Analysis of Algorithms
CS 477/677

Instructor: Monica Nicolescu

Lecture 25

Shortest Path Problems

- How can we find the shortest route between two points on a map?
- Model the problem as a graph problem:
  - Road map is a weighted graph:

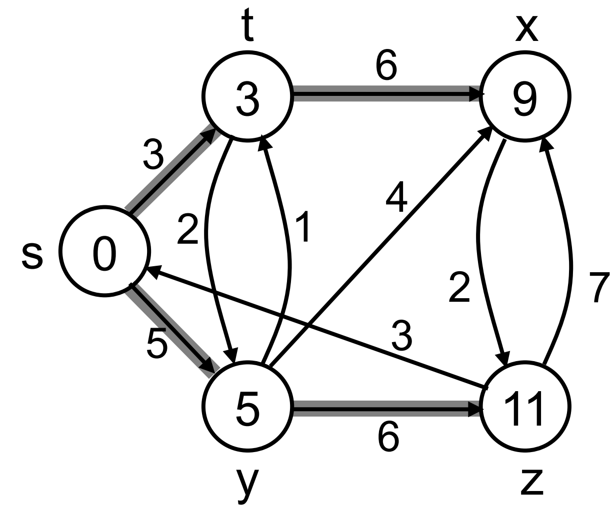
    vertices = cities

    edges = road segments between cities

    edge weights = road distances
  - Goal: find a shortest path between two vertices (cities)

Shortest Path Problems

- **Input:**
  - Directed graph $G = (V, E)$
  - Weight function $w : E \rightarrow \mathbf{R}$



- **Weight of path** $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- **Shortest-path weight** from **u** to **v**:

$$\delta(u, v) = \min \begin{cases} w(p) : u \overset{p}{\rightsquigarrow} v & \text{if there exists a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- Shortest path **u** to **v** is any path **p** such that **w(p) = δ(u, v)**

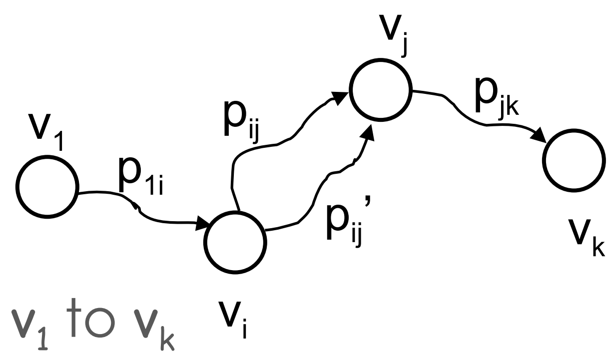
Variants of Shortest Paths

- **Single-source shortest path**
  - G = (V, E) ⇒ find a shortest path from a given source vertex **s** to each vertex **v** ∈ **V**
- **Single-destination shortest path**
  - Find a shortest path to a given destination vertex **t** from each vertex **v**
  - Reverse the direction of each edge ⇒ single-source
- **Single-pair shortest path**
  - Find a shortest path from **u** to **v** for given vertices **u** and **v**
  - Still have to solve the single-source problem
- **All-pairs shortest-paths**
  - Find a shortest path from **u** to **v** for every pair of vertices **u** and **v**

Optimal Substructure of Shortest Paths

Given:

- A weighted, directed graph $G = (V, E)$
- A weight function $w: E \rightarrow \mathbf{R}$,
- A shortest path $\mathbf{p} = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A subpath of $\mathbf{p}$: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \le i \le j \le k$



Then: $\mathbf{p_{ij}}$ is a shortest path from $\mathbf{v_i}$ to $\mathbf{v_j}$

Proof: $p = v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$$

Assume $\exists\ p_{ij}'$ from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$ contradiction!

Negative-Weight Edges

- $s \rightarrow a$: only one path

  $\boldsymbol{\delta}(s, a) = w(s, a) = 3$

- $s \rightarrow b$: only one path

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

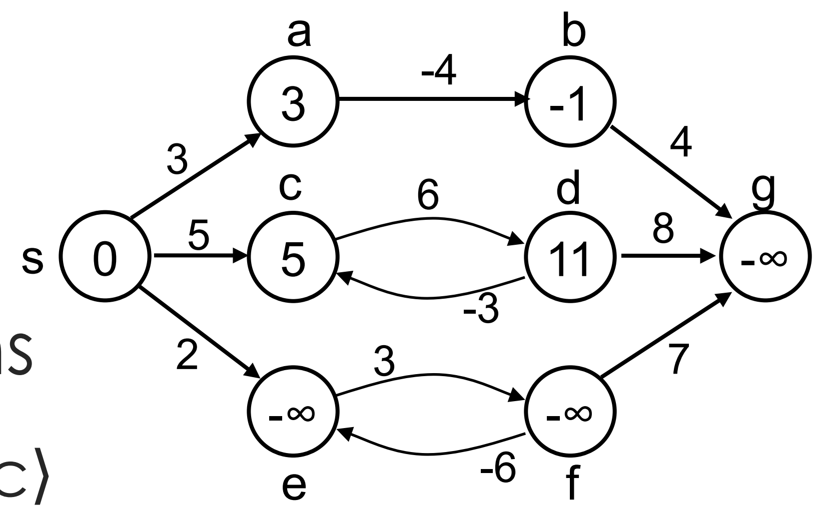
- $s \rightarrow c$: infinitely many paths

  $\langle s, c \rangle, \langle s, c, d, c \rangle, \langle s, c, d, c, d, c \rangle$

  cycle $\langle c, d, c \rangle$ has positive weight $(6 - 3 = 3)$

  $\langle s, c \rangle$ is shortest path with weight $\boldsymbol{\delta}(s, b) = w(s, c) = 5$

What if we have negative-weight edges?

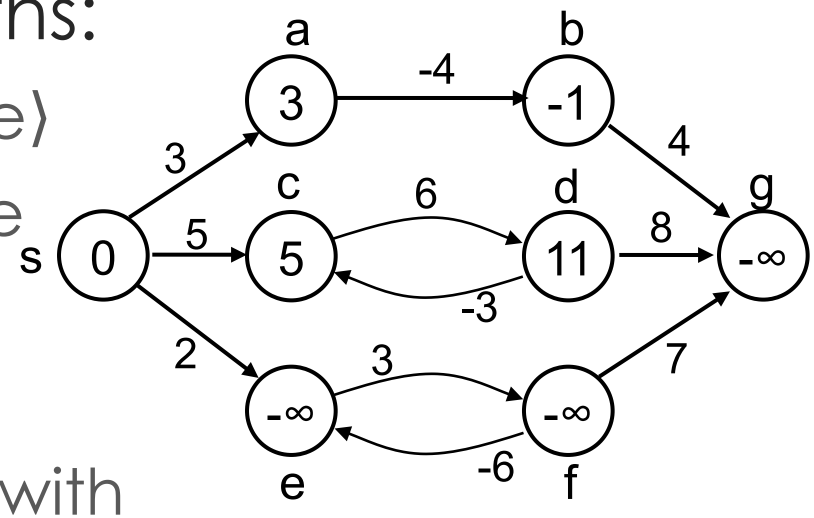


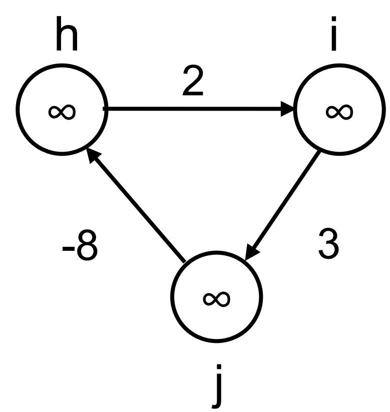
Negative-Weight Edges

- s → e: infinitely many paths:
  ⟨s, e⟩, ⟨s, e, f, e⟩, ⟨s, e, f, e, f, e⟩
  - cycle ⟨e, f, e⟩ has negative weight:

    $$3 + (-6) = -3$$
  - can find paths from ***s*** to ***e*** with arbitrarily large negative weights
  - $\boldsymbol{\delta}(s, e) = -\infty \Rightarrow$ no shortest path exists between ***s*** and ***e***
- Similarly: $\boldsymbol{\delta}(s, f) = -\infty$,

  $\boldsymbol{\delta}(s, g) = -\infty$



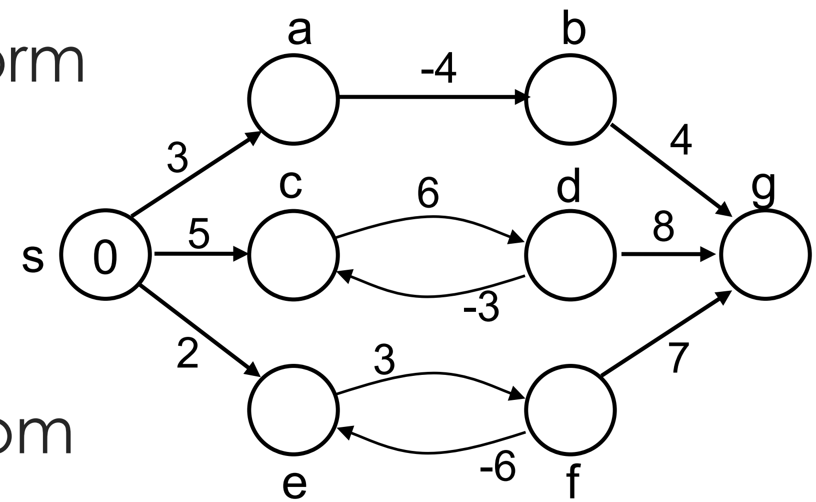


h, i, j not reachable from s

$\boldsymbol{\delta}(s, h) = \boldsymbol{\delta}(s, i) = \boldsymbol{\delta}(s, j) = \infty$

Negative-Weight Edges

- Negative-weight edges may form negative-weight cycles
- If such cycles are reachable from the source: $\delta(s, v)$ is not properly defined for any node **v** on the cycle
  - Keep going around the cycle, and get w(s, v) = - ∞ for all **v** on the cycle



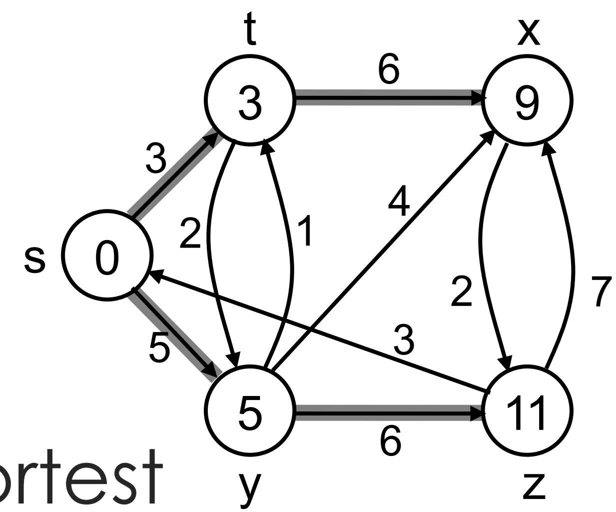
Cycles

- Can shortest paths contain cycles?
- Negative-weight cycles No!
- Positive-weight cycles: No!
  - By removing the cycle we can get a shorter path
- Zero-weight cycles
  - No reason to use them
  - Can remove them to obtain a path with same weight
- We will assume that when we are finding shortest paths, the paths will have no cycles

Shortest-Path Representation

For each vertex v ∈ V:

- d[v] = δ(s, v): a **shortest-path estimate**
  - Initially, d[v]=∞
  - Reduces as algorithms progress
- **π**[v] = **predecessor** of **v** on a shortest path from ***s***
  - If no predecessor, **π**[v] = NIL
  - **π** induces a tree—**shortest-path tree**
- Shortest paths & shortest path trees are not unique



Initialization

Alg.: INITIALIZE-SINGLE-SOURCE(V, s)

1. **for** each $v \in V$
2. **do** $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow$ NIL
4. $d[s] \leftarrow 0$

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

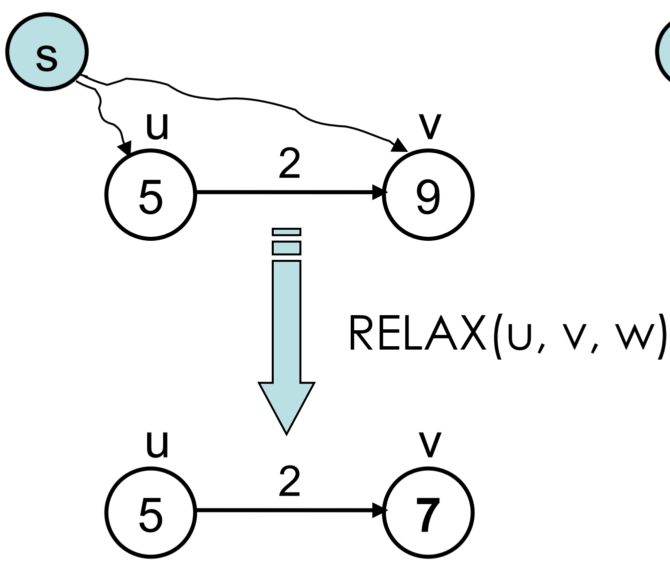
Relaxation

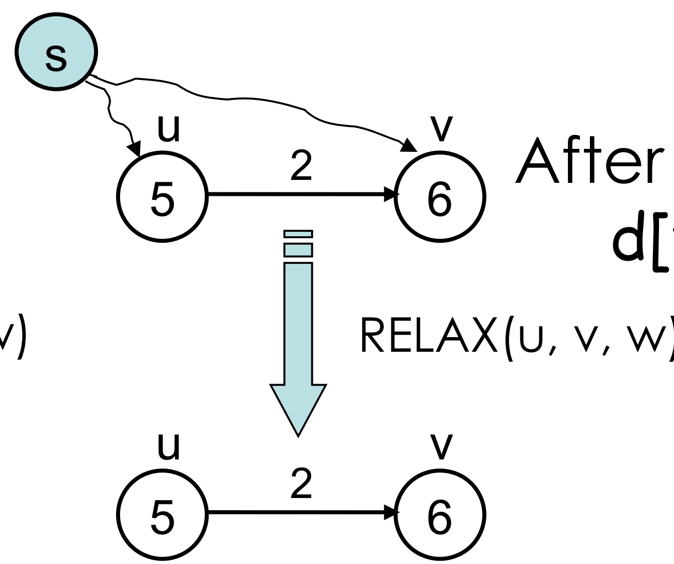
- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

  If d[v] > d[u] + w(u, v)

  we can improve the shortest path to v

  ⇒ update d[v] and **π**[v]





After relaxation:

d[v] ≤ d[u] + w(u, v)

RELAX(u, v, w)

1. **if** d[v] > d[u] + w(u, v)
2. **then** d[v] ← d[u] + w(u, v)
3. π[v] ← u

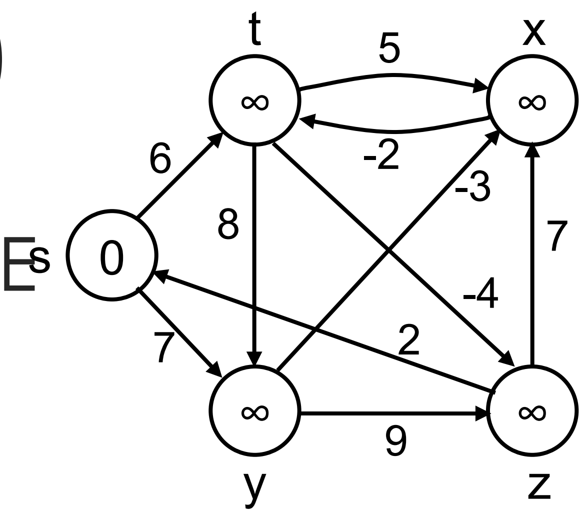
- All the single-source shortest-paths algorithms
  - start by calling INIT-SINGLE-SOURCE
  - then relax edges
- The algorithms differ in the order and how many times they relax each edge

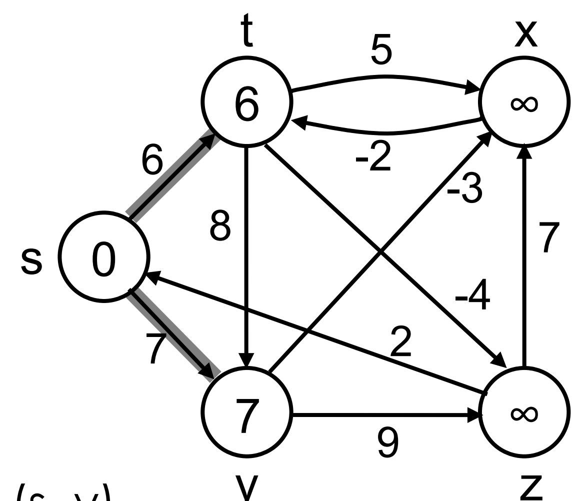
Bellman-Ford Algorithm

- Single-source shortest paths problem
  - Computes d[v] and **π**[v] for all $v \in V$
- Allows negative edge weights and cycles
- Returns:
  - TRUE if no negative-weight cycles are reachable from the source s
  - FALSE otherwise ⇒ 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
```



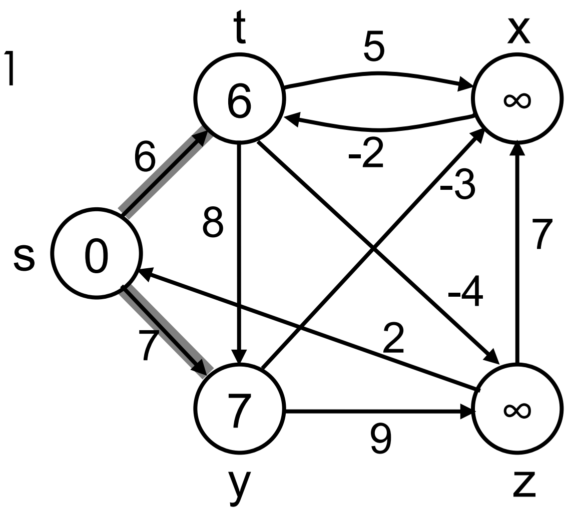


unning time: $\Theta(VE)$

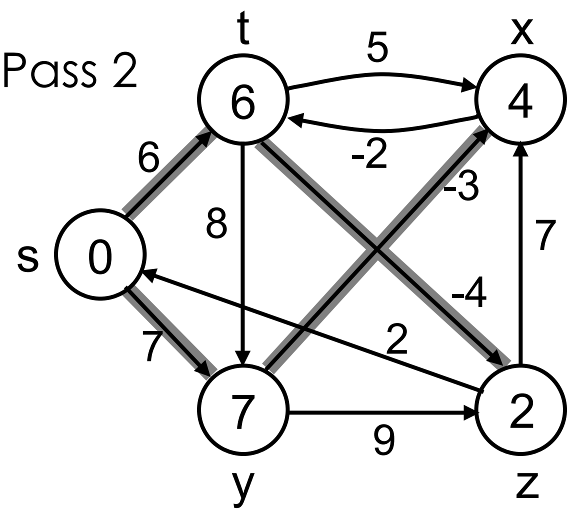
E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

Example (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

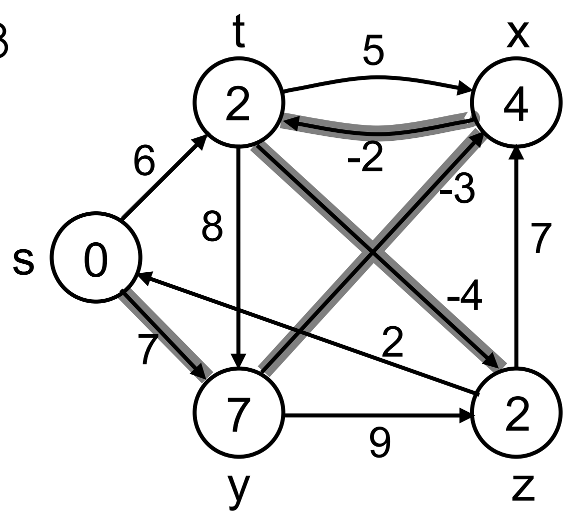
Pass 1



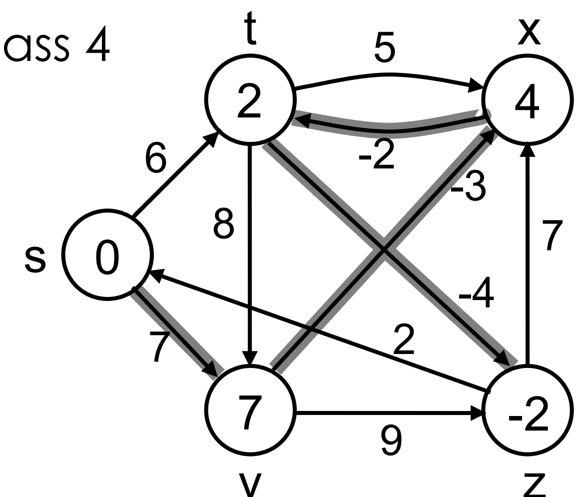
Pass 2



Pass 3

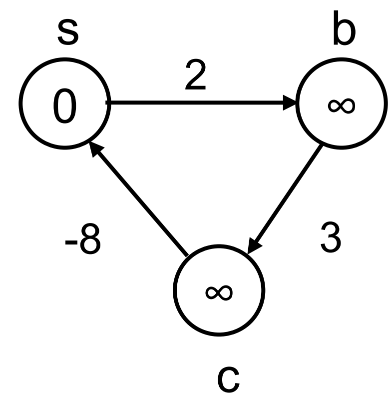


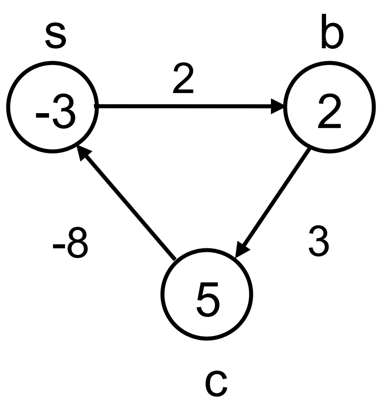
Pass 4

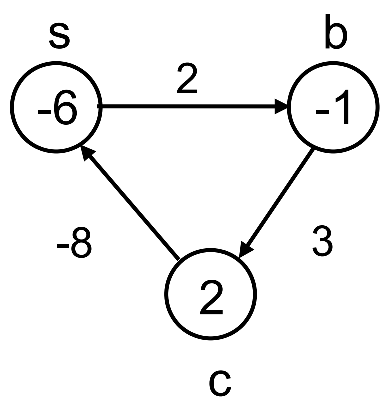


Detecting Negative Cycles

- **for** each edge (u, v) ∈ E
- **do if** d[v] > d[u] + w(u, v)
- **then return** FALSE
- **return** TRUE





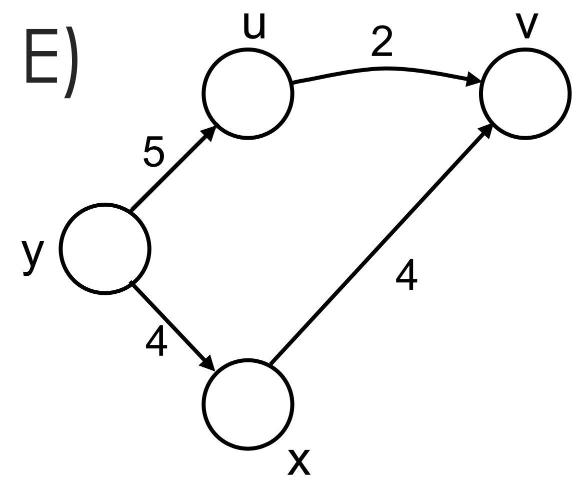


Look at edge (s, b):

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

⇒ d[b] > d[s] + w(s, b)

Single-Source Shortest Paths in DAGs



- Given a weighted DAG: $G = (V, E)$ solve the shortest path problem
- Idea:
  - Topologically sort 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, (negative-weight) cycles cannot exist

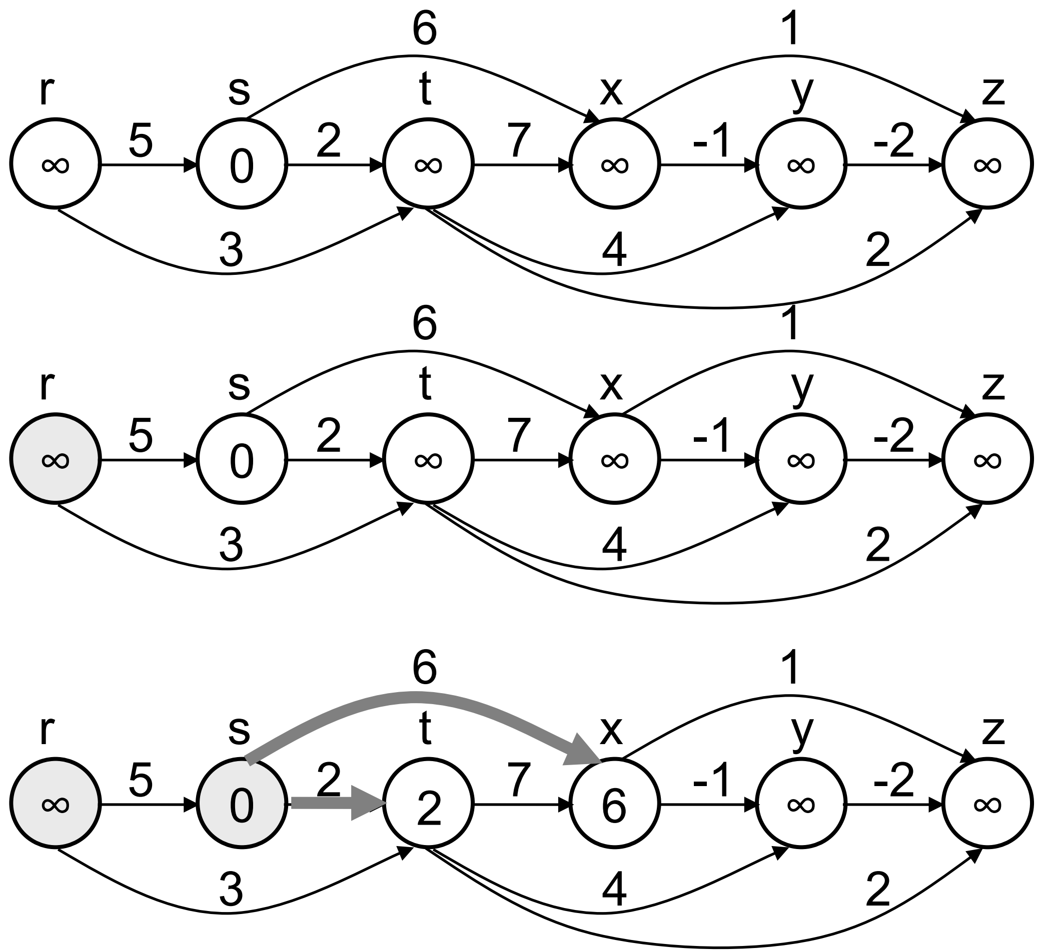
DAG-SHORTEST-PATHS(G, w, s)

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

$\Theta(E)$

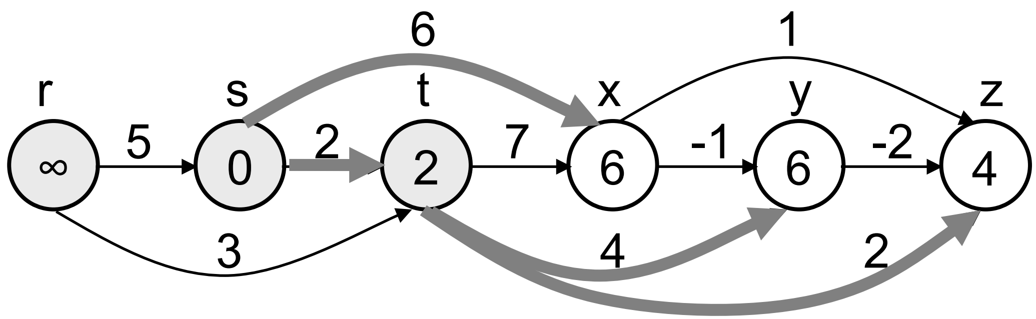
Running time: $\Theta(V+E)$

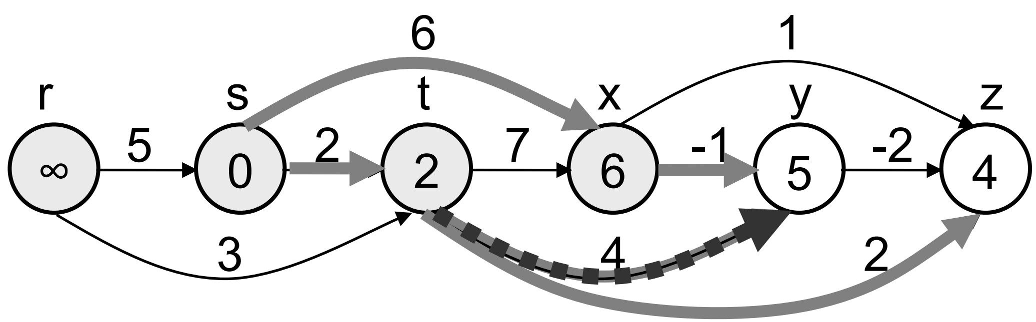
Example

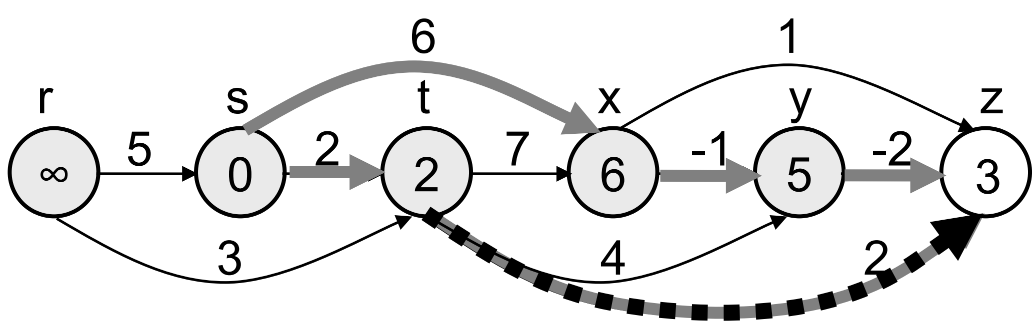


6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
∞
∞
∞
∞
3
4
2
6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
∞
∞
∞
∞
3
4
2
6
1
r
s
t
x
y
z
5
2
7
-1
-2
∞
0
2
6
∞
∞
3
4
2

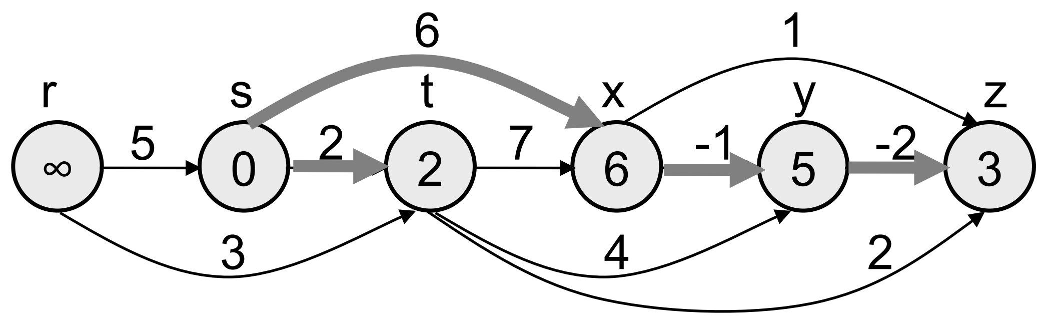
Example (cont.)







Example (cont.)

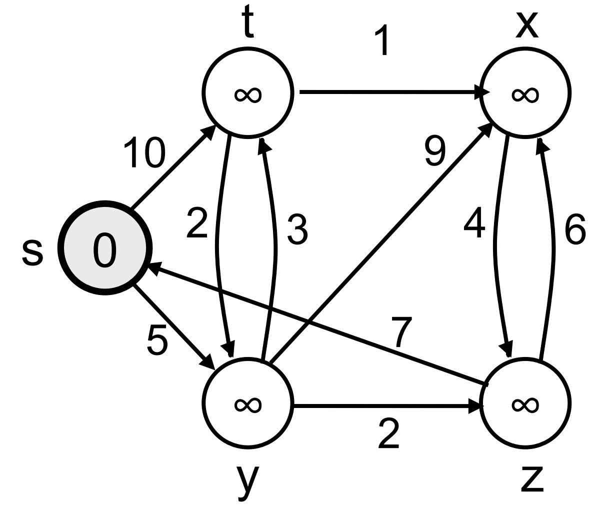


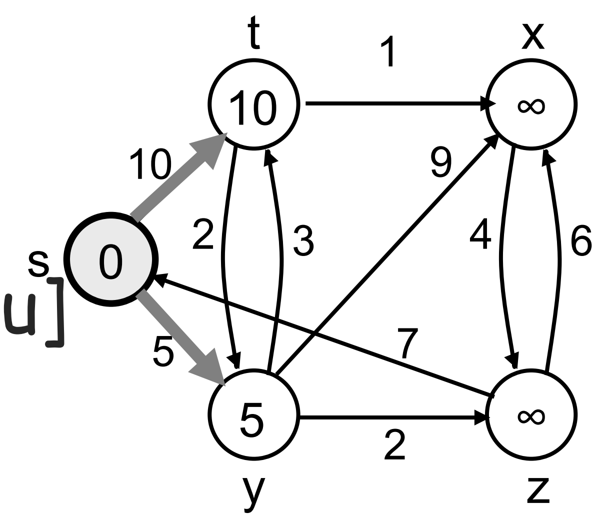
Dijkstra's Algorithm

- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0 \; \forall \; (u, v) \in E$
- Maintains two sets of vertices:
  - S = vertices whose final shortest-path weights have already been determined
  - Q = vertices in V – S: min-priority queue
    - Keys in Q are estimates of shortest-path weights (d[v])
- Repeatedly select a vertex $u \in V - S$, with the minimum shortest-path estimate d[v]

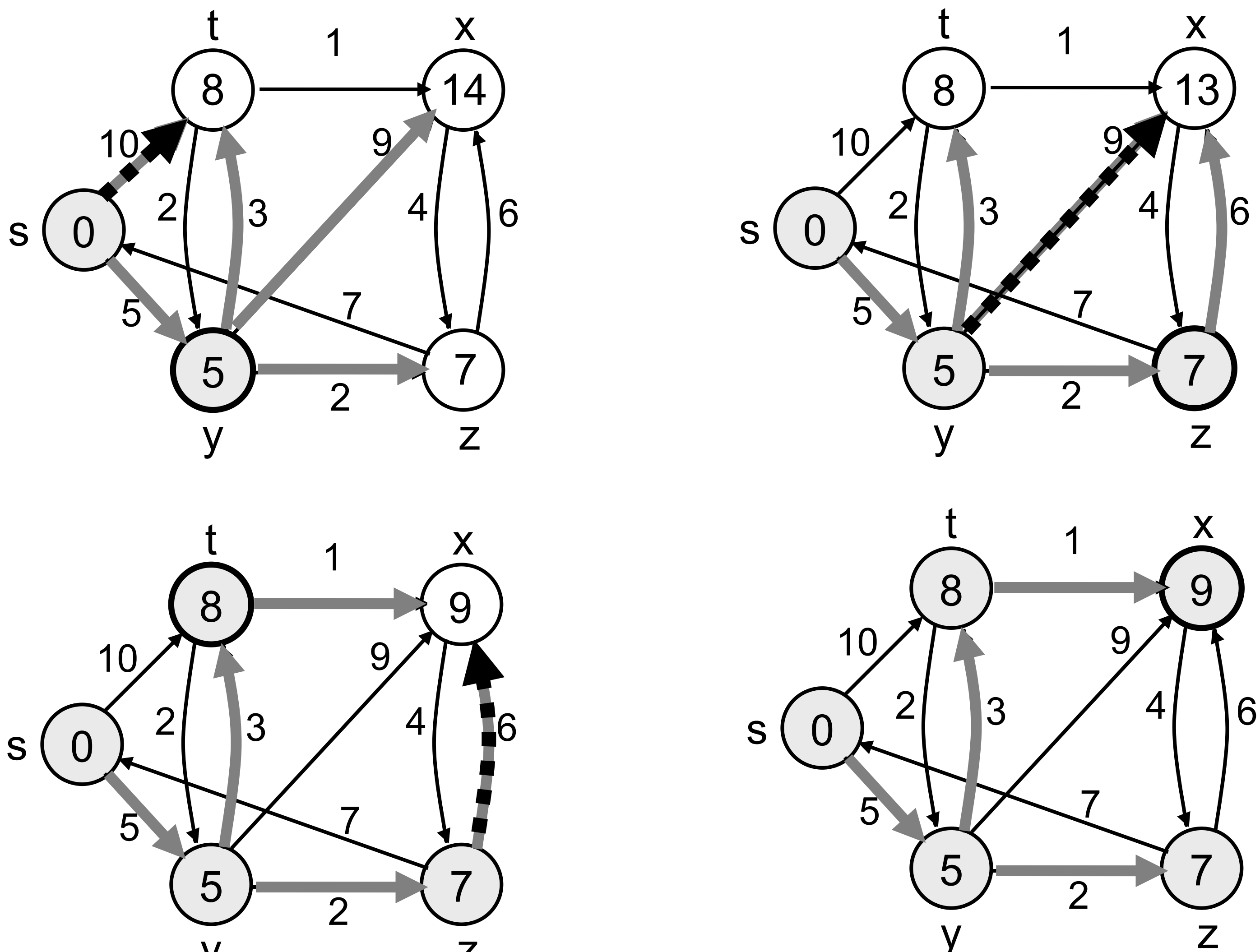
Dijkstra (G, w, s)

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





Example



t
x
1
8
14
10
9
2
3
4
6
s
0
5
7
5
7
2
y
z
t
x
1
8
13
10
9
2
3
4
6
s
0
5
7
5
7
2
y
z
t
x
1
8
9
10
9
2
3
4
6
s
0
5
7
5
7
2
y
z
t
x
1
8
9
10
9
2
3
4
6
s
0
5
7
5
7
2
y
z

Dijkstra (G, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)        ← Θ(V)
S ← ∅
Q ← V[G] ← O(V) build min-heap
while Q ≠ ∅ ← Executed O(V) times
    do u ← EXTRACT-MIN(Q)  ← O(lgV)
        S ← S ∪ {u}
        for each vertex v ∈ Adj[u]
            do RELAX(u, v, w)  ← O(E) times; O(lgV)
```

Running time: *O(VlgV + ElgV) = O(ElgV)* optional

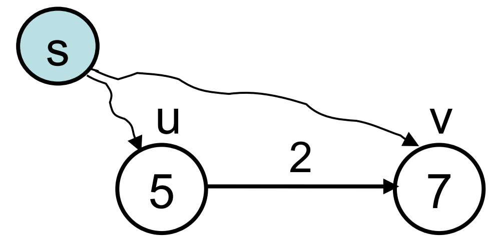
ADDITIONAL SLIDES

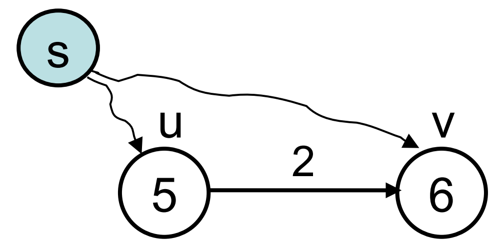
Shortest Path Properties

- **Triangle inequality**

  For all (u, v) ∈ E, we have:

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





- If **u** is on the shortest path to **v** we have the equality sign

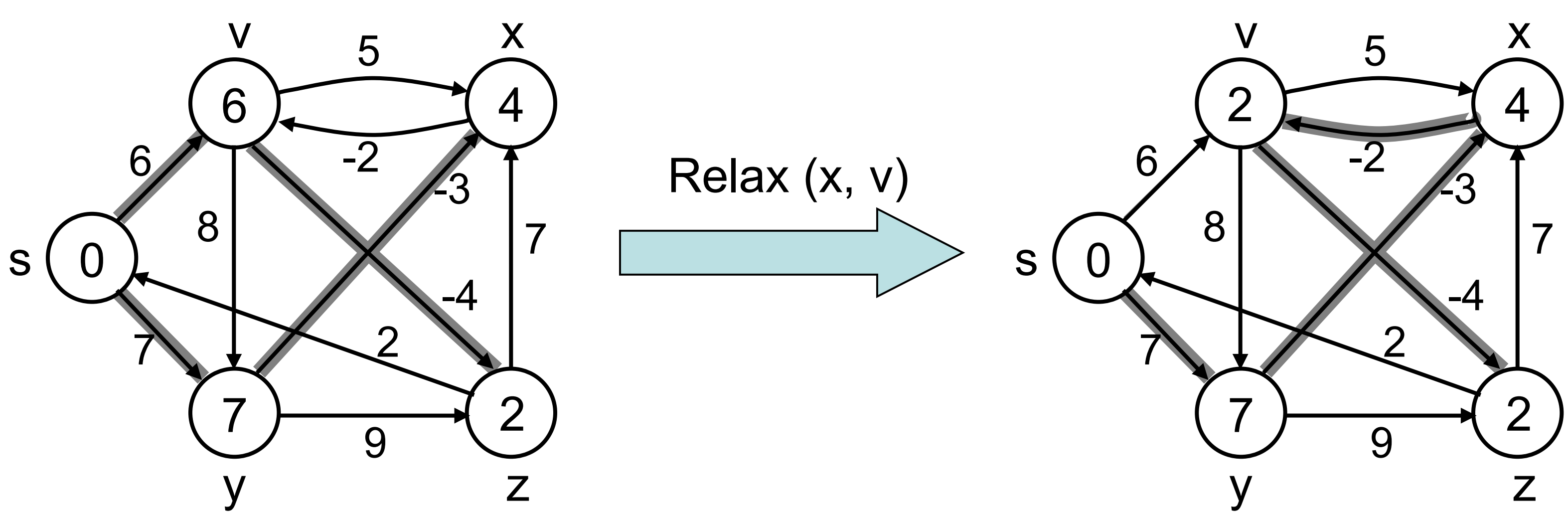
Shortest Path Properties

- **Upper-bound property**

  We always have $d[v] \geq \delta(s, v)$ for all v.

  Once $d[v] = \delta(s, v)$, it never changes.

  – The estimate never goes up – relaxation only lowers the estimate

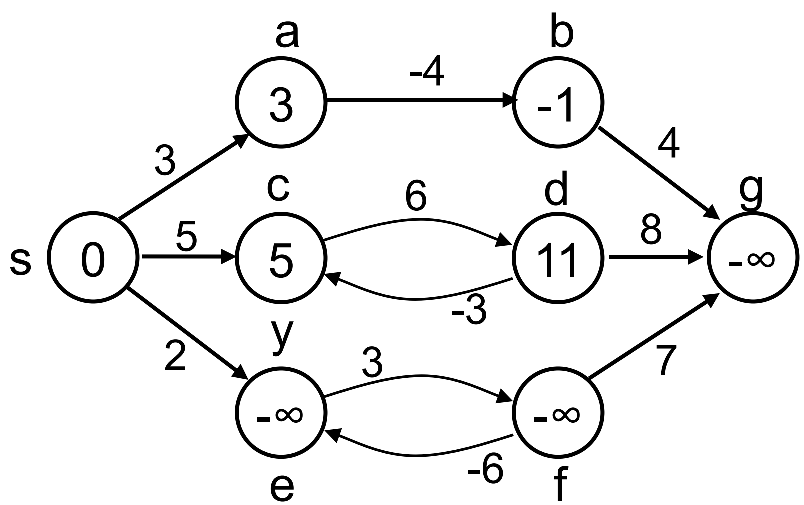


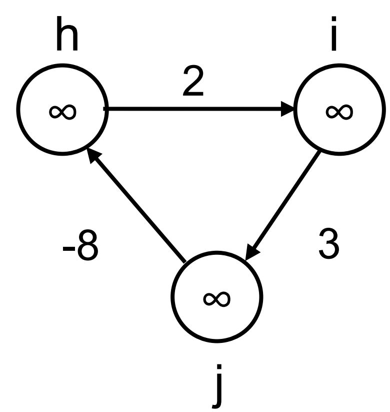
Shortest Path Properties

- **No-path property**

  If there is no path from s to **v** then d[v] = ∞ always.

  - $\delta(s, h) = \infty$ and $d[h] \geq \delta(s, h) \Rightarrow d[h] = \infty$





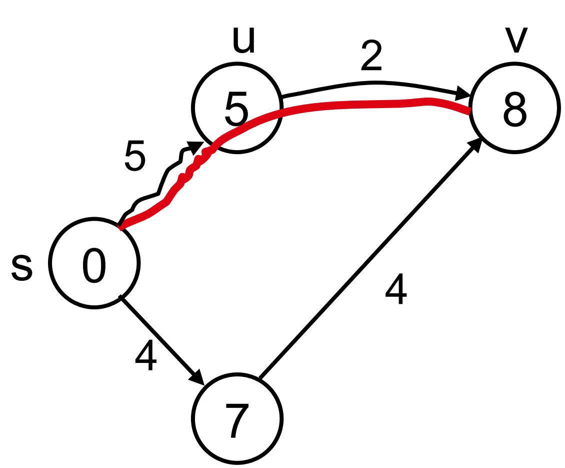
h, i, j not reachable from s

$\boldsymbol{\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty}$

Shortest Path Properties

- **Convergence property**

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



- If $d[v] > \delta(s, v) \Rightarrow$ after relaxation:

  $d[v] = d[u] + w(u, v)$

  $d[v] = 5 + 2 = 7$

  - Otherwise, the value remains unchanged, because it must have been the shortest path value

Shortest Path Properties

- **Path relaxation property**

  Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k. If we relax, in order, (v_0, v_1), (v_1, v_2), . . . , (v_{k-1}, v_k), even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$.