

Algorithms

Lecture Topic: Single-Source Shortest Paths

Anxiao (Andrew) Jiang

Roadmap of this lecture:

1. Single-Source Shortest Paths

1.1 Define the "Single-Source Shortest Paths Problem".

1.2 Bellman-Ford Algorithm for finding single-source shortest paths.

1.3 Correctness of the single-source shortest paths algorithm.

1.4 Finding single-source shortest paths in a directed acyclic graph.

Single-Source Shortest Paths

Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node.

Single-Source Shortest Paths

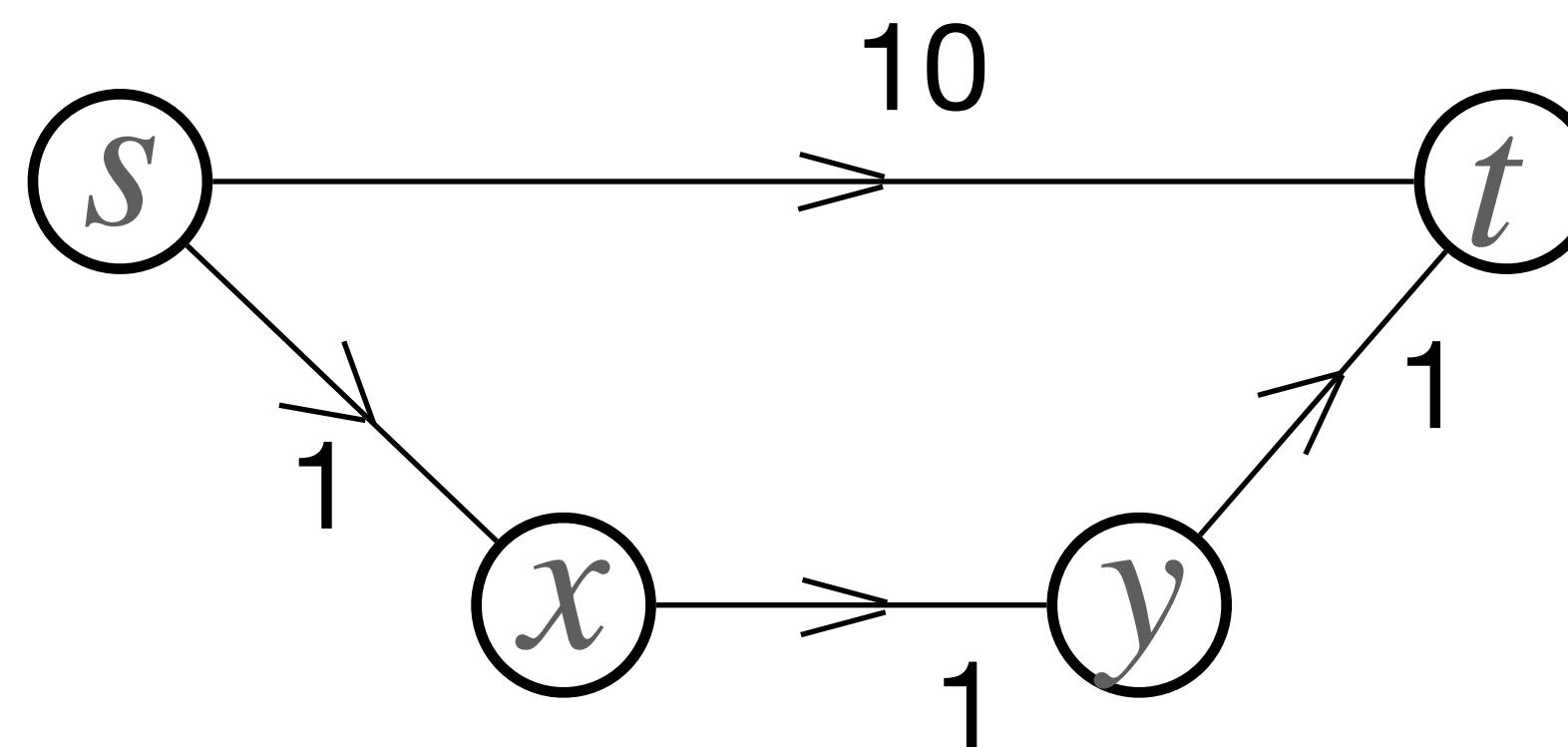
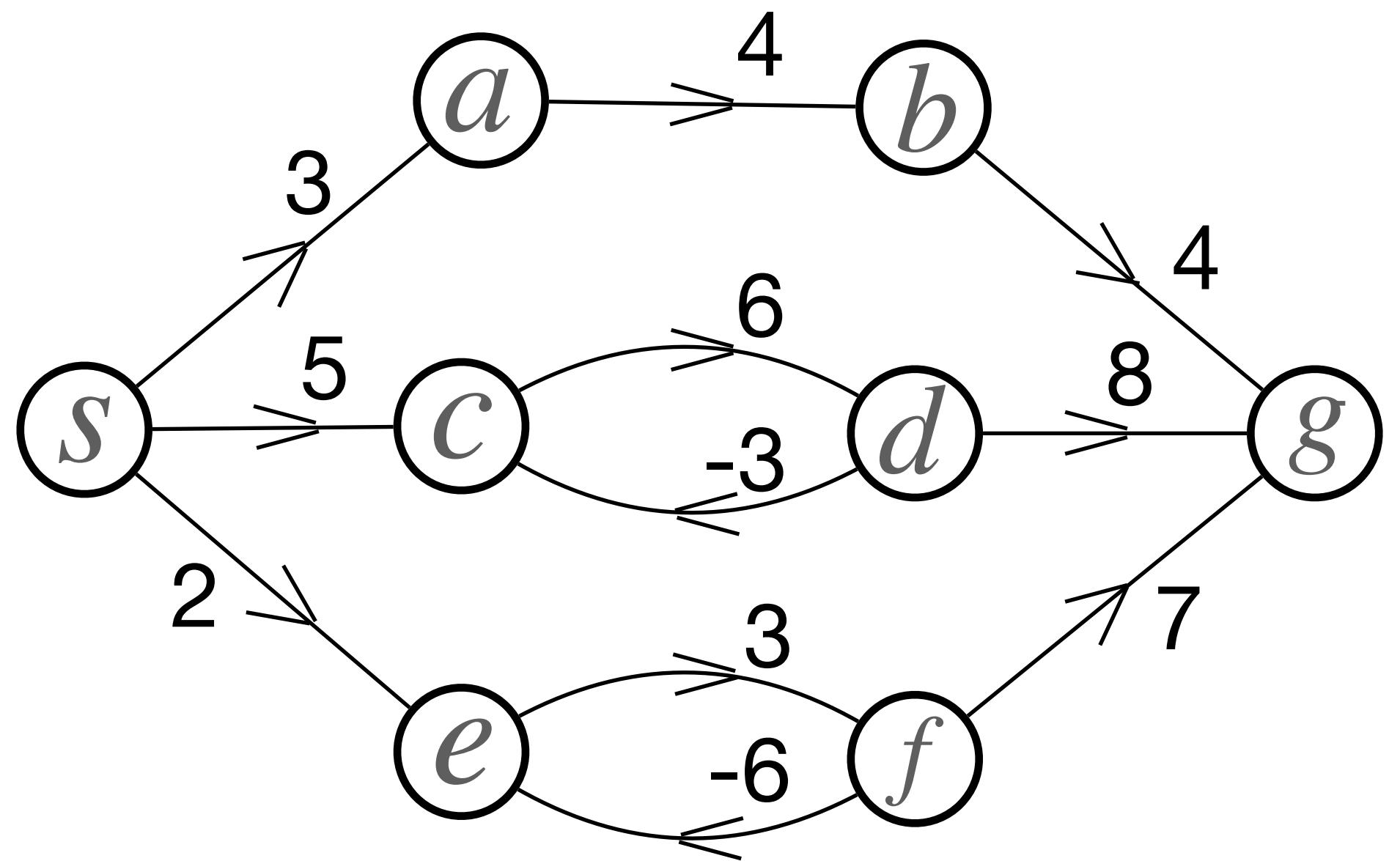
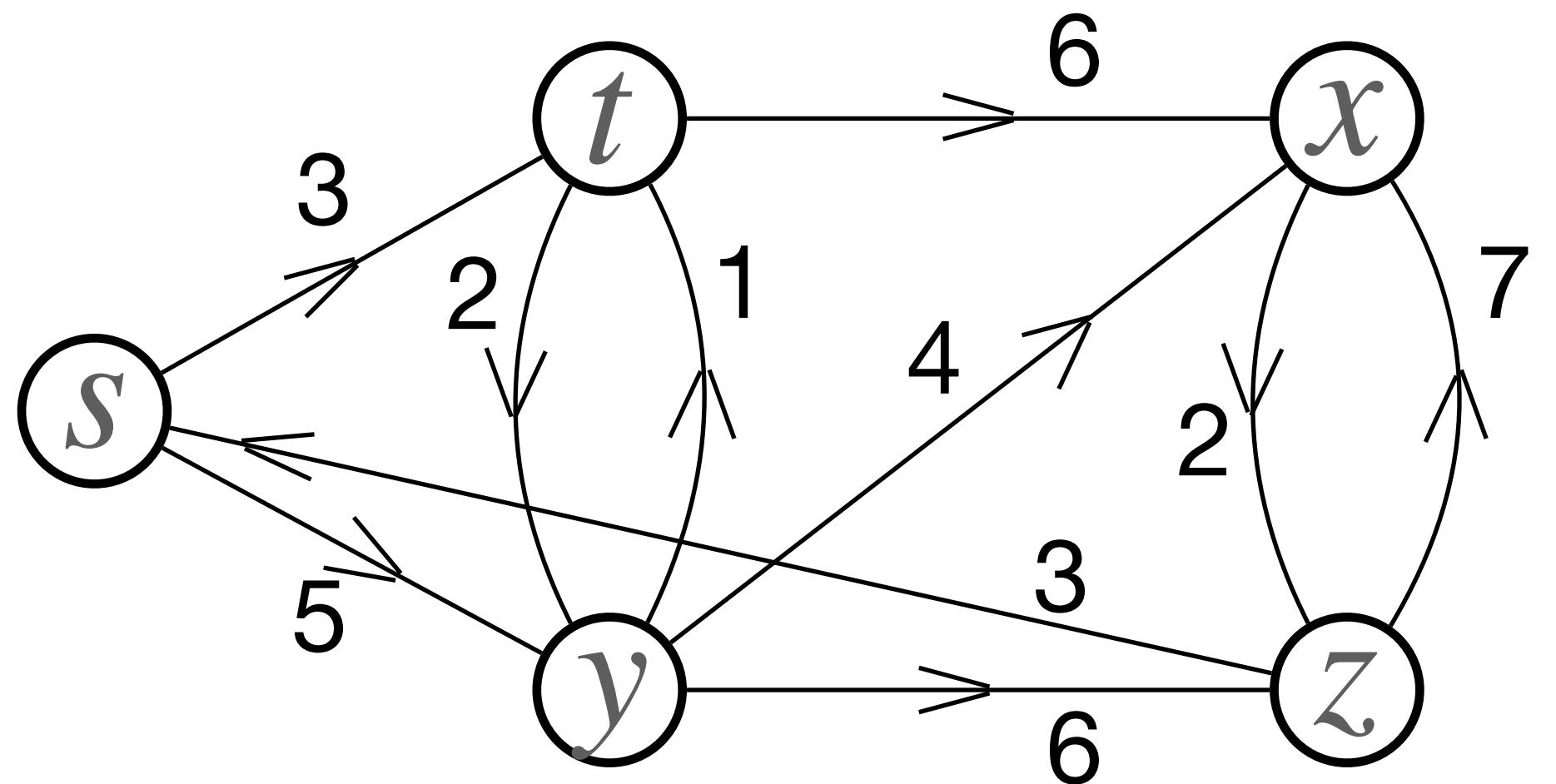
Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

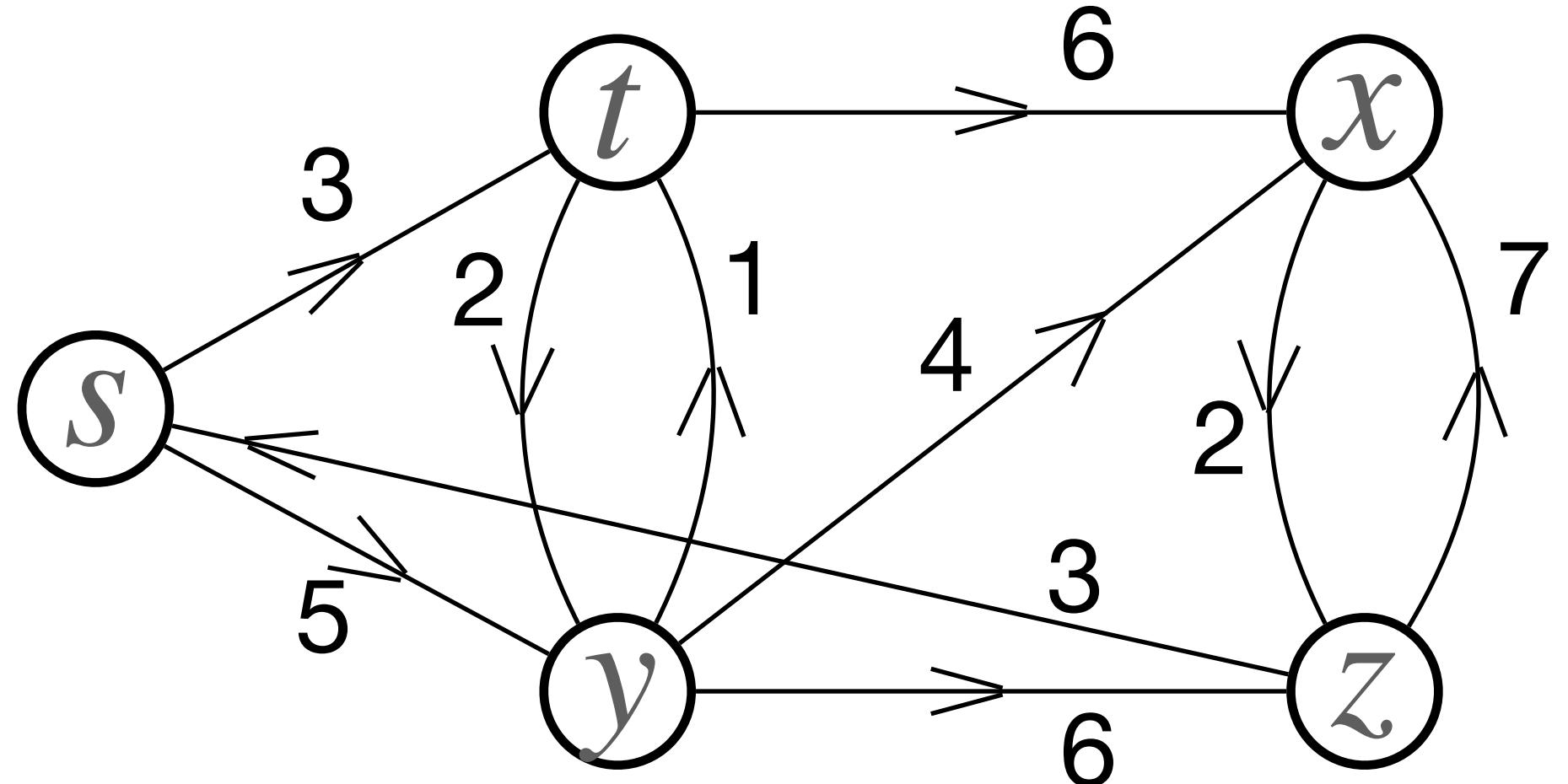
Output: Find a shortest path from s to every other node.

Byproduct: Those shortest paths will form a **shortest-path tree** rooted at s .

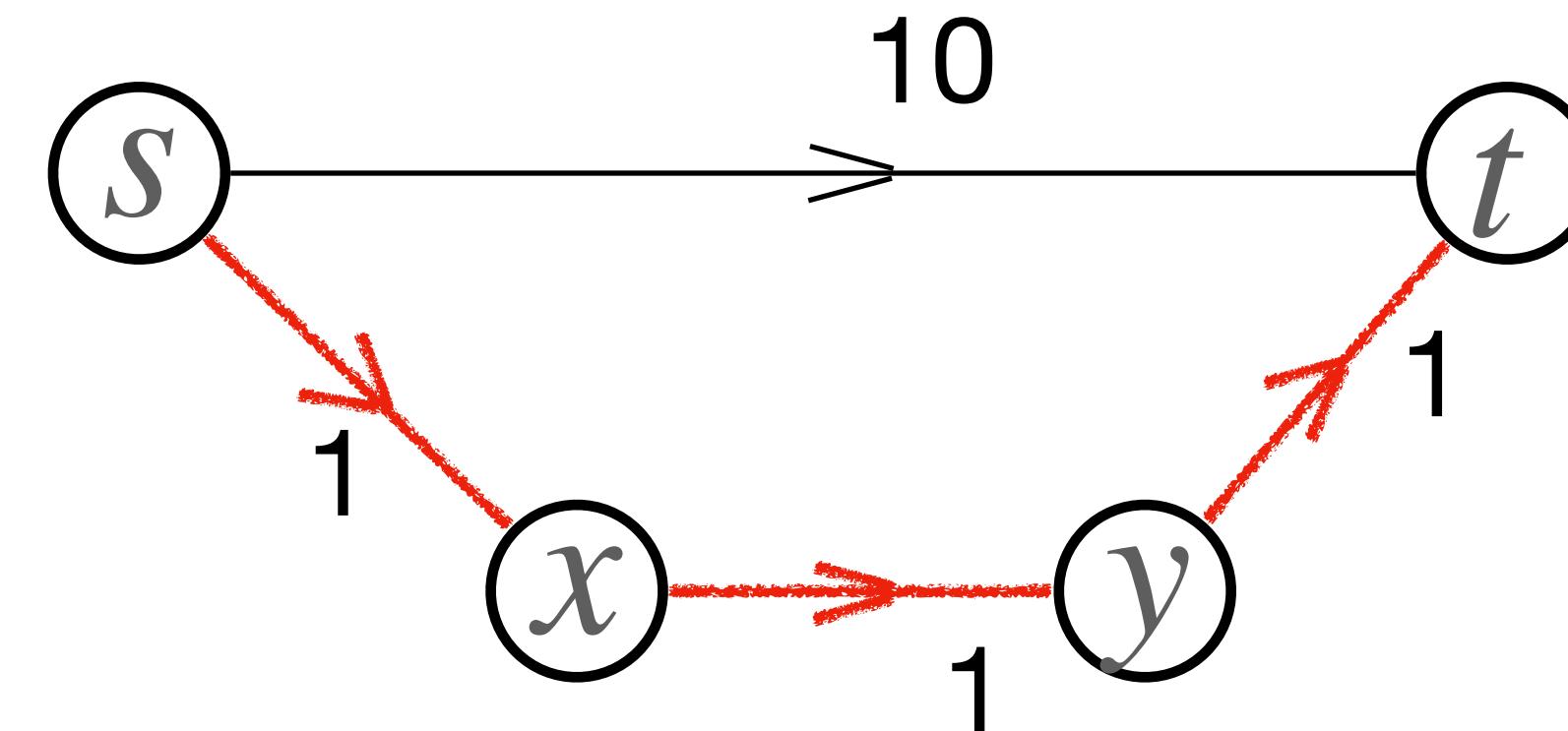
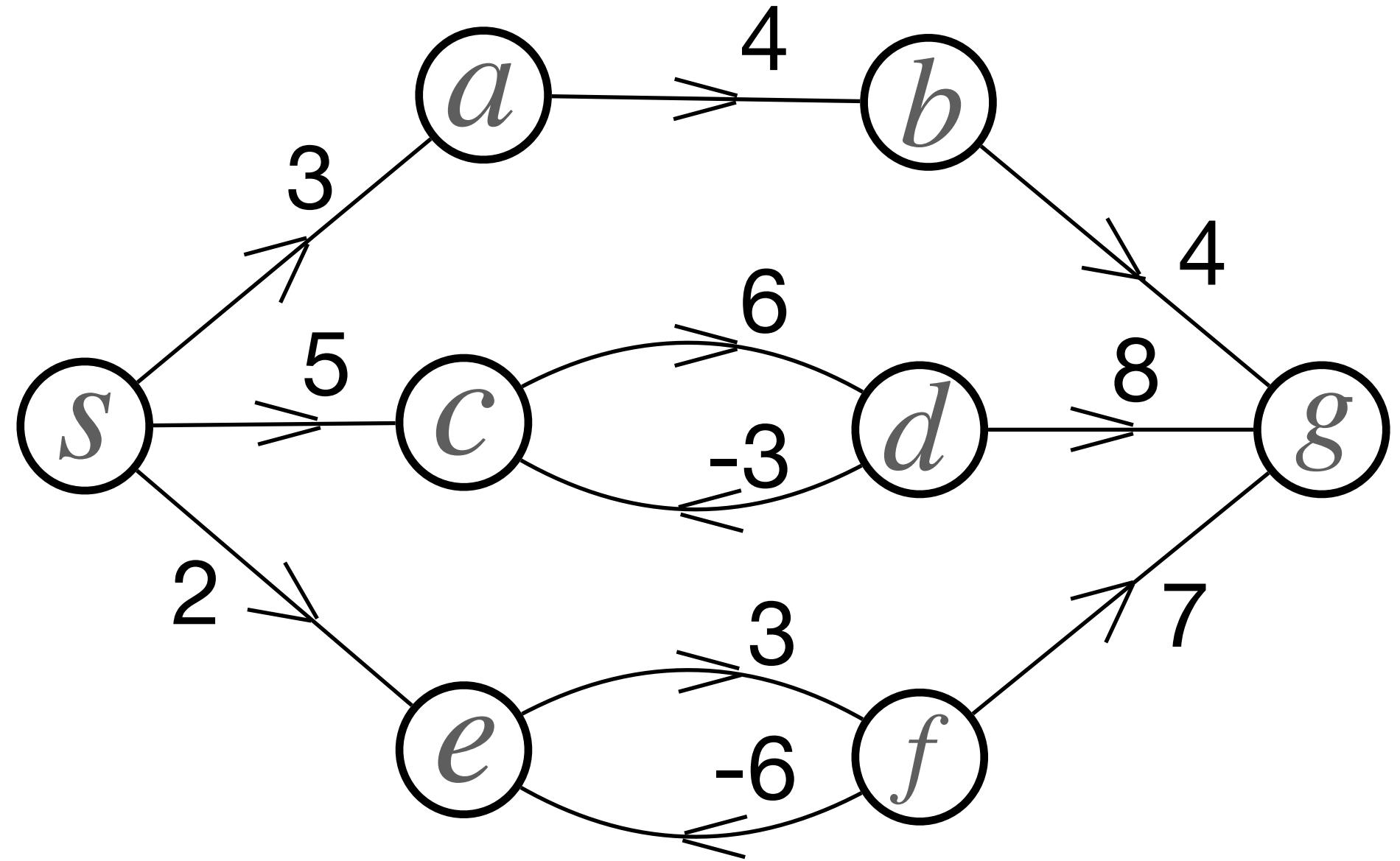
Single-Source Shortest Paths



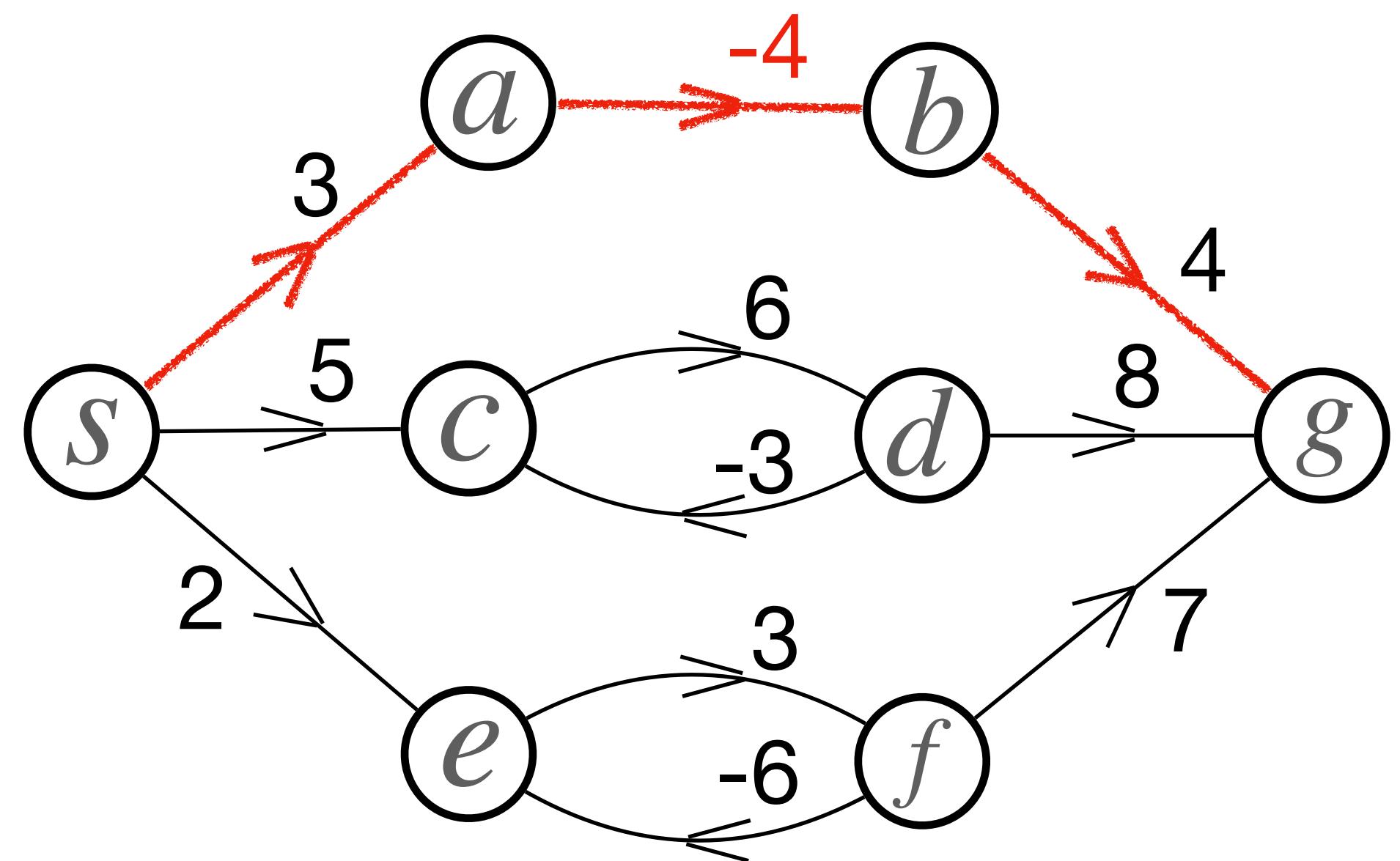
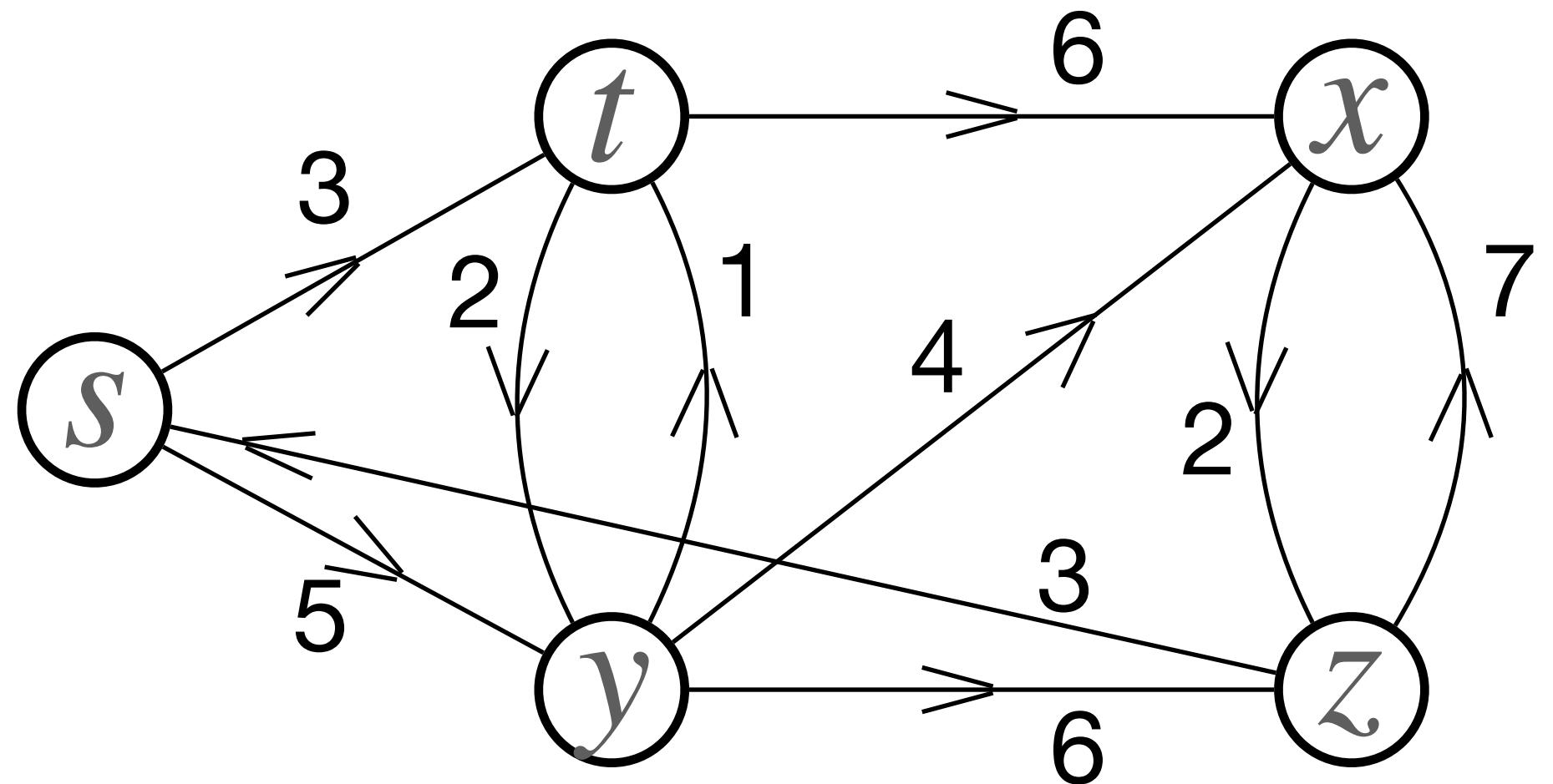
Single-Source Shortest Paths



A shortest path may have many edges.

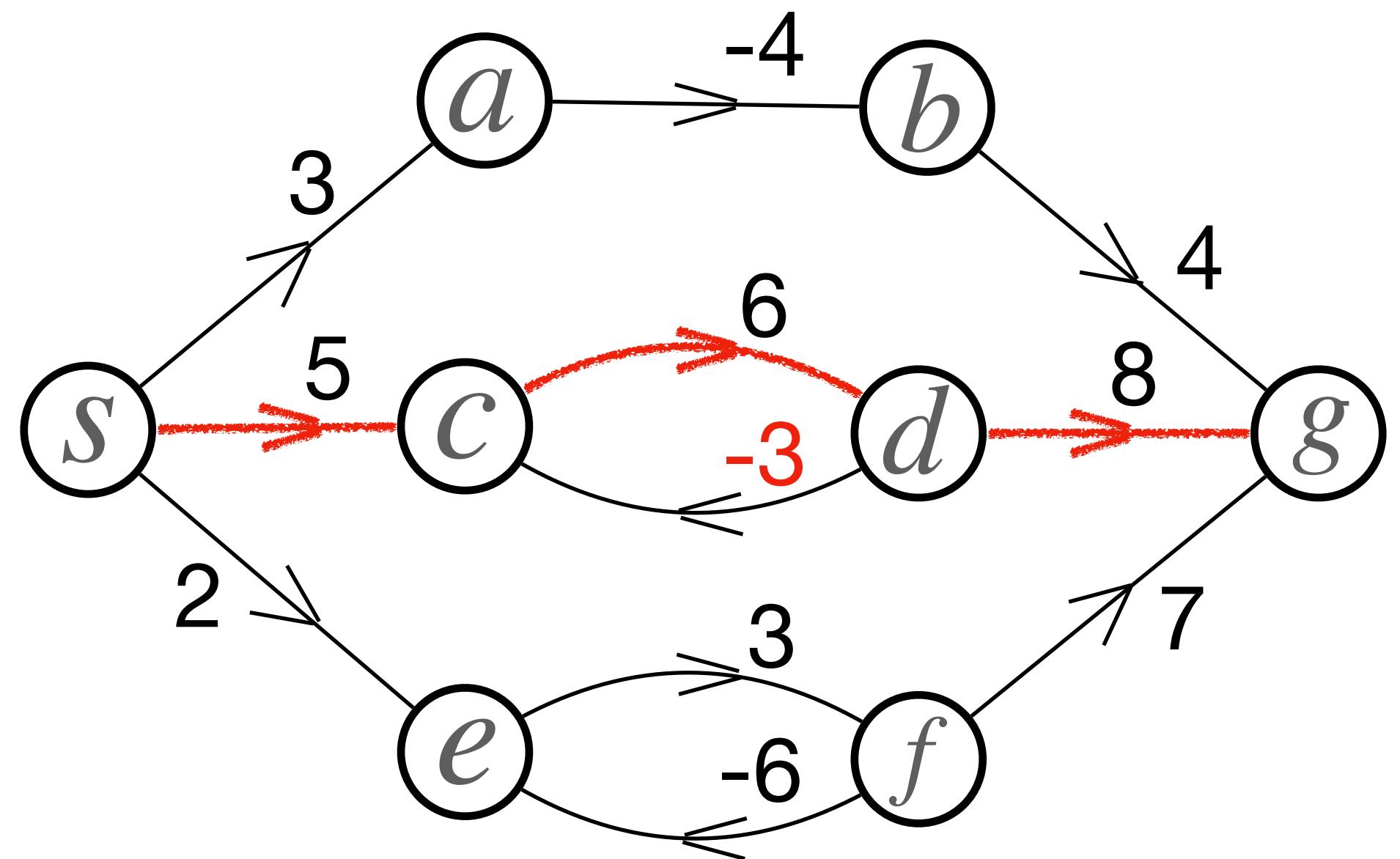
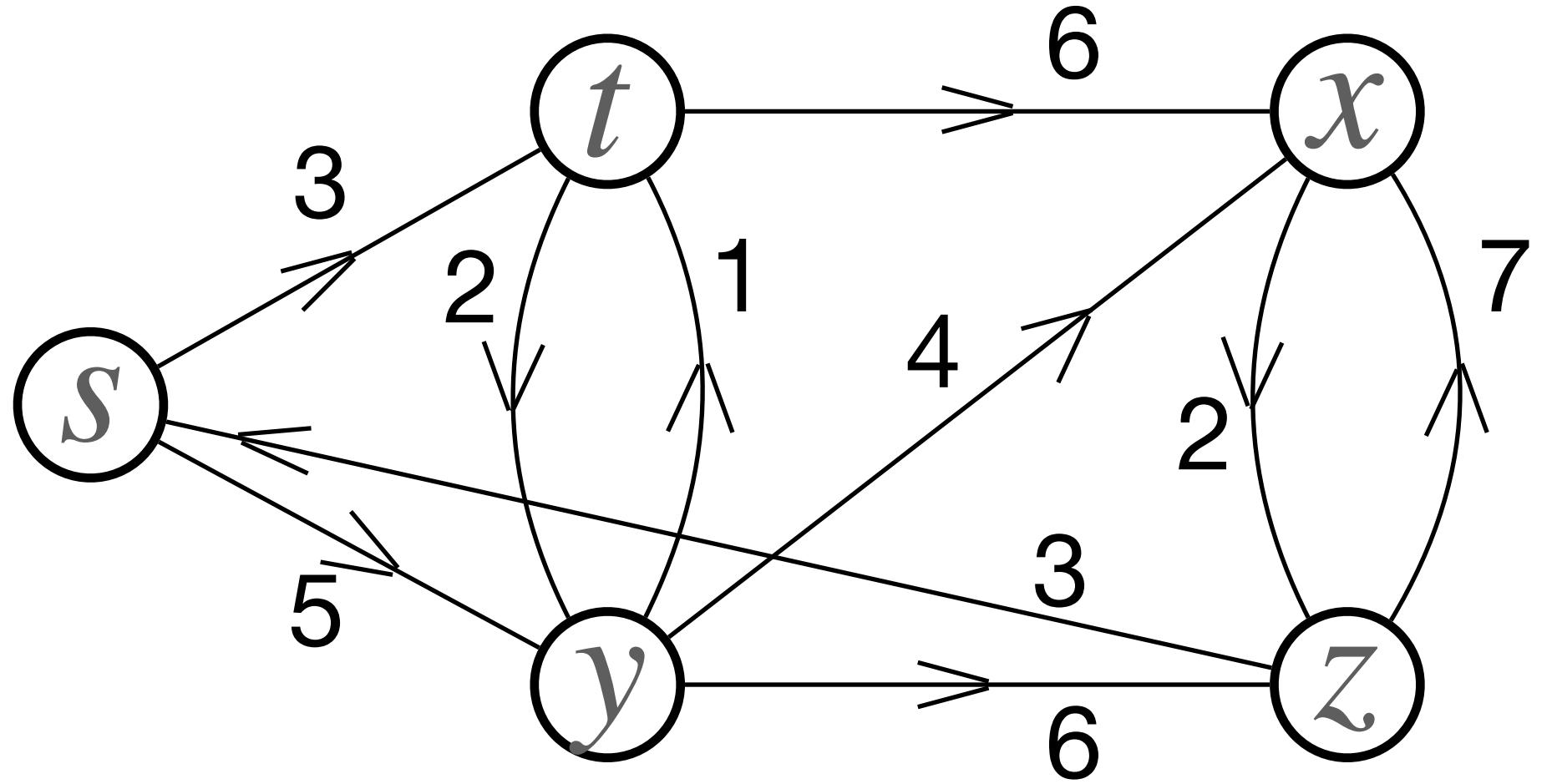


Single-Source Shortest Paths



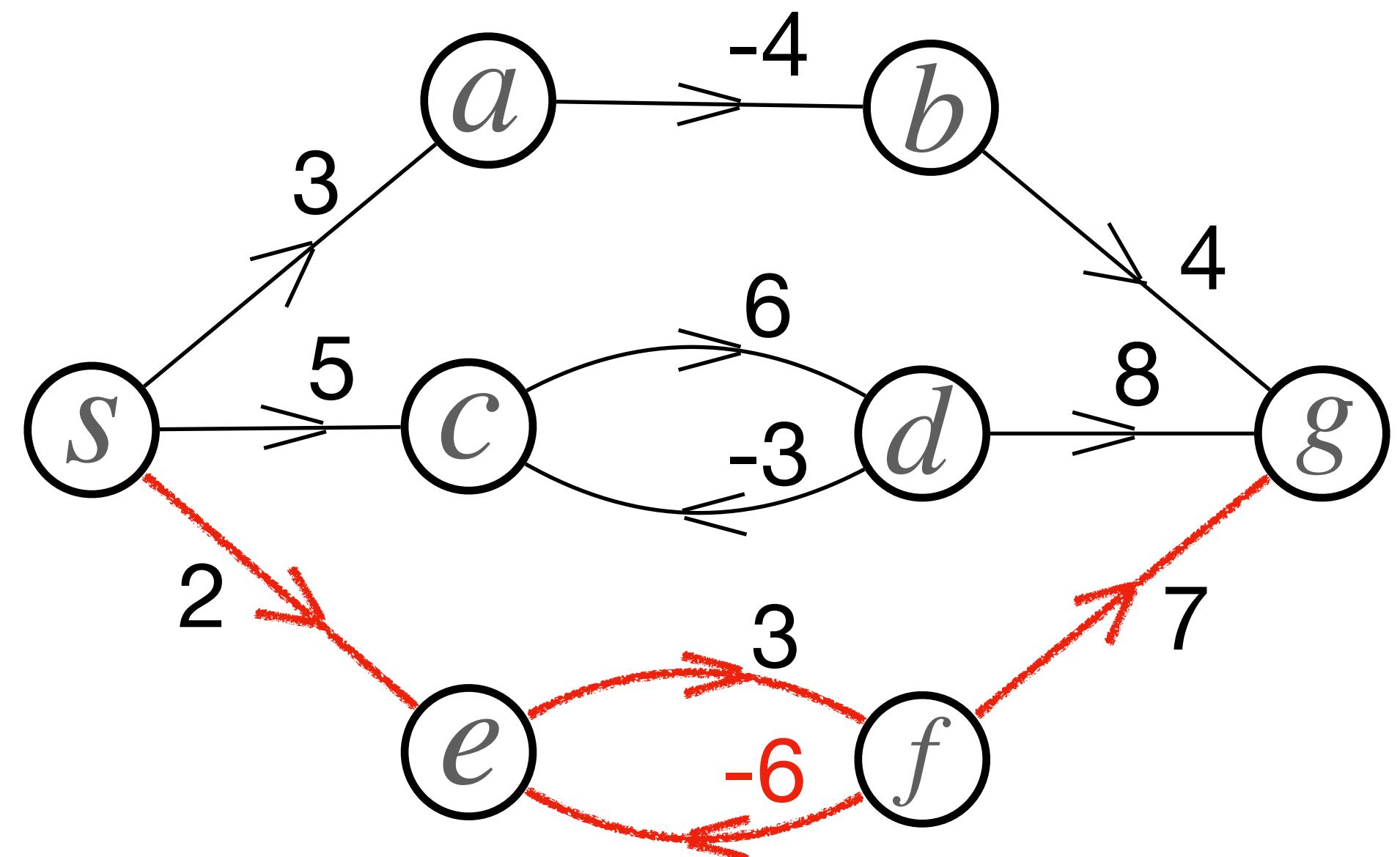
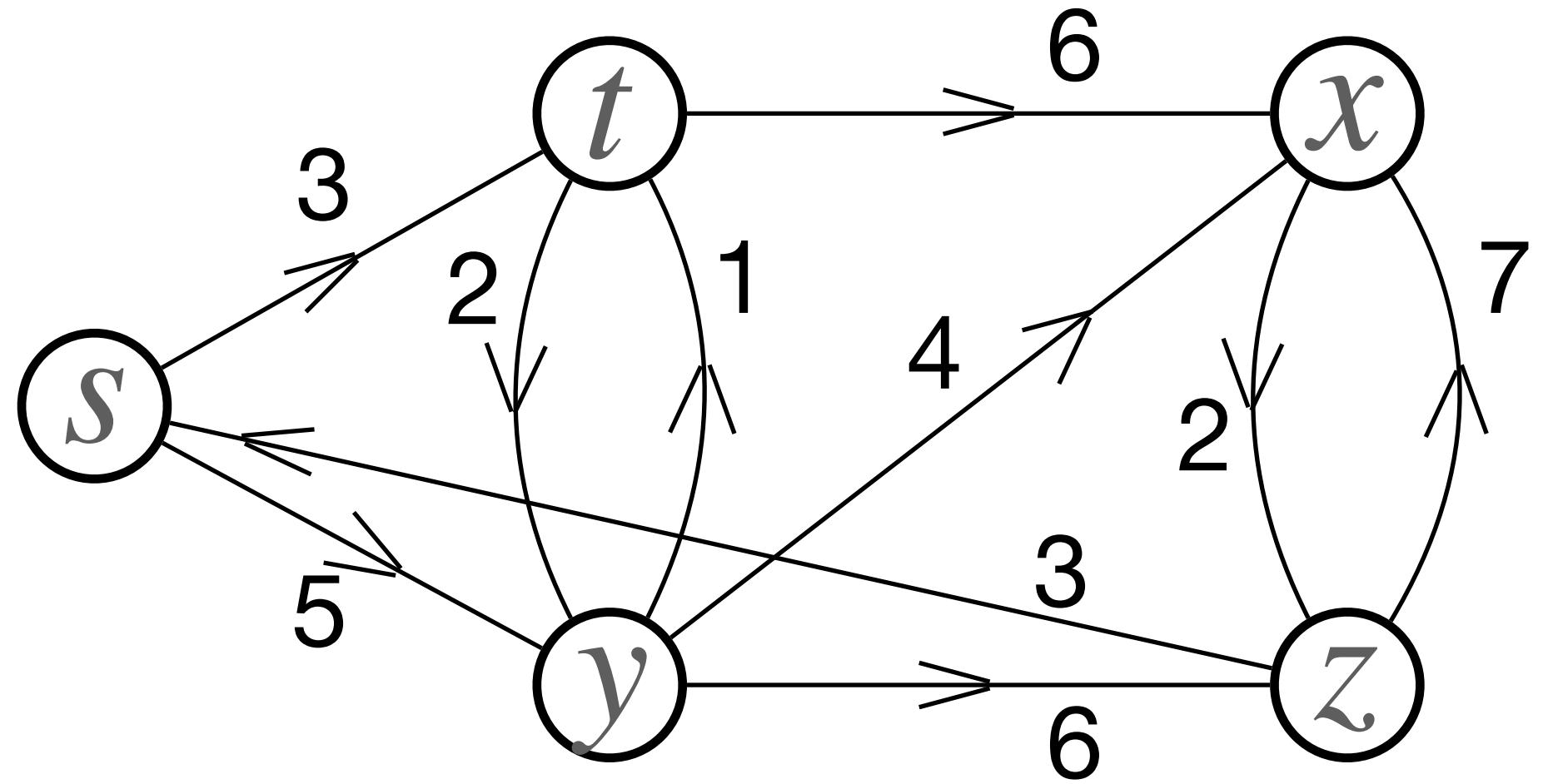
Edge weights may be negative.

Single-Source Shortest Paths



Edge weights may be negative.

Single-Source Shortest Paths



There can be negative cycles.

Single-Source Shortest Paths

Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node **if G has no negative cycle that s can reach.**
(Otherwise, we return “There is a negative cycle that s can reach”.)

Quiz questions:

1. How do negative cycles complicate the shortest path problem?
2. What are the applications of finding shortest paths in a graph?

Roadmap of this lecture:

1. Single-Source Shortest Paths

1.1 Define the "Single-Source Shortest Paths Problem".

1.2 Bellman-Ford Algorithm for finding single-source shortest paths.

1.3 Correctness of the single-source shortest paths algorithm.

1.4 Finding single-source shortest paths in a directed acyclic graph.

Single-Source Shortest Paths

Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node **if G has no negative cycle that s can reach.**
(Otherwise, we return “There is a negative cycle that s can reach”.)

Theorem: A subpath of a shortest path is also a shortest path.

Single-Source Shortest Paths

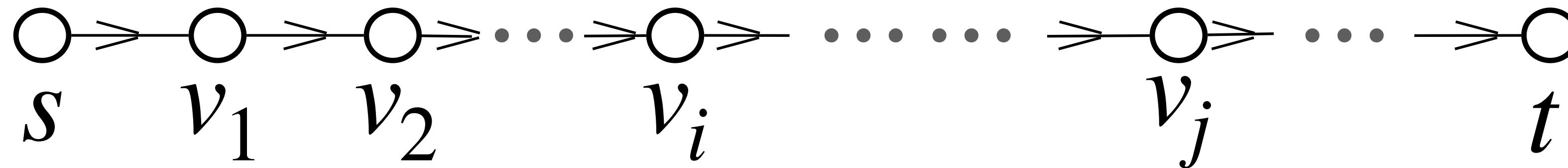
Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node **if G has no negative cycle that s can reach.**
(Otherwise, we return “There is a negative cycle that s can reach”.)

Theorem: A subpath of a shortest path is also a shortest path.

Proof: Consider a shortest path from s to t .



Single-Source Shortest Paths

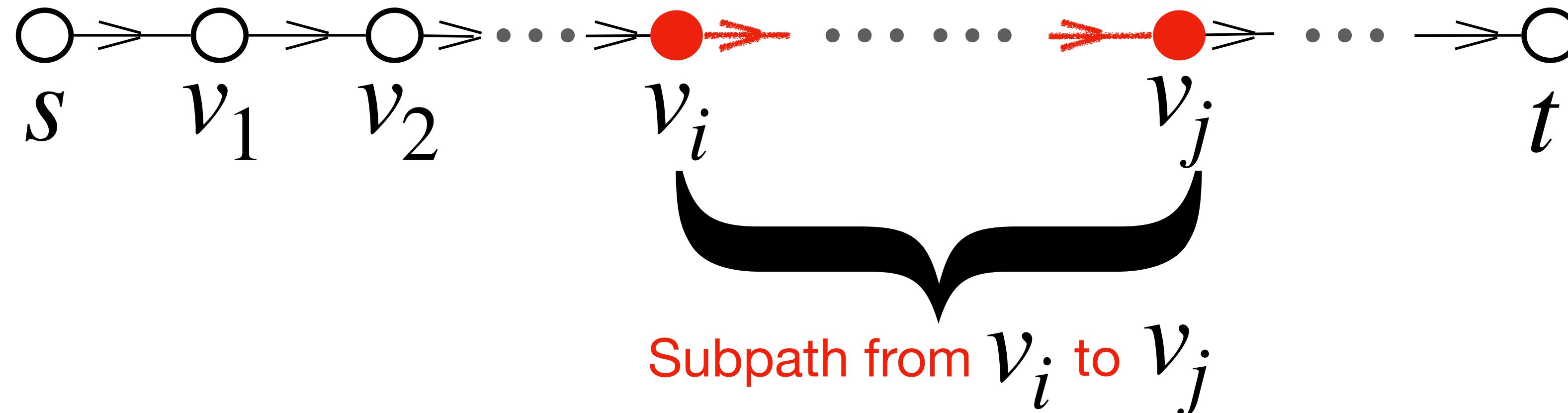
Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node **if G has no negative cycle that s can reach.**
(Otherwise, we return “There is a negative cycle that s can reach”.)

Theorem: A subpath of a shortest path is also a shortest path.

Proof: Consider a shortest path from s to t .



Single-Source Shortest Paths

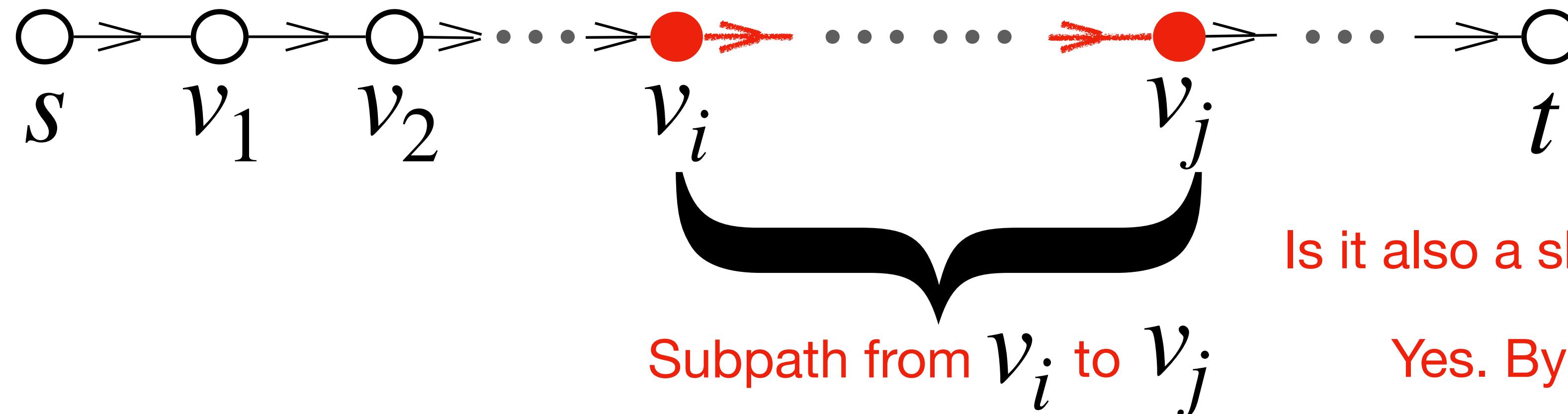
Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node **if G has no negative cycle that s can reach.**
(Otherwise, we return “There is a negative cycle that s can reach”.)

Theorem: A subpath of a shortest path is also a shortest path.

Proof: Consider a shortest path from s to t .



Yes. By contradiction.

Single-Source Shortest Paths

Input: A directed graph $G=(V,E)$ where every edge $e \in E$ has a weight $w(e)$.

Let $s \in V$ be a node called the “source”.

Output: Find a shortest path from s to every other node **if G has no negative cycle that s can reach.**
(Otherwise, we return “There is a negative cycle that s can reach”.)

For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Single-Source Shortest Paths

For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

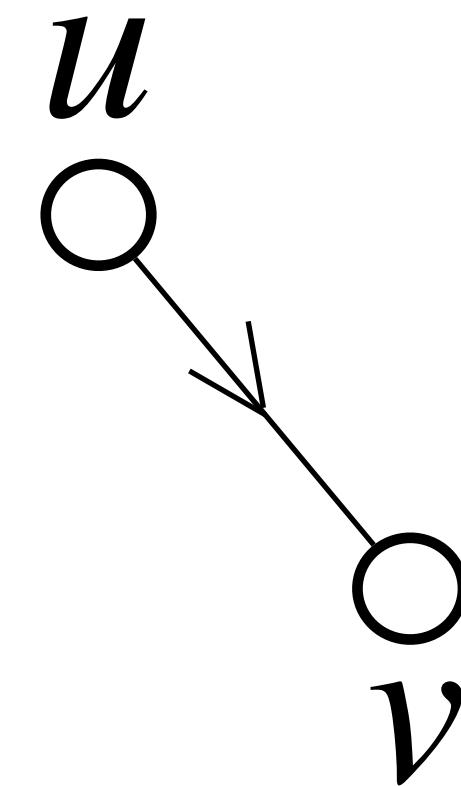
Single-Source Shortest Paths

For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

- 1) Find shorter and shorter paths from s to each node v .



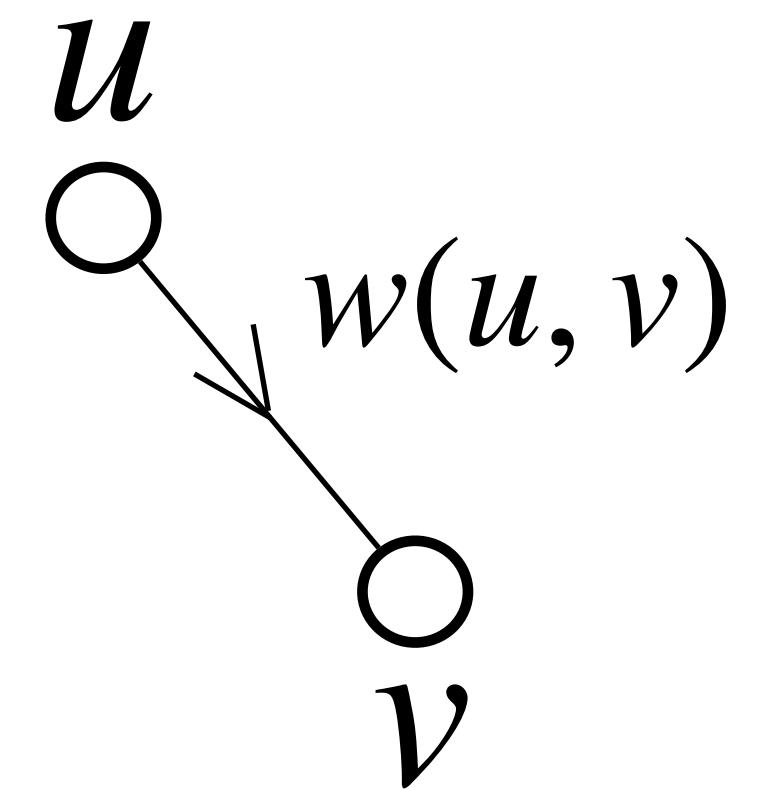
Single-Source Shortest Paths

For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

- 1) Find shorter and shorter paths from s to each node v .



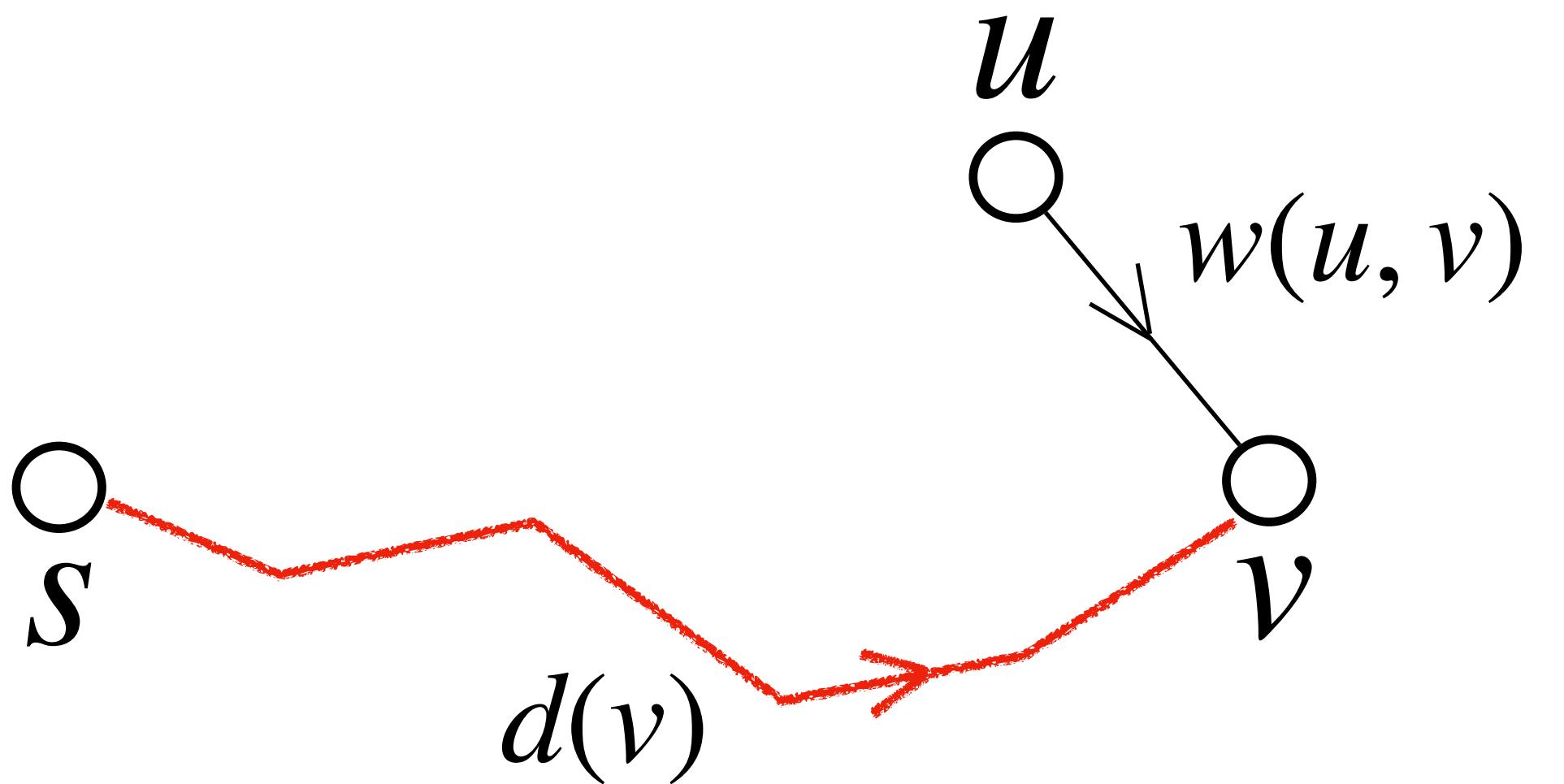
Single-Source Shortest Paths

For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

- 1) Find shorter and shorter paths from s to each node v .



Single-Source Shortest Paths

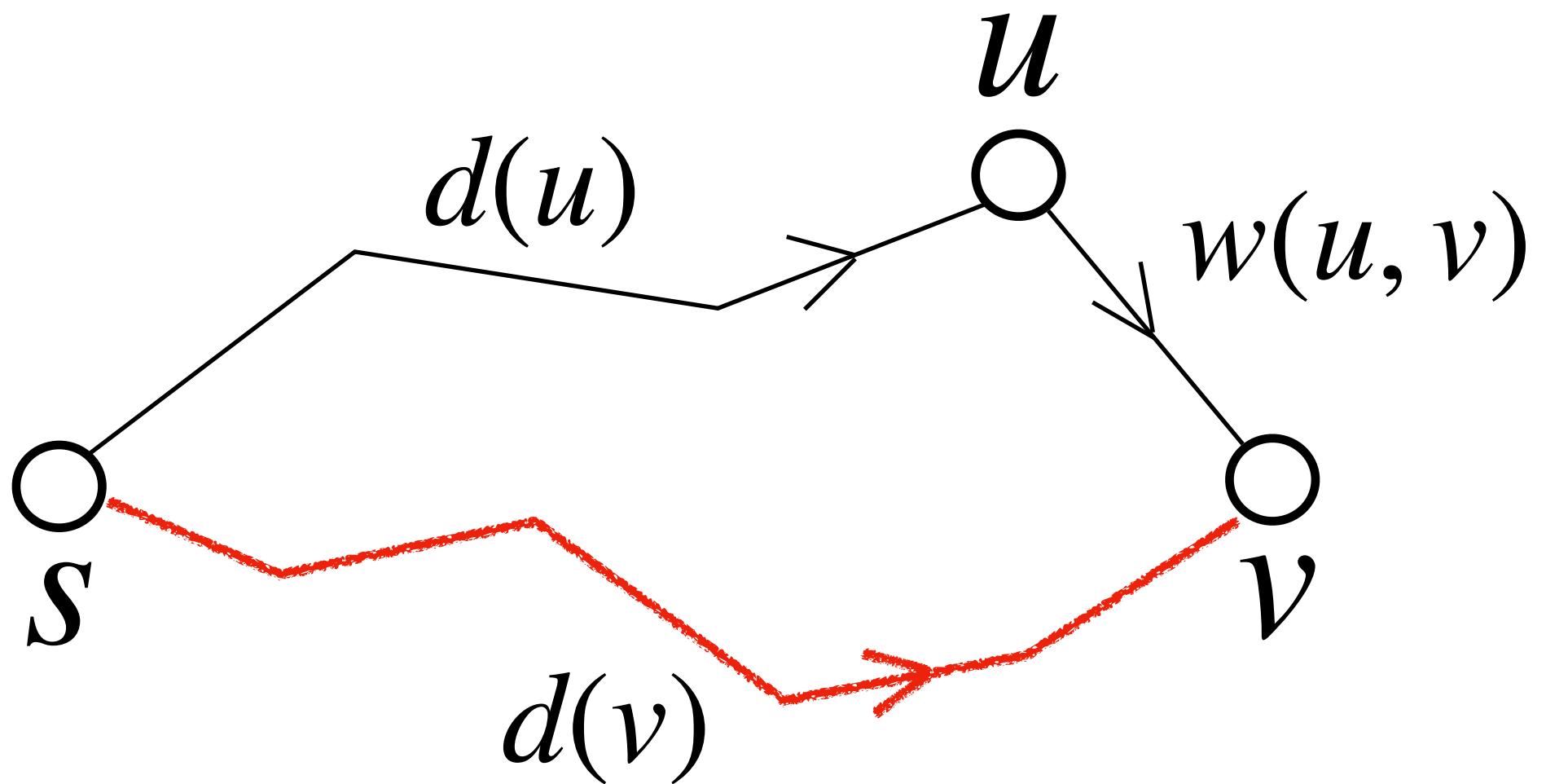
For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

- 1) Find shorter and shorter paths from s to each node v .

if $d(u) + w(u, v) < d(v)$



Single-Source Shortest Paths

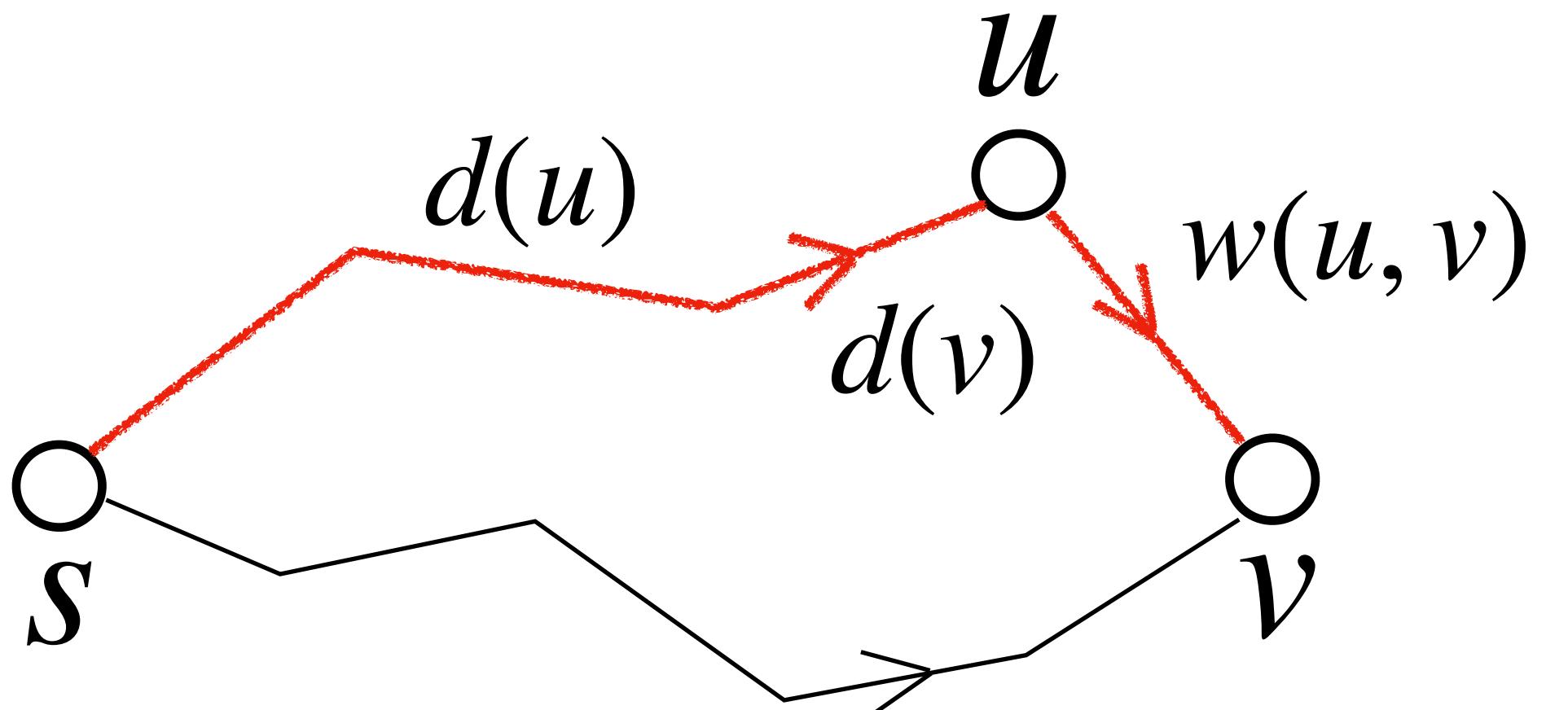
For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

- 1) Find shorter and shorter paths from s to each node v .

if $d(u) + w(u, v) < d(v)$



Single-Source Shortest Paths

For every node $v \in V$,

- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

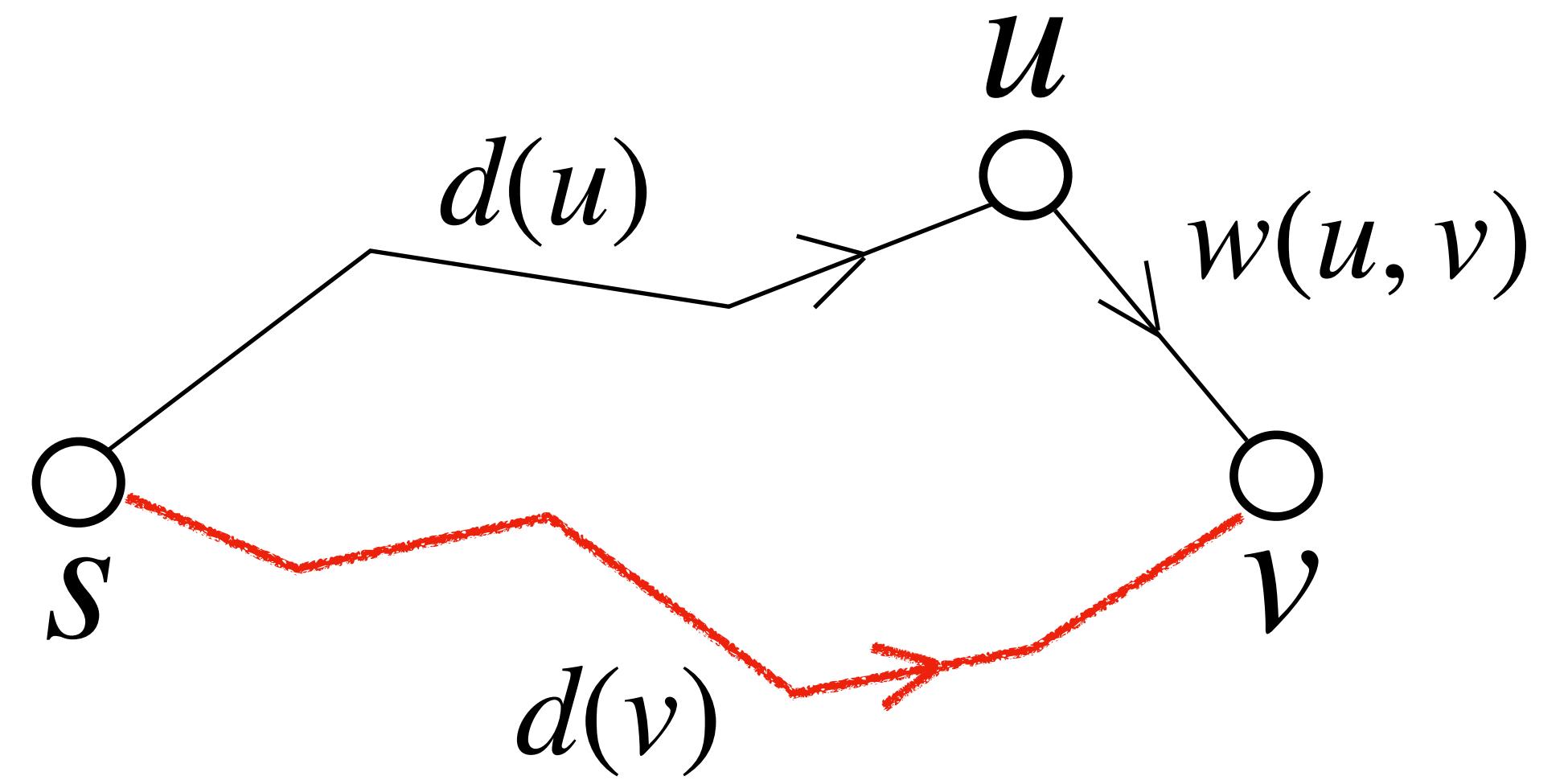
1) Find shorter and shorter paths from s to each node v .

2) Each time, we look at an edge (u, v) , and do:

if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$



Single-Source Shortest Paths

For every node $v \in V$,

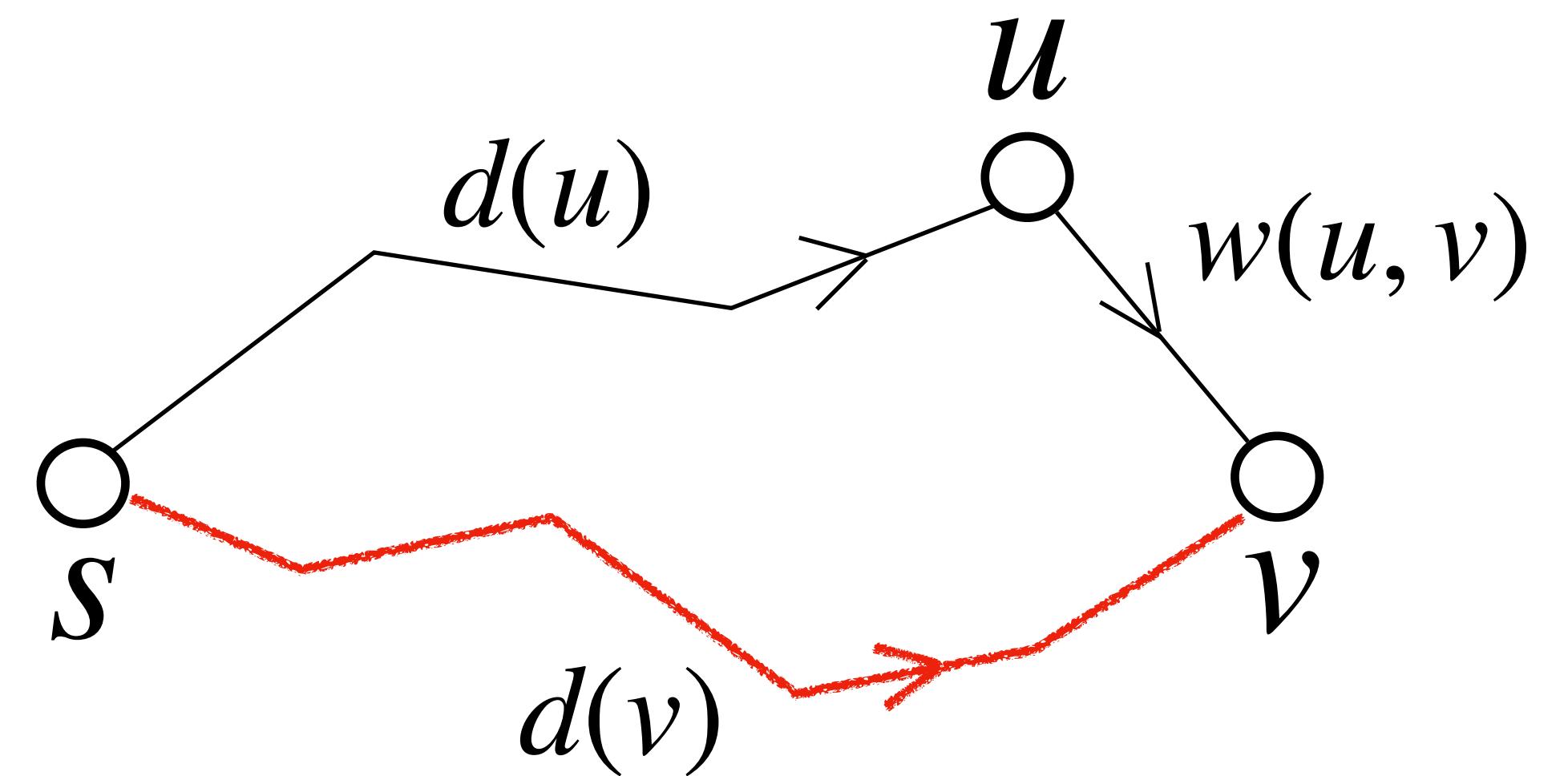
- 1) Let $d(v)$ denote the length of the shortest path **we have found so far** from s to v .
- 2) Let $\pi(v)$ be the node in front of v in the shortest path we have found so far from s to v .

Idea of our algorithm:

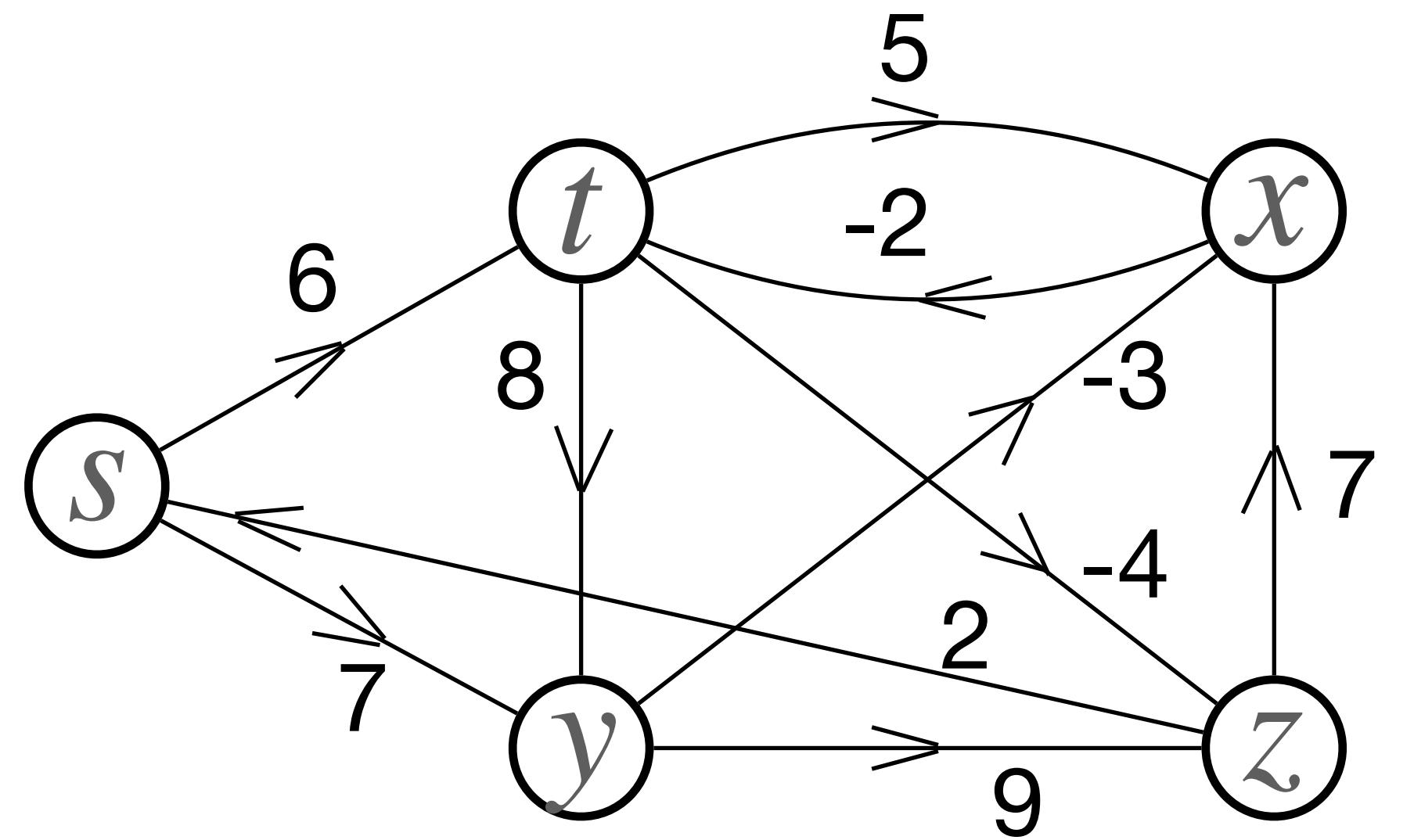
- 1) Find shorter and shorter paths from s to each node v .
- 2) Each time, we look at an edge (u, v) , and do:

if $d(u) + w(u, v) < d(v)$:
 $d(v) = d(u) + w(u, v)$
 $\pi(v) = u$

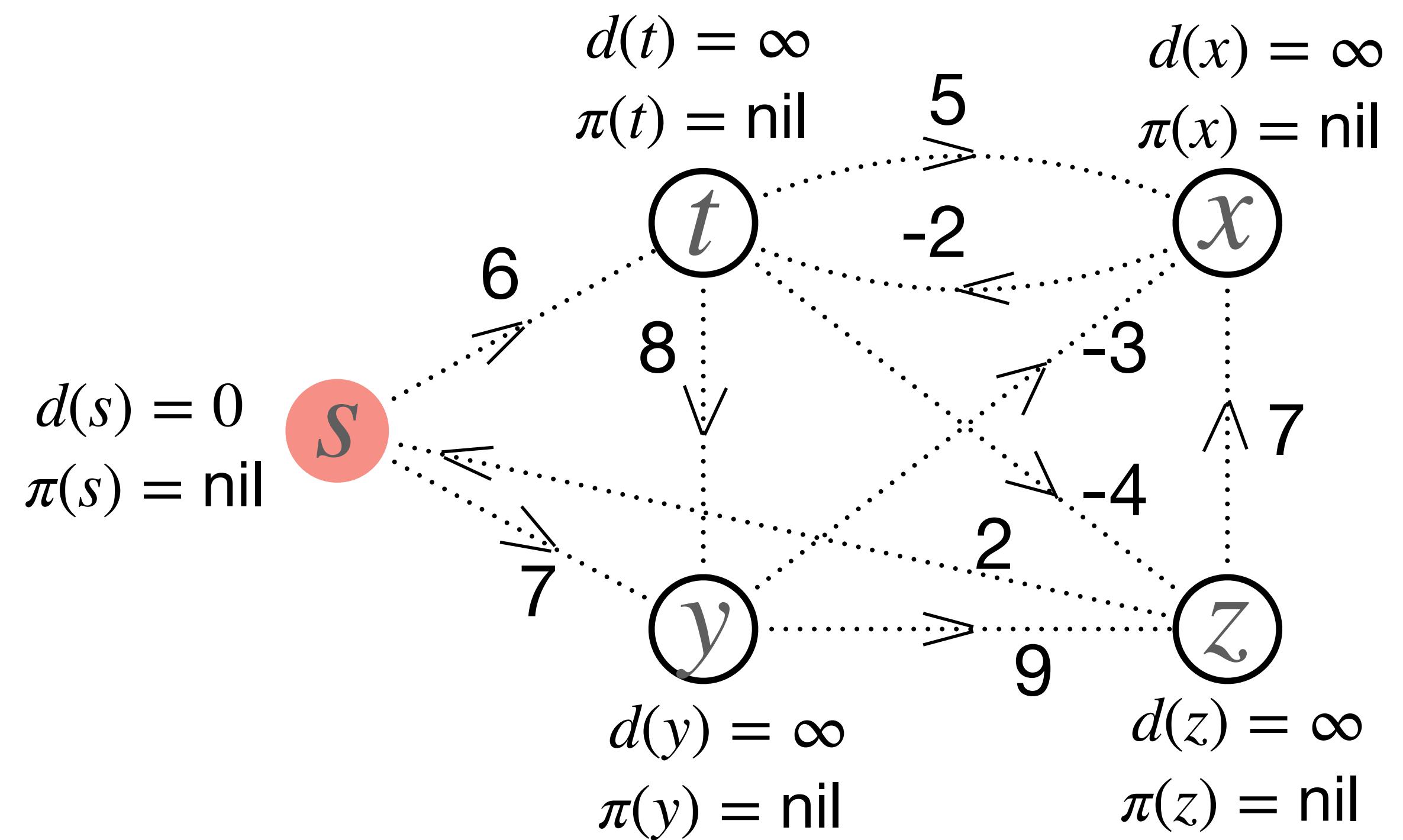
Relaxation



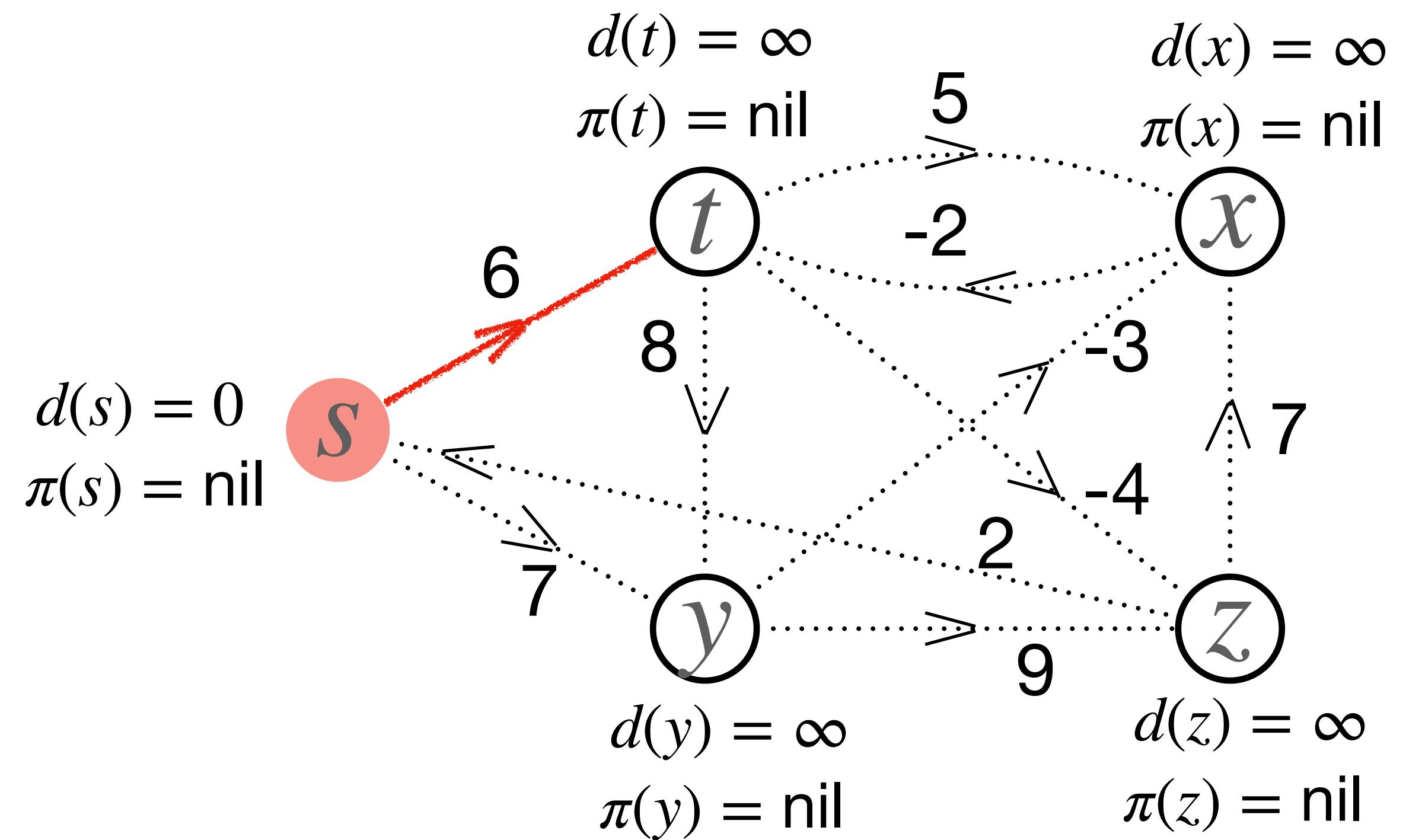
Single-Source Shortest Paths



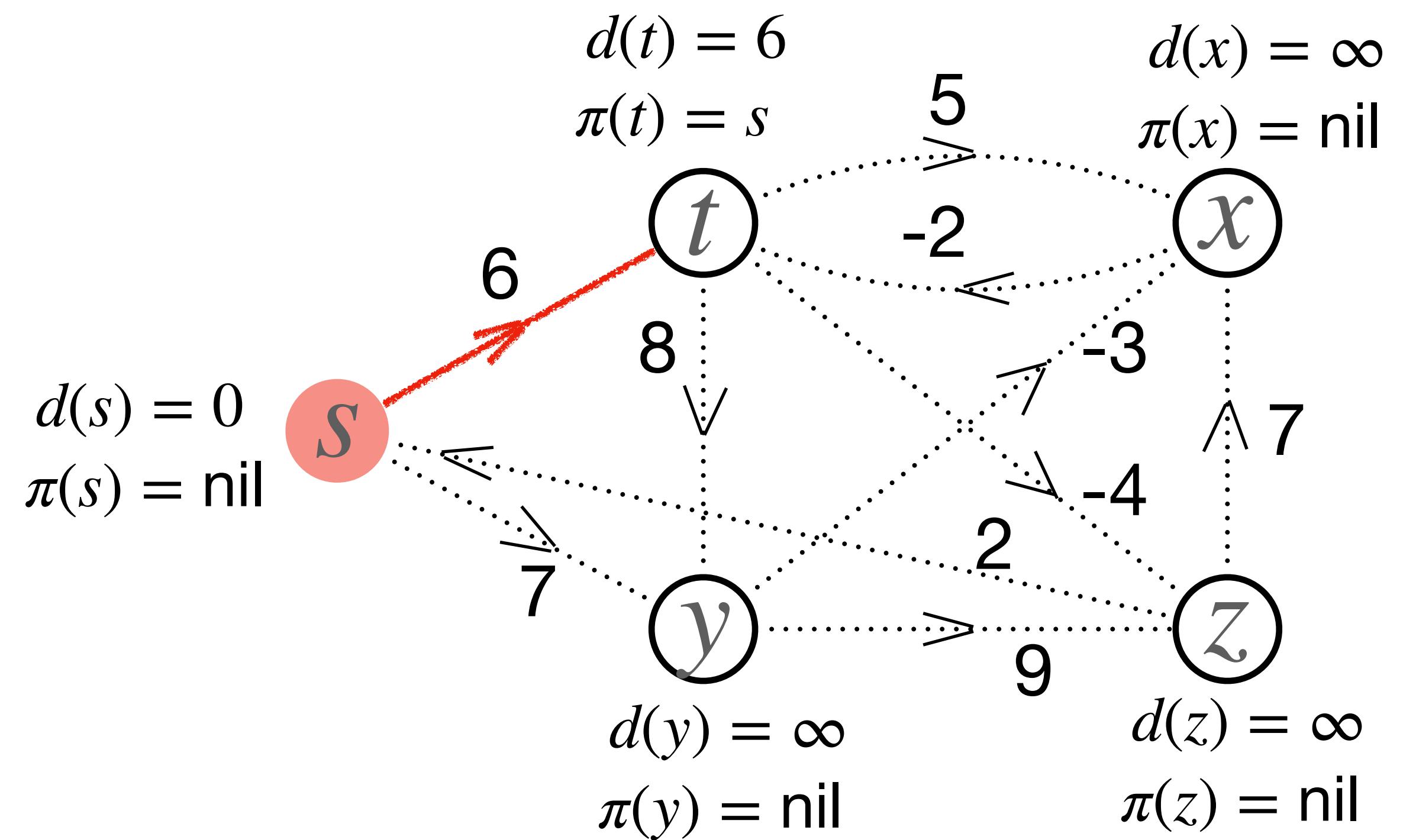
Single-Source Shortest Paths



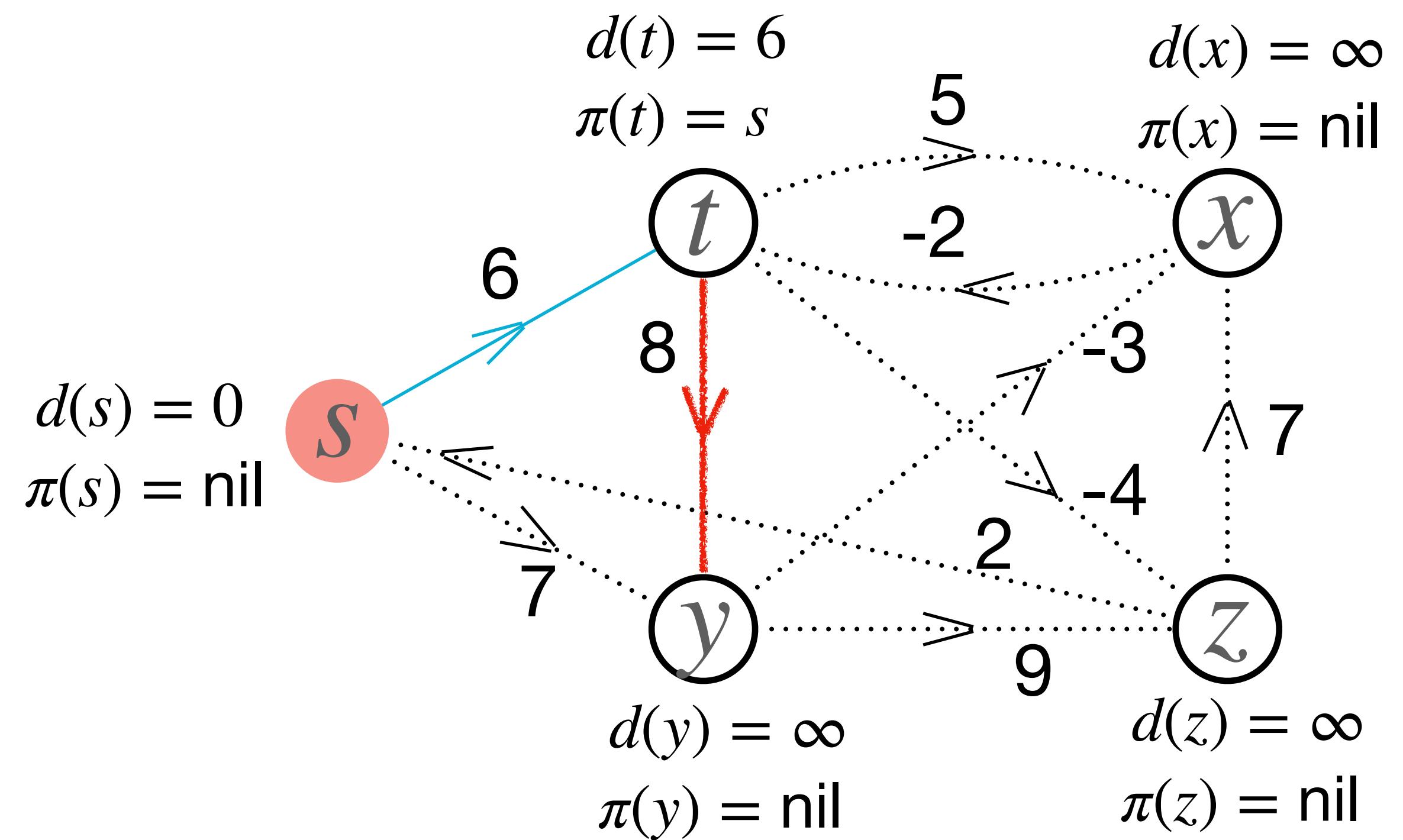
Single-Source Shortest Paths



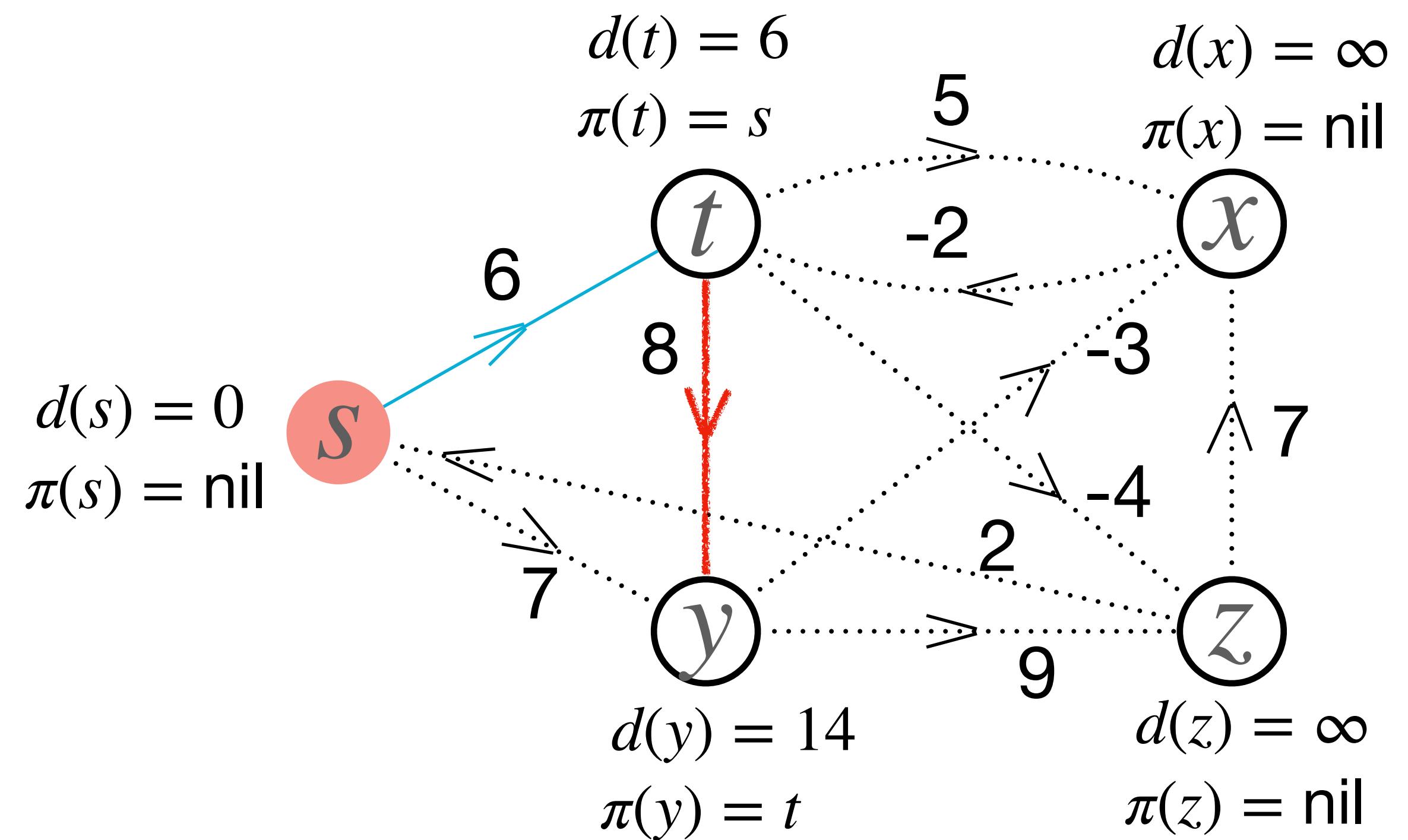
Single-Source Shortest Paths



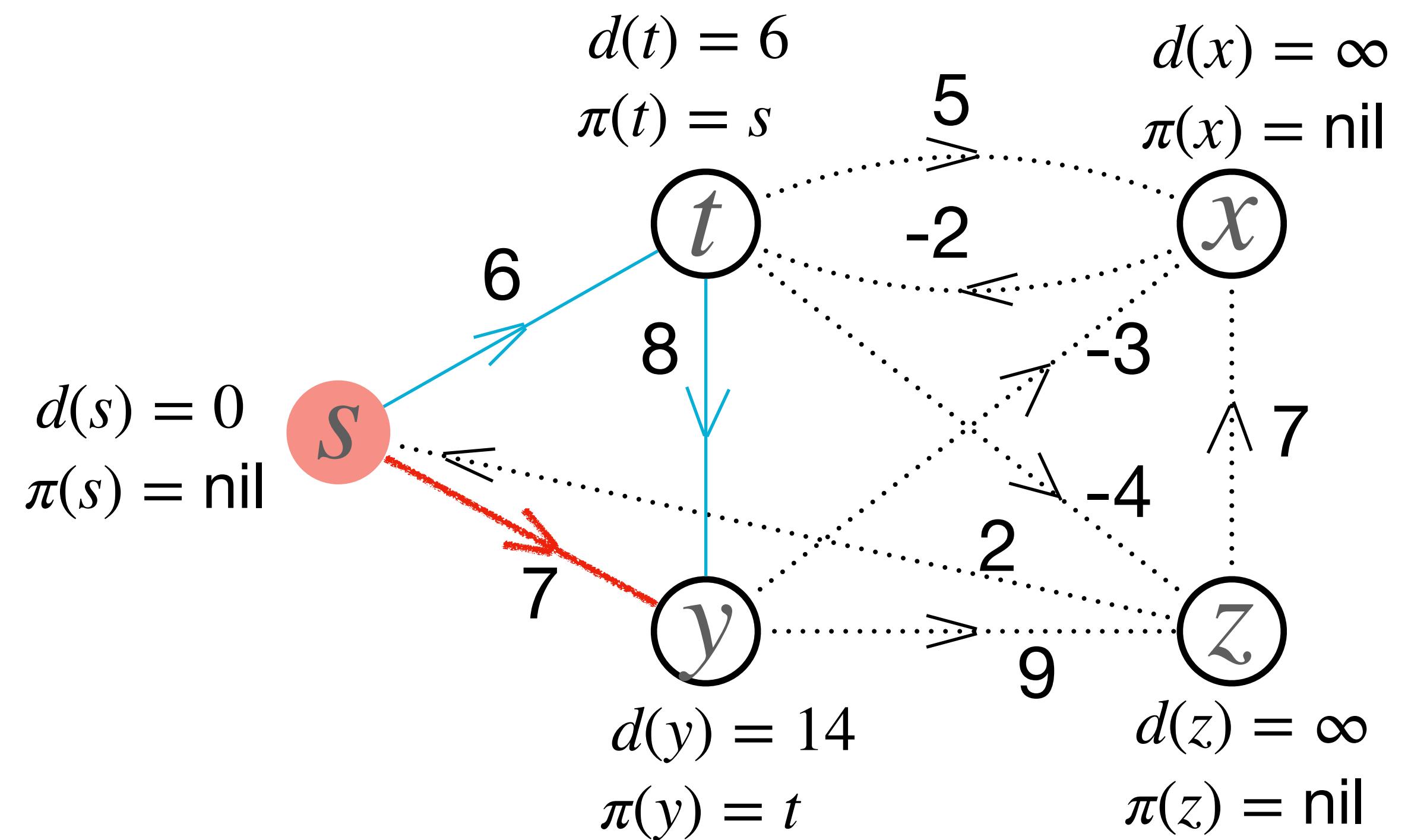
Single-Source Shortest Paths



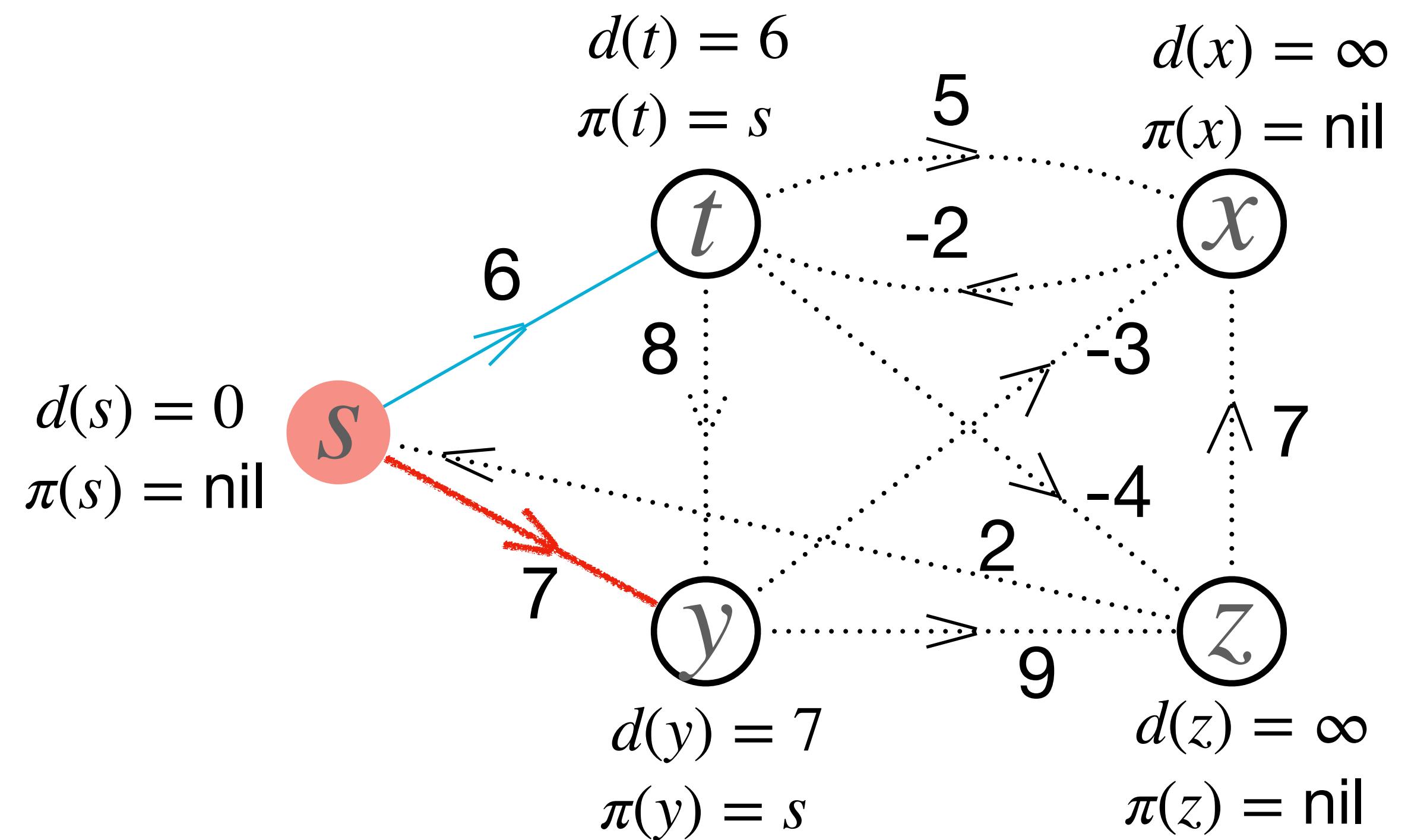
Single-Source Shortest Paths



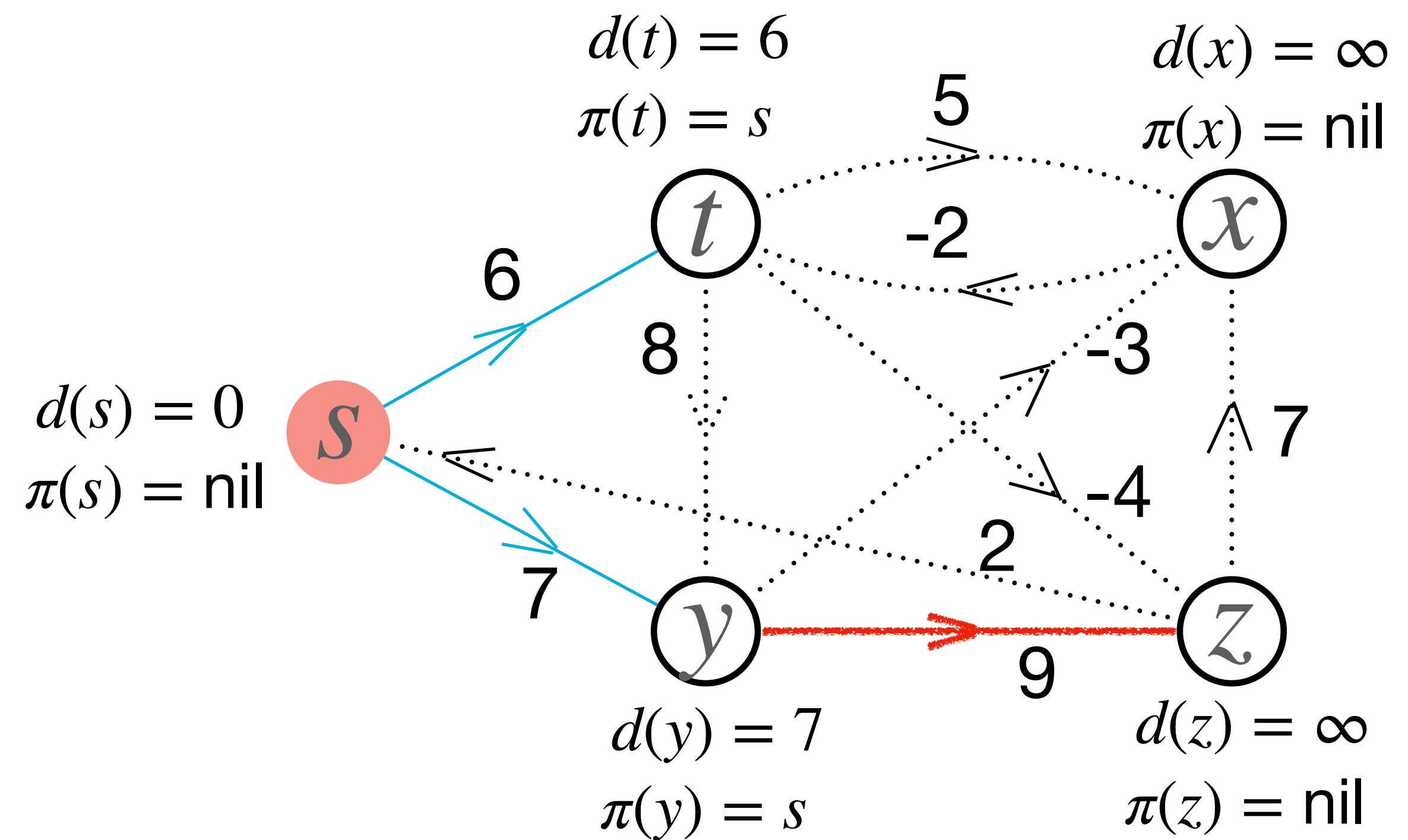
Single-Source Shortest Paths



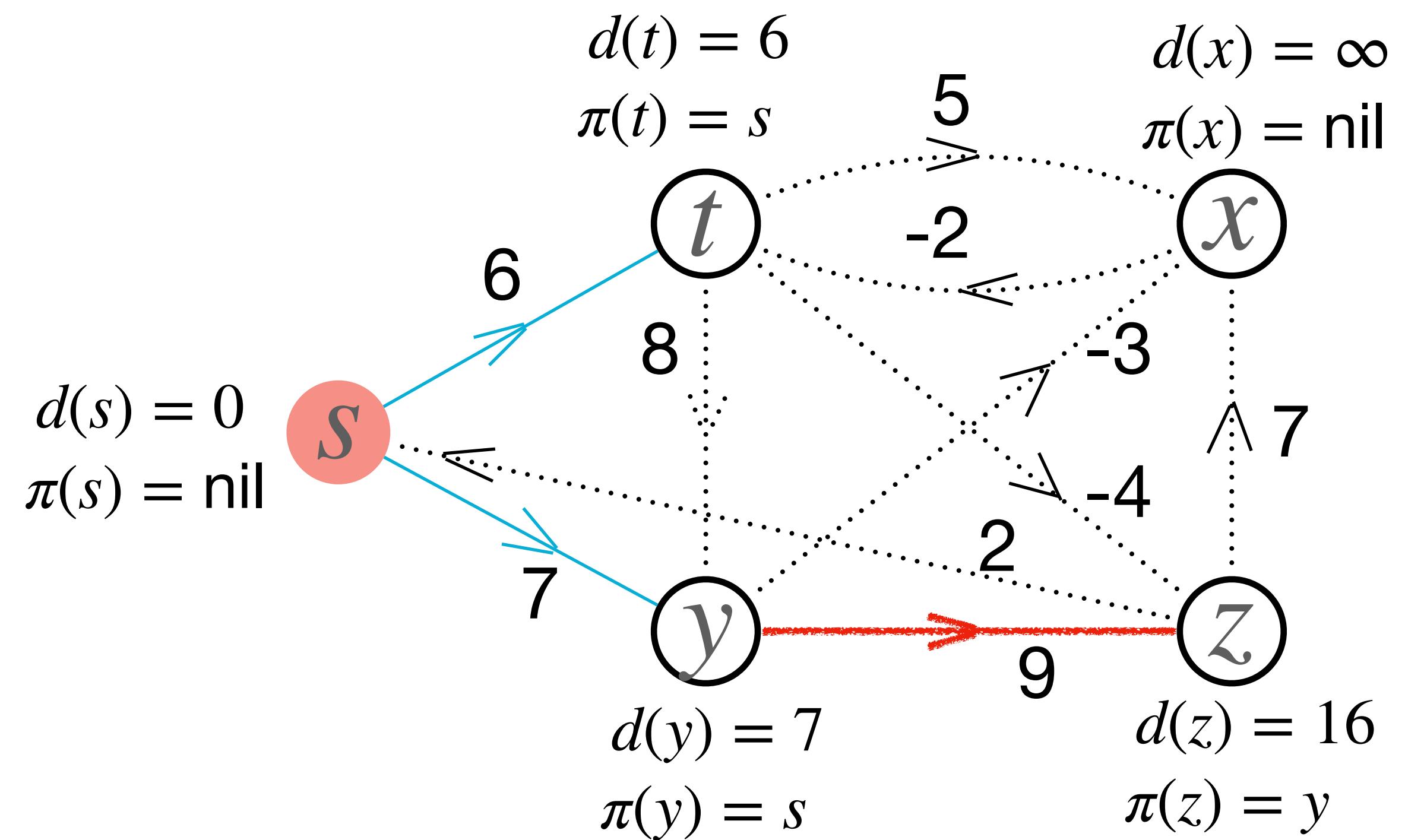
Single-Source Shortest Paths



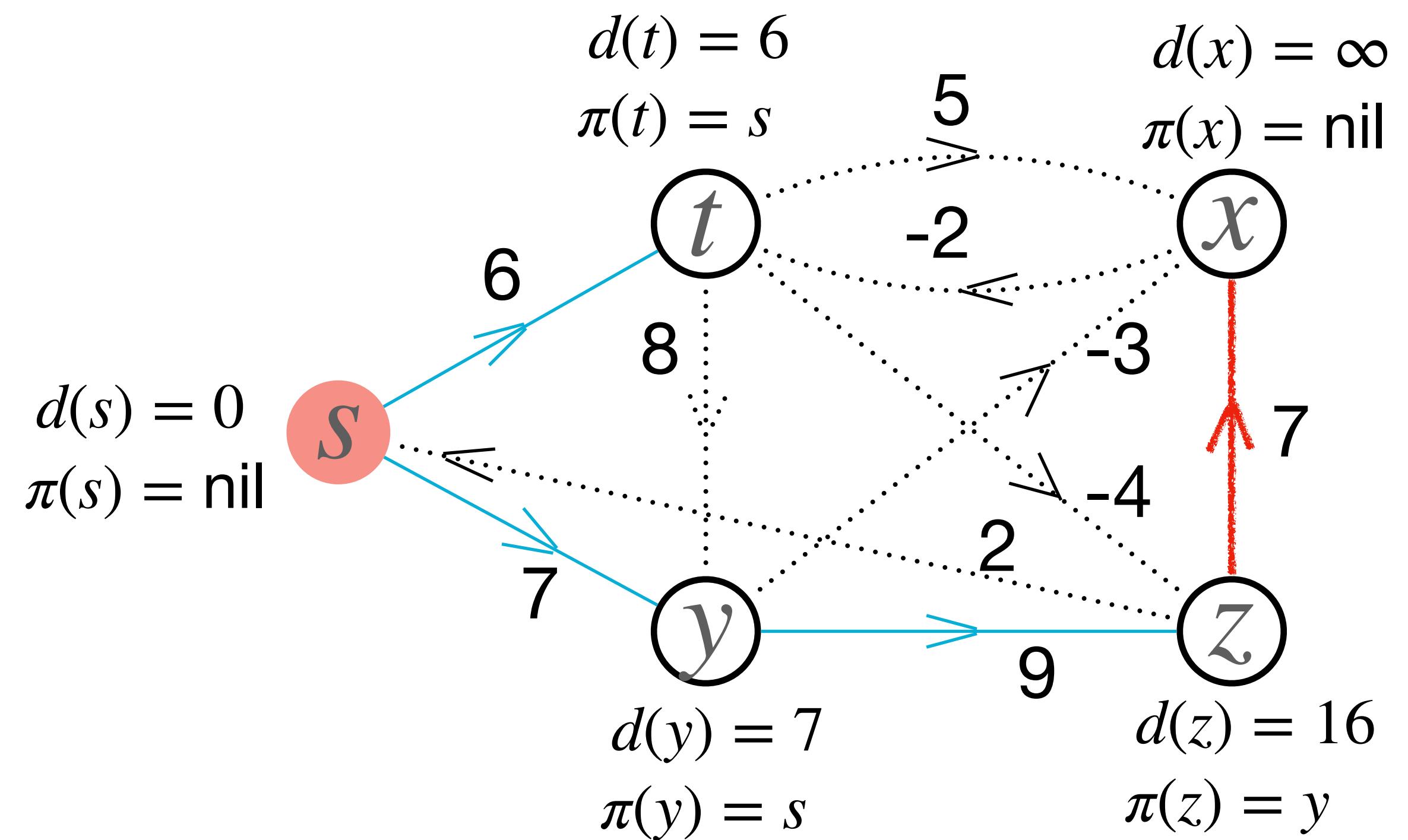
Single-Source Shortest Paths



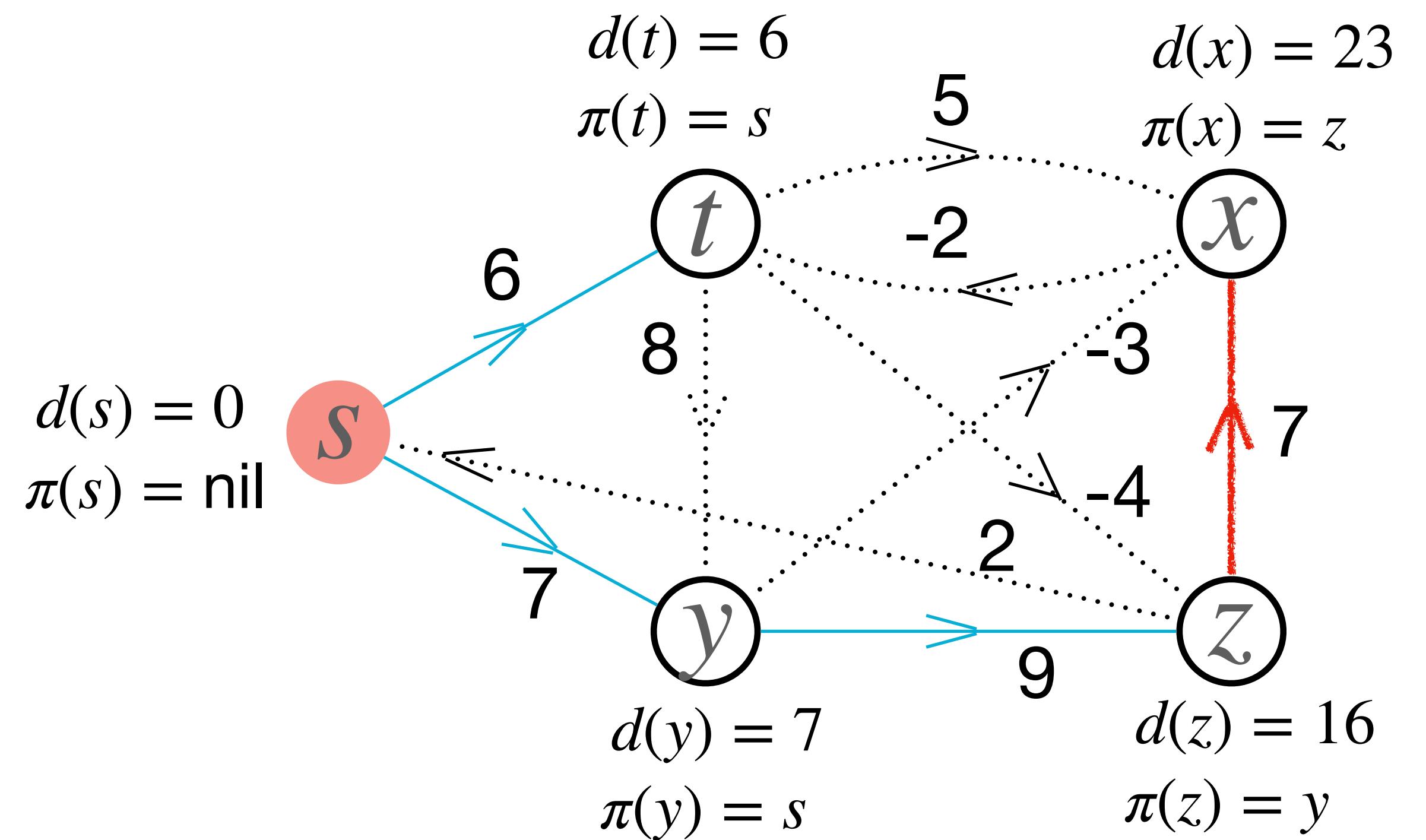
Single-Source Shortest Paths



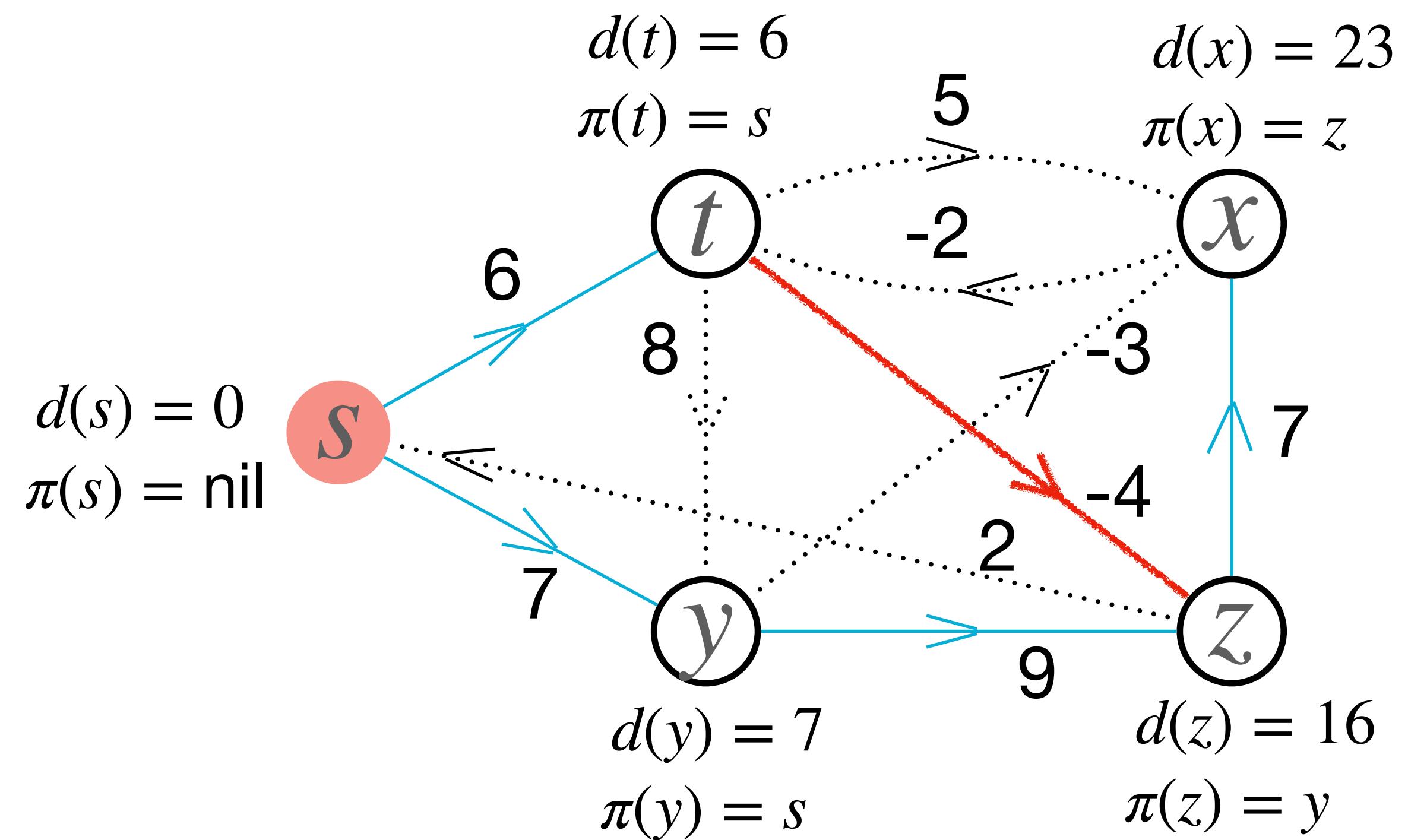
Single-Source Shortest Paths



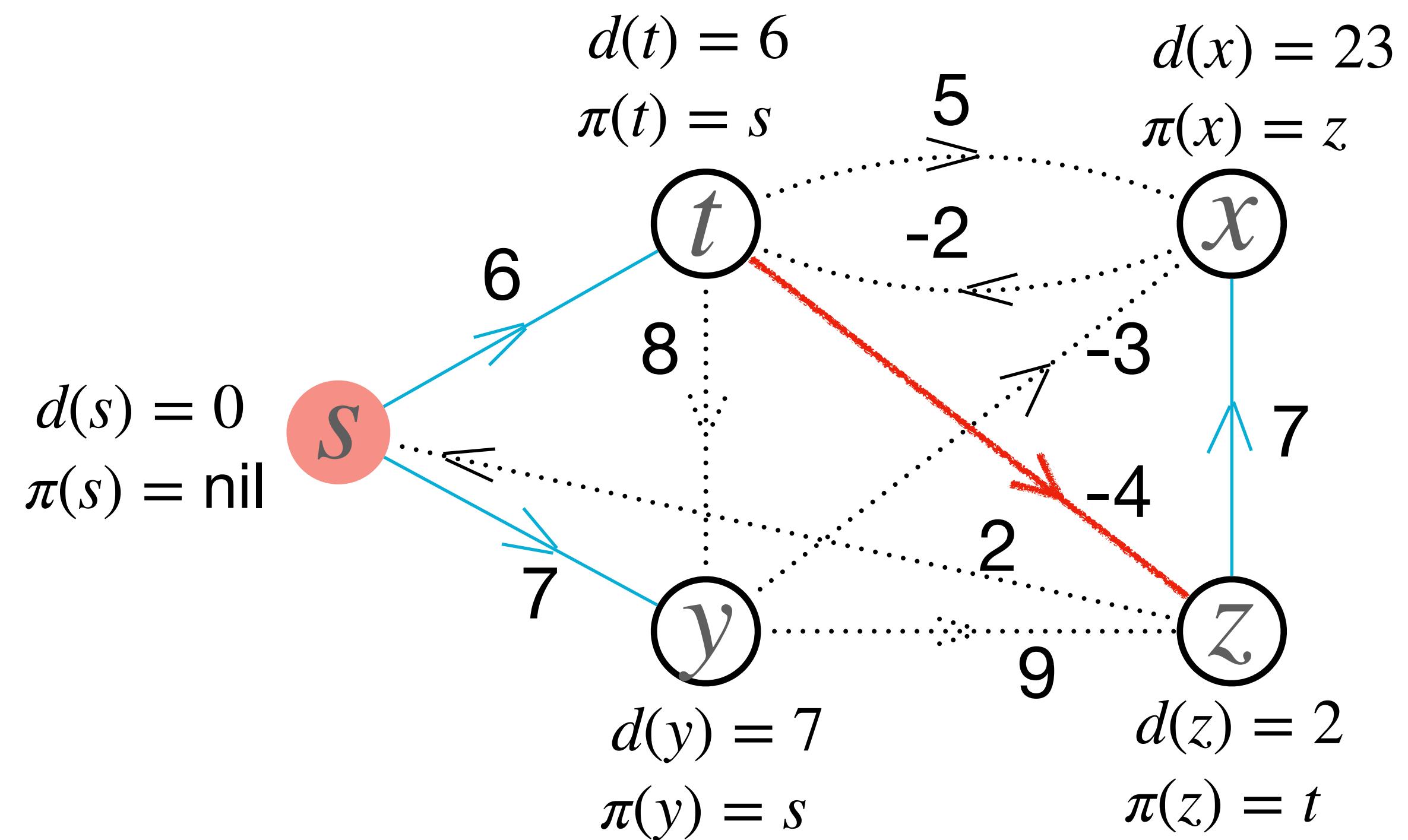
Single-Source Shortest Paths



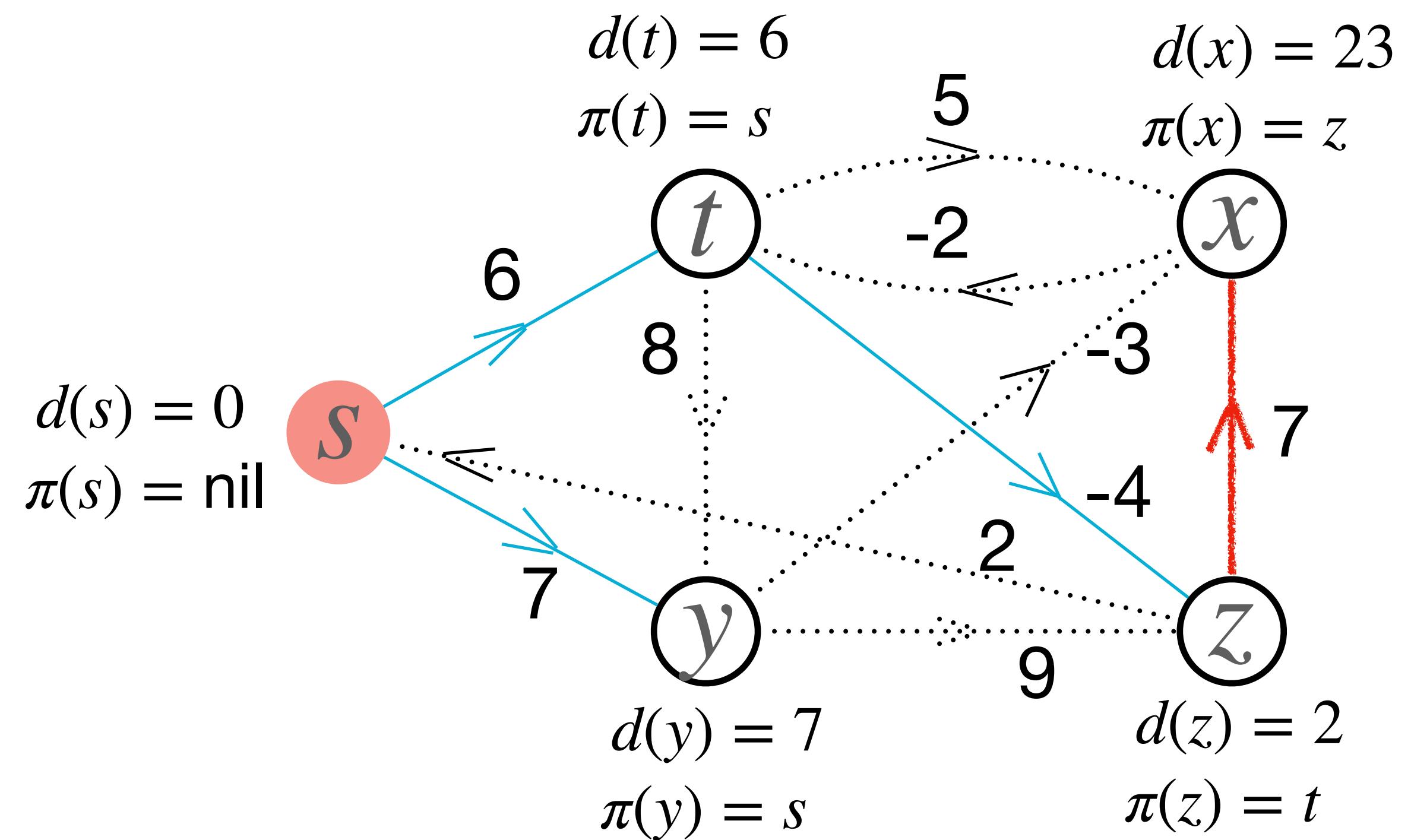
Single-Source Shortest Paths



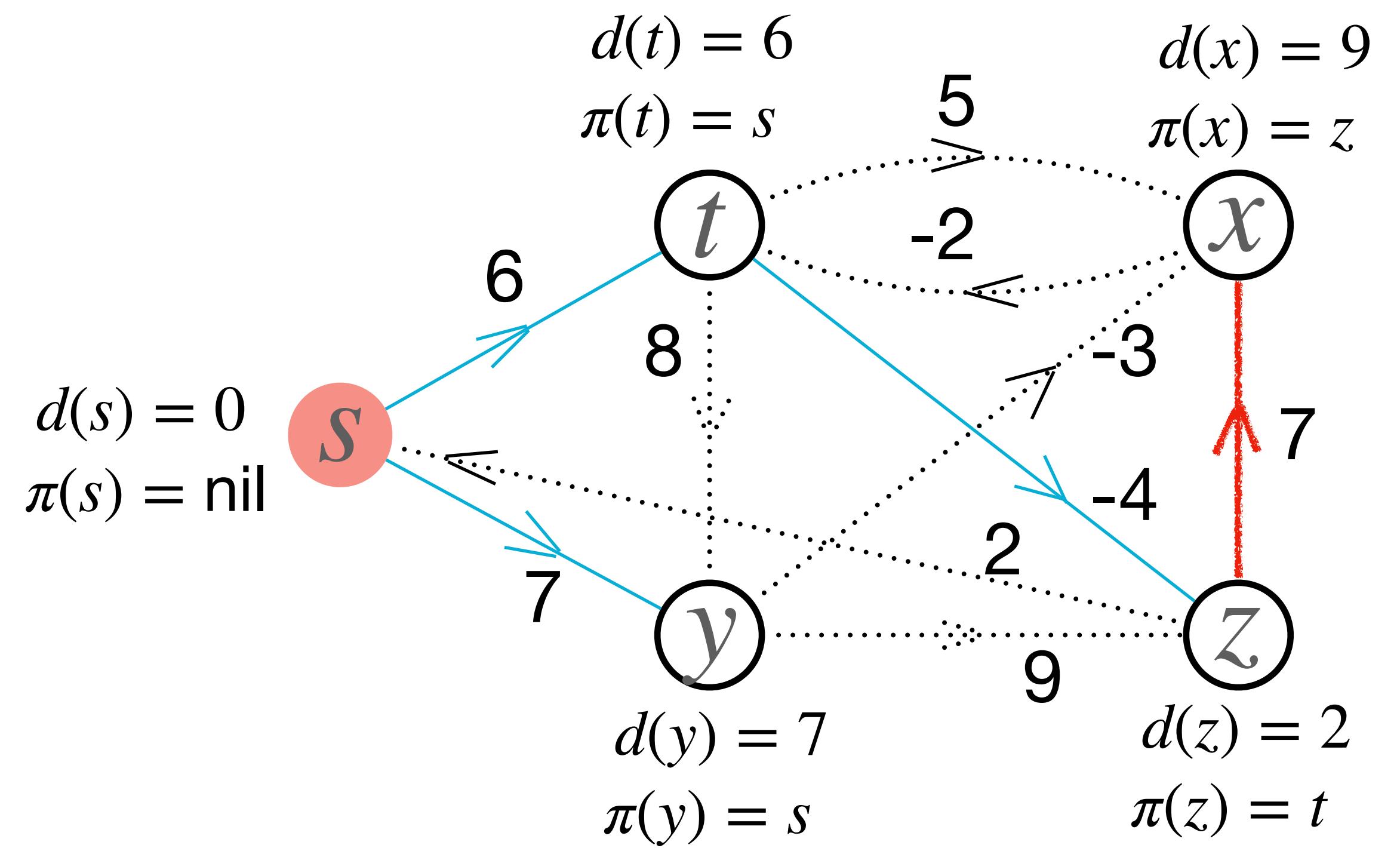
Single-Source Shortest Paths



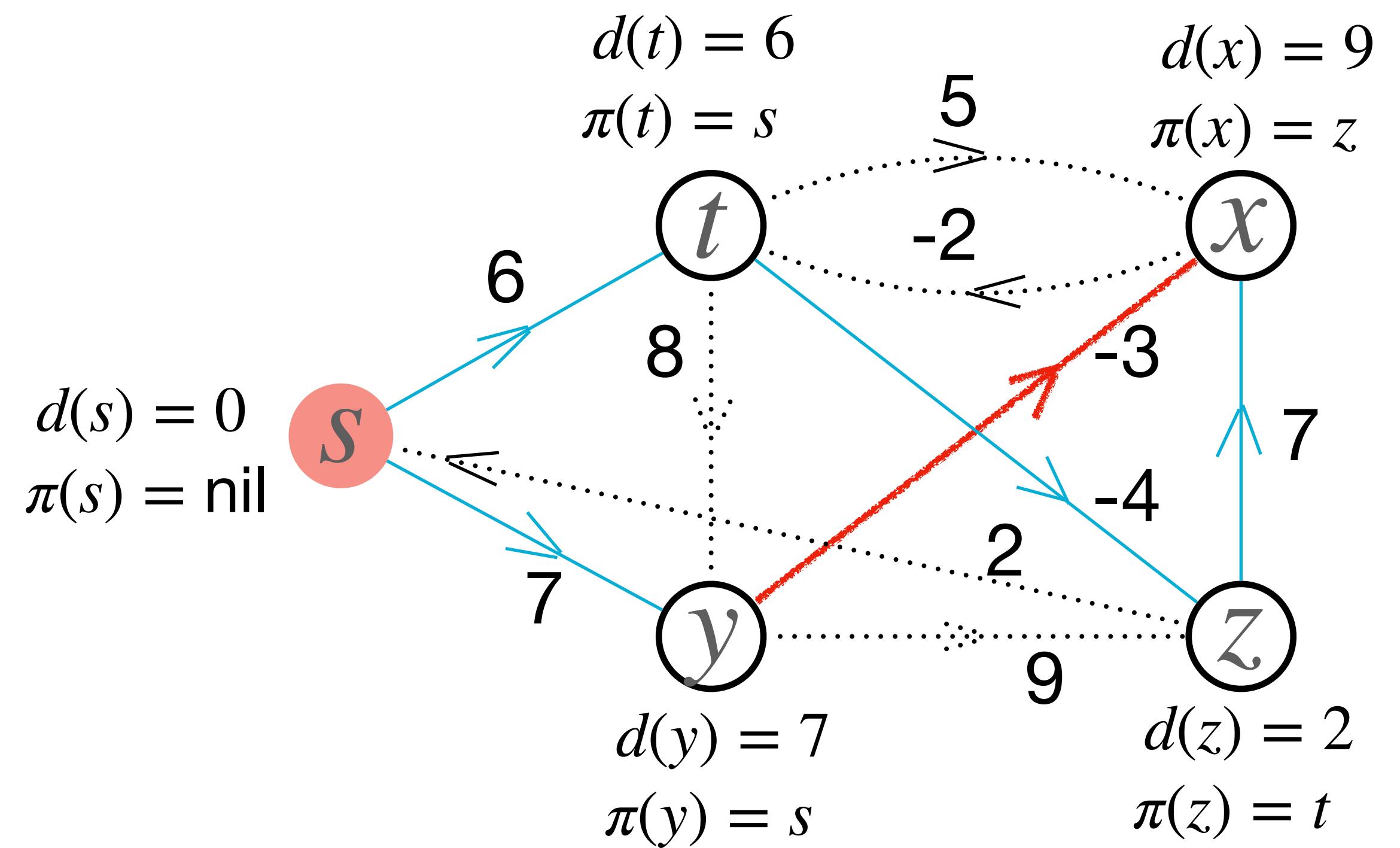
Single-Source Shortest Paths



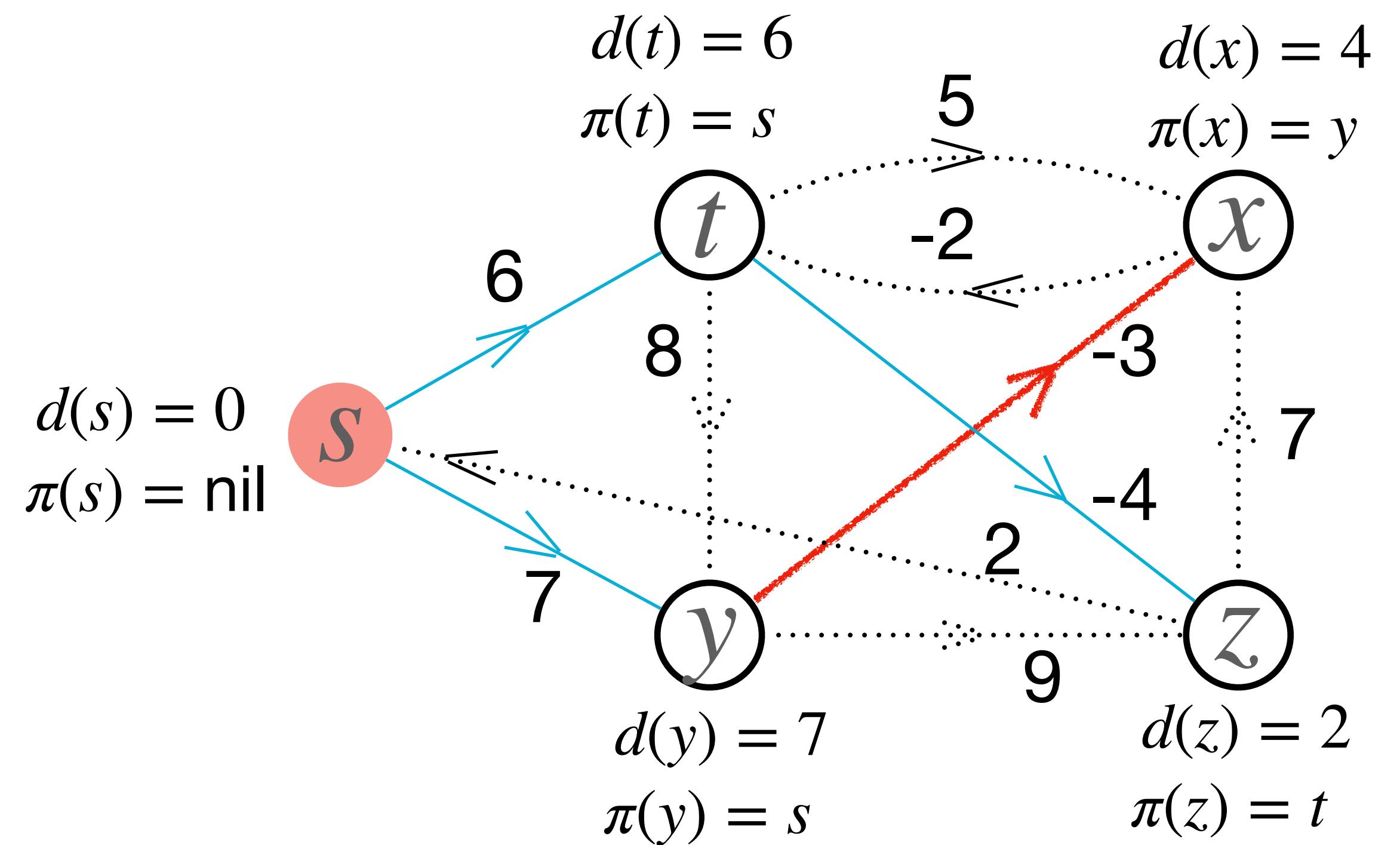
Single-Source Shortest Paths



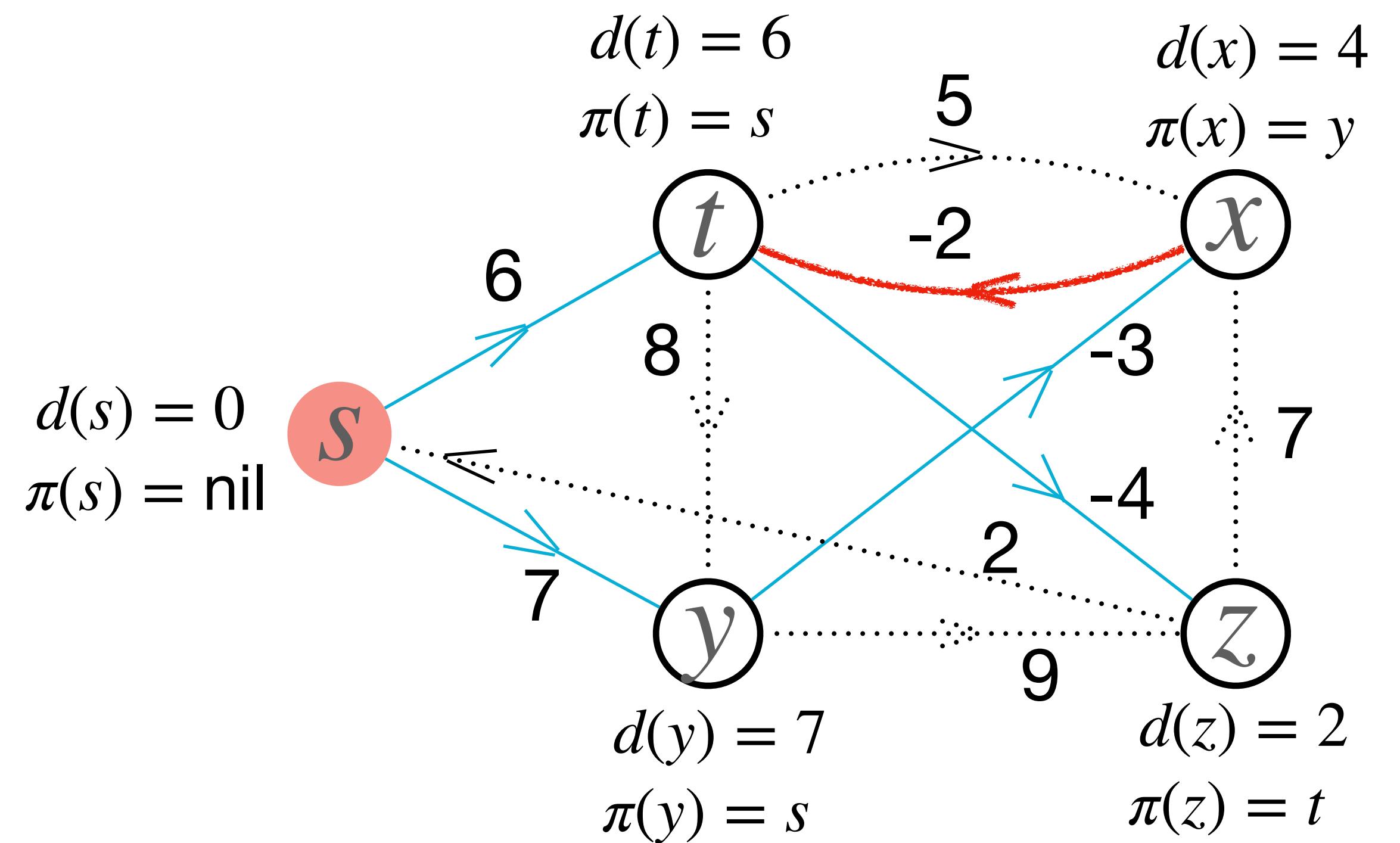
Single-Source Shortest Paths



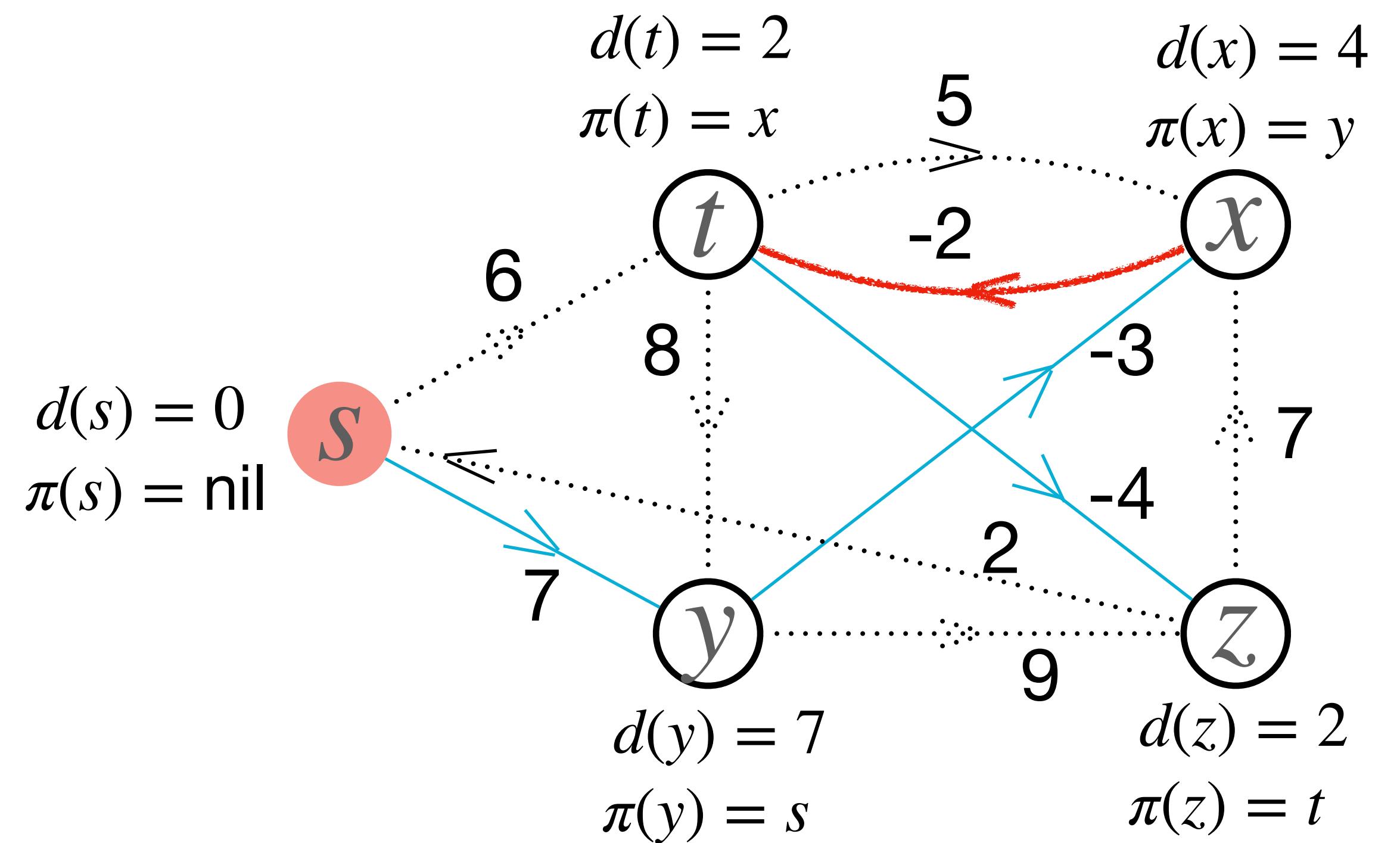
Single-Source Shortest Paths



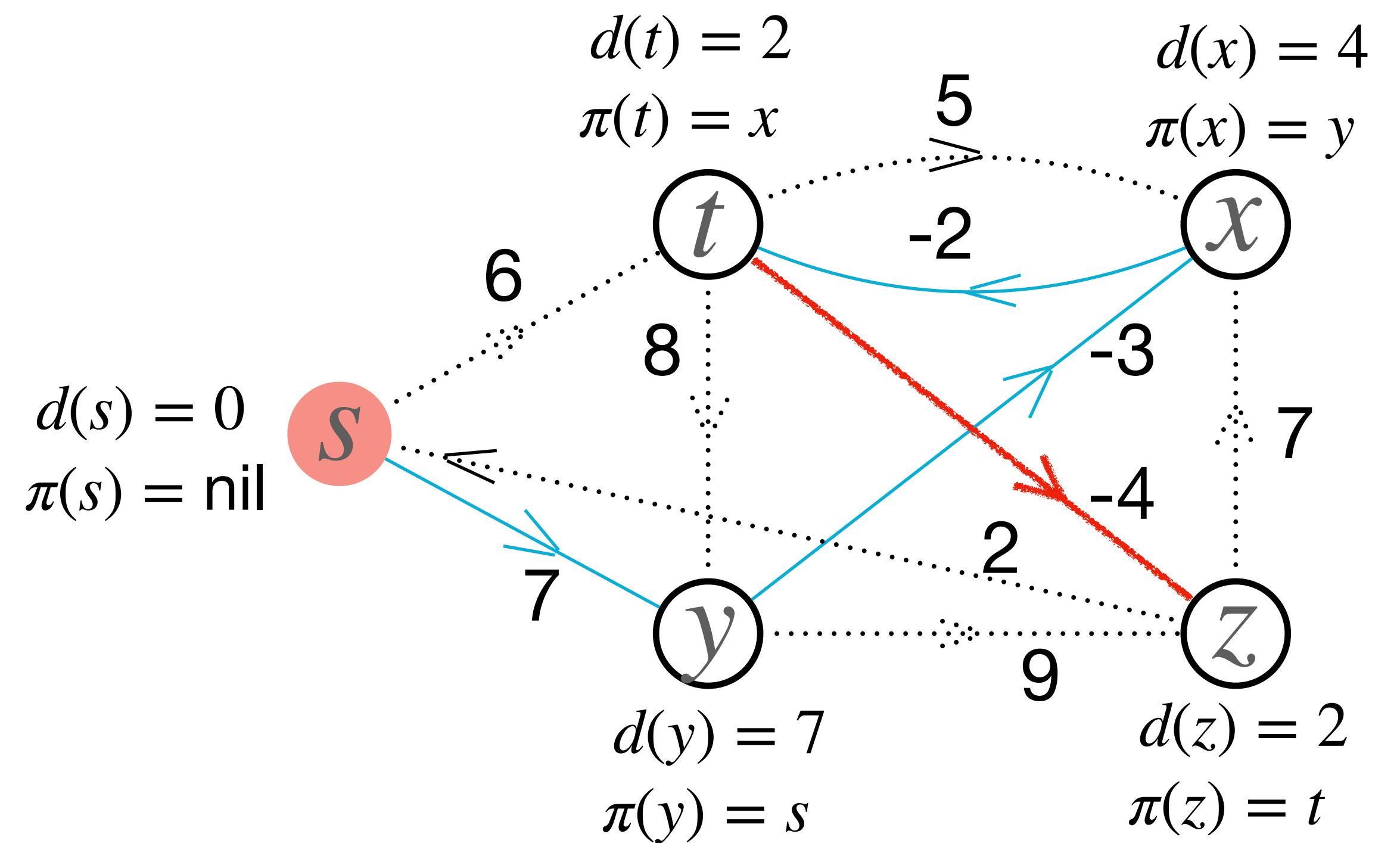
Single-Source Shortest Paths



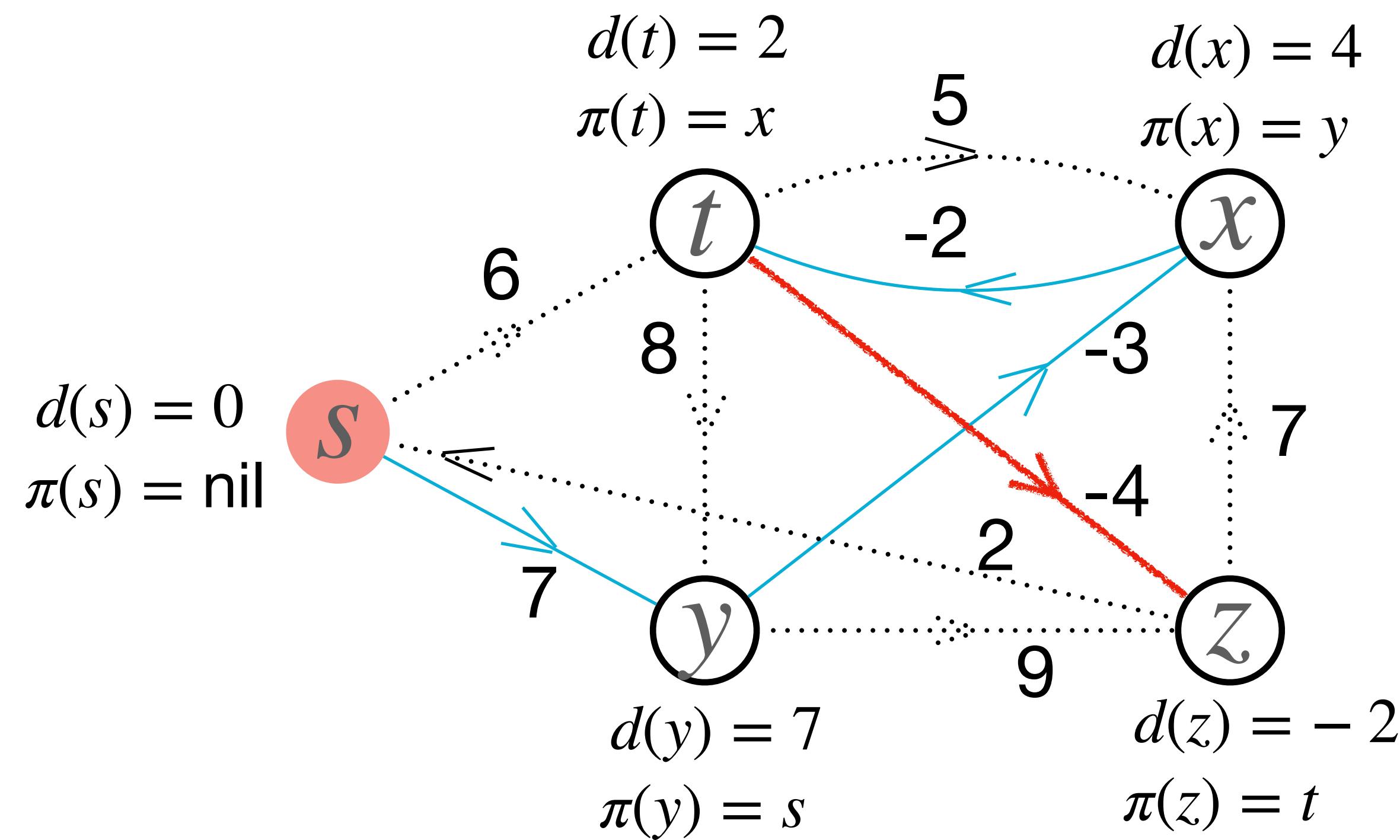
Single-Source Shortest Paths



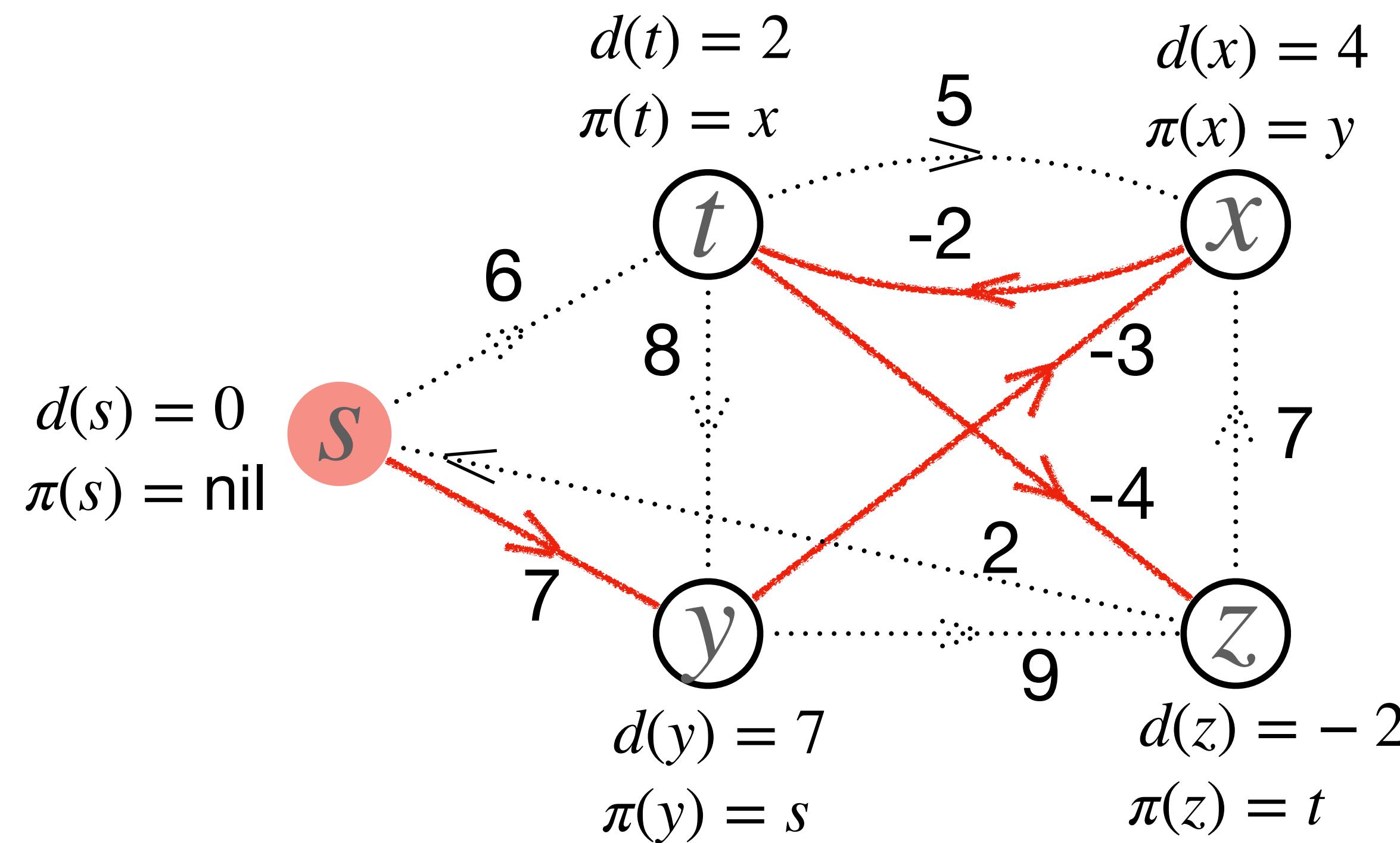
Single-Source Shortest Paths



Single-Source Shortest Paths



Single-Source Shortest Paths



Shortest-Path Tree

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

2) for $i = 1$ to $|V| - 1$:

for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$

Relaxation

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v

(which form a shortest-path tree).

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$

2) for $i = 1$ to $|V| - 1$:

for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

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

$\pi(v) = u$

Relaxation

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v

(which form a shortest-path tree).

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

2) for $i = 1$ to $|V| - 1$:

for every edge $(u, v) \in E$:

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

Do Relaxation for $|V|-1$ rounds.

In each round, do Relaxation for all edges.

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

2) for $i = 1$ to $|V| - 1$:

for every edge $(u, v) \in E$:

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Why are $|V|-1$ rounds enough?

Do Relaxation for $|V|-1$ rounds.

In each round, do Relaxation for all edges.

Does their order matter?

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

2) for $i = 1$ to $|V| - 1$:

for every edge $(u, v) \in E$:

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Are the above $|V|-1$ rounds of Relaxation
still not enough?

If still not enough, then something is wrong
(i.e, there is a negative cycle).

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

2) for $i = 1$ to $|V| - 1$:

for every edge $(u, v) \in E$:

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

$|V|-1$ rounds of Relaxation turn out to be enough.
Let's output the shortest-path tree.

Quiz questions:

1. What is the main idea of the above shortest-path algorithm?
2. What does “relaxation” mean in the above algorithm?

Roadmap of this lecture:

1. Single-Source Shortest Paths

1.1 Define the "Single-Source Shortest Paths Problem".

1.2 Bellman-Ford Algorithm for finding single-source shortest paths.

1.3 Correctness of the single-source shortest paths algorithm.

1.4 Finding single-source shortest paths in a directed acyclic graph.

Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

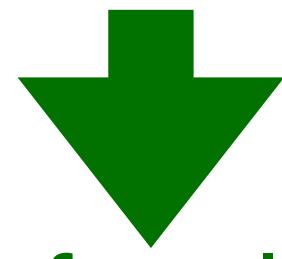
Step 2 of the
Bellman-Ford algorithm

Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges),
for every node v , $d(v)$ has become the shortest-path distance from s to v .

Proof:



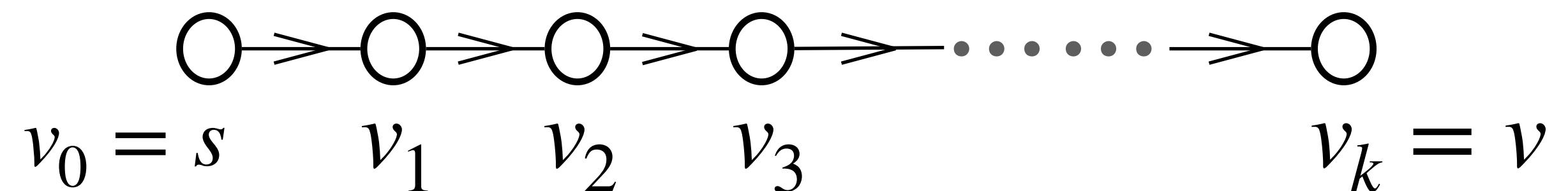
Step 2 of the
Bellman-Ford algorithm

Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

Proof: Consider a **shortest path** from s to v in graph $G=(V,E)$.



Step 2 of the
Bellman-Ford algorithm

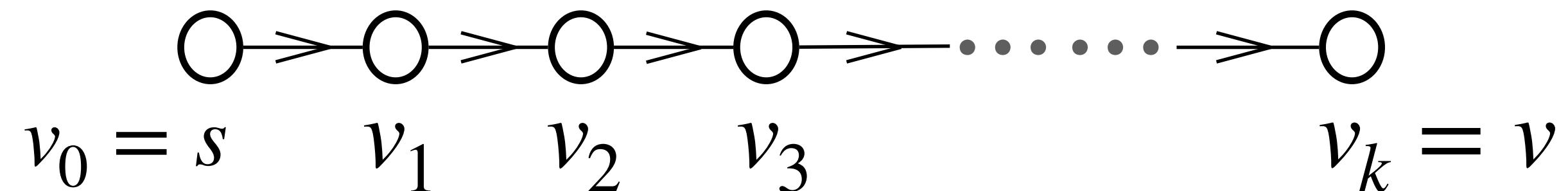
Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

Proof: Consider a **shortest path** from s to v in graph $G=(V,E)$.

Since there is no negative cycle, we can safely assume **this path contains no cycle**.



Step 2 of the
Bellman-Ford algorithm

Single-Source Shortest Paths

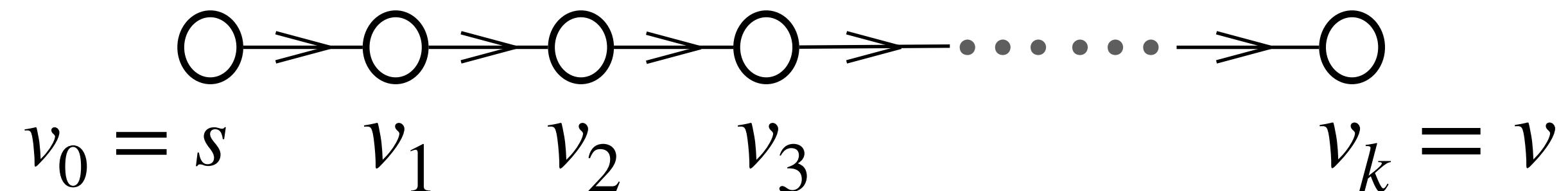
Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

Proof: Consider a **shortest path** from s to v in graph $G=(V,E)$.

Since there is no negative cycle, we can safely assume **this path contains no cycle**.

This path contains **at most $|V|-1$ edges**.



Step 2 of the
Bellman-Ford algorithm

Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

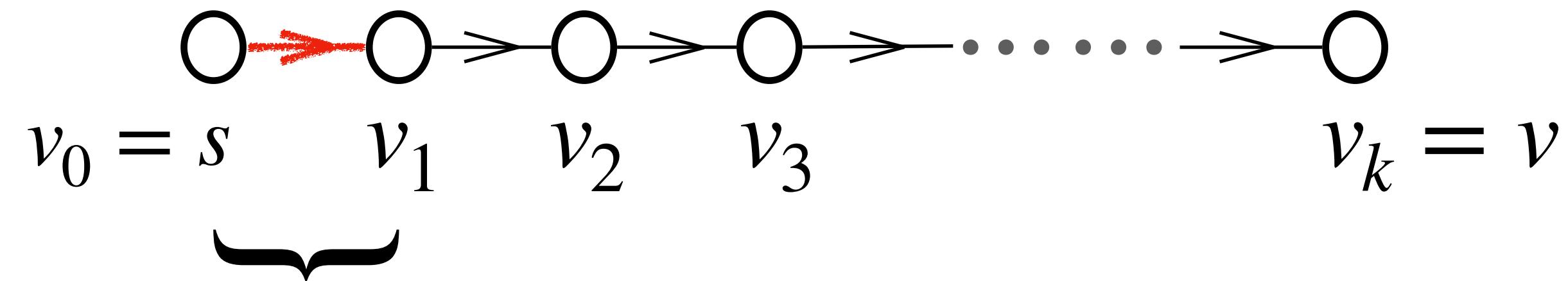
Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

Proof: Consider a **shortest path** from s to v in graph $G=(V,E)$.

Since there is no negative cycle, we can safely assume **this path contains no cycle**.

This path contains at most $|V|-1$ edges.

1) After round 1 of Relaxation, $d(v_1)$ will be the **shortest-path distance** from s to v_1 .



This sub-path is a shortest path.

Step 2 of the
Bellman-Ford algorithm

Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

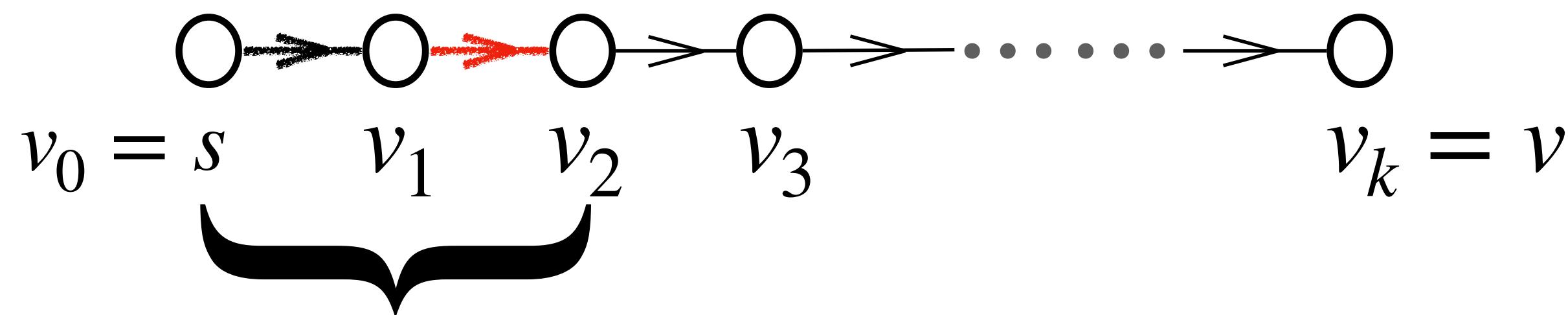


Proof: Consider a **shortest path** from s to v in graph $G=(V,E)$.

Since there is no negative cycle, we can safely assume **this path contains no cycle**.

This path contains at most $|V|-1$ edges.

- 1) After round 1 of Relaxation, $d(v_1)$ will be the shortest-path distance from s to v_1 .
- 2) After round 2 of Relaxation, $d(v_2)$ will be the **shortest-path distance** from s to v_2 .



This sub-path is a shortest path.

And so on.
Q.E.D.

Single-Source Shortest Paths

Lemma: Assume graph G has no negative cycle that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), for every node v , $d(v)$ has become the shortest-path distance from s to v .

Bellman-Ford Algorithm

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Then the “if” condition in Step 3
of Bellman-Ford Algorithm will not be true.
The algorithm will return the shortest-path tree.

Single-Source Shortest Paths

Lemma: Assume graph G **has a negative cycle** that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), there will still be some edge $(u, v) \in E$ that has $d(u) + w(u, v) < d(v)$.

Bellman-Ford Algorithm

3) for every edge $(u, v) \in E$:

if $d(u) + w(u, v) < d(v)$:

Return “there is a negative cycle
that s can reach”

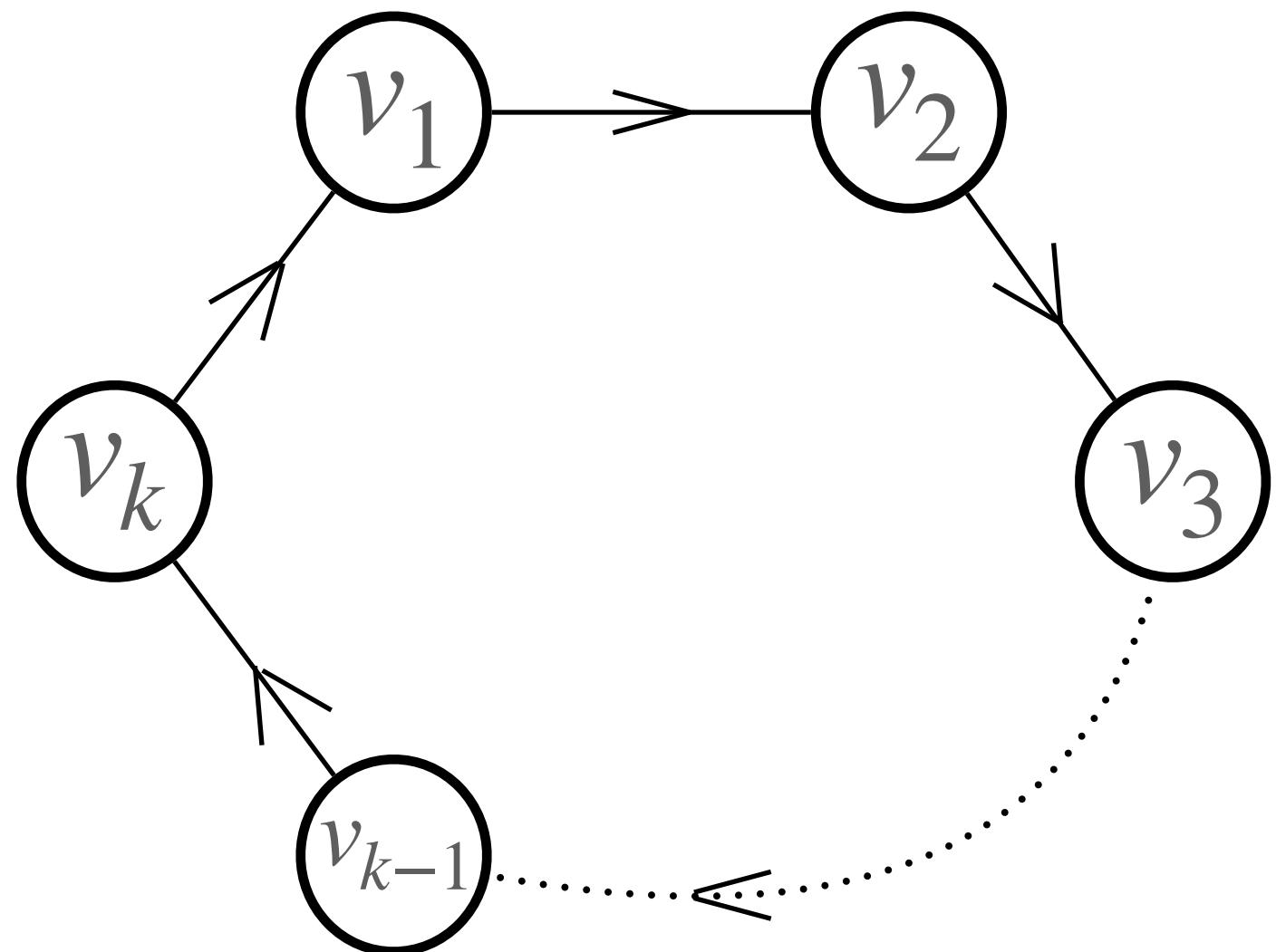
So the algorithm will return
“There is a negative cycle that s can reach”.

Single-Source Shortest Paths

Lemma: Assume graph G **has a negative cycle** that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), there will still be some edge $(u, v) \in E$ that has $d(u) + w(u, v) < d(v)$.

Proof: Consider **a negative cycle** that s can reach.

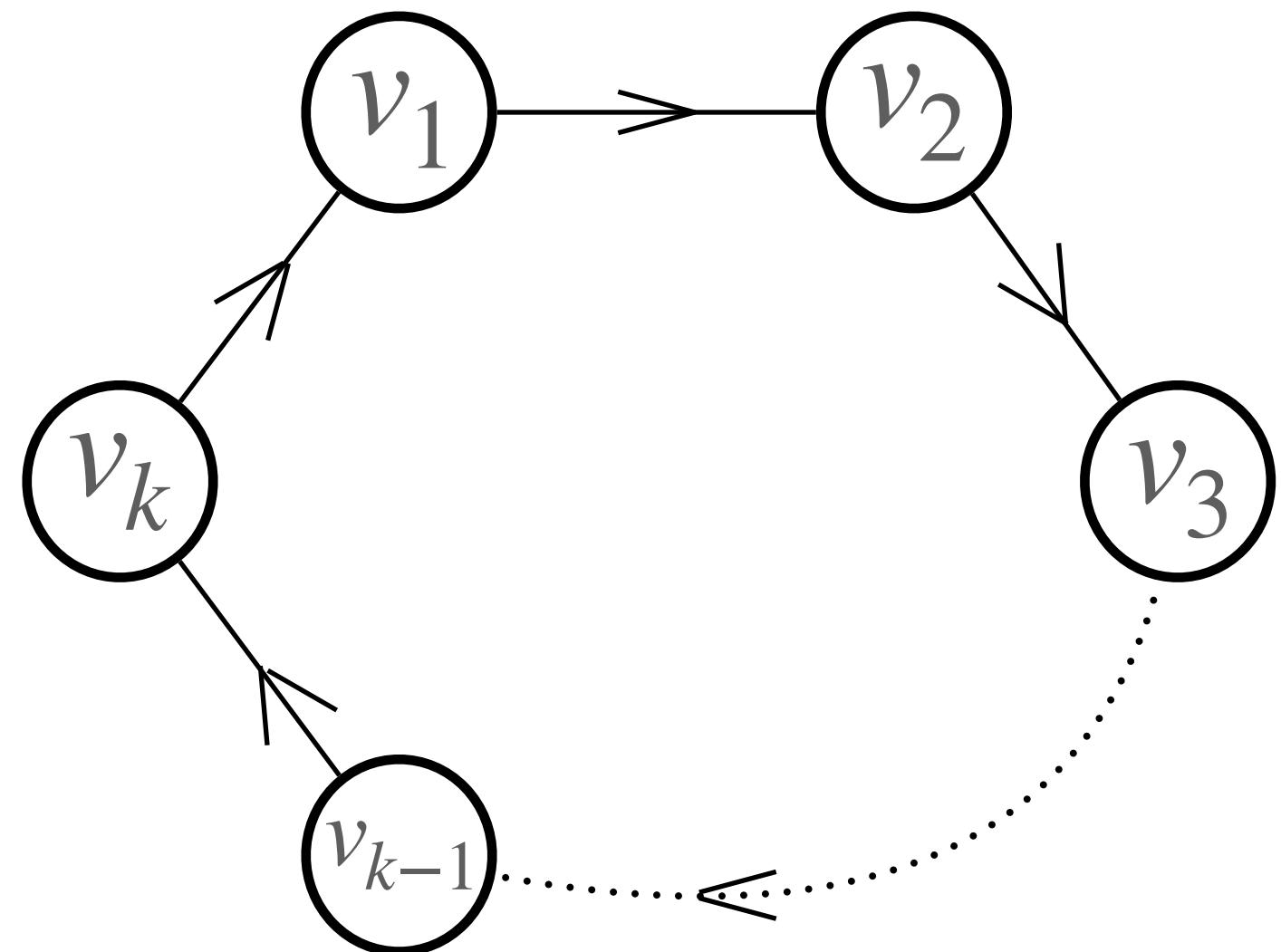


Single-Source Shortest Paths

Lemma: Assume graph G **has a negative cycle** that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), there will still be some edge $(u, v) \in E$ that has $d(u) + w(u, v) < d(v)$.

Proof: Consider **a negative cycle** that s can reach.



Assume:

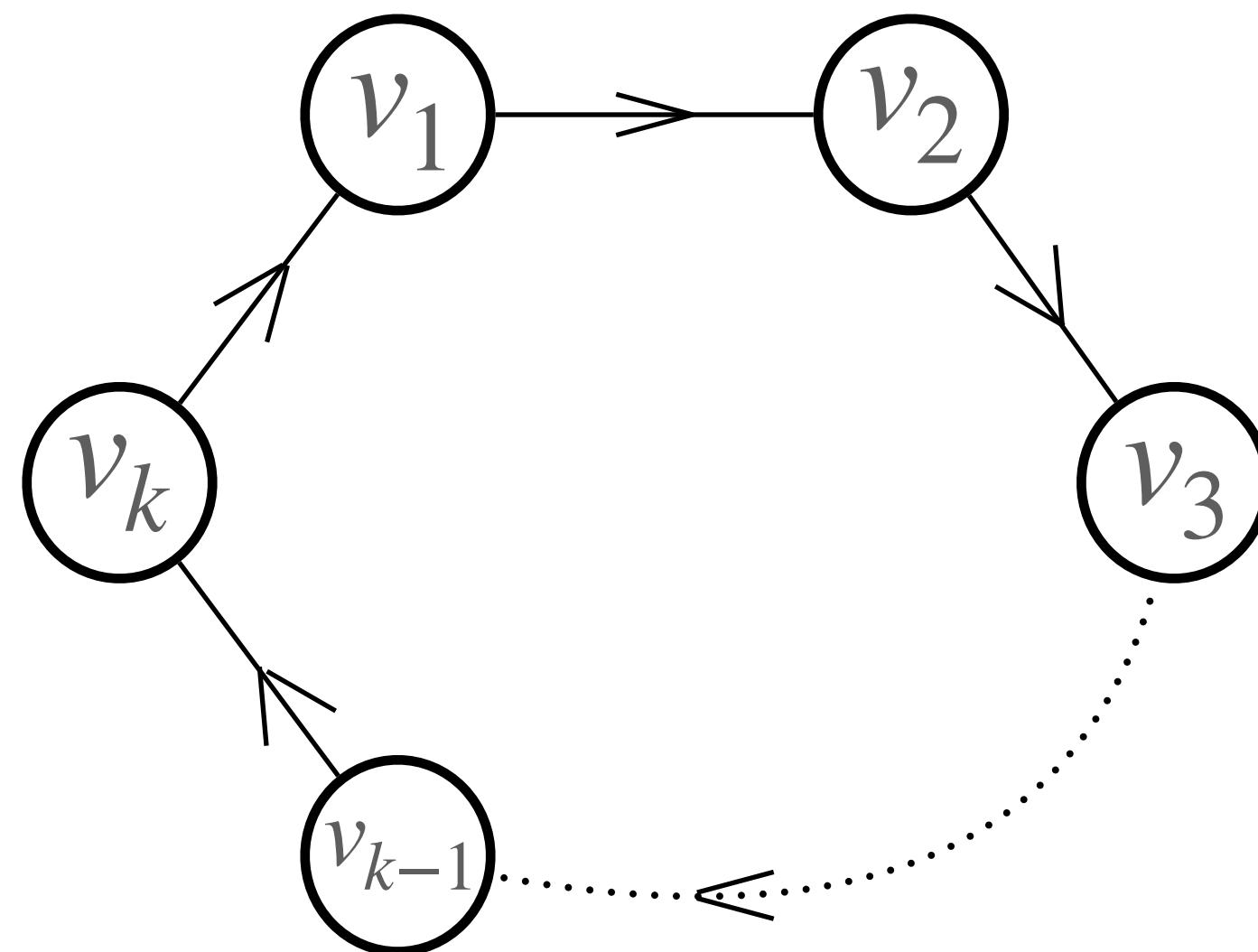
$$\begin{aligned}d(v_1) + w(v_1, v_2) &\geq d(v_2) \\d(v_2) + w(v_2, v_3) &\geq d(v_3) \\d(v_3) + w(v_3, v_4) &\geq d(v_4) \\\vdots \\d(v_{k-1}) + w(v_{k-1}, v_k) &\geq d(v_k) \\d(v_k) + w(v_k, v_1) &\geq d(v_1)\end{aligned}$$

Single-Source Shortest Paths

Lemma: Assume graph G **has a negative cycle** that s can reach.

Then after $|V|-1$ rounds of Relaxation (where each round does Relaxation for all edges), there will still be some edge $(u, v) \in E$ that has $d(u) + w(u, v) < d(v)$.

Proof: Consider **a negative cycle** that s can reach.



Assume:

$$\begin{aligned}d(v_1) + w(v_1, v_2) &\geq d(v_2) \\d(v_2) + w(v_2, v_3) &\geq d(v_3) \\d(v_3) + w(v_3, v_4) &\geq d(v_4) \\\vdots \\d(v_{k-1}) + w(v_{k-1}, v_k) &\geq d(v_k) \\d(v_k) + w(v_k, v_1) &\geq d(v_1)\end{aligned}$$

Sum them: $w(v_1, v_2) + w(v_2, v_3) + \dots + w(v_k, v_1) \geq 0$

The total weight of the negative cycle is non-negative! Contradiction.

Quiz questions:

1. Why is it enough to do $|V| - 1$ rounds of relaxation in the algorithm?
2. Why can the algorithm find out if there is a negative cycle reachable by the source in the graph?

Roadmap of this lecture:

1. Single-Source Shortest Paths

1.1 Define the "Single-Source Shortest Paths Problem".

1.2 Bellman-Ford Algorithm for finding single-source shortest paths.

1.3 Correctness of the single-source shortest paths algorithm.

1.4 Finding single-source shortest paths in a directed acyclic graph.

Single-Source Shortest Paths

Bellman-Ford Algorithm

1) Initialization: $\forall v \in V, \pi(v) = \text{nil}$
 $d(s) = 0 \quad \forall v \neq s, d(v) = \infty$

2) for $i = 1$ to $|V| - 1$:
 for every edge $(u, v) \in E$: **Relaxation**

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

3) for every edge $(u, v) \in E$:
 if $d(u) + w(u, v) < d(v)$:
 Return “there is a negative cycle
 that s can reach”

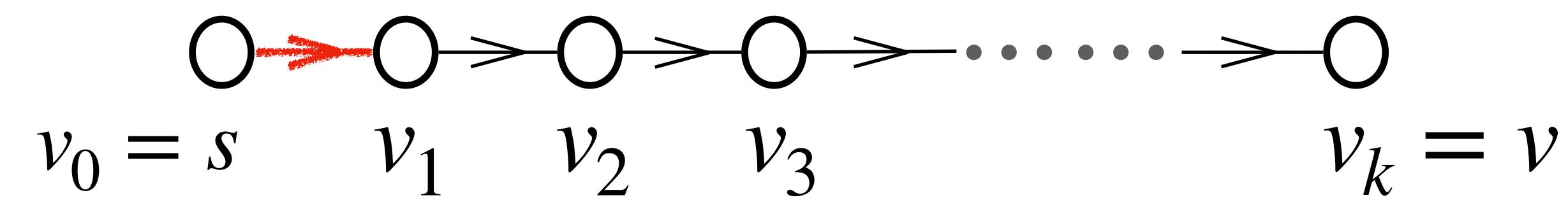
4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Time Complexity: $O(VE)$

This time complexity is actually
relatively high for very large graphs.

Difficulty: we do not know the order of
edges in the “shortest paths”. If we do,
We can do relaxation following the order.

Are there graphs where the “order” can be known?



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v

(which form a shortest-path tree).

Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

A simplified Bellman-Ford Algorithm where:

- 1) Every edge is relaxed only once.
- 2) No need to check negative cycle.

Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

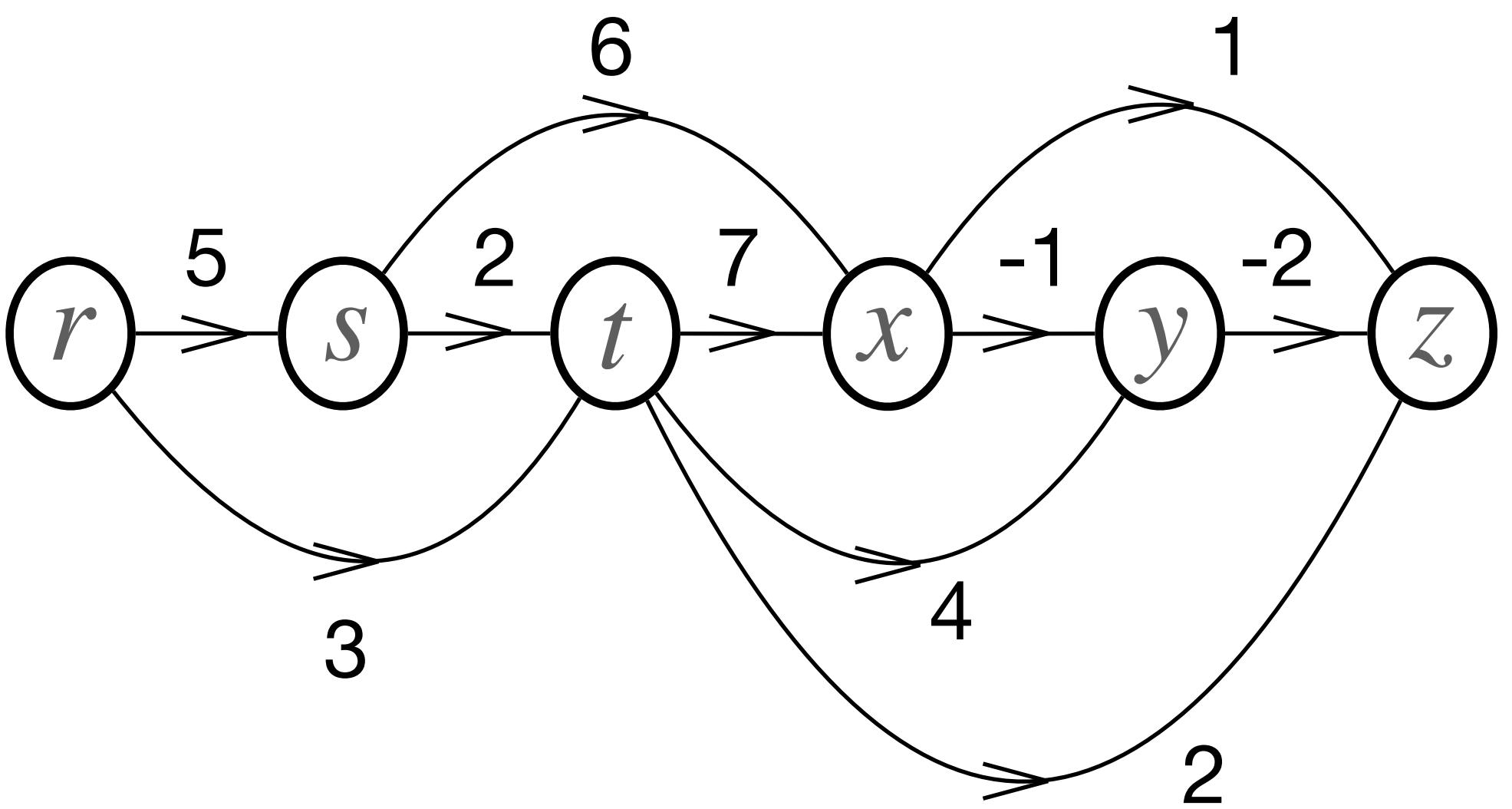
- 1) Topologically sort the graph $G=(V,E)$
- 2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$
 $d(s) = 0 \quad \forall v \neq s, d(v) = \infty$

- 3) For each node u , taken in the topologically sorted order:
for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

- 4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

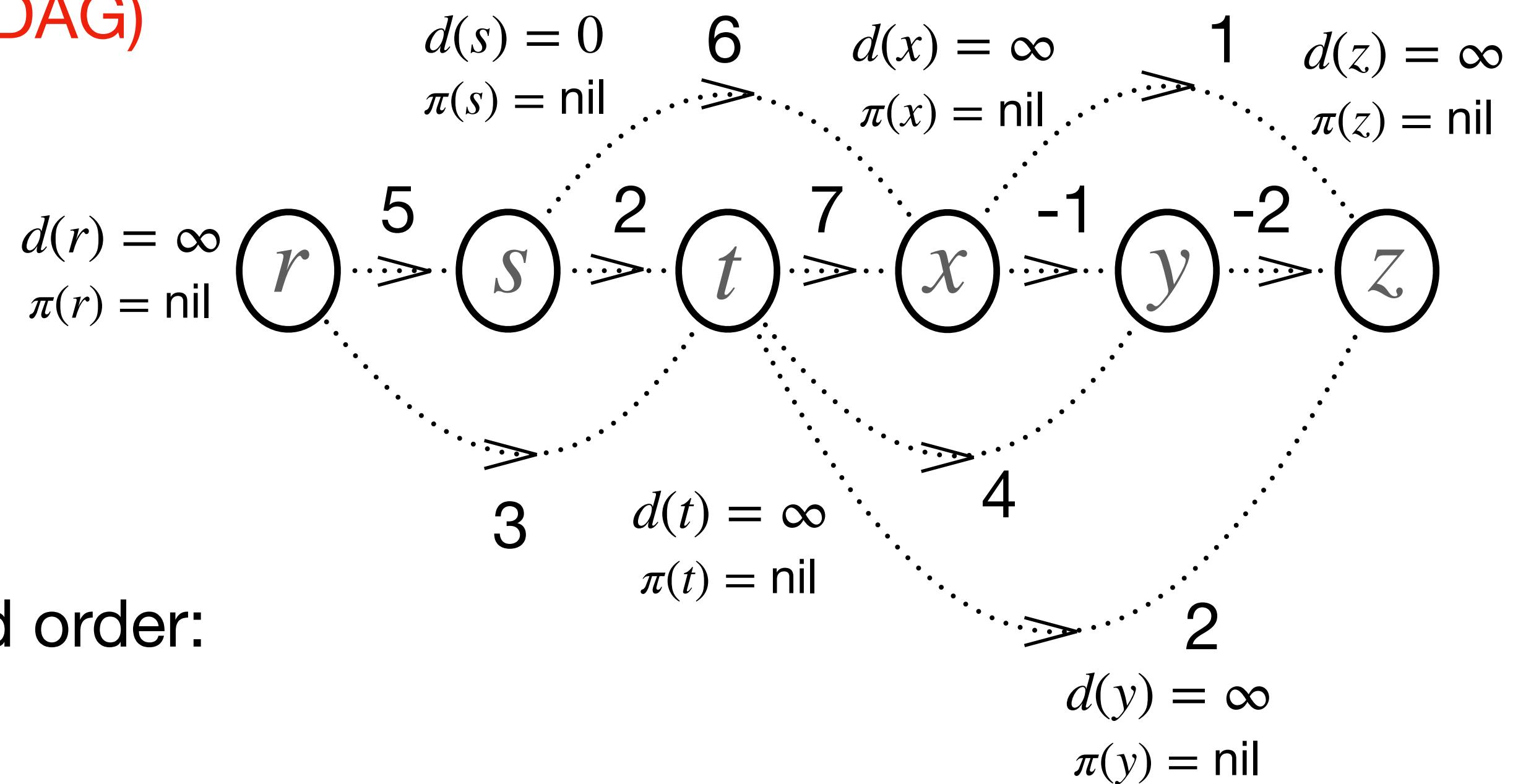
if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

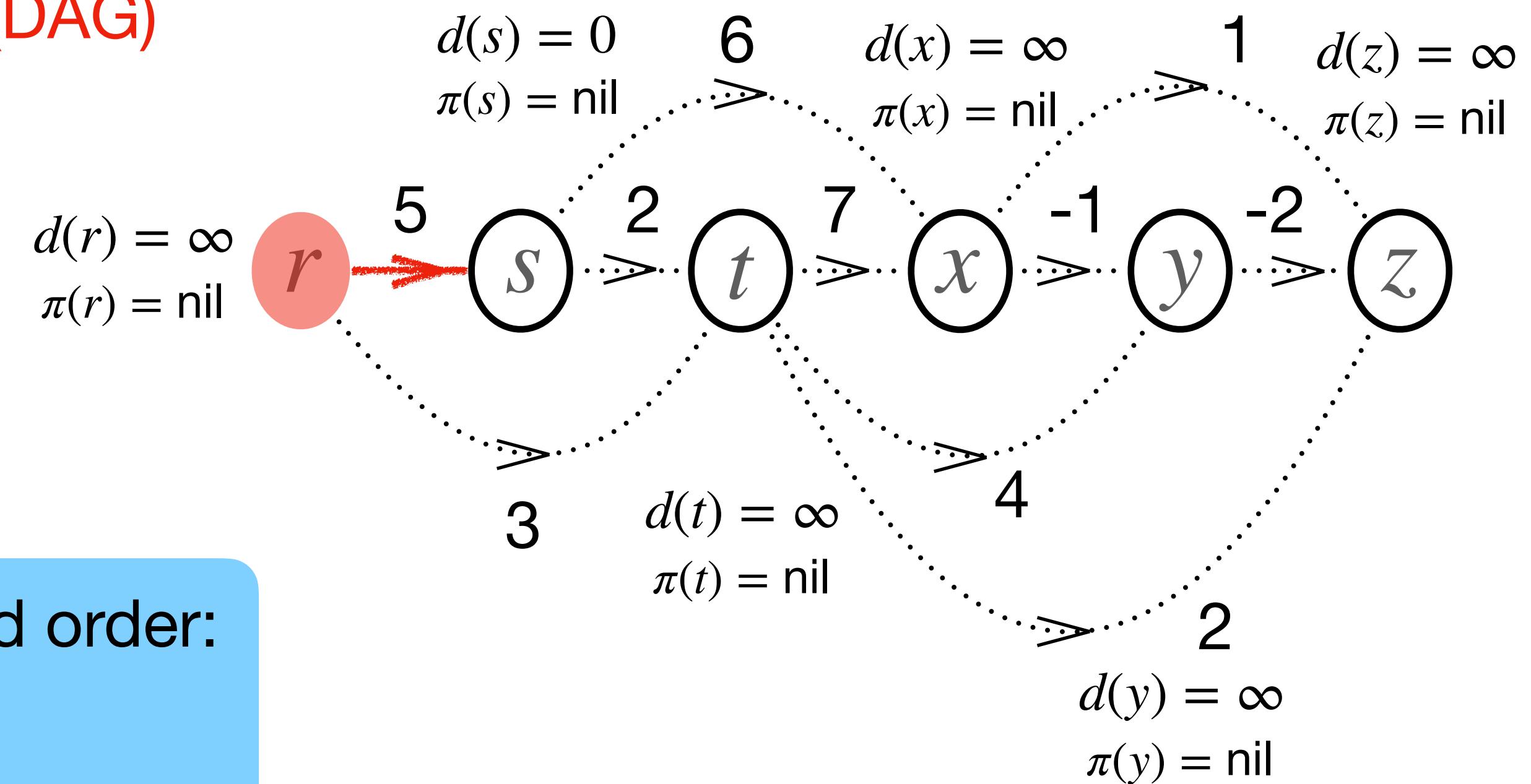
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

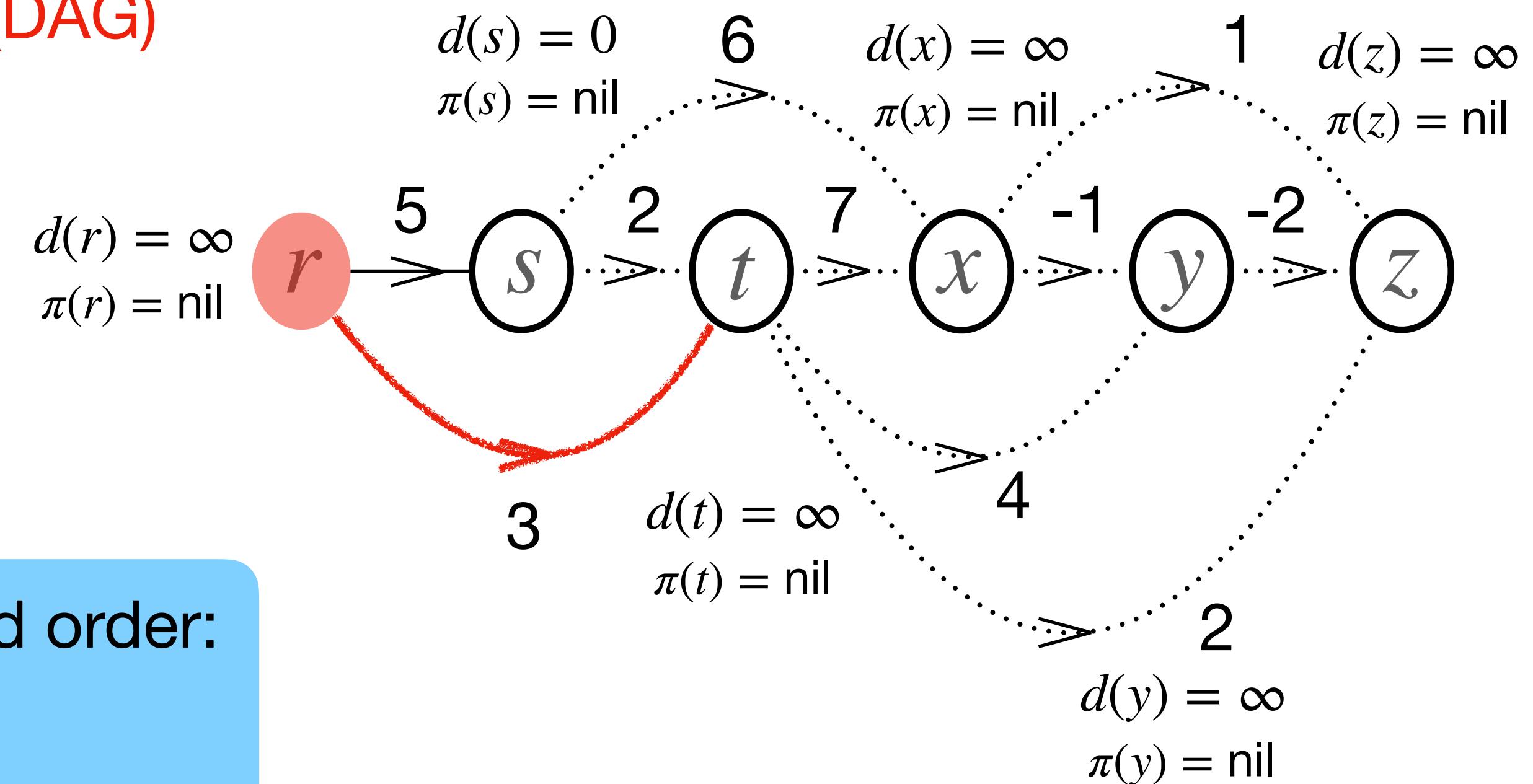
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

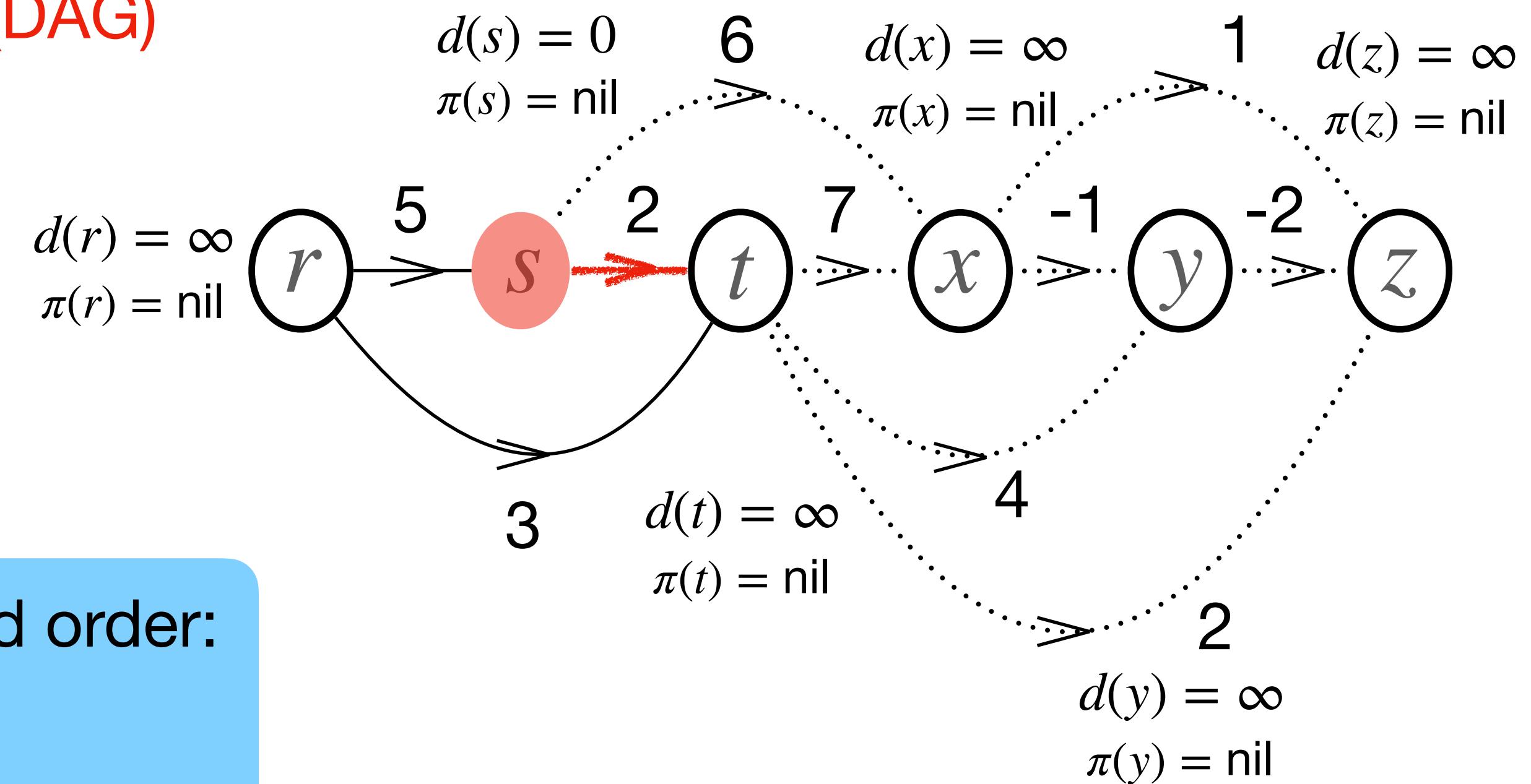
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

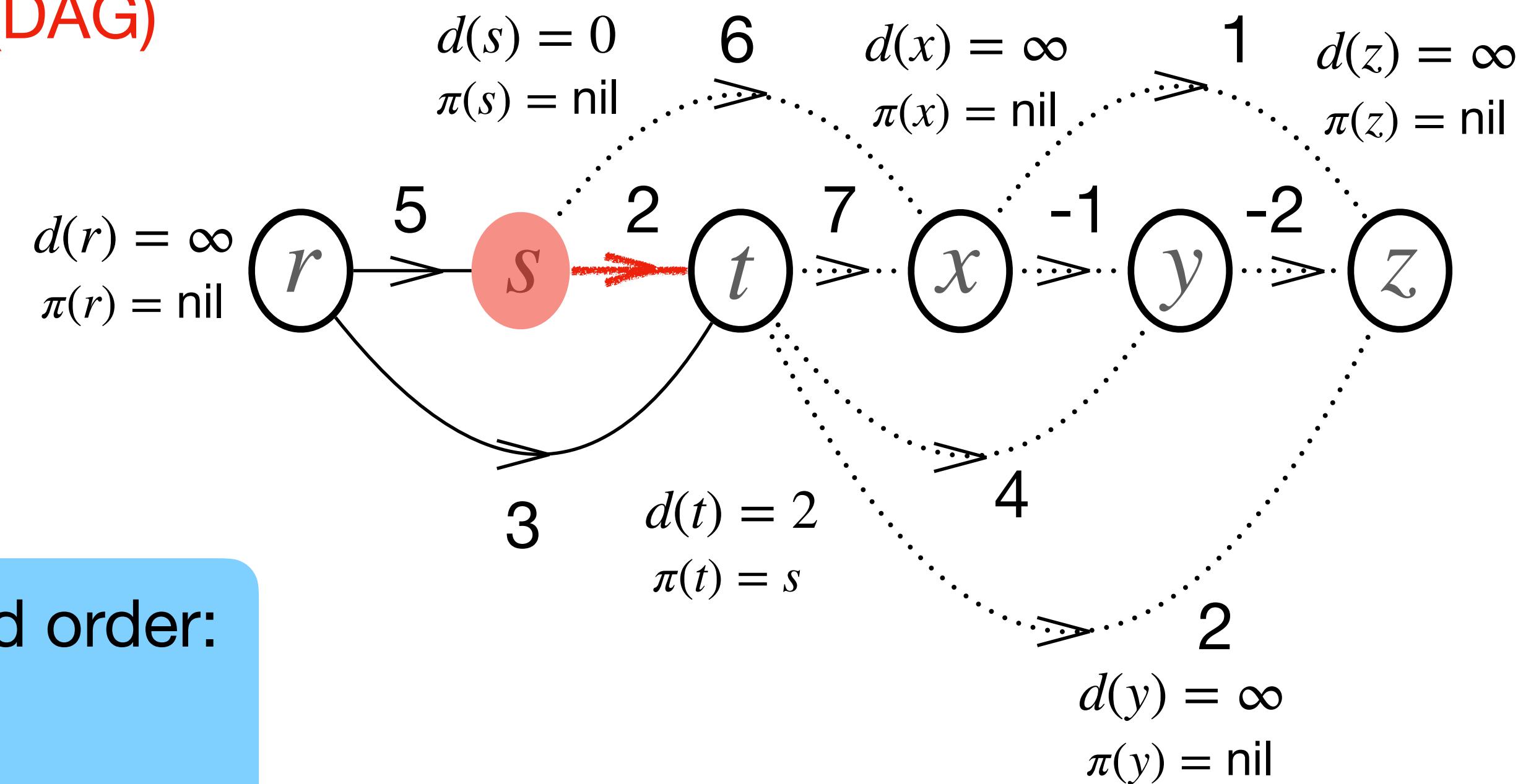
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

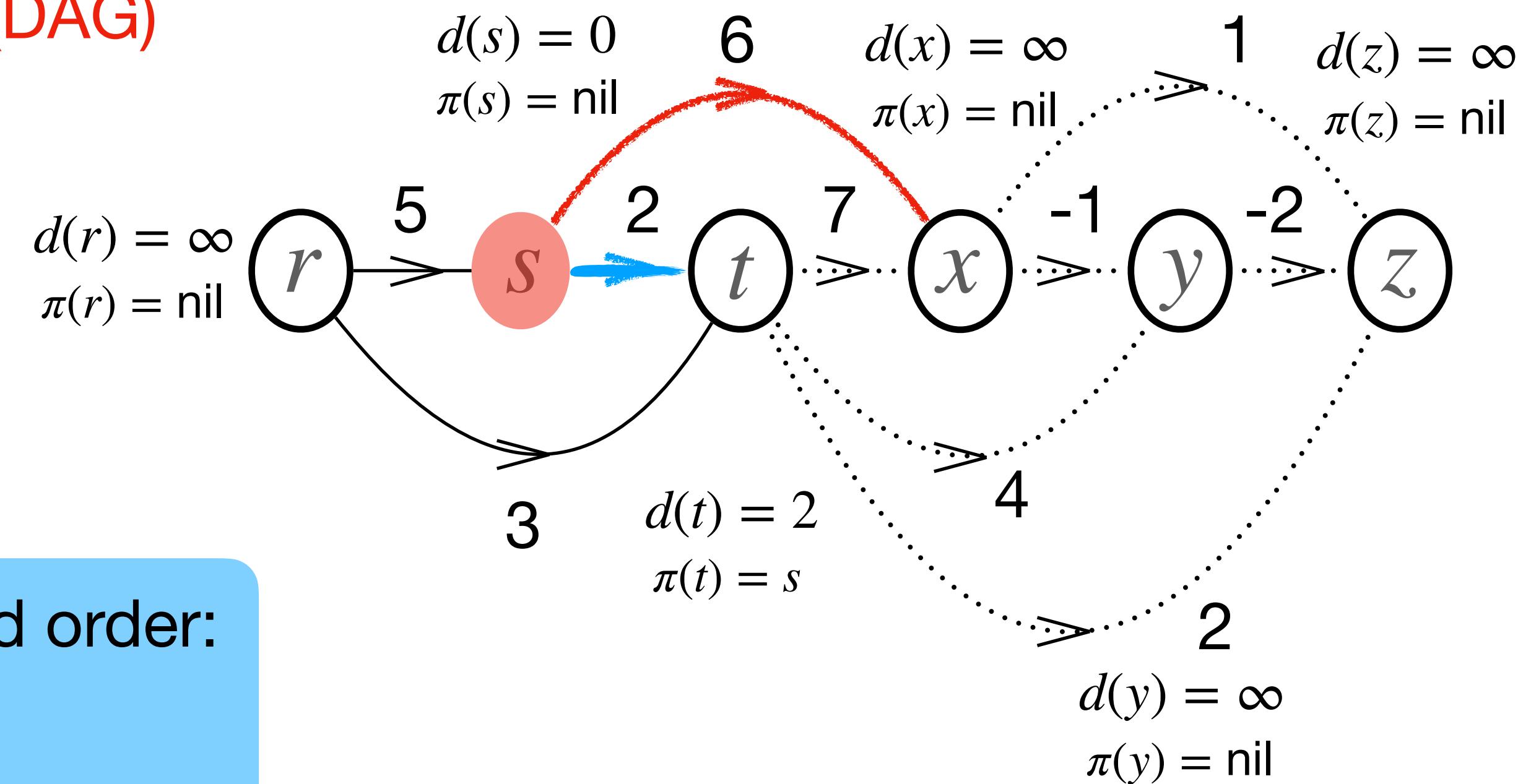
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

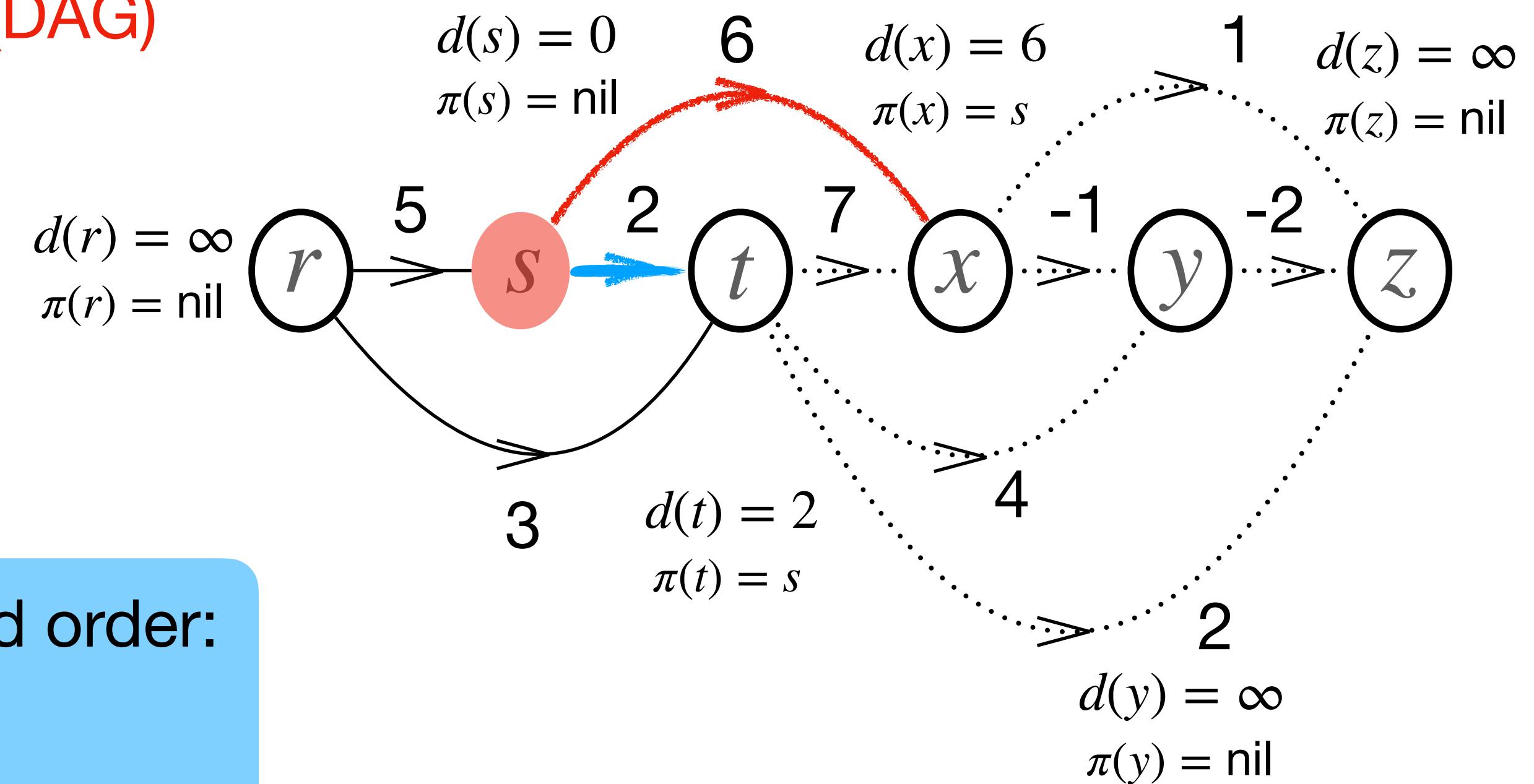
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

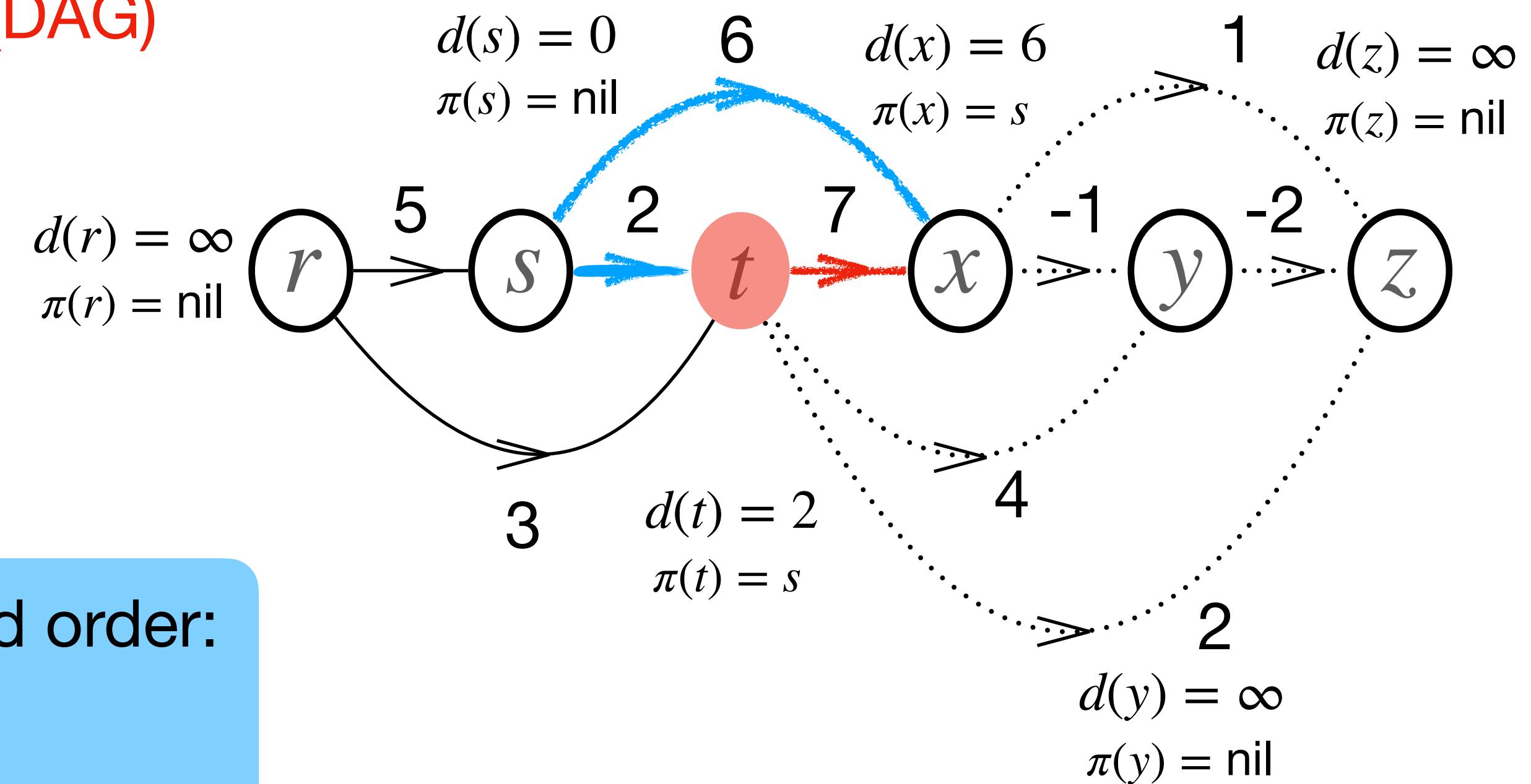
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

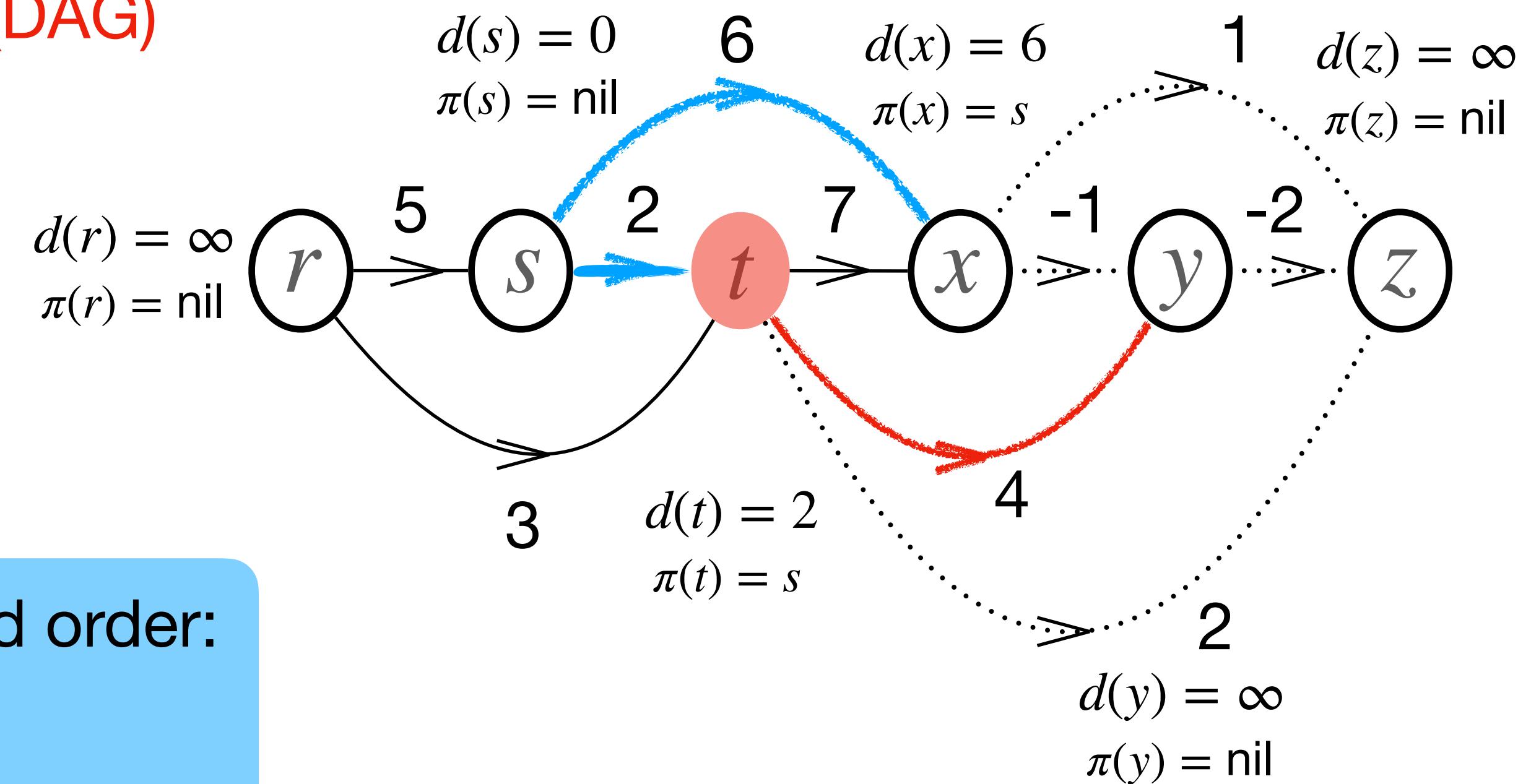
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

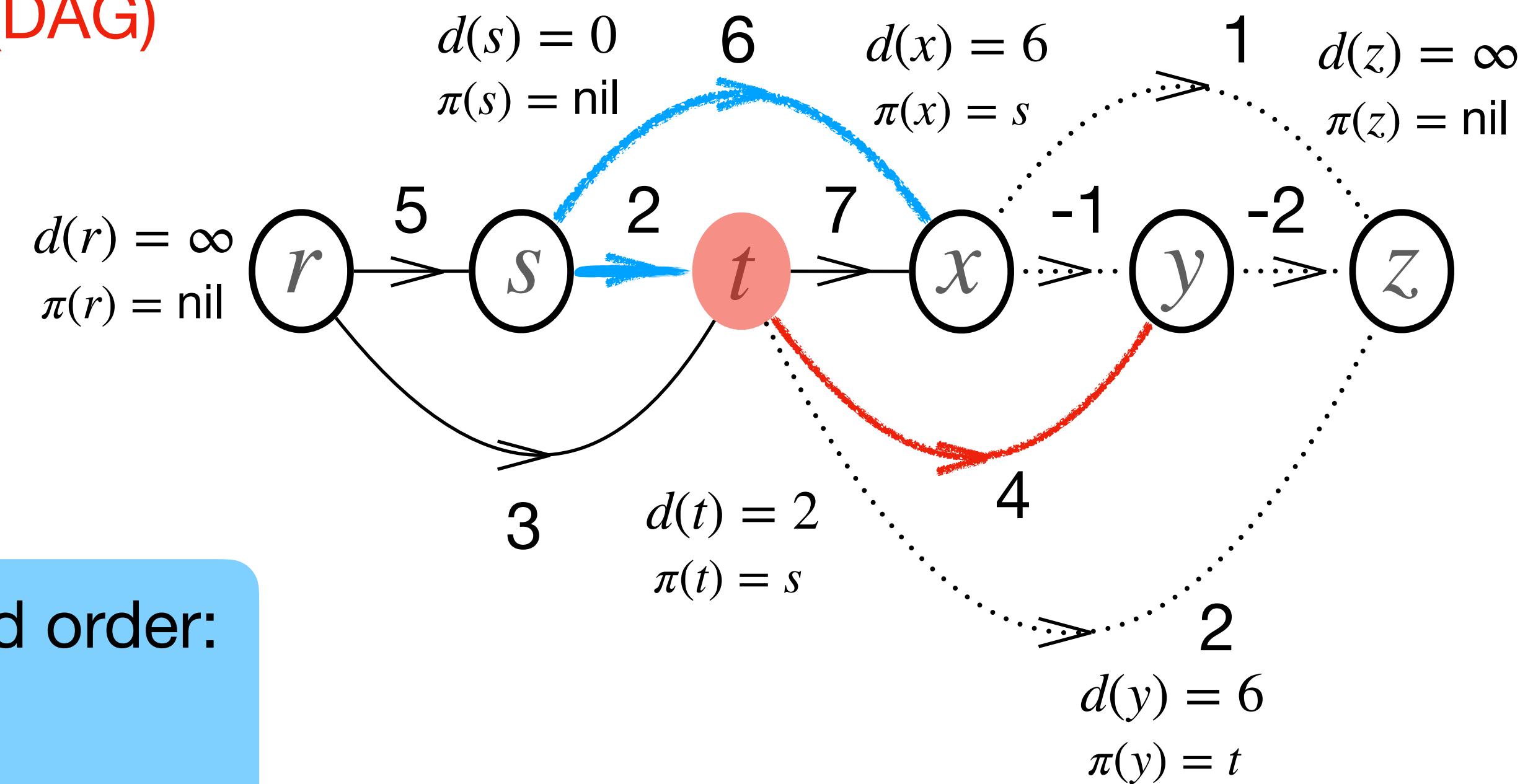
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

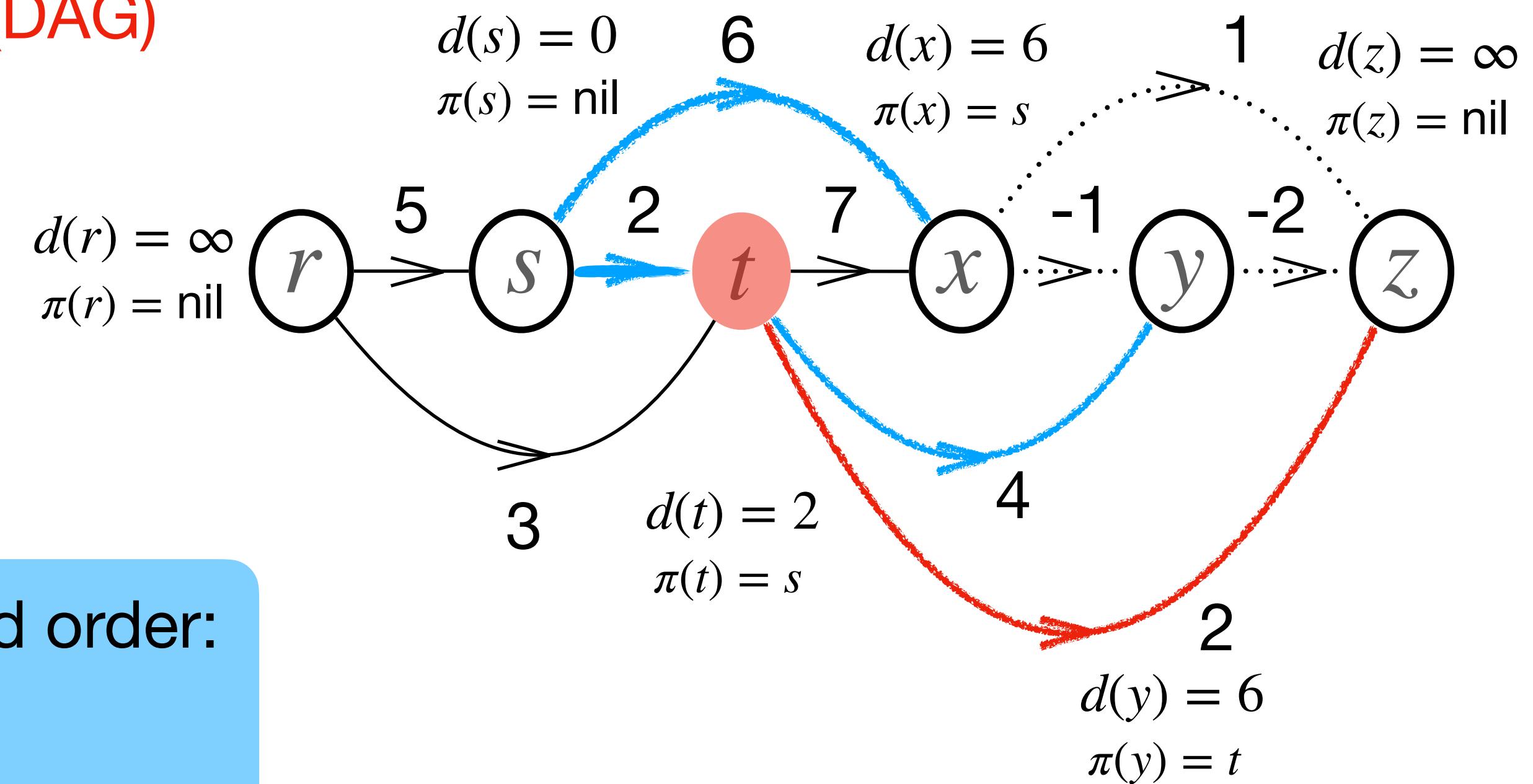
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

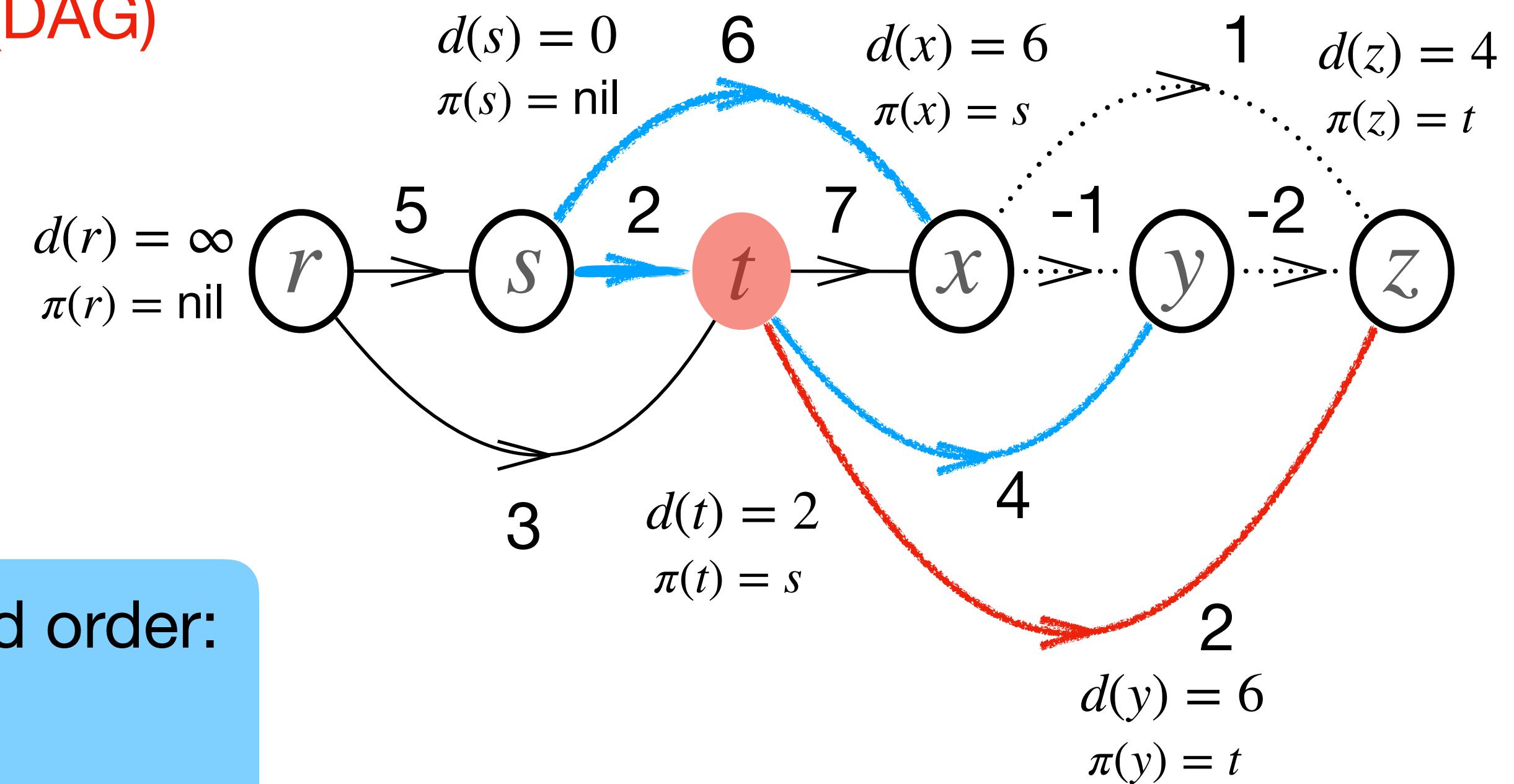
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

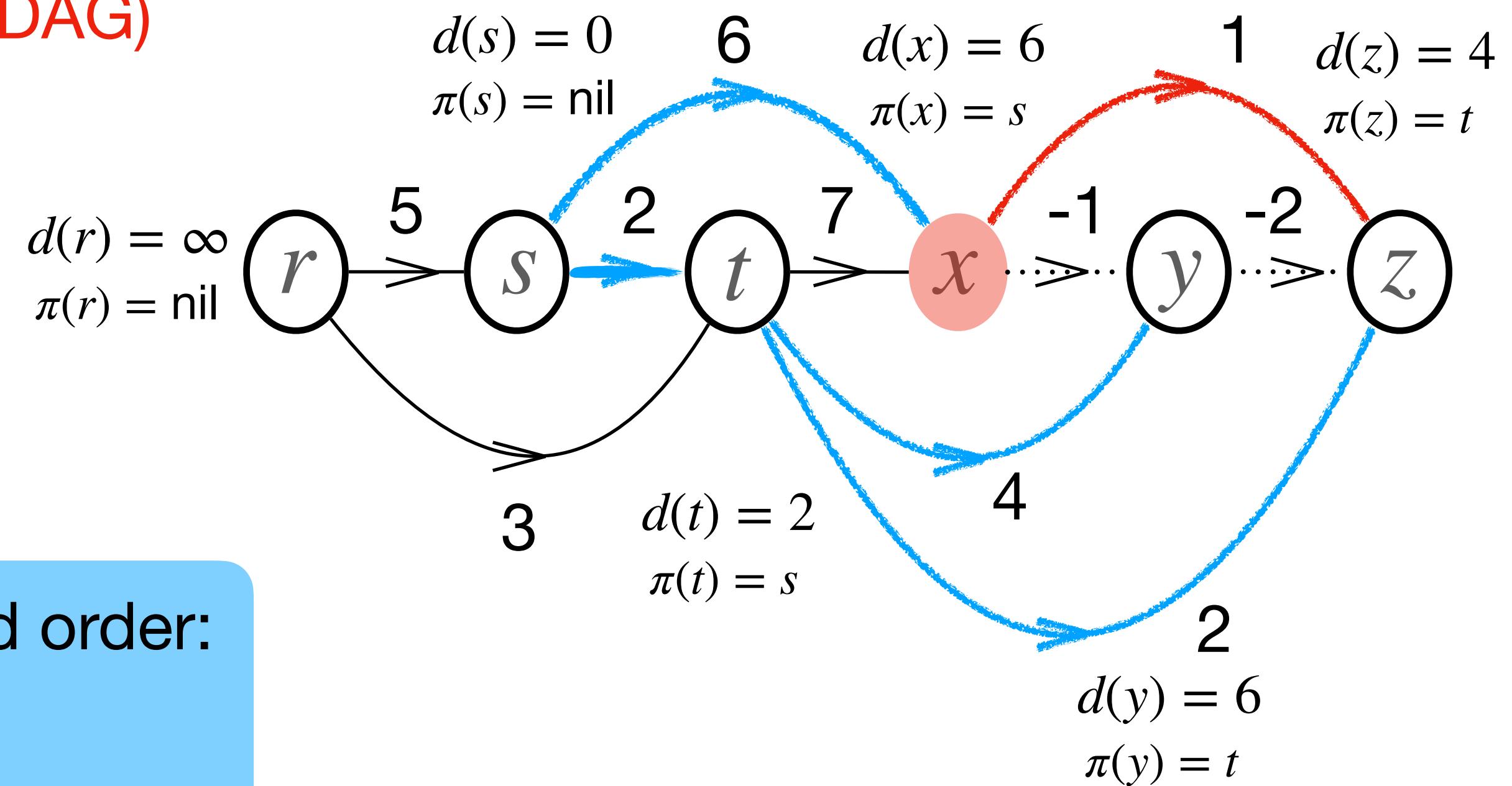
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

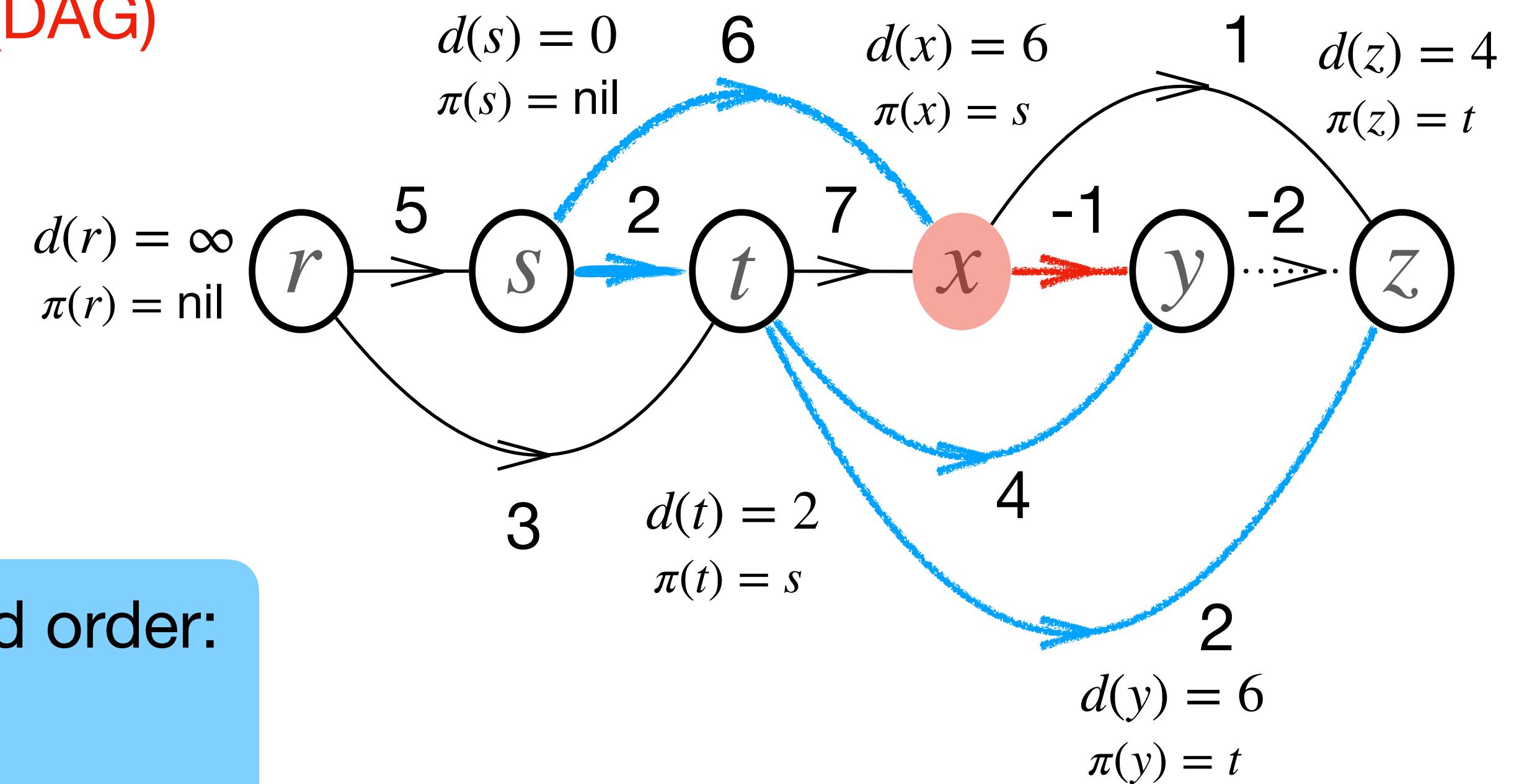
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

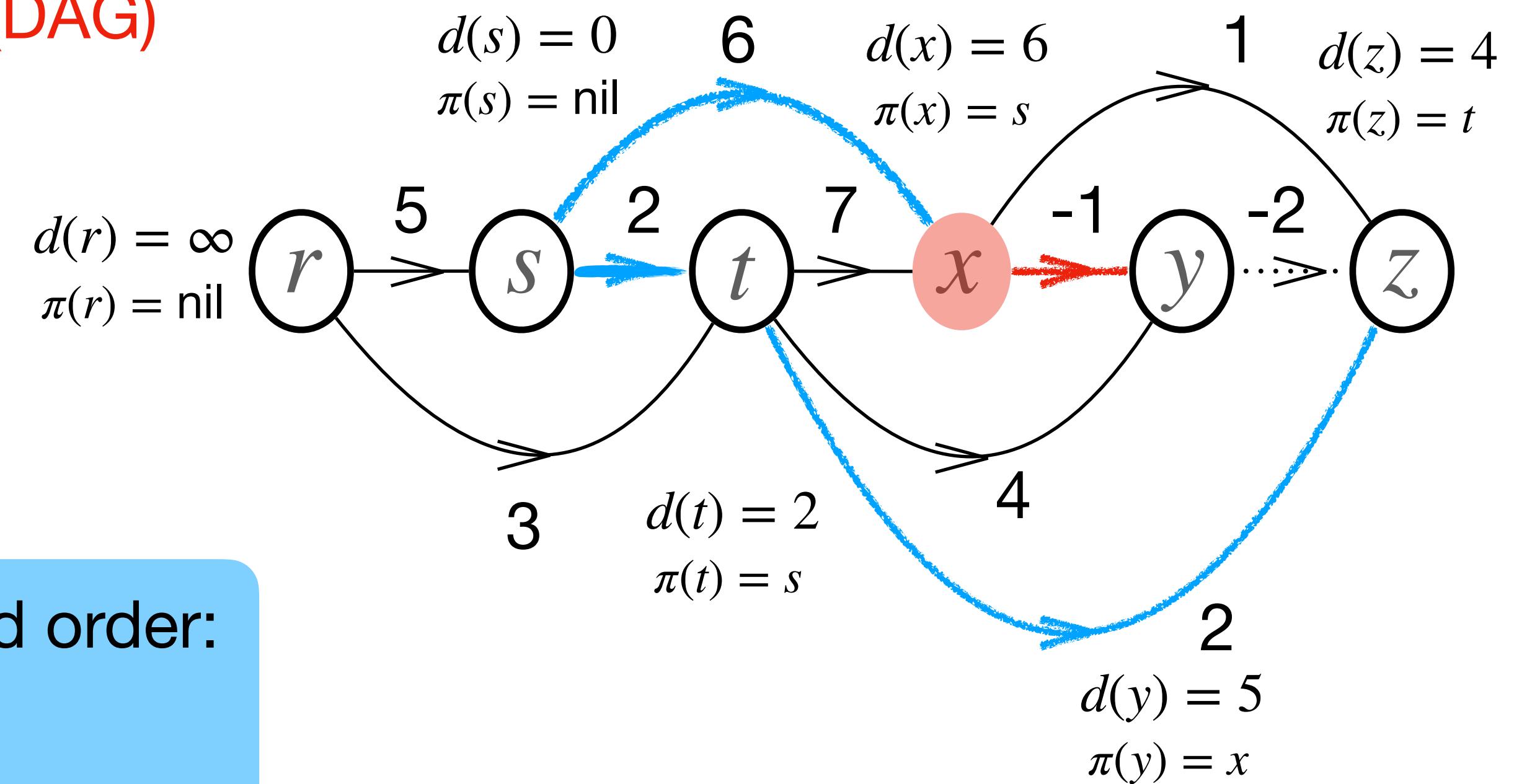
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

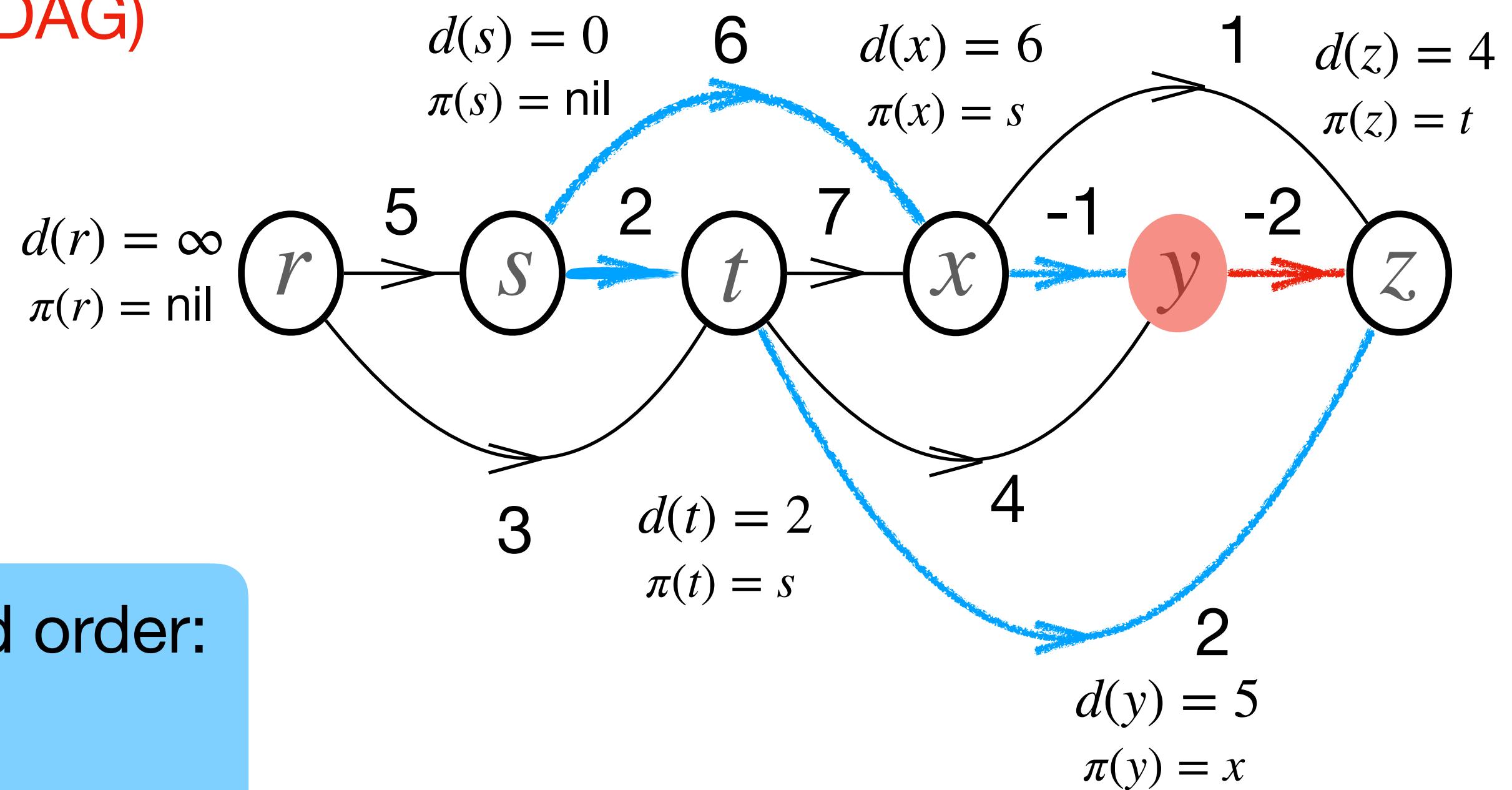
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

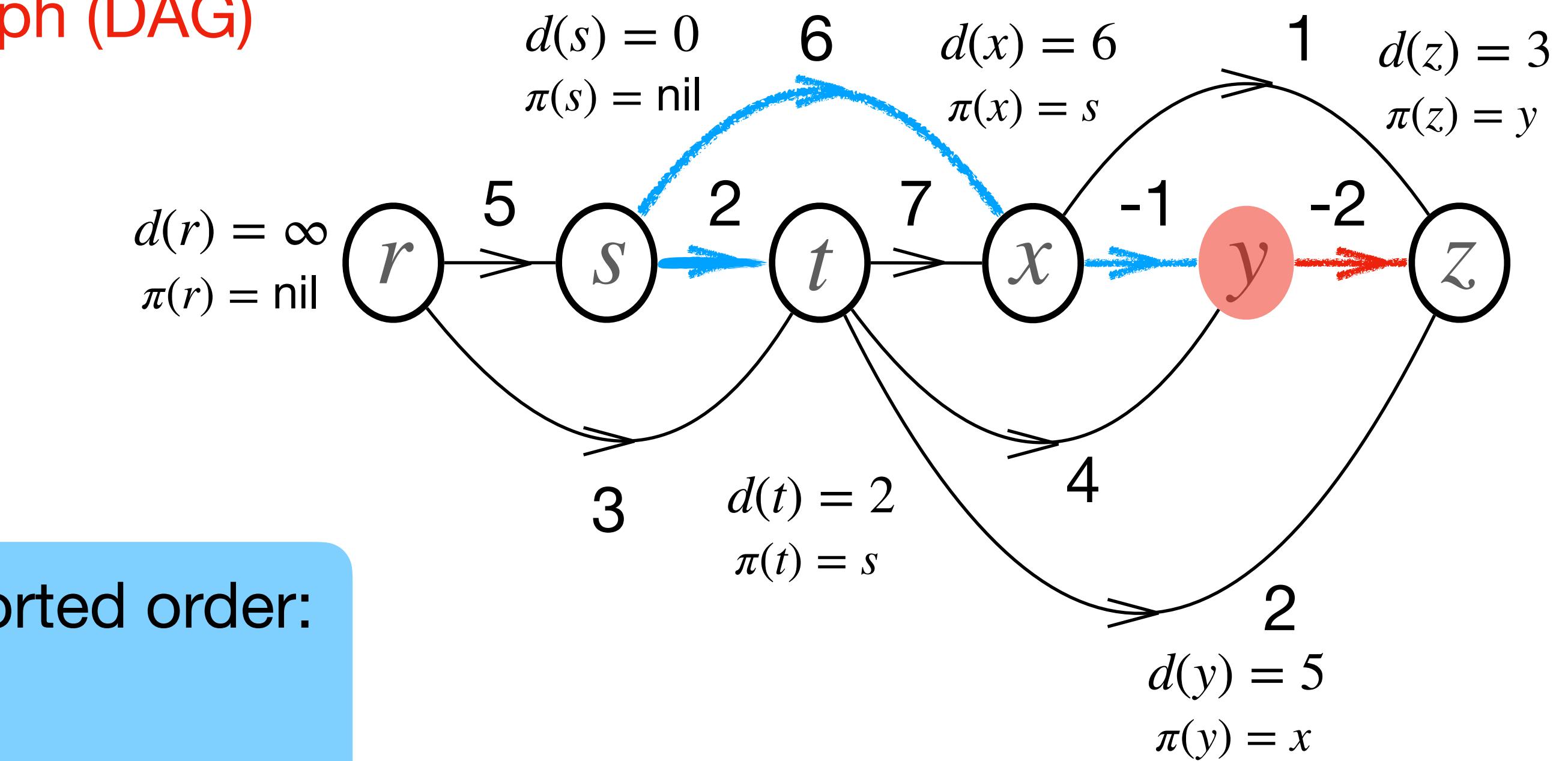
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

if $d(u) + w(u, v) < d(v)$:

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

$\pi(v) = u$



4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Relaxation

Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

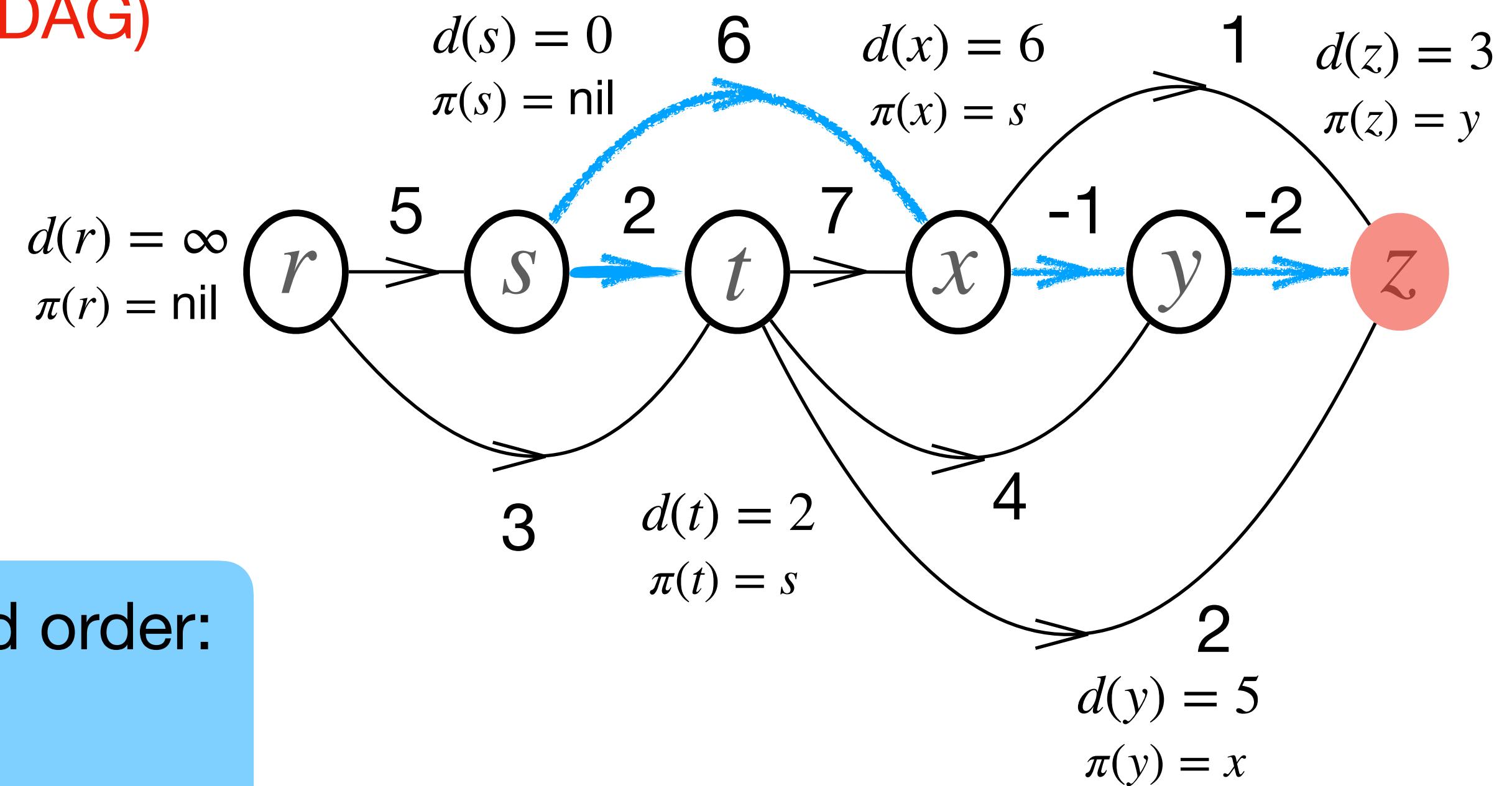
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

```
if  $d(u) + w(u, v) < d(v)$  :  
     $d(v) = d(u) + w(u, v)$   
     $\pi(v) = u$ 
```

Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).



Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

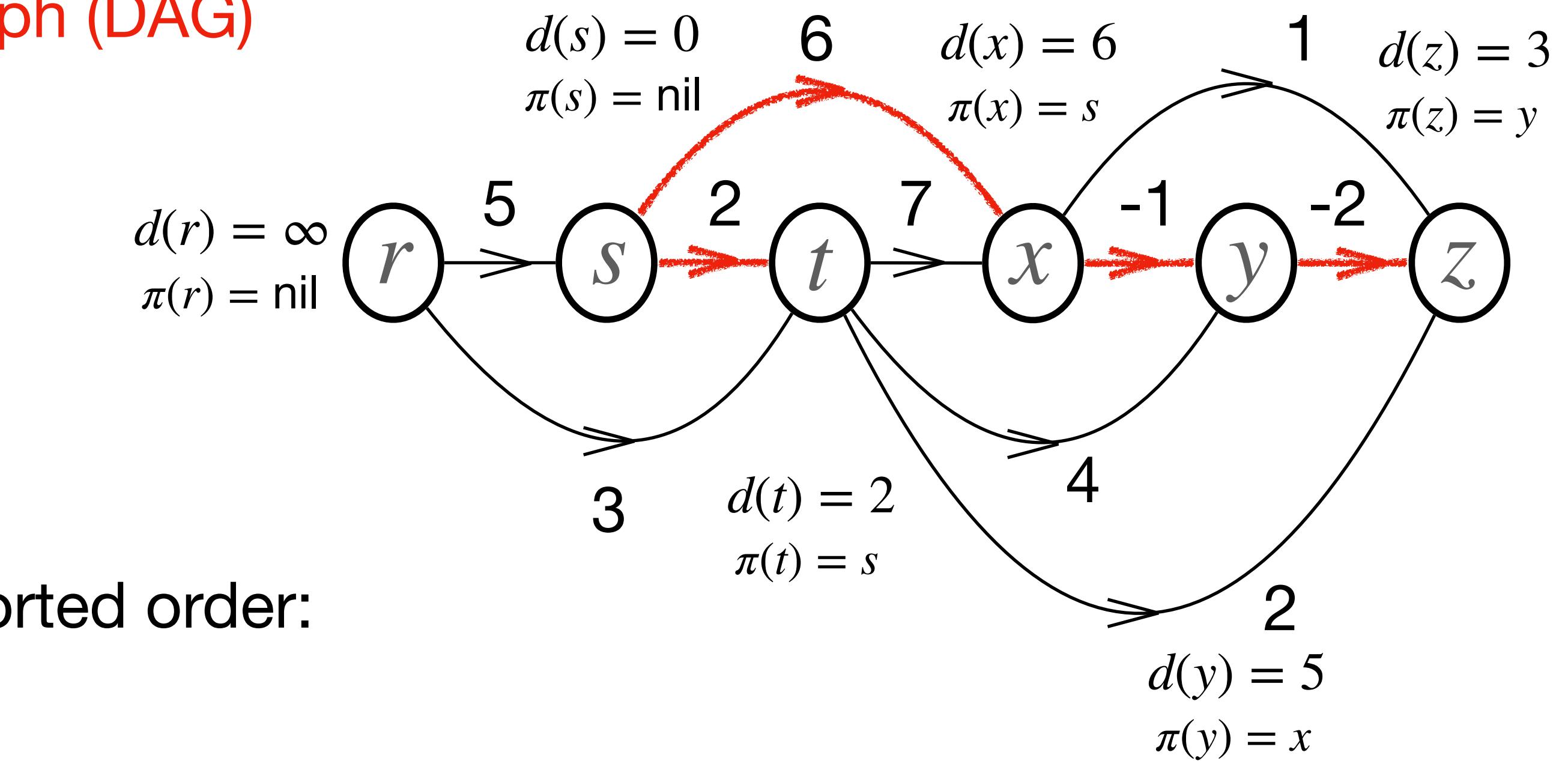
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

if $d(u) + w(u, v) < d(v)$:

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

$\pi(v) = u$



Relaxation

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Single-Source Shortest Paths

Shortest-Path Problem for **Directed Acyclic Graph (DAG)**

Algorithm:

1) Topologically sort the graph $G=(V,E)$

2) Initialization: $\forall v \in V, \pi(v) = \text{nil}$

$$d(s) = 0 \quad \forall v \neq s, d(v) = \infty$$

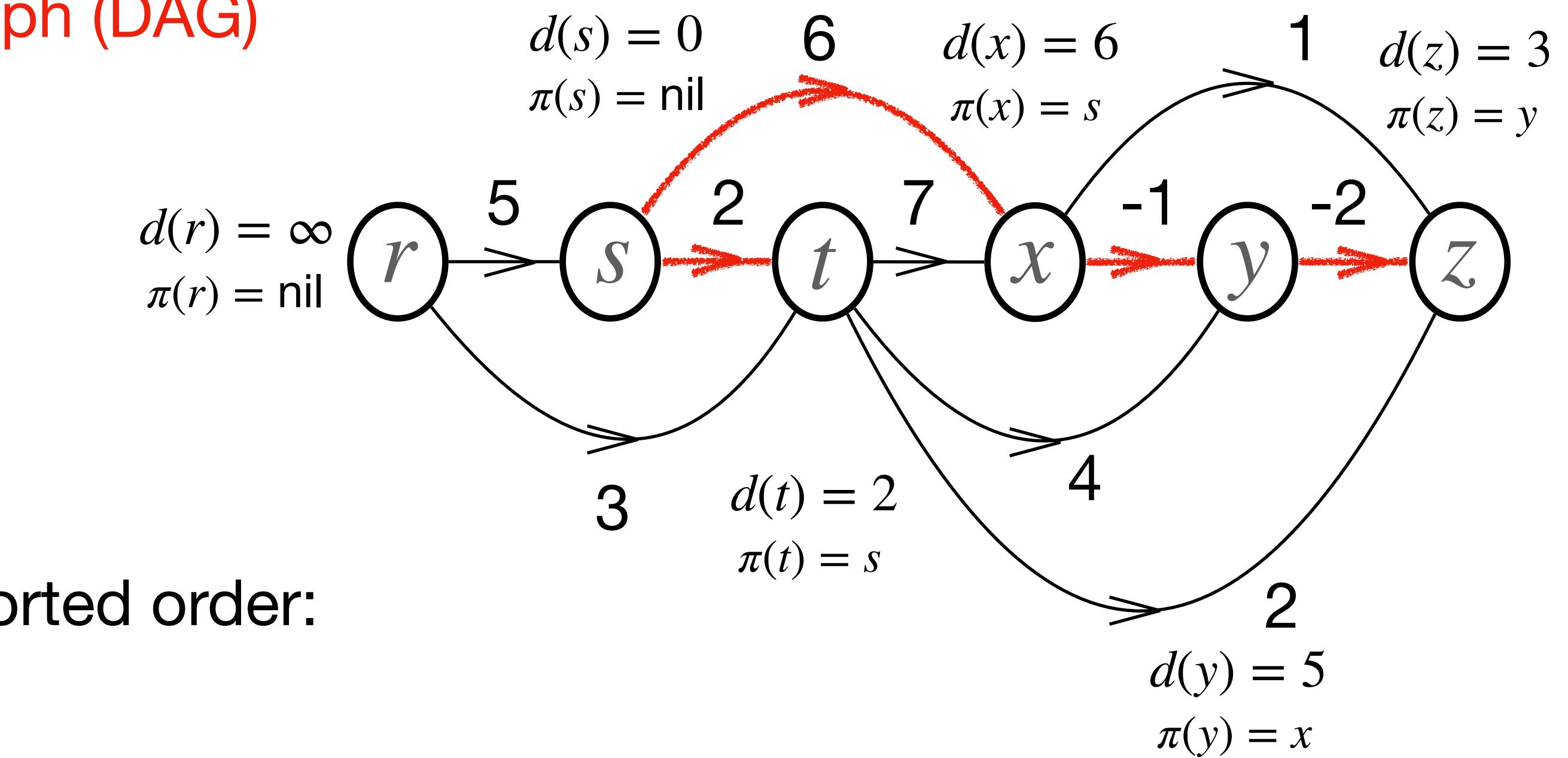
3) For each node u , taken in the topologically sorted order:

for every edge (u, v) leaving u :

if $d(u) + w(u, v) < d(v)$:

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

$$\pi(v) = u$$



Relaxation

Time Complexity:

$$O(V + E)$$

4) Return $d(v)$ and $\pi(v)$ for every node v
(which form a shortest-path tree).

Quiz question:

- I. Why is topological sort helpful for finding shortest paths in a DAG?