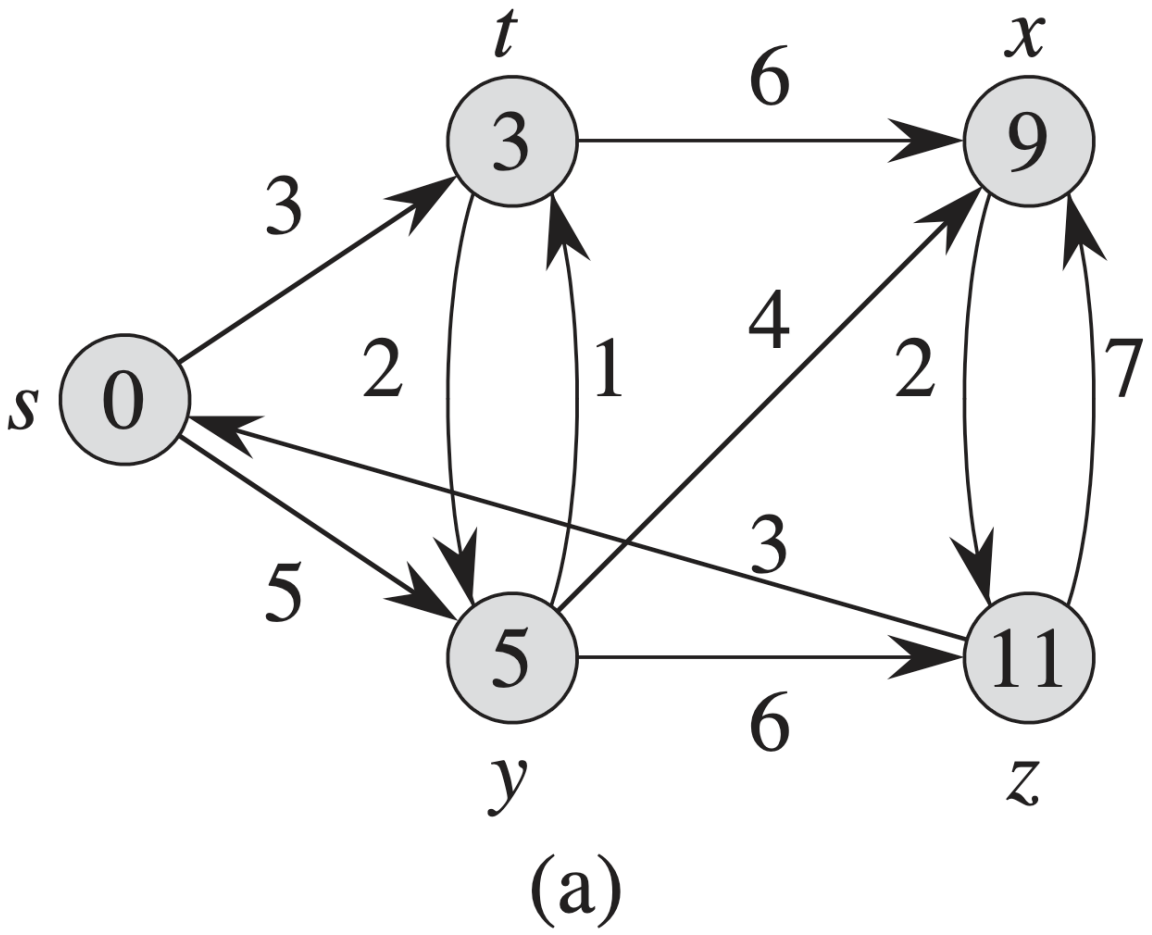
CS 590: Algorithms

Lecture 12: Shortest Paths

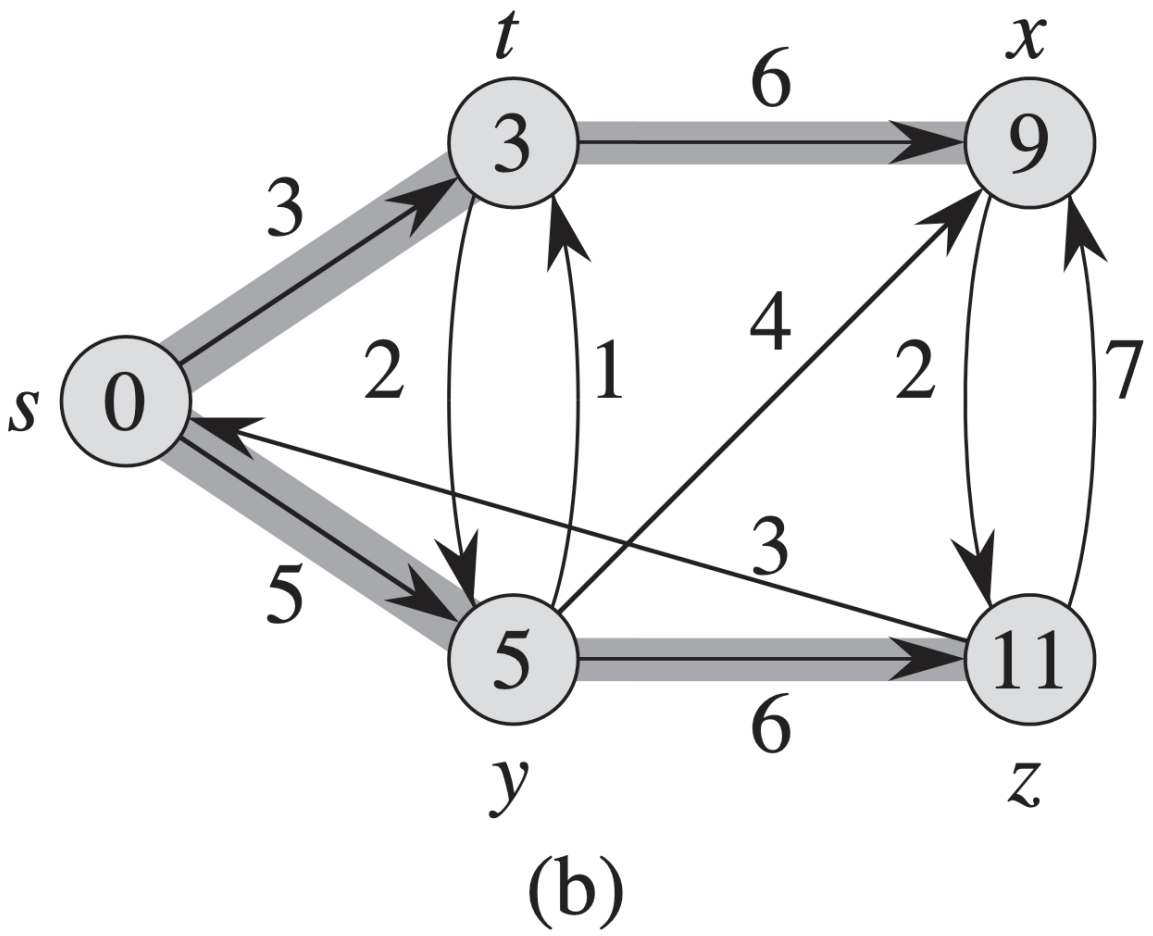
Single-Source Shortest Paths

- **Input:**
  - Directed graph $G = (V, E)$
  - Weight $w: E \to \mathbb{R}$
- **Weight of path:**
  - $p = \langle v_0, \cdots, v_k \rangle$ is $\sum_{i=1}^{k} w(v_{i-1}, v_i)$.
- **Shortest-path weight *u* to *v*:**
  - $\delta(u, v) = \begin{cases} \min\{w(p): u \to v\} & \text{if there exists a path} \\ \infty & \text{otherwise} \end{cases}$
  - The shortest path u to v is any path p such that $w(p) = \delta(u, v)$.

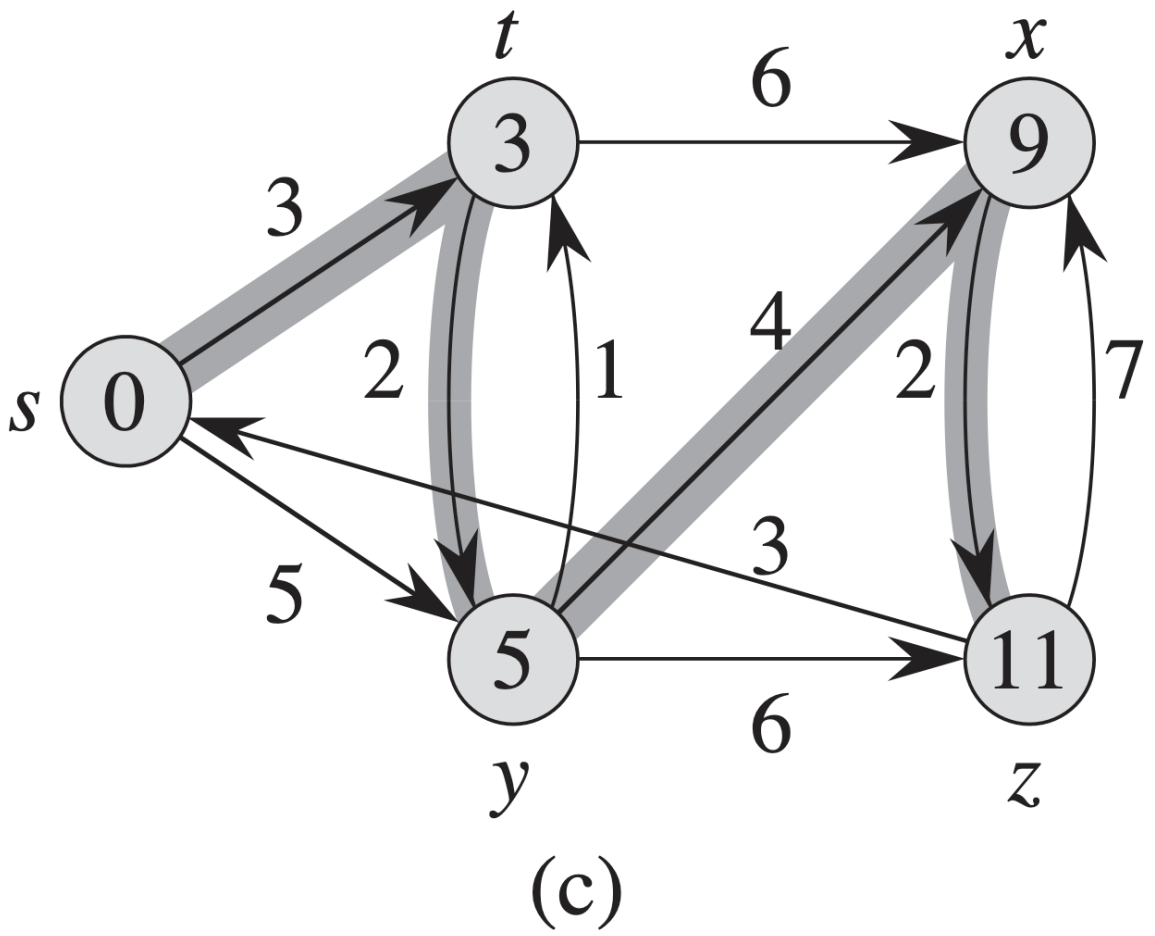
Single-Source Shortest Paths



Single-Source Shortest Paths



(b)



(c)

Single-Source Shortest Paths

- The shortest path observation:
  - The shortest path might be a **local solution**.
  - The shortest paths from one vertex to all other vertices are organized as a **tree**.
  - We can think of weights as a representation of any accumulative measurement we want to minimize.
  - It is a generalization of BFS to weighted graphs.

Single-Source Shortest Paths

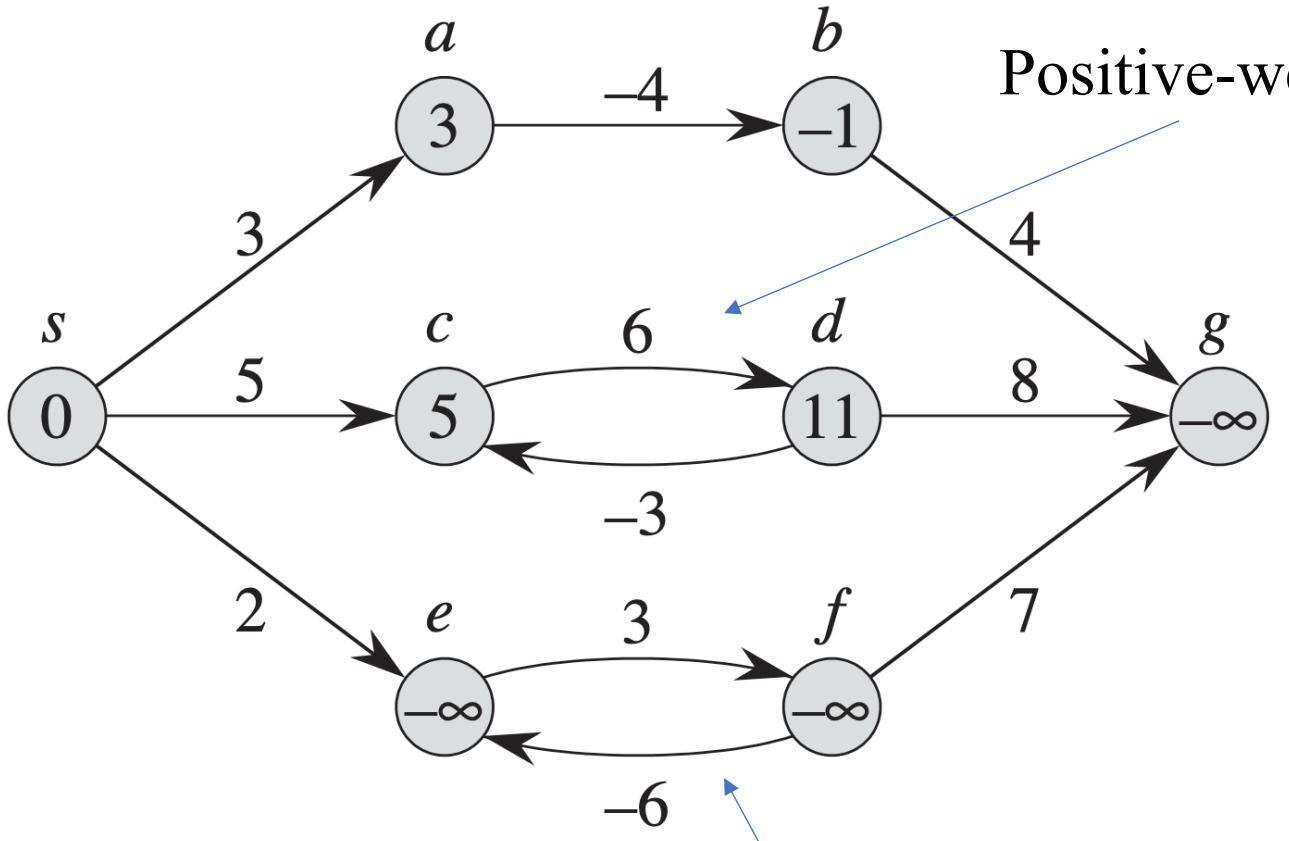
Variants:

- **Single-source**: Find the shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$.
- **Single-destination**: Find the shortest paths to a given destination.
- **Single-pair**: Find the shortest path from u to v.
- **All-pairs**: Find the shortest path from u to v for all $u, v \in V$.

Negative-weight edges:

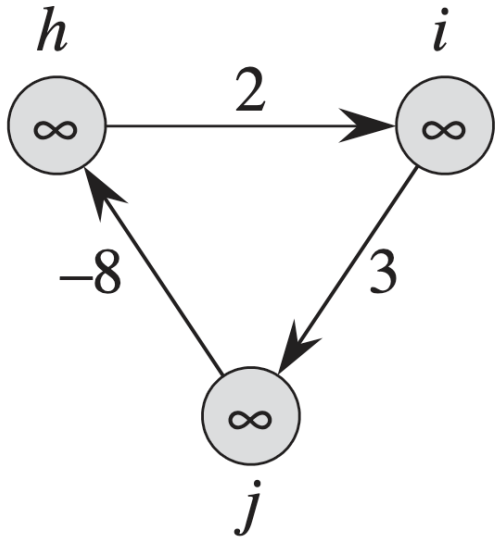
- If a graph has a negative-weight cycle, we can just keep going around it and get $w(s, v) = -\infty$ for all v on the cycle.
- If a negative-weight cycle is not reachable, it is fine to have it.
- Some algorithms restrict negative-weight edges $\Rightarrow \forall\, w_i \geq 0$.

Single-Source Shortest Paths



Positive-weight cycle

Negative-weight cycle



h, *i*, and *j* are not reachable from s!

Single-Source Shortest Paths

Optimal substructure:

- Any sub-path of the shortest path is the shortest path.
  - Suppose there is a path p: $u \to x \to y \to v$.
  - Let the shortest path be $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.
  - If there is a new path found $x \to y, p'_{xy}$ whose weight is less than $p_{xy}: w(p'_{xy}) < w(p_{xy})$, then the new shortest path is

$$\delta(u, v) = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$$
$$< w(p_{ux}) + w(p_{xy}) + w(p_{yv})$$

Single-Source Shortest Paths

Optimal substructure:

- The shortest path cannot contain cycles.
  - Negative-weight cycles are ruled out.
  - Positive-weight cycles will be omitted in our solution.
  - Zero-weight cycles do not have reasons to be used.
    - We assume that they are not a part of solutions.

Single-Source Shortest Paths

The output of the single-source δ algorithm:

- For each vertex $v \in V$:
- The shortest-path estimator: $v.d = \delta(s, v)$
  - Have all $v.d = \infty$ initially.
  - Reduce it as the algorithm progresses but always maintain $v.d \geq \delta(s, v)$.
- The shortest-path tree, $v.\pi$: the processor of v on the shortest path from s.
  - $v.\pi = NIL$, if there is no predecessor.
  - π includes a tree.

Single-Source Shortest Paths

Initialization:

- For all δ algorithms, we will start will INIT-SINGLE-SOURCE(G,s) algorithm.

Algorithm (INIT-SINGLE-SOURCE(G, s)):

```
1    Foreach (v ∈ G.V) do
2        v.d = ∞
3        v.π = NIL
4    s.d = 0
```

Single-Source Shortest Paths

Relaxing an edge (u, v):

- Use RELAX(u, v, w) algorithm to improve the shortest-path estimate for v by going through u and taking (u,v).

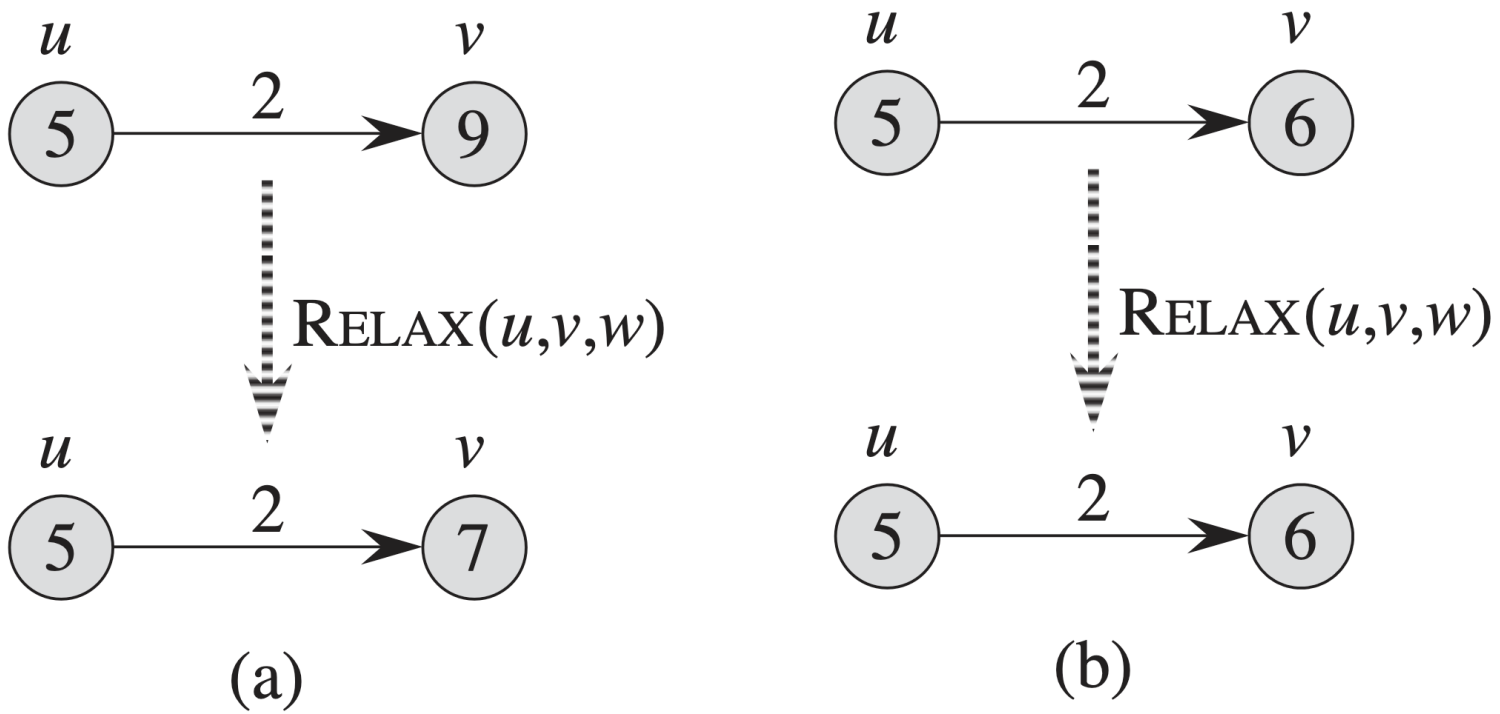
Algorithm (RELAX(u, v, w)):

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

Single-Source Shortest Paths

Algorithm (RELAX(u, v, w)):

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



Single-Source Shortest Paths

δ properties:

- INIT-SINGLE-SOURCE will be called once.
- RELAX may be called zero or more times.

Triangle inequality:

- $\forall\ (u, v) \in E, \delta(s, v) \leq \delta(s, u) + w(u, v)$.
  - $\delta(s, v)$ for $s \to v$ is always $\leq\ \forall\ p \in s \to v$.
  - If $u \to v$ is in $s \to v$, then $\delta(s, u)$ will be used in $\delta(s, v)$.

Single-Source Shortest Paths

Upper-bound property:

- $v.d \geq \delta(s, v)\ \forall\ v$
  - Once we have $v.d = \delta(s, v)$, it will never change.
    - Suppose we have $v.d = \delta(s, v)$.
    - Let u be the vertex that changes $v.d$:
      - $v.d < \delta(s, v) \leq \delta(s, u) + w(u, v) \leq u.d + w(u, v)$
  - Once $v.d$ reaches $\delta(s, v)$, it never goes up or down because of relaxations → it only lowers the shortest-path estimates.

Single-Source Shortest Paths

No-path property:

- If $\delta(s, v) = \infty$, then we always have $v.d = \infty$.
  - $v.d \geq \delta(s, v) = \infty \Rightarrow v.d = \infty$.

Convergence property:

- If $s \to u \to v$ is the shortest path, $u.d = \delta(s, u)$ and RELAX will converge $v.d = \delta(s, v)$ afterwards.
  - After the relaxation,

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

  - Since $v.d \geq \delta(s, v)$, we must have $v.d = \delta(s, v)$.

Single-Source Shortest Paths

Path relaxation property:

- Let $p = \langle v_0, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k.
- If we relax in the order of $(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$ even intermixed with other relaxations, then $v_k.d = \delta(s, v_k)$.
  - Consider as inductive fashion: $v_i.d = \delta(s, v_i)$ after $\delta(v_{i-1}, v_i)$.
  - Basis: if $i = 0$,
    - $v_0.d = 0$
    - $\delta(s, v_0) = \delta(s, s)$
  - Inductive steps:
    - Assume that $v_{i-1}.d = \delta(s, v_{i-1})$.
    - If we relax (v_{i-1}, v_i), then $v_i.d = \delta(s, v_i)$ afterwards and $v_i.d$ will never change ⇒ the convergence property.

Single-Source Shortest Paths

Bellman-Ford Algorithm:

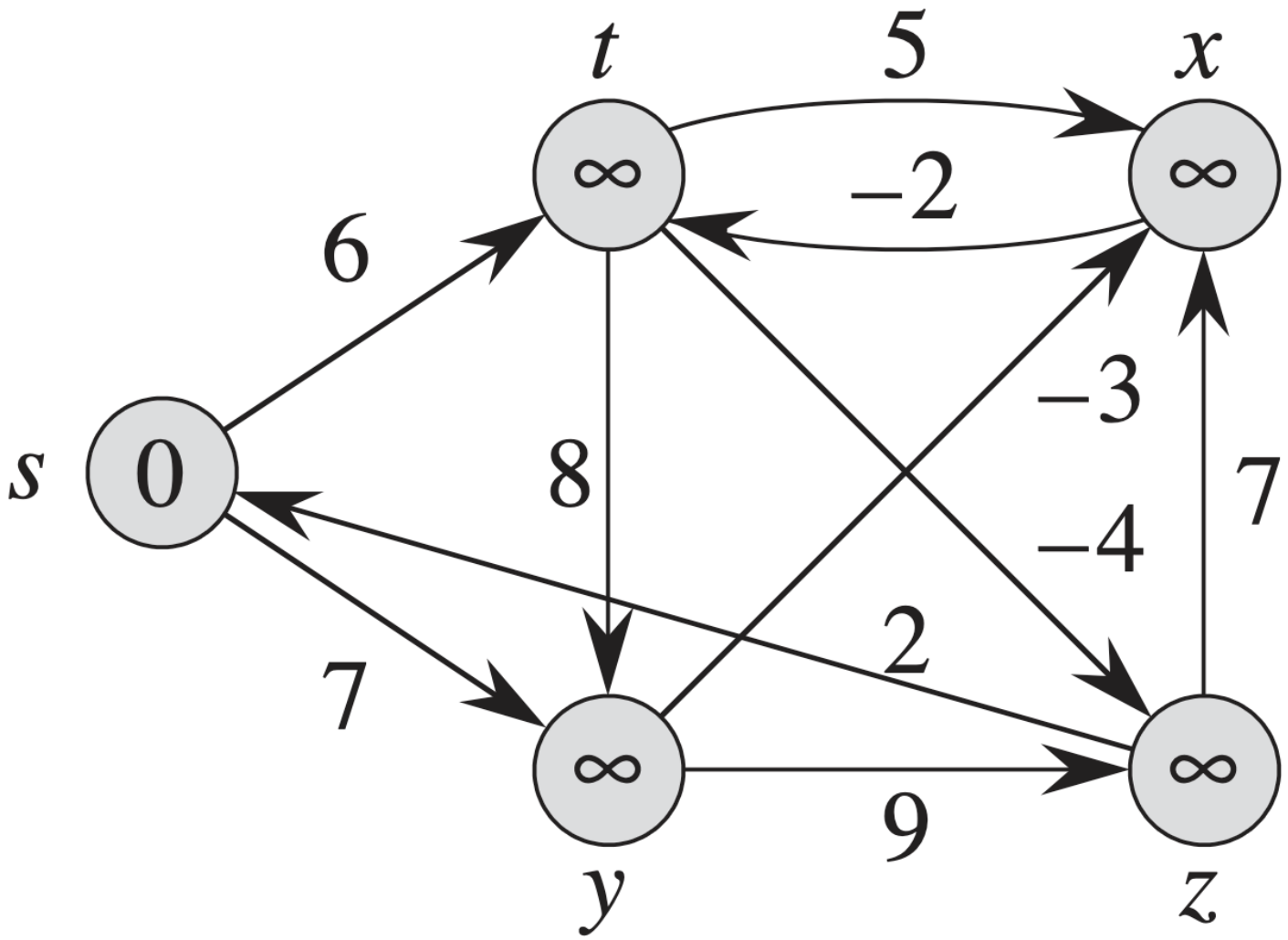
- Allows for negative-weight edges.
- Computes $v.d$ and $v.\pi$ $\forall\ v \in V$.
- Returns TRUE if no negative-weight cycles are reachable from s, FALSE otherwise.

Algorithm (BELLMAN-FORD(G, w, s)):

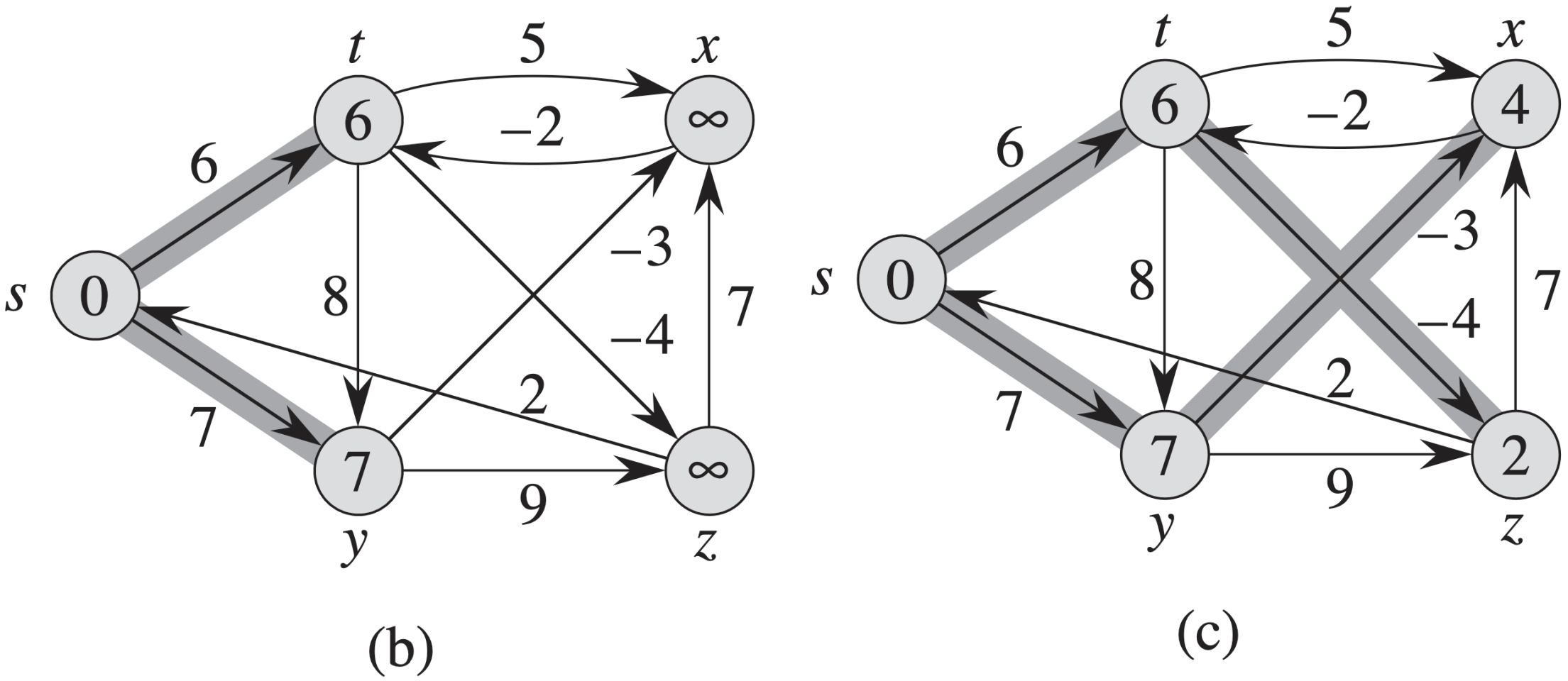
```
1  INIT-SINGLE-SOURCE(G, s)
2  for (1 ≤ i ≤ |G.V| − 1) do
3     foreach ((u, v) ∈ G.E) do
4        RELAX(u, v, w)
5   foreach ((u, v) ∈ G.E) do
6      if (v.d > u.d + w(u, v)) then
7         return FALSE
8   return TRUE
```

- For-loops relax all edges $|V| - 1$ times
- Time: $\Theta(VE)$

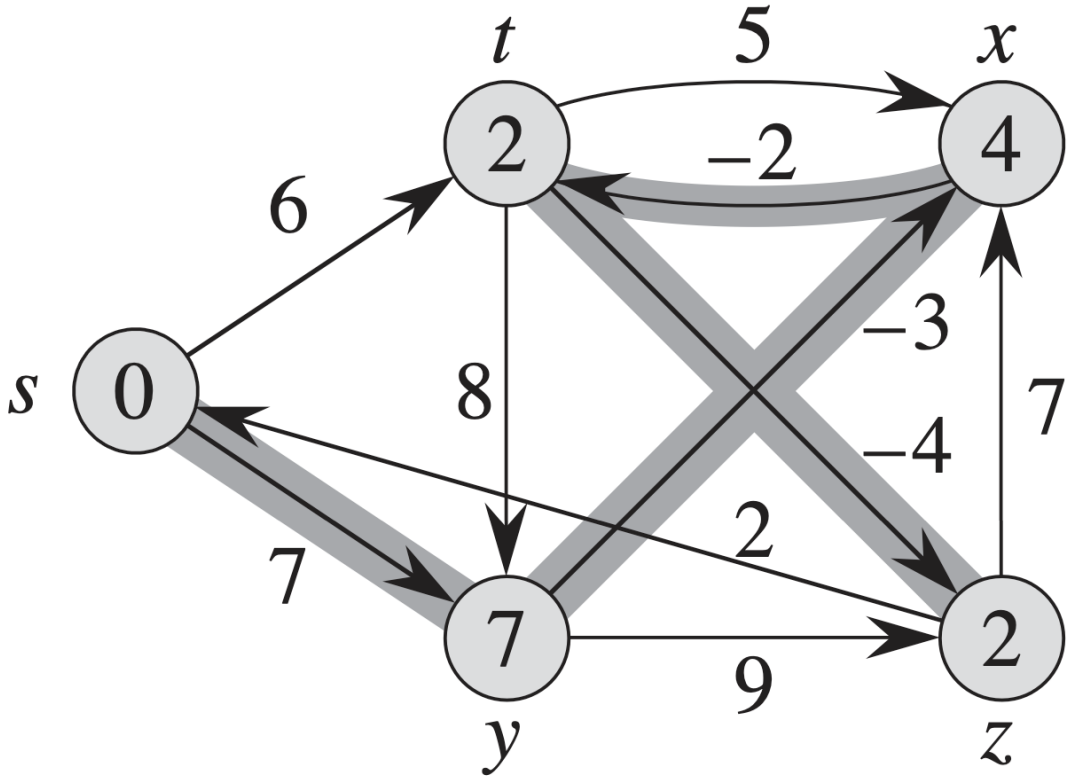
Single-Source Shortest Paths



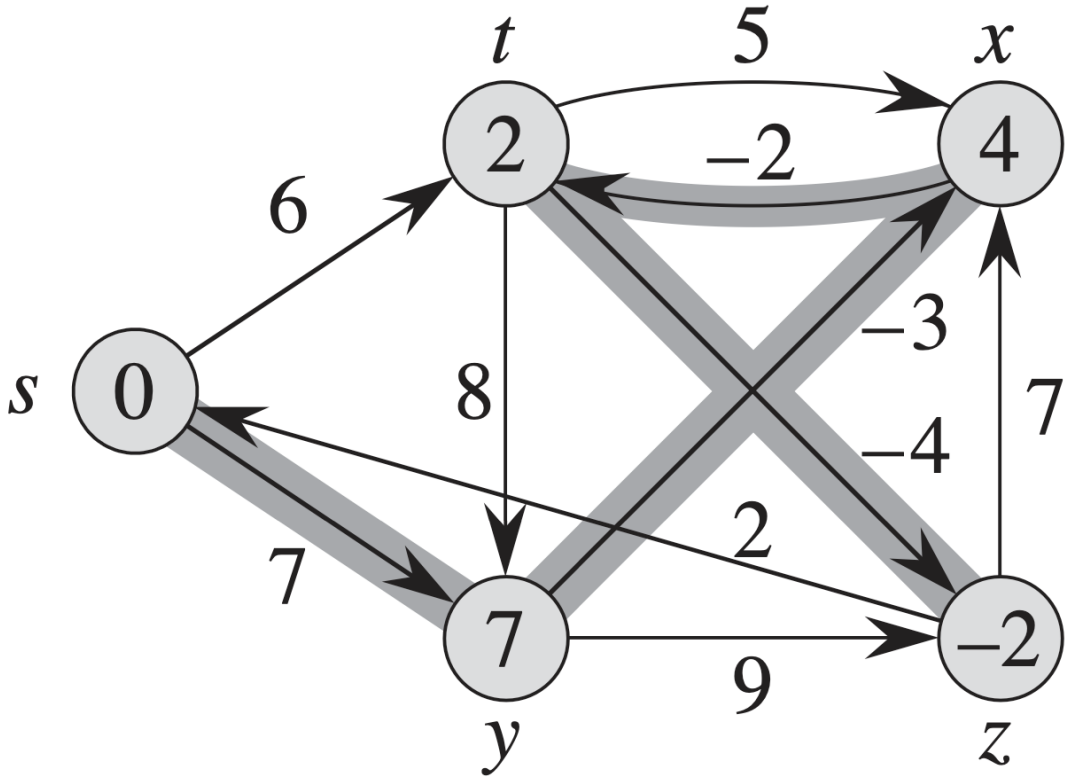
Single-Source Shortest Paths



Single-Source Shortest Paths



(d)



(e)

Single-Source Shortest Paths

Path-relaxation property:

- The values we get on each pass and the convergence quickness are depended on the order of relaxation.
- But it guarantees the convergence after $|V| - 1$ passes, assuming there are no negative-weight cycles.
  - Let v be reachable from s, and let $p = \langle v_0, \ldots, v_k \rangle$ be $\delta(s, v)$ where $v_0 = s$ and $v_k = v$.
  - Then, it has $|E| \leq |V| - 1$ so $k \leq |V| - 1$ since p is acyclic.
    - The for-loop relaxes all edges iteratively.
- We have $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$.

Single-Source Shortest Paths

BELLMAN-FORD correctness:

- Suppose there is no negative-weight cycle reachable from s. At the termination.
  - Then $\forall\ (u, v) \in E,\ \ v.d = \delta(s, v) = u.d + w(u, v)$.
  - $\therefore$ BELLMAN-FORD returns TRUE.
- If there is a negative-weight cycle, $c = \langle v_0, \ldots, v_k \rangle$, $v_0 = v_k$ and reachable from s, then $\sum_{i=1}^{k} (v_{i-1}, v_i) < 0$.
  - For contradiction, suppose BELLMAN-FORD returns TRUE.
  - Then, $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$ for $i = 1, \ldots, k$.
  - $\sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)$
  - The extension gives $\sum_{i=1}^{k} v_{i-1}.d + \sum_{i-1}^{k} w(v_{i-1}, v_i)$.
  - It contradicts c being a negative-weight cycle $\Rightarrow$ the latter term is less than 0.

Single-Source Shortest Paths

DAG-SHORTEST-PATHS Algorithm:

- For directed acyclic graphs.

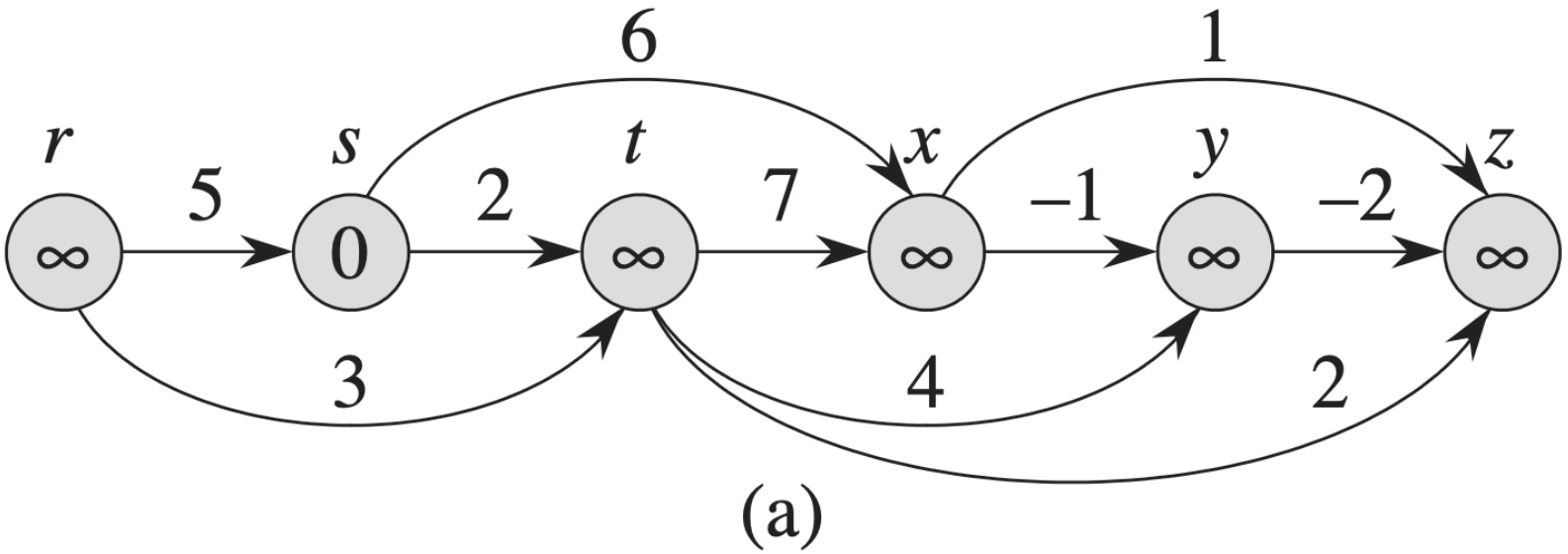
Algorithm (DAG-SHORTEST-PATHS(G, w, s)):

```
1 topologically sort the vertices
2 INIT-SINGLE-SOURCE(G,s)
3 foreach u do
4    foreach (v ∈ G.Adj[u]) do
4       RELAX(u,v,w)
```

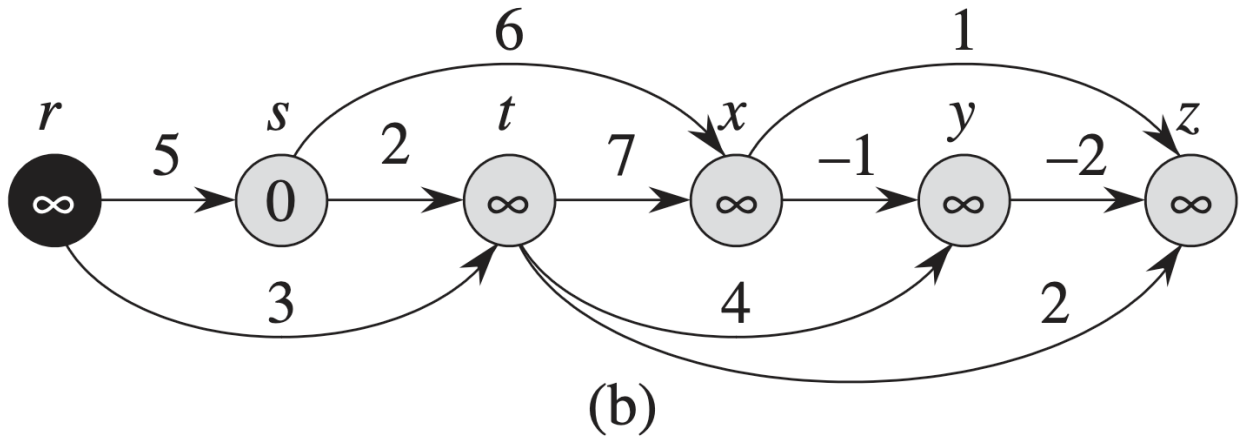
Correctness:

- Vertices are topologically sorted, and edges of any path must be relaxed in order of appearance in the path.
  - Edges of any shortest path are relaxed in the order by path-relaxation property.

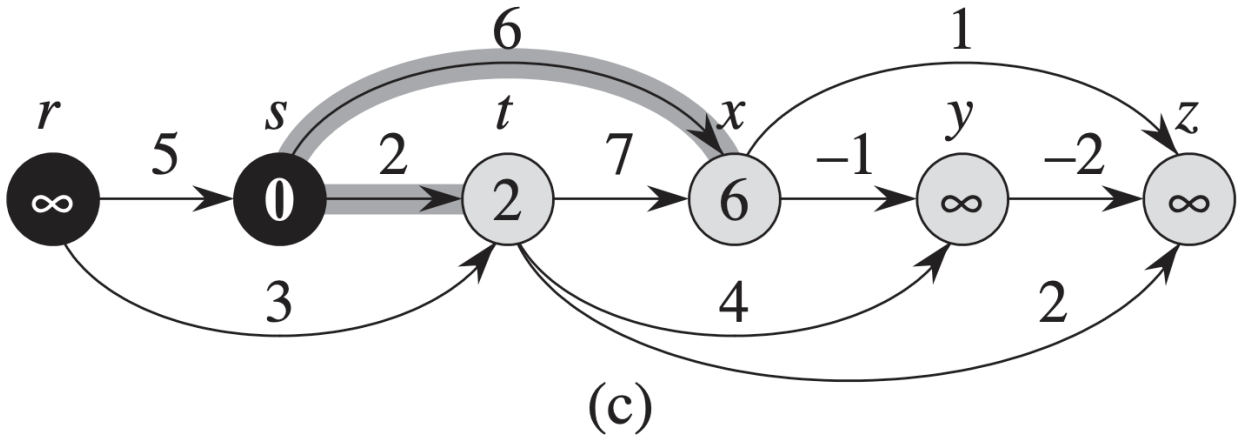
Single-Source Shortest Paths



Single-Source Shortest Paths

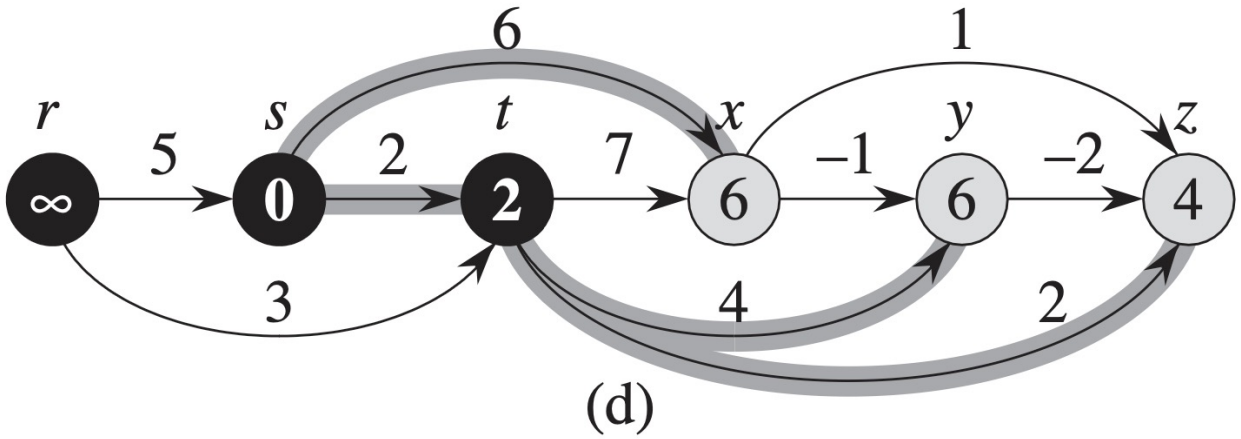


(b)

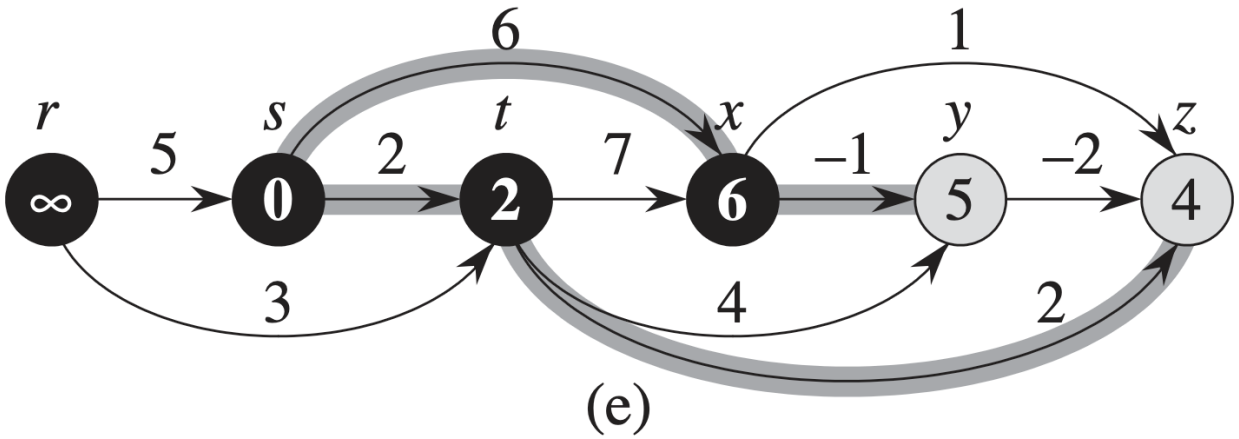


(c)

Single-Source Shortest Paths

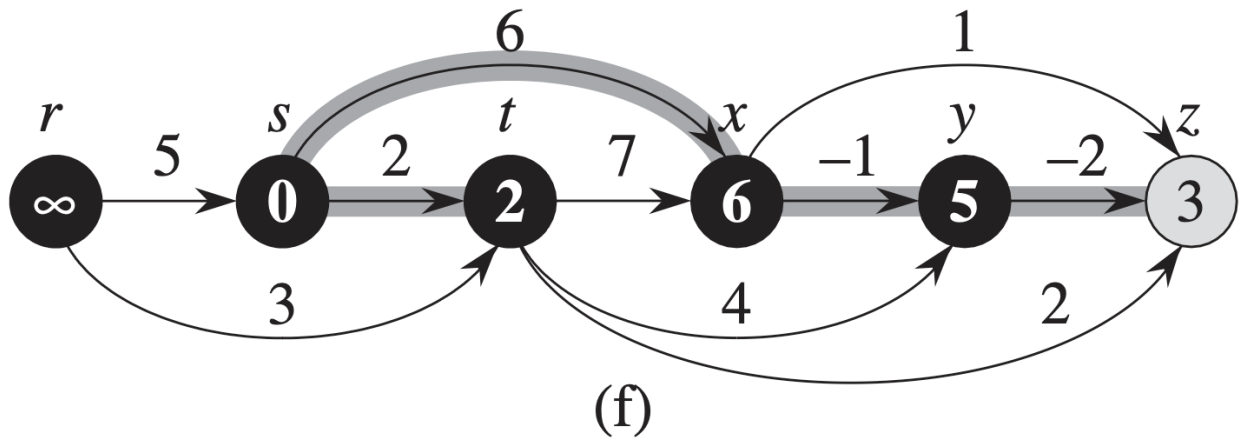


(d)

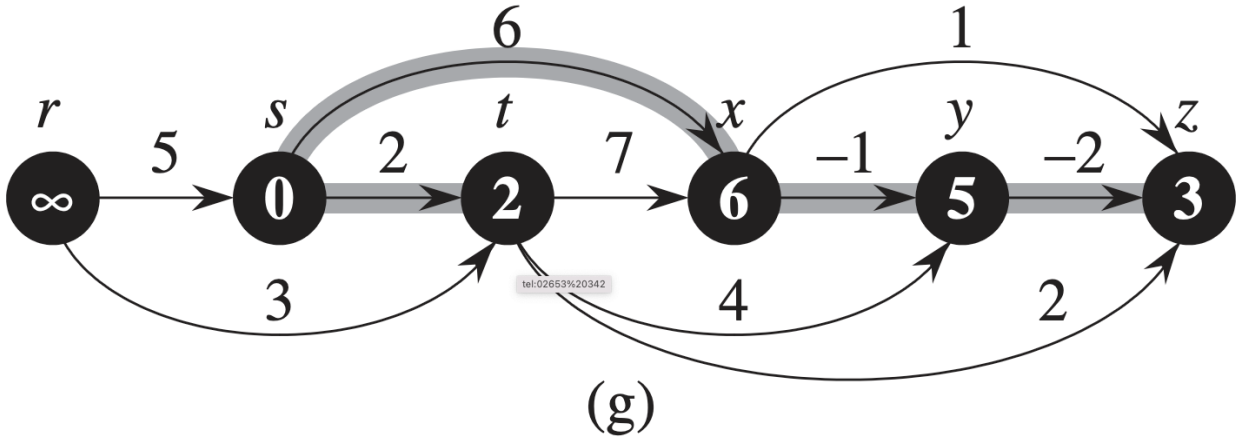


(e)

Single-Source Shortest Paths



(f)



(g)

Single-Source Shortest Paths

Dijkstra Algorithm:

- No negative-weight edges.
- A weighted BFS generalization:
  - Use a priority queue.
  - Keys are the shortest-path weights (v.d).
- Two sets of vertices:
  - S: vertices whose final shortest-path weights are determined.
  - Q: Q=V-S.

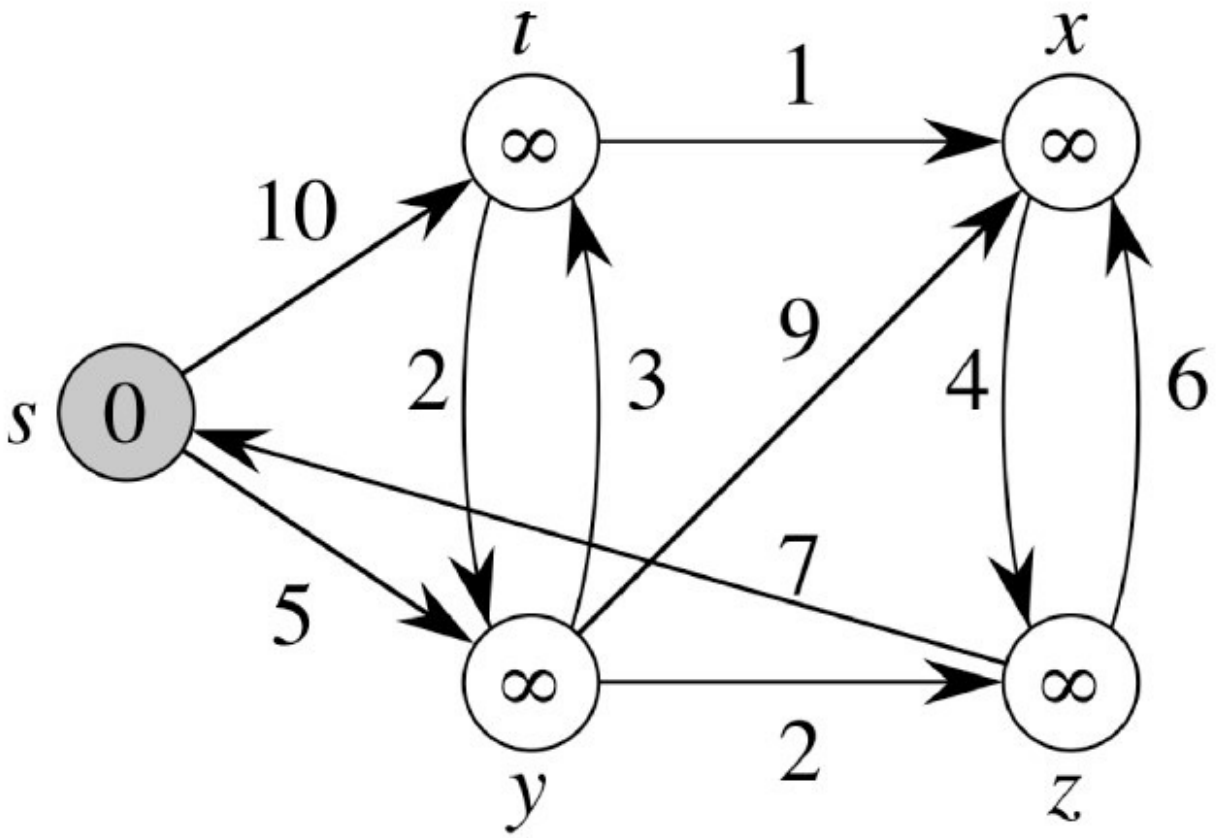
Single-Source Shortest Paths

Algorithm (DIJKSTRA(G, w, s)):

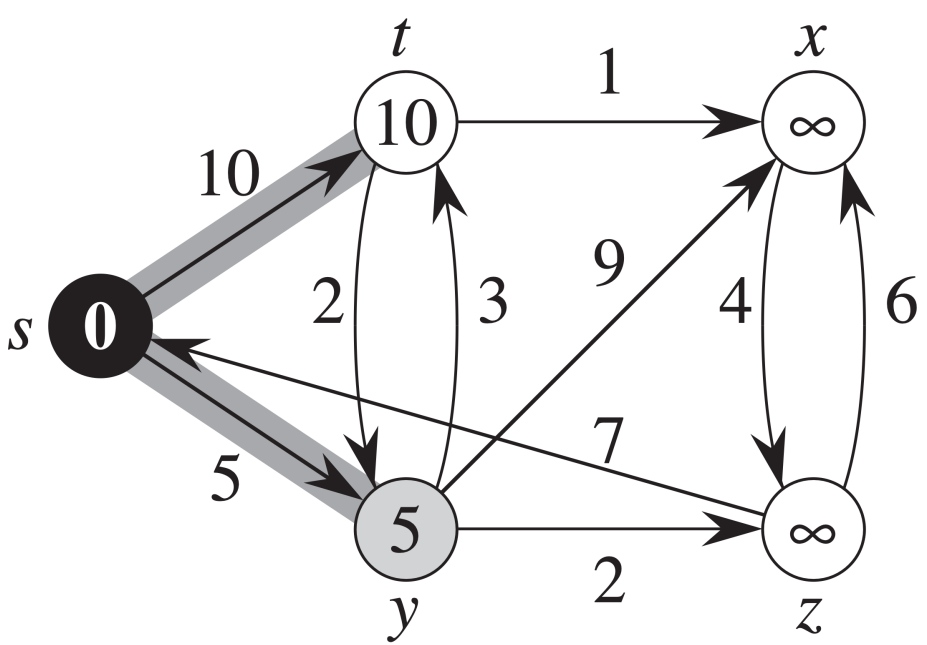
```
1  INIT-SINGLE-SOURCE(G,s)
2  S = ∅
3  Q = G.V
4  while (Q ≠ ∅) do
5     u = EXTRACT-MIN(Q)
6     S = S ∪ {u}
7     foreach (v ∈ G.Adj[u]) do
8        RELAX(u,v,w)
```

Single-Source Shortest Paths

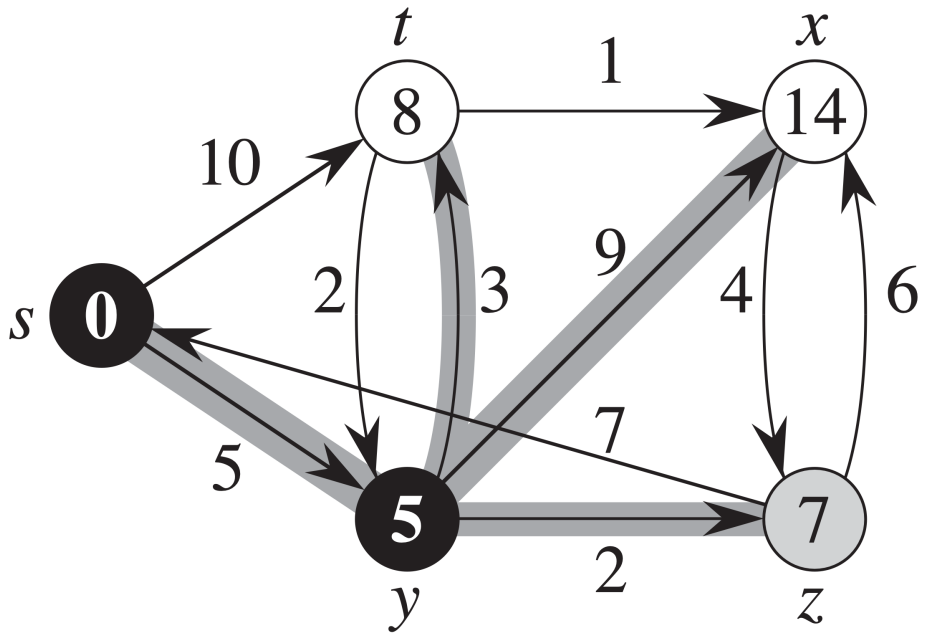
Example:



Single-Source Shortest Paths

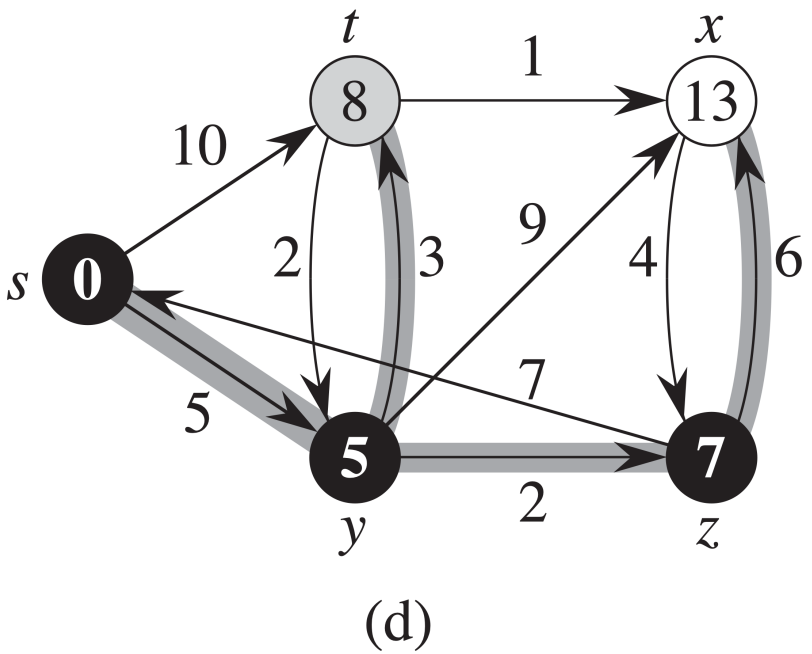


(b)

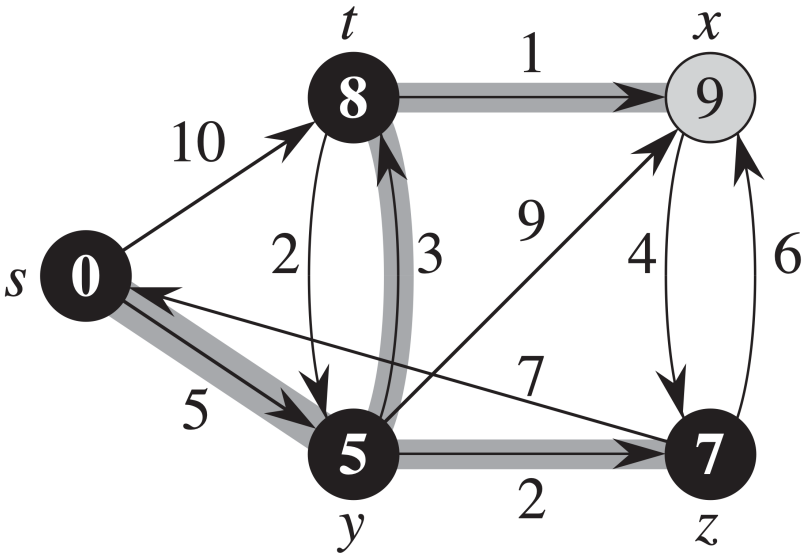


(c)

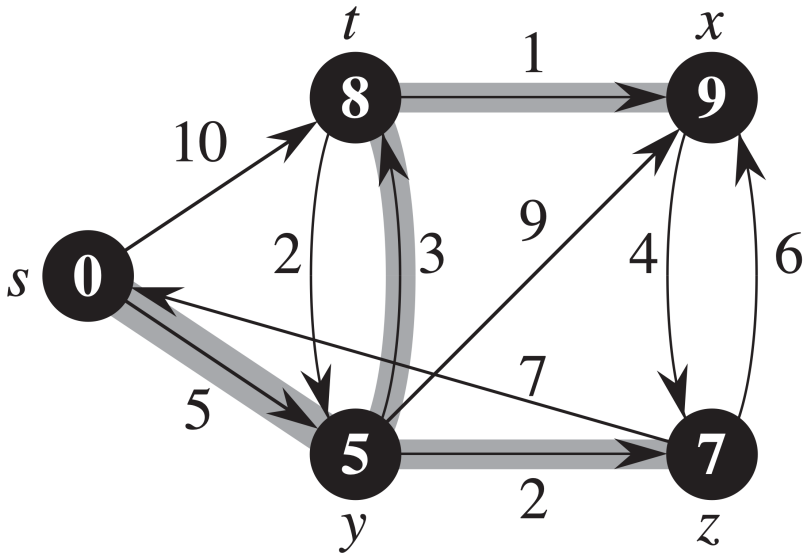
Single-Source Shortest Paths



(d)



(e)



(f)

Single-Source Shortest Paths

Dijkstra Algorithm correctness:

- Similar to Prim's algorithm except for the computation of v.d and the use of shortest-path weights as keys.
- It can be viewed as greedy.
  - Always choose the "lightest" or "closest" vertex in $V - S$ to add to S.
- Loop Invariants: At the start of each while-loop iteration, $v.d = \delta(s, v) \forall\ v \in V$.
  - Initialization: Start with $S = \emptyset$.
  - Termination: $Q = \emptyset \Rightarrow S = V \Rightarrow v.d = \delta(s, v)\ \forall\ v \in V$.
  - Maintenance: when u is added to S, $u.d = \delta(s, u)$.

Single-Source Shortest Paths

- Suppose there exists u such that $u.d \neq \delta(s,u)$ and let u be the first vertex added to S.
- We find that $u \neq s$ and there is some path $s \rightarrow u$. Otherwise, $u.d = \infty$.
- Just before u is added, $s \in S$ and $u \in V - S$.
- Assume that there are vertex y in $s \rightarrow u$ that is in $V - S$ and a vertex x that is in S. Then $p_1 = \langle s, x, y, u \rangle$ or $p_2 = \langle s, x, u \rangle$.
- Then we find that $\delta(s,y) \leq \delta(s,u)$.
- If u and y are in Q, we choose u so that gives $\delta(s,u) \leq \delta(s,y)$.
- Therefore, $u.d = y.d$ all the time.
- This proves that Dijkstra algorithm is always correct.

Single-Source Shortest Paths

Dijkstra Algorithm Analysis:

- It depends on the implementation of the priority queue.
- If Binary Heap is used, each operation will take $O(\lg V)$ time so $T = O(E \lg V)$.

All-Pairs Shortest Paths

- A weighted directed graph $G = (V, E)$ with $w: E \rightarrow \mathbb{R}$ and $|V| = n$.
- Assume that vertices are numbered from 1 to n.
- We are going to create $n \times n$ matrix $D = (d_{ij})$ of shortest-path distances so $d_{ij} = \delta(i, j) \; \forall \; i, j$.
- We can treat the problem as a single-source shortest path on every vertex.
  - If BELLMAN-FORD is used for each vertex, we will have $O(V^2 E)$.
    - It can be $O(V^4)$ if a graph is dense (e.g., $|E| = |V^2|$)
  - If the Dijkstra algorithm is used,
    - Suppose the graph has no negative-weight cycles.
    - It can be $O(VE \lg V)$.
      - For a dense graph, the running time will be $O(V^3 \lg V)$.

All-Pairs Shortest Paths

SLOW-APSP Algorithm:

Shortest paths matrix:

- Assume that G is given as the adjacency matrix of weights $W = (w_{ij})$ with vertices numbered from 1 to n.:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w_{ij} & \text{if } i \neq j, (i,j) \in E. \\ \infty & \text{if } i \neq j, (i,j) \notin E \end{cases}$$

Optimal substructure:

- Recall that the sub-paths of the shortest path are the shortest paths.

All-Pairs Shortest Paths

Recursive solution:

- Let $l_{ij}^{(m)} = \delta(i, j)$ that contains $\leq m$ edges.
  - If $m = 0$, there is a shortest path $i \to j$ if and only if $i = j$:

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}.$$

  - If $m \geq 1$,

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right)$$

where k ranges over all possible predecessor of j.

Since $w_{jj} = 0$, it is equivalent to

$$l_{ij}^{(m)} = \min_{1 \leq k \leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}.$$

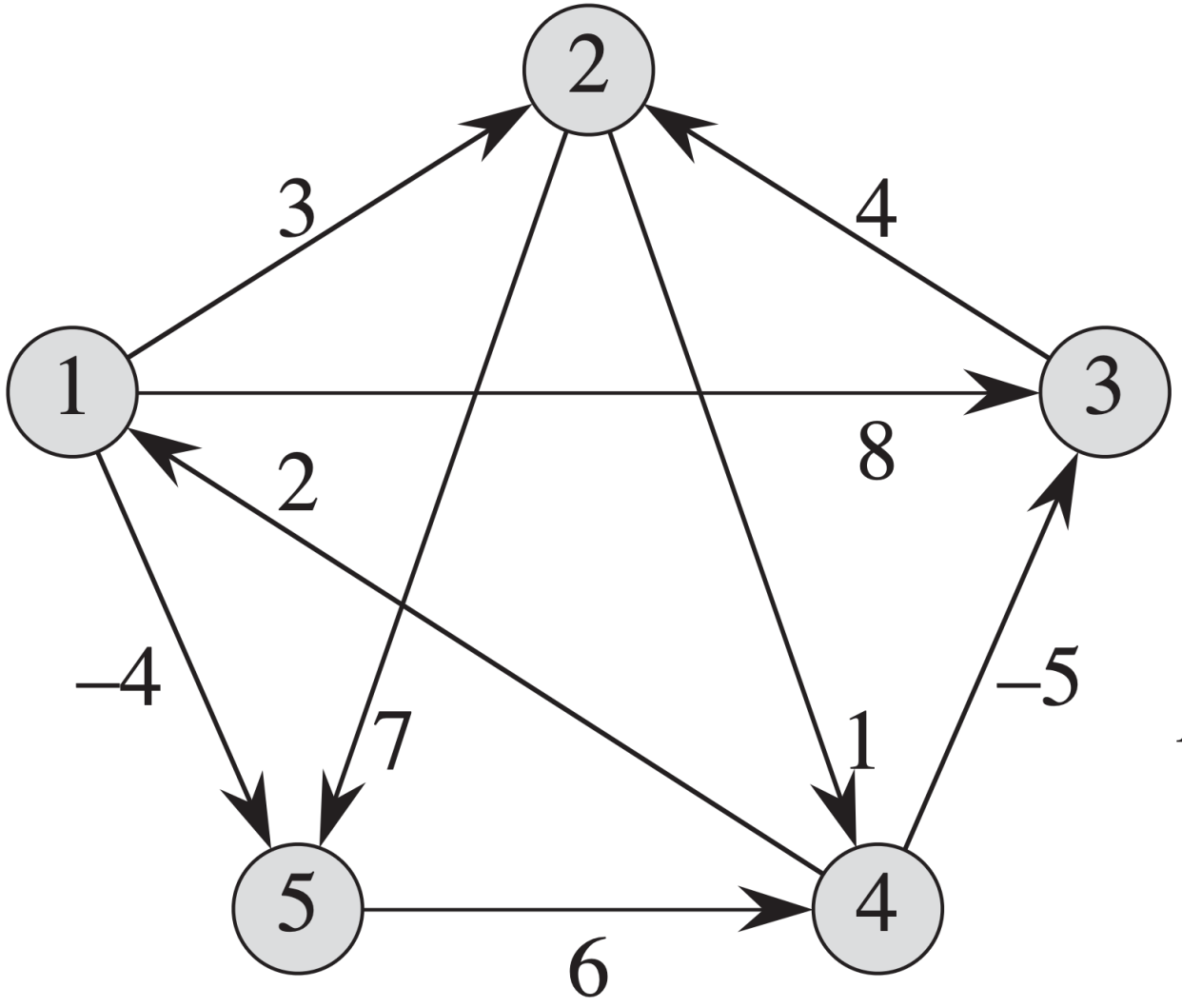
  - When $m = 1$, $l_{ij}^{(1)} = w_{ij}$.
  - All simple shortest paths contain $\leq n - 1$ edges.
    - $\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots$.

All-Pairs Shortest Paths

Bottom-up fashion:

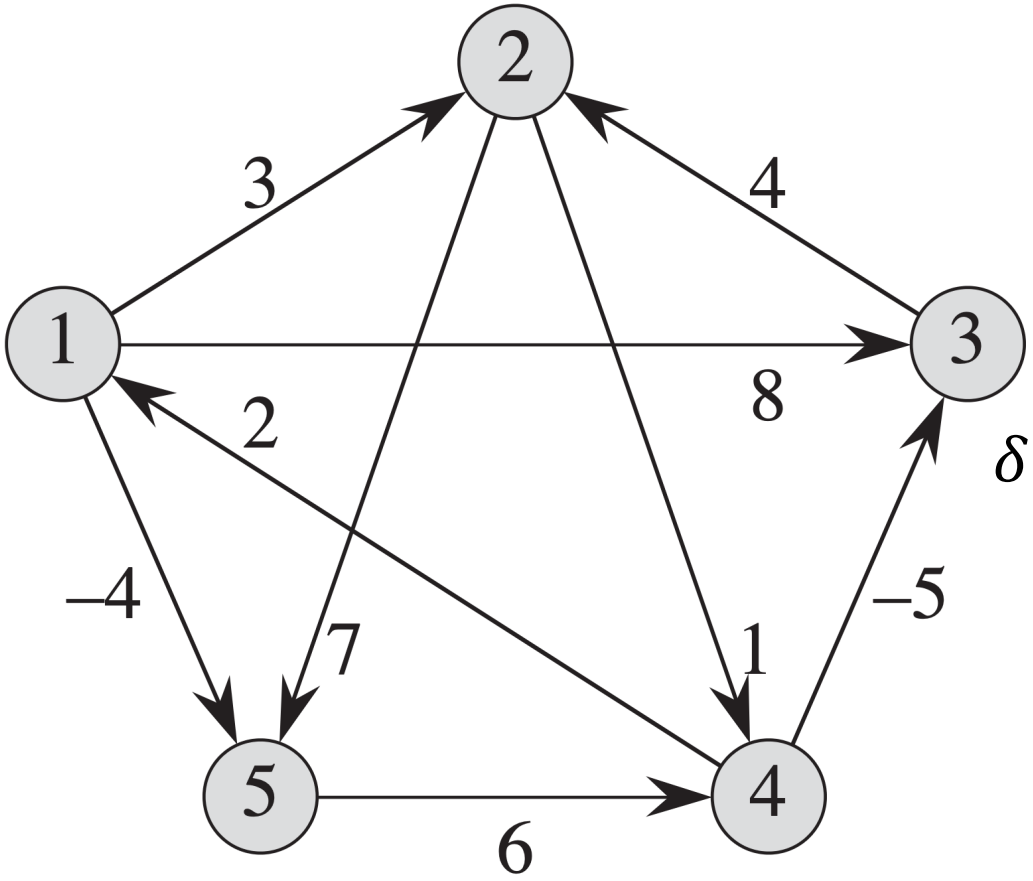
- We compute $L^m = \left(l_{ij}^{(m)}\right)$:
  - $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$
- We start with $L^{(1)} = W$ and go from $L^{(m-1)}$ to $L^{(m)}$.

All-Pairs Shortest Paths



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

All-Pairs Shortest Paths



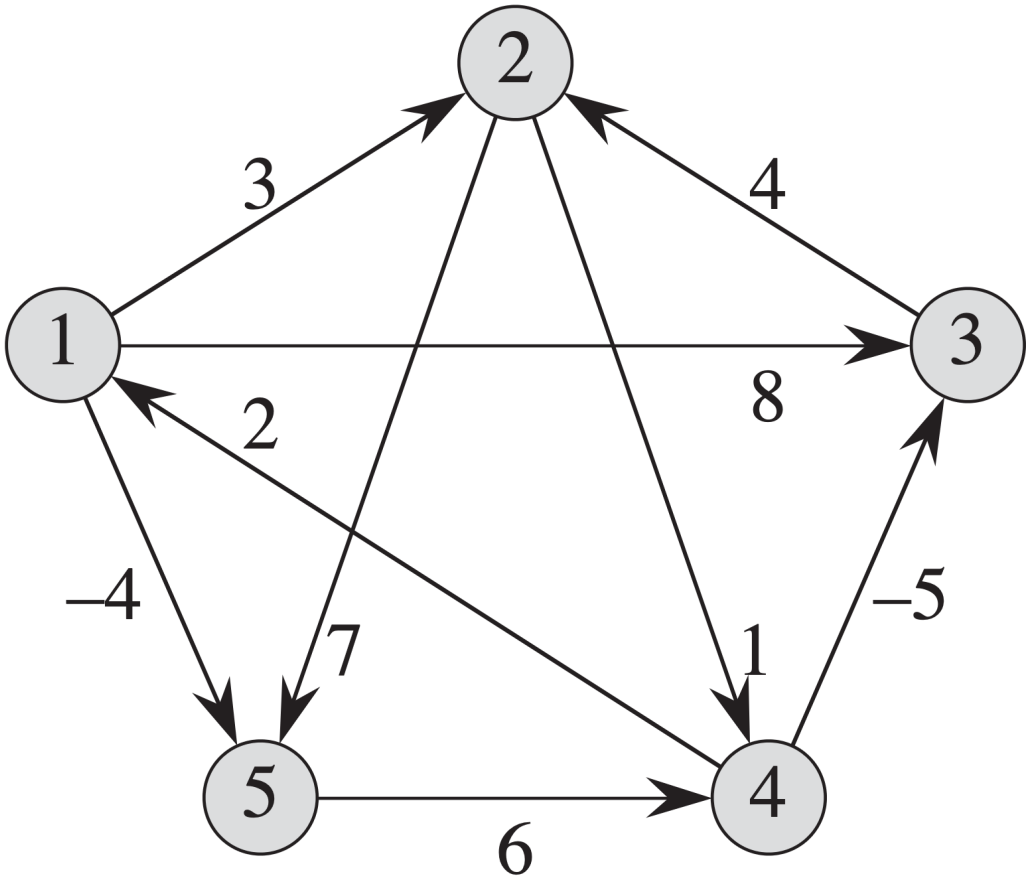
$$\delta(1,4) = \min(\langle 1,2,4\rangle, \langle 1,5,4\rangle)$$

$$L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$\delta(1,3) = \min\left(\langle 1,3\rangle, \left\langle l^{(2)}_{\langle 1,2,4\rangle}, 3\right\rangle, \left\langle l^{(2)}_{\langle 1,5,4\rangle}, 3\right\rangle\right)$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

All-Pairs Shortest Paths



$$m = 5 - 1$$

$$L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

All-Pairs Shortest Paths

Algorithm (EXTEND(L, W, n)):

```
1 Let L' = (l'_ij)
2 for (1 ≤ i ≤ n) do
3    for (1 ≤ j ≤ n) do
4       l'_ij = ∞
5       for (1 ≤ k ≤ n) do
6          l'_ij = min(l'_ij, l_ik + w_kj)
7 return L'
```

Algorithm (SLOW-APSP(W, n)):

```
1 L^(1) = W
2 for (2 ≤ m ≤ n − 1) do
3    let L^(m) = ∅
4    L^(m) = EXTEND(L^(m−1), W, n)
5 return L^(n−1)
```

All-Pairs Shortest Paths

FASTER-APSP Algorithm:

- We see that it is okay to overshoot by the convergence theorem.
- We can update m twice faster.

Algorithm (FASTER-APSP(W, n)):

```
1  m = 1
2  while (m ≤ n − 1) do
3     let L^(2m) = ∅
4     L^(2m) = EXTEND(L^(m), L^(m), n)
5     m = 2m
6  return L^(m)
```

All-Pairs Shortest Paths

Floyd-Warshall algorithm

- It uses a different dynamic-programming approach.
- For $p = \langle v_1, \ldots, v_l \rangle$, an intermediate vertex is any vertex of p other than v_1 or v_l.
- Let $d_{ij}^{(k)}$ be shortest-path weight of any path $i \to j$ with all intermediate vertices in $\{1,2, \ldots, k\}$.
- Consider a shortest path $i \to j$ with all intermediate vertices of p are in $\{1,2, \ldots, k\}$:
  - If k is not an intermediate vertex, then all intermediate vertices of p are in $\{1,2, \ldots, k-1\}$.

All-Pairs Shortest Paths

Recursive formulation:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} & \text{if } k \geq 1 \end{cases}$$

- $d_{ij}^{(0)} = w_{ij}$, because no intermediate vertices $\Rightarrow \leq 1$ edge.
- We want $D^{(n)} = \left(d_{ij}^{(n)}\right)$, since all vertices numbered $\leq n$.

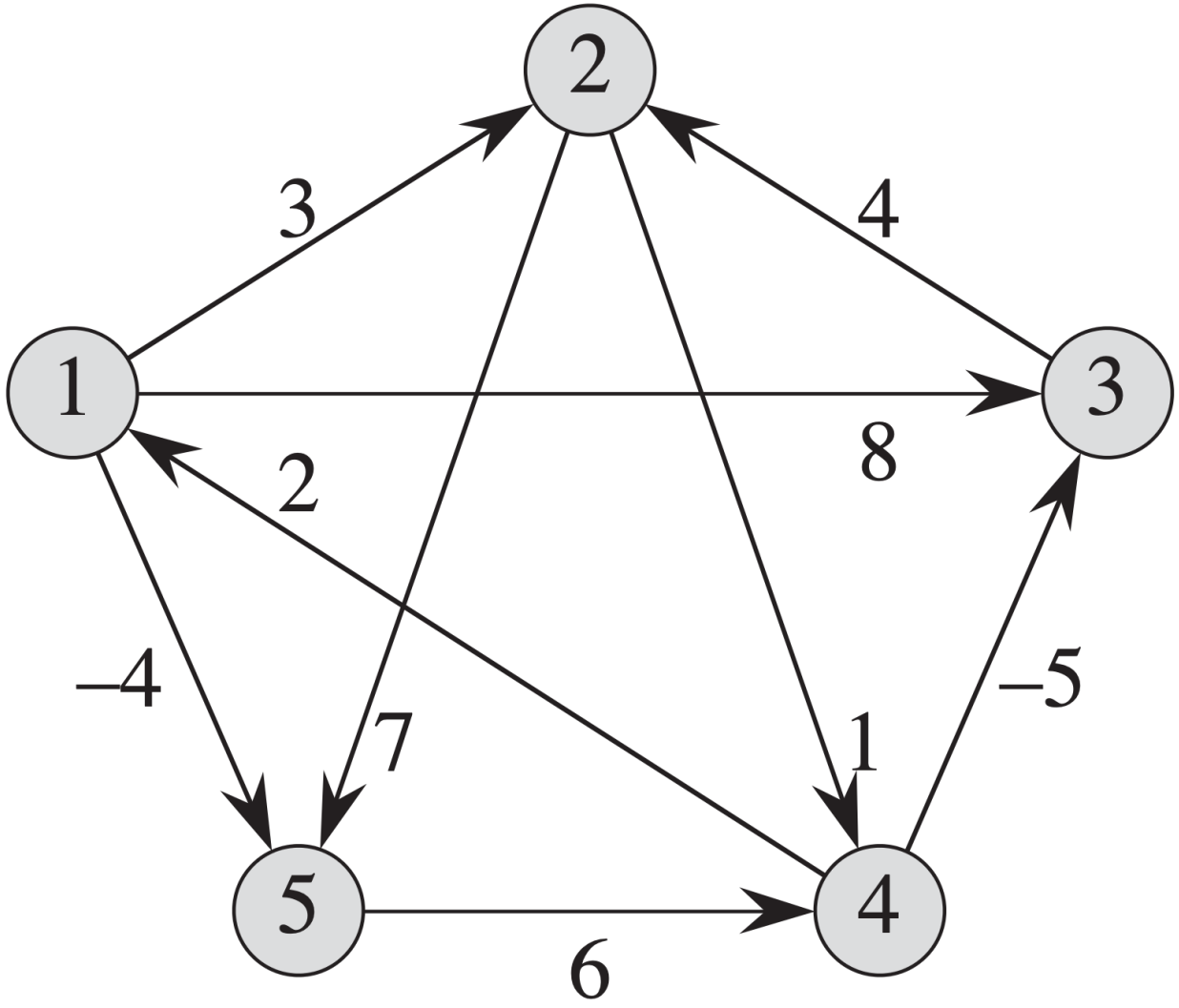
Bottom-up Fashion: compute in increasing order of k.

All-Pairs Shortest Paths

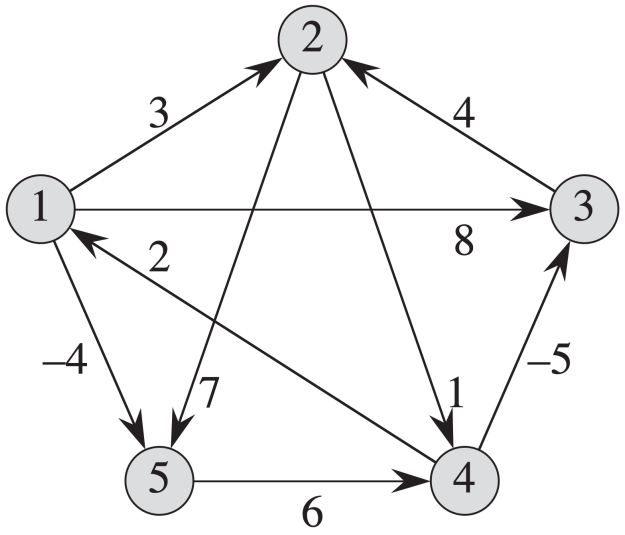
Algorithm (FLOYD-WARSHALL(W, n)):

1 $\mathbf{D}^{(1)} = \mathbf{W}$
2 **for** $(1 \leq k \leq n)$ **do**
3 **let** $D^{(k)} = \left(d_{ij}^{(k)}\right)$
3 **for** $(1 \leq i \leq n)$ **do**
4 **for** $(1 \leq j \leq n)$ **do**
5 $d_{ij}^{(k)} = min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
7 **return** $D^{(n)}$

All-Pairs Shortest Paths

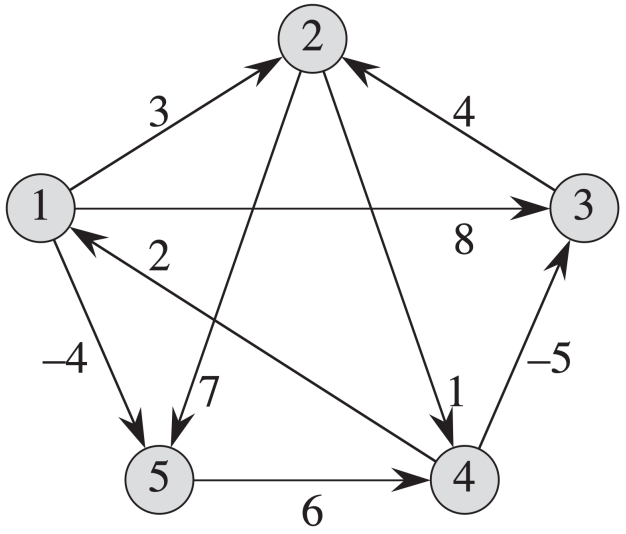


All-Pairs Shortest Paths



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

All-Pairs Shortest Paths

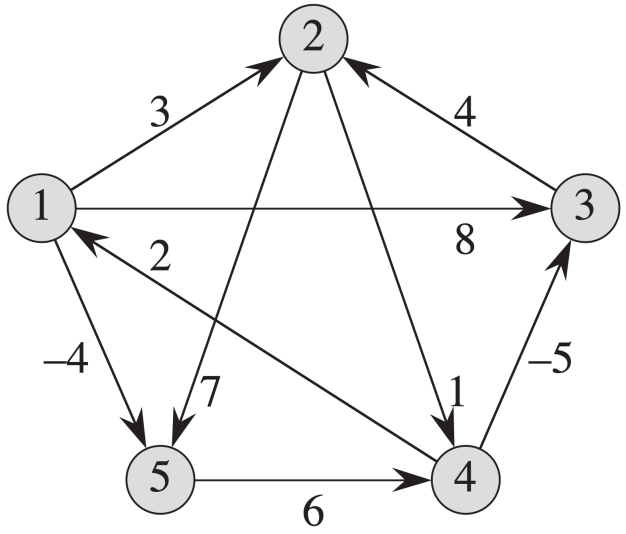


$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

All-Pairs Shortest Paths



$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$