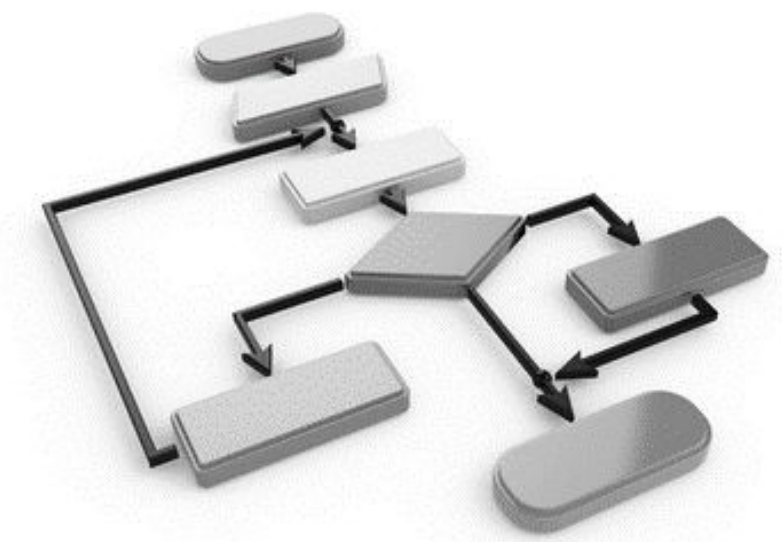
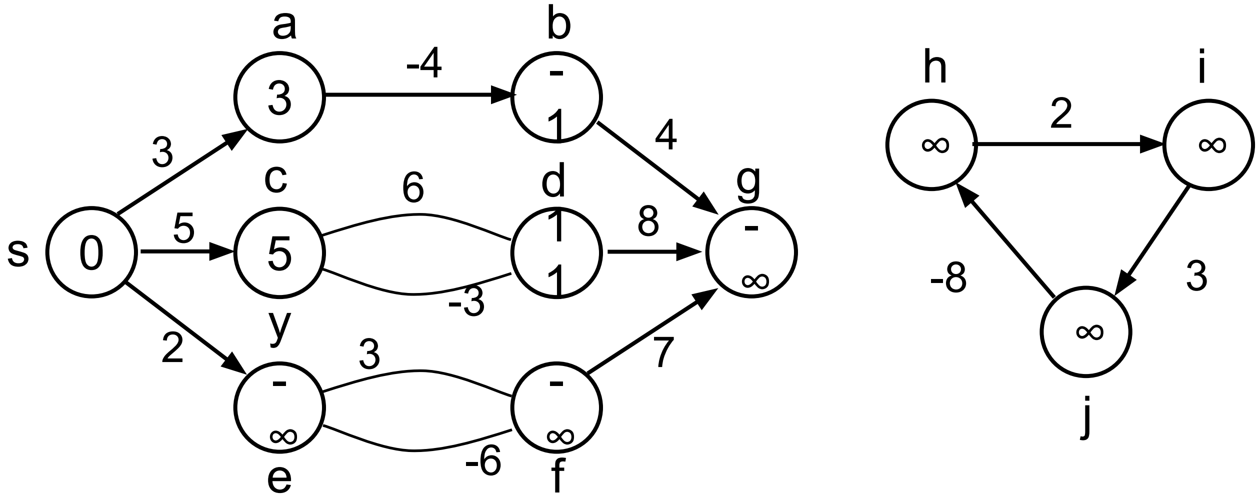


Graph-Based Algorithms

CSE 301: Combinatorial Optimization

Negative-Weight Edges

What if we have negative-weight edges?



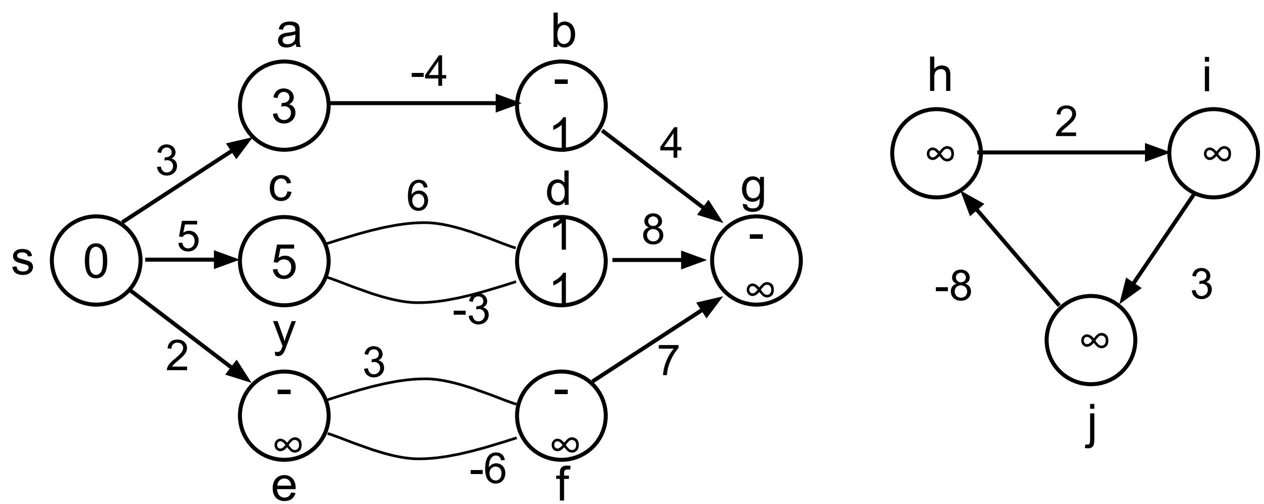
Negative-Weight Edges

s → a: only one path

$\delta(s, a) = w(s, a) = 3$

s → b: only one path

$\delta(s, b) = w(s, a) + w(a, b) = -1$



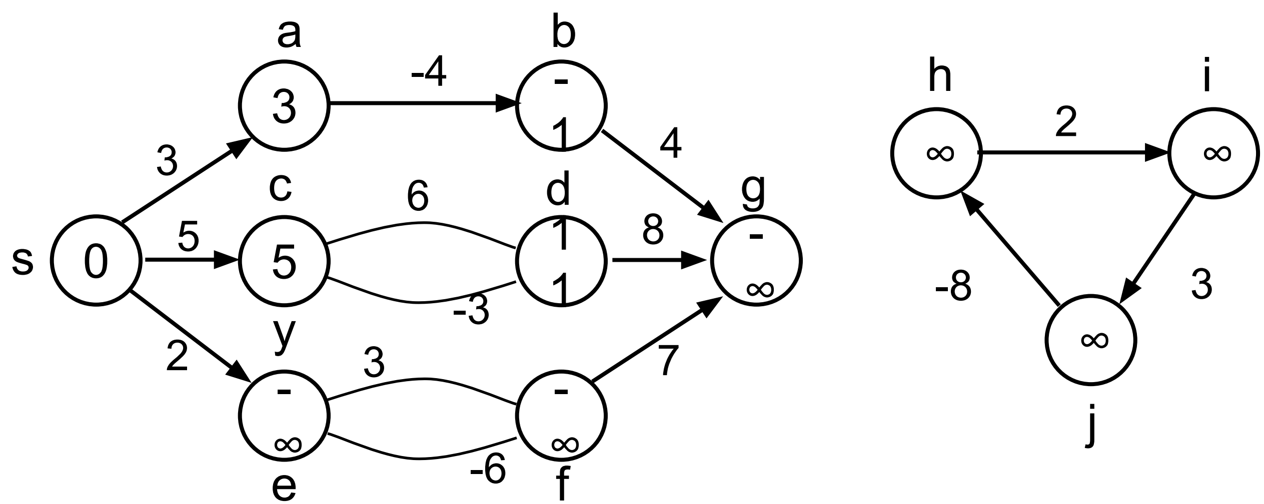
Negative-Weight Edges

s → c: infinitely many paths

⟨s, c⟩, ⟨s, c, d, c⟩, ⟨s, c, d, c, d, c⟩

cycle ⟨c, d, c⟩ has positive weight (6 - 3 = 3)

⟨s, c⟩ is shortest path with weight δ(s, b) = w(s, c) = 5



Negative-Weight Edges

s → e: infinitely many paths:

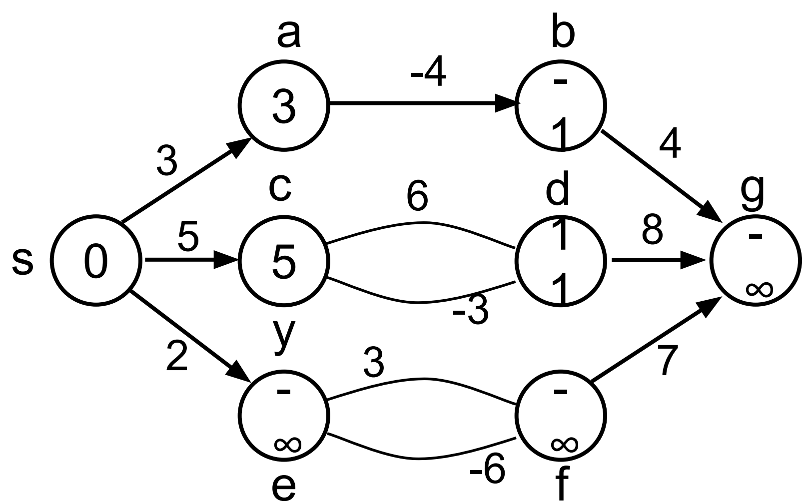
⟨s, e⟩, ⟨s, e, f, e⟩, ⟨s, e, f, e, f, e⟩

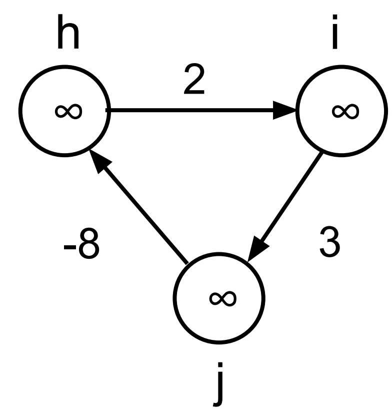
cycle ⟨e, f, e⟩ has negative weight: 3 + (- 6) = -3

many paths from *s* to *e* with arbitrarily large negative weights

δ(s, e) = - ∞ ⇒ no shortest path exists between *s* and *e*

Similarly: δ(s, f) = - ∞, δ(s, g) = - ∞



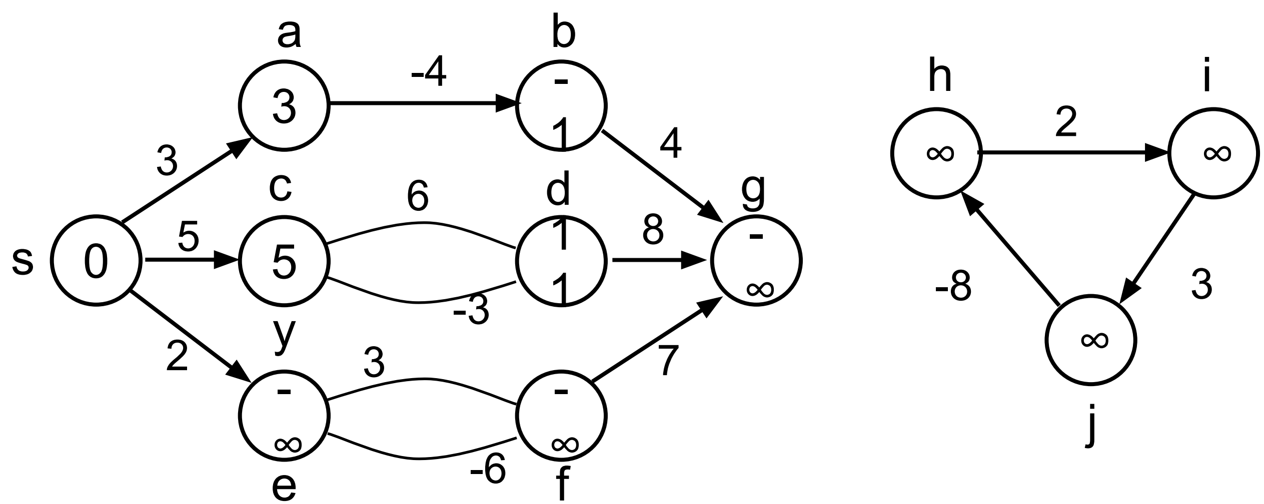


h, i, j not reachable from s

δ(s, h) = δ(s, i) = δ(s, j) = ∞

Negative-Weight Edges

- Negative-weight edges may form negative-weight cycles
- If such cycles are reachable from the source: δ(s, v) is not properly defined



Cycles

Can shortest paths contain cycles?

Negative-weight cycles No!

Positive-weight cycles: No!

By removing the cycle we can get a shorter path

We will assume that when we are finding shortest paths, the paths will have no cycles

Bellman-Ford Algorithm

Single-source shortest paths problem

- Computes d[v] and π[v] for all v $\in$ V

Allows negative edge weights

Returns:

- TRUE if no negative-weight cycles are reachable from the source s
- FALSE otherwise $\Rightarrow$ no solution exists

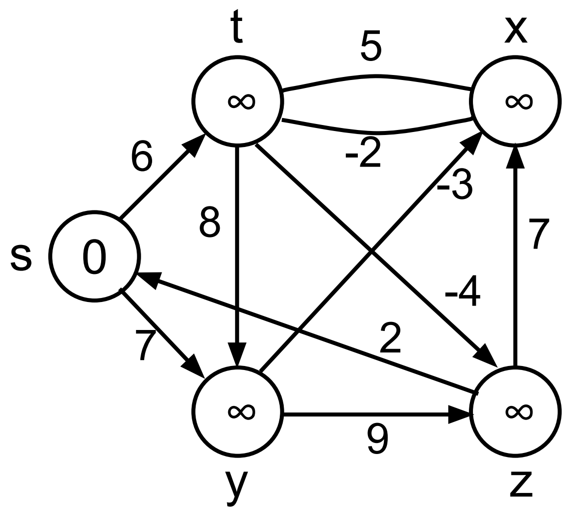
Idea:

- Traverse all the edges |V – 1| times, every time performing a relaxation step of each edge

BELLMAN-FORD(V, E, w, s)

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

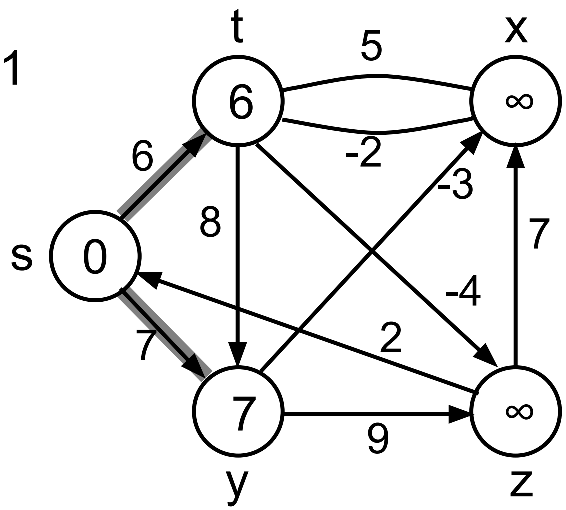
Example



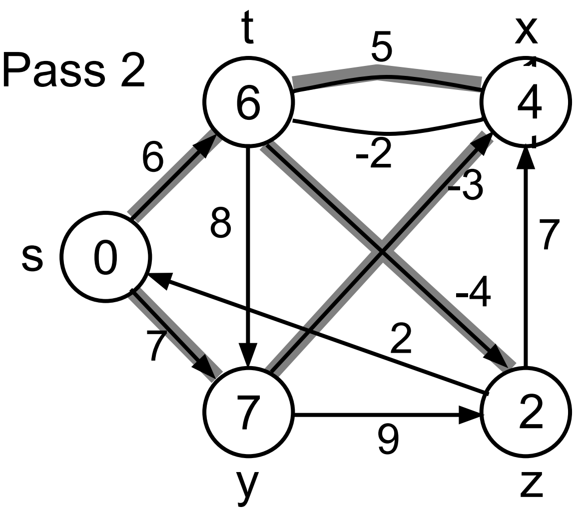
E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Example

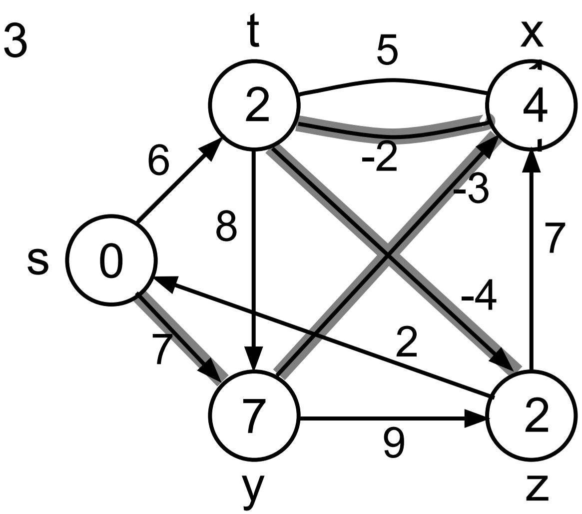
Pass 1



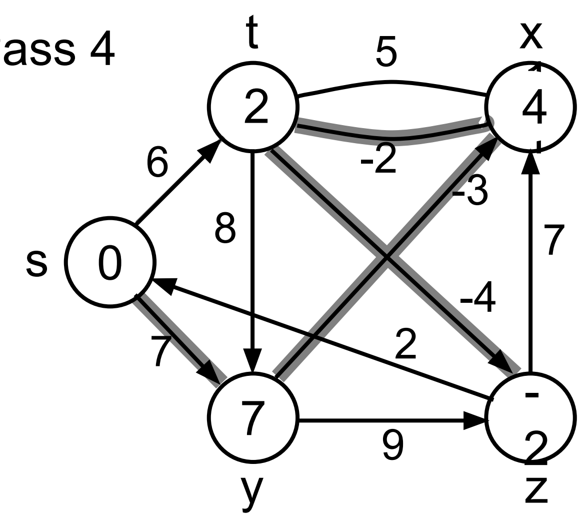
Pass 2



Pass 3



Pass 4



E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Why Bellman-Ford Works

- On the first pass, we find δ (s,u) for all vertices whose shortest paths have one edge.
- On the second pass, the d[*u*] values computed for the one-edge-away vertices are correct (= δ (s,u)), so they are used to compute the correct d values for vertices whose shortest paths have two edges.
- Since no shortest path can have more than |V[G]|-1 edges, after that many passes all d values are correct.
- Note: all vertices not reachable from s will have their original values of infinity. (Same, by the way, for Dijkstra).

Detecting Negative Cycles

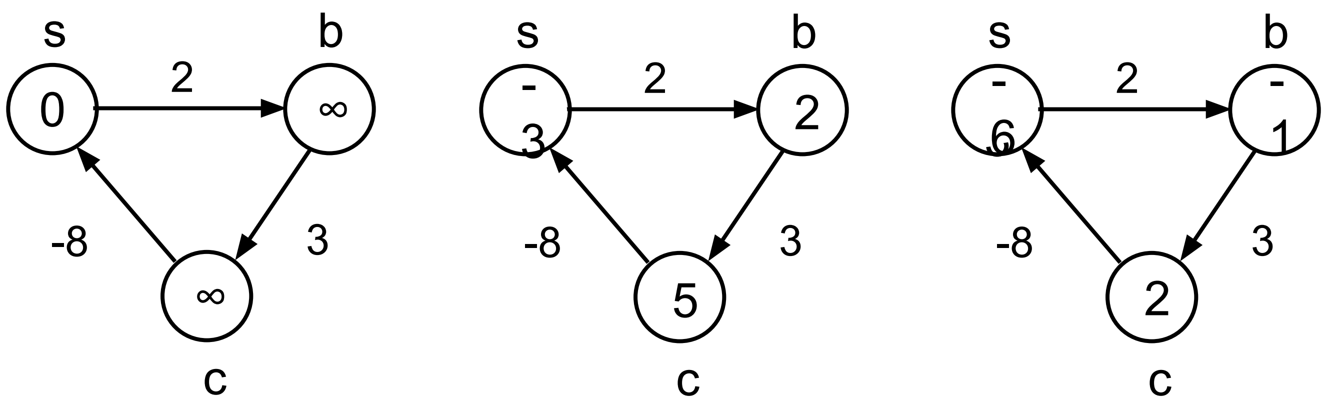
for each edge (u, v) $\in$ E

  do if d[v] > d[u] + w(u, v)

    then return FALSE

return TRUE

E: (s, b), (b,c), (c,s)



Observe edge (s, b):

d[b] = -1, d[s] + w(s, b) = -4

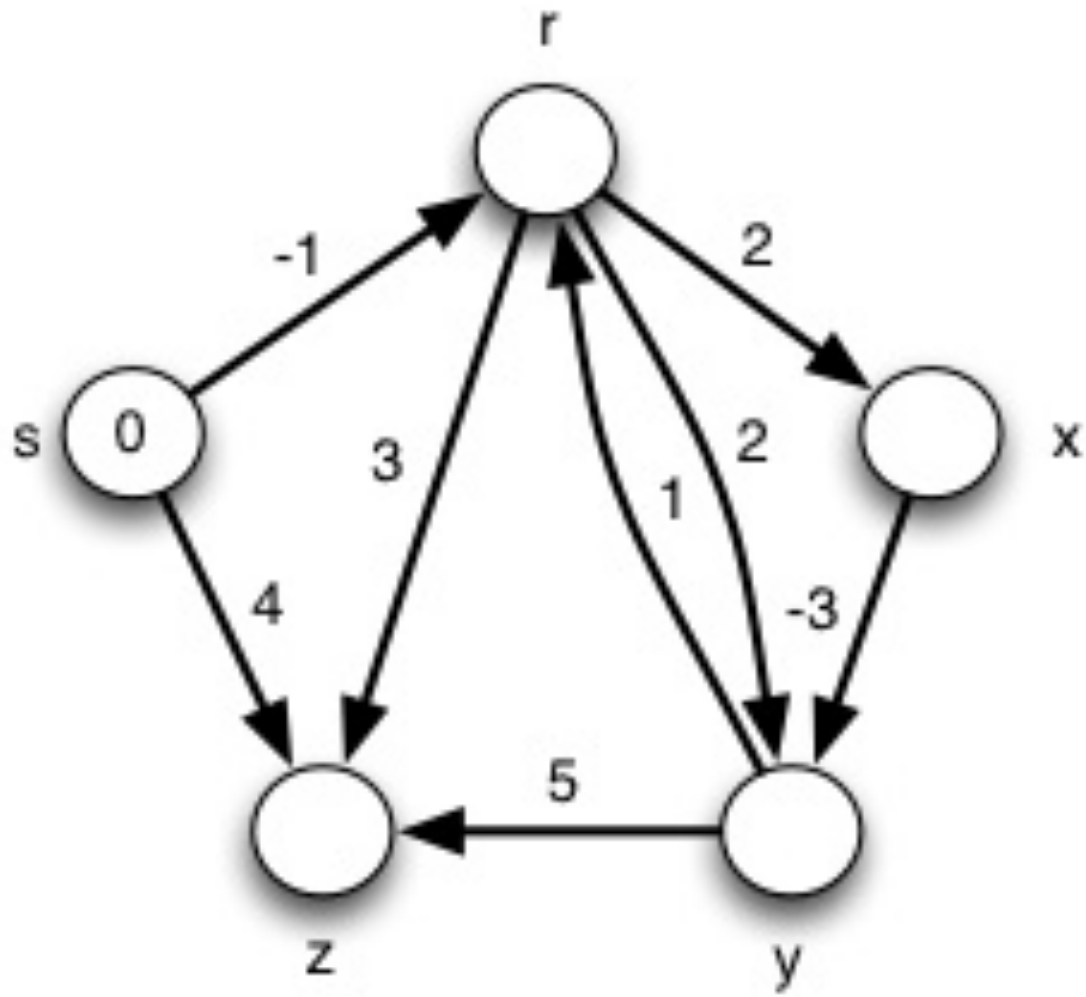
$\Rightarrow$ d[b] > d[s] + w(s, b)

BELLMAN-FORD(V, E, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)          ← Θ(V)
for i ← 1 to |V| - 1                    ← O(V)  ⎫
    do for each edge (u, v) ∈ E         ← O(E)  ⎬ O(VE)
            do RELAX(u, v, w)                   ⎭
for each edge (u, v) ∈ E                ← O(E)
    do if d[v] > d[u] + w(u, v)
          then return FALSE
return TRUE
```

Running time: O(VE)

Exercise: Apply Bellman-Ford algorithm



E: (y, z), (y,r), (x, y), (r, x), (r,y), (r, z), (s, r), (s, z)

Single-Source Shortest Paths in DAGs

Given a weighted Directed Acyclic Graph (DAG): G = (V, E) – solve the shortest path problem

Idea:

- Topologically sort the vertices of the graph
- Relax the edges according to the order given by the topological sort
  - for each vertex, we relax each edge that starts from that vertex

Are shortest-paths well defined in a DAG?

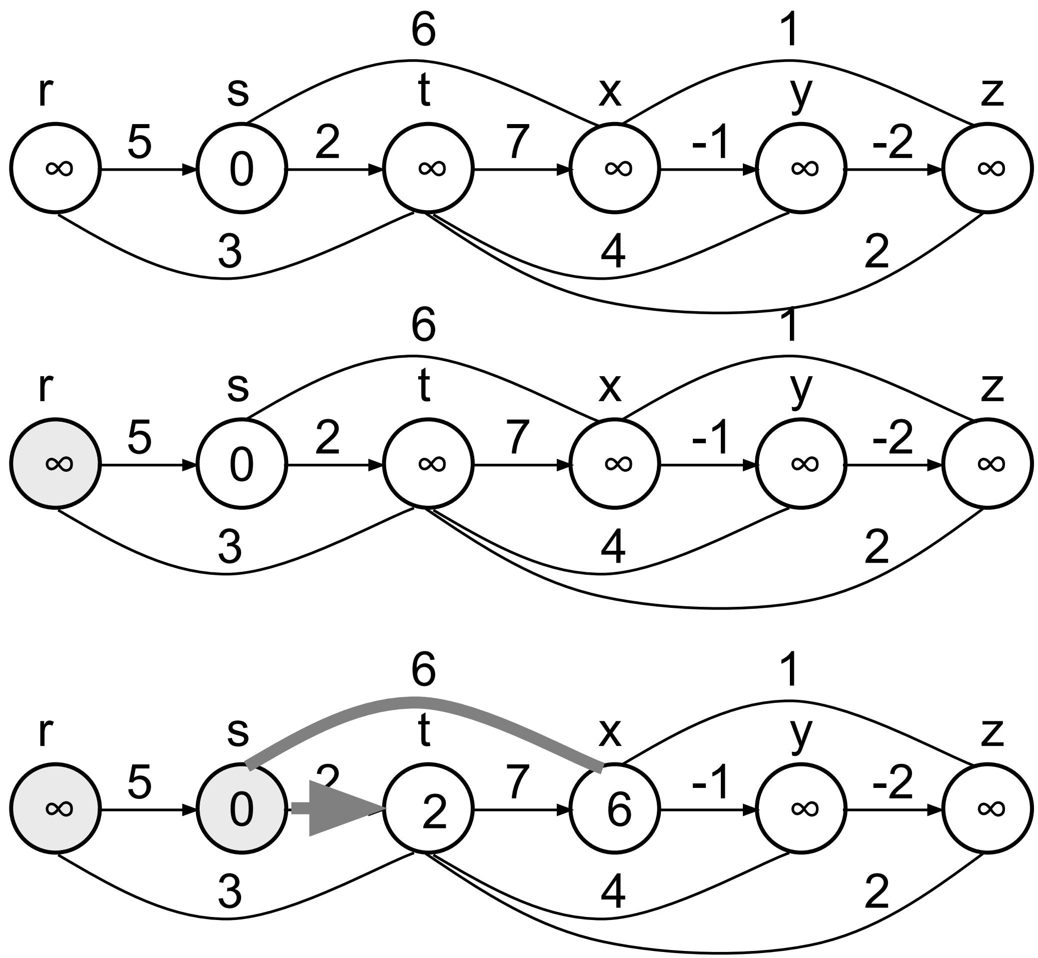
- Yes, since cycles cannot exist in a DAG

DAG-SHORTEST-PATHS(G, w, s)

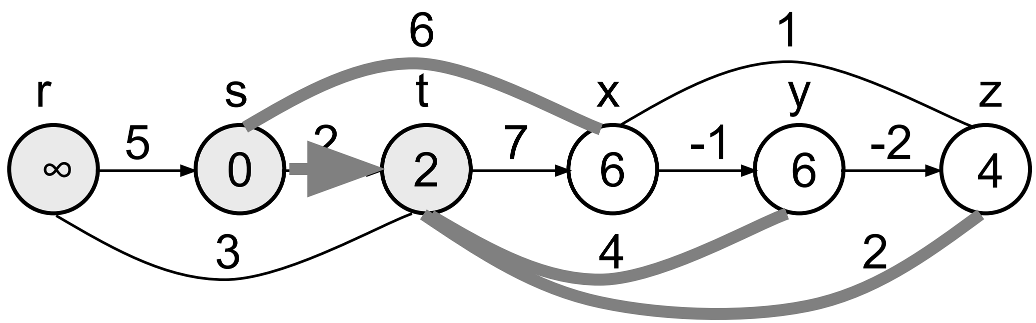
1. topologically sort the vertices of G $\longleftarrow \Theta(V+E)$
2. INITIALIZE-SINGLE-SOURCE(V, s) $\longleftarrow \Theta(V)$
3. **for** each vertex u, taken in topologically sorted order
4. **do for** each vertex v $\in$ Adj[u] $\longleftarrow \Theta(E)$
5. **do** RELAX(u, v, w)

Running time: $\Theta(V+E)$

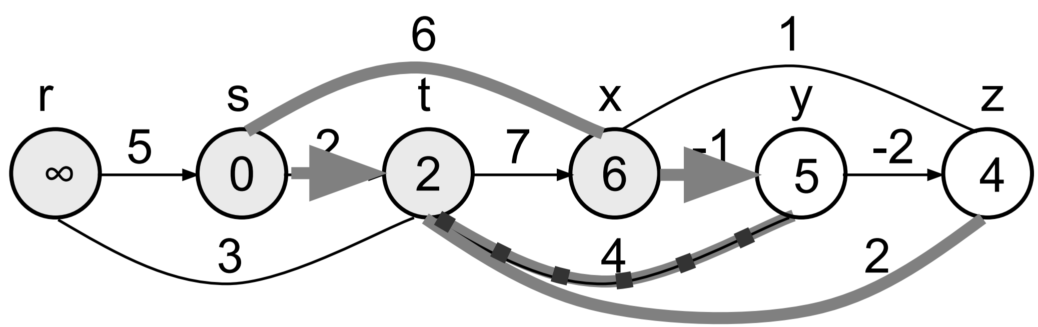
Example


6
1
r
s
t
x
y
z
∞
5
0
2
∞
7
∞
-1
∞
-2
∞
3
4
2
6
1
r
s
t
x
y
z
∞
5
0
2
∞
7
∞
-1
∞
-2
∞
3
4
2
6
1
r
s
t
x
y
z
∞
5
0
2
2
7
6
-1
∞
-2
∞
3
4
2

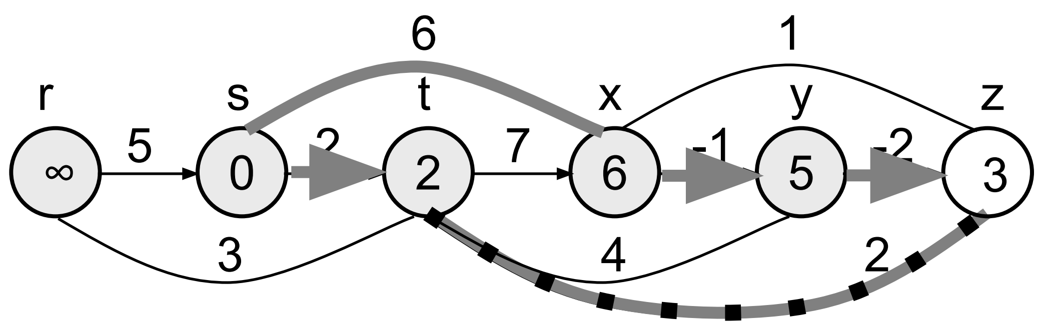
Example



6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
2
6
6
4
3
4
2

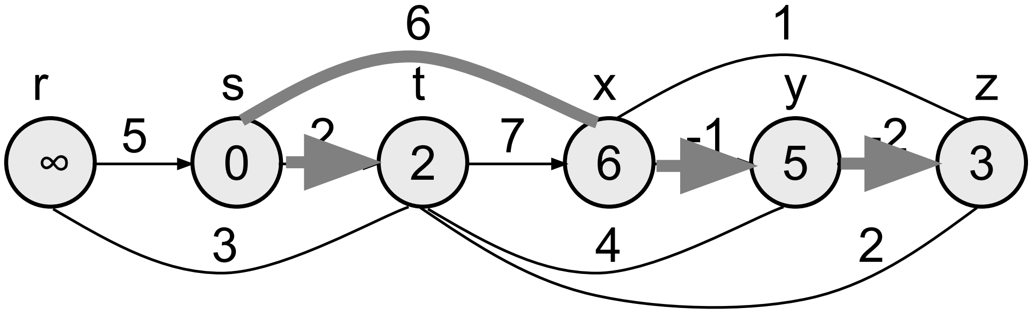


6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
2
6
5
4
3
4
2



6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
2
6
5
3
3
4
2

Example



SSSP in a DAG Theorem

Theorem: For any vertex u in a dag, if all the vertices before u in a topological sort of the dag have been updated, then $d[u] = \delta(s,u)$.

Proof: By induction on the position of a vertex in the topological sort.

<u>Base case</u>: $d[s]$ is initialized to 0.

<u>Inductive case</u>: Assume all vertices before u have been updated, and for all such vertices x, $d[x]=\delta(s,x)$. (continued)

Proof, Continued

Some edge (v,u) must be on the shortest path to u. Therefore v must precede u in topological order and as such must have been updated before u. So d[v] = $\delta(s,v)$.

When u is updated, we set d[u] to d[v]+w(v,u)

$= \delta(s,v) + w(v,u)$

$= \delta(s,u)$ ■