Data Structures & Algorithms

18: Single Source Shortest Path

(Bellman-Ford & DAG Shortest Path)

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

Dijkstra's Algorithm

Dijkstra's Algorithm

Dijkstra's algorithm solves single source shortest path on weighted directed graph G(V,E).

It is a generalization of Breadth First Search (BFS) on weighted graphs. The weights here are non-negative.

The source vertex **s** is explicitly provided.
Find shortest path from **s** to all remaining vertices.

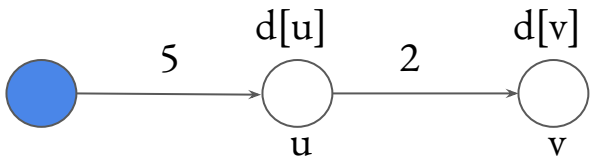
Dijkstra's Algorithm

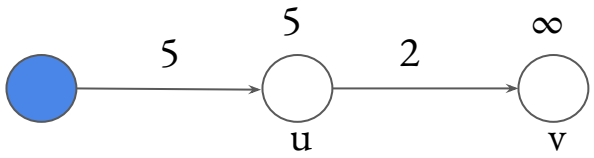
```
if (d[u] + w(u,v) < d[v])
   d[v] = d[u] + w(u,v)
```

```
d[u] = 5
w(u,v) = 2
d[v] = ∞
```

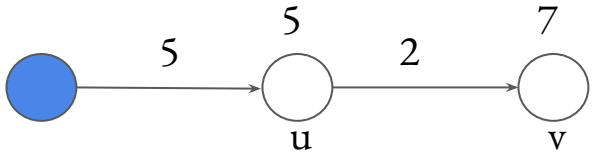
5 + 2 < ∞ is true

d[v] = 5 + 2 = 7



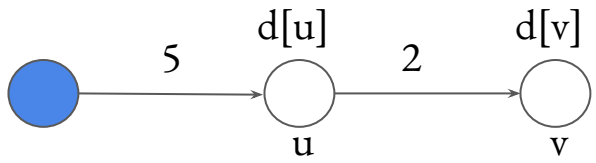


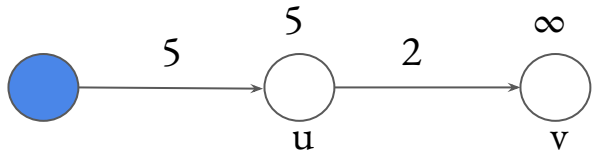
no direct path



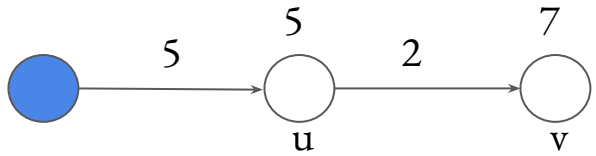
Dijkstra's Algorithm

```
if (d[u] + w(u,v) < d[v])
   d[v] = d[u] + w(u,v)
```





no direct path



$\textsc{Relax}(u, v, w)$

if $v.d > u.d + w(u, v)$

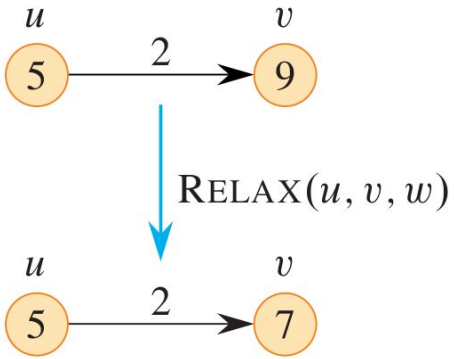
$v.d = u.d + w(u, v)$

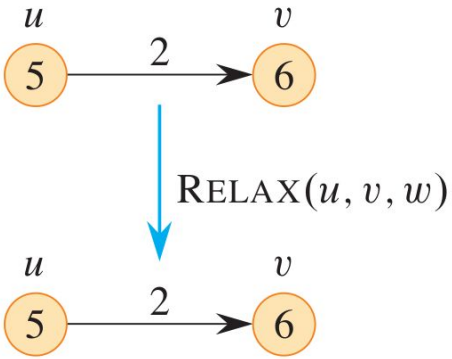
$v.\pi = u$

Dijkstra's Algorithm

```
if (d[u] + w(u,v) < d[v])
   d[v] = d[u] + w(u,v)
```

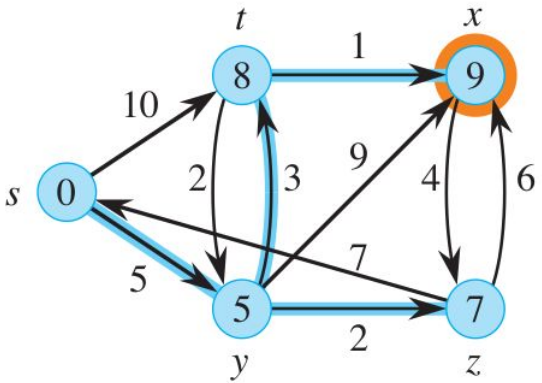
$$\textsc{Relax}(u, v, w)$$
$$\textbf{if } v.d > u.d + w(u, v)$$
$$v.d = u.d + w(u, v)$$
$$v.\pi = u$$





Dijkstra's Algorithm

Selected vertex	Visited set	d[t]	d[x]	d[y]	d[z]
s	{s}	10	∞	(5)	∞
y	{s, y}	8	14	(5)	(7)
z	{s, y, z}	(8)	13	(5)	(7)
t	{s, y, z, t}	(8)	(9)	(5)	(7)
x	{s, y, z, t, x}	(8)	(9)	(5)	(7)



Dijkstra's Algorithm

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = ∅
  for each vertex u ∈ G.V
      INSERT(Q, u)
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v in G.Adj[u]
          RELAX(u, v, w)
          if the call of RELAX decreased v.d
              DECREASE-KEY(Q, v, v.d)
```

Dijkstra's Algorithm

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in G.V$

$v.d = \infty$

$v.\pi = \text{NIL}$

$s.d = 0$

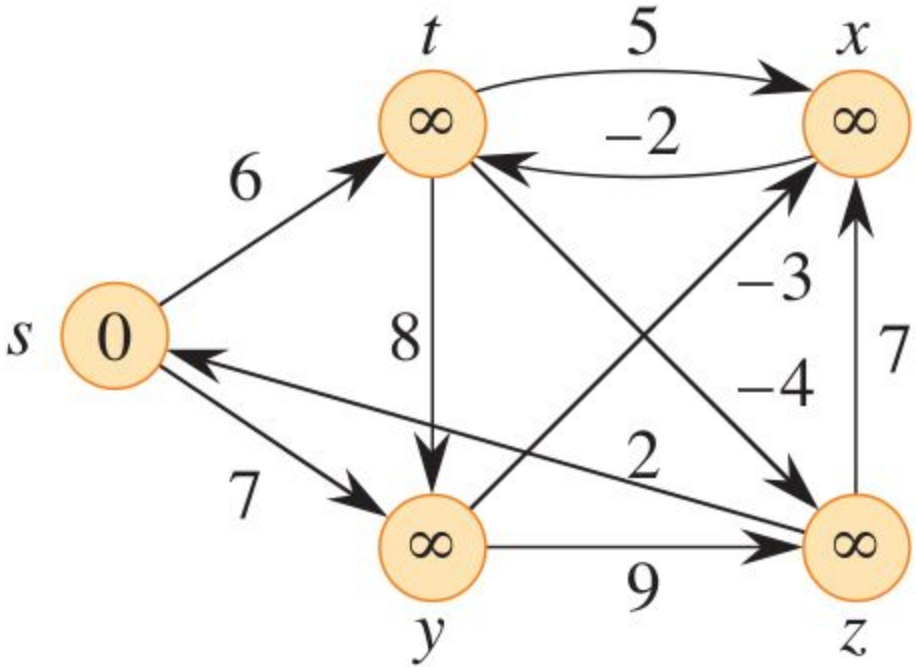
Bellman-Ford Algorithm

Bellman-Ford Algorithm

Bellman-Ford

- Allows negative-weight edges.
- Computes $v.d$ and $v.\pi$ for all $v \in V$.
- Returns TRUE if no negative-weight cycles reachable from s, FALSE otherwise.

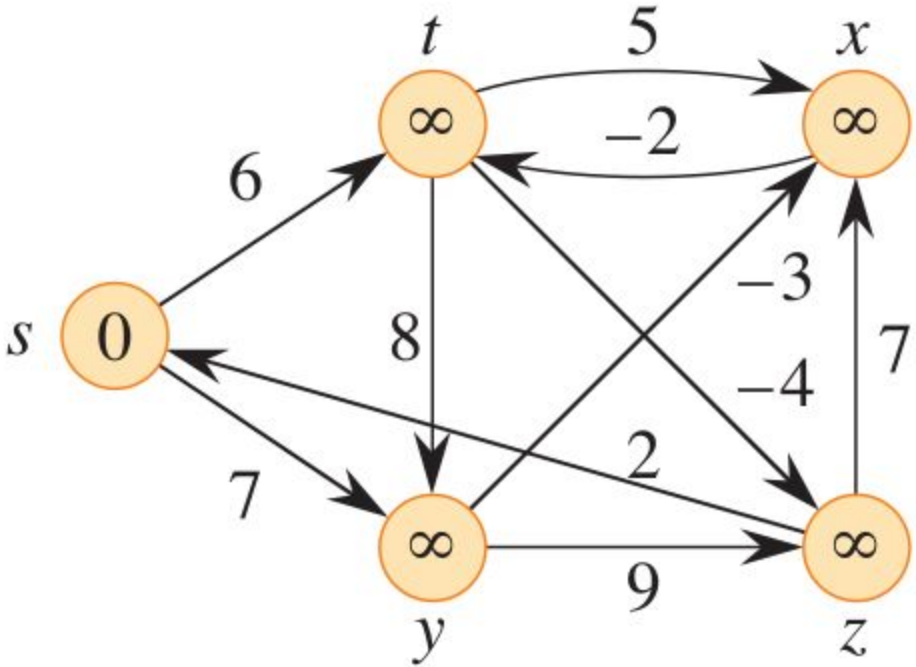
Bellman-Ford Algorithm



Bellman-Ford Algorithm

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

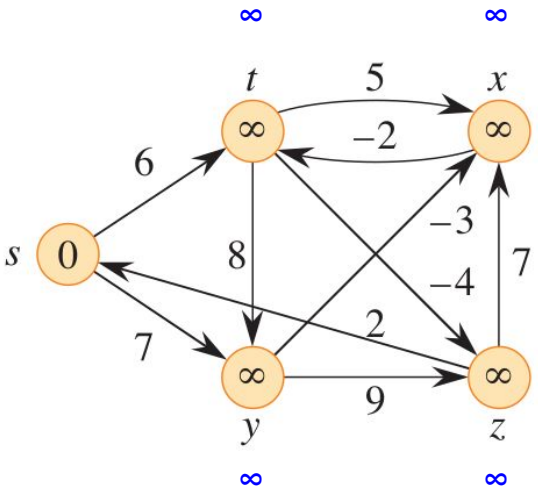
Bellman-Ford Algorithm



Iteration - 1

Bellman-Ford Algorithm

Iteration - 1

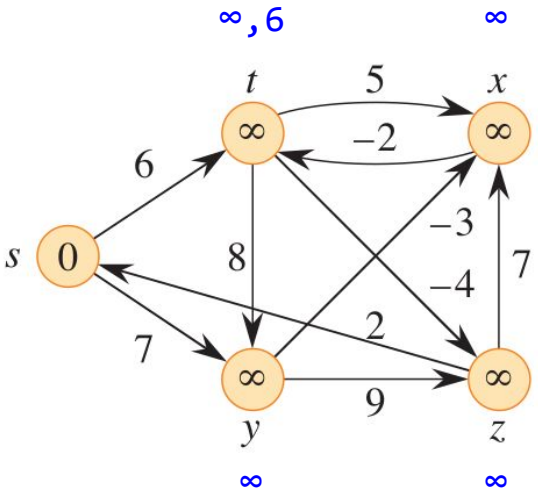



```
s - t
s - y
t - x
t - y
t - z
```

```
x - t
y - x
y - z
z - x
z - s
```

Bellman-Ford Algorithm

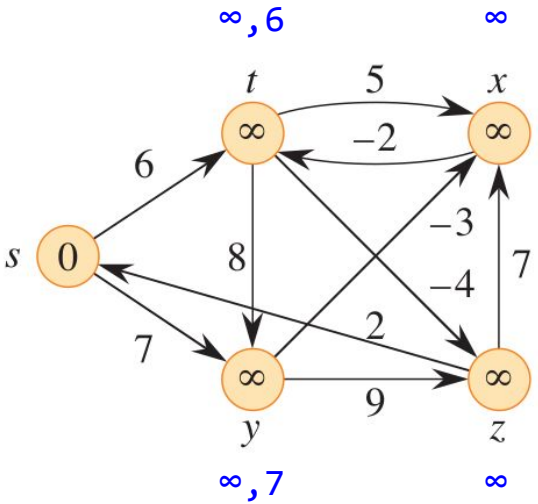
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

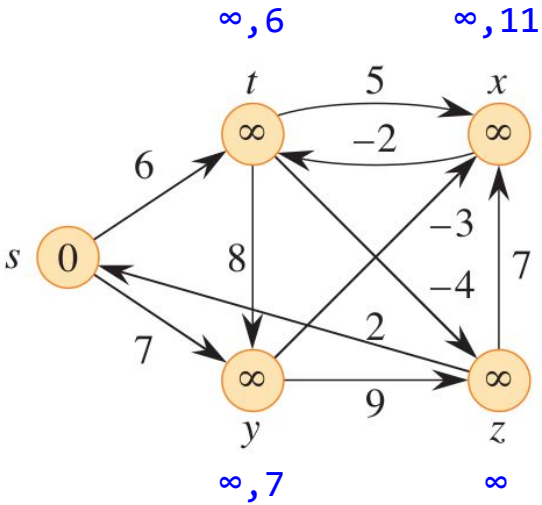
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

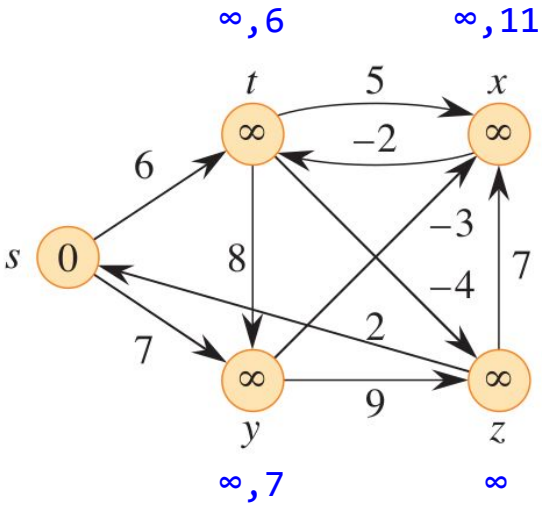
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

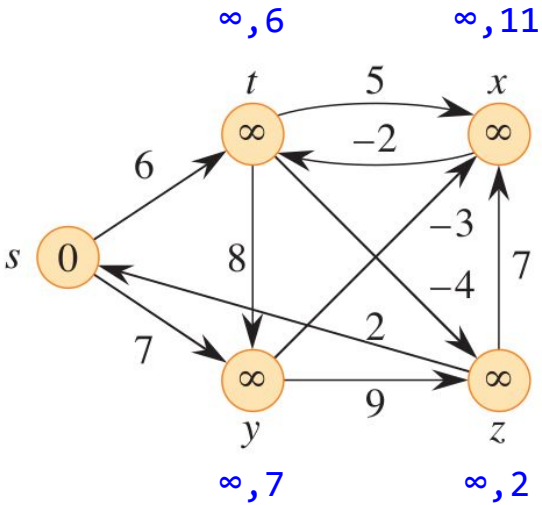
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

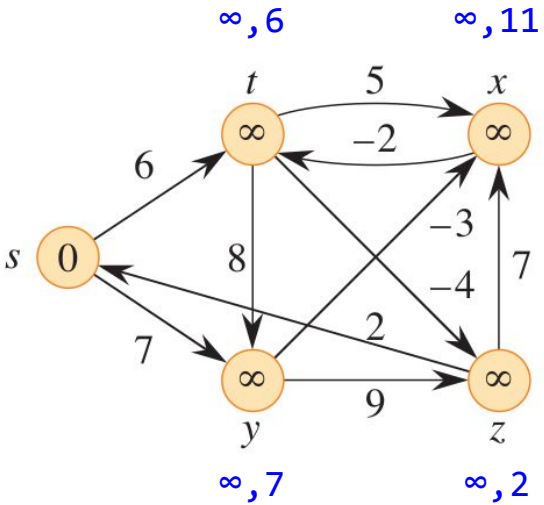
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

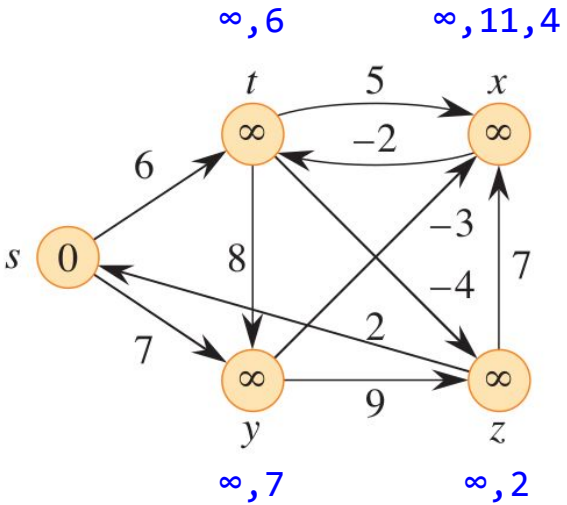
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

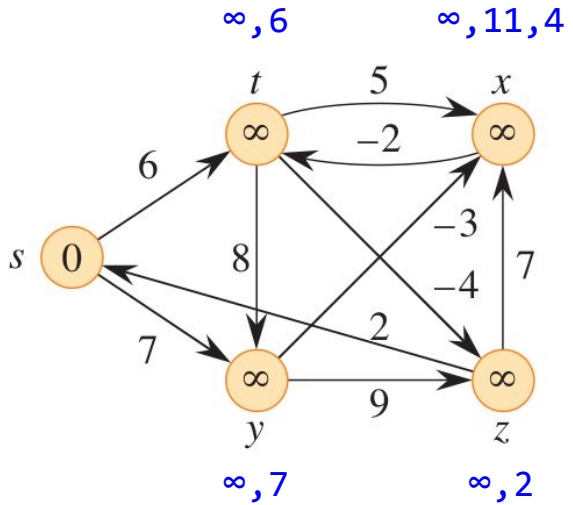
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

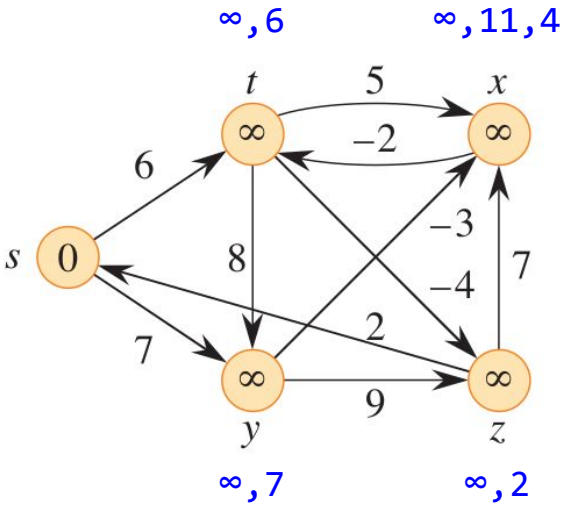
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

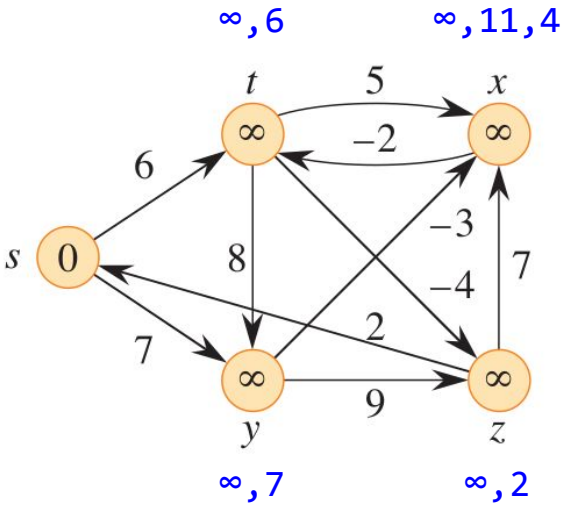
Iteration - 1



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

Iteration - 1

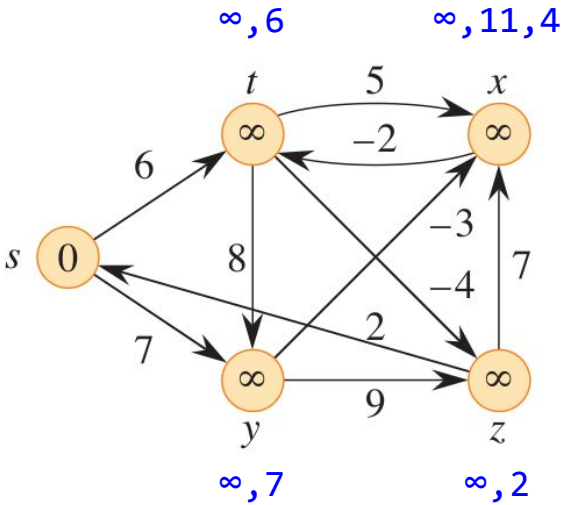


s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Iteration - 2

Bellman-Ford Algorithm

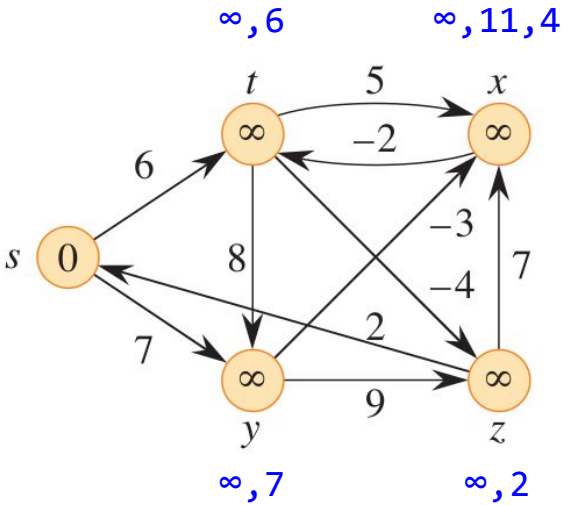
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

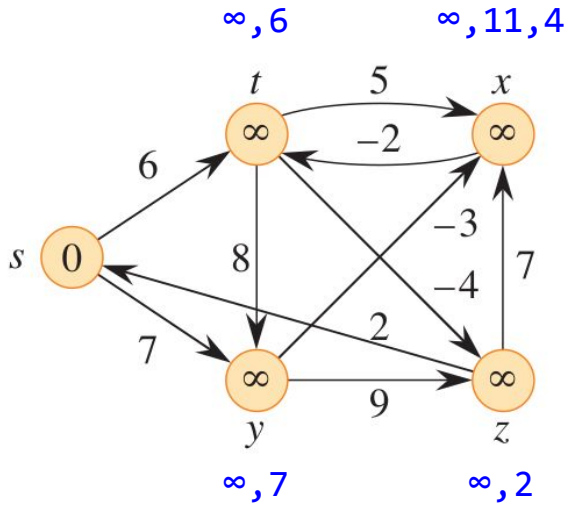
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

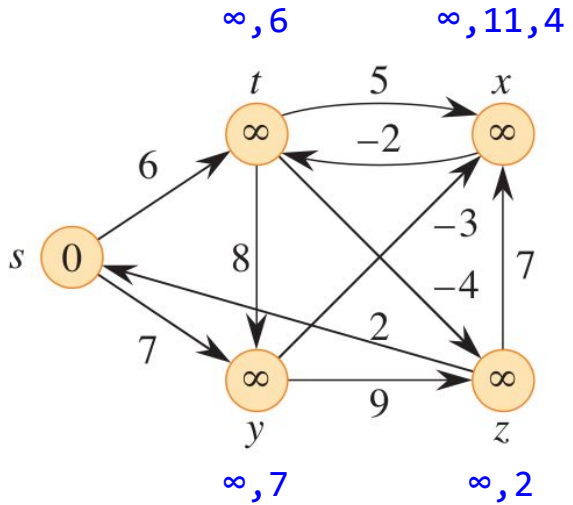
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

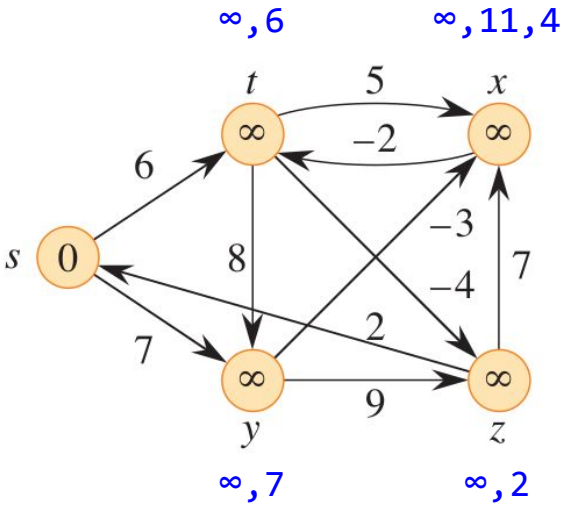
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

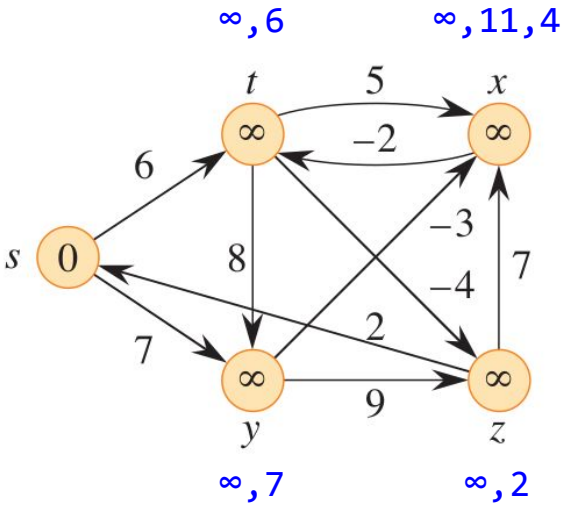
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

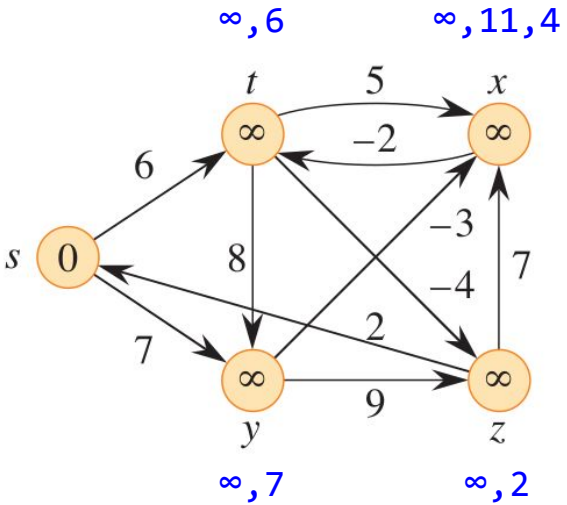
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

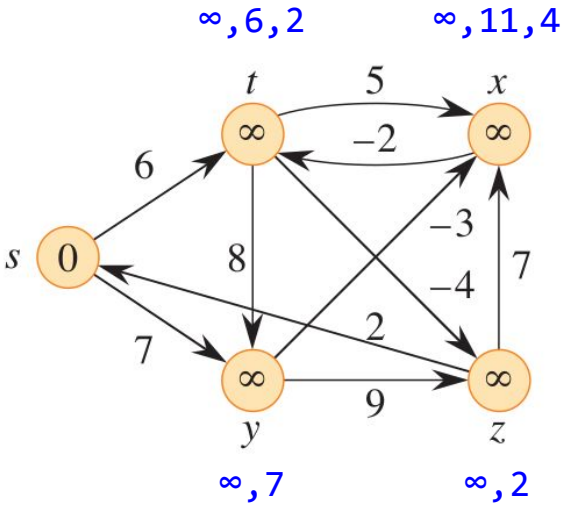
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

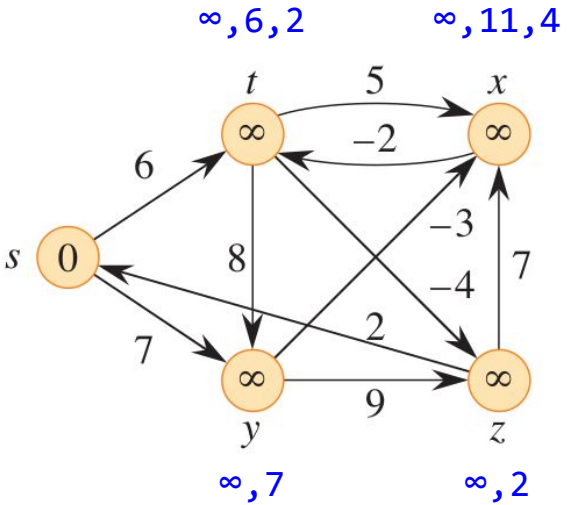
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

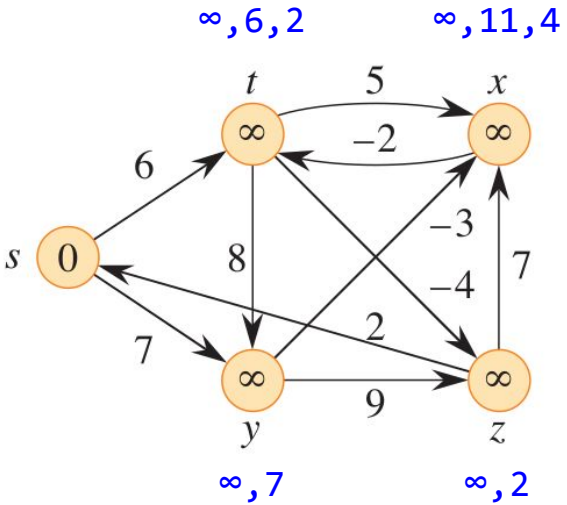
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

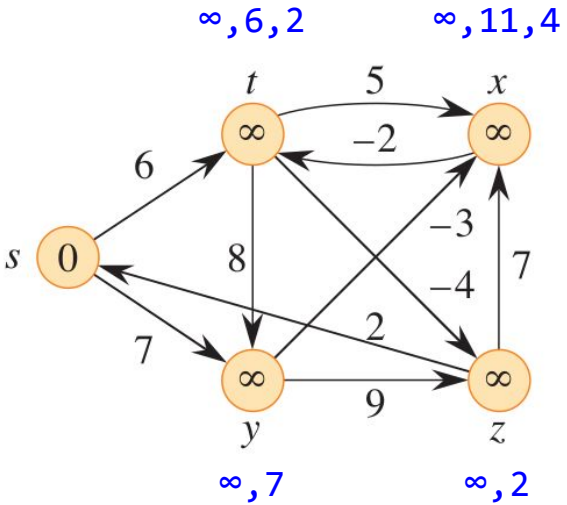
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

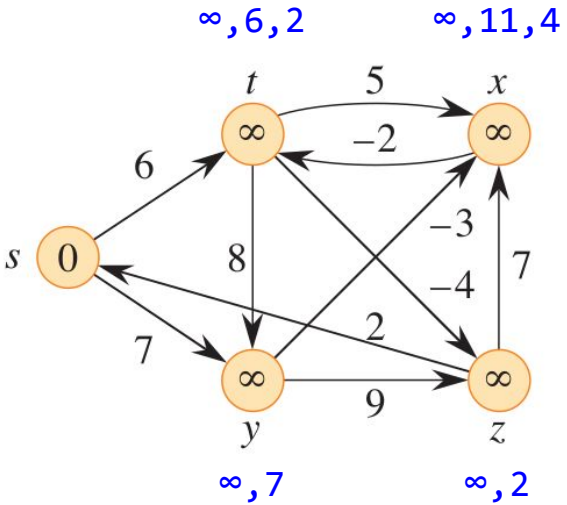
Iteration - 2



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

Iteration - 2

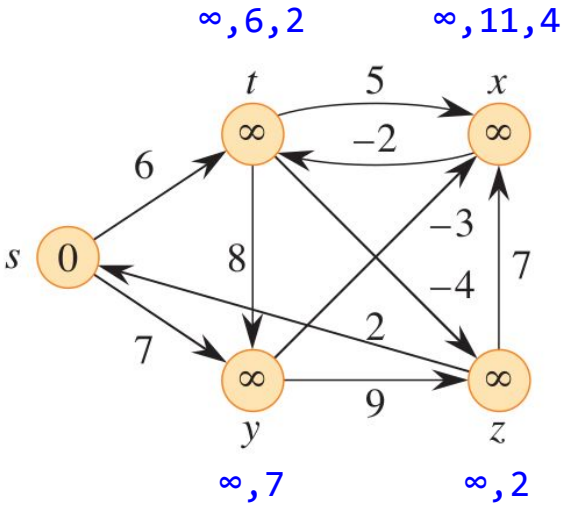


s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Iteration - 3

Bellman-Ford Algorithm

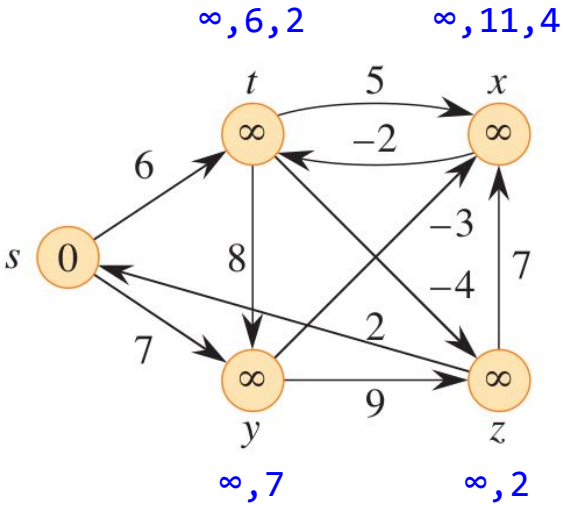
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

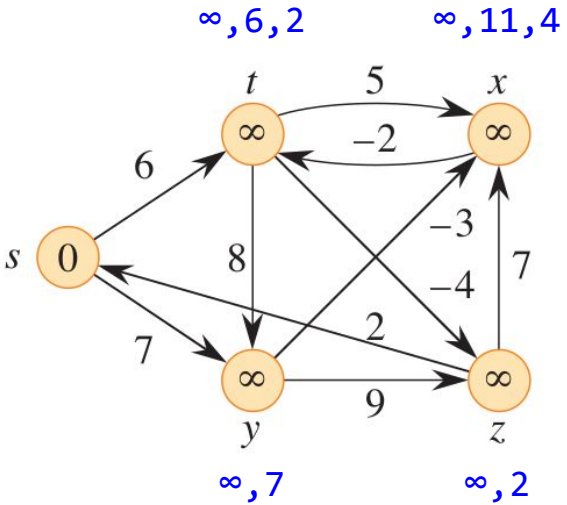
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

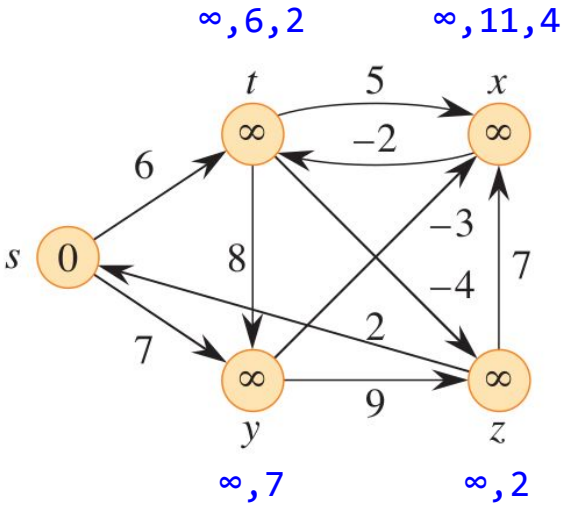
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

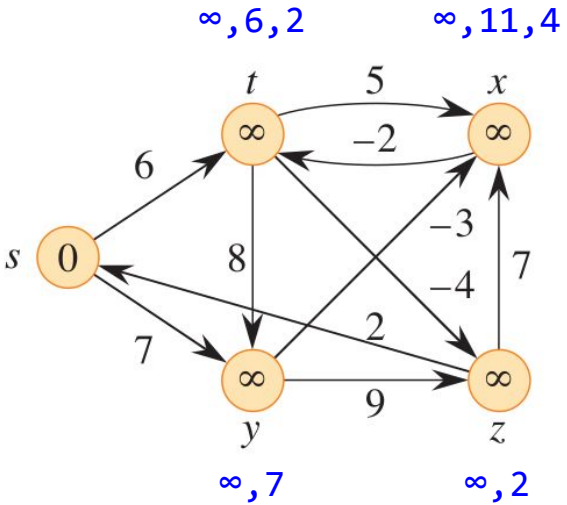
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

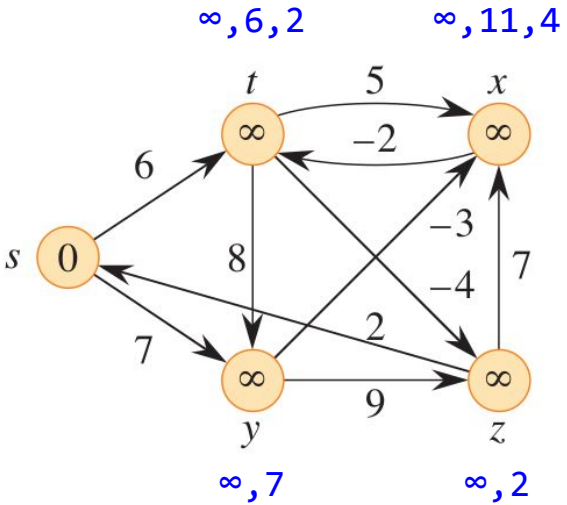
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

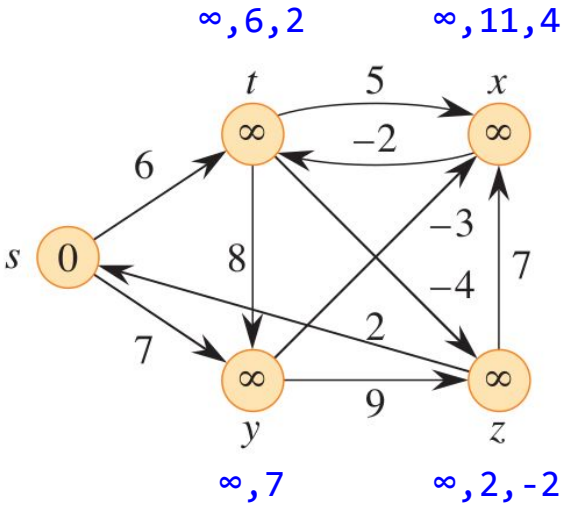
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

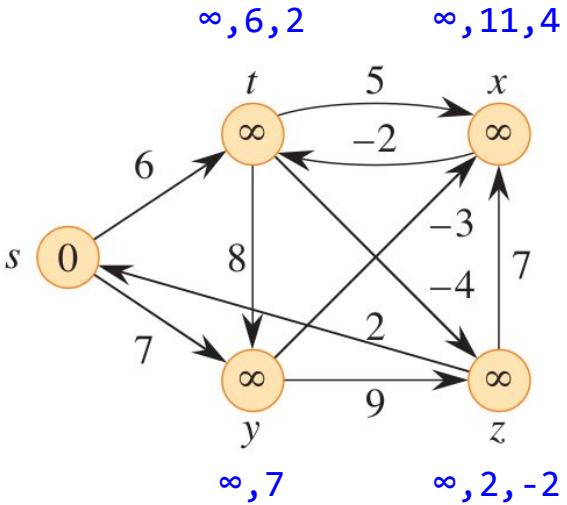
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

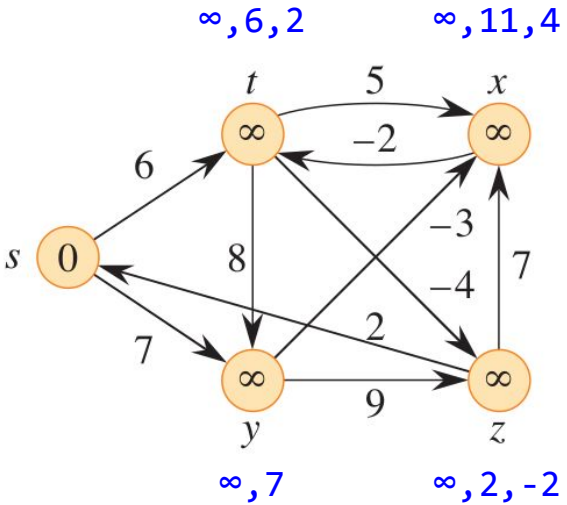
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

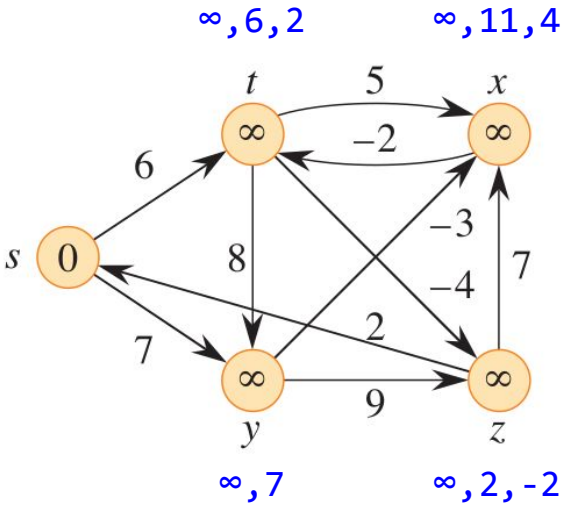
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

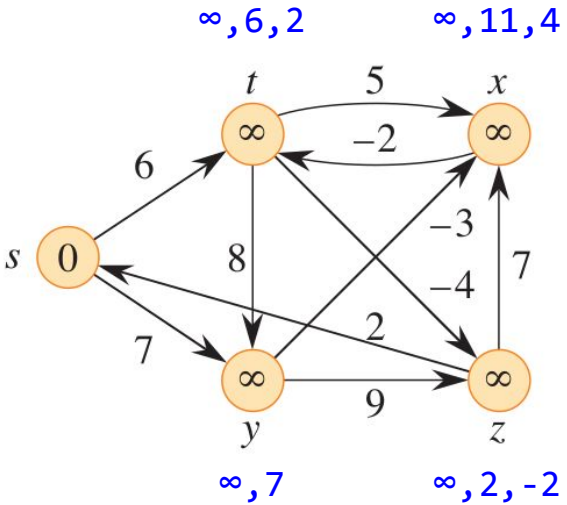
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

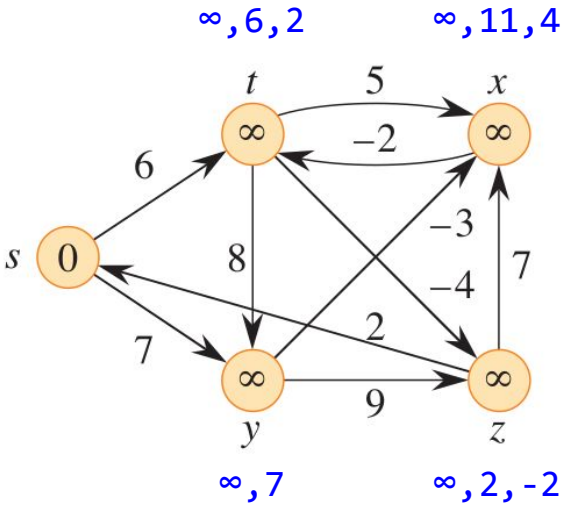
Iteration - 3



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

Iteration - 3

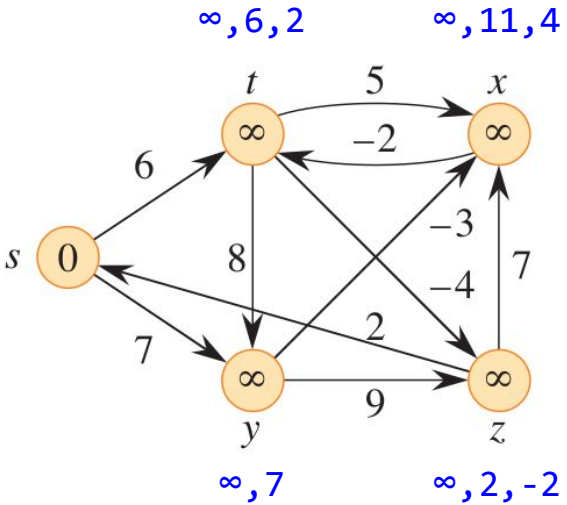


s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Iteration - 4

Bellman-Ford Algorithm

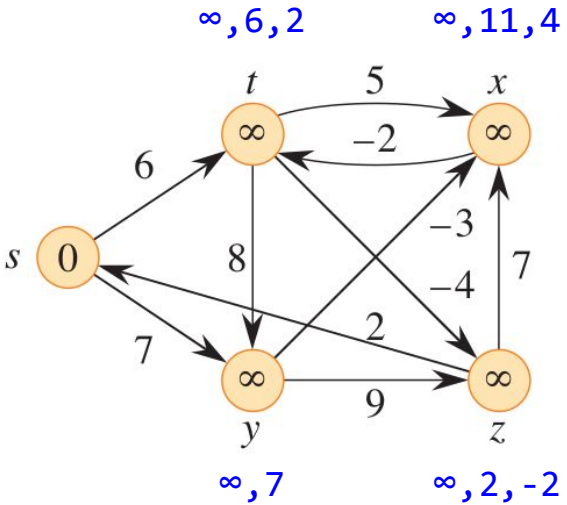
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

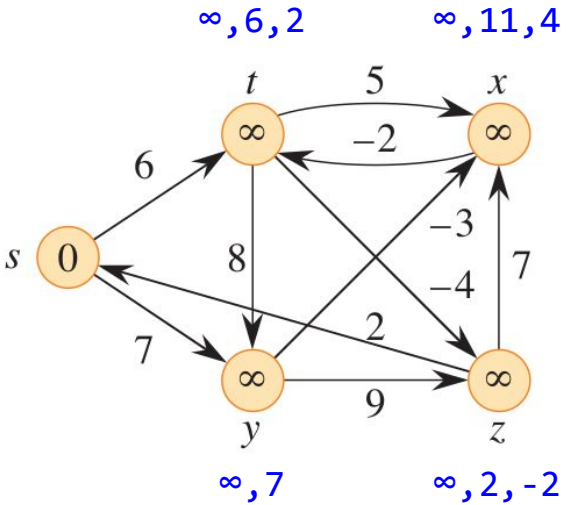
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

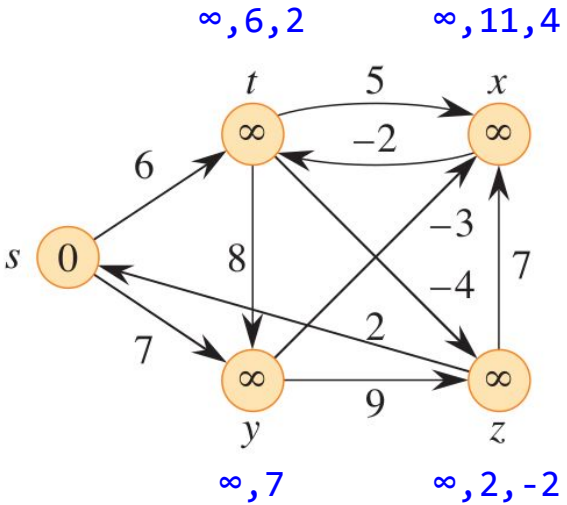
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

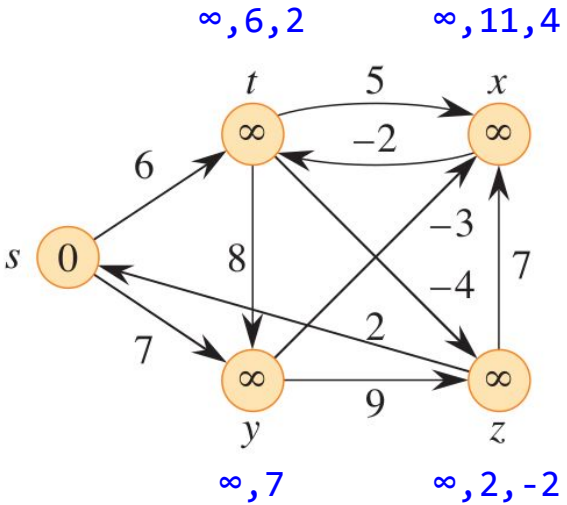
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

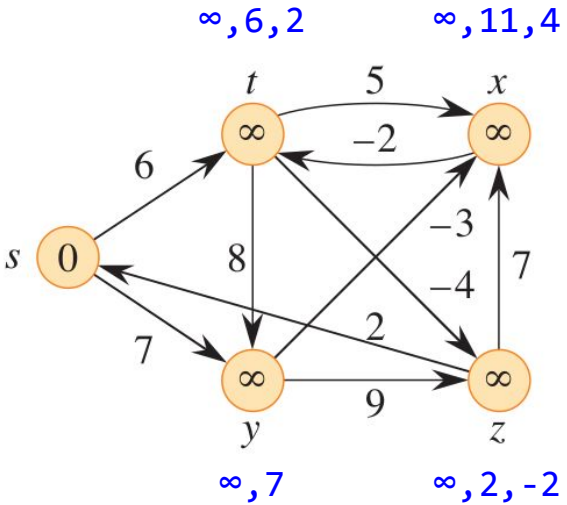
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

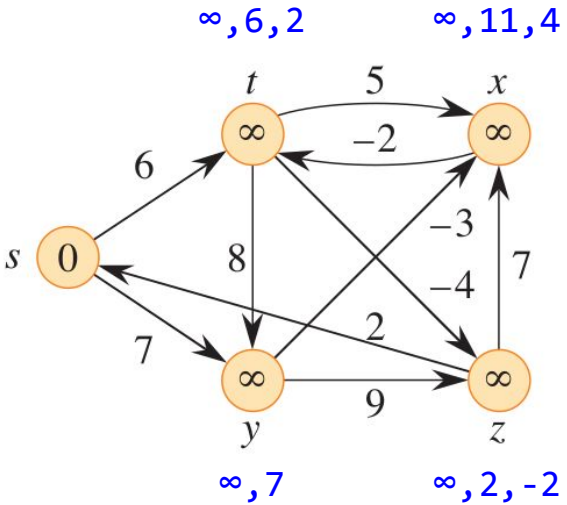
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

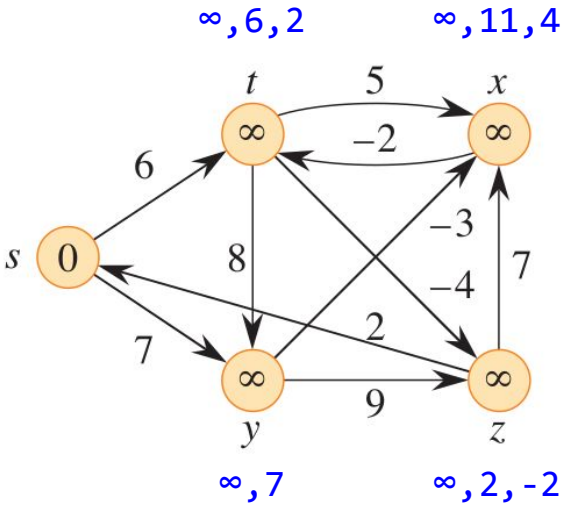
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

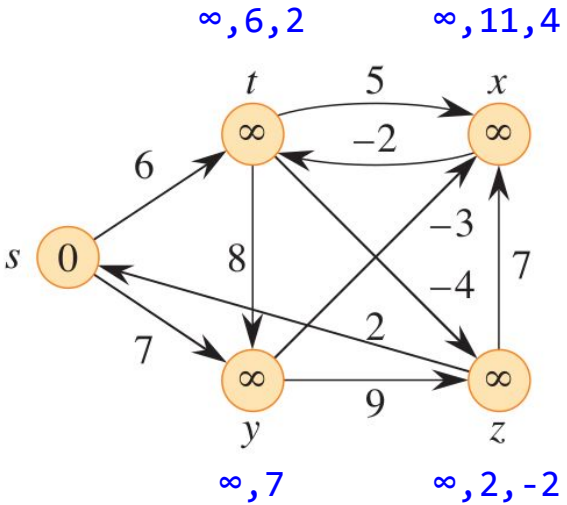
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

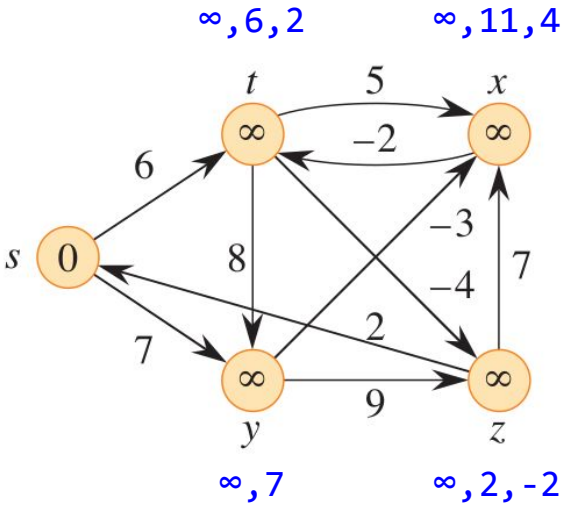
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

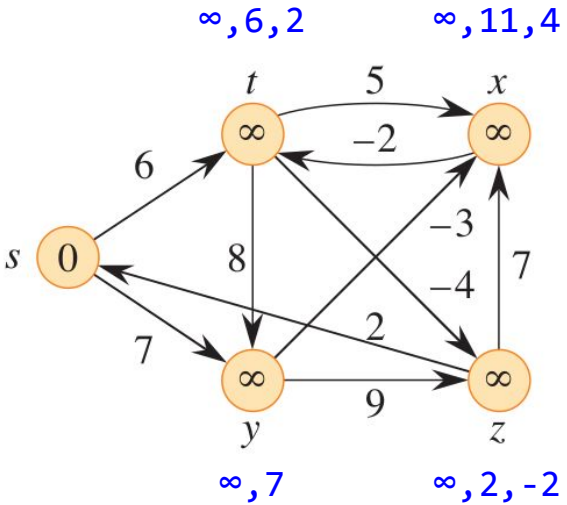
Iteration - 4



s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

Bellman-Ford Algorithm

Iteration - 4

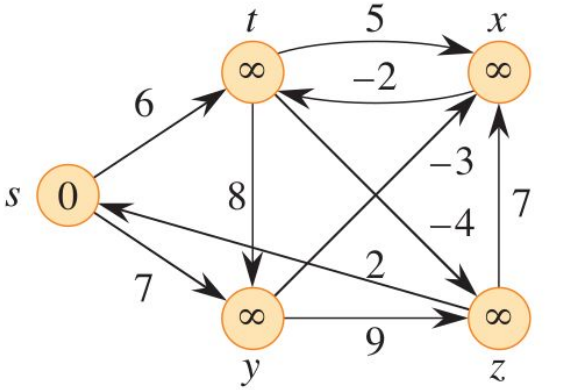


s - t	x - t
s - y	y - x
t - x	y - z
t - y	z - x
t - z	z - s

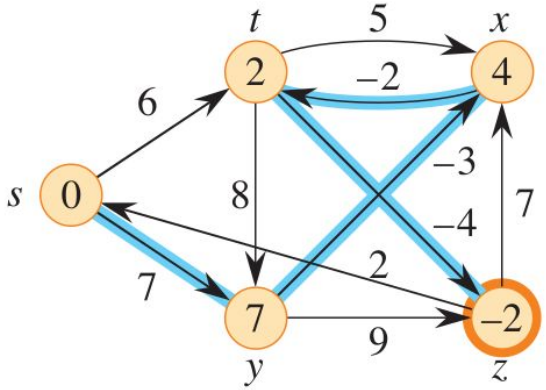
Bellman-Ford Algorithm

Iteration - 4

∞,6,2 ∞,11,4



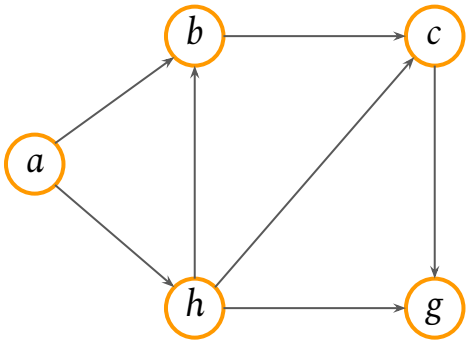
∞,7 ∞,2,-2



Topological Sorting

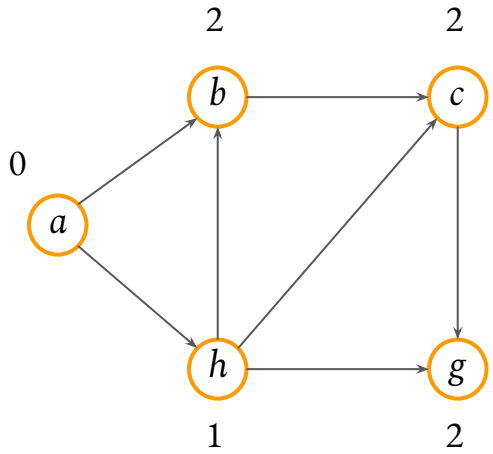
Topological Sorting

Arrange the vertices based on in-degree update.



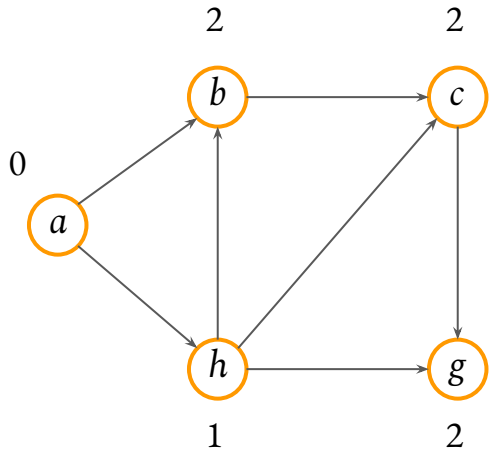
Topological Sorting

Arrange the vertices based on in-degree update.



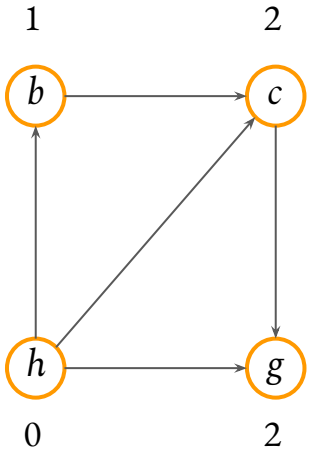
Topological Sorting

Arrange the vertices based on in-degree update.



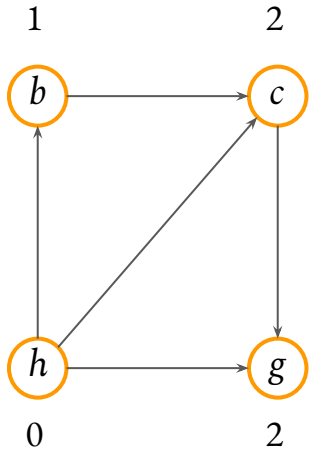
Topological Sorting

Arrange the vertices based on in-degree update.



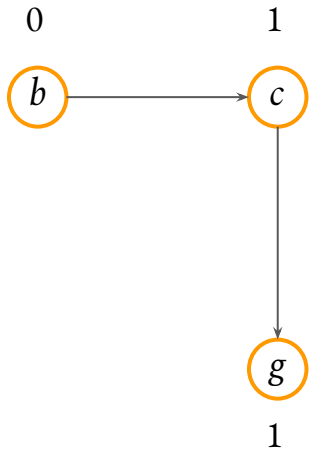
Topological Sorting

Arrange the vertices based on in-degree update.



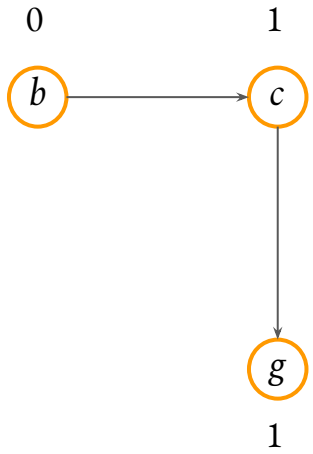
Topological Sorting

Arrange the vertices based on in-degree update.



Topological Sorting

Arrange the vertices based on in-degree update.



Topological Sorting

Arrange the vertices based on in-degree update.

Topological Sorting

Arrange the vertices based on in-degree update.

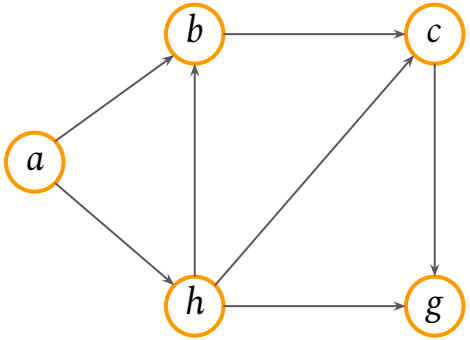
Topological Sorting

Arrange the vertices based on in-degree update.

0

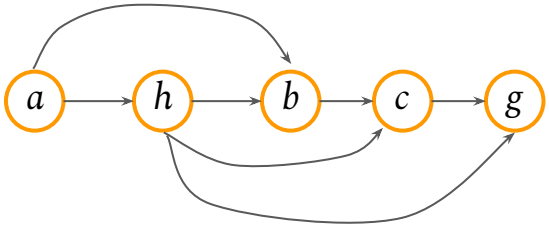
Topological Sorting

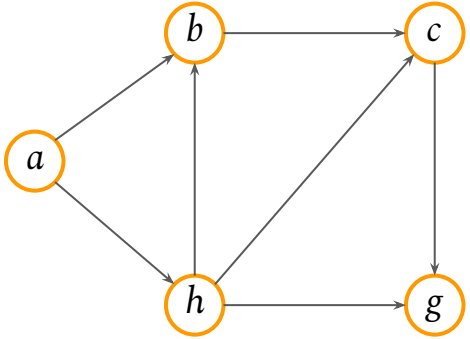
Arrange the vertices based on in-degree update.



Topological Sorting

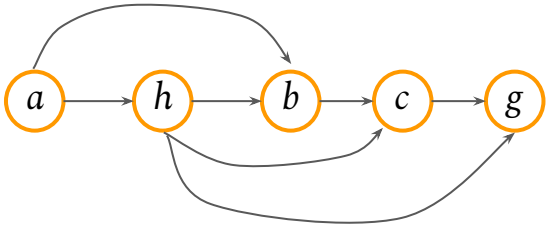
Arrange the vertices based on in-degree update.

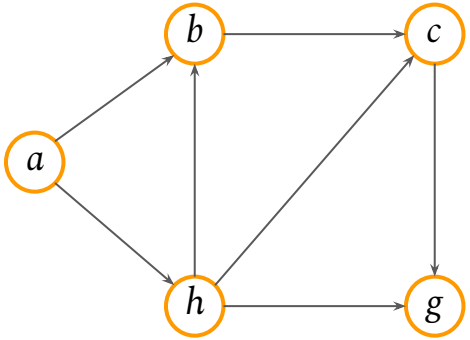




Topological Sorting

Arrange the vertices based on in-degree update.





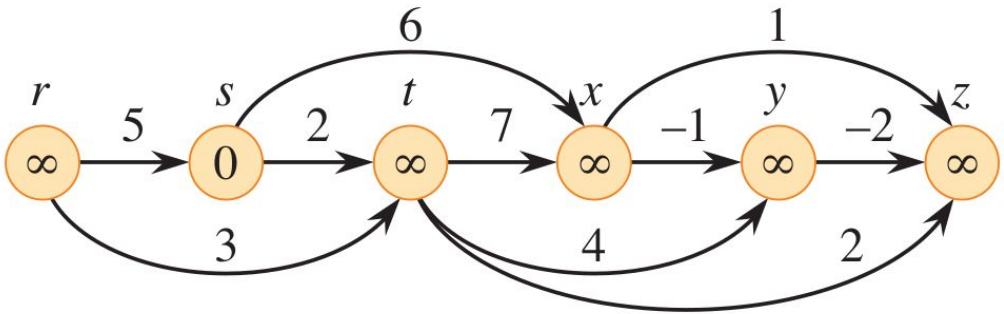
Can be implemented using DFS based on the finish time.

DAG Shortest Paths

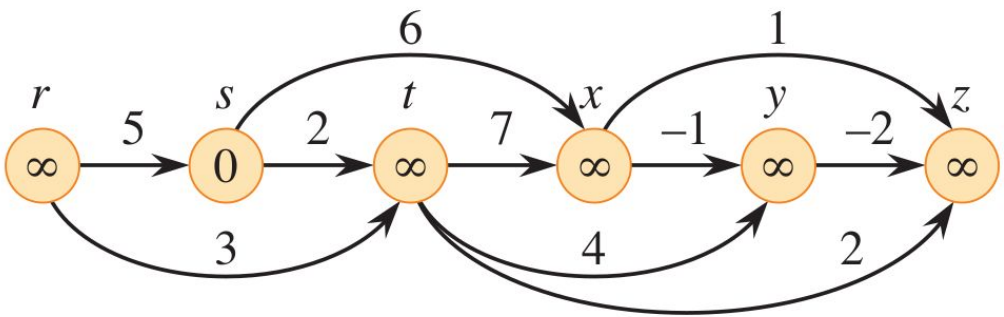
DAG Shortest Paths

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

DAG Shortest Paths

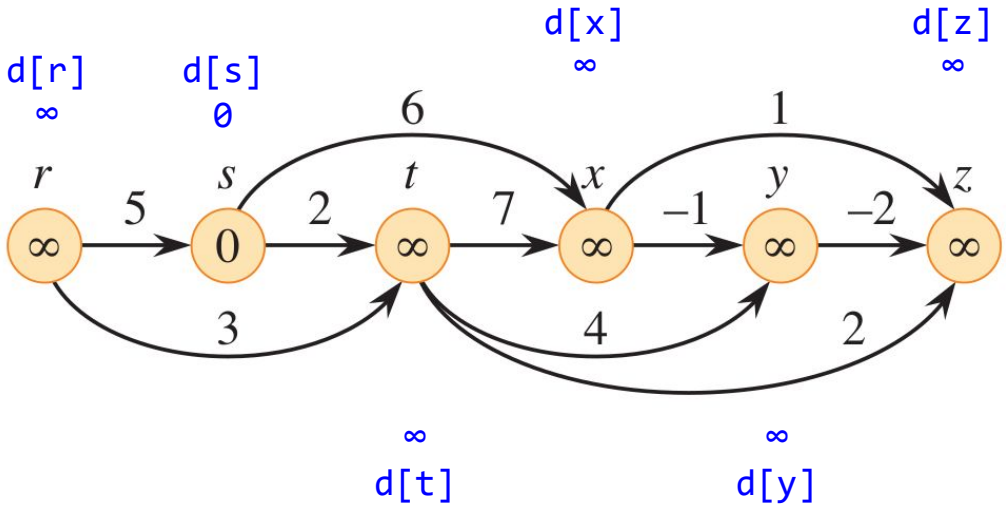


DAG Shortest Paths



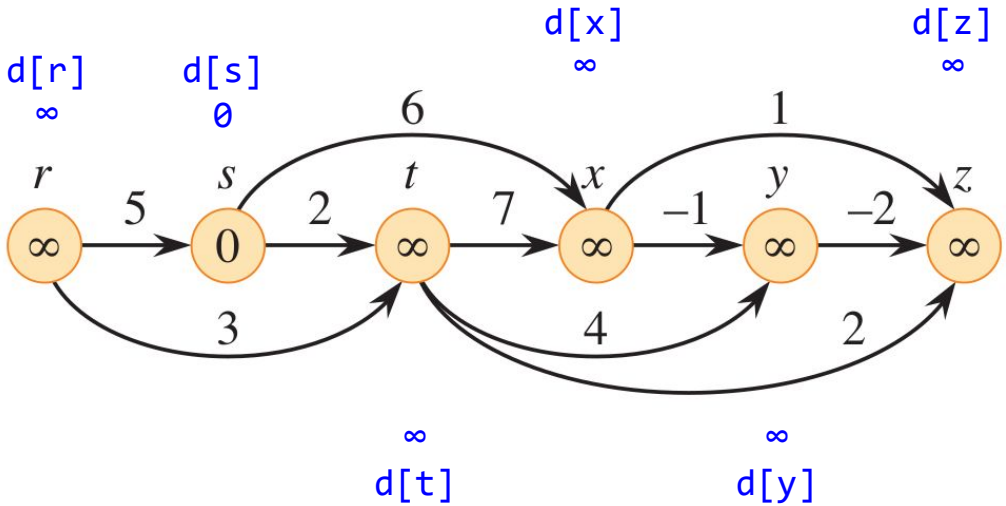

```
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}
```

DAG Shortest Paths



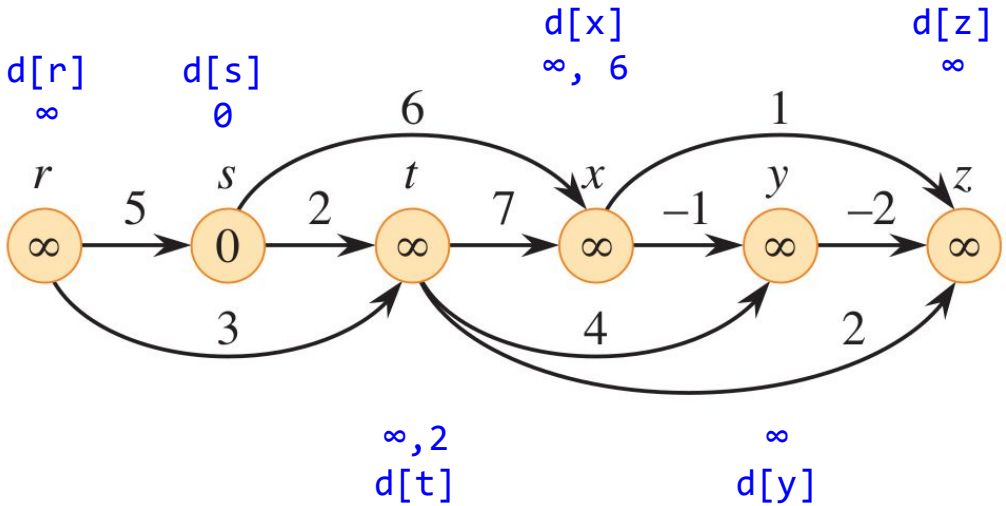
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



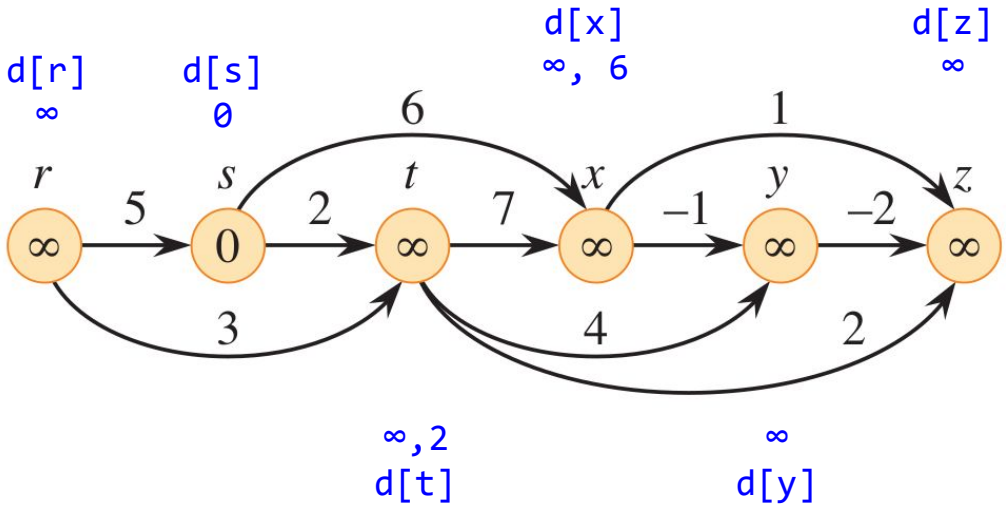
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



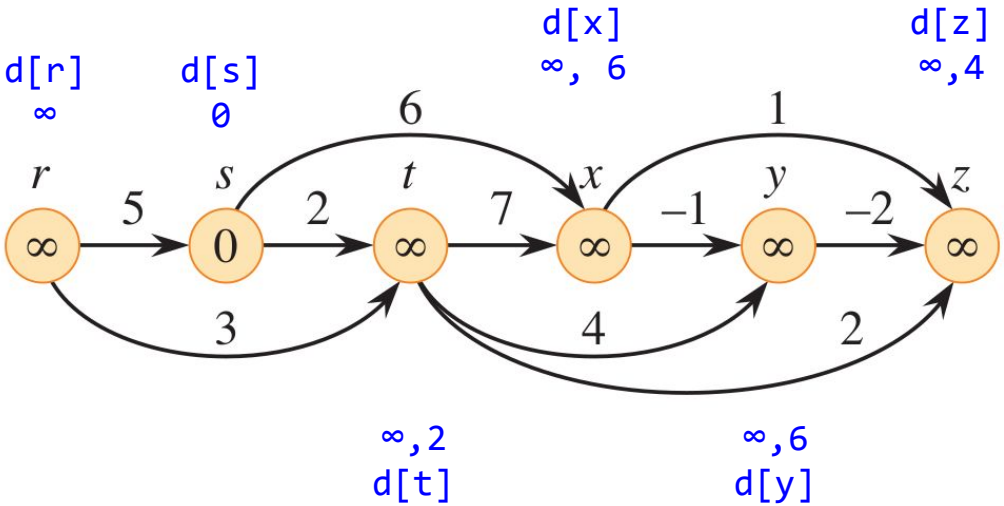
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



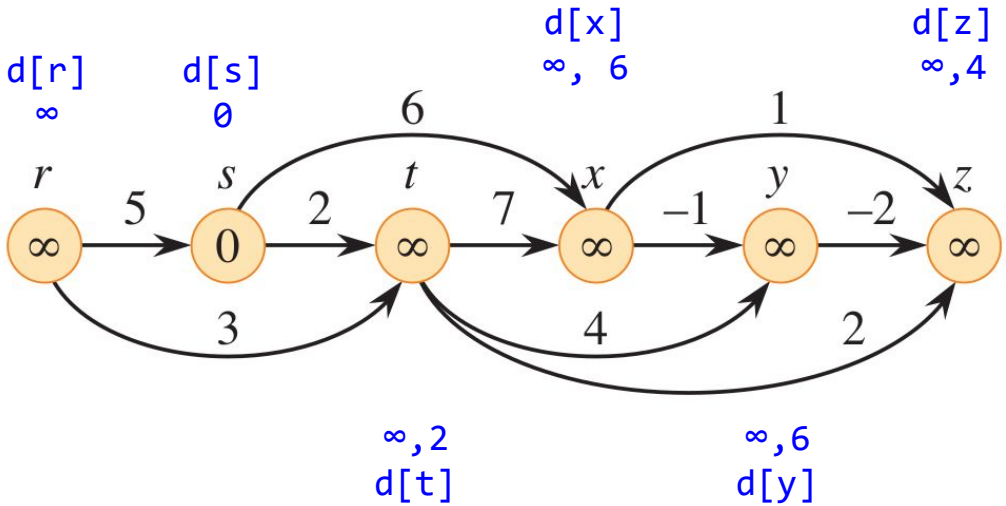
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



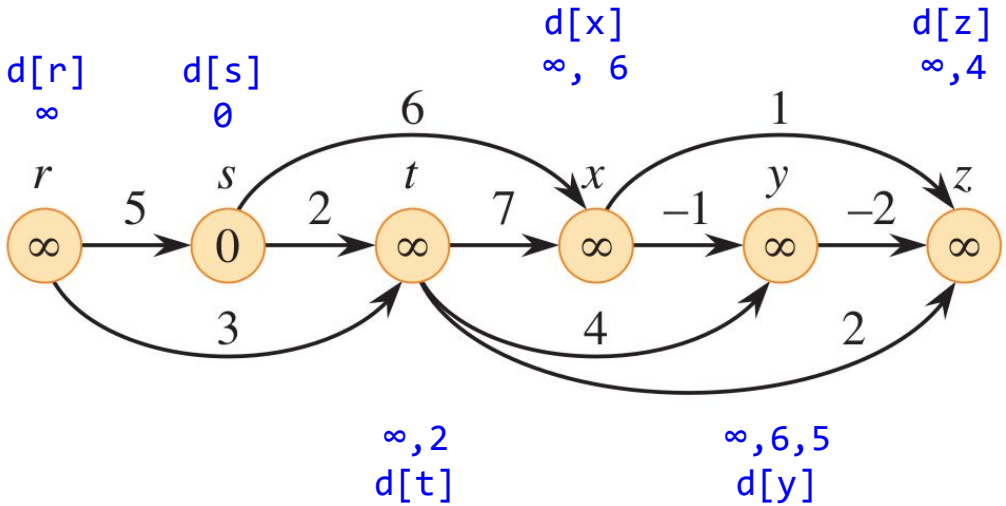
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



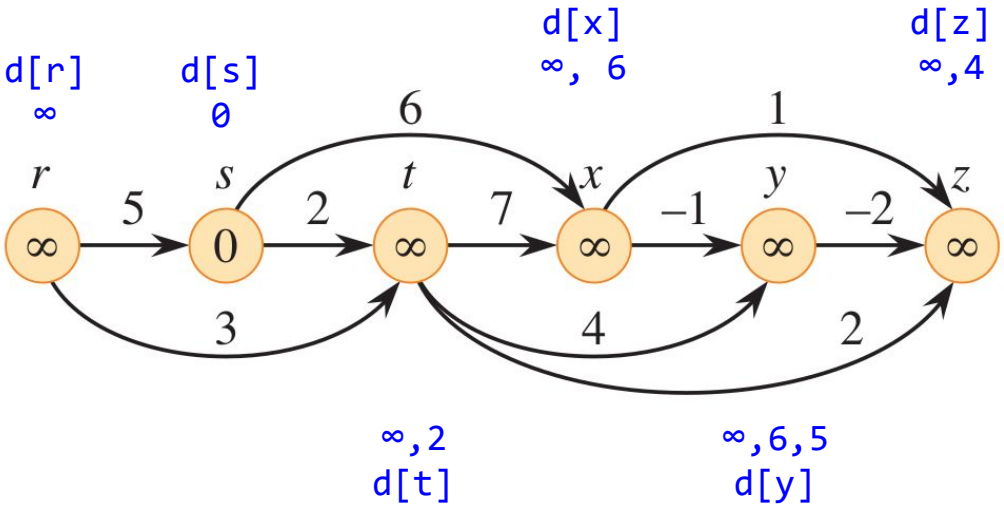
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



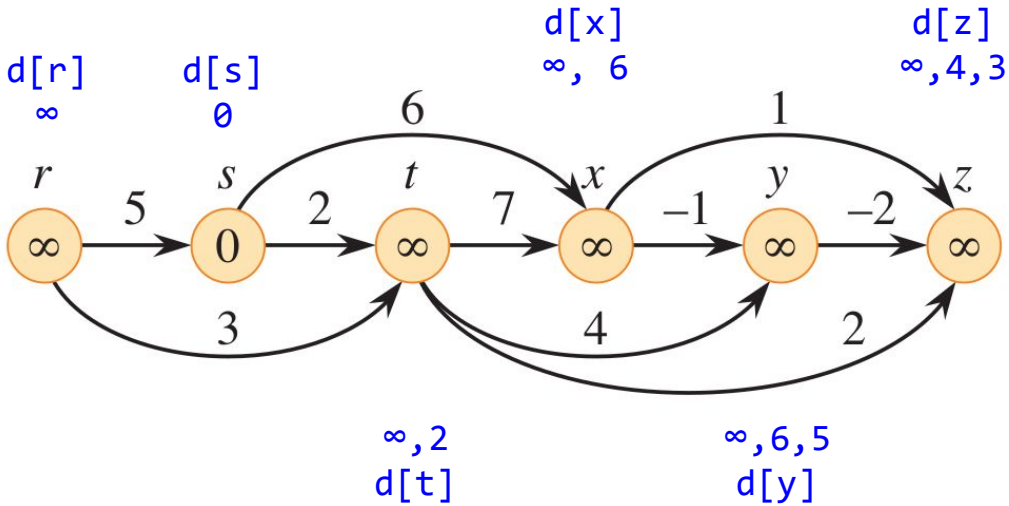
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



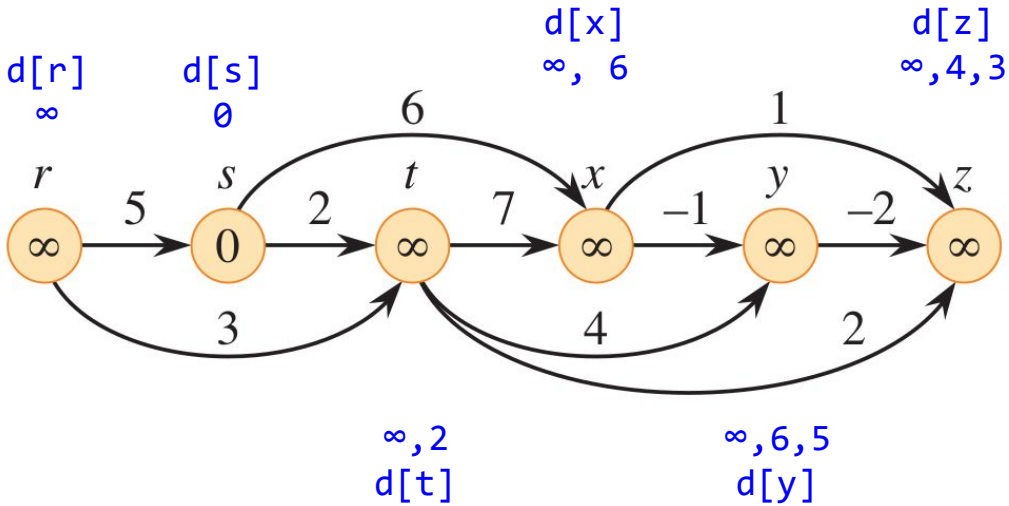
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



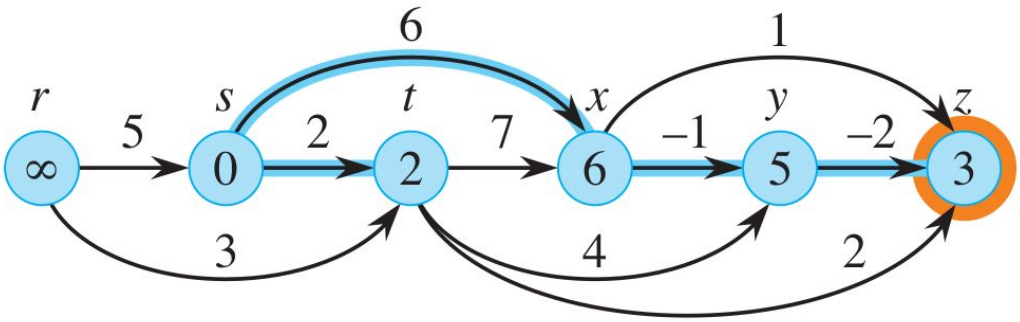
r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



r → {s, t}
s → {t, x}
t → {x, y, z}
x → {y, z}
y → {z}
z → {}

DAG Shortest Paths



DAG Shortest Paths

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