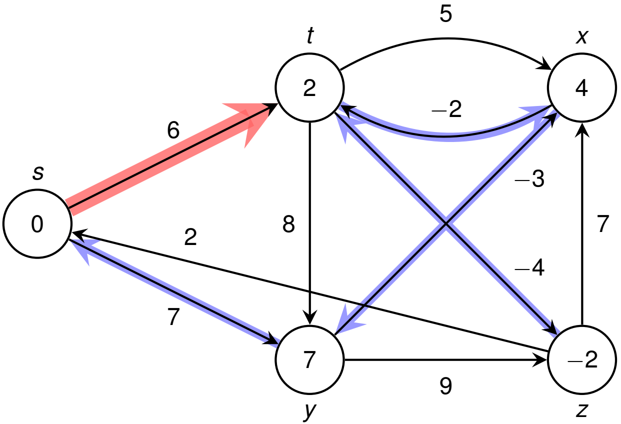
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

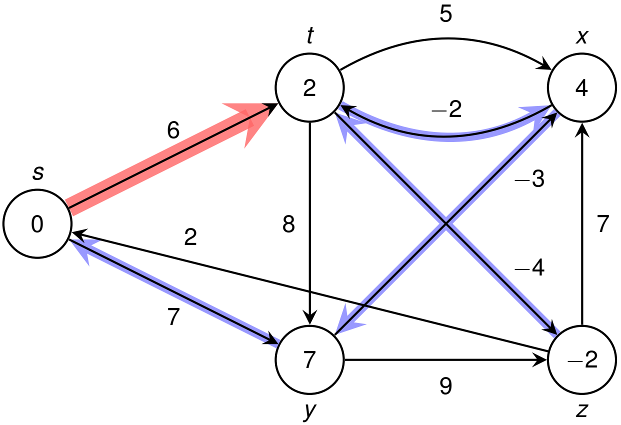
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

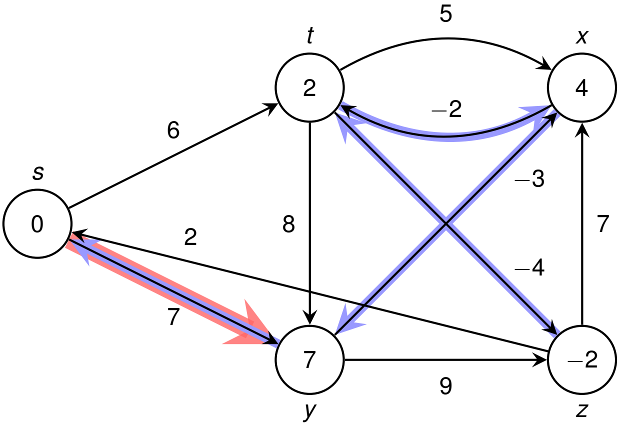
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

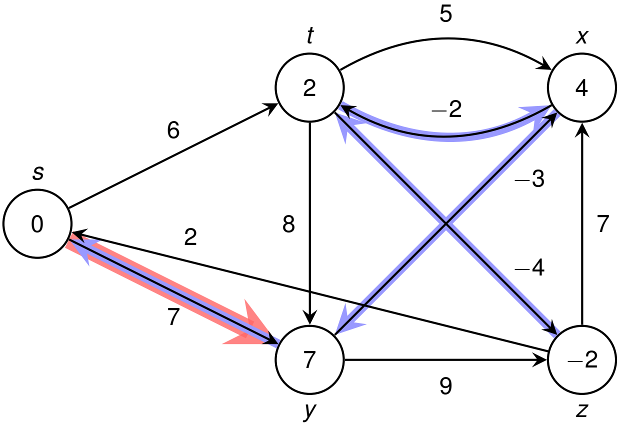
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

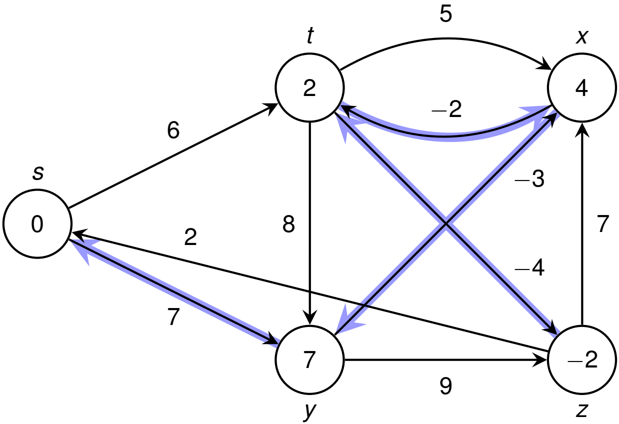
Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Execution of Bellman-Ford (Figure 24.4)

Pass: 4

Relaxation Order: (t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)



Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u, v)$ for all edges

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u, v)$ for all edges
- Let $(u, v) \in E$ be any edge. After $|V| - 1$ passes:

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V|-1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V|-1$
- Path-Relaxation Property $\Rightarrow$ after $|V|-1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u,v)$ for all edges
- Let $(u,v) \in E$ be any edge. After $|V|-1$ passes:

$$v.d = v.\delta$$

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V|-1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V|-1$
- Path-Relaxation Property $\Rightarrow$ after $|V|-1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u,v)$ for all edges
- Let $(u,v) \in E$ be any edge. After $|V|-1$ passes:

$$v.d = v.\delta \leq u.\delta + w(u,v)$$

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u, v)$ for all edges
- Let $(u, v) \in E$ be any edge. After $|V| - 1$ passes:

$$v.d = v.\delta \leq u.\delta + w(u, v) = u.d + w(u, v)$$

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u, v)$ for all edges
- Let $(u, v) \in E$ be any edge. After $|V| - 1$ passes:

$$v.d = v.\delta \leq u.\delta + w(u, v) = u.d + w(u, v) \qquad \square$$

Bellman-Ford Algorithm: Correctness (1/2)

Lemma 24.2/Theorem 24.3

Assume that G contains no negative-weight cycles that are reachable from s. Then after $|V| - 1$ passes, we have $v.d = v.\delta$ for all vertices $v \in V$ (that are reachable) and Bellman-Ford returns TRUE.

Proof that $v.d = v.\delta$

- Let v be a vertex reachable from s
- Let $p = (v_0 = s, v_1, \ldots, v_k = v)$ be a shortest path from s to v
- p is simple, hence $k \leq |V| - 1$
- Path-Relaxation Property $\Rightarrow$ after $|V| - 1$ passes, $v.d = v.\delta$

Proof that Bellman-Ford returns TRUE

- Need to prove: $v.d \leq u.d + w(u, v)$ for all edges
- Let $(u, v) \in E$ be any edge. After $|V| - 1$ passes:

$$v.d = v.\delta \leq u.\delta + w(u, v) = u.d + w(u, v) \qquad \square$$

Triangle inequality (holds even if $w(u, v) < 0$!)

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Proof:

- Let $c = (v_0, v_1, \ldots, v_k = v_0)$ be a negative-weight cycle reachable from s

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Proof:

- Let $c = (v_0, v_1, \ldots, v_k = v_0)$ be a negative-weight cycle reachable from s
- If Bellman-Ford returns TRUE, then for every $1 \leq i < k$,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$$

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Proof:

- Let $c = (v_0, v_1, \ldots, v_k = v_0)$ be a negative-weight cycle reachable from s
- If Bellman-Ford returns TRUE, then for every $1 \leq i < k$,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$$

$$\Rightarrow \quad \sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Proof:

- Let $c = (v_0, v_1, \ldots, v_k = v_0)$ be a negative-weight cycle reachable from s
- If Bellman-Ford returns TRUE, then for every $1 \leq i < k$,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$$

$$\Rightarrow \quad \sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\Rightarrow \quad 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Proof:

- Let $c = (v_0, v_1, \ldots, v_k = v_0)$ be a negative-weight cycle reachable from s
- If Bellman-Ford returns TRUE, then for every $1 \leq i < k$,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$$

$$\Rightarrow \quad \sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\Rightarrow \quad 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

This cancellation is only valid if all $.d$-values are finite!

Bellman-Ford Algorithm: Correctness (2/2)

Theorem 24.3

If G contains a negative-weight cycle reachable from s, then Bellman-Ford returns FALSE.

Proof:

- Let $c = (v_0, v_1, \ldots, v_k = v_0)$ be a negative-weight cycle reachable from s
- If Bellman-Ford returns TRUE, then for every $1 \leq i < k$,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$$

$$\Rightarrow \quad \sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\Rightarrow \quad 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

This cancellation is only valid if all $.d$-values are finite!

- This contradicts the assumption that c is a negative-weight cycle! □

The Bellman-Ford Algorithm

```
BELLMAN-FORD(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0

repeat |V|-1 times
   for e in G.edges()
   Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
      if e.start.d + e.weight.d < e.end.d:
         e.end.d = e.start.d + e.weight
         e.end.predecessor = e.start

for e in G.edges()
   if e.start.d + e.weight.d < e.end.d:
      return FALSE
return TRUE
```

The Bellman-Ford Algorithm

```
BELLMAN-FORD(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0
repeat |V|-1 times
   for e in G.edges()
   Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
      if e.start.d + e.weight.d < e.end.d:
         e.end.d = e.start.d + e.weight
         e.end.predecessor = e.start
for e in G.edges()
   if e.start.d + e.weight.d < e.end.d:
      return FALSE
return TRUE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

The Bellman-Ford Algorithm

```
BELLMAN-FORD(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0

repeat |V|-1 times
   for e in G.edges()
   Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
      if e.start.d + e.weight.d < e.end.d:
         e.end.d = e.start.d + e.weight
         e.end.predecessor = e.start

for e in G.edges()
   if e.start.d + e.weight.d < e.end.d:
      return FALSE
return TRUE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

Yes, because if pass i keeps all *.d* variables, then so does pass $i + 1$.

The Bellman-Ford Algorithm (modified)

```
BELLMAN-FORD-NEW(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0

repeat |V| times
   flag = 0
   for e in G.edges()
   Relax edge e=(u,v):  Check if u.d + w(u,v) < v.d
       if e.start.d + e.weight.d < e.end.d:
          e.end.d = e.start.d + e.weight
          e.end.predecessor = e.start
          flag = 1
   if flag = 0 return TRUE

return FALSE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

Yes, because if pass i keeps all *.d* variables, then so does pass $i + 1$.

The Bellman-Ford Algorithm (modified)

```
 BELLMAN-FORD-NEW(G,w,s)
assert(s in G.vertices())
for v in G.vertices()
   v.predecessor = None
   v.d = Infinity
s.d = 0
repeat |V| times
   flag = 0
   for e in G.edges()
   Relax edge e=(u,v): Check if u.d + w(u,v) < v.d
       if e.start.d + e.weight.d < e.end.d:
          e.end.d = e.start.d + e.weight
          e.end.predecessor = e.start
          flag = 1
   if flag = 0 return TRUE
return FALSE
```

Can we terminate earlier if there is a pass that keeps all *.d* variables?

Yes, because if pass i keeps all *.d* variables, then so does pass $i + 1$.