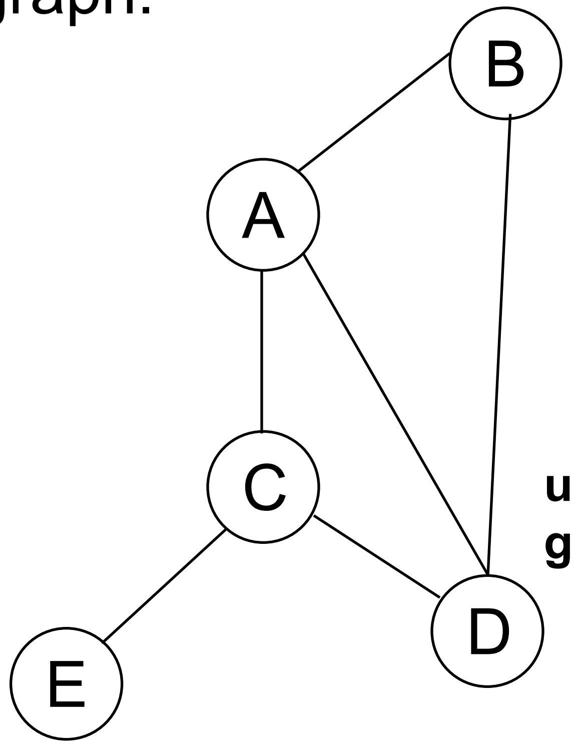
CSE 2202
Design and Analysis of Algorithms – I

Single Source Shortest Path
(Dijkstra and Bellman Ford)

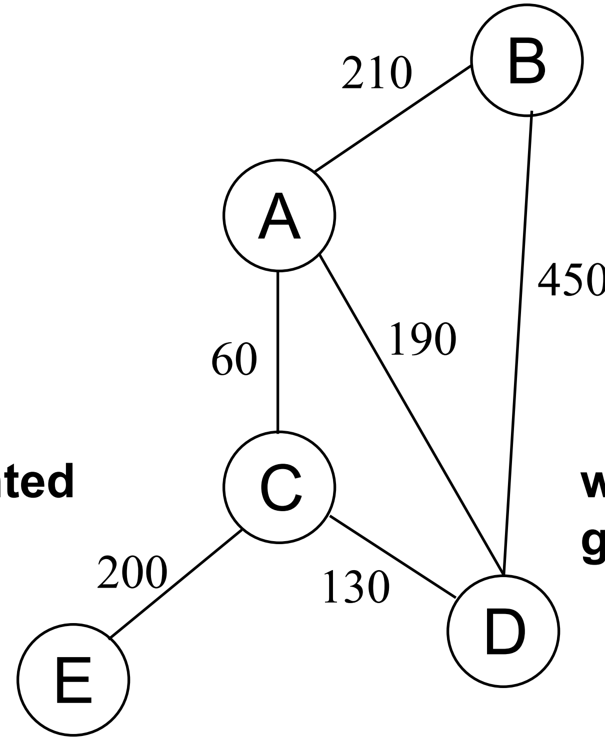
SINGLE SOURCE SHORTEST PATH(DIJKSTRA'S ALGORITHM)

Shortest Path Problems

- **What is shortest path ?**
  - <u>shortest length between two vertices</u> for an unweighted graph:
  - <u>smallest cost between two vertices</u> *for* a weighted graph:



unweighted graph



weighted graph

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:

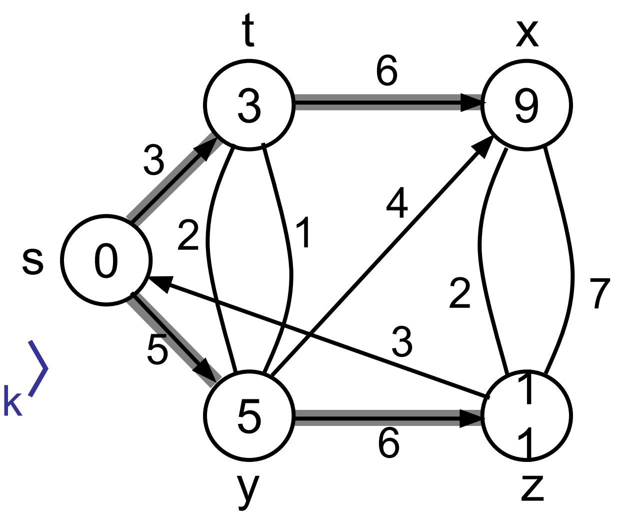
    vertices = cities

    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 = ⟨v_0, v_1, . . . , v_k⟩

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

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

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

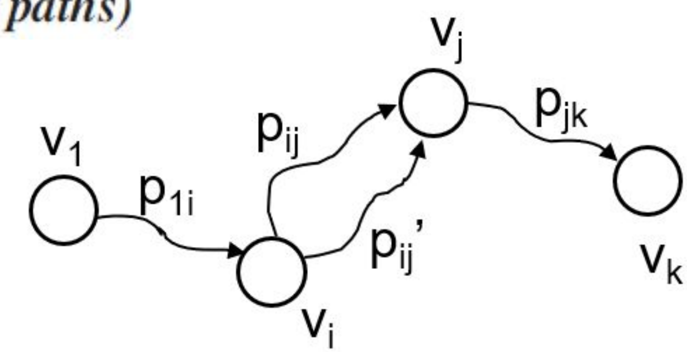
- Shortest path u to v is any path p such that w(p) = δ(u, v)

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
  - 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

Lemma 24.1 (Subpaths of shortest paths are shortest paths)



Given:

- A weighted, directed graph $G = (V, E)$
- A weight function $w: E \rightarrow \mathbf{R}$,
- 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: $p = v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$

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

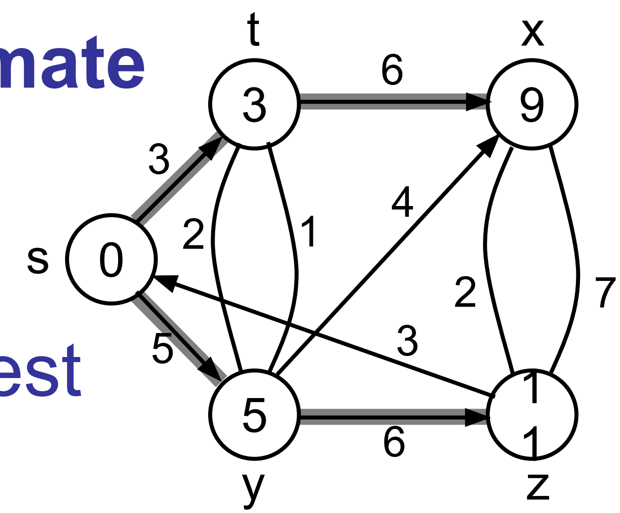
Assume $\exists\, p'_{ij}$ from v_i to v_j with $w(p'_{ij}) < w(p_{ij})$

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

Shortest-Path Representation

For each vertex v ∈ V:

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



Initialization

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$

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

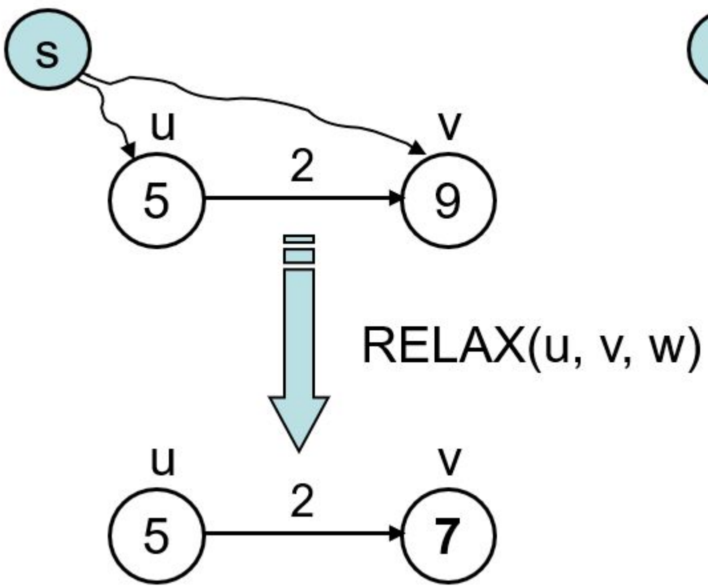
After initialization, we have $v.\pi = \text{NIL}$ for all $v \in V$, $s.d = 0$, and $v.d = \infty$ for $v \in V - \{s\}$.

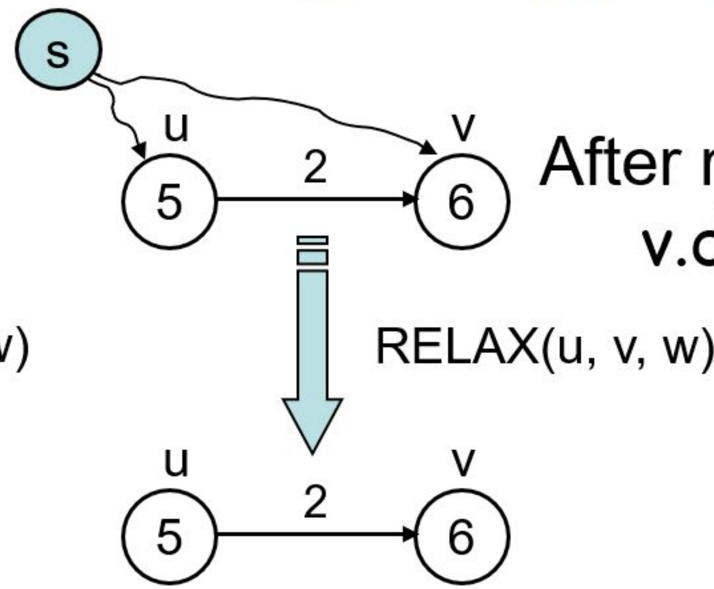
Relaxation

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

RELAX(u, v, w)

```
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```





After relaxation:

$v.d \le u.d + w(u, v)$

RELAX(u, v, w)

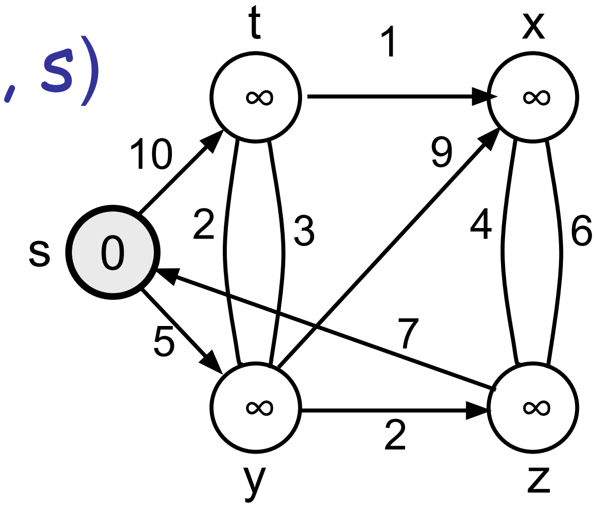
- All the single-source shortest-paths algorithms
  - start by calling INIT-SINGLE-SOURCE
  - then relax edges
- The algorithms differ in the order and how many times they relax each edge

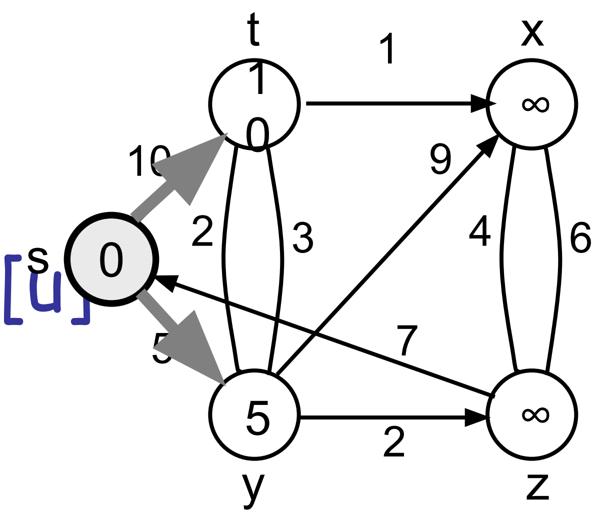
Dijkstra's Algorithm

- 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 (v.d)
- Repeatedly select a vertex $u \in V - S$, with the minimum shortest-path estimate v.d

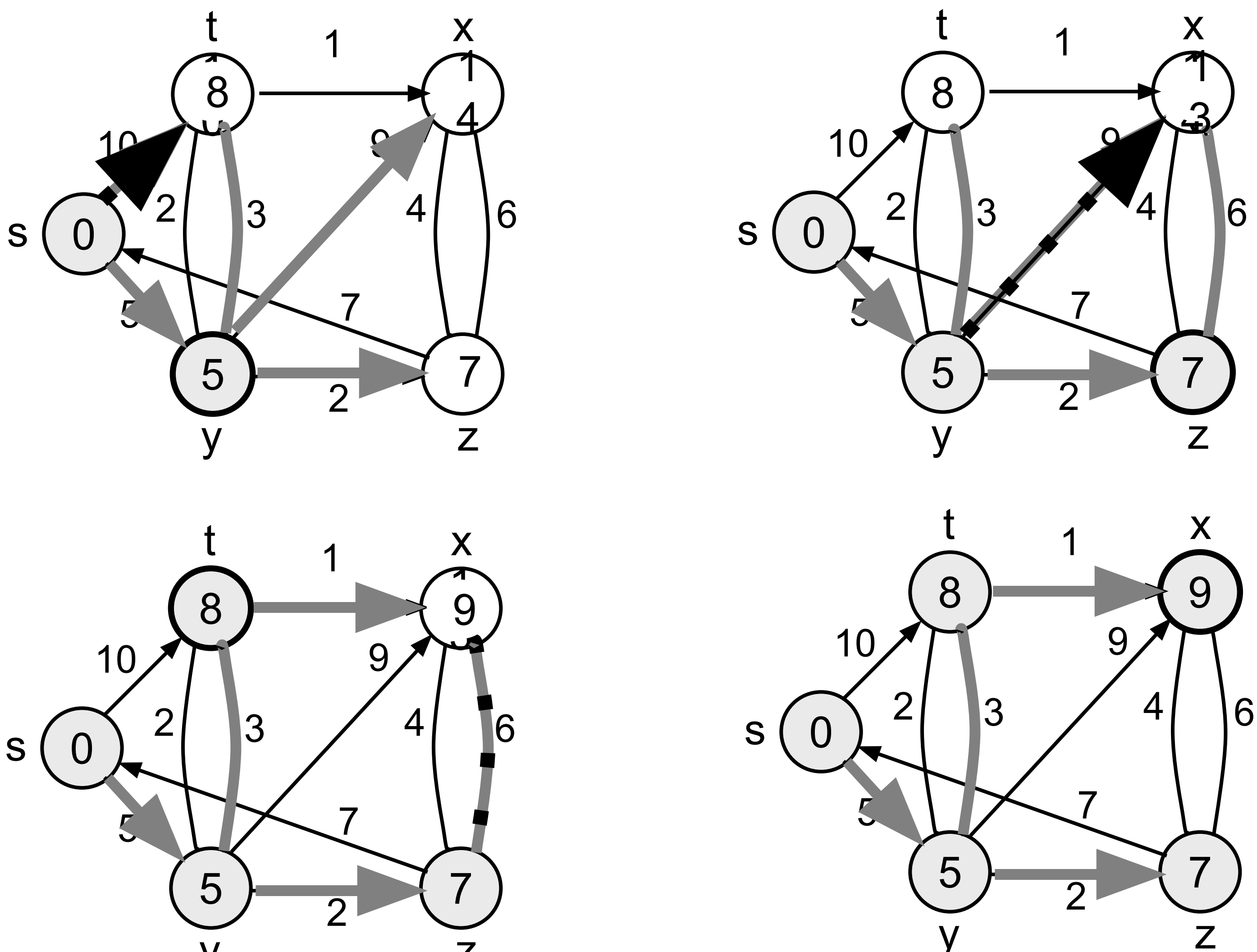
Dijkstra (G, w, s)

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





Example



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

Dijkstra's Pseudo Code

- Graph *G*, weight function *w*, root *s*

DIJKSTRA(G, w, s)

for each $v \in V$
    do $d[v] \leftarrow \infty$
$d[s] \leftarrow 0$
$S \leftarrow \emptyset$ $\triangleright$ Set of discovered nodes
$Q \leftarrow V$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$

relaxing edges

Dijkstra (G, w, s)

```
1.  INITIALIZE-SINGLE-SOURCE(G, s)      ← Θ(V)
2.  S ← ∅                               never inserts vertices into Q after line 3
3.  Q ← G.V                             ← 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 ∈ G.Adj[u]
8.              do RELAX(u, v, w)       ← O(E) times; O(lgV)
```

Running time: $O(V\lg V + E\lg V) = O(E\lg V)$

Dijkstra's Running Time

- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V|\ T_{\text{Extract-Min}} + |E|\ T_{\text{Decrease-Key}}$
- T depends on different Q implementations

Q	T(Extract-Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$ (amort.)	$O(V \lg V + E)$

Question

- Prove that, if there exists negative edge, dijkstra's shortest path algorithm may fail to find the shortest path
- Print the shortest path for dijkstra's algorithm
- Suppose you are given a graph where each edge represents the path cost and each vertex has also a cost which represents that, if you select a path using this node, the cost will be added with the path cost. How can it be solved using Dijkstra's algorithm?

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

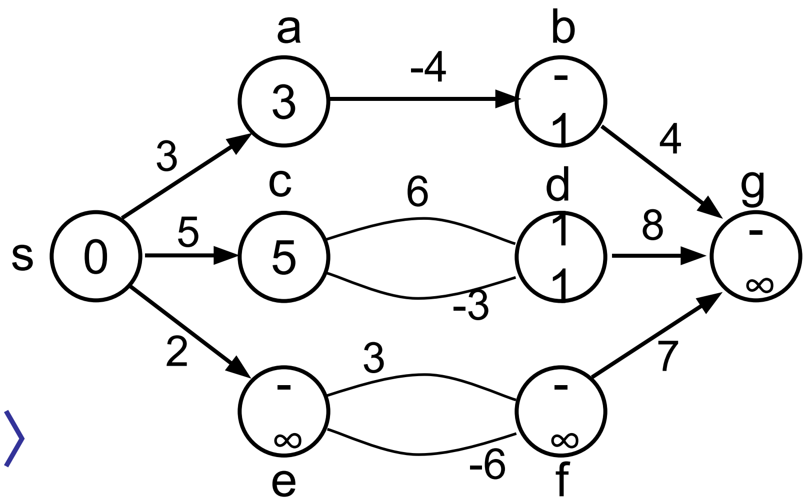
- s ⟶ c: infinitely many paths

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

  cycle has positive weight (6 - 3 = 3)

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

What if we have negative-weight edges?

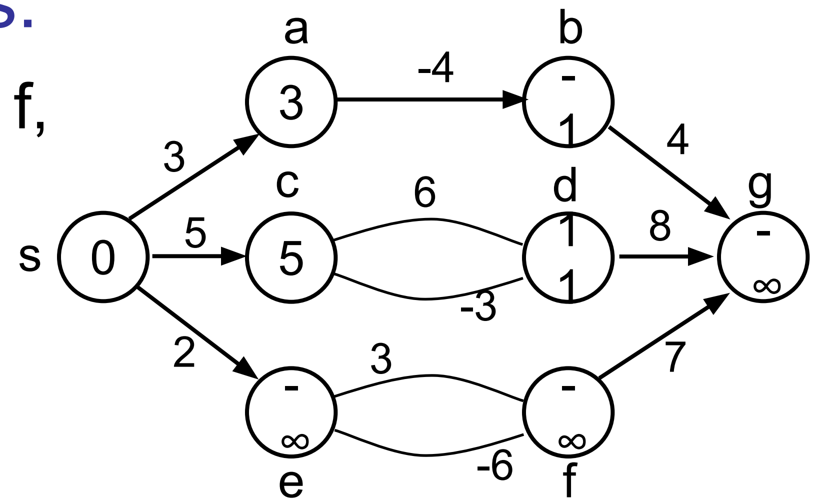


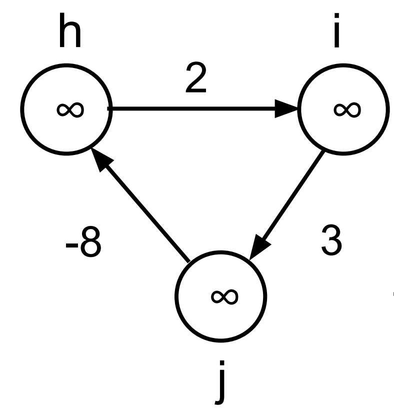
Negative-Weight Edges

- 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) = - ∞



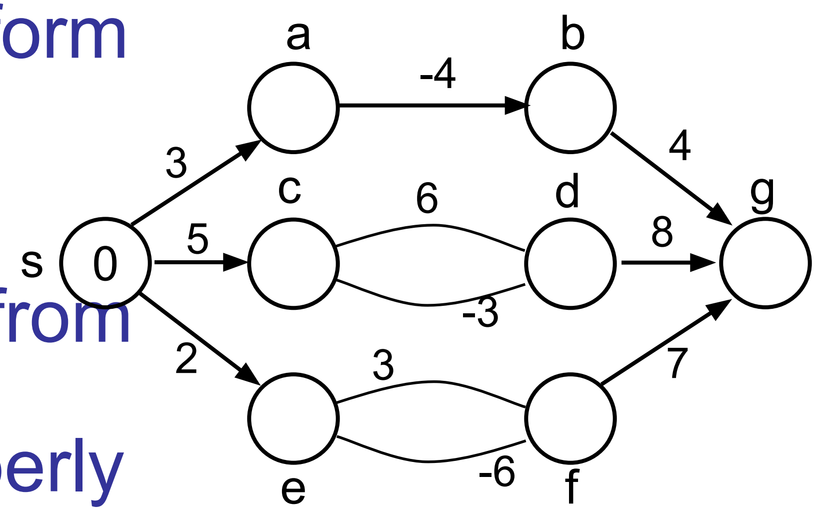


h, i, j not reachable from s

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
  - Keep going around the cycle, and get $w(s, v) = -\infty$ 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
- We will assume that when we are finding shortest paths, the paths will have no cycles

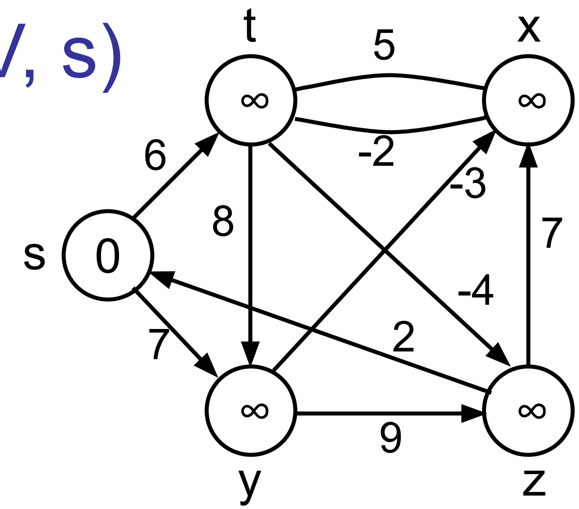
BELLMAN FORD

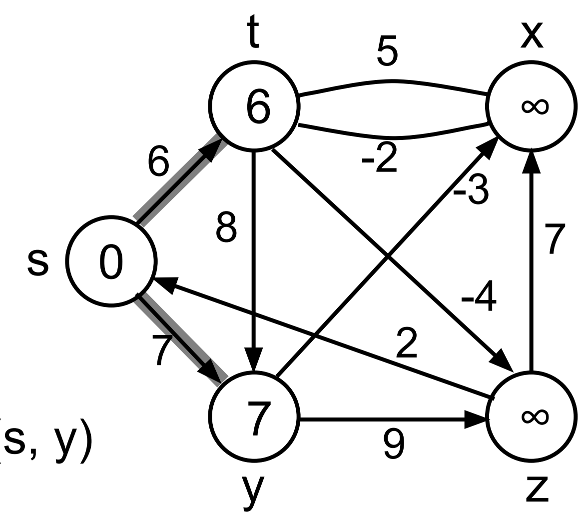
Bellman-Ford Algorithm

- Single-source shortest paths problem
  - Computes v.d and v.π for all v ∈ V
- Allows negative edge weights
- 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
  - This is because, in the **worst-case scenario**, any vertex's path length can be changed N times to an even shorter path length.

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
```



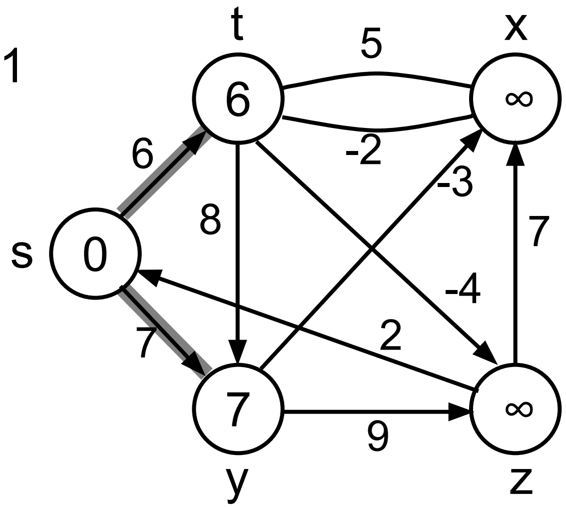


E: (t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)

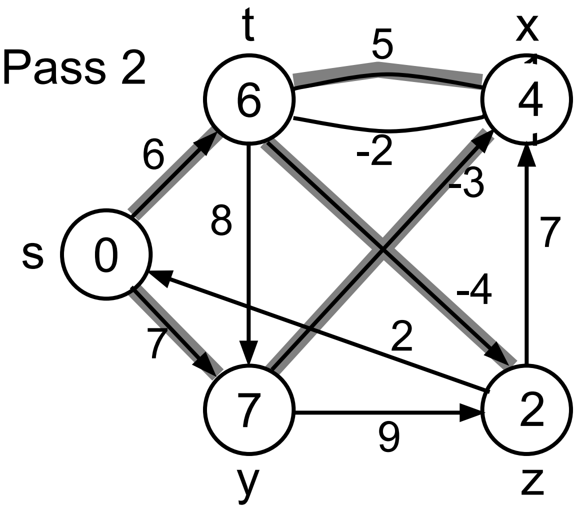
Example

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

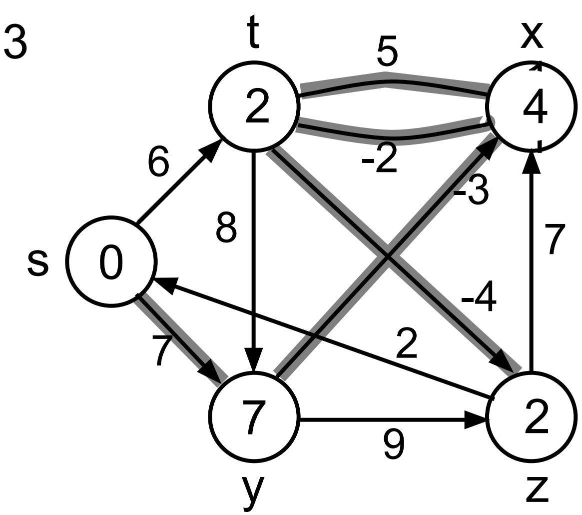
Pass 1



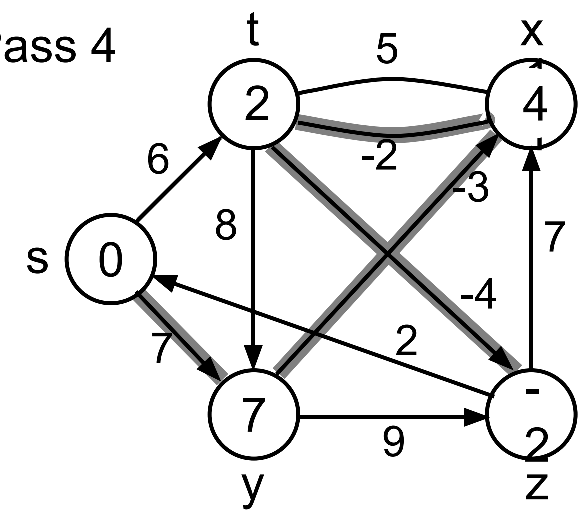
Pass 2



Pass 3

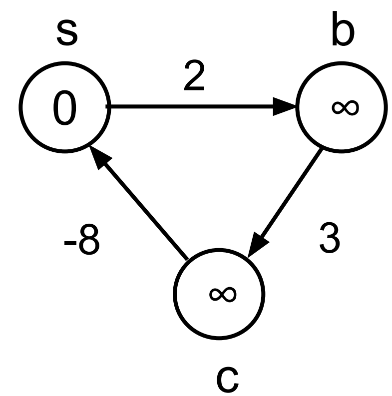


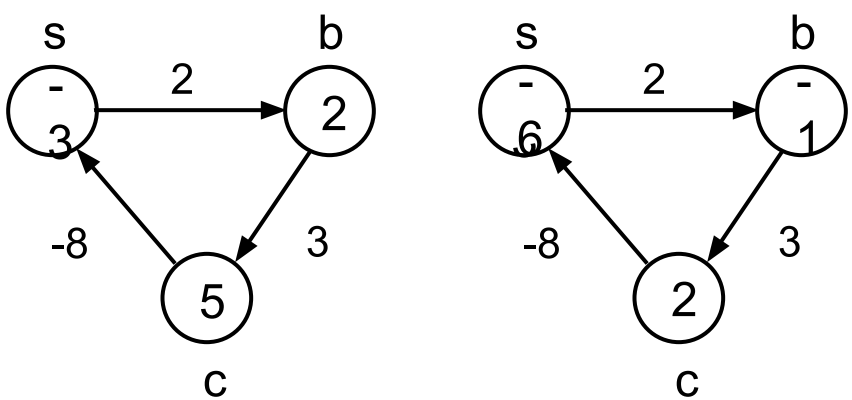
Pass 4



Detecting Negative Cycles

- **for** each edge $(u, v) \in E$
- **do if** $v.d > u.d + w(u, v)$
- **then return** FALSE
- **return** TRUE





Look at edge (s, b):

$b.d = -1$
$s.d + w(s, b) = -4$

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

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

```
INITIALIZE-SINGLE-SOURCE(V, s)          ← Θ(V)
for i ← 1 to |G.V| - 1                  ← O(V)  ┐
    do for each edge (u, v) ∈ G.E       ← O(E)  ├ O(VE)
        do RELAX(u, v, w)                       ┘
for each edge (u, v) ∈ G.E              ← O(E)
    do if v.d > u.d + w(u, v)
        then return FALSE
return TRUE
```

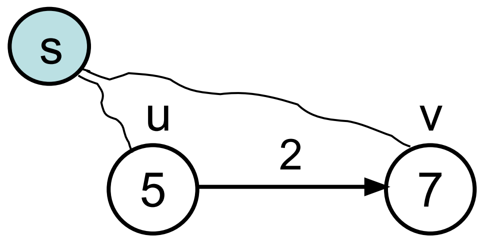
Running time: O(VE)

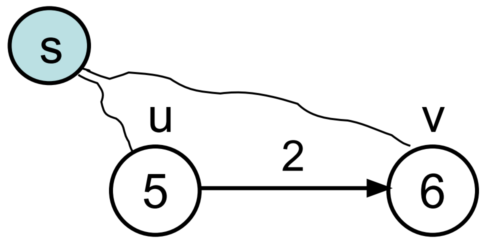
Shortest Path Properties

- **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

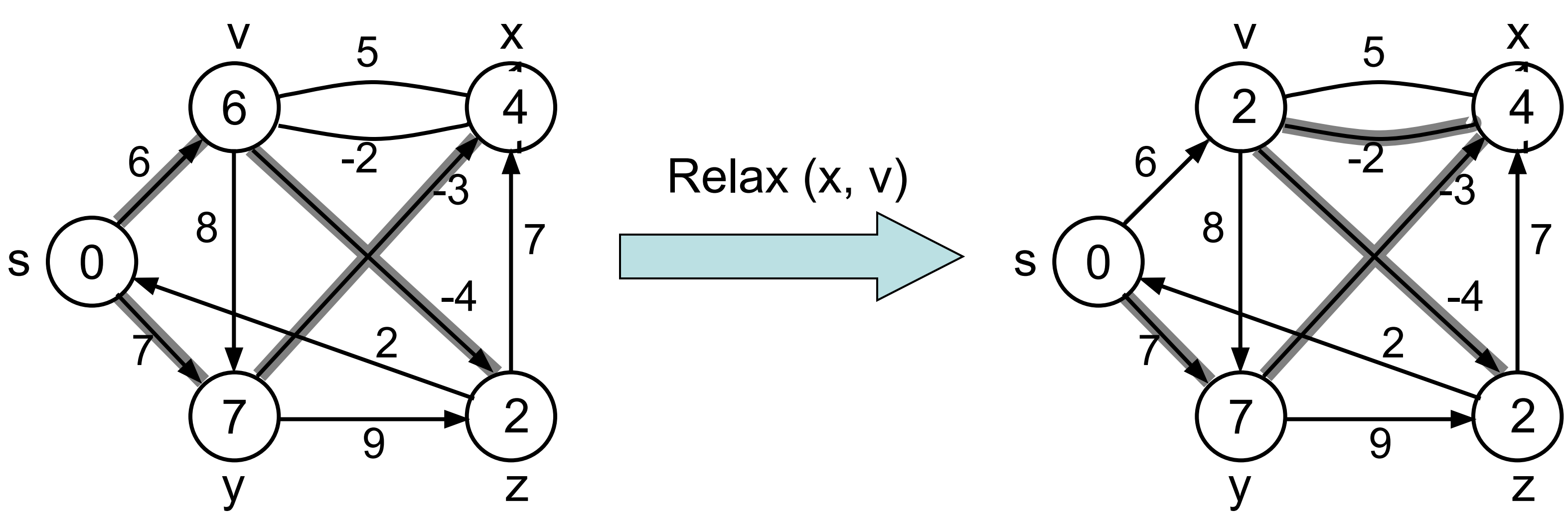
Shortest Path Properties

- **Upper-bound property**

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

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

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

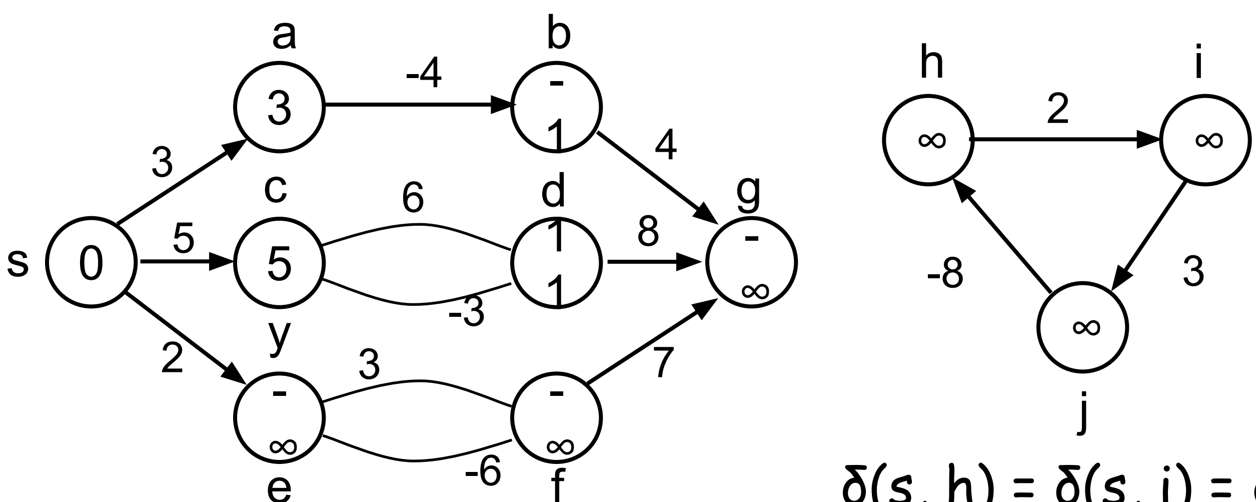


Shortest Path Properties

- **No-path property**

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

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



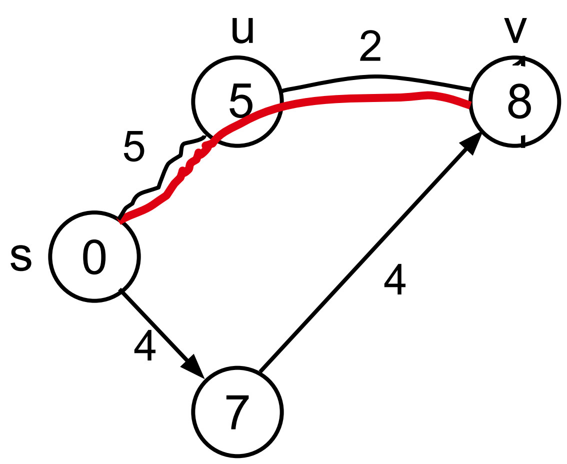
h, i, j not reachable from s

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

Shortest Path Properties

- **Convergence property**

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



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

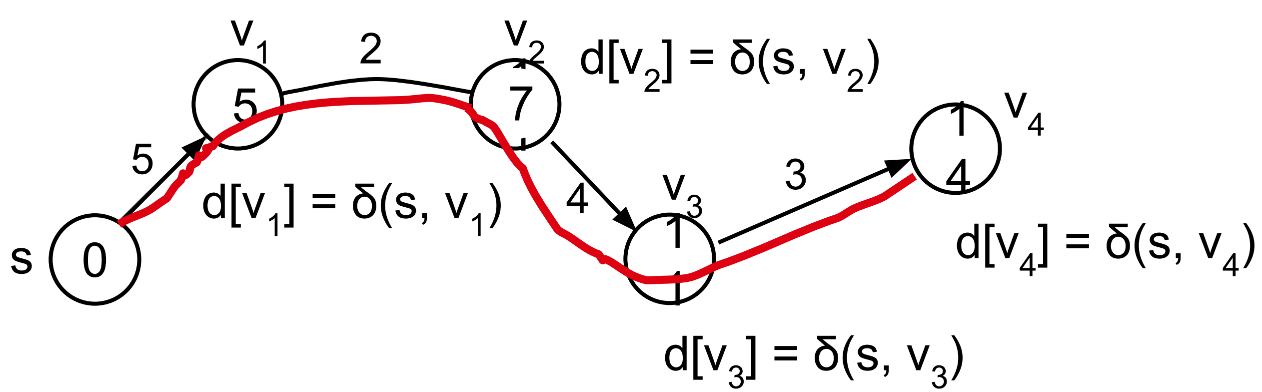
  $v.d = u.d + w(u, v)$

  $v.d = 5 + 2 = 7$
- Otherwise, the value remains unchanged, because it must have been the shortest path value

Shortest Path Properties

- **Path relaxation property**

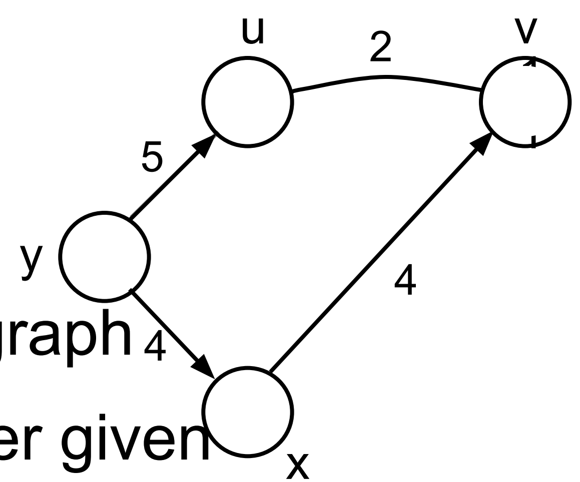
  Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k. If we relax, in order, (v_0, v_1), (v_1, v_2), . . . , (v_{k-1}, v_k), even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$.



SINGLE-SOURCE SHORTEST PATHS IN DAGS

Single-Source Shortest Paths in DAGs

- Given a weighted 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, (negative-weight) cycles cannot exist

In such setting, we can compute shortest paths from a single source in time:

$$\Theta(V + E)$$

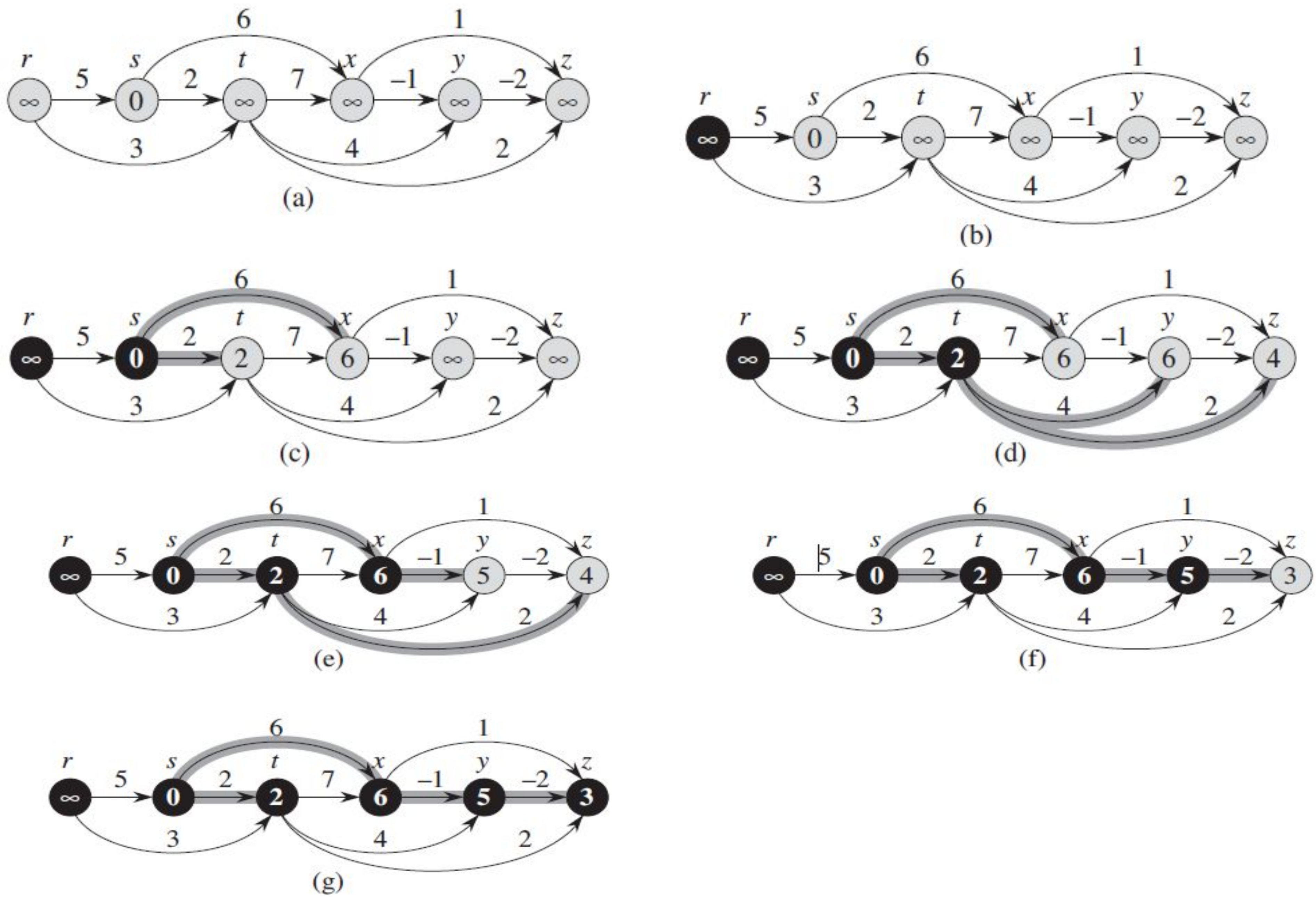
DAG-SHORTEST-PATHS(G, w, s)

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 ∈ *G.Adj[u]*
5. **do** RELAX(u, v, w)

$\Theta(E)$

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

We have used an aggregate analysis here



The newly blackened vertex in each iteration was used as u in that iteration.

Readings

- Chapter 24
- Exercise
  - 24.1-6 – Find negative cycle
  - 24.2-4 – Total Number of paths in a DAG
- Difficult Problems (Solve these if you want):
  - 24.3-6 modify dijkstra
  - 24-2 – nesting boxes
  - 24-3 - Arbitrage
  - 24.6 – Bitonic Shortest path