Week -9
Topic - Graph Algorithms
CSE – 5311

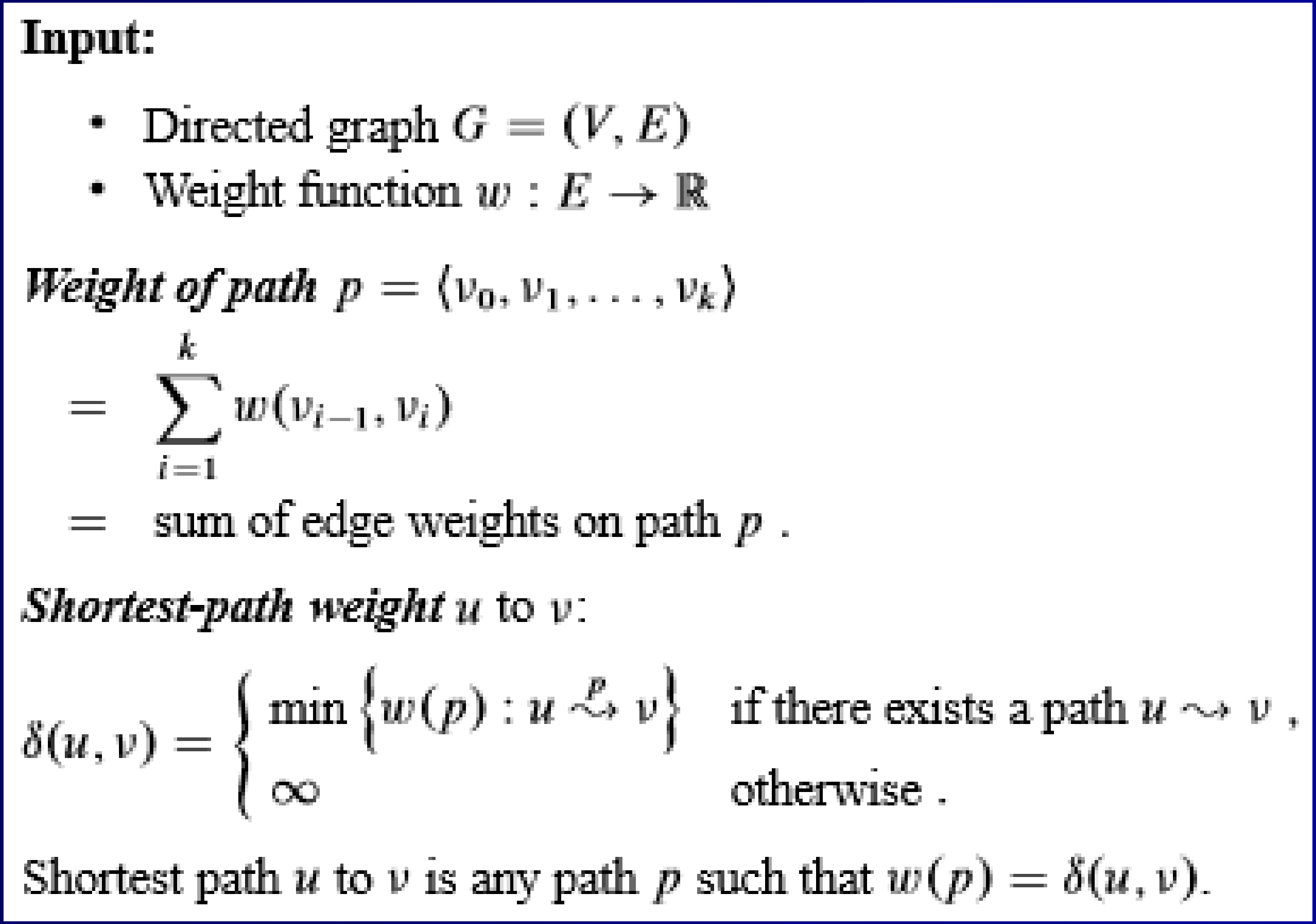
Prepared by:-
Sushruth Puttaswamy
Lekhendro Lisham

Contents

Shortest Paths

- The problem is to find shortest paths among nodes.
- There are 4 different versions of this problem as shown below:
  1. Single source single destination shortest path (SSSP)
  2. Single source all destinations shortest path (SASP)
  3. Shortest path to single destination
  4. All pairs shortest path.
- Shortest paths are based on edge weights
- Weights may represent measures like, time, cost, tolls, loss

Shortest Paths Notations



Input:

- Directed graph $G = (V, E)$
- Weight function $w : E \to \mathbb{R}$

Weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

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

$$= \text{ sum of edge weights on path } p \,.$$

Shortest-path weight u to v:

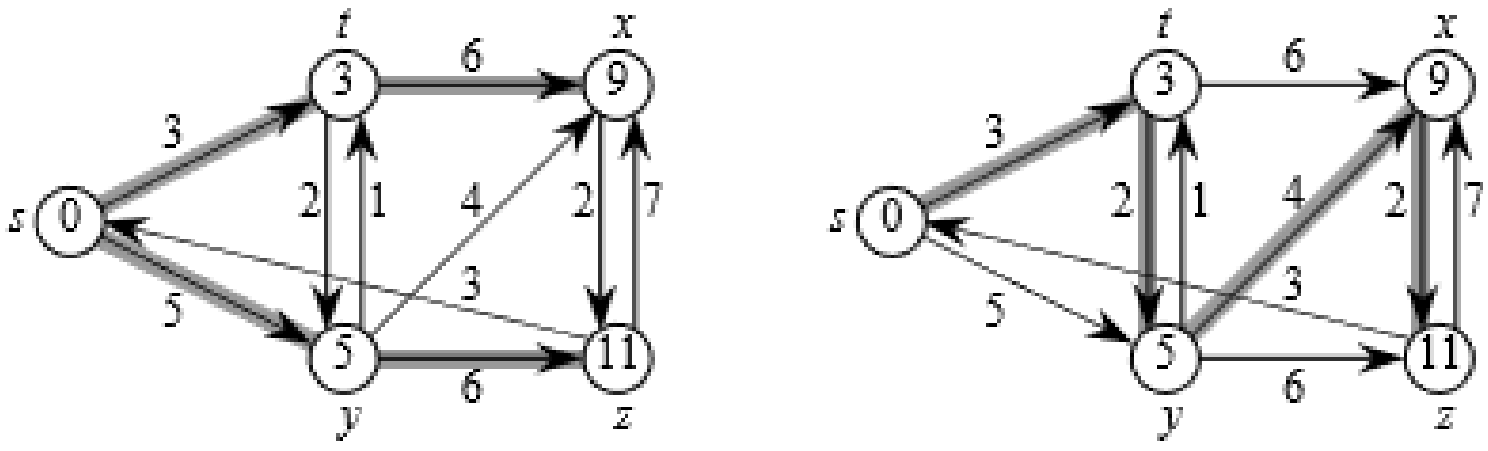
$$\delta(u, v) = \begin{cases} \min\left\{w(p) : u \overset{p}{\leadsto} v\right\} & \text{if there exists a path } u \leadsto v \,, \\ \infty & \text{otherwise} \,. \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u, v)$.

Shortest Paths, Example

shortest paths from *s*

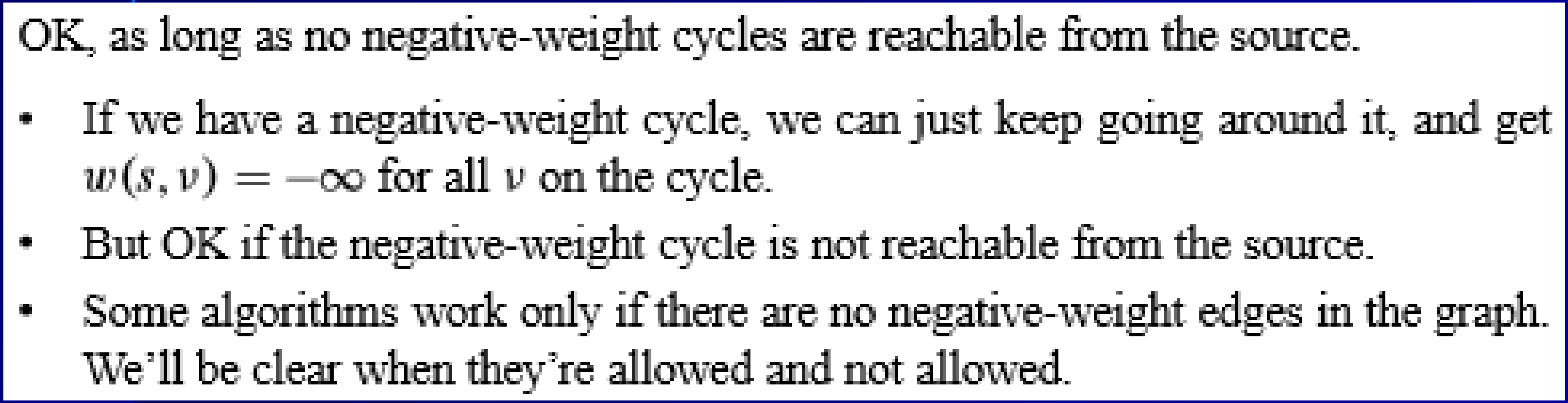
[δ values appear inside vertices. Shaded edges show shortest paths.]



This example shows that the shortest path might not be unique.

It also shows that when we look at shortest paths from one vertex to all other vertices, the shortest paths are organized as a tree.

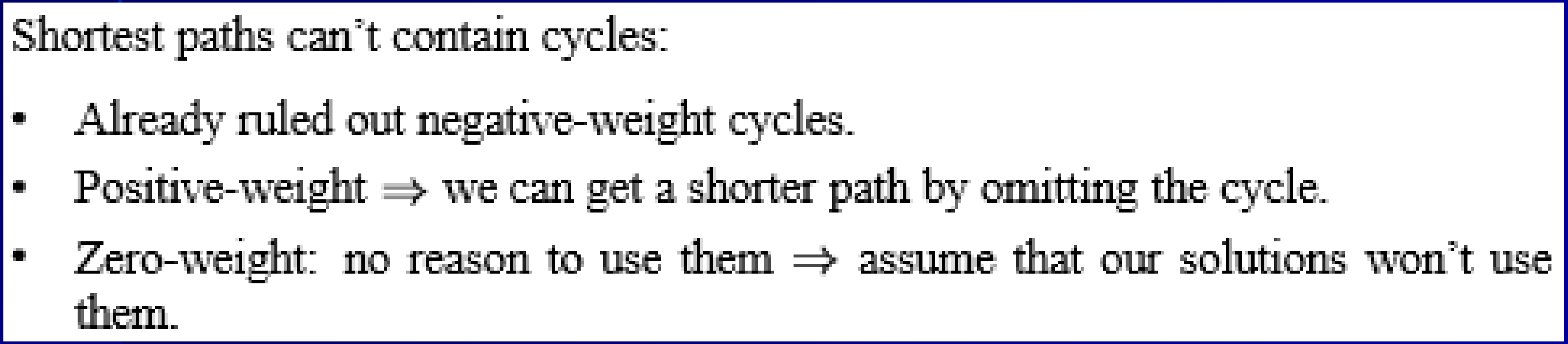
Negative-weight Edges



OK, as long as no negative-weight cycles are reachable from the source.

- If we have a negative-weight cycle, we can just keep going around it, and get $w(s, v) = -\infty$ for all v on the cycle.
- But OK if the negative-weight cycle is not reachable from the source.
- Some algorithms work only if there are no negative-weight edges in the graph. We'll be clear when they're allowed and not allowed.

Cycles



Shortest paths can't contain cycles:

- Already ruled out negative-weight cycles.
- Positive-weight $\Rightarrow$ we can get a shorter path by omitting the cycle.
- Zero-weight: no reason to use them $\Rightarrow$ assume that our solutions won't use them.

Lemma: Optimal Substructure

Any subpath of a shortest path is a shortest path.

Proof Cut-and-paste.

$$\begin{tikzcd} u \arrow[r, squiggly, "p_{ux}"] & x \arrow[r, squiggly, "p_{xy}"] & y \arrow[r, squiggly, "p_{yv}"] & v \end{tikzcd}$$

Suppose this path p is a shortest path from u to v.

Then $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Now suppose there exists a shorter path $x \overset{p'_{xy}}{\leadsto} y$.

Then $w(p'_{xy}) < w(p_{xy})$.

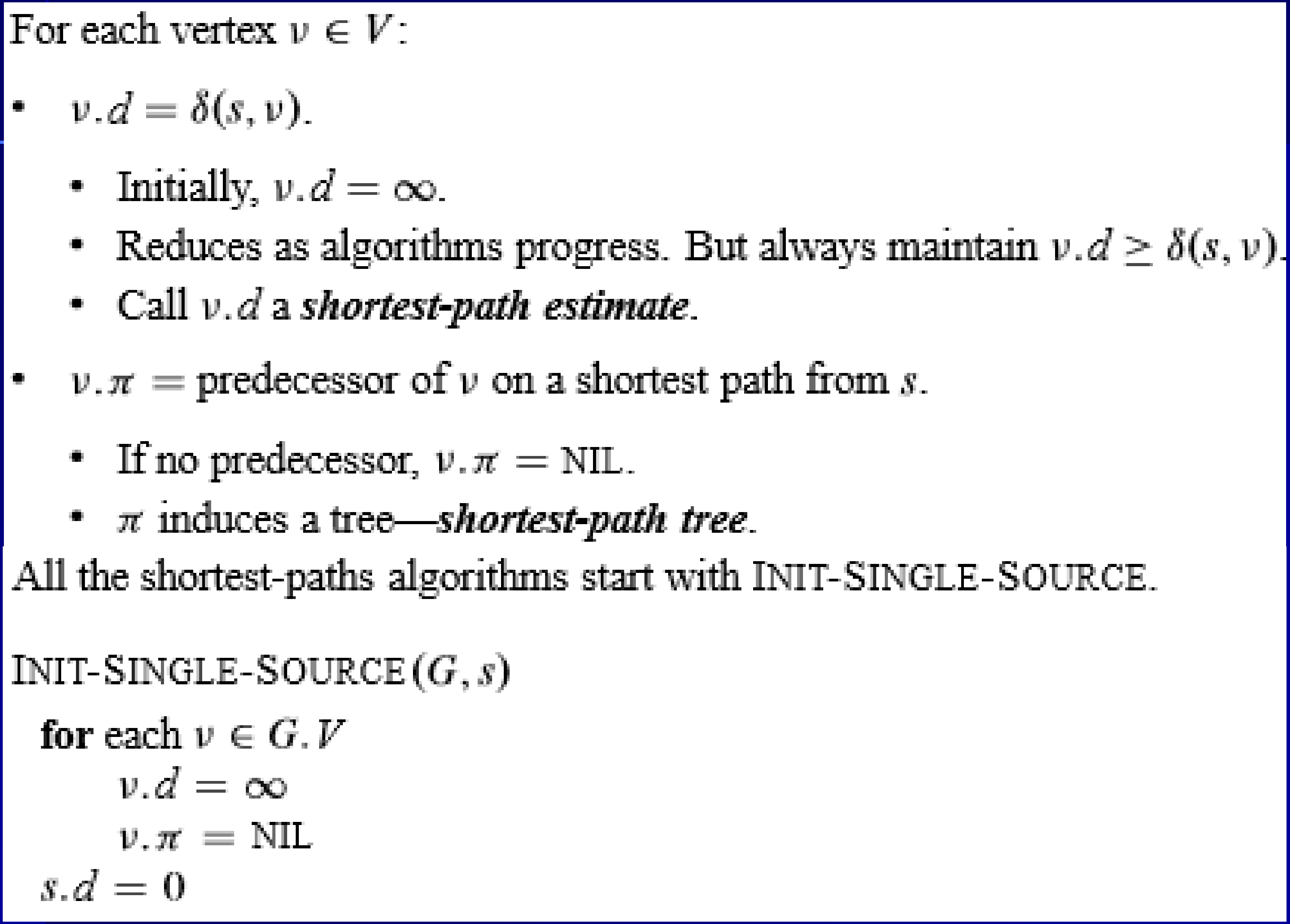
Construct p':

$$\begin{tikzcd} u \arrow[r, squiggly, "p_{ux}"] & x \arrow[r, squiggly, "p'_{xy}"] & y \arrow[r, squiggly, "p_{yv}"] & v \end{tikzcd}$$

$$\begin{aligned} w(p') &= w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) \\ &< w(p_{ux}) + w(p_{xy}) + w(p_{yv}) \\ &= w(p)\,. \end{aligned}$$

Contradicts the assumption that p is a shortest path.

Single-source Shortest-path Algorithm



For each vertex $v \in V$:

- $v.d = \delta(s, v)$.
  - Initially, $v.d = \infty$.
  - Reduces as algorithms progress. But always maintain $v.d \geq \delta(s, v)$.
  - Call $v.d$ a ***shortest-path estimate***.
- $v.\pi =$ predecessor of v on a shortest path from s.
  - If no predecessor, $v.\pi =$ NIL.
  - π induces a tree—***shortest-path tree***.

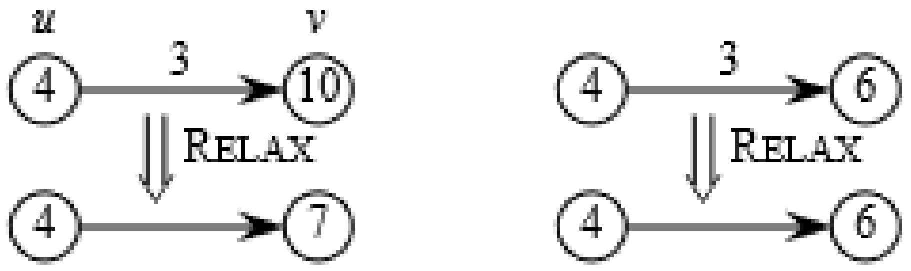
All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

```
INIT-SINGLE-SOURCE$(G, s)$
  for each $v \in G.V$
      $v.d = \infty$
      $v.\pi =$ NIL
  $s.d = 0$
```

Relaxing and Edge (u, v)

The test to see if we can improve the shortest-path estimate for v by going through u and taking (u, v)?

```
RELAX(u, v, w)
  if $v.d > u.d + w(u, v)$
      $v.d = u.d + w(u, v)$
      $v.\pi = u$
```



For all the single-source shortest-paths algorithms we'll look at,

- start by calling INIT-SINGLE-SOURCE,
- then relax edges.

The algorithms differ in the order and how many times they relax each edge.

Some Shortest-paths Properties

Triangle Inequality

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

Proof Weight of shortest path $s \rightsquigarrow v$ is $\le$ weight of any path $s \rightsquigarrow v$. Path $s \rightsquigarrow u \rightarrow v$ is a path $s \rightsquigarrow v$, and if we use a shortest path $s \rightsquigarrow u$, its weight is $\delta(s, u) + w(u, v)$. ■

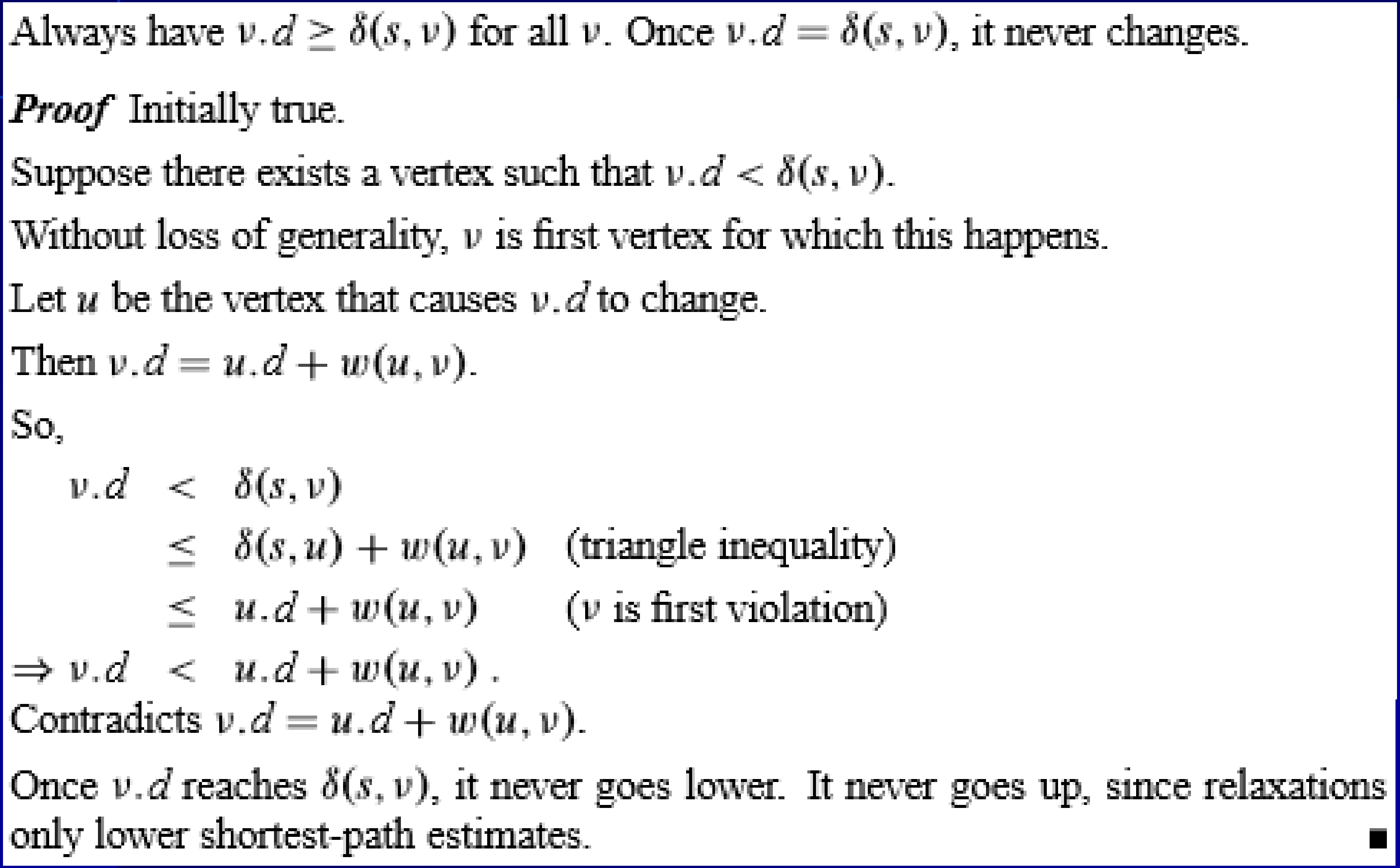
No-path Property

If $\delta(s, v) = \infty$, then $v.d = \infty$ always.

Proof $v.d \ge \delta(s, v) = \infty \Rightarrow v.d = \infty$. ■

Some Shortest-paths Properties

Upper-bound Property



Always have $v.d \geq \delta(s, v)$ for all v. Once $v.d = \delta(s, v)$, it never changes.

Proof Initially true.

Suppose there exists a vertex such that $v.d < \delta(s, v)$.

Without loss of generality, v is first vertex for which this happens.

Let u be the vertex that causes $v.d$ to change.

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

So,

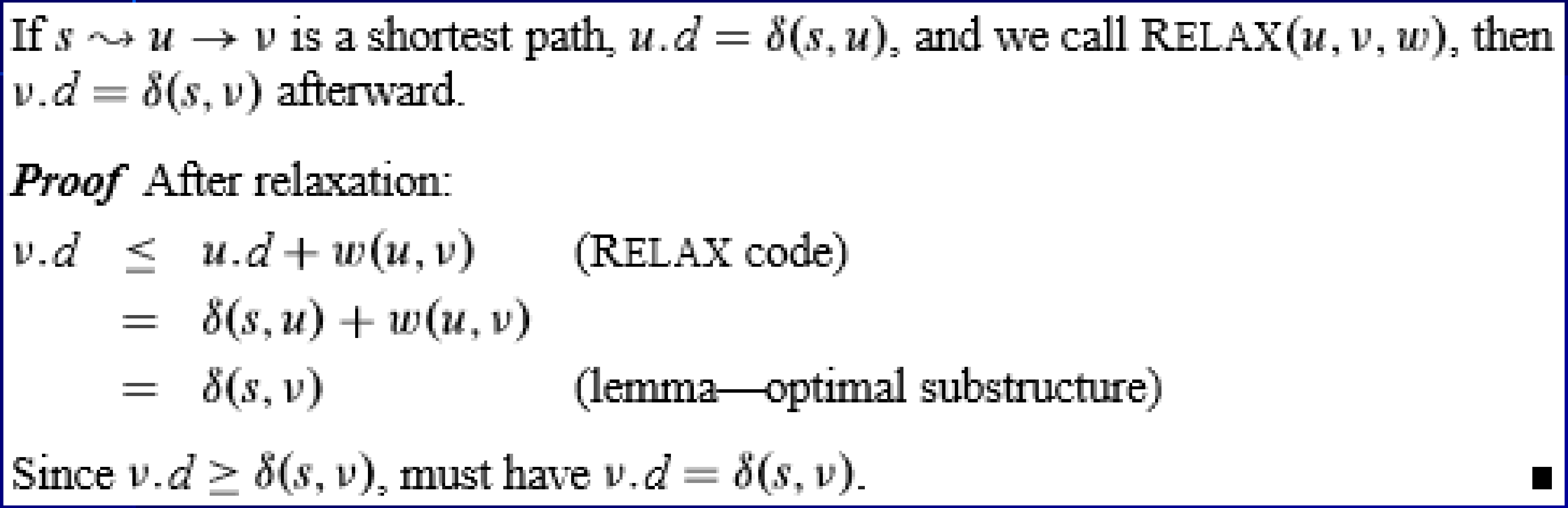
$$\begin{aligned} v.d &< \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) && \text{(triangle inequality)} \\ &\leq u.d + w(u, v) && \text{(v is first violation)} \\ \Rightarrow v.d &< u.d + w(u, v) \, . \end{aligned}$$

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

Once $v.d$ reaches $\delta(s, v)$, it never goes lower. It never goes up, since relaxations only lower shortest-path estimates. ■

Some Shortest-paths Properties

Convergence Property



If $s \leadsto u \to v$ is a shortest path, $u.d = \delta(s, u)$, and we call RELAX(u, v, w), then $v.d = \delta(s, v)$ afterward.

Proof After relaxation:

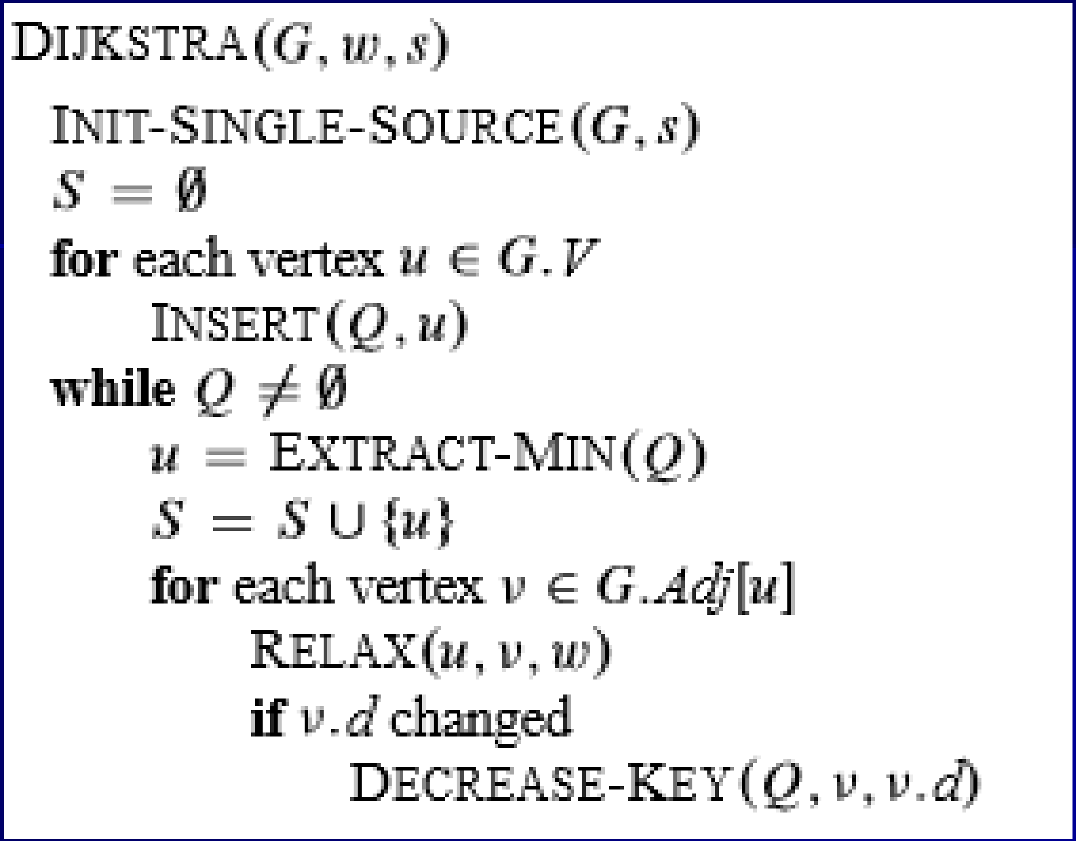
$$\begin{aligned} v.d &\leq u.d + w(u, v) && \text{(RELAX code)} \\ &= \delta(s, u) + w(u, v) \\ &= \delta(s, v) && \text{(lemma—optimal substructure)} \end{aligned}$$

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

Dijkstra’s Algorithm

- This is an algorithm for finding the shortest path in a single-source weighted DAG
- The shortest path from a single node to all destinations is a tree
- Essentially a weighted version of BFS
- No negative weight edges

Dijkstra's Pseudocode



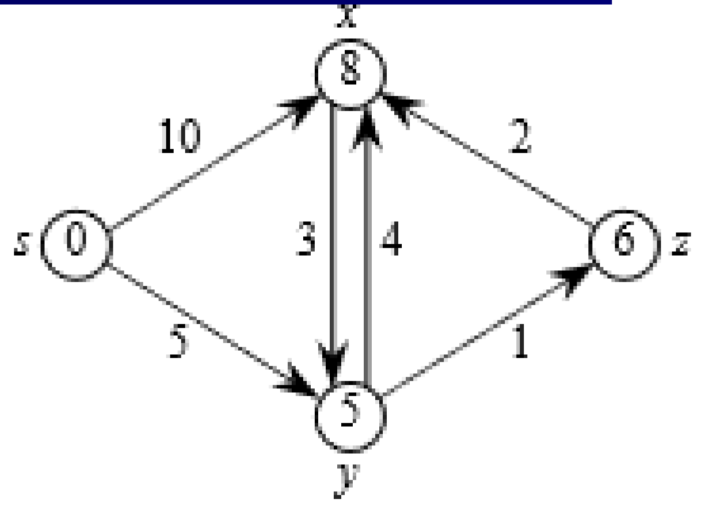
```
DIJKSTRA$(G, w, s)$
  INIT-SINGLE-SOURCE$(G, s)$
  $S = \emptyset$
  for each vertex $u \in G.V$
      INSERT$(Q, u)$
  while $Q \neq \emptyset$
      $u$ = EXTRACT-MIN$(Q)$
      $S = S \cup \{u\}$
      for each vertex $v \in G.Adj[u]$
          RELAX$(u, v, w)$
          if $v.d$ changed
              DECREASE-KEY$(Q, v, v.d)$
```

- Uses two sets of vertices:
  - S = Vertices whose final shortest path weights have been determined
  - Q = Priority queue (Min-priority) = $V - S$
- Algorithm can be viewed as greedy, since it always chooses the lightest ("closest") vertex in $V - S$ to add to S

Dijkstra, Example

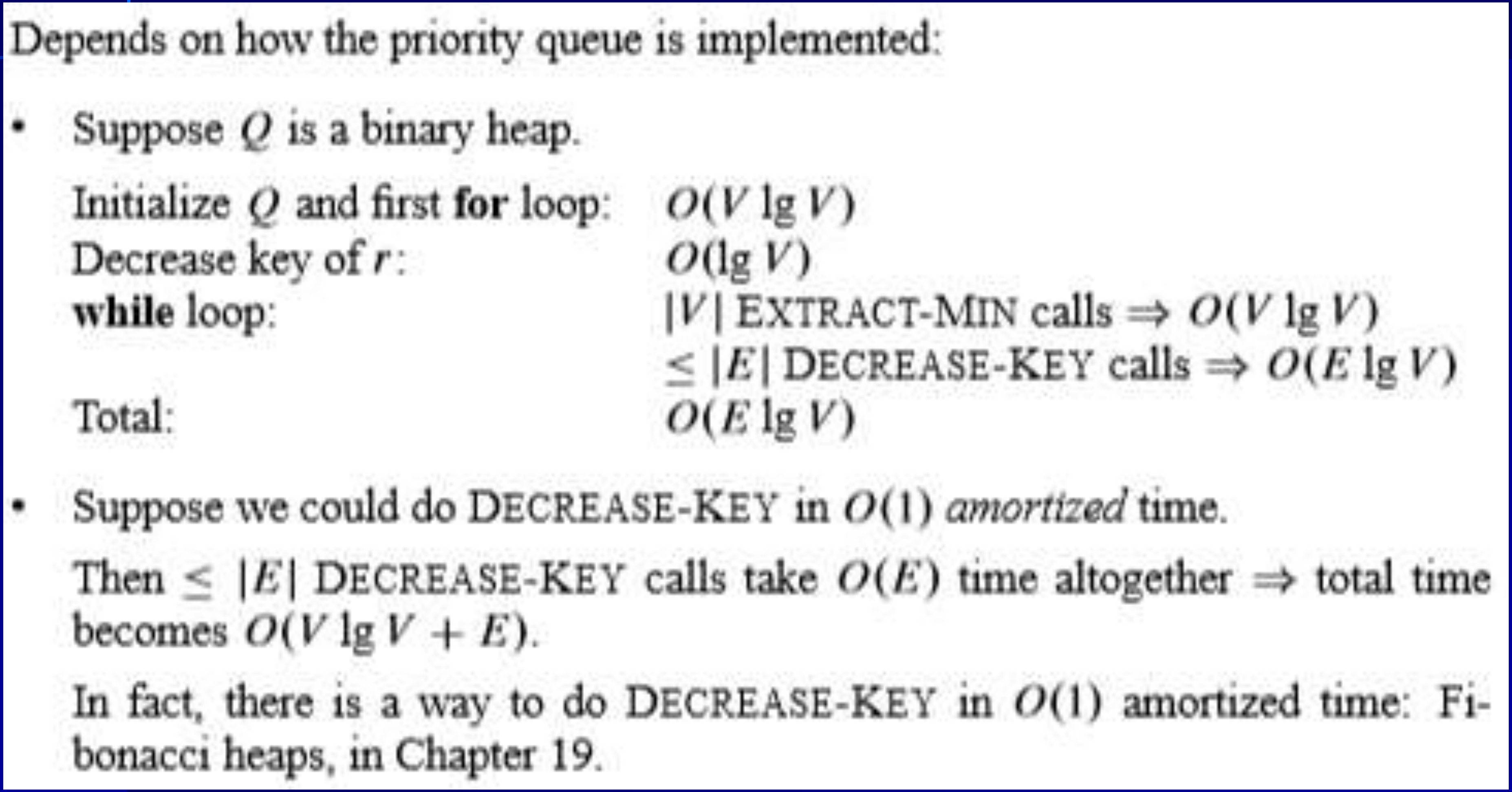
```
DIJKSTRA$(G, w, s)$
  INIT-SINGLE-SOURCE$(G, s)$
  $S = \emptyset$
  for each vertex $u \in G.V$
        INSERT$(Q, u)$
  while $Q \neq \emptyset$
        $u$ = EXTRACT-MIN$(Q)$
        $S = S \cup \{u\}$
        for each vertex $v \in G.Adj[u]$
              RELAX$(u, v, w)$
              if $v.d$ changed
                    DECREASE-KEY$(Q, v, v.d)$
```

Example



Order of adding to S: s, y, z, x.

Analysis of Dijkstra's Alg.



Depends on how the priority queue is implemented:

- Suppose Q is a binary heap.

Initialize Q and first **for** loop:	$O(V \lg V)$
Decrease key of r:	$O(\lg V)$
while loop:	$\lvert V \rvert$ EXTRACT-MIN calls $\Rightarrow O(V \lg V)$
	$\le \lvert E \rvert$ DECREASE-KEY calls $\Rightarrow O(E \lg V)$
Total:	$O(E \lg V)$

- Suppose we could do DECREASE-KEY in $O(1)$ *amortized* time.

  Then $\le \lvert E \rvert$ DECREASE-KEY calls take $O(E)$ time altogether $\Rightarrow$ total time becomes $O(V \lg V + E)$.

  In fact, there is a way to do DECREASE-KEY in $O(1)$ amortized time: Fibonacci heaps, in Chapter 19.

HW

- 22.1-6, 22.1-7
- 22.2-3, 22.2-5
- 22.3-4, 22.3-5, 22.3-8, 22.3-12
- 22.4-3
- 23.1-1, 23.1-4, 23.1-6