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)
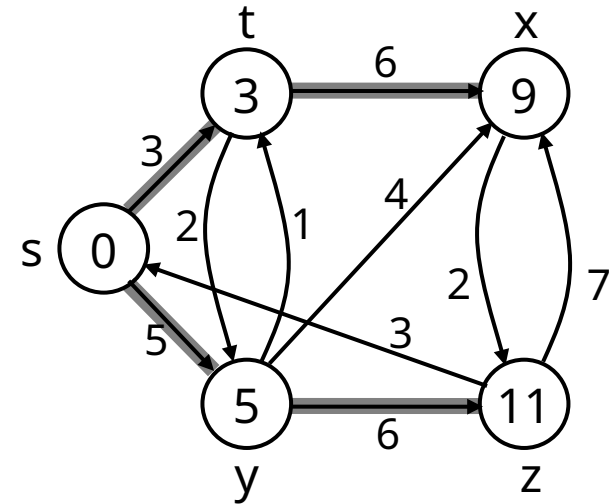 - Weight function w : E → **R**

- **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:

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

- 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 $w: E \rightarrow R$,
- A shortest path $p = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A subpath of p: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \leq i \leq j \leq k$
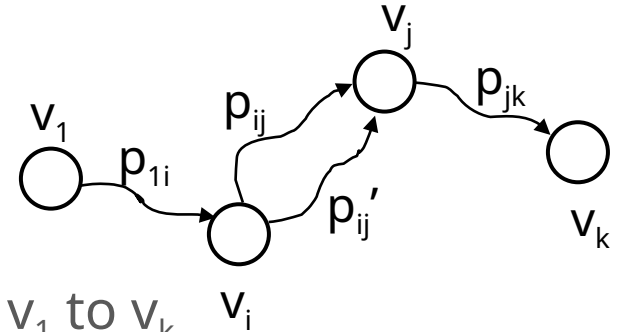
Then: p_{ij} is a shortest path from v_i to v_j

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 → a: only one path

  δ(s, a) = w(s, a) = 3

- s → b: only one path

  δ(s, b) = w(s, a) + w(a, b) = -1

- s → c: infinitely many paths
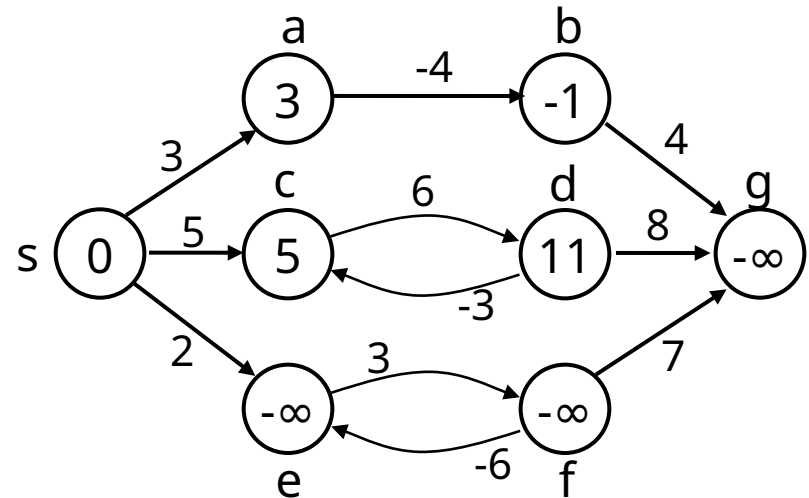
  ⟨s, c⟩, ⟨s, c, d, c⟩, ⟨s, c, d, c, d, c⟩

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

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

What if we have negative-weight edges?

Negative-Weight Edges

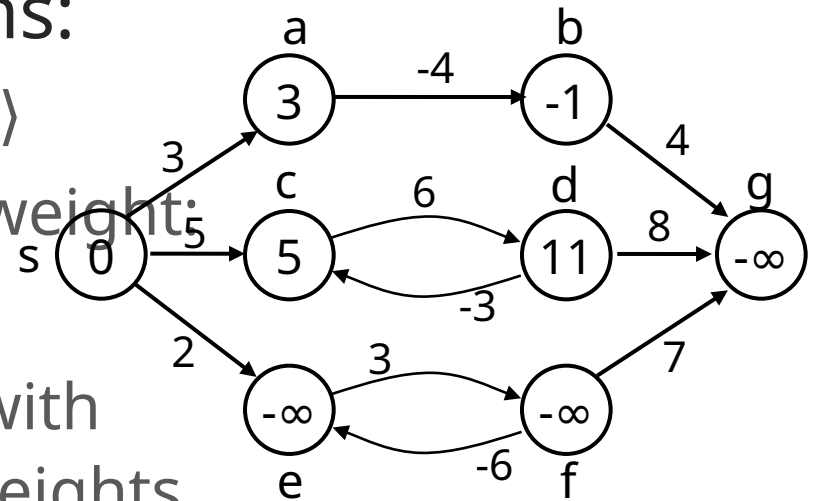- s → e: infinitely many paths:
  ⟨s, e⟩, ⟨s, e, f, e⟩, ⟨s, e, f, e, f, e⟩
  - cycle ⟨e, f, e⟩ has negative weight:
    3 + (- 6) = -3
  - can find paths from s to e with arbitrarily large negative weights
  - δ(s, e) = - ∞ ⟹ no shortest path exists between s and e
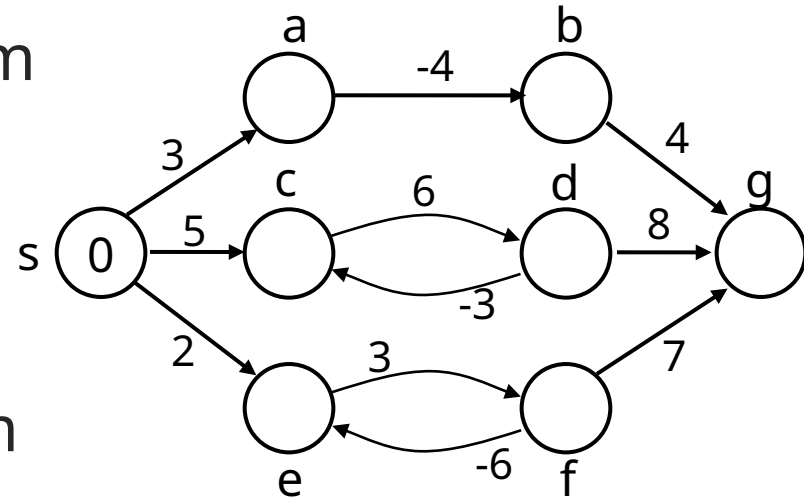- Similarly: δ(s, f) = - ∞,
  δ(s, g) = - ∞

δ(s, h) = δ(s, i) = δ(s, j) = ∞
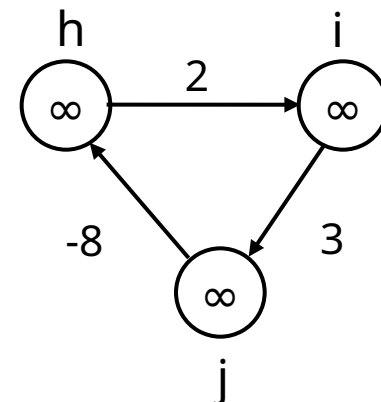
h, i, j not reachable from s

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 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
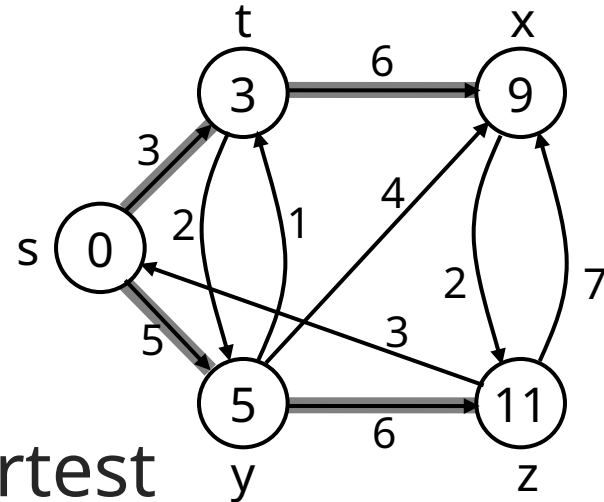
Initialization

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

1. **for** each v ∈ V

2.   **do** d[v] ← ∞

3.   π[v] ← NIL

4. d[s] ← 0

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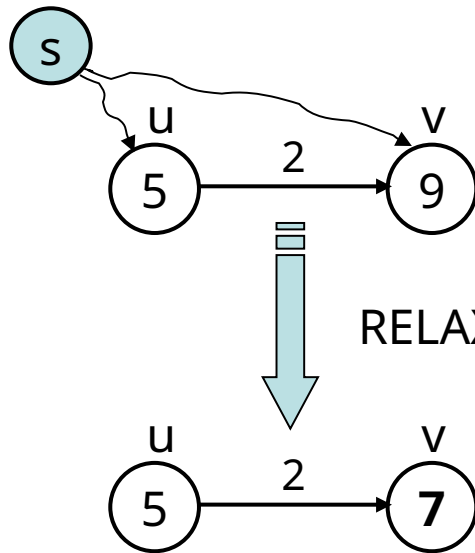
Relaxation

- **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

  $\Rightarrow$ update $d[v]$ and $\pi[v]$
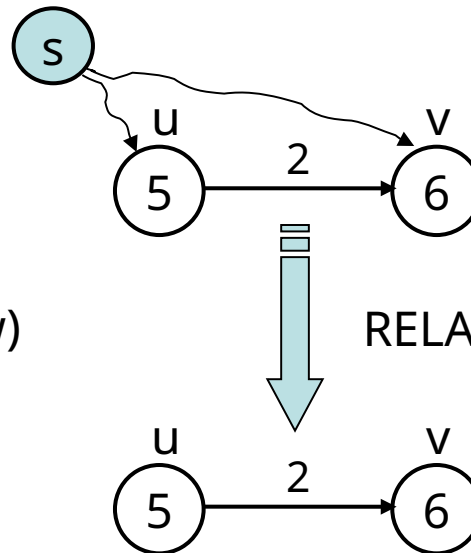
After relaxation:
$d[v] \leq d[u] + w(u, v)$

RELAX(u, v, w)

RELAX(u, v, w)

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 $\Rightarrow$ 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)

1. INITIALIZE-SINGLE-SOURCE(V, s)
2. **for** i ← 1 to |V| - 1
3.   **do for** each edge (u, v) ∈ E
4.   **do** RELAX(u, v, w)
5. **for** each edge (u, v) ∈ E
6.   **do if** d[v] > d[u] + w(u, v)
7.   **then return** FALSE
8. **return** TRUE

Running time: Θ(VE)

E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)
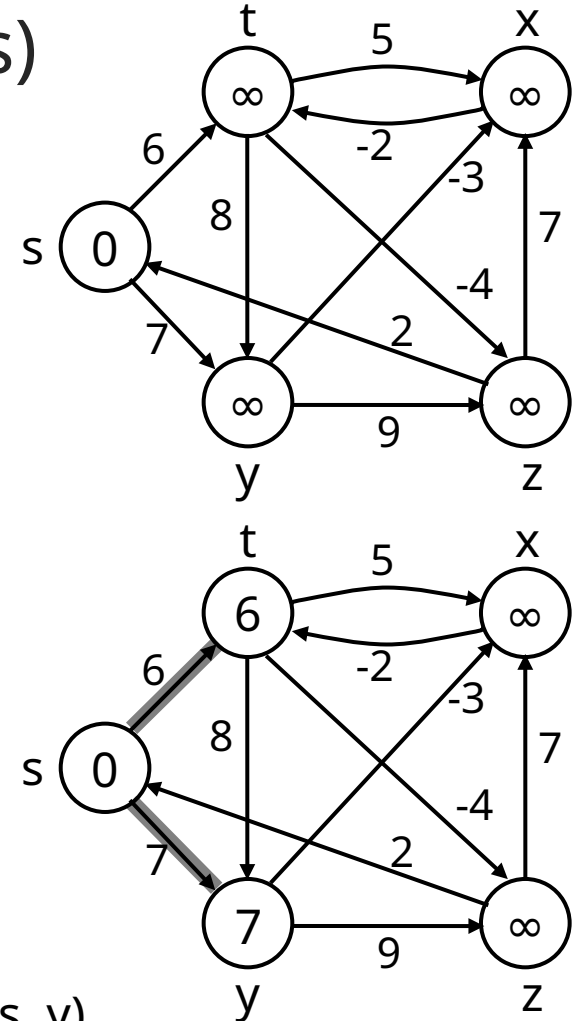
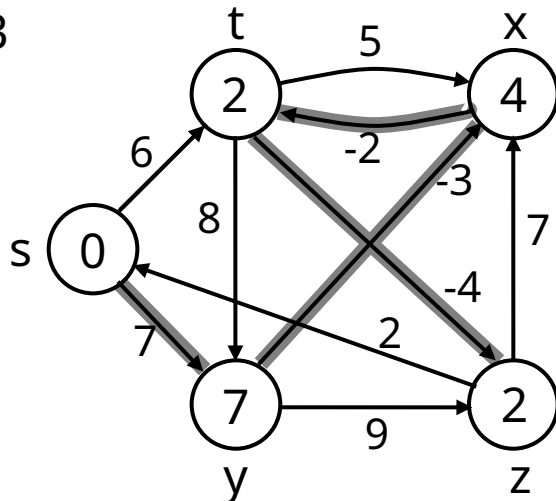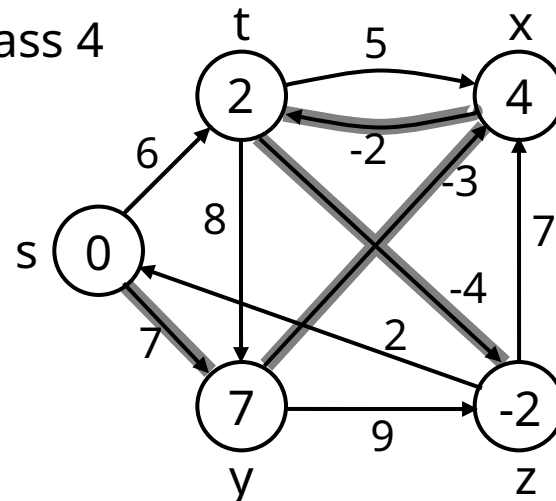# Example   (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)
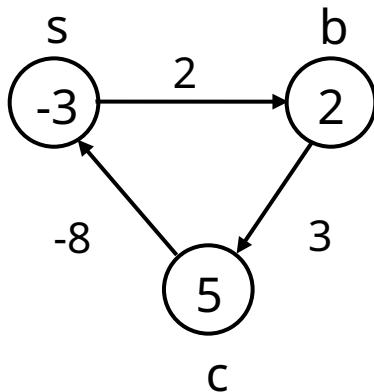
Pass 1

t 6   5   x ∞
     -2
6   8   -3   7
s 0   -4
7   2
   7   9   ∞
y   z

Pass 2

t 6   5   x 4
     -2
6   8   -3   7
s 0   -4
7   2
   7   9   2
y   z

Pass 3

t 2   5   x 4
     -2
6   8   -3   7
s 0   -4
7   2
   7   9   2
y   z

Pass 4

t 2   5   x 4
     -2
6   8   -3   7
s 0   -4
7   2
   7   9   -2
y   z
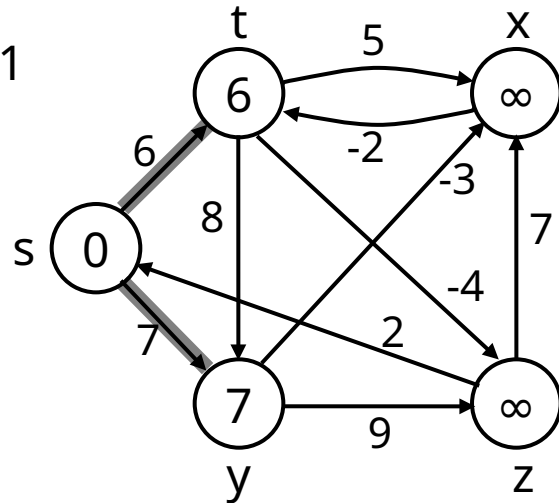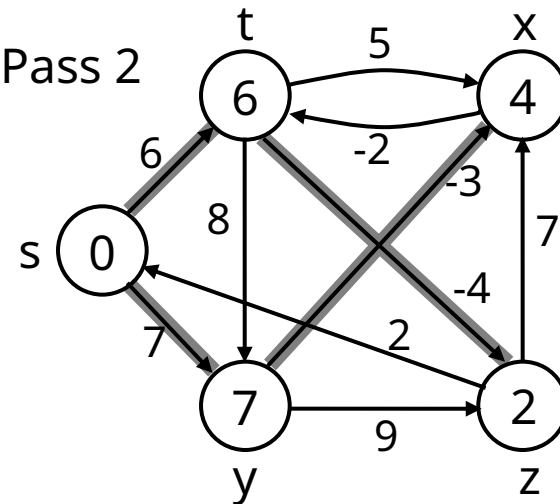
Detecting Negative Cycles

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

s
b
0 —2→ ∞
-8   3
c ∞

s
b
-3 —2→ 2
-8   3
5
c

s
b
-6 —2→ -1
-8   3
2
c
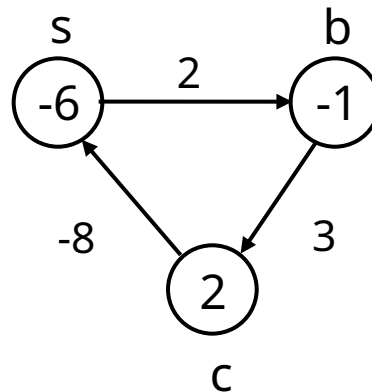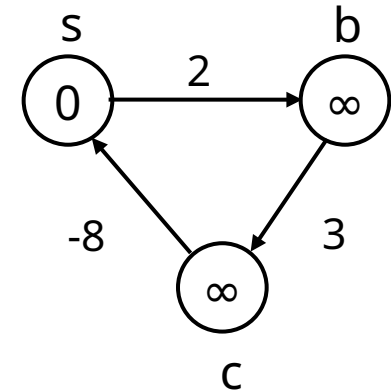
Look at edge (s, b):

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?
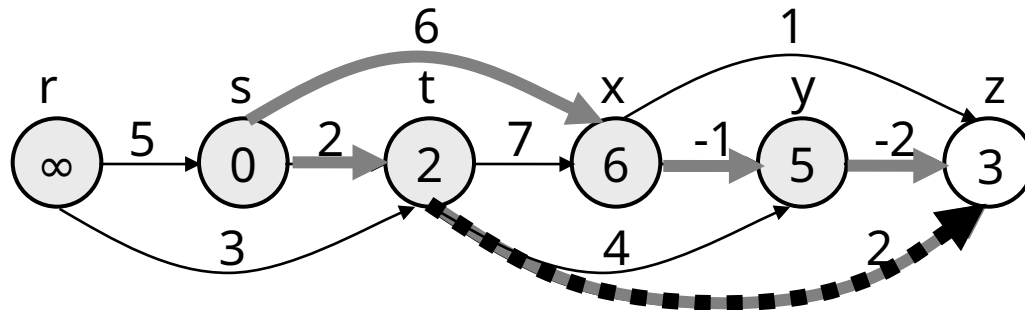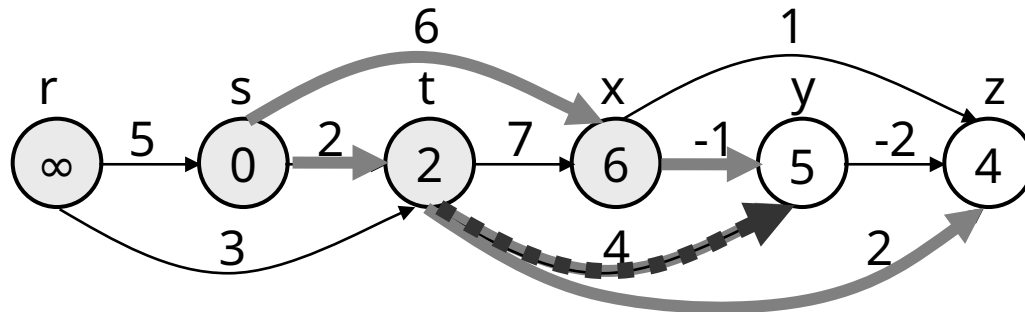 - Yes, (negative-weight) cycles cannot exist
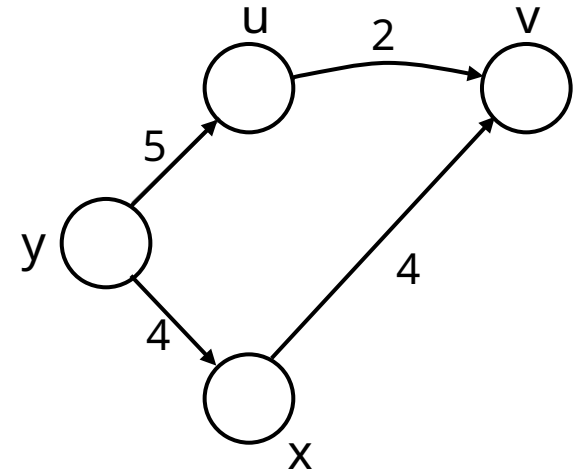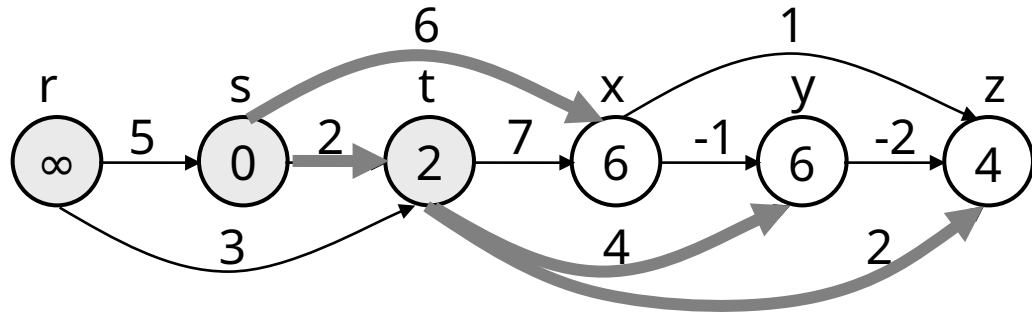
DAG-SHORTEST-PATHS(G, w, s)

1. topologically sort the vertices of G ⟵ Θ(V+E)

2. INITIALIZE-SINGLE-SOURCE(V, s) ⟵ Θ(V)

3. **for** each vertex u, taken in   Θ(V) ⎫
    topologically sorted order   ⎬ Θ(E)

4.   **do for** each vertex v ∈ Adj[u]   ⎪

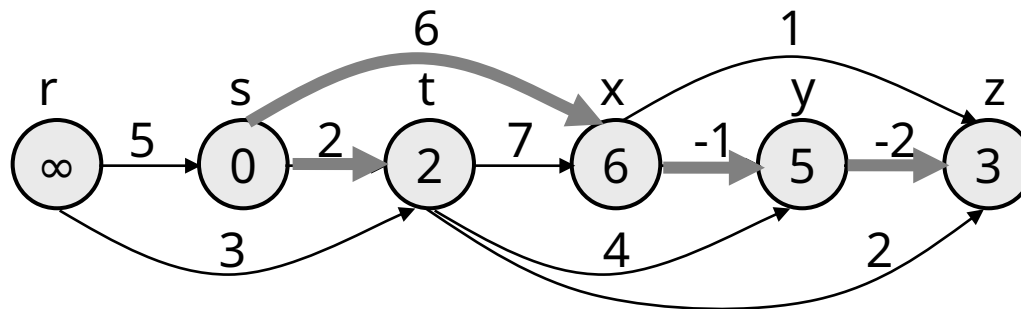5.   **do** RELAX(u, v, w)   ⎭

Running time: Θ(V+E)
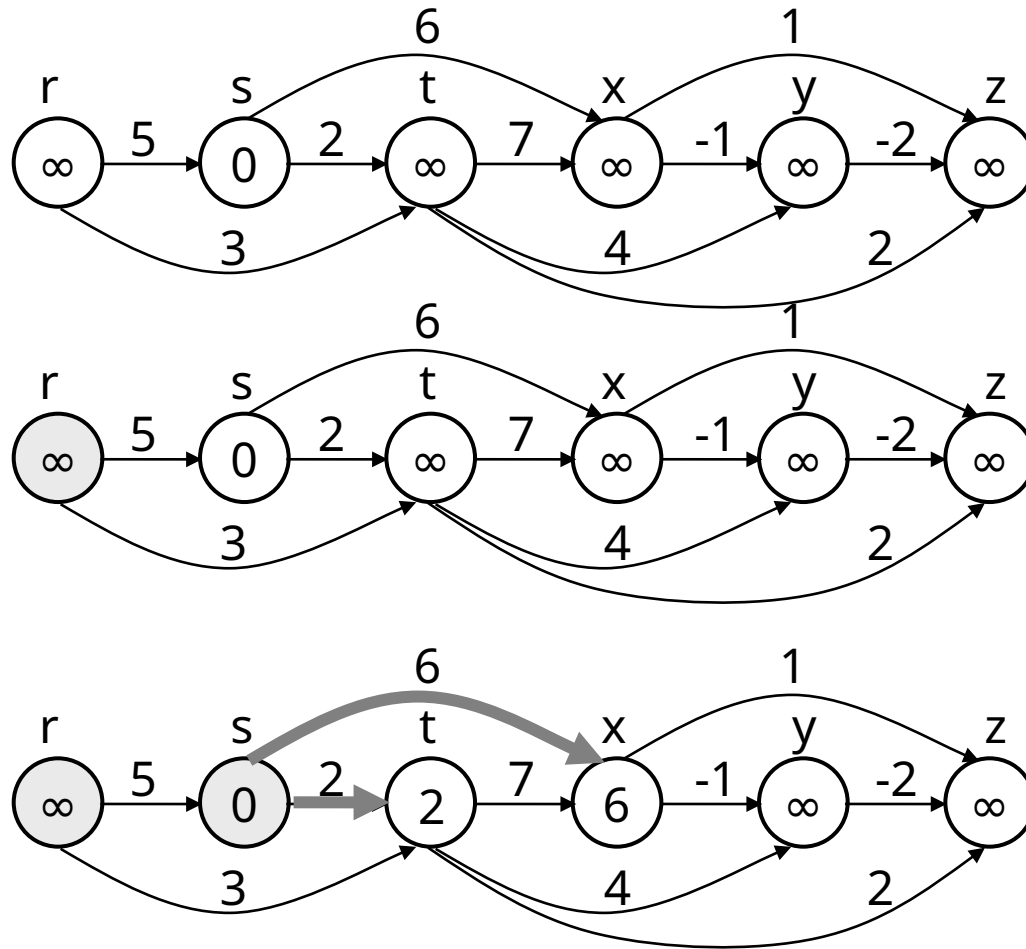
Example

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)

1. INITIALIZE-SINGLE-SOURCE(V, s)
2. S ← ∅
3. Q ← V[G]
4. **while** Q ≠ ∅
5.   **do** u ← EXTRACT-MIN(Q)
6.   S ← S ∪ {u}
7.   **for** each vertex v ∈ Adj[u]
8.   **do** RELAX(u, v, w)
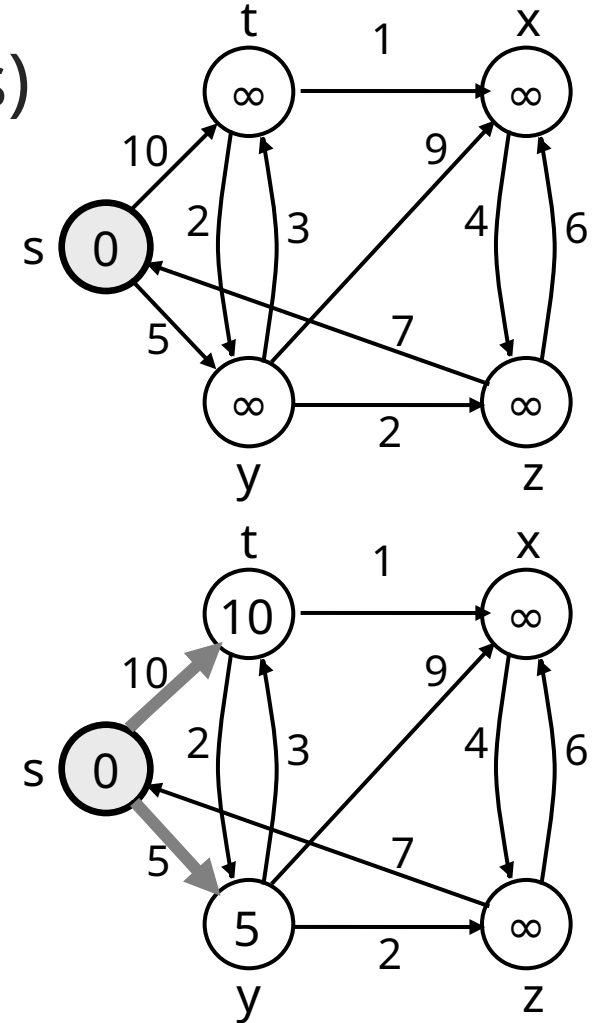
Example
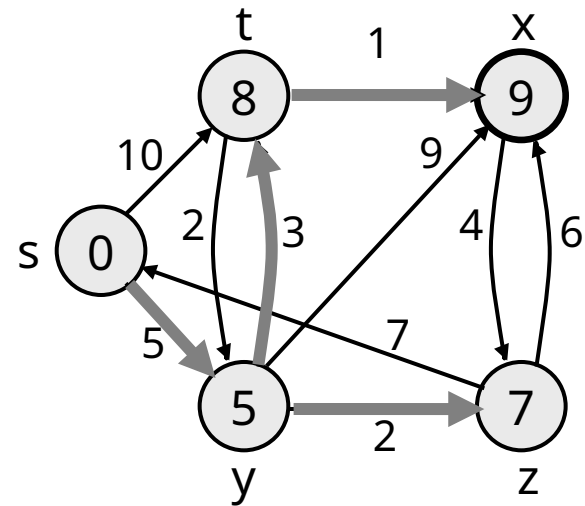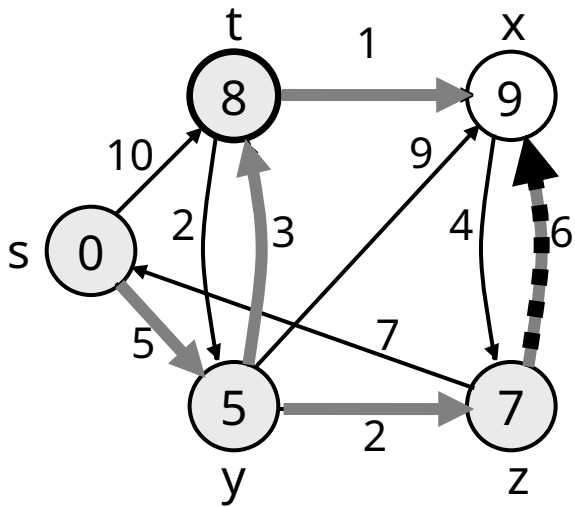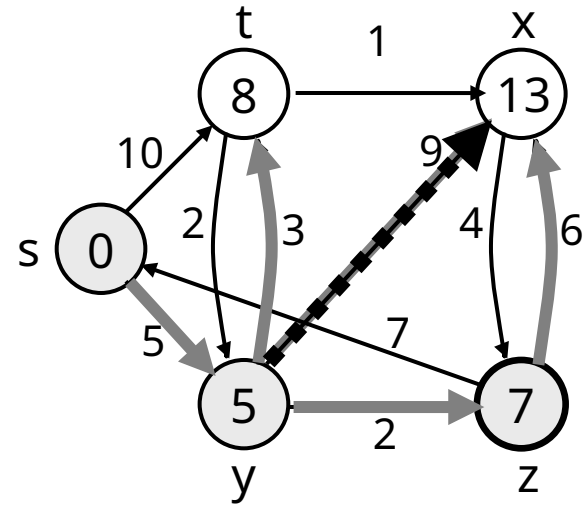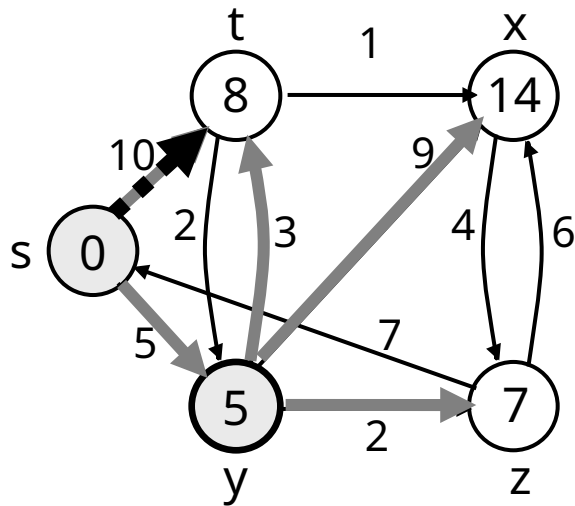
Dijkstra (G, w, s)

1. INITIALIZE-SINGLE-SOURCE(V, s) ⟵ Θ(V)

2. S ← ∅

3. Q ← V[G] ⟵ O(V) build min-heap

4. **while** Q ≠ ∅ ⟵ Executed O(V) times

5.   **do** u ← EXTRACT-MIN(Q) ⟵ O(lgV)

6.   S ← S ⋃ {u}

7.   **for** each vertex v ∈ Adj[u]

8.   **do** RELAX(u, v, w) ⟵ O(E) times; O(lgV)

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

Readings
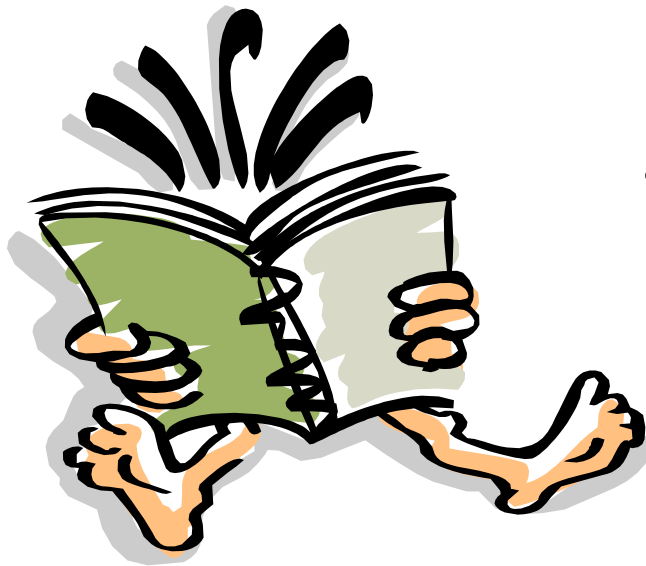
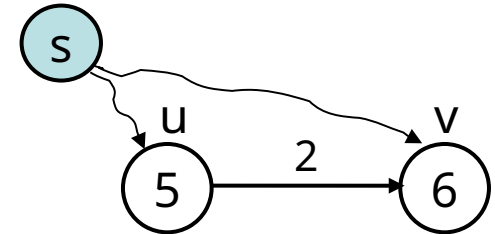- Chapters 25, 31
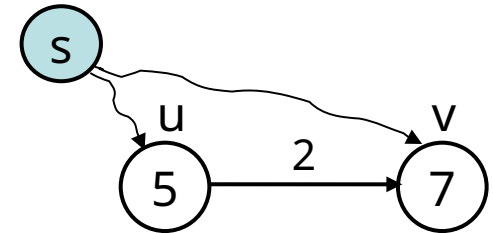
optional

ADDITIONAL SLIDES

Shortest Path Properties

- **Triangle inequality**

  For all $(u, v) \in E$, we have:

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

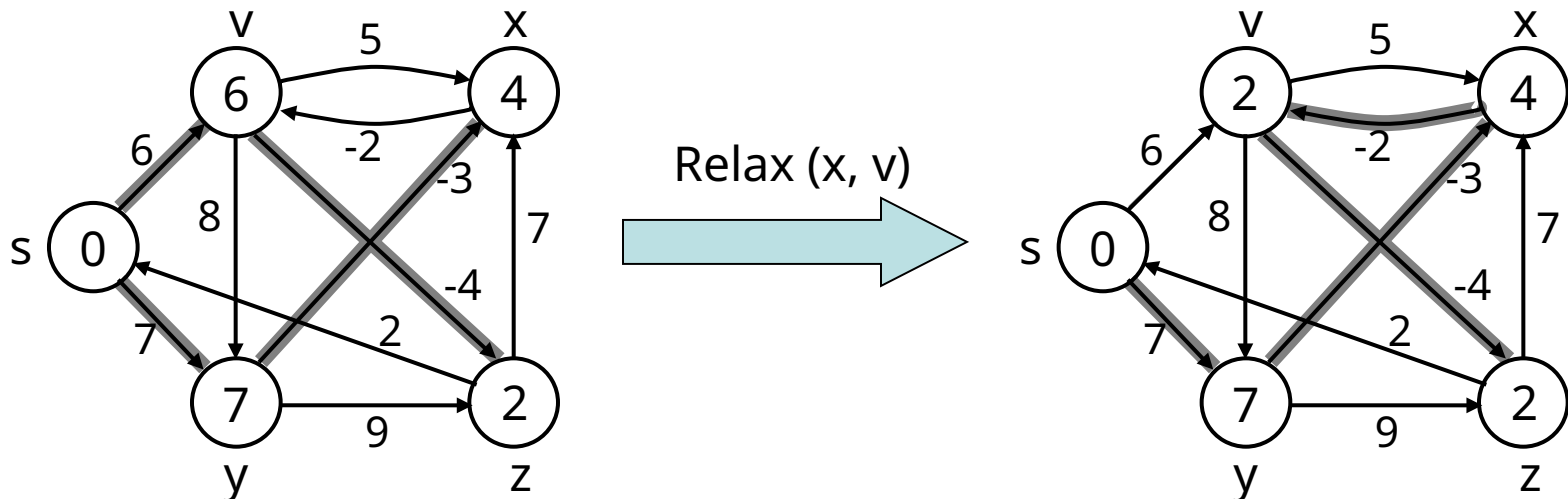- If u is on the shortest path to v we have the equality sign

Shortest Path Properties

- **Upper-bound property**

We always have d[v] ≥ δ(s, v) for all v.

Once d[v] = δ(s, v), it never changes.

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

Shortest Path Properties
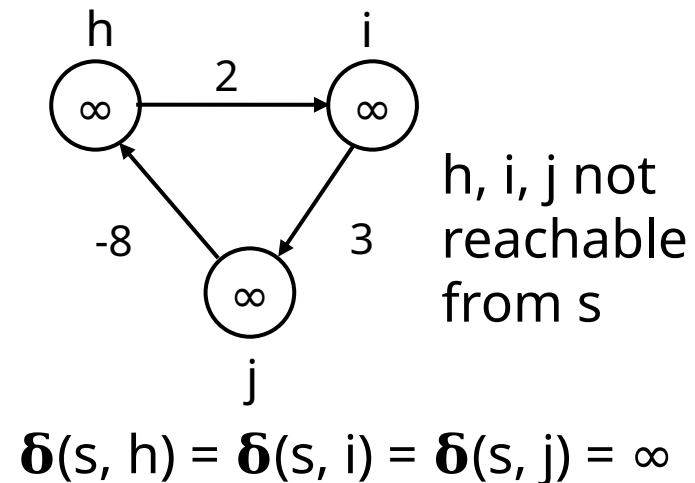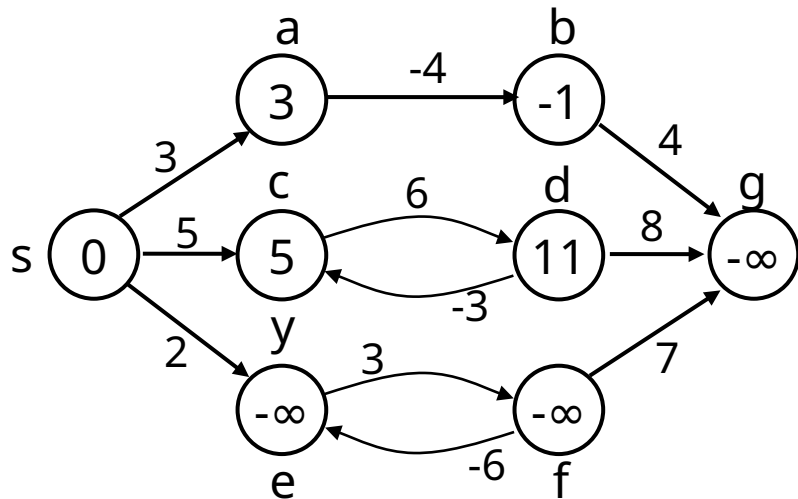
- **No-path property**

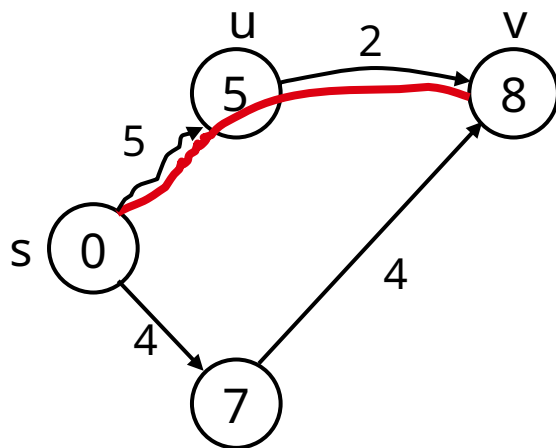 If there is no path from s to v then d[v] = ∞ always.

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

h, i, j not reachable from s

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

Shortest Path Properties

- **Convergence property**

  If s ⤳ u → v is a shortest path, and if d[u] = δ(s, u)
  at any time prior to relaxing edge (u, v), then
  d[v] = δ(s, v) at all times afterward

- If d[v] > δ(s, v) ⇒ 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 = ⟨v_0, v_1, . . . , v_k⟩ 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] = δ(s, v_k).