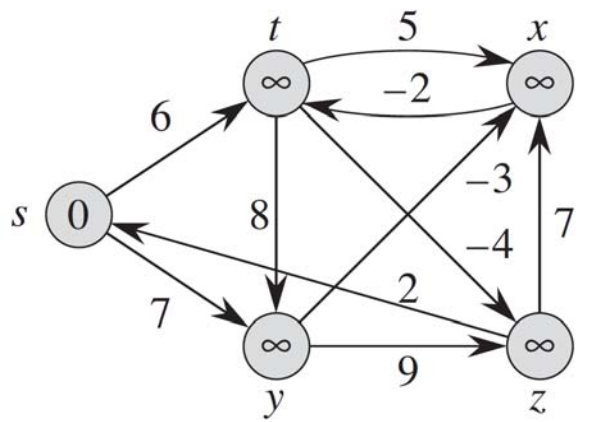
The Bellman-Ford Algorithm (2/3)

- 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.

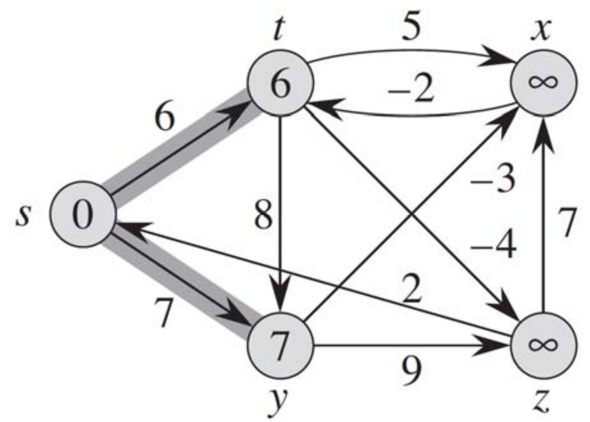
The Bellman-Ford Algorithm (3/3)

```
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
```

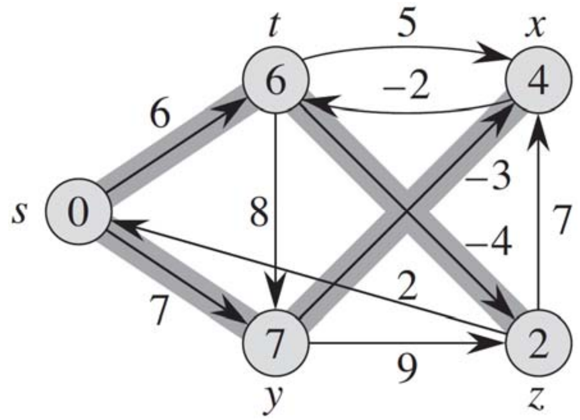

Example (1/5)



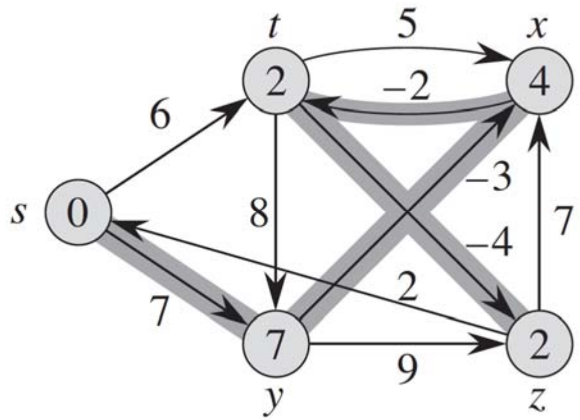
Example (2/5)



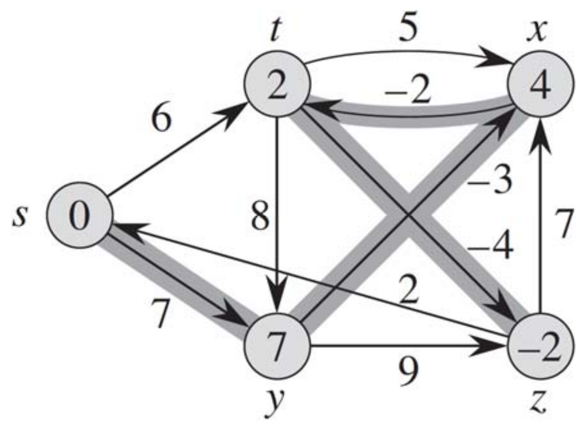
Example (3/5)



Example (4/5)



Example (5/5)



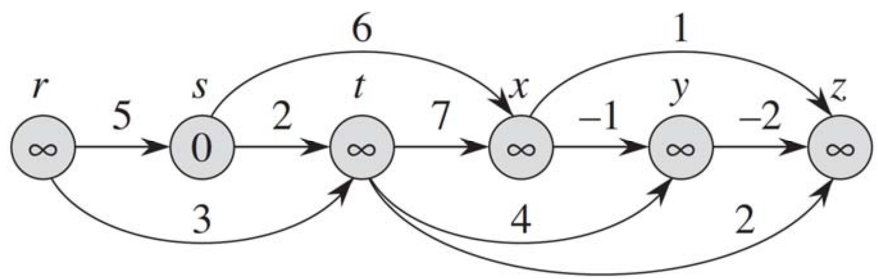
Analysis of Bellman-Ford Algorithm

- The Bellman-Ford algorithm runs in time $O(VE)$
  - The initialization in line 1 takes $\Theta(V)$ time.
  - Each of the $|V| - 1$ passes over the edges in lines 2–4 takes $\Theta(E)$ time.
  - The **for** loop of lines 5–7 takes $O(E)$ time.

Outline

- Shortest Paths
- Shortest-Paths Properties
- The Bellman-Ford Algorithm
- ***Single-Source Shortest Paths in Directed Acyclic Graphs***
- Dijkstra's Algorithm
- Difference Constraints and Shortest Paths

Single-Source Shortest Paths in DAG (1/3)



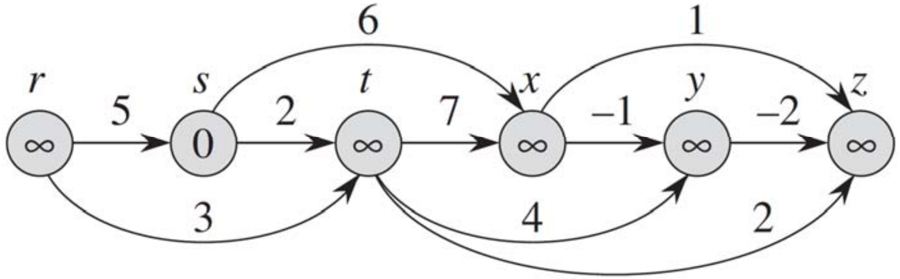
Single-Source Shortest Paths in DAG (2/3)

- Shortest paths are always well defined in a dag (directed acyclic graph), since even if there are negative-weight edges, no negative-weight cycles can exist.
- By relaxing the edges of a weighted dag $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(V + E)$ time.

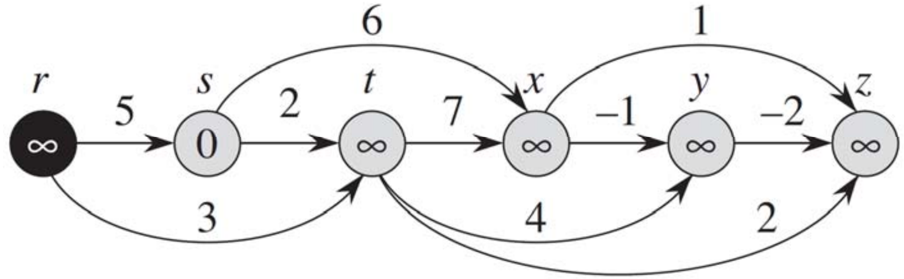
Single-Source Shortest Paths in DAG (3/3)

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

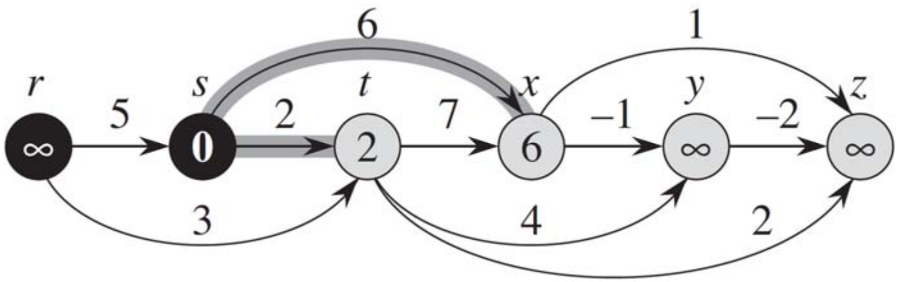
Example (1/7)



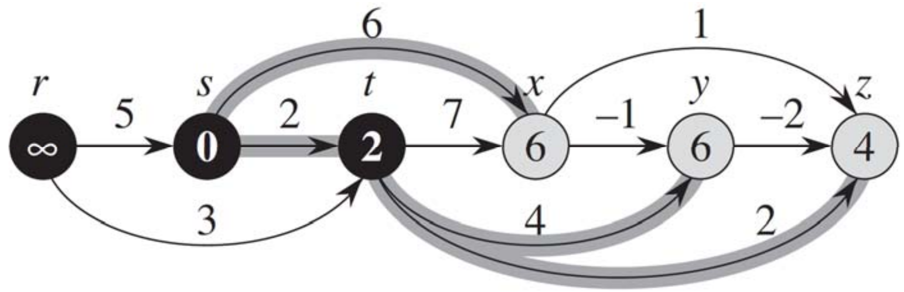
Example (2/7)



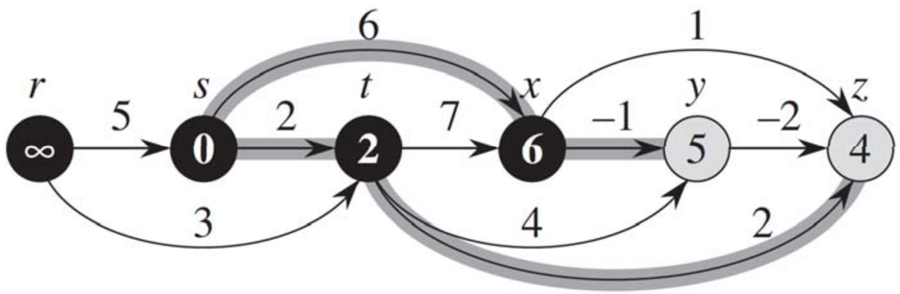
Example (3/7)



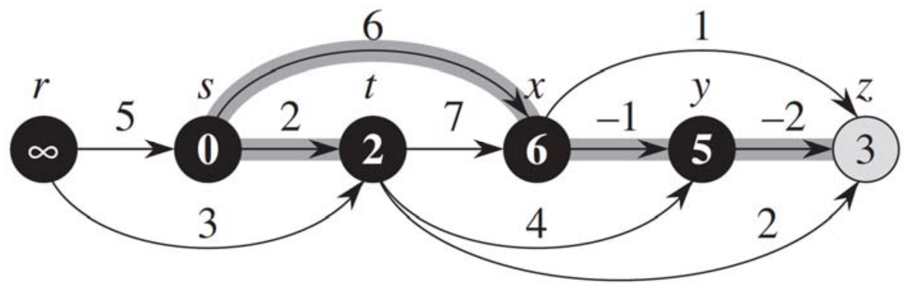
Example (4/7)



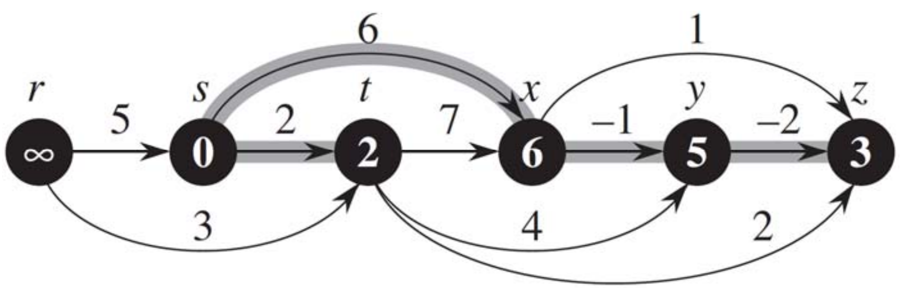
Example (5/7)



Example (6/7)



Example (7/7)



Correctness

- Because we process vertices in topologically sorted order, edges of any path must be relaxed in order of appearance in the path.
- ➡ Edges on any shortest path are relaxed in order.
- ➡ By path-relaxation property, correct.

Outline

- Shortest Paths
- Shortest-Paths Properties
- The Bellman-Ford Algorithm
- Single-Source Shortest Paths in Directed Acyclic Graphs
- ***Dijkstra's Algorithm***
- Difference Constraints and Shortest Paths

Dijkstra's Algorithm (1/3)

- No negative-weight edges.
- Essentially a weighted version of breadth-first search.
  - Instead of a FIFO queue, uses a priority queue.
  - Keys are shortest-path weights ($v.d$).
- Have two sets of vertices:
  - S = vertices whose final shortest-path weights are determined,
  - Q = priority queue = $V - S$.

Dijkstra's Algorithm (2/3)

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


Dijkstra's Algorithm (3/3)

- Looks a lot like Prim's algorithm, but computing $v.d$, and using shortest-path weights as keys.
- Dijkstra's algorithm can be viewed as greedy, since it always chooses the "lightest" ("closest"?) vertex in $V - S$ to add to S.