# Chap 24 – Single-Source Shortest Paths

24.1 The Bellman-Ford algorithm

24.2 Single-source shortest paths in DAGs

24.3 Dijkstra's algorithm

24.4* Difference constraints and shortest paths

24.5* Proofs of shortest-paths properties

# Shortest Paths

- Shortest path
  - A directed graph $G = (V, E)$ with weight function $w : E \to \mathbf{R}$
  - Weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

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

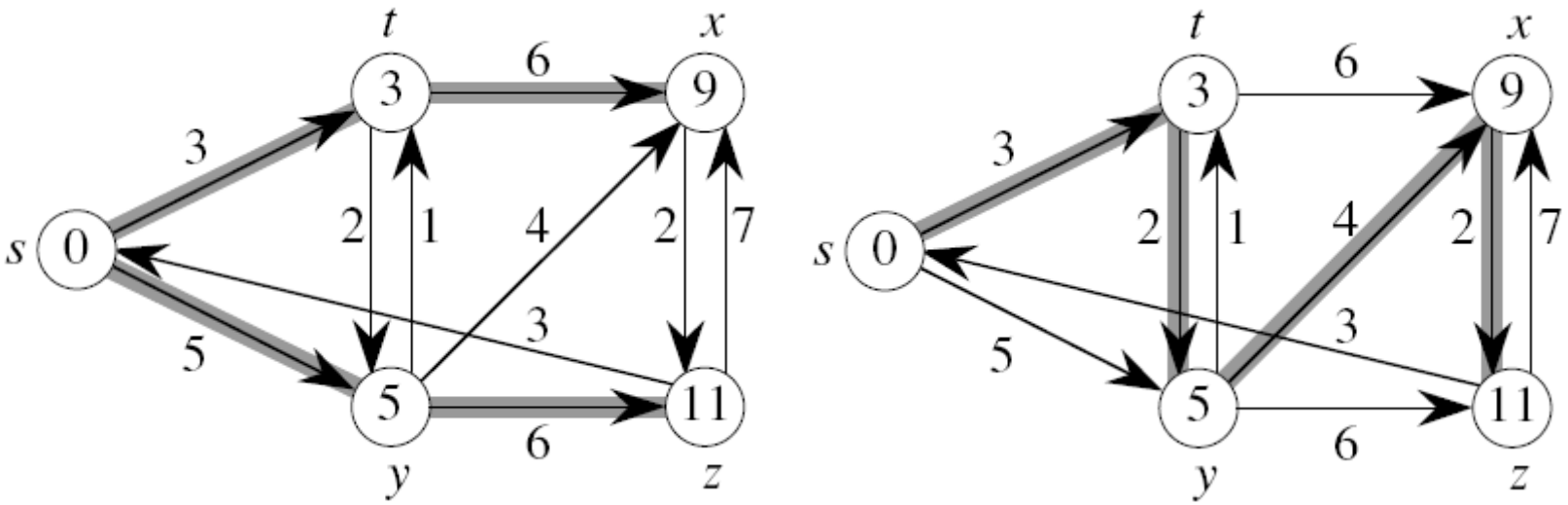
  - Shortest-path weight $u$ *to* $v$

$$\delta(u, v) = \min\{w(p) : u \rightsquigarrow^p v\} \quad \text{if a path } u \rightsquigarrow v \text{ exists}$$
$$= \infty \quad \text{otherwise}$$

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

# Shortest Paths

- Example




  - The shortest path might not be unique
  - The shortest paths $s \rightsquigarrow v$, for all $v$, are organized as a tree.

# Shortest Paths

- Variants
  - Single-source (This chapter)

    Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$
  - Single-destination

    Can be reduced to single-source by reversing the direction of each edge in the graph
  - Single-pair

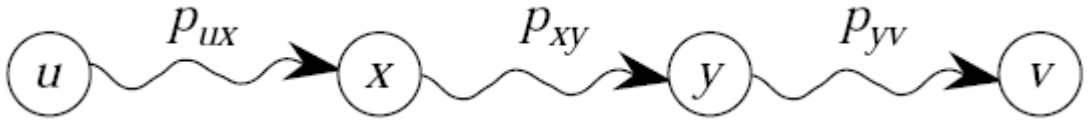
    No known algorithm runs faster than single-source
  - All-pairs (Chap 25)

# Shortest Paths

- Optimal substructure of a shortest path

  **LEMMA** Any subpath of a shortest path is a shortest path.

  *Proof* Cut-and-paste




  Suppose this path $p$ is a shortest path $u \rightsquigarrow v$. Then,

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

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

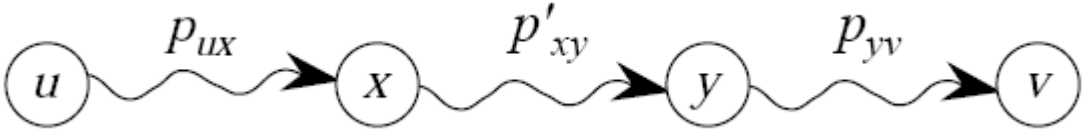
  Then, $w(p'_{xy}) < w(p_{xy})$

# Shortest Paths

- Optimal substructure of a shortest path

  **LEMMA** (Cont'd)

  Construct $p'$




  Then,

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

  A contradiction.

# Shortest Paths

- Negative-weight edges
  - Some algorithms don't allow negative-weight edges
  - Some algorithms allow negative-weight edges, provided that no negative-weight cycles are reachable from the sources.
- Cycles
  - Shortest paths can't contain cycles.
    - Negative-weight: already ruled out
    - Positive-weight: can get a shorter path by omitting the cycle
    - Zero-weight: no reason to use them

# Shortest Paths

- Representing shortest paths

  For each vertex $v \in V$

  - $v.d = \delta(s, v)$
    - Initially, $v.d = \infty$
    - Reduce 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.
    - $\pi$ induces a tree – ***shortest-path tree***.

# Shortest Paths

- Initialization
  - All the shortest-paths algorithms start with:

    INIT-SINGLE-SOURCE$(G, s)$

    **for** each $v \in G.V$

    $v.d = \infty$

    $v.\pi =$ NIL

    $s.d = 0$
- Relaxing an edge $(u, v)$
  - Check if we can improve the shortest path $s \rightsquigarrow v$ found so far by $s \rightsquigarrow u \rightarrow v$

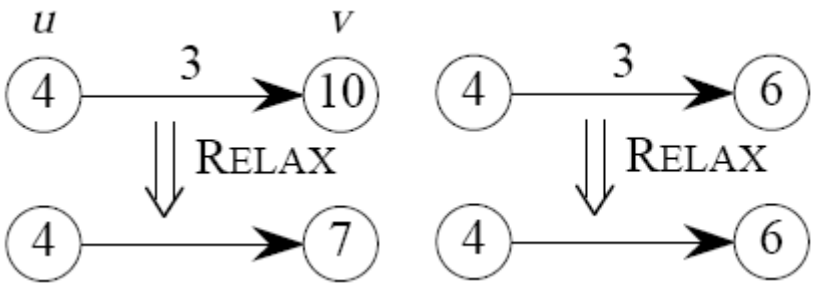
# Shortest Paths

- Relaxing an edge $(u, v)$
  - RELAX$(u, v, w)$

    **if** $v.d > u.d + w(u, v)$

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

    $v.\pi = u$




  - All single-source shortest paths algorithms discussed below
    - start with INIT-SINGLE-SOURCE
    - then relax edges

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

# Shortest Paths

- Shortest-paths properties
  - Triangle inequality

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

    *Proof*

    Weight of shortest path $s \rightsquigarrow v \leq$ weight of any path $s \rightsquigarrow v$

    $\Rightarrow \delta(s, v) \leq$ weight of path $s \rightsquigarrow u \rightarrow v$

    $\Rightarrow \delta(s, v) \leq$ weight of any path $s \rightsquigarrow u + w(u, v)$

    $\Rightarrow \delta(s, v) \leq$ weight of shortest path $s \rightsquigarrow u + w(u, v)$

    $\Rightarrow \delta(s, v) \leq \delta(s, u) + w(u, v)$

# Shortest Paths

- Shortest-paths properties
  - Upper-bound property

    $v.d \geq \delta(s, v)$ for all $v$.

    Once $v.d = \delta(s, v)$, it never changes.

    *Proof*

    Initially, $v.d = \infty \geq \delta(s, v)$ is true.

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

    Without loss of generality, let $v$ be the first vertex for which this happens.

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

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

# Shortest Paths

- Shortest-paths properties
  - Upper-bound property (Cont'd)

    So,

    $v.d < \delta(s,v)$

    $\leq \delta(s,u) + w(u,v)$ $\quad \because$ triangle inequality

    $\leq u.d + w(u,v)$ $\quad \because v$ is first violation

    A contradiction.

    Thus, $v.d \geq \delta(s,v)$

    Moreover, since relaxations only lower estimates, $v.d$ never goes up.

    Thus, once $v.d = \delta(s,v)$, it never changes.

# Shortest Paths

- Shortest-paths properties
  - No-path property

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

    $\because v.d \geq \delta(s, v) = \infty \Rightarrow v.d = \infty$
  - Convergence property

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

    $\because$ After relaxation

    $v.d \leq u.d + w(u, v)$ $\quad \because$ RELAX code

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

    $= \delta(s, v)$ $\quad \because$ optimal substructure

    Since $v.d \geq \delta(s, v)$, it follows that $v.d = \delta(s, v)$.

# Shortest Paths

- Shortest-paths properties
  - Path relaxation property

    Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path $s = v_0 \rightsquigarrow v_k$

    If we relax, in order, $(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$,

    even intermixed with other relaxations, then

    $v_k.d = \delta(s, v_k)$

    *Proof by induction*

    $v_i.d = \delta(s, v_i)$ after $(v_{i-1}, v_i)$ is relaxed

    Basis: $i = 0$.

    Initially, no edge is relaxed.

    $v_0.d = s.d = 0 = \delta(s, s) = \delta(s, v_0)$

# Shortest Paths

- Shortest-paths properties
  - Path relaxation property (Cont'd)

    Inductive step

    Inductive hypothesis $v_{i-1}.d = \delta(s, v_{i-1})$ $\cdots (1)$

    Relax $(v_{i-1}, v_i)$

    By optimal substructure:

    $s = v_0 \rightsquigarrow v_{v-1} \rightarrow v_i$ is a shortest path $\cdots (2)$

    From $(1), (2)$, and convergence property, we have

    $v_i.d = \delta(s, v_i)$

    Moreover, by upper-bound property, $v_i.d = \delta(s, v_i)$ never changes.

# 24.1 The Bellman-Ford algorithm

- The Bellman-Ford algorithm
  - Allow negative-weight edges
  - Return TRUE, if no negative-weight cycles reachable from $s$

    Return FALSE, otherwise
  - BELLMAN-FORD$(G, w, s)$

    INIT-SINGLE-SOURCE$(G, s)$ $\cdots \Theta(V)$ time

    **for** $i = 1$ **to** $|G.V| - 1$ $\cdots \Theta(VE)$ time

    **for** each edge $(u, v) \in G.E$

    RELAX$(u, v, w)$

    **for** each edge $(u, v) \in G.E$ $\cdots O(E)$ time

    **if** $v.d > u.d + w(u, v)$ **return** FALSE

    **return** TRUE

# 24.1 The Bellman-Ford algorithm

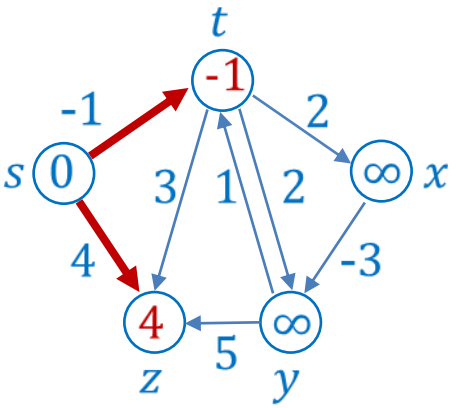
- The Bellman-Ford algorithm: An example

  Arbitrary relaxation order

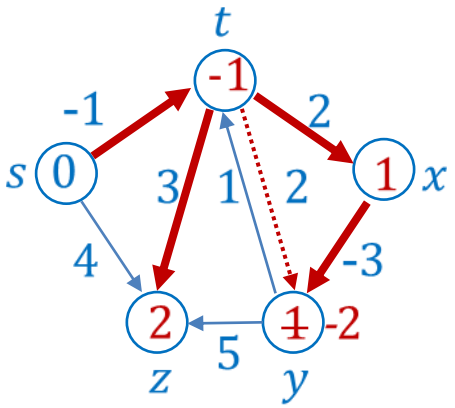
  $(t,x),(t,y),(t,z),(x,y),(y,t),(y,z),(s,t),(s,z)$

  But guaranteed to converge after $|V|-1$ passes, assuming no negative-weight cycles.

pass 1




pass 2 (converged)



# 24.1 The Bellman-Ford algorithm

- The Bellman-Ford algorithm
  - **LEMMA**
    If there are no negative-weight cycles reachable from $s$, then, after the $|V| - 1$ iterations of relaxation, $v.d = \delta(s, v)$ for all $v \in V$.
    *Proof*
    Case 1: $v$ is reachable from $s$
    Let $p = \langle s = v_0, v_1, \ldots, v_k = v \rangle$ be a shortest path $s \rightsquigarrow v$
    Since $p$ is acyclic, it has $\leq |V| - 1$ edges.
    So, $k \leq |V| - 1$

# 24.1 The Bellman-Ford algorithm

- The Bellman-Ford algorithm
  - **LEMMA** (Cont'd)

    Since there are $|V| - 1$ $(\geq k)$ iterations and each iteration relaxes all edges, we have, in particular,
    - first iteration relaxes $(s = v_0, v_1)$
    - second iteration relaxes $(v_1, v_2)$

      ⋮
    - $k^{\text{th}}$ iteration relaxes $(v_{k-1}, v_k = v)$

    Thus, by path relaxation property, $v.d = \delta(s, v)$.

    Case 2: $v$ isn't reachable from $s$

    By no-path property, $v.d = \delta(s, v) = \infty$.

# 24.1 The Bellman-Ford algorithm

- The Bellman-Ford algorithm
  - **THEOREM**

    If there are no negative-weight cycles reachable from $s$, the algorithm returns TRUE; otherwise, it returns FALSE.

    *Proof*

    Suppose no negative-weight cycle reachable from $s$

    At termination, for all $(u, v) \in E$

    $$\begin{aligned} v.d &= \delta(s, v) && \because \text{lemma} \\ &\leq \delta(s, u) + w(u, v) && \because \text{triangle inequality} \\ &= u.d + w(u, v) && \because \text{lemma} \end{aligned}$$

    So, BELLMAN-FORD returns TRUE.

# 24.1 The Bellman-Ford algorithm

- The Bellman-Ford algorithm
  - **THEOREM** (Cont'd)

    Now suppose there exists negative-weight cycle

    $c = \langle v_0, v_1, \ldots, v_k = v_0 \rangle$

    reachable from $s$.

    Suppose that BELLMAN-FORD returns TRUE. Then,

    $\forall (u, v) \in E, v.d \leq u.d + w(u, v)$

    $\Rightarrow \forall 1 \leq i \leq k, v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$

    $\Rightarrow \sum_{i=1}^{k} v_i.d \leq \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i))$

    $\Rightarrow 0 = v_k.d - v_0.d \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$ A contradiction!

# 24.2 Single-source shortest paths in DAGs

- Shortest paths in directed acyclic graphs
  - Shortest paths are always well-defined in a DAG, since we are guaranteed no negative-weight cycles.
  - DAG-SHORTEST-PATHS$(G, w, s)$

    topologically sort the vertices $\cdots \Theta(V + E)$ time, Sec. 22.4

    INIT-SINGLE-SOURCE$(G, s)$ $\cdots \Theta(V)$ time

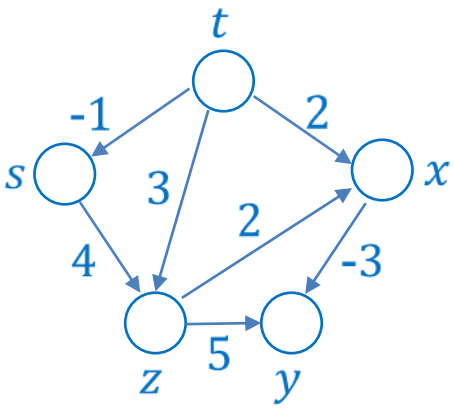
    **for** each vertex $u$, taken in topologically sorted order

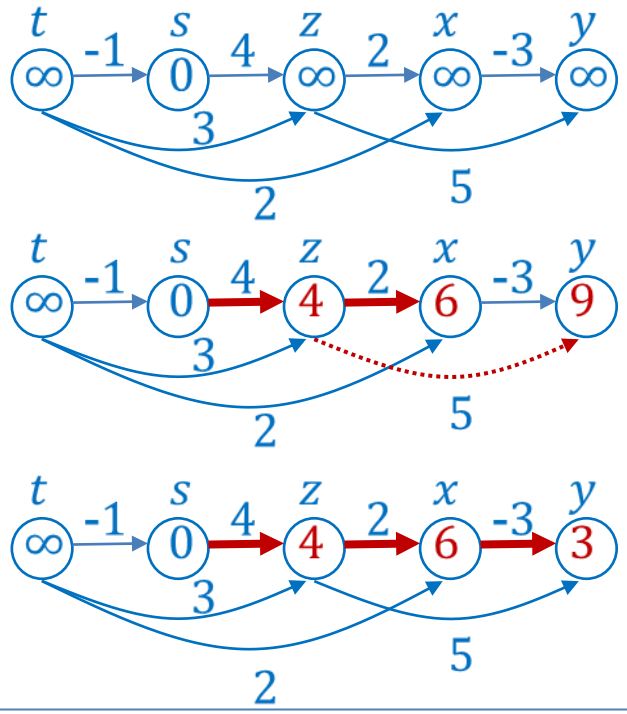
        **for** each vertex $v \in G.Adj[u]$ $\cdots \Theta(E)$ time

            RELAX$(u, v, w)$

# 24.2 Single-source shortest paths in DAGs

- Shortest paths in directed acyclic graphs
  - A topological sort of a dag is an ordering of its vertices along a horizontal line so that all directed edges go from left to right.
  - Example



# 24.2 Single-source shortest paths in DAGs

- Shortest paths in directed acyclic graphs
  - **THEOREM** The algorithm is correct.

    *Proof*

    Vertices are processed in topological order

    ⇒ Edges of *any* path must be relaxed in order of appearance in the path

    ⇒ Edges on *any shortest* path are relaxed in order

    ⇒ By path-relaxation property, the algorithm is correct

# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm
  - No negative-weight *edges*.
  - DIJKSTRA$(G, w, s)$

    INIT-SINGLE-SOURCE$(G, s)$ are determined

    $S = \emptyset$ // $S$ = vertices whose shortest-path weights

    $Q = G.V$ // insert all vertices into $Q$; $Q = V - S$

    **while** $Q \neq \emptyset$ ∵ $|V|$ INSERTS

      $u =$ EXTRACT-MIN$(Q)$ ⋯ $|V|$ EXTRACT-MINS

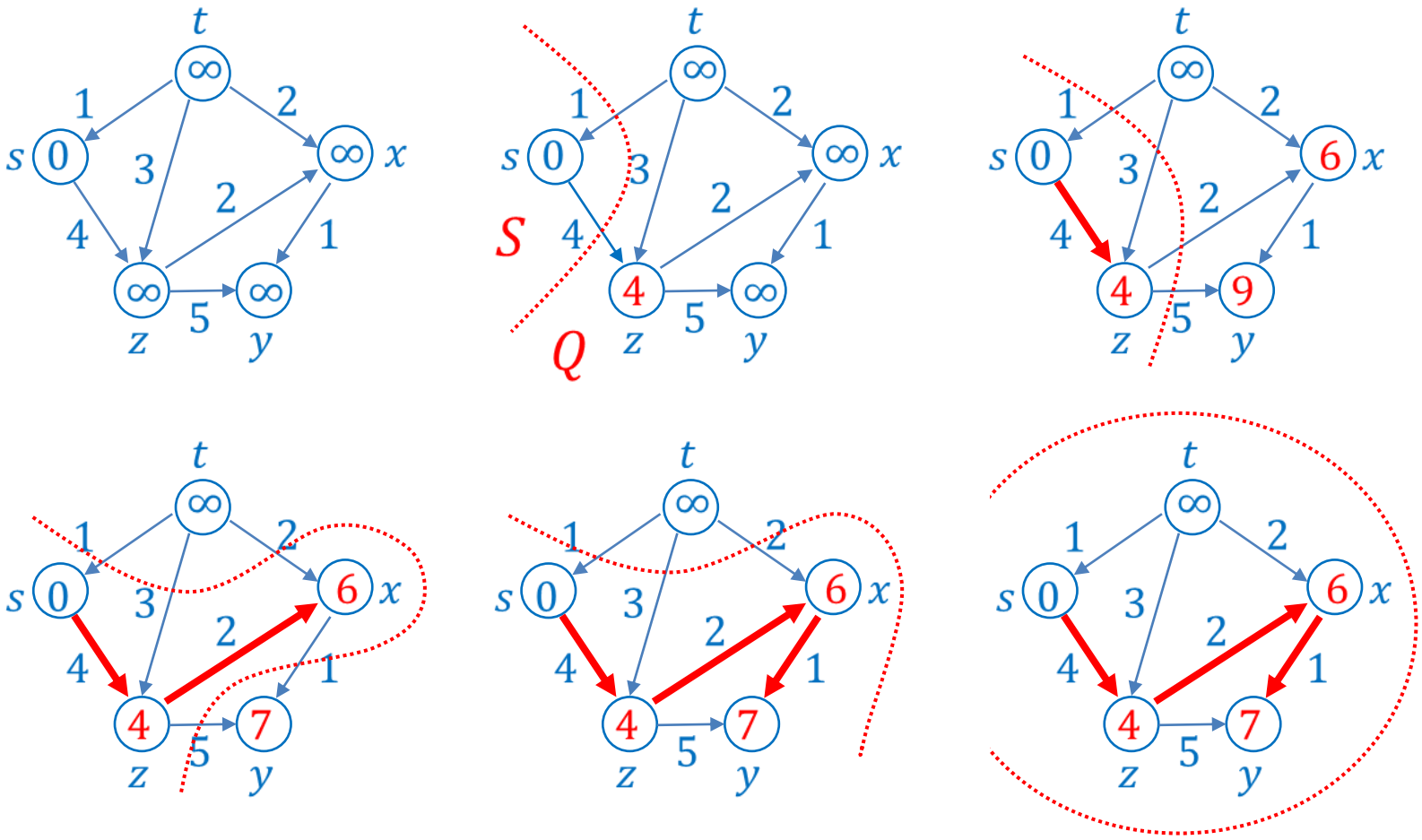
      $S = S \cup \{u\}$

      **for** each vertex $v \in G.Adj[u]$

        RELAX$(u, v, w)$ ⋯ ≤ $|E|$ DECREASE-KEY

# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm: An example



# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm
  - **THEOREM** Dijkstra's greedy algorithm is correct.

    *Proof*

    Loop invariant: At the start of each iteration of the **while** loop, $v.d = \delta(s, v)$ for all $v \in S$.

    **Initialization:** Initially, $S = \emptyset$, so trivially true.

    **Termination**

    At end, $Q = \emptyset \Rightarrow S = V \Rightarrow v.d = \delta(s, v)$ for all $v \in V$

    **Maintenance**

    Need to show that $u.d = \delta(s, u)$ when $u$ is added to $S$ in each iteration.

# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm
  - **THEOREM** (Cont'd)

    Suppose there exists $u$ such that $u.d \neq \delta(s,u)$.

    Without loss of generality, let $u$ be the first vertex for which $u.d \neq \delta(s,u)$ when $u$ is added to $S$.

    Observations:
    - $u \neq s$, since $s.d = \delta(s,s) = 0$.
    - When $u$ is added to $S$, $s \in S$. ($\because$ $s$ is the 1st vertex that is moved from $Q$ to $S$.)
    - There must be some path $s \rightsquigarrow u$, since otherwise $u.d = \delta(s,u) = \infty$ by no-path property.

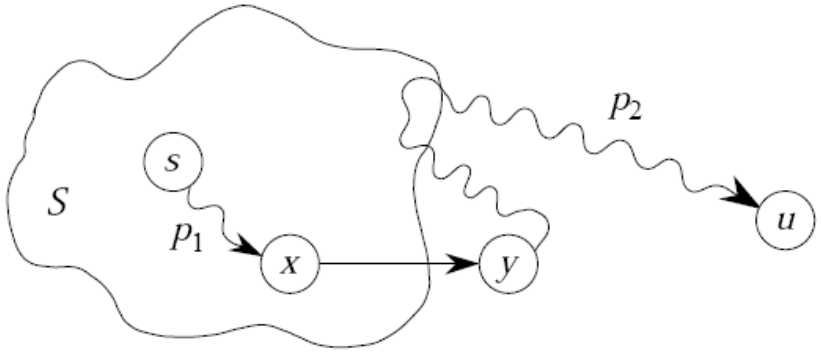
# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm
  - **THEOREM** (Cont'd)

    So, ∃ a path $s \rightsquigarrow u \Rightarrow$ ∃ a shortest path $s \overset{p}{\rightsquigarrow} u$.

    Just before $u$ is added to $S$, path $p$ connects a vertex in $S$ (i.e., $s$) to a vertex in $V - S$ (i.e., $u$).

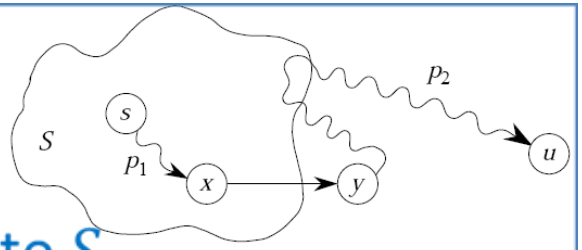
    Let $y$ be first vertex along $p$ that's in $V - S$, and let $x \in S$ be $y$'s predecessor.




$$s \overset{p}{\rightsquigarrow} u = s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$$

# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm
  - **THEOREM** (Cont'd)




**CLAIM** $y.d = \delta(s, y)$ when $u$ is added to $S$

*Proof*

$s \rightsquigarrow^{p} u$ is a shortest path

$\Rightarrow s \rightsquigarrow^{p_1} x \rightarrow y$ is a shortest path $\because$ optimal substructure

$x \in S$ and $u$ is the first vertex such that $u.d \neq \delta(s, u)$ when $u$ is added to $S$

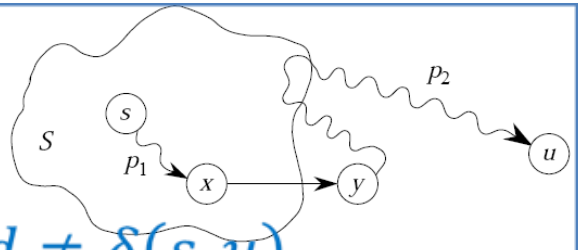
$\Rightarrow x.d = \delta(s, x)$ when $x$ is added to $S$

Relaxed $(x, y)$ at that time

$\Rightarrow y.d = \delta(s, y)$ by convergence property ■

# 24.3 Dijkstra's algorithm

- Dijkstra's greedy algorithm
  - **THEOREM** (Cont'd)




We can now get a contradiction to $u.d \neq \delta(s,u)$.

$$y.d = \delta(s,y)$$
$$\leq \delta(s,u) \quad \because s \rightsquigarrow y \rightsquigarrow u \text{ is shortest and all edges} \geq 0$$
$$\leq u.d \quad \because \text{upper-bound property}$$
$$\leq y.d \quad \because \text{both } y \text{ and } u \in Q \text{ when we chose } u$$

Therefore,

$$y.d = \delta(s,y) = \delta(s,u) = u.d$$

A contradiction.

# 24.3 Dijkstra's algorithm

- Analysis of Dijkstra's greedy algorithm
  - Dijkstra's algorithm looks a lot like Prim's algorithm.
  - Dijkstra's algorithm has the same time complexity as Prim's algorithm.
  - Like Prim's algorithm, depend on implementation of priority queue.
    - With binary heap, $O(E \lg V)$
    - With Fibonacci heap, $O(E + V \lg V)$