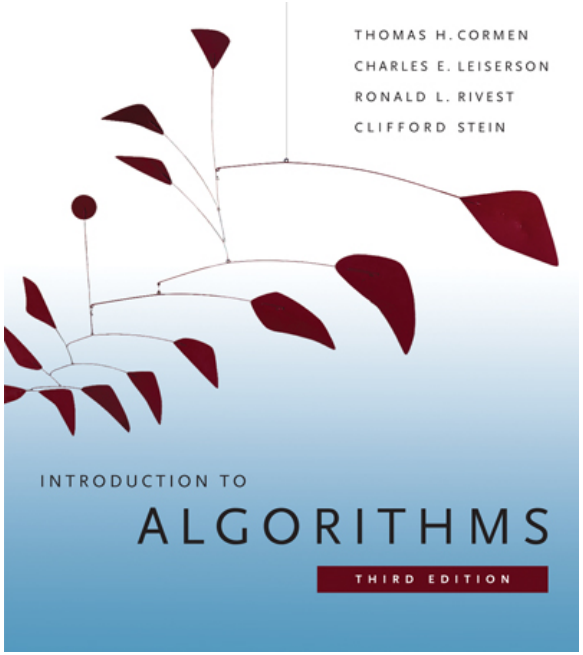
6.006- *Introduction to Algorithms*



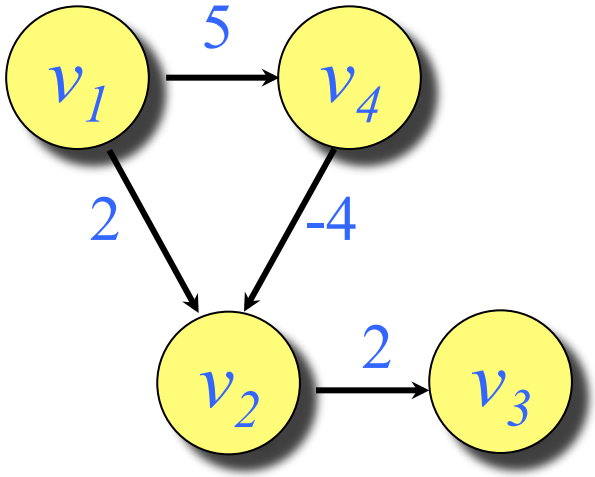
Lecture 16

Prof. Patrick Jaillet

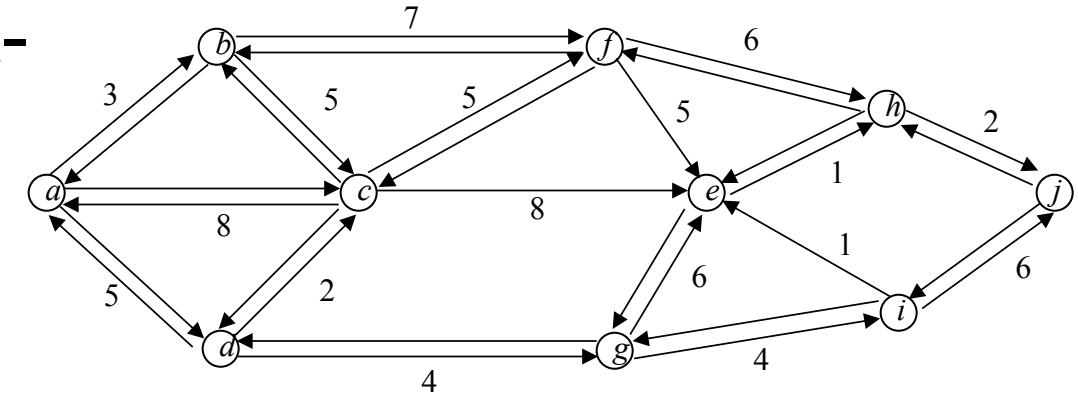
Lecture overview

Shortest paths III

– Bellman-Ford on a DAG (CLRS 24.2)

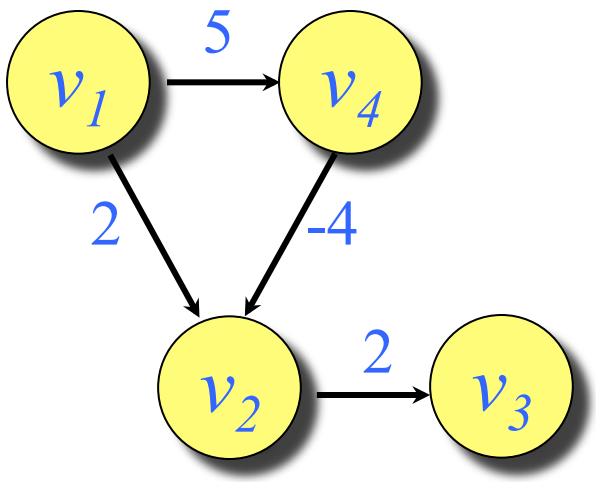


– Dijkstra algorithm for the case with non-negative weights (CLRS 24.3)



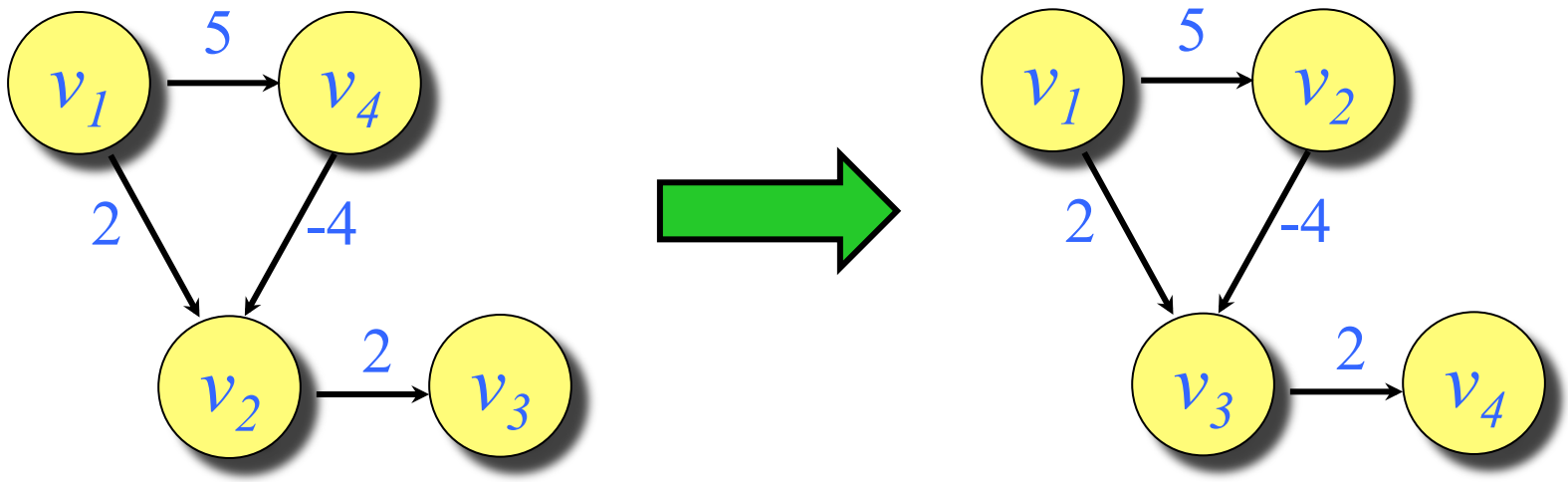
This graph has a special structure: DAG. How to use it within Bellman-Ford?

$E=\{(v_1,v_2); (v_1,v_4); (v_2,v_3); (v_4,v_2)\}$



... from lecture 15: think about it for next time ...

... first use topological sorting ...

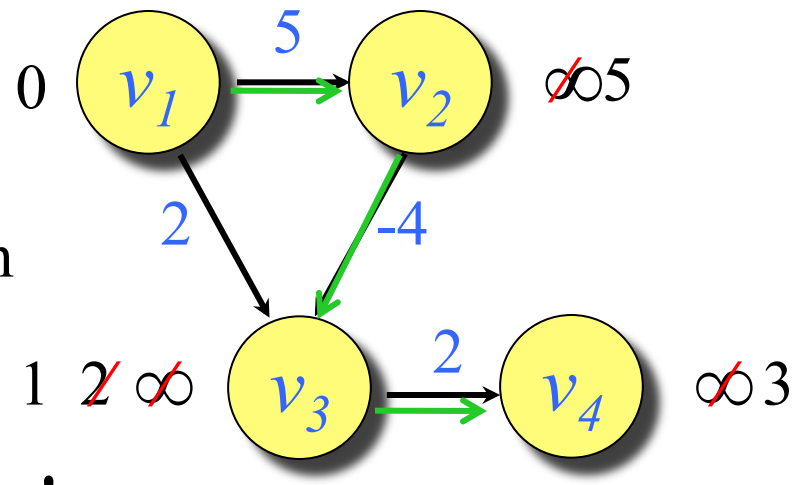


E={(v_1,v_2); (v_1,v_4); (v_2,v_3); (v_4,v_2)}

E={(v_1,v_2); (v_1,v_3); (v_2,v_3); (v_3,v_4)}

... Bellman-Ford ...

$E=\{(v_1,v_2); (v_1,v_3); (v_2,v_3); (v_3,v_4)\}$



end of first iteration

and we are done !

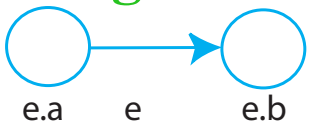
the shortest paths from v_1

Bellman-Ford algorithm on DAG

topologically sort the vertices V

($f: V \to \{1, 2, \ldots, |V|\}$ *such that* $(u,v) \in E \Rightarrow f(u) < f(v)$)

arrange E in lexicographical order of *(f(e.a), f(e.b))* $O(n+m)$



$d[s] \leftarrow 0$; $\pi[s] \leftarrow$ s
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ nil

initialization $O(n)$

do for each edge $(u, v) \in E$
  do if $d[v] > d[u] + w(u, v)$
    then $d[v] \leftarrow d[u] + w(u, v)$
      $\pi[v] \leftarrow$ u

one iteration of relaxation steps $O(m)$

for each edge $(u, v) \in E$
  do if $d[v] > d[u] + w(u, v)$
    then report a negative cycle

final steps not needed

... why does this work? ...

- there are no cycles in a dag => even with negative-weight edges, there are no negative-weight cycles ...
- topological ordering implies a linear ordering of the vertices; every path in a dag is a subsequence of topologically sorted vertex order; processing vertices in that order, an edge can’t be relaxed more than once ...

The case of non-negative weights

Problem: Given a directed graph $G = (V, E)$ with edge-weight function $w : E \rightarrow \mathbb{R}^+$, and a node s, find the shortest-path weight $\delta(s, v)$ (and a corresponding shortest path) from s to each v in V.

Greedy iterative approach

1. maintain a set S of vertices whose shortest-path distances from s are known.
2. at each step add to S the vertex $v \in V - S$ whose distance estimate from s is minimal.
3. update distance estimates of vertices adjacent to v.

Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$

initialization

while $Q \neq \varnothing$ (Q min-priority queue maintaining $V - S$)
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$

relaxation steps

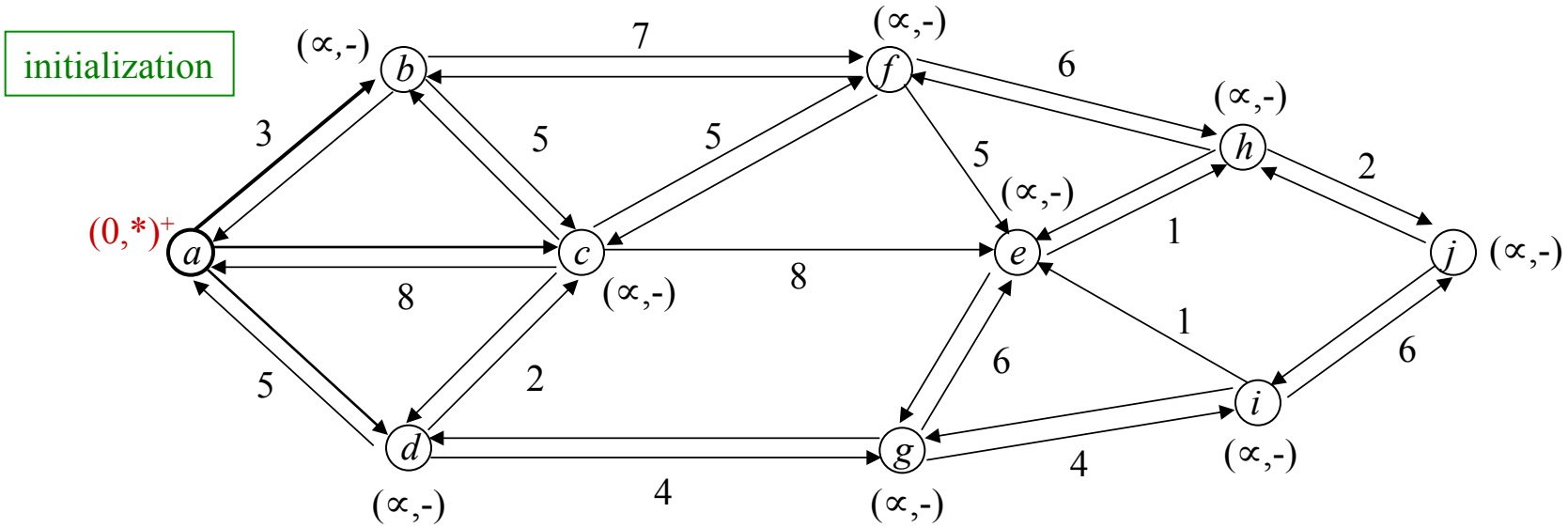
(Implicit DECREASE-KEY)

[Digression - Min- Priority Queue (see Lect 9)]

This is an ***abstract datatype*** implementing a set S of elements, each associated with a key, supporting the following operations:

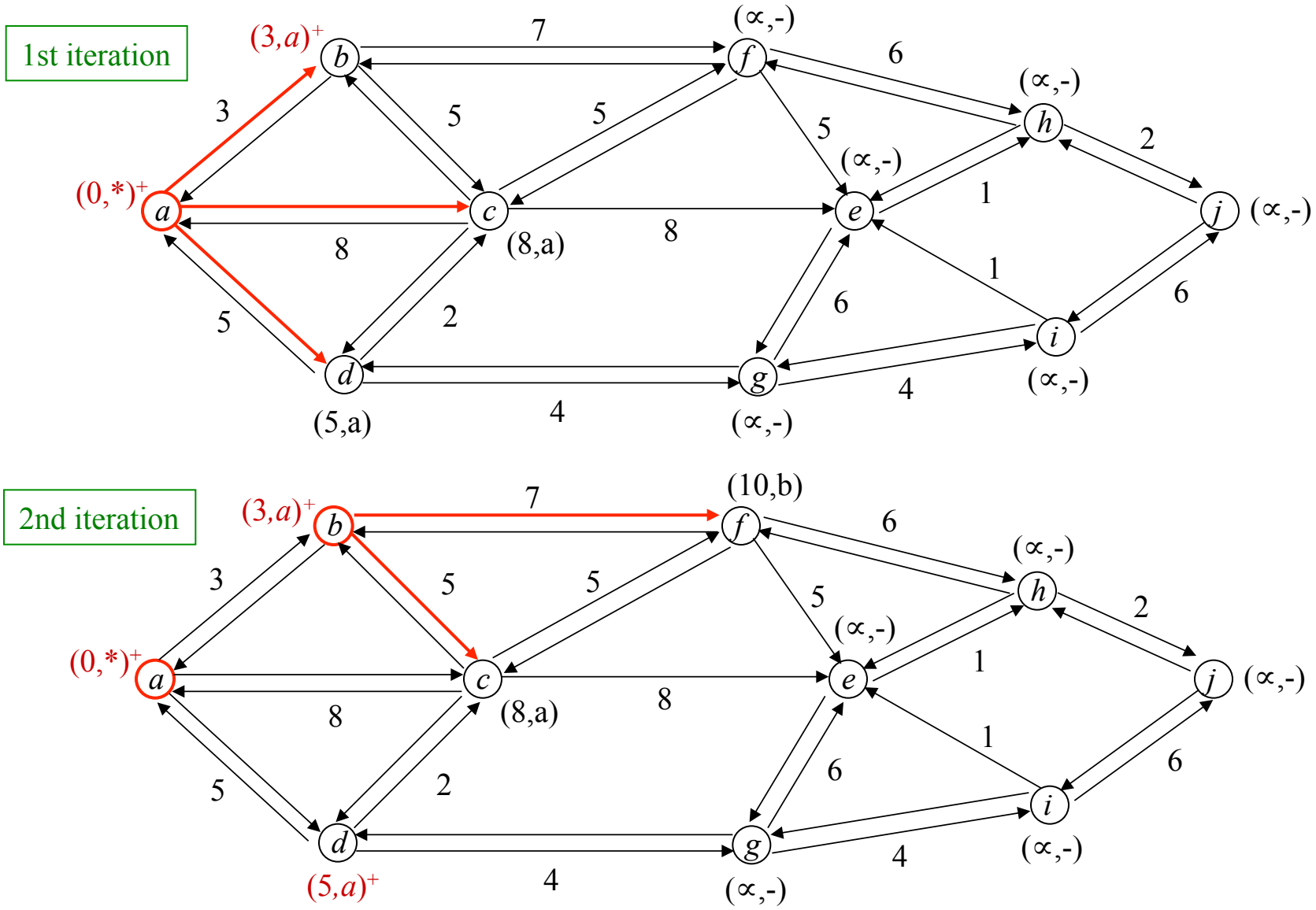
insert(S, x) :	insert element x into set S
min(S) :	return element of S with smallest key
extract_min(S) :	return element of S with smallest key and remove it from S
decrease_key(S, x, k) :	change the key-value of element x to the value k (assumed to not larger than current value)

Dijkstra: Example



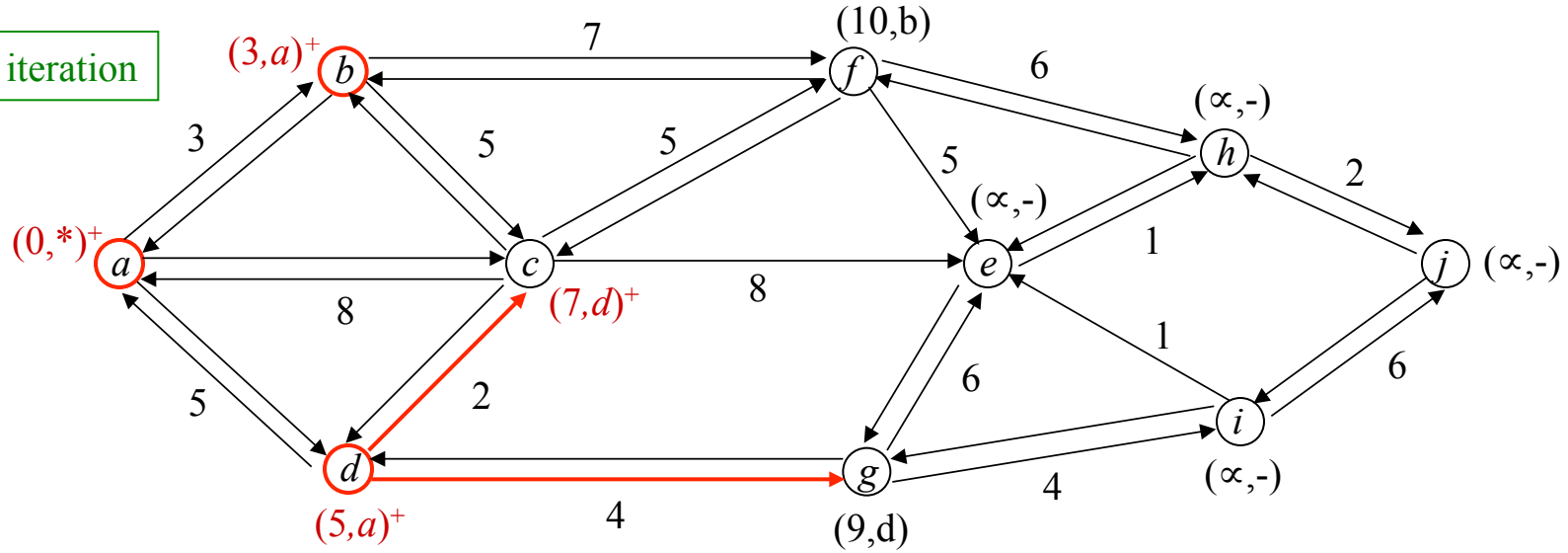
$Q = V$, a = EXTRACT-MIN(Q)

Dijkstra: Example

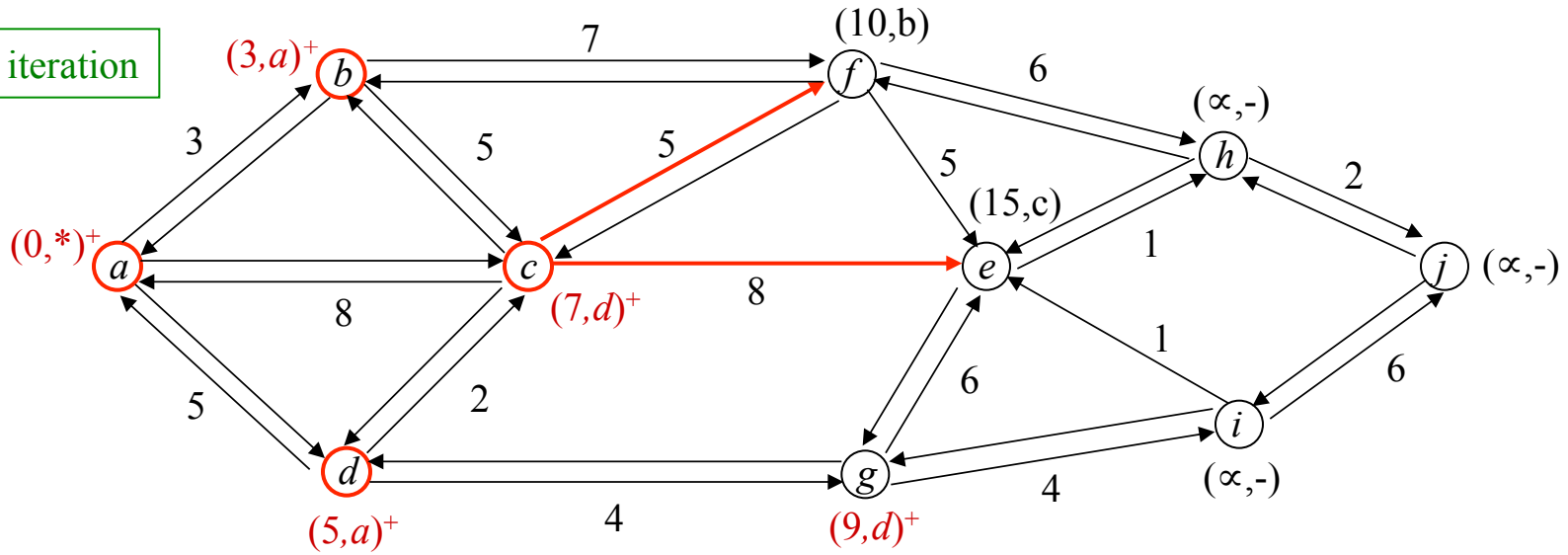


Dijkstra: Example

3rd iteration

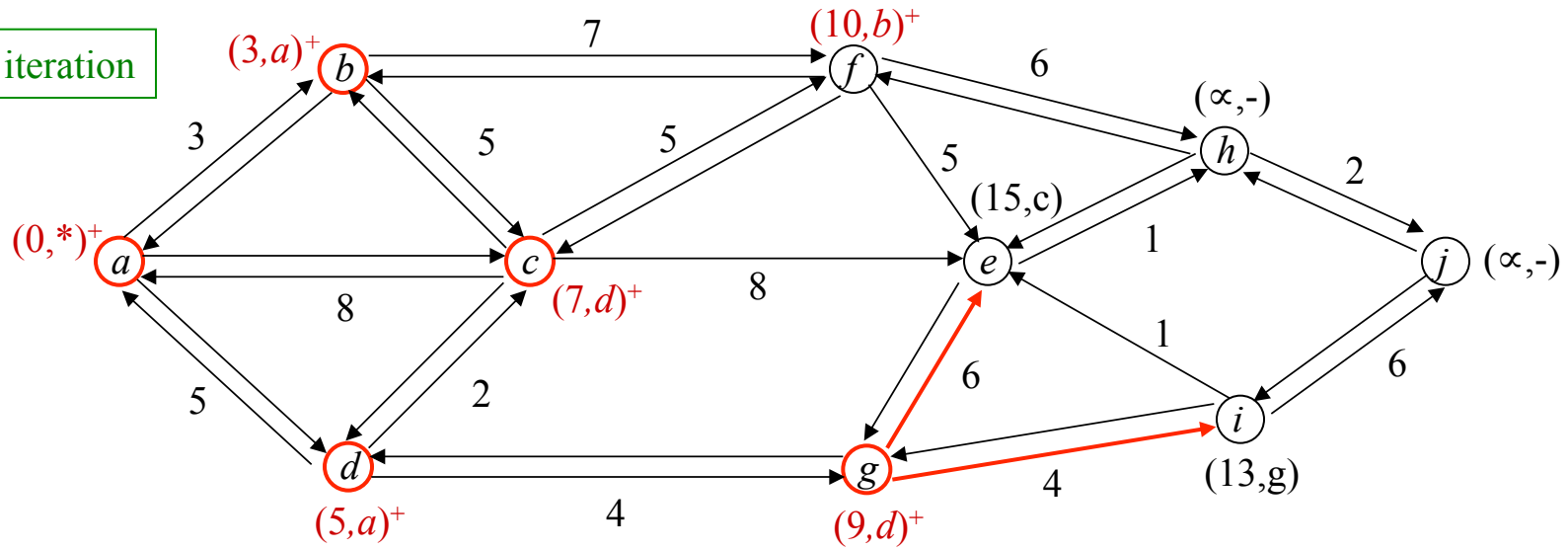


4th iteration

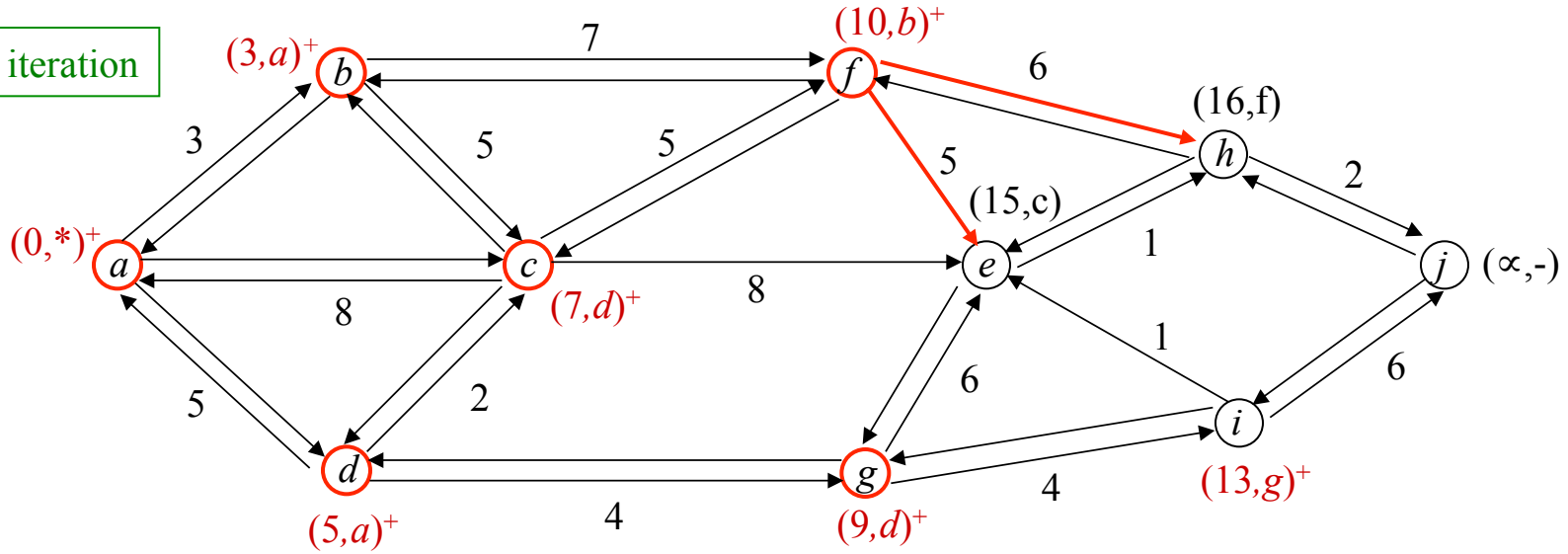


Dijkstra: Example

5th iteration

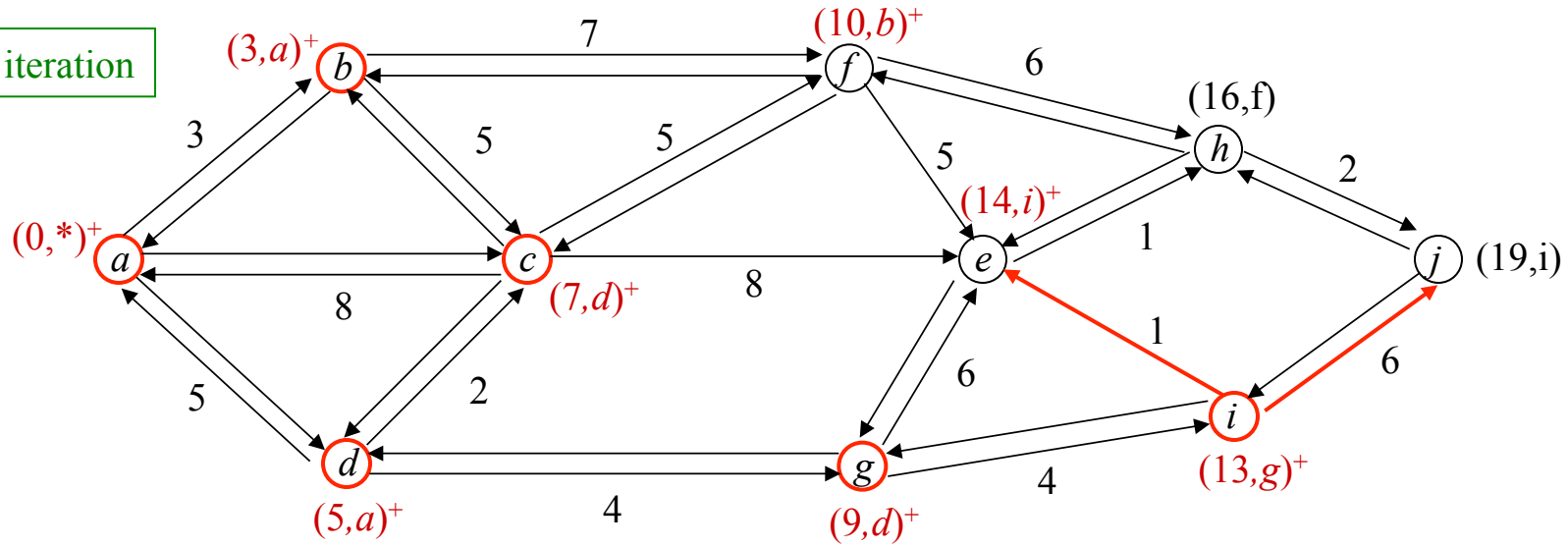


6th iteration

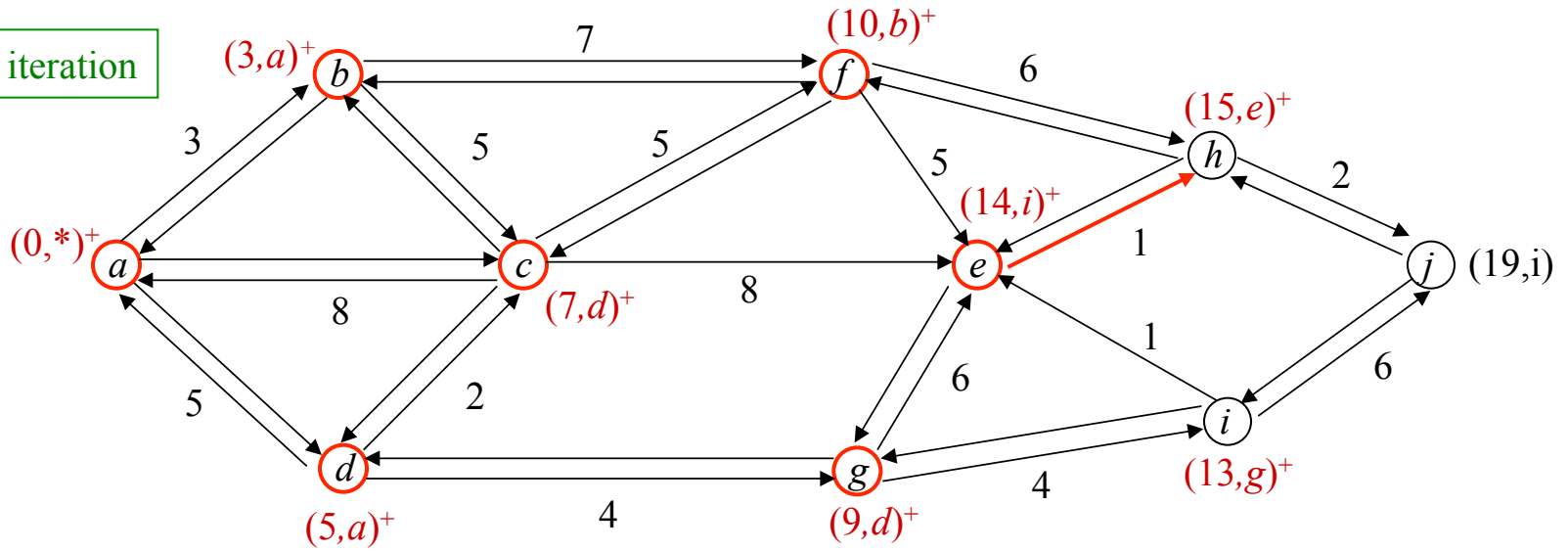


Dijkstra: Example

7th iteration

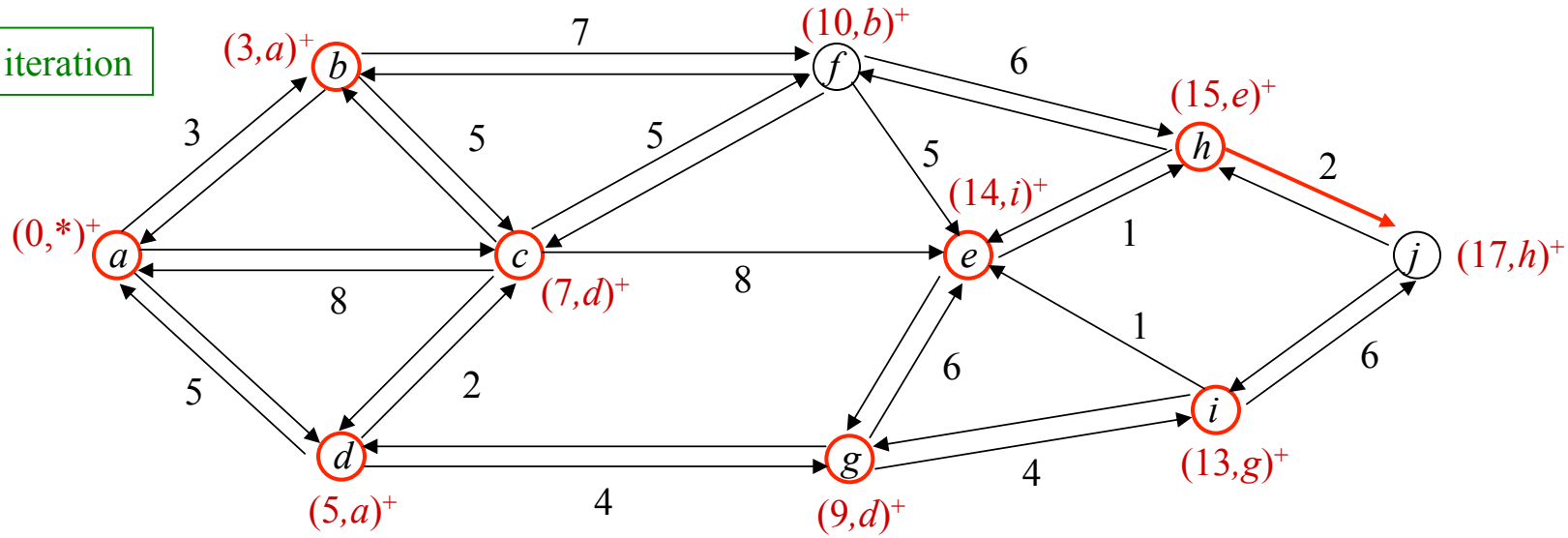


8th iteration

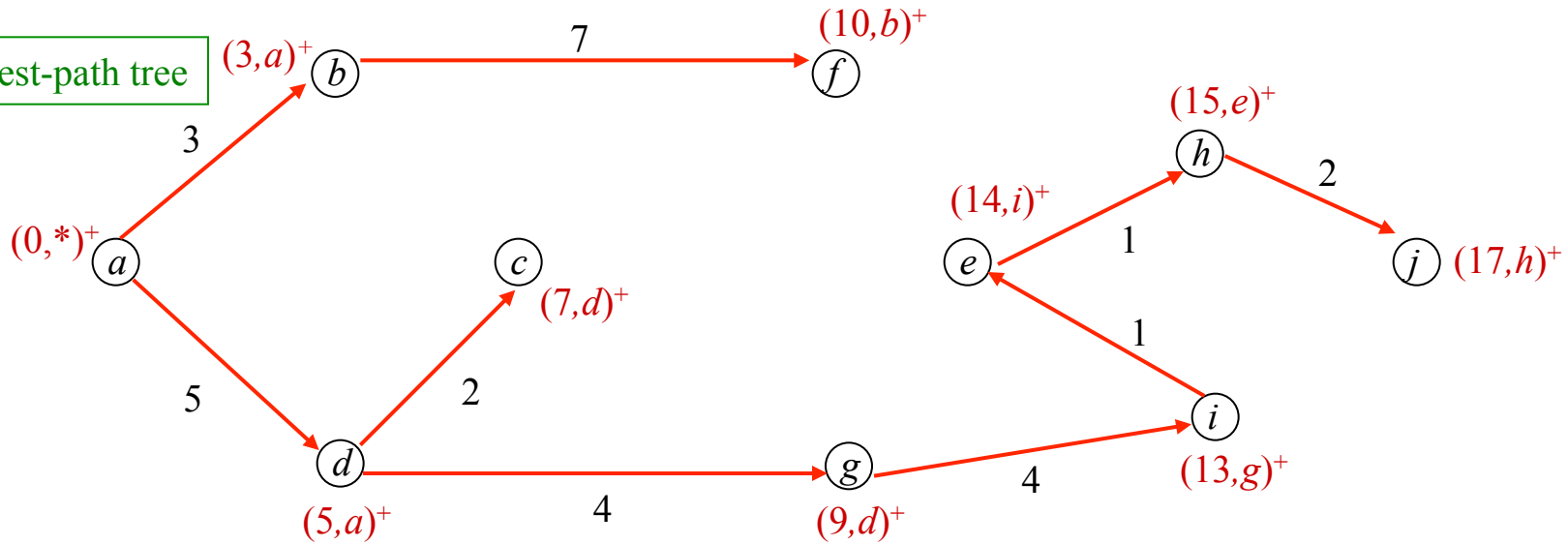


Dijkstra: Example

9th iteration



Shortest-path tree

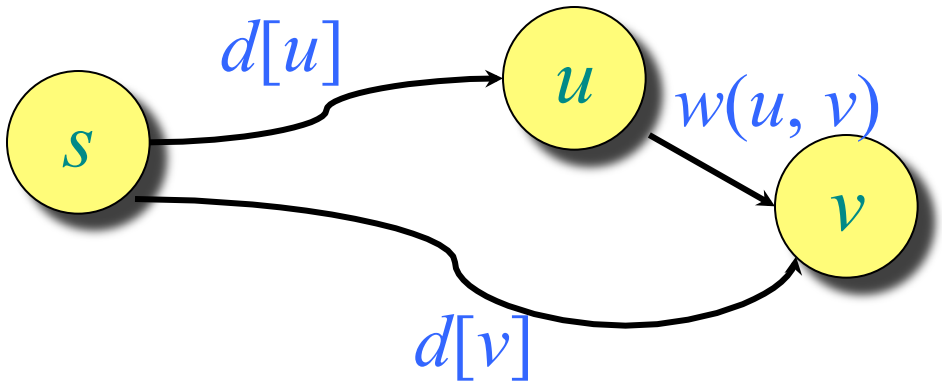


Correctness — Part I

Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof. Recall relaxation step:

if $d[v] > d[u] + w(u, v)$ **set** $d[v] \leftarrow d[u] + w(u, v)$



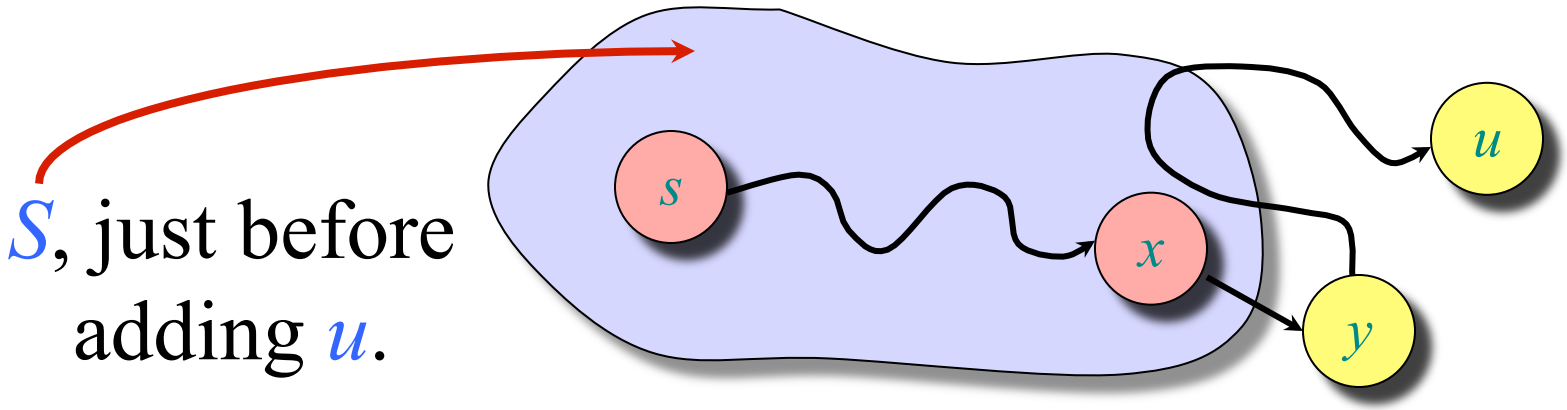
Correctness — Part II

Theorem. Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

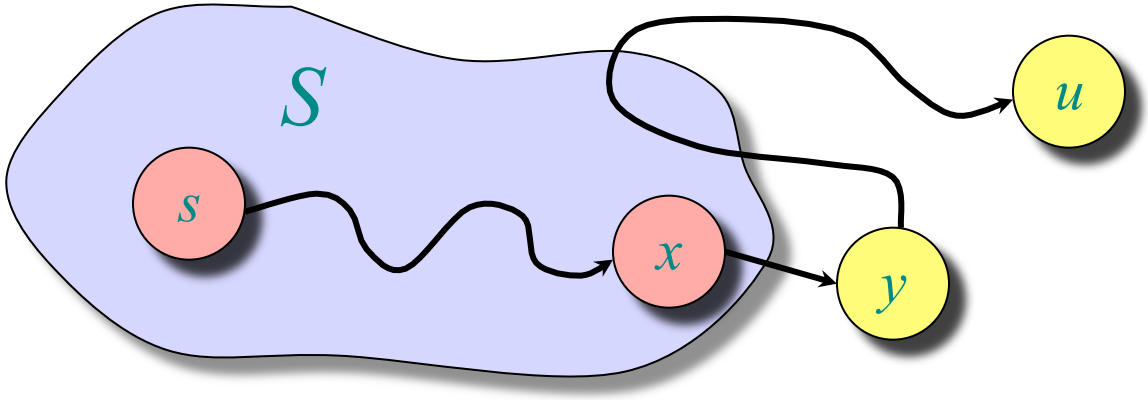
Proof.

• It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when v is added to S

• Suppose u is the first vertex added to S for which $d[u] \neq \delta(s, u)$. Let y be the first vertex in $V - S$ along a shortest path from s to u, and let x be its predecessor:



Correctness — Part II (continued)



• Since u is the first vertex violating the claimed invariant, we have $d[x] = \delta(s, x)$

• Since subpaths of shortest paths are shortest paths, it follows that $d[y]$ was set to $\delta(s, x) + w(x, y) = \delta(s, y)$ just after x was added to S

• Consequently, we have $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$

• But, $d[y] \geq d[u]$ since the algorithm chose u first => a contradiction