Lecture 7
Single-Source Shortest Paths Problems

1. Related Notions
2. Variants of Shortest-Paths Problems
3. Properties of Shortest-Paths and Relaxation
4. The Bellman-Ford Algorithm

Shortest Paths—Notions

1. Given a weighted, directed graph $G = (V, E; W)$, the **weight of path** $p = <v_0, v_1, \ldots, v_k>$ is the sum of the weights of its constituent edges, that is, $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$
2. The **shortest-path weight** from u to v is defined as

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

3. A **shortest path** from vertex u to v is any path p with weight $w(p) = \delta(u, v)$.

Single-source shortest path problem

- Input: A weighted directed graph $G=(V, E; W)$ and a source vertex s.
- Output: Shortest-path weight from s to each vertex v in V, and a shortest path from s to each vertex v in V if v is reachable from s.

Variants of Single-source shortest-path problem

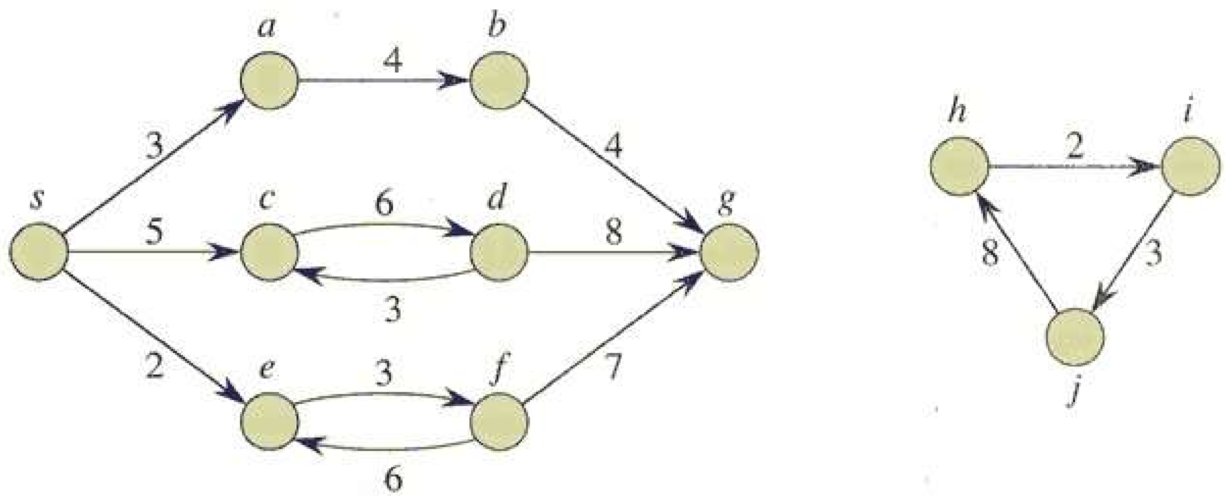
- Single-source shortest-paths problem
- Single-destination shortest-paths problem

    --- *reverse the direction of edge*
- Single-pair shortest-path problem

    --- *consider u as source vertex*
- All-pairs shortest-paths problem

    --- *topic of next chapter*

Shortest paths – an example



Please caculate δ(s,a), δ(s,b), …

Optimal Substructure of Shortest Paths

- Lemma 24.1 (Subpaths of shortest paths are shortest paths)
- Given a weighted, directed graph $G = (V, E; W)$, let $p = <v_1, v_2, \ldots, v_k>$ be a shortest path from vertex v_1 to vertex v_k and, for any i and j such that $1 \leq i \leq j \leq k$, let $p_{ij} = <v_i, v_{i+1}, \ldots, v_j>$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

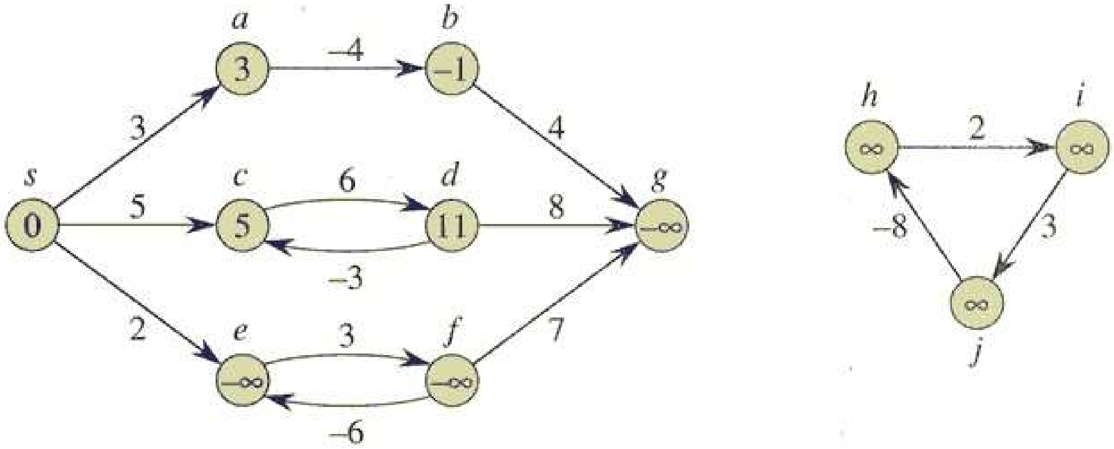
Proof

- The path p can be decomposed into

$$v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$$

- Then we have $w(p)=w(p_{1i})+w(p_{ij})+w(p_{jk})$.
- Assume thet there is a path p_{ij}' from v_i to v_j with weight $w(p_{ij}')< w(p_{ij})$.
- Then $v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}'} v_j \xrightarrow{p_{jk}} v_k$ is a path from v_1 to v_k whose weight is less than $w(p)$, which contradicts the assumption that p is a shortest path from v_1 to v_k.

Negative-weight edge and negative-weight cycle



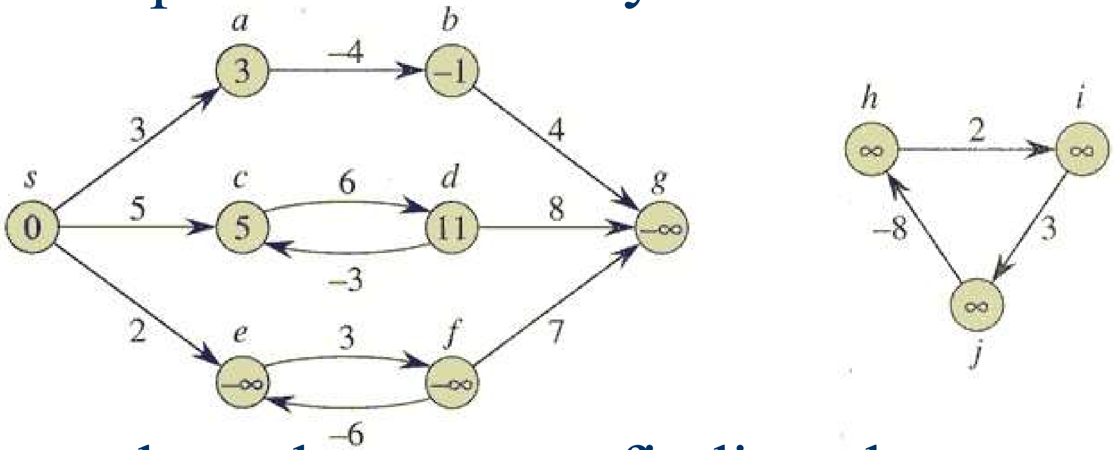
Please caculate δ(s,a), δ(s,b), δ(s,c),…

If there is a negative-weight cycle on some path from *s* to *v*, we define δ(*s*, *v*) = -∞.

Cycles

Three typies of cycle: negative-weight cycle,
positive-weight cycle,
0-weight cycle

Can a shortest path contains a cycle?



We can assume that when we are finding shortest paths, they are cycle-free. Any acyclic path contains at most $|V|$ distinct vertices, it also contains at most $|V|$-1edges.Thus, we can restrict our attention to shortest paths of at most $|V|$-1 edges.

Representing Shortest-Paths

- Given a graph $G = (V, E)$, maintain for each vertex $v \in V$ a **predecessor** $\pi[v]$ that is either another vertex or NIL.
- Given a vertex v for which $\pi[v] \neq$ NIL, the procedure PRINT-PATH(G, s, v) prints a shortest path from s to v.

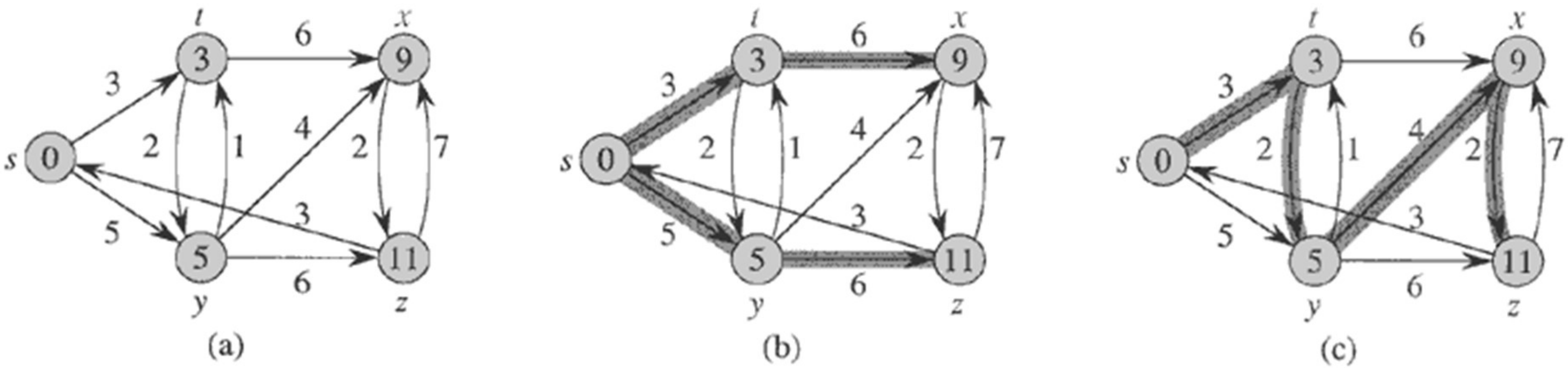
```
PRINT-PATH(G, s, v)
if v = s
    then print s
    else if π[v] = NIL
            then print "no path from" s "to" v "exists"
            else PRINT-PATH(G, s, π[v])
                 print v
```

Predecessor subgraph

- Predecessor subgraph $G_\pi = (V_\pi, E_\pi)$:
  - $V_\pi = \{v \in V: \pi[v] \neq \text{NIL}\} \cup \{s\}$
  - $E_\pi = \{(\pi[v]\ , v) \in E: v \in V_\pi - \{s\}\}$
- We shall prove that π values produced by the algorithms in this chapter have the property that at termination G_π is a "shortest-paths tree"

Shortest-Paths Tree

- Let $G = (V, E; W)$ be a weighted, directed graph, and assume that G contains no negative cycles that are reachable from s. A **shortest-paths tree** rooted at s is a directed sub-graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that
  - V' is the set of vertices reachable from s in G,
  - G' forms a rooted tree with root s, and
  - For all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.
- The shortest-path tree may not be unique.



The technique of Relaxation

- In our shortest-paths problem, we maintain an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to v.
- We call $d[v]$ a shortest-path estimate.

  i.e., $d[v]$ is an estimate of $\delta(s, v)$.
- The term Relaxation is used here for an operation that tightens an upper bound.

Relaxation--Initialization

- The initial estimate of $\delta(s, v)$ can be given by:

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in V[G]$
  do $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

Relaxation Process

- Relaxing an edge (u, v) consists:
  - Testing whether we can improve the shortest path from s to v found so far by going through u to v, and if so,
  - Updating $d[v]$ and $\pi[v]$.
- A relaxation step may either **decrease** the value of the shortest-path estimate $d[v]$ and **update** v's predecessor $\pi[v]$, or **cause no change**.

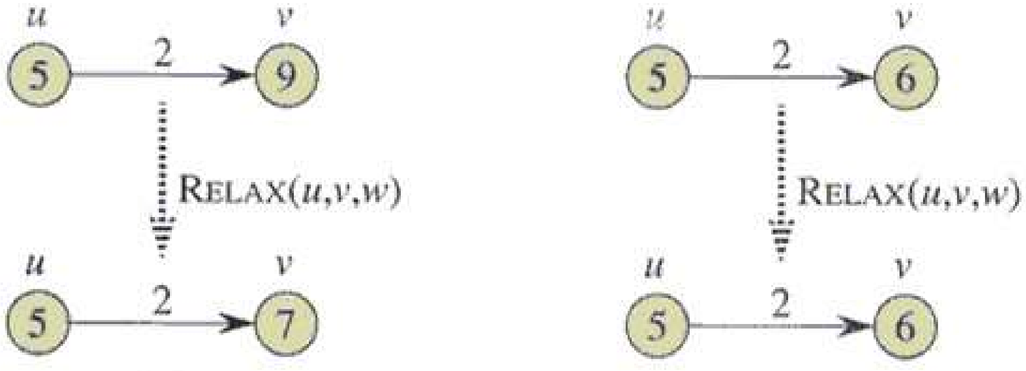
A Relaxation Step

RELAX(u, v, w)

  if $d[v] > d[u] + w(u, v)$

      then $d[v] \leftarrow d[u] + w(u, v)$

            $\pi[v] \leftarrow u$



(a) (b)

Some notes about Relaxation

- Note:
  - Each algorithm in this chapter calls INITIALIZE-SINGLE-SOURCE and then repeatedly relaxes edges.
  - Relaxation is the only means by which shortest-path estimates $d[]$ and predecessors $\pi[]$ change.
  - The algorithms differ in **how many times** they relax each edge and **the order** in which they relax edges.
  - Bellman-Ford algorithm relaxes each edge many times, while Dijkstra's algorithm and the algorithm for directed acyclic graphs relax each edge exactly once.

Properties of Shortest Paths and Relaxation

- Triangle inequality (Lemma 24.10)
  - For any edge $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$
- Upper-bound property (Lemma 24.11)
  - We always have $d[v] \ge \delta(s, v)$ for all vertices $v \in V$, and once $d[v]$ achieves the value $\delta(s, v)$, it never changes.
- No-path property (Lemma 24.12)
  - If there is no path from s to v, then we always have $d[v] = \delta(s, v) = \infty$.

Properties of Shortest Paths and Relaxation

- Convergence property (Lemma 24.14)
  - If $s \sim\rightarrow u \rightarrow v$ is a shortest path from s to v in G, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $d[v] = \delta(s, v)$ at all times afterward.
- **Path-relaxation property** (Lemma 24.15)
  - If $p = \langle s, v_1, \ldots, v_k \rangle$ is a shortest path from s to v_k, and the edges of p are relaxed in the order $(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then after relaxing all the edges, $d[v_k] = \delta(s, v_k)$.
  - Note that other relaxations may take place among these relaxations.

Properties of Shortest Paths and Relaxation

- Predecessor-subgraph property (Lemma 24.17)
  - Once $d[v] = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

Note: Proofs for all these properties can be found in your textbook.

Single-source shortest path problem

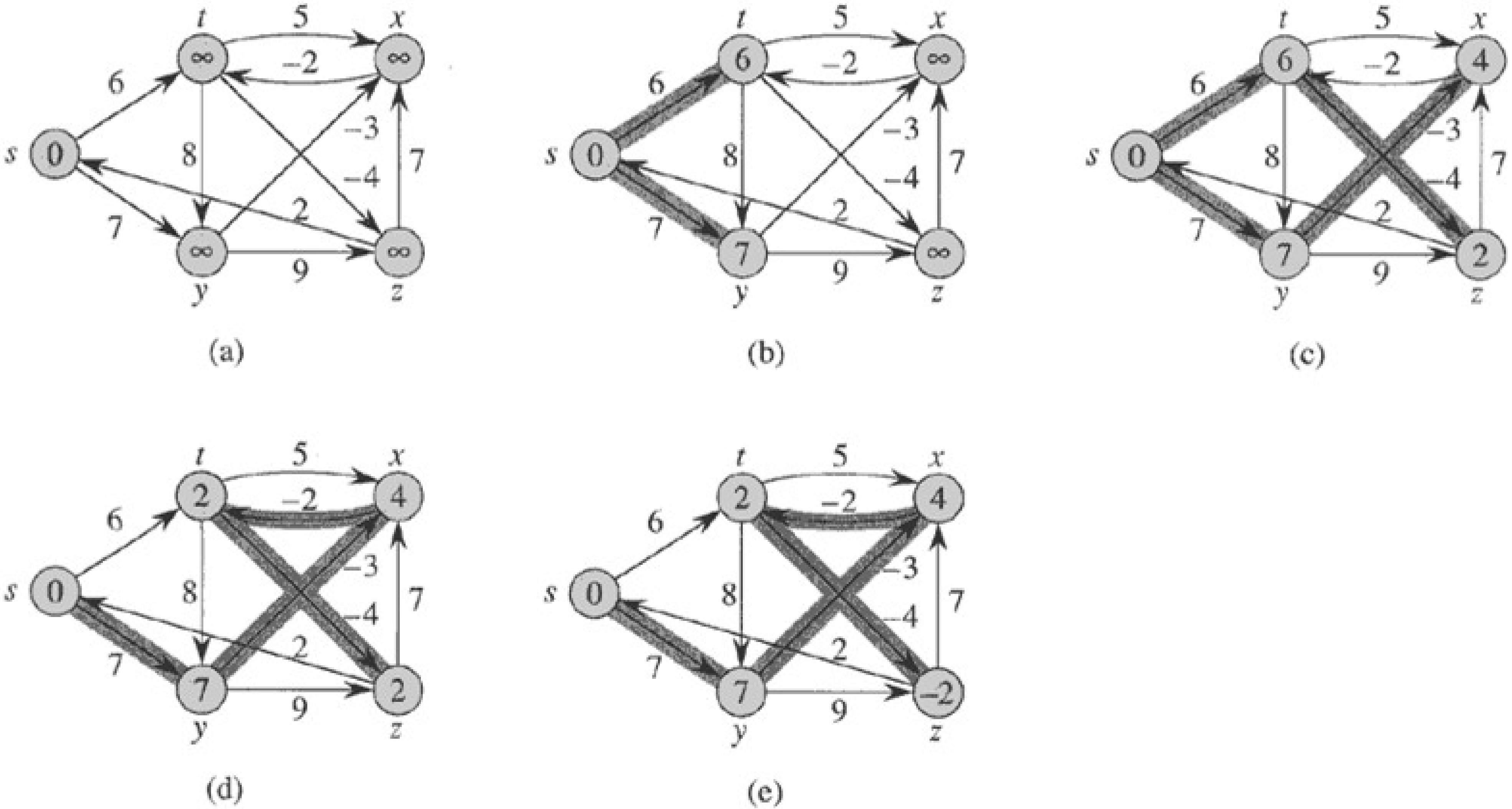
- Input: A weighted directed graph $G=(V, E; W)$ and a source vertex s.
- Output: Shortest-path weight from s to each vertex v in V, and a shortest path from s to each vertex v in V if v is reachable from s.

The Bellman-Ford Algorithm

- To solve the single-source shortest-paths problem
  - The algorithm returns a **boolean** value indicating whether or not there is a negative-weight cycle that is reachable from the source.
  - If there is a negative-weight cycle reachable from the source *s*, the algorithm indicates no solution exists.
  - If there are no such cycles, the algorithm produces the shortest paths and their weights.

The Bellman-Ford $O(VE)$ Algorithm

```
BELLMAN-FORD(G, w, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    for i ← 1 to |V[G]| − 1
        do for each edge (u, v) ∈ E[G]
            do RELAX(u, v, w)
    for each edge (u, v) ∈ E[G]
        do if d[v] > d[u] + w(u, v)
            then return FALSE
    return TRUE
```



Note: each pass relaxes the edges in the order:

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

Time Complexity of The Bellman-Ford

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| − 1
        do for each edge (u, v) ∈ E[G]
                do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
        do if d[v] > d[u] + w(u, v)
                then return FALSE
return TRUE
```

The time complexity is $O(VE)$

Correctness of The Bellman-Ford

- Lemma 24.2
  - Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $w: E \rightarrow \mathbf{R}$, and assume that G contains **no negative cycles that are reachable from *s***. Then, after the $|V| - 1$ iterations of the **for** loop of lines 2-4 of BELLMAN-FORD, we have $d[v] = \delta(s, v)$ for all vertices v reachable from s.

Proof of Lemma 24.2

- Proof: Consider any vertex v that is reachable from s, and let $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$, be any acyclic shortest path from s to v. Path p has at most $|V|$-1 edges, and so $k \leq |V|$-1. Each of the $|V|$-1 iterations of the **for** loop of lines 2-4 relaxes all $|E|$ edges. Among the edges relaxed in the ith iteration, the edge in P is (v_{i-1}, v_i), for $i = 1, 2, \ldots, k$. By the *path-relaxation property*, $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$.

Correctness of The Bellman-Ford

- Corollary 24.3
  - Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $w: E \to \mathbf{R}$. Then for each vertex $v \in V$, there is a path from s to v if and only if BELLMAN-FORD terminates with $d[v] < \infty$ when it is run on G.
  - Proof :Exercise 24.1-2.
  - ($\Rightarrow$)If there is a path P=$\langle v_0, v_1, \ldots, v_k \rangle$ from s to v, where $s=v_0$, $v=v_k$, we show by induction that after the ith iteration, d[v_i]$<\infty$.
  - ($\Leftarrow$) By the *no-path property*.

Correctness of The Bellman-Ford

- Theorem 24.4
  - Let BELLMAN-FORD be run on a weighted, directed graph $G = (V, E)$ with source s and weight function $w: E \rightarrow \mathbf{R}$. If G contains no negative cycles that are reachable from s, then the algorithm returns TRUE, we have $d[v] = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-paths tree rooted at s. If G does contain a negative weight cycle reachable from s, then the algorithm returns FALSE.

Proof of Theorem 24.4

1. Suppose G contains no negative-weight cycles that are reachable from s.
   - Prove that at termination, $d[v] = \delta(s, v)$, for all vertices v in V.
     - Case 1: v is reachable from s; --> *Lemma 24.2*
     - Case 2: v is not reachable from s; --> *No Path Property*
   - Prove that the algorithm returns TRUE

     For each edge (u,v),

     $d[v]= \delta(s, v) \leq \delta(s,u)+w(u,v) = d[u]+w(u,v)$.
2. Suppose that G contains a negative-weight cycle that is reachable from s.
   - Prove that the algorithm returns FALSE

     For each edge on a negative-weight cycle $C=<v_0,v_1\ldots,v_k>$, where $v_0=v_k$

     $$d[v_i] \leq d[v_{i-1}]+w(v_{i-1},v_i), \text{ for } 1 \leq i \leq k.$$

     Summing both sides of equations yields $0 \leq w(C)$, contradiction.

Conclusion

- Relaxation
- The Bellman-Ford Algorithm

Homework

- 24.1-1,24.1-6