Algorithm Analysis and Data Structures

CSCI 7432 - Fall 2022

Single-Source Shortest Paths

Dr. Yao XU

Assistant Professor

Department of Computer Science

Georgia Southern University

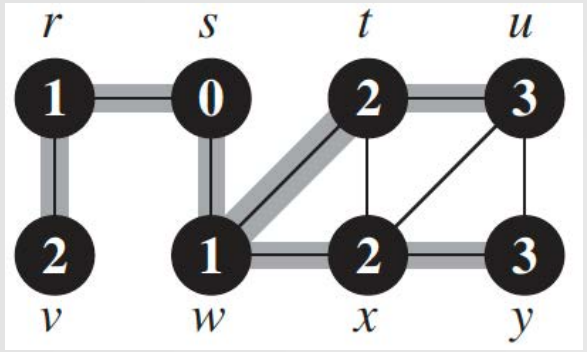
Email: yxu@georgiasouthern.edu

Table of Contents

The Single-Source Shortest Path problem (SSSP)

Shortest Path

- *Recall:* In **BFS**, $v.d$ = smallest # of edges from s to v
- **Example:**



- If the edges have weights, then a ***shortest path*** is a shortest weighted path = sum of weights of all edges on the path.
- Define $\delta(s, v)$ as the weight of a shortest path from s to v.
  - If there is no path from s to v, then we set $\delta(s, v) = \infty$.

Shortest Path Problems

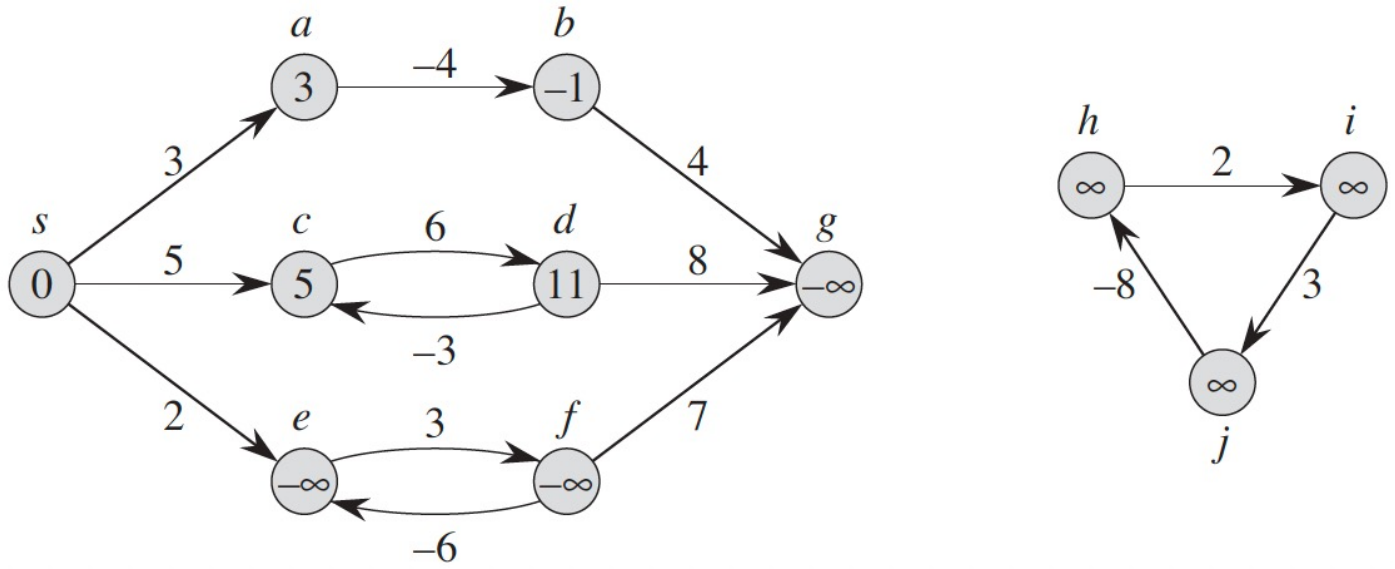
Input: A digraph $G = (V, E)$, with weight $w(u, v)$ on each edge $(u, v) \in E$.

Variants of shortest path problems:

- ***Single-source shortest-paths problem***:
  - **Additional input:** A source vertex $s \in V$.
  - **Output:** A shortest path from s to each vertex $v \in V$.
- ***Single-destination shortest-paths problem***:
  - **Additional input:** A destination vertex $t \in V$.
  - **Output:** A shortest path to t from each vertex $v \in V$.
- ***Single-pair shortest-path problem***:
  - **Additional input:** A starting vertex u and an ending vertex v.
  - **Output:** A shortest path from u to v.
- ***All-pairs shortest-paths problem***:
  - **Output:** A shortest path from u to v for every pair of vertices u and v.

Single-Source Shortest Paths (SSSP)

- Edge weights can be negative.
- $\delta(s, v) =$ the shortest path weight from s to v
- $\delta(s, v) = \infty$ if v is not reachable from s (no $s \rightsquigarrow v$ path).
- $\delta(s, v) = -\infty$ if there is a negative weight cycle on some $s \rightsquigarrow v$ path;
- **Example:** (δ values appear inside the vertices)



Shortest Path Properties

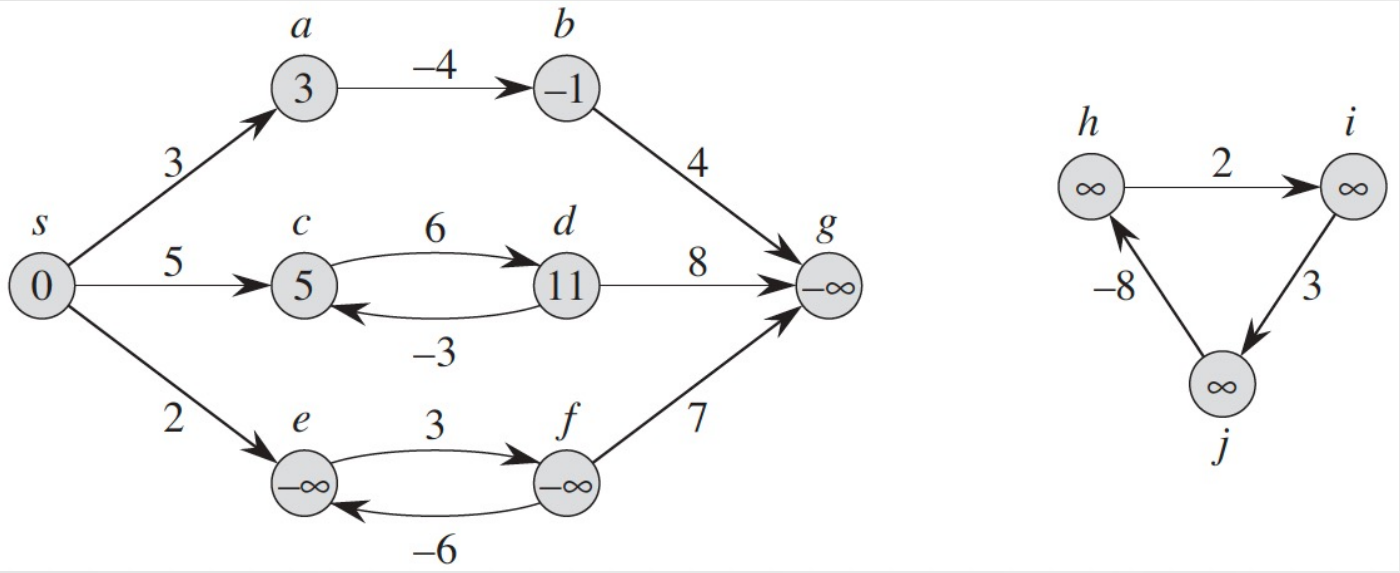
- **Optimal substructure** *(Lemma 24.1 in textbook)*:

  Let $p = \langle v_0, v_1, \cdots, v_{k-1}, v_k \rangle$ be a shortest path from v_0 to v_k. Then, the subpath $\langle v_i, v_{i+1}, \cdots, v_j \rangle$ of p with $0 \leq i < j \leq k$ must be a shortest path from v_i to v_j.

- **Triangle inequality** *(Lemma 24.10 in textbook)*:

  For all $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

- **Example:**



The Single-Source Shortest Path problem (SSSP)

Outline of SSSP Algorithms

Outline of SSSP Algorithms

Each vertex $v \in V$ will have two attributes:

- $v.d$: Initially $= \infty$, always $\geq \delta(s, v)$, and finally $= \delta(s, v)$
- $v.\pi$ = predecessor of v on a shortest path from s

All SSSP algorithms

1. Start with INITIALIZE-SINGLE-SOURCE:

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

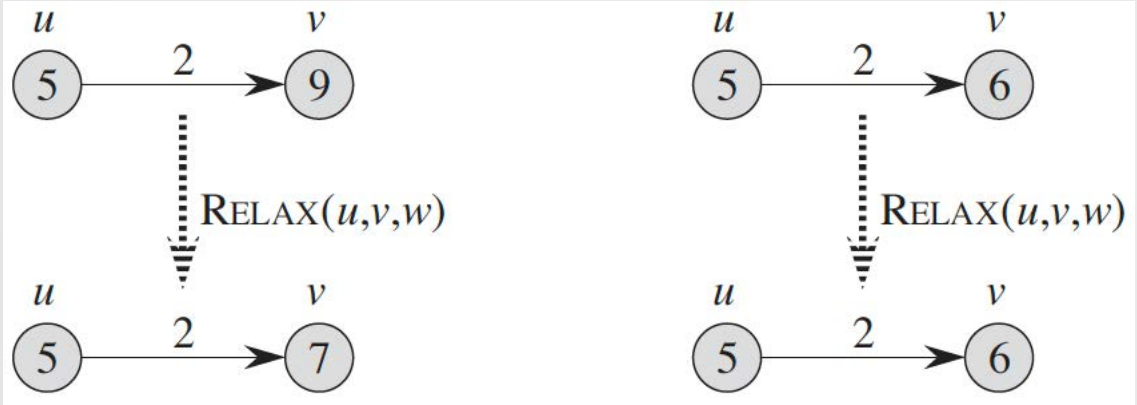
2. Then ***relax*** edges to update $v.d$ values.

Relaxing An Edge

- In a SSSP algorithm, the $v.d$ value is only updated by RELAX:

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

- **Examples:**

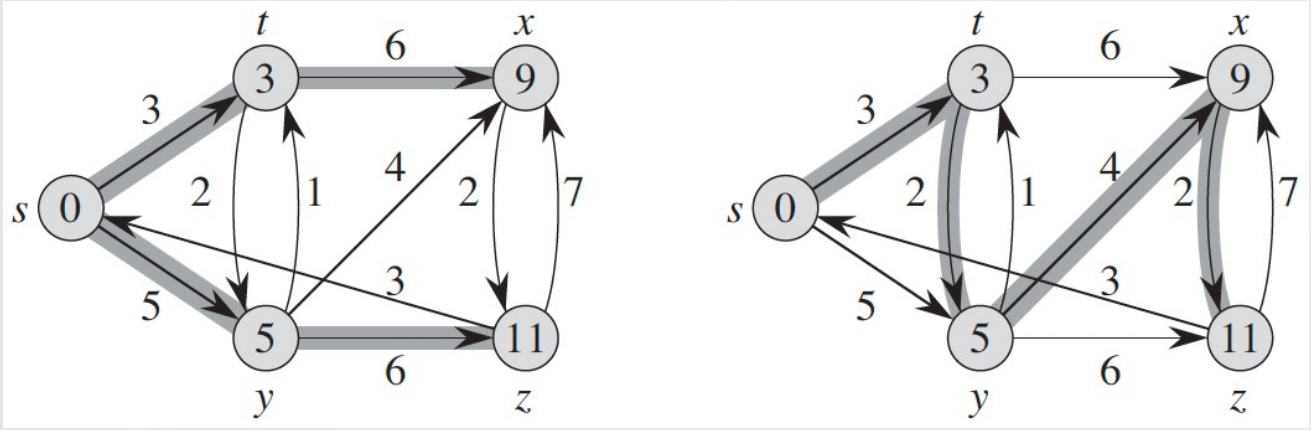


Shorted-Paths Tree of SSSP

- **Predecessor-subgraph property** *(Lemma 24.17 in textbook)*:

  The shortest paths from s to all $v \in V$ that are reachable from s form a ***shorted-paths tree*** rooted at s.

- **Example:**



Algorithms for SSSP

- **Dijkstra's algorithm**
  - Only works for graphs with NO negative-weight edges.
  - Uses greedy approach
- **Bellman-Ford algorithm**
  - Allows negative-weight edges.
  - **Output**:
    - If the graph contains a negative-weight cycle, return FALSE;
    - Otherwise, return TRUE and compute all the $\delta(s, v)$ values and the shorted-paths tree.

Dijkstra's Algorithm for SSSP

Dijkstra's Algorithm

Outline of Dijkstra's Algorithm

- For graphs with NO negative-weight edges.
- ***Dijkstra's algorithm*** is a **greedy** algorithm.
  - Greedy choice: Find the vertex that is the closest to s.
- Outline of **Dijkstra's algorithm**:

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  for i = 1 to n                    // n = |V|
4      find vertex u in V − S that is the closest to s (i'th closest to s in V)
5      S = S ∪ {u}
6      update d values using RELAX
```

- **Q:** How to implement lines 4 and 6 in the **for** loop?

Finding the *i*'th Closest Vertex

Claim: Let $v \in V$ be the i'th closest vertex to s and P be a shortest path from s to v. If S contains the $i-1$ closest vertices to s, then all intermediate vertices in P are in S.

Proof. (by contradiction)

- Suppose there is an intermediate vertex u in P and $u \notin S$.
- u is closer to s than v. (Due to no negative-weight edges)
- $\Rightarrow v$ is not the i'th closest vertex to s (since S already contains the $i-1$ closest vertices to s) – A contradiction

□

Claim $\Rightarrow$ If v is the i'th closest vertex to s, then we must have

$$\delta(s,v) = \delta(s,u) + w(u,v) \text{ for some } u \in S.$$

Dijkstra's Algorithm

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

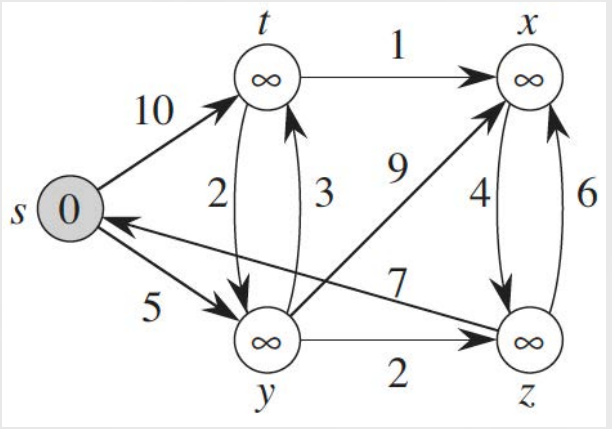
```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

At each iteration i of the **while** loop,

- S contains the $i-1$ closest vertices to s.
- $x.d = \delta(s, x)$ for every $x \in S$.
- $Q = V - S$ and vertex u with min d value in Q will be the i'th closest vertex to s
- Every edge (u, v) is then relaxed to update $v.d$ for each $v \in Adj[u]$.

Example:



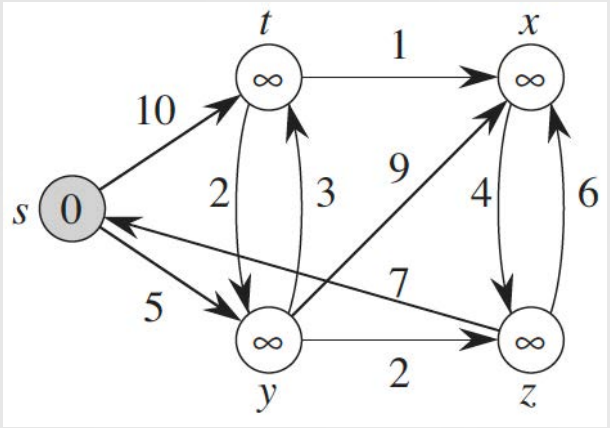
Dijkstra's Algorithm Example (1/3)

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

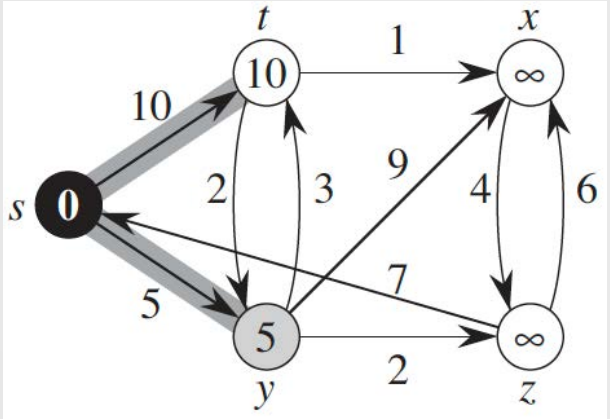
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Example:



$S = \emptyset, Q = \{s, t, x, y, z\}$

v	s	t	x	y	z
$v.\pi$	NIL	NIL	NIL	NIL	NIL
$v.d$	0	∞	∞	∞	∞



$S = \{s\}, Q = \{t, x, y, z\}$

v	s	t	x	y	z
$v.\pi$	NIL	s	NIL	s	NIL
$v.d$	0	10	∞	5	∞

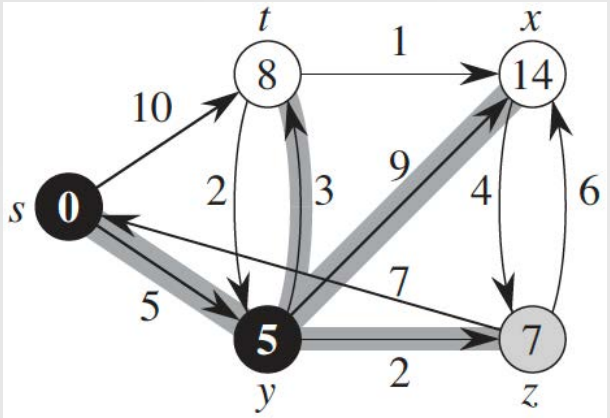
Dijkstra's Algorithm Example (2/3)

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

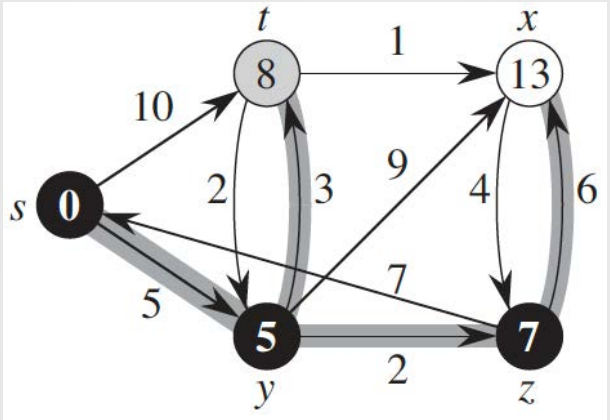
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Example (cont'd):



$S = \{s, y\}, Q = \{t, x, z\}$

v	s	t	x	y	z
$v.\pi$	NIL	y	y	s	y
$v.d$	0	8	14	5	7



$S = \{s, y, z\}, Q = \{t, x\}$

v	s	t	x	y	z
$v.\pi$	NIL	y	z	s	y
$v.d$	0	8	13	5	7

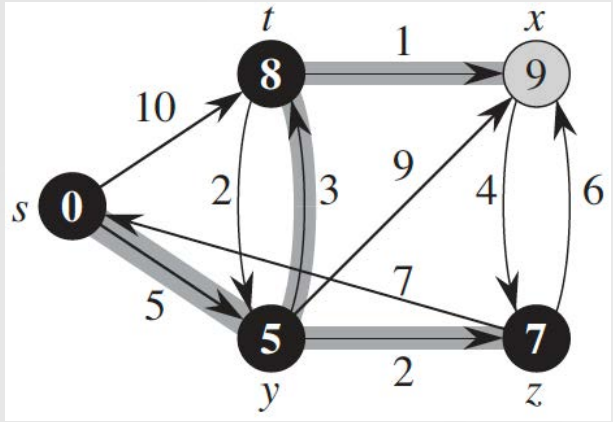
Dijkstra's Algorithm Example (3/3)

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

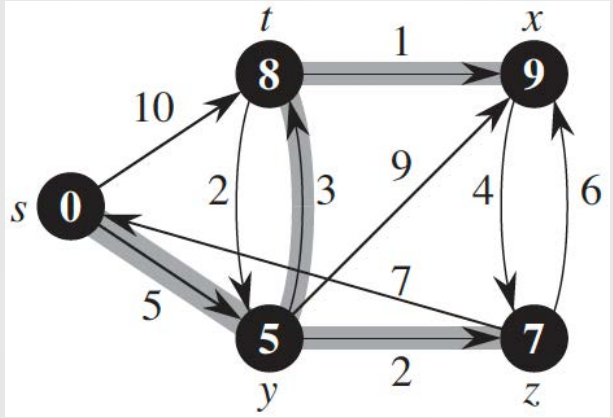
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Example (cont'd):



$S = \{s, y, z, t\}, Q = \{x\}$

v	s	t	x	y	z
$v.\pi$	NIL	y	z	s	y
$v.d$	0	8	9	5	7



$S = \{s, y, z, t, x\}, Q = \emptyset$

v	s	t	x	y	z
$v.\pi$	NIL	y	z	s	y
$v.d$	0	8	9	5	7

Running Time of Dijkstra's Algorithm

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
4      DECREASE-KEY(Q, v, v.d)
```

- Assume G is represented by adjacency lists.
- Q can be implemented as a min-heap,
  - Line 3: BUILD-MIN-HEAP takes $O(n)$ time
  - Line 5: EXTRACT-MIN takes $O(\log n)$ time
  - Add line 4 DECREASE-KEY to RELAX: takes $O(\log n)$ time.
- **Running time of** DIJKSTRA:

$$T(n) = O(n) + O(\sum_v (\log n + \deg(v) \cdot \log n))$$
$$= O((n+m)\log n)$$
$$= O(m \log n)$$

Dijkstra's Algorithm for SSSP

Correctness of Dijkstra's Algorithm

Correctness of Dijkstra's Algorithm (1/3)

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Theorem: For a digraph G with no negative-weight edges and a source vertex s, Algorithm DIJKSTRA terminates with $v.d = \delta(s, v)$ for every $v \in V$.

Proof. Use loop invariant.

LI: At the beginning of each iteration of the **while** loop, $v.d = \delta(s, v)$ for every $v \in S$.

- **Initialization:** $S = \emptyset$, LI is trivially true.
- **Maintenance:** If **LI** is true for some iteration and u is the vertex selected in line 5, then **LI** is still true at the end of this iteration. (See next slide)
- **Termination:** At the end of the **while** loop, $Q = \emptyset$, so $S = V$. LI implies that $v.d = \delta(s, v)$ for every $v \in V$.

Correctness of Dijkstra's Algorithm (2/3)

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

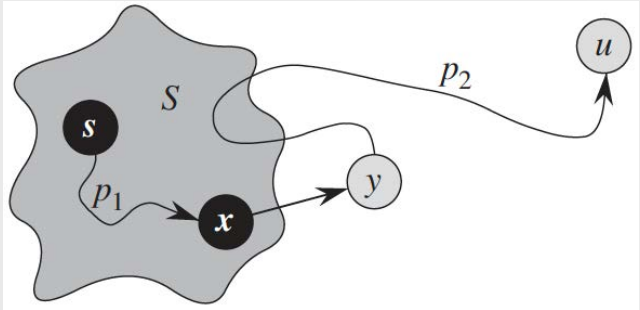
Proof. (cont'd)

LI: At the beginning of each iteration of the **while** loop, $v.d = \delta(s, v)$ for every $v \in S$.

Maintenance:

- Suppose that $v.d = \delta(s, v)$ for every $v \in S$ at the beginning of some iteration.
- Let u be the vertex selected in line 5 in this iteration.
- Need to show that $u.d = \delta(s, u)$ at the end of this iteration. (Only u's d value might be changed by the RELAX procedure.)

Consider a shortest path from s to u, through edge (x, y), where $x \in S$ and $y \in Q$.



Correctness of Dijkstra's Algorithm (3/3)

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

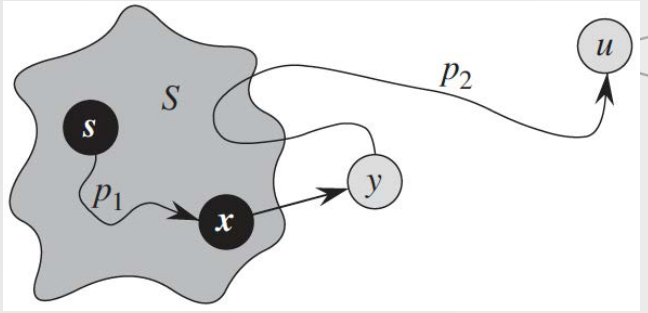
```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Proof. (cont'd)

Need to show: $u.d = \delta(s,u)$

Consider a shortest path from s to u, through (x,y).



(1) $x \in S \Rightarrow x.d = \delta(s,x)$

(2) $y.d \geq \delta(s,y)$ and $u.d \geq \delta(s,u)$

(3) (x,y) was relaxed when x was added to S:

$$y.d \leq x.d + w(x,y) = \delta(s,x) + w(x,y) = \delta(s,y)$$

(4) $\delta(s,y) \leq \delta(s,u)$ as y is on the shortest path from s to u

(5) (2)-(4) $\Rightarrow y.d = \delta(s,y) \leq \delta(s,u) \leq u.d$

(6) $u.d \leq y.d$ as $u = \text{EXTRACT-MIN}(Q)$

(7) (5)&(6) $\Rightarrow u.d = y.d = \delta(s,y) = \delta(s,u)$ □

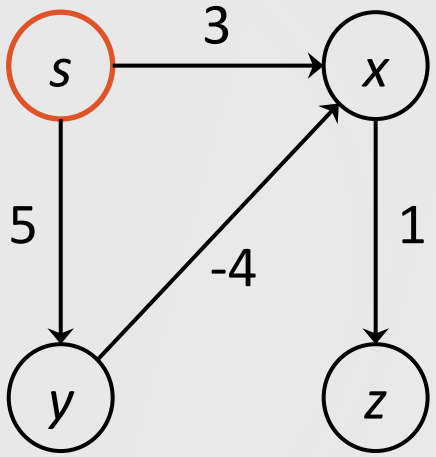
When There are Negative-Weight Edges

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

- The following example shows that Dijkstra's algorithm may fail when the input graph has negative-weight edges.
- **Example:** Source vertex is s.



Before the **while** loop:

$S = \emptyset, Q = \{s, x, y, z\}$

v	s	x	y	z
$v.\pi$				
$v.d$	0	∞	∞	∞

Bellman-Ford Algorithm for SSSP

Bellman-Ford Algorithm

- Allows negative-weight edges.
- Returns **FALSE** if the graph contains a negative-weight cycle;
- Otherwise, returns **TRUE** and computes all the $\delta(s, v)$ values and the shorted-paths tree.

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

- The nested **for** loops (in lines 2-4) relax all edges $n - 1$ times.
- **Running time** of BELLMAN-FORD:

$$T(n) = O(n) + O(nm) + O(m)$$

$$\Rightarrow T(n) \in O(nm)$$

Bellman-Ford Algorithm Example 1 (1/2)

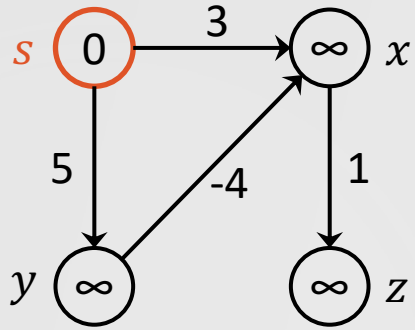
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

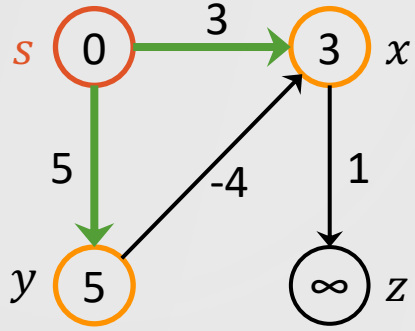
Example 1: G has no negative cycle. Source is s.

Edge order: $(x, z), (y, x), (s, x), (s, y)$



v	s	x	y	z
$v.\pi$	NIL	NIL	NIL	NIL
$v.d$	0	∞	∞	∞

1st iteration:



v	s	x	y	z
$v.\pi$	NIL	s	s	NIL
$v.d$	0	3	5	∞

Bellman-Ford Algorithm Example 1 (2/2)

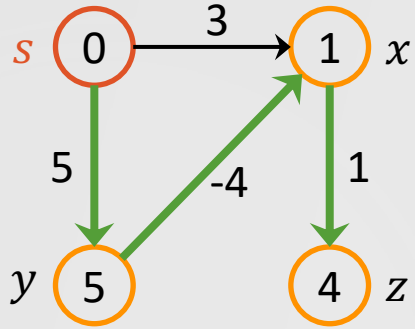
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

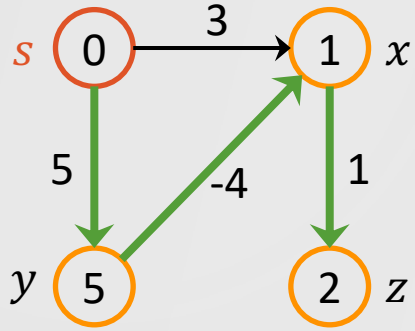
Example 1: Edge order: $(x, z), (y, x), (s, x), (s, y)$

2nd iteration:



v	s	x	y	z
$v.\pi$	NIL	y	s	x
$v.d$	0	1	5	4

3rd iteration:



v	s	x	y	z
$v.\pi$	NIL	y	s	x
$v.d$	0	1	5	2

Correctness of Bellman-Ford Algorithm (1/2)

Lemma 1: Suppose that G has no negative-weight cycle reachable from source s. Then after $n-1$ iterations, $v.d = \delta(s,v)$ for all $v \in V$.

Proof. Consider any vertex $v \in V$ and let path $p = \langle v_0, v_1, \cdots, v_{k-1}, v_k \rangle$ be the shortest path from s to v, where $v_0 = s$ and $v_k = v$.

We prove by induction on i that after iteration $i \geq 0$, $v_i.d = \delta(s, v_i)$.

- **Base case**: $i = 0$ is trivial.
- **Inductive step:** IH: $v_i.d = \delta(s, v_i)$ after iteration i, for $0 \leq i \leq k-1$.

  Need to show: $v_{i+1}.d = \delta(s, v_{i+1})$ after iteration $i+1$.
  - When (v_i, v_{i+1}) is relaxed during iteration $i+1$, we have

    $$v_{i+1}.d \leq v_i.d + w(v_i, v_{i+1})$$
    $$= \delta(s, v_i) + w(v_i, v_{i+1}) = \delta(s, v_{i+1})$$
  - $v_{i+1}.d \geq \delta(s, v_{i+1}) \Rightarrow v_{i+1}.d = \delta(s, v_{i+1})$. □

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

Correctness of Bellman-Ford Algorithm (2/2)

Lemma 2: If G has a negative-weight cycle reachable from source s, then the algorithm returns FALSE.

Proof. Suppose there is a negative-weight cycle $c = \langle v_0, v_1, \cdots, v_k \rangle$, with $v_0 = v_k$. The weight of c is $w(c) = \sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$.

- Suppose (for contradiction) that the algorithm returns TRUE. Then,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i) \text{ for all } i = 1, 2, \cdots, k.$$

- Sum around cycle c:

$$\begin{aligned} \sum_{i=1}^{k} v_i.d &\leq \sum_{i=1}^{k} \left(v_{i-1}.d + w(v_{i-1}, v_i)\right) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i), \end{aligned}$$

- but $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$ as $v_0 = v_k$
- This implies: $w(c) \geq 0$ – contradicts $w(c) < 0$. □

Bellman-Ford Algorithm Example 2 (1/2)

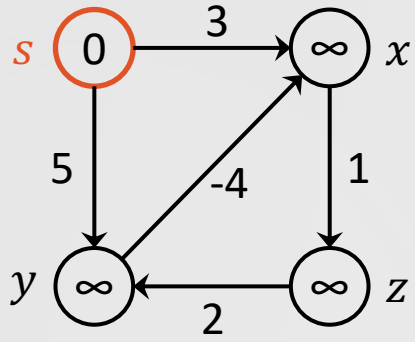
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

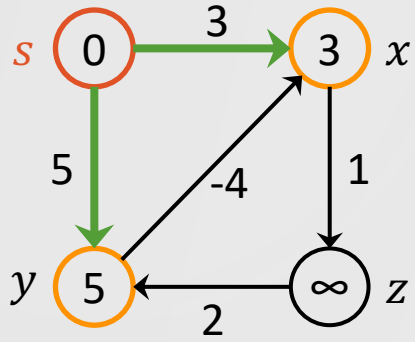
Example 2: G has negative cycle. Source is s.

Edge order: $(z, y), (x, z), (y, x), (s, x), (s, y)$



v	s	x	y	z
$v.\pi$	NIL	NIL	NIL	NIL
$v.d$	0	∞	∞	∞

1st iteration:



v	s	x	y	z
$v.\pi$	NIL	s	s	NIL
$v.d$	0	3	5	∞

Bellman-Ford Algorithm Example 2 (2/2)

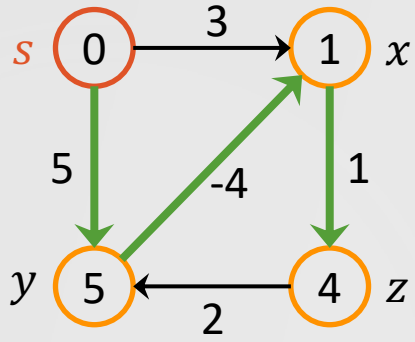
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

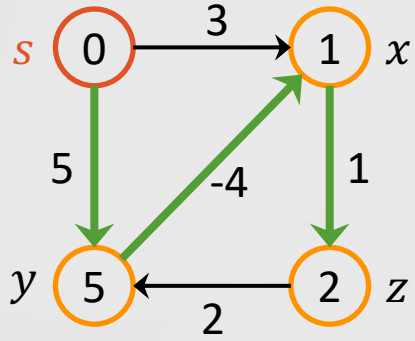
Example 2: Edge order: $(z, y), (x, z), (y, x), (s, x), (s, y)$

2nd iteration:



v	s	x	y	z
$v.\pi$	NIL	y	s	x
$v.d$	0	1	5	4

3rd iteration:



v	s	x	y	z
$v.\pi$	NIL	y	s	x
$v.d$	0	1	5	2

Thank you!
Questions?