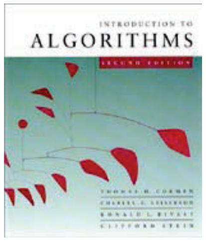
Shortest Path Algorithms

The slides source: Prof. Erik Demaine
Link: https://engineeringppt.com/category/electrical-engineering/

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

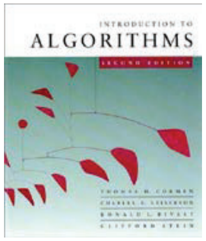


Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

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

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

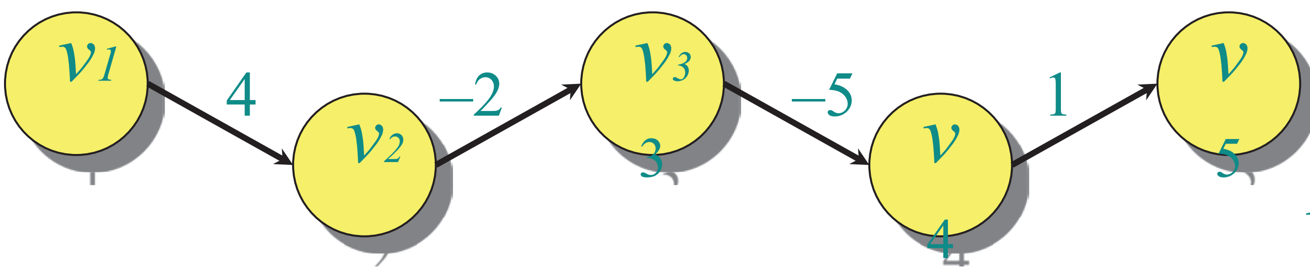


Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$. The ***weight*** of path $p = v_1 \to v_2 \to \cdots \to v_k$ is defined to be

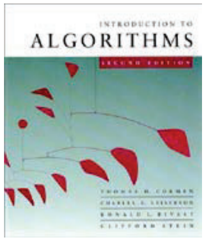
$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}) .$$

Example:



$w(p) = -2$

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Shortest paths

A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

$$\delta(u, v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}.$$

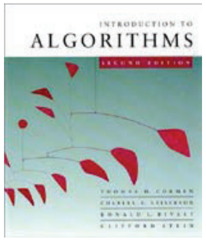
Note: $\delta(u, v) = \infty$ if no path from u to v exists.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Well-definedness of shortest paths

If a graph G contains a negative-weight cycle, then some shortest paths do not exist.

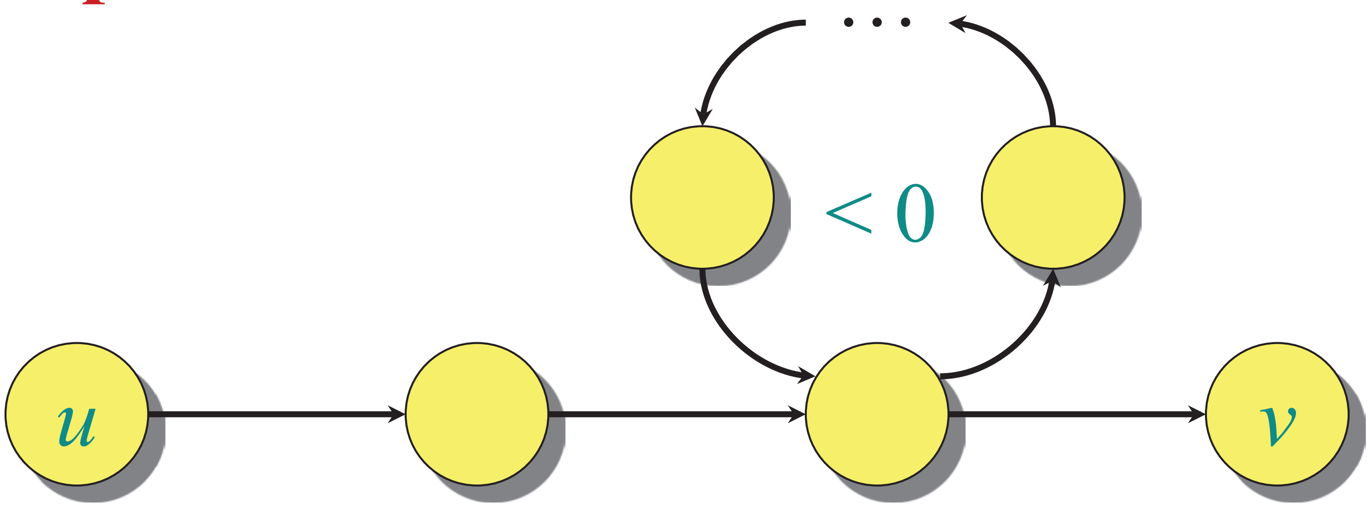
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Well-definedness of shortest paths

If a graph G contains a negative-weight cycle, then some shortest paths do not exist.

Example:



Optimal substructure

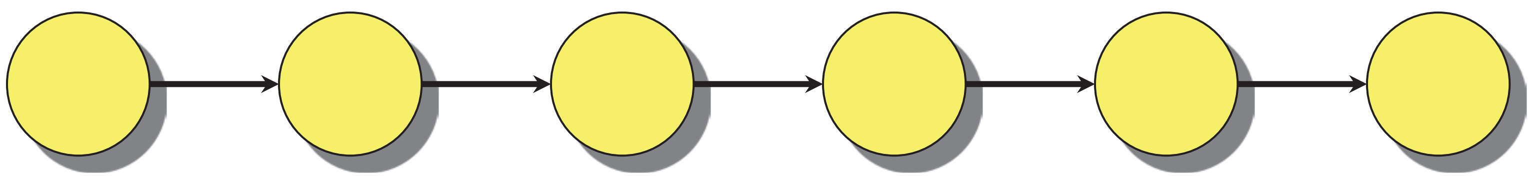
Theorem. A subpath of a shortest path is a shortest path.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

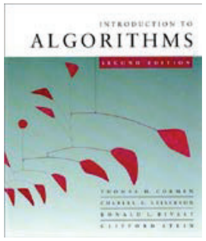
Optimal substructure

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:



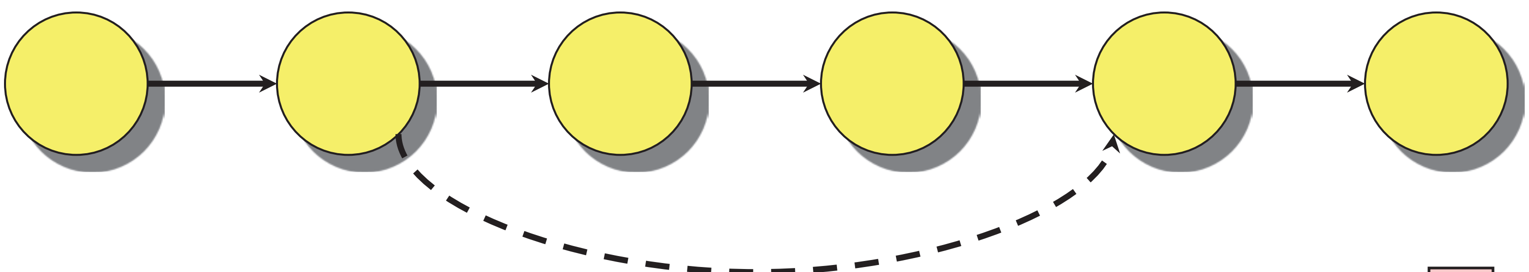
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Optimal substructure

Theorem. A subpath of a shortest path is a shortest path.

Proof. Cut and paste:



□

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Triangle inequality

Theorem. For all $u, v, x \in V$, we have

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

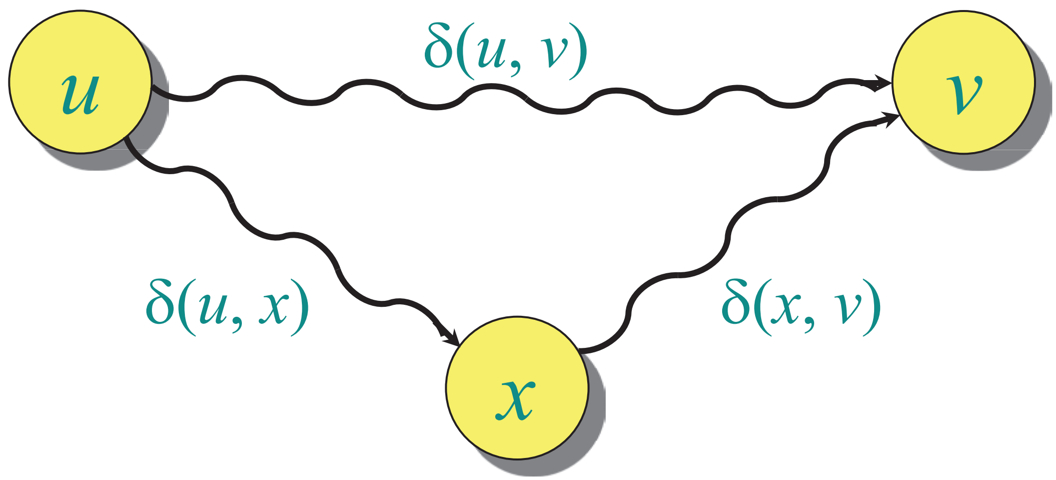
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

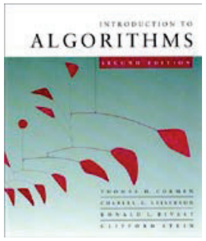
Triangle inequality

Theorem. For all $u, v, x \in V$, we have

$$\delta(u, v) \le \delta(u, x) + \delta(x, v).$$

Proof.





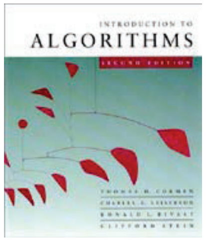
Single-source shortest paths (nonnegative edge weights)

Problem. Assume that $w(u, v) \geq 0$ for all $(u, v) \in E$. (Hence, all shortest-path weights must exist.) From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$.

IDEA: Greedy.

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 minimum.
3. Update the distance estimates of vertices adjacent to v.

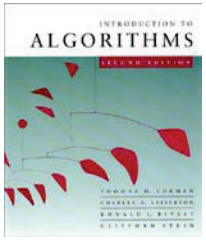
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$, keyed on $d[v]$

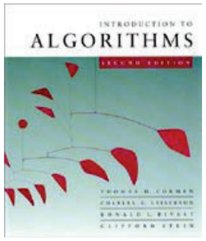
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$, keyed on $d[v]$
while $Q \neq \varnothing$
  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 $d[v] \leftarrow d[u] + w(u, v)$

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

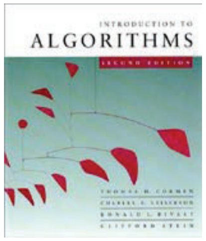


Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$, keyed on $d[v]$
while $Q \neq \varnothing$
    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 $d[v] \leftarrow d[u] + w(u, v)$

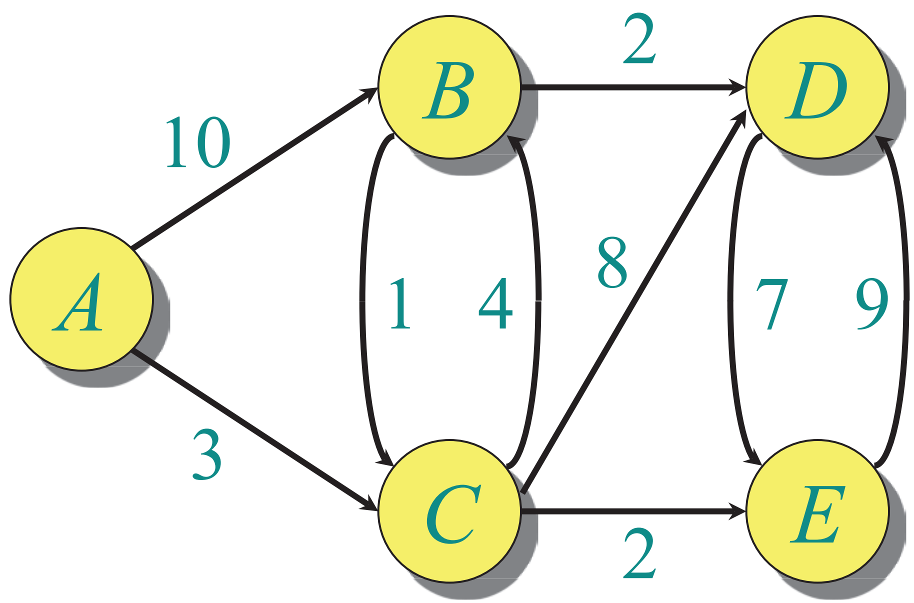
relaxation step

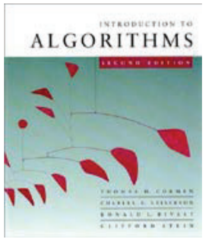
Implicit DECREASE-KEY



Example of Dijkstra's algorithm

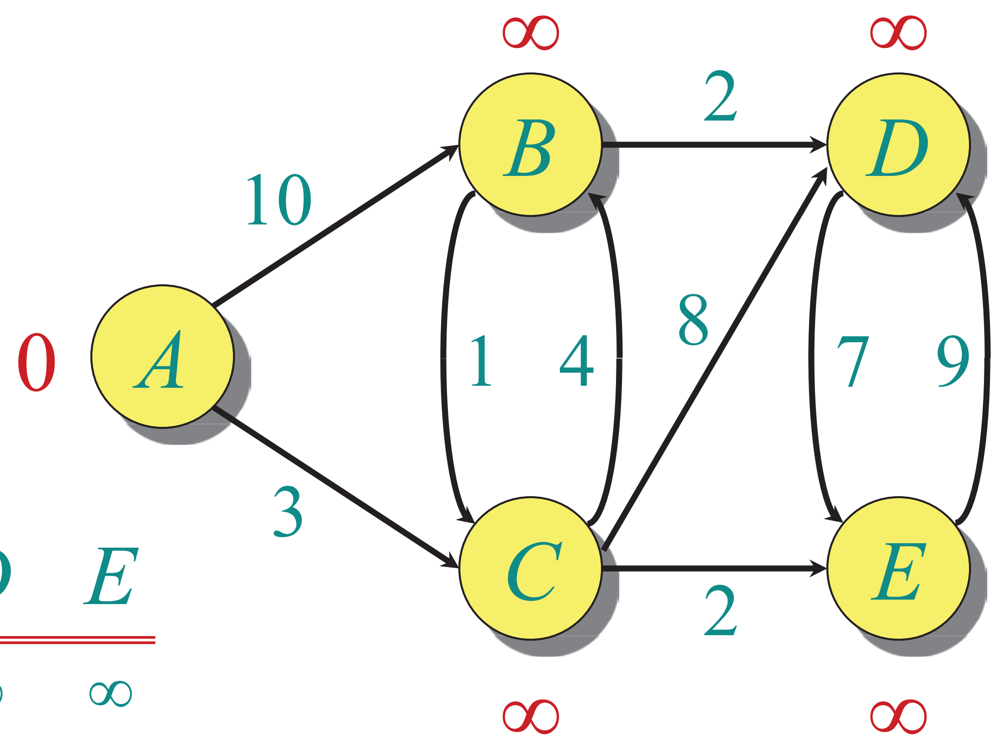
Graph with nonnegative edge weights:





Example of Dijkstra's algorithm

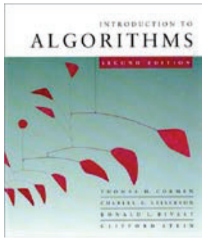
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

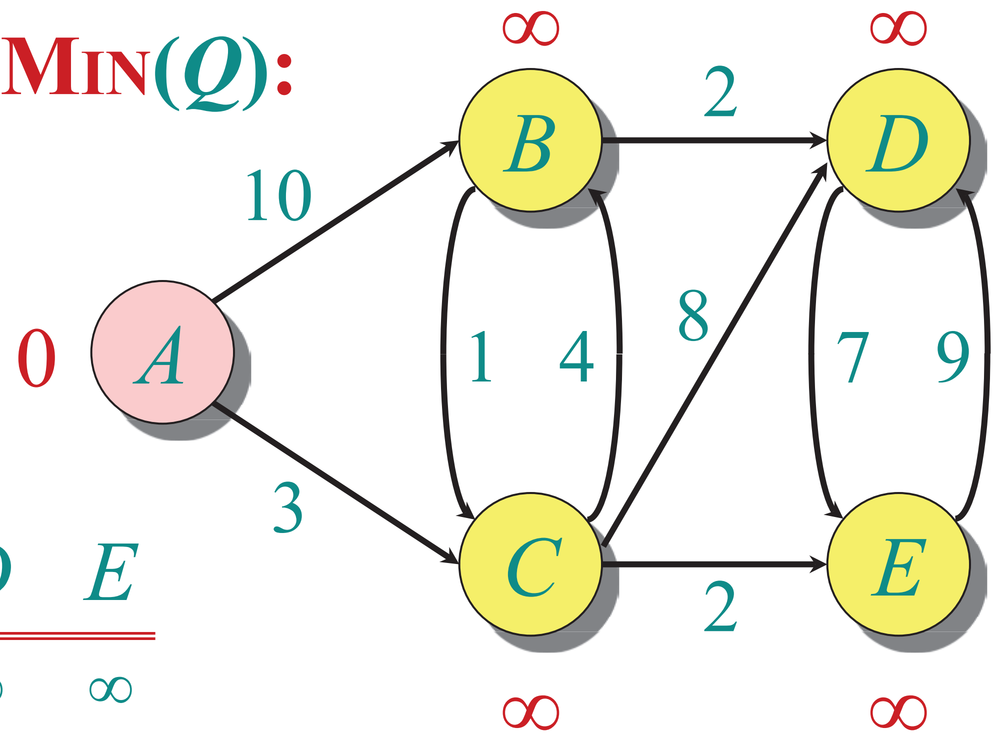
S: { }

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Example of Dijkstra’s algorithm

“A” ← EXTRACT-MIN(Q):

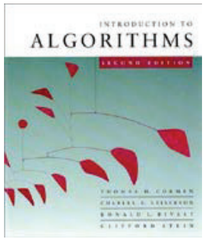


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

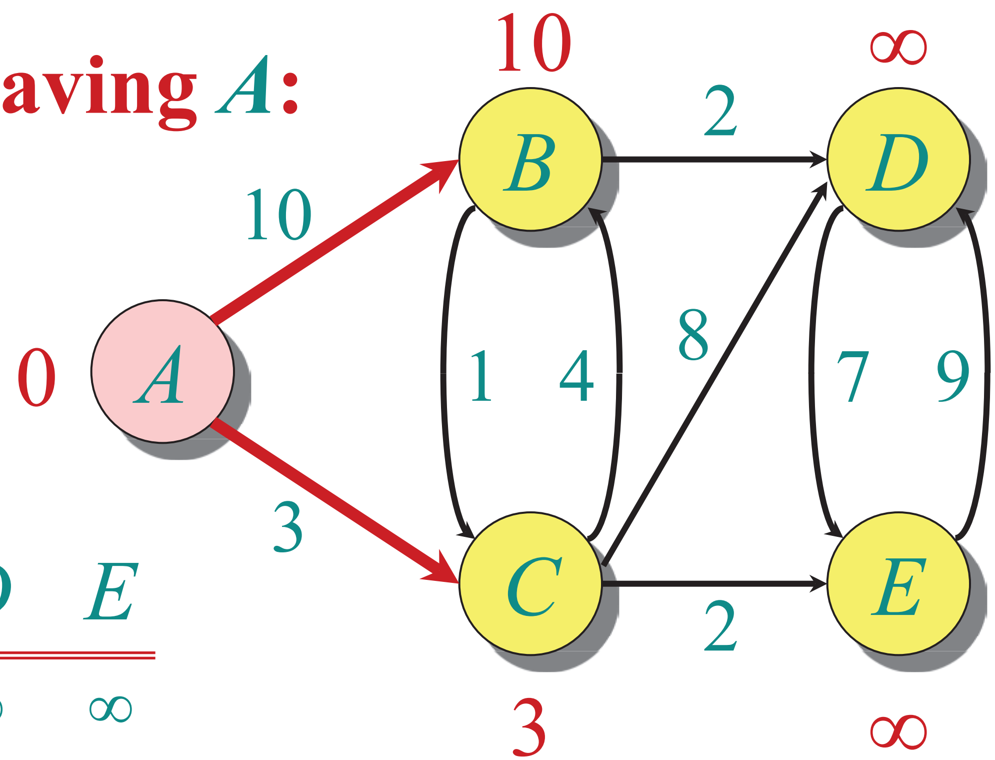
S: { A }

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Example of Dijkstra's algorithm



Relax all edges leaving A:



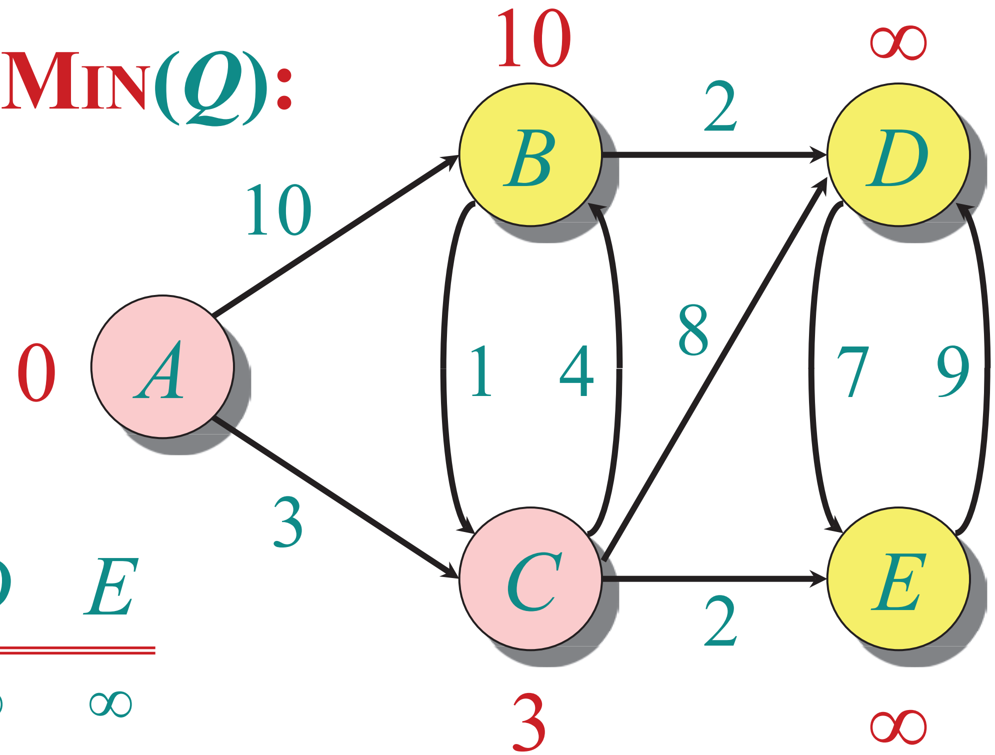
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Example of Dijkstra's algorithm

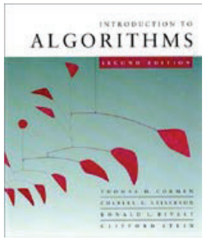
"*C*" ← **Extract-Min(*Q*):**



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

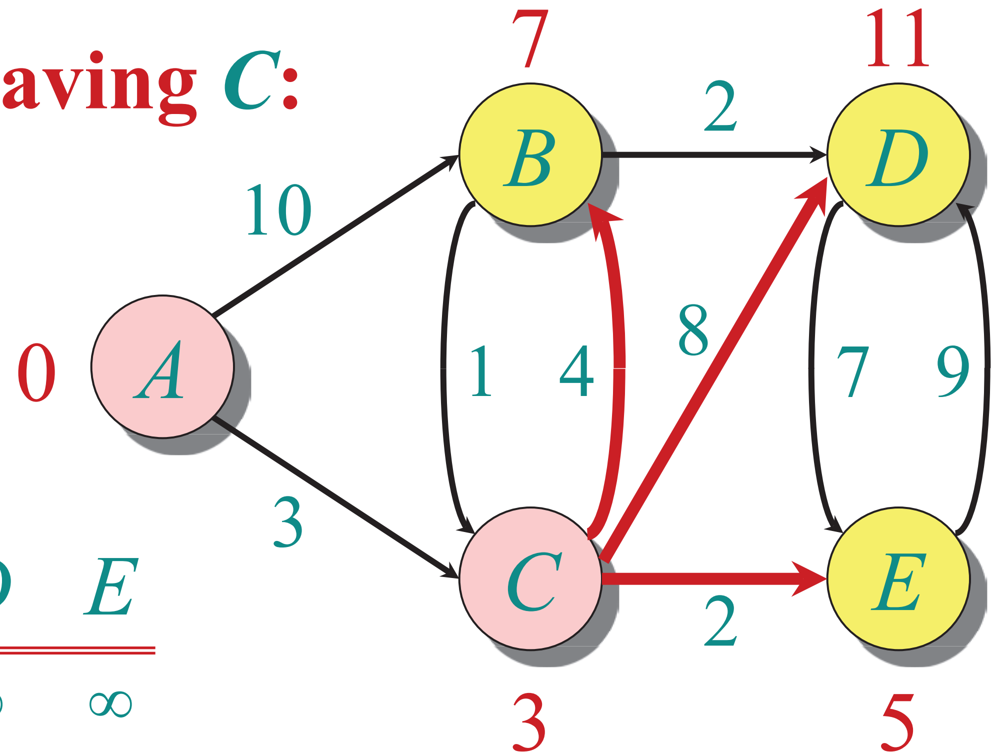
S: { *A*, *C* }

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Example of Dijkstra's algorithm

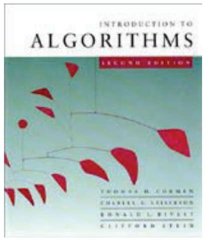
Relax all edges leaving *C*:



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	5

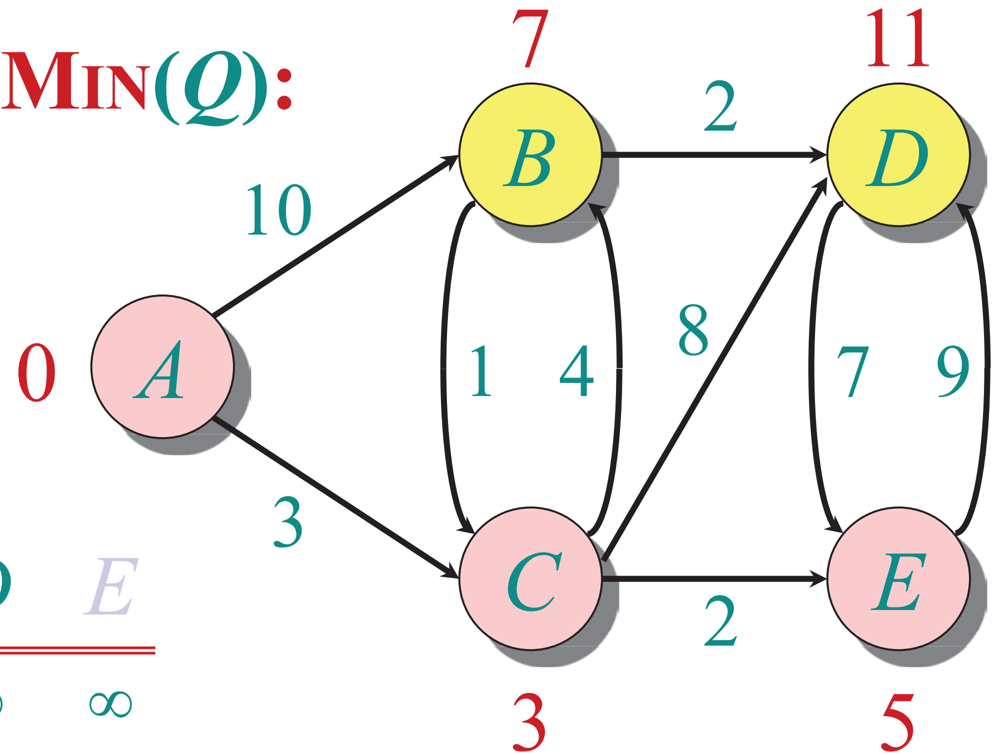
S: { A, C }

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Example of Dijkstra's algorithm

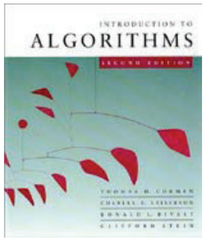
"*E*" ← EXTRACT-MIN(*Q*):



Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

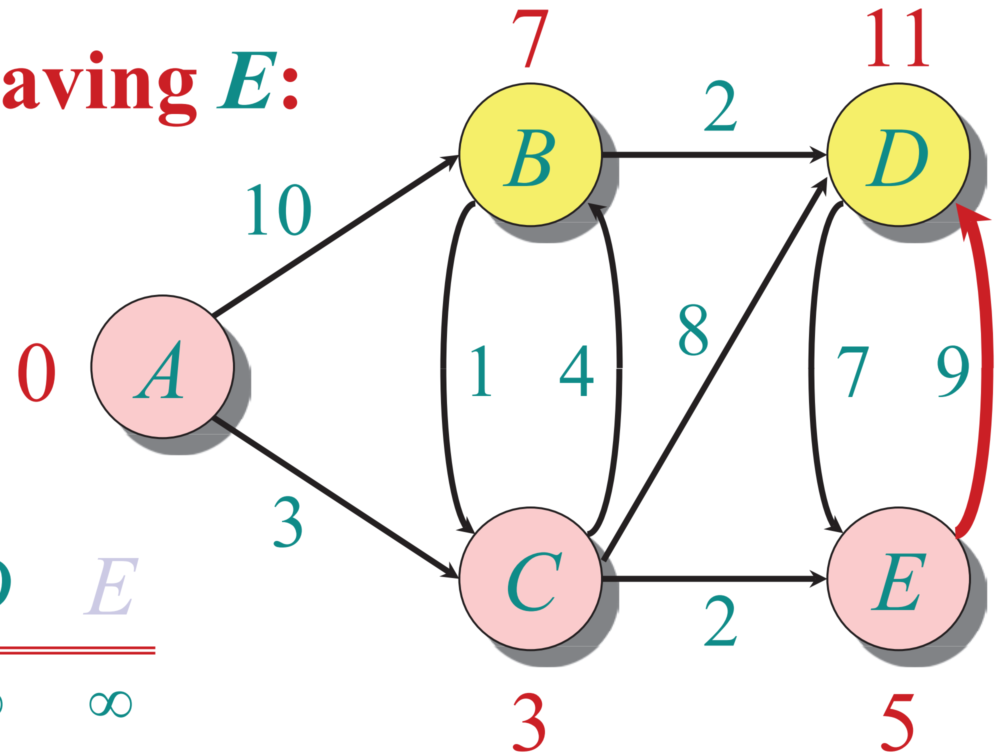
S: { *A*, *C*, *E* }

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Example of Dijkstra's algorithm

Relax all edges leaving *E*:

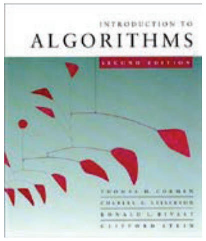


Q: A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	

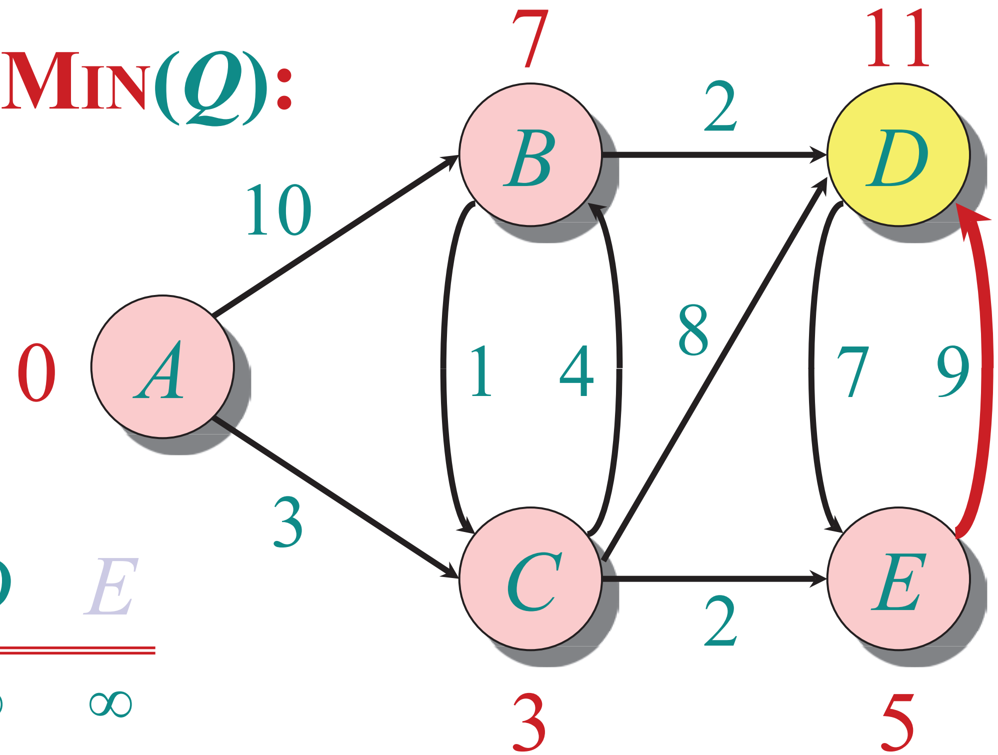
S: { A, C, E }

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Example of Dijkstra's algorithm

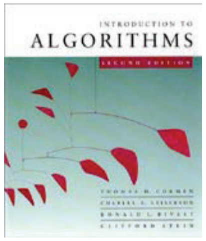


"B" ← EXTRACT-MIN(Q):



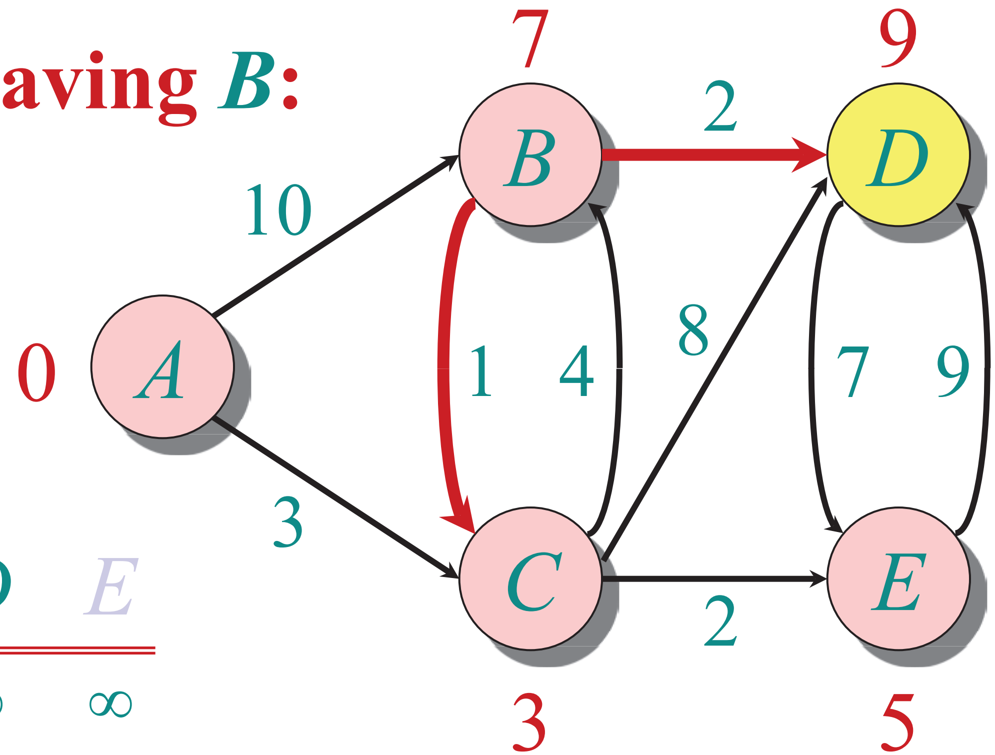
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }



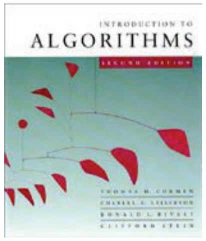
Example of Dijkstra's algorithm

Relax all edges leaving *B*:



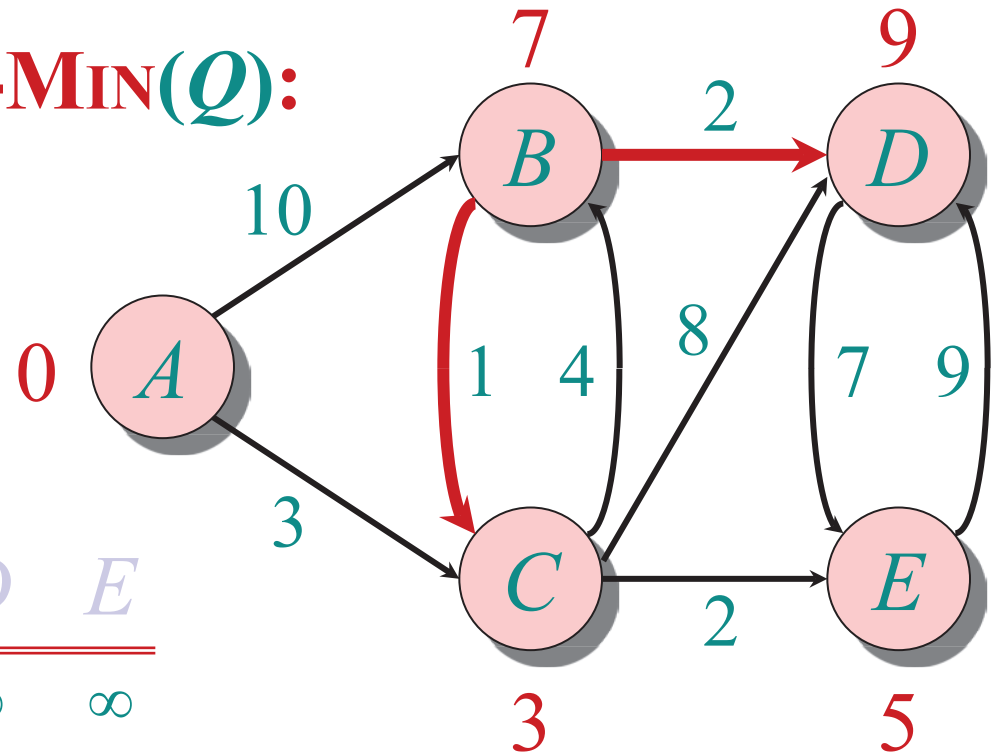
Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				9	

S: { A, C, E, B }



Example of Dijkstra's algorithm

"D" $\leftarrow$ EXTRACT-MIN(Q):

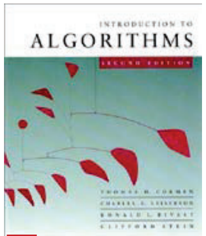


Q:

A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	10	$\boxed{3}$	∞	∞
	7		11	$\boxed{5}$
	$\boxed{7}$		11	
			$\boxed{9}$	

S: { A, C, E, B, D }

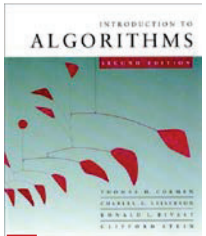
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



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.

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Correctness — Part I

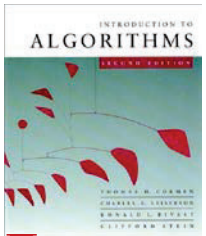
Lemma. Initializing $d[s] \leftarrow 0$ and $d[v] \leftarrow \infty$ for all $v \in V - \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps.

Proof. Suppose not. Let v be the first vertex for which $d[v] < \delta(s, v)$, and let u be the vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u, v)$. Then,

$d[v] < \delta(s, v)$	supposition
$\leq \delta(s, u) + \delta(u, v)$	triangle inequality
$\leq \delta(s,u) + w(u, v)$	sh. path $\leq$ specific path
$\leq d[u] + w(u, v)$	v is first violation

Contradiction. $\square$

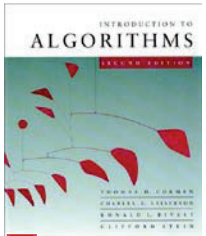
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Correctness — Part II

Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Correctness — Part II

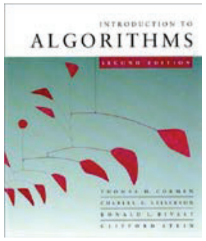
Lemma. Let u be v's predecessor on a shortest path from s to v. Then, if $d[u] = \delta(s, u)$ and edge (u, v) is relaxed, we have $d[v] = \delta(s, v)$ after the relaxation.

Proof. Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$. Suppose that $d[v] > \delta(s, v)$ before the relaxation. (Otherwise, we're done.) Then, the test $d[v] > d[u] + w(u, v)$ succeeds, because $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$, and the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$. ◻

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Correctness — Part III

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

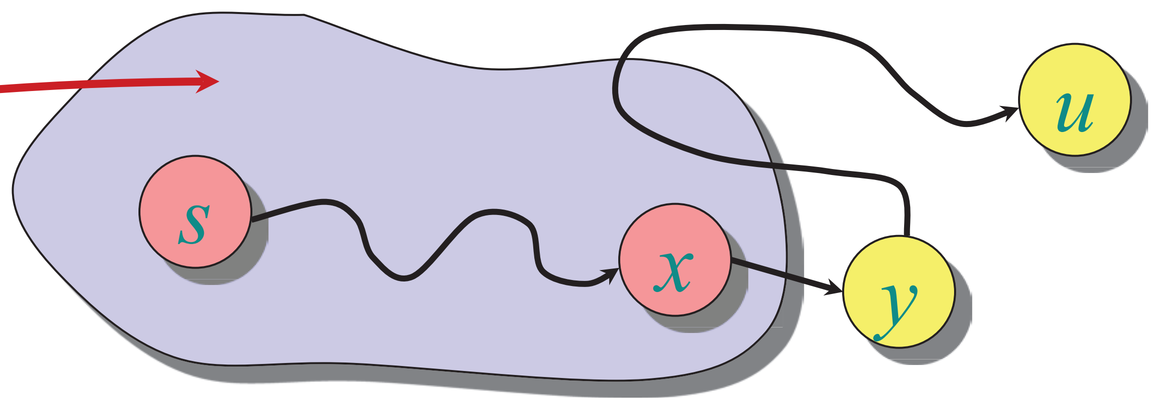


Correctness — Part III

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

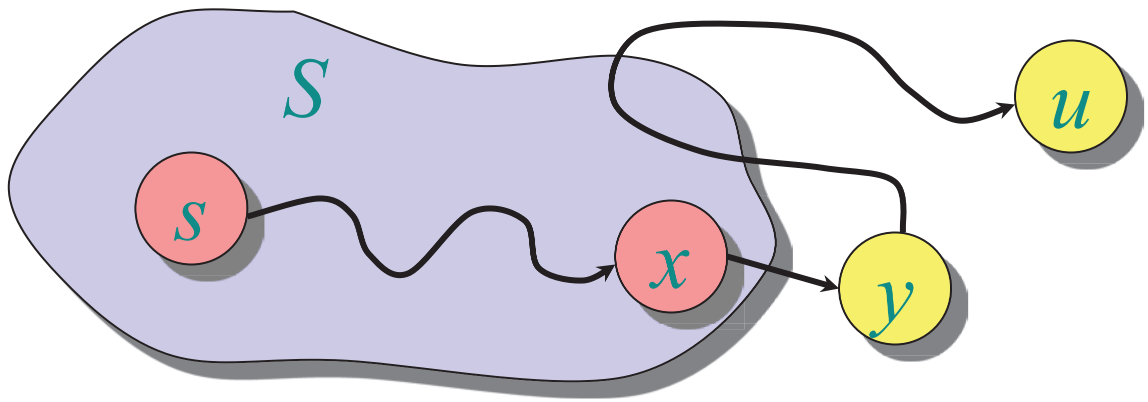
Proof. It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when v is added to S. Suppose u is the first vertex added to S for which $d[u] > \delta(s, u)$. Let y be the first vertex in $V - S$ along a shortest path from s to u, and let x be its predecessor:

S, just before adding u.



 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Correctness — Part III (continued)



Since u is the first vertex violating the claimed invariant, we have $d[x] = \delta(s, x)$. When x was added to S, the edge (x, y) was relaxed, which implies that $d[y] = \delta(s, y) \le \delta(s, u) < d[u]$. But, $d[u] \le d[y]$ by our choice of u. Contradiction. $\square$

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Analysis of Dijkstra

while $Q \neq \emptyset$

  do $u \leftarrow \textsc{Extract-Min}(Q)$

    $S \leftarrow S \cup \{u\}$

    for each $v \in Adj[u]$

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

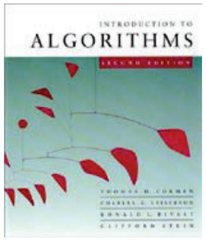
        then $d[v] \leftarrow d[u] + w(u, v)$

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Analysis of Dijkstra

$|V|$ times

while $Q \neq \emptyset$
  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 $d[v] \leftarrow d[u] + w(u, v)$



Analysis of Dijkstra

$|V|$ times

while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$

$degree(u)$ times

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

Analysis of Dijkstra

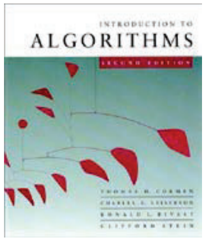
$|V|$ times

while $Q \neq \emptyset$
  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 $d[v] \leftarrow d[u] + w(u, v)$

degree(u) times

Handshaking Lemma $\Rightarrow$ $\Theta(E)$ implicit DECREASE-KEY's.

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Analysis of Dijkstra

$|V|$ times:

while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$

  $degree(u)$ times:

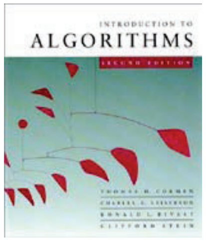
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

Time $= \Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$

Note: Same formula as in the analysis of Prim's minimum spanning tree algorithm.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Analysis of Dijkstra (continued)

Time = $\Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Analysis of Dijkstra (continued)

Time = $\Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$

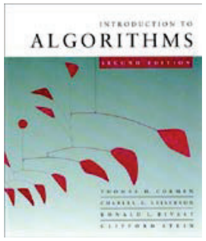
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Analysis of Dijkstra (continued)

Time = $\Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

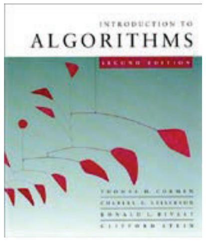


Analysis of Dijkstra (continued)

Time = $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case

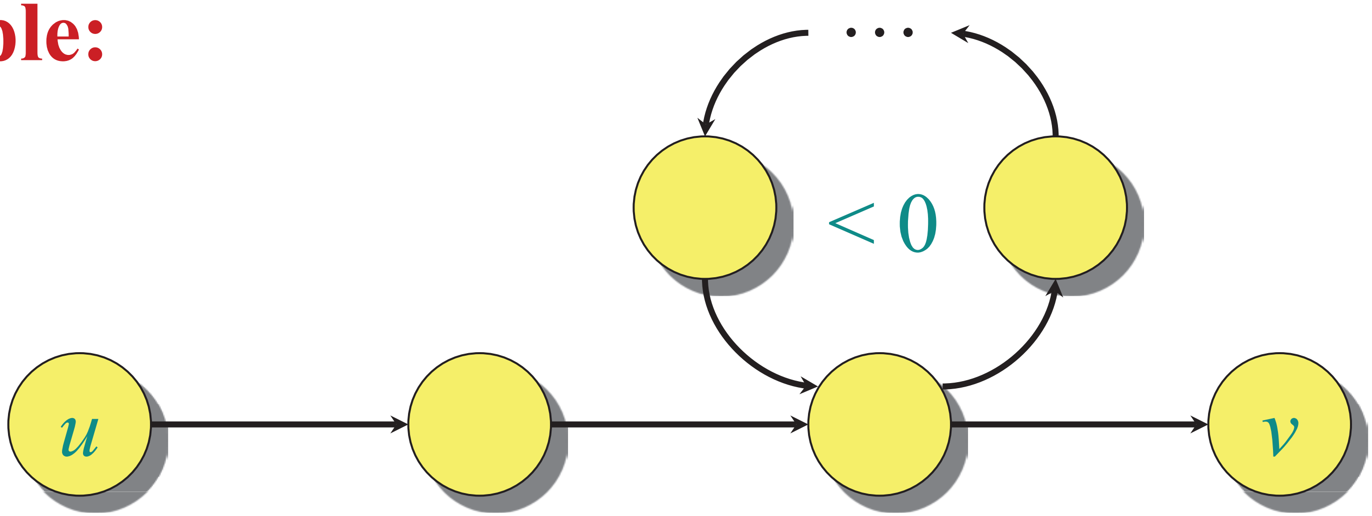
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



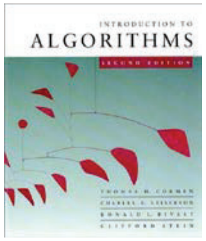
Negative-weight cycles

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:



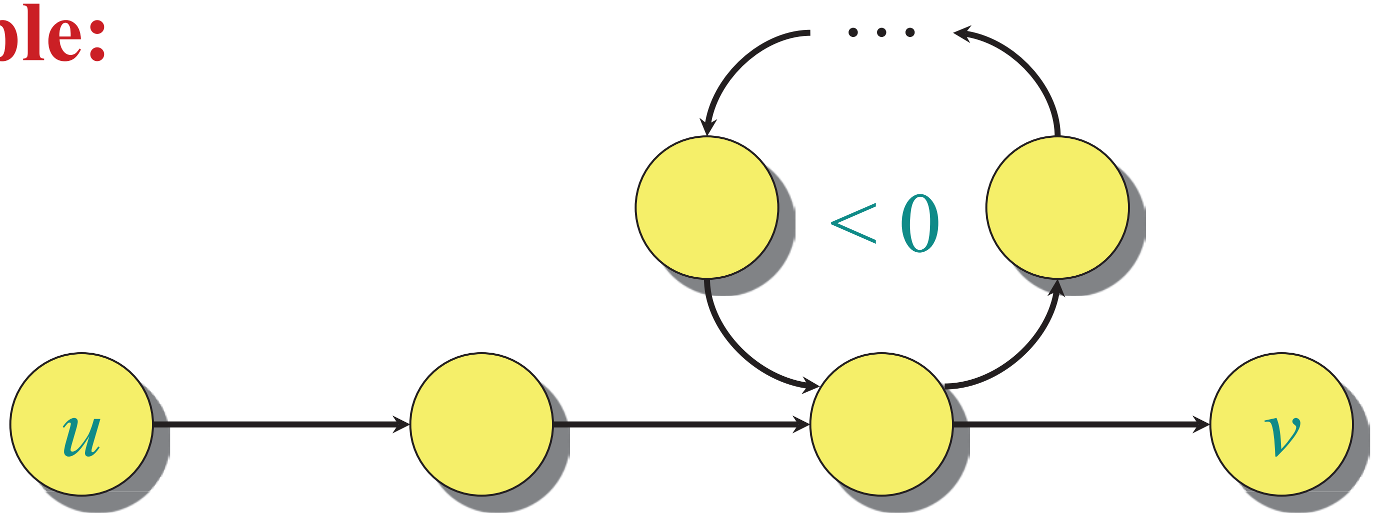
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Negative-weight cycles

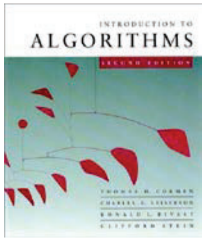
Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

Example:



Bellman-Ford algorithm: Finds all shortest-path lengths from a ***source*** $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Bellman-Ford algorithm

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

initialization

for $i \leftarrow 1$ **to** $|V| - 1$
    do for each edge $(u, v) \in E$
        do if $d[v] > d[u] + w(u, v)$
            then $d[v] \leftarrow d[u] + w(u, v)$

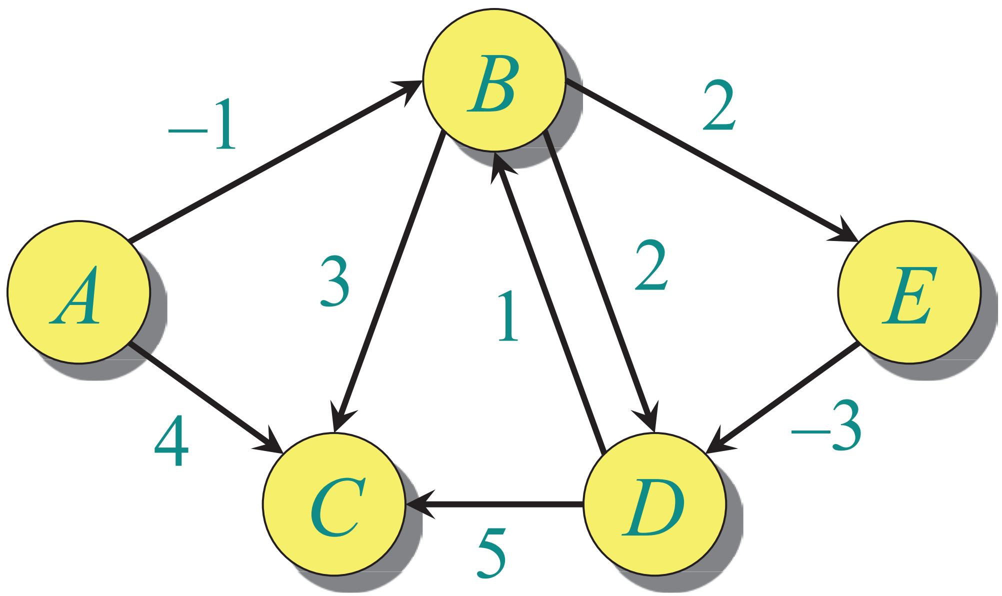
relaxation step

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

At the end, $d[v] = \delta(s, v)$, if no negative-weight cycles.
Time $= O(VE)$.

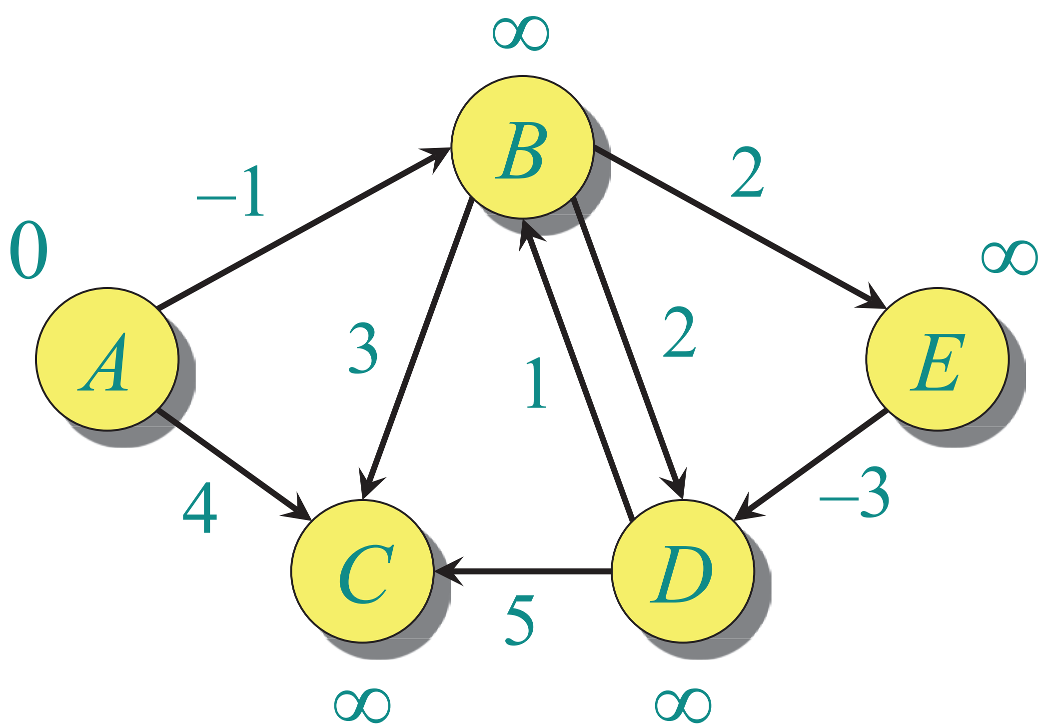
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Example of Bellman-Ford

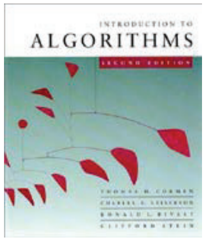


Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

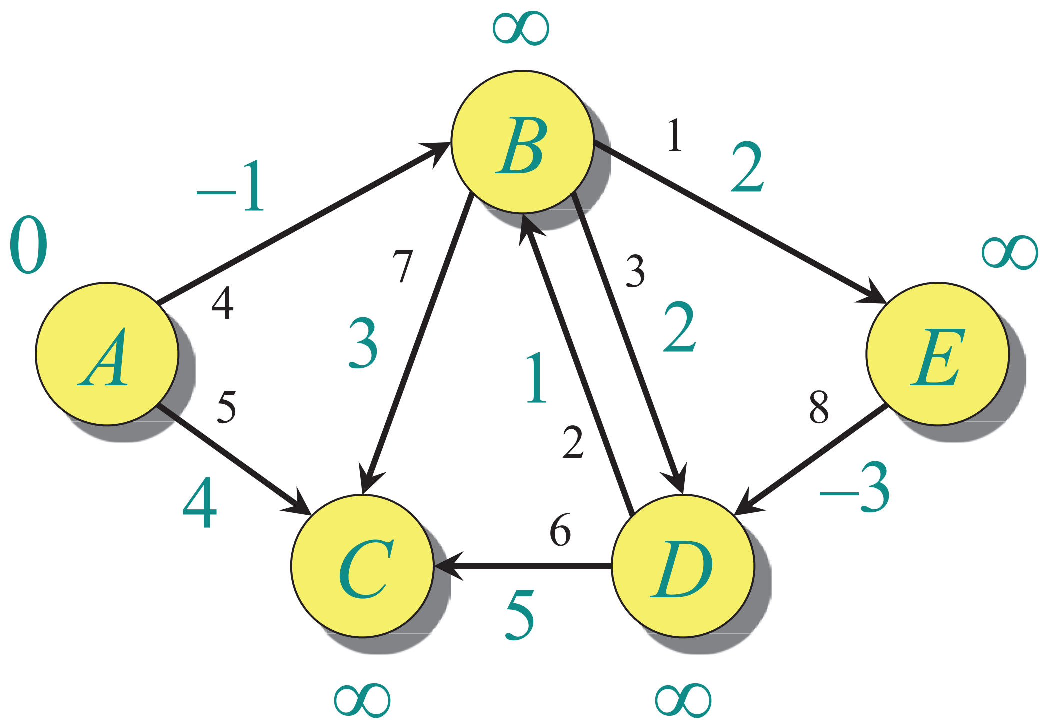
Example of Bellman-Ford



Initialization.



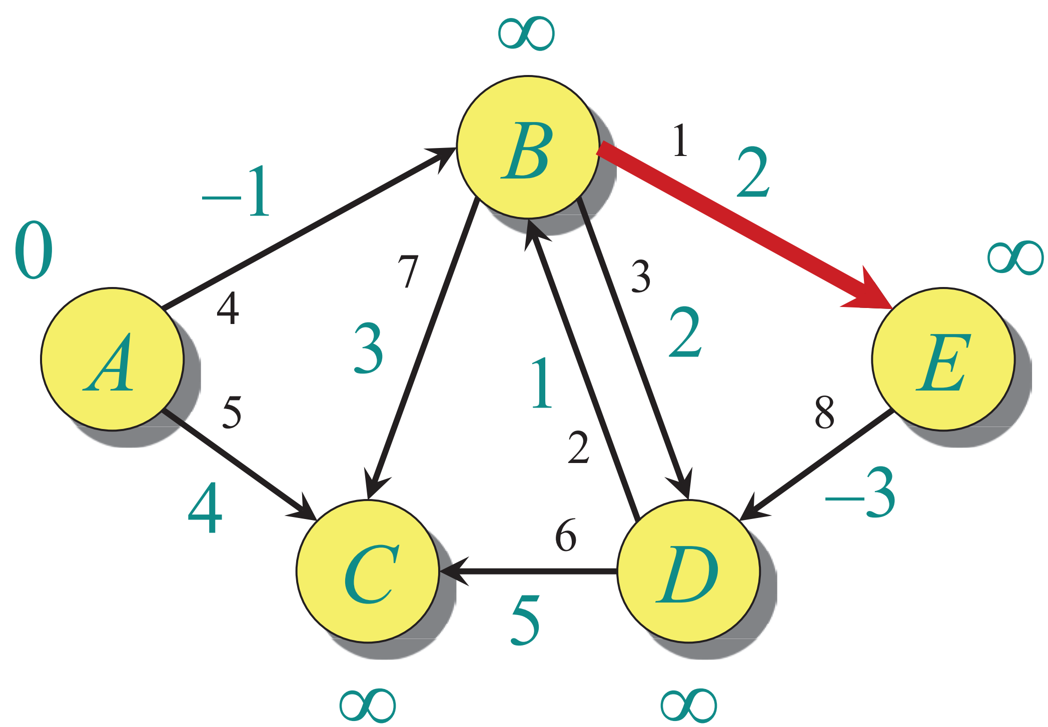
Example of Bellman-Ford



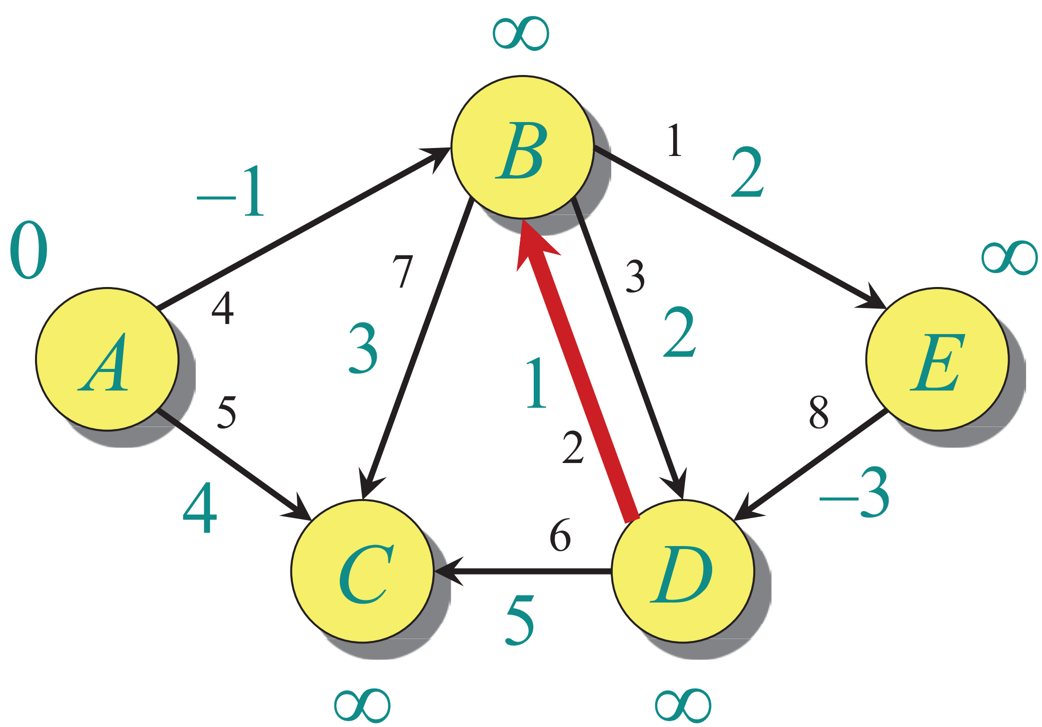
Order of edge relaxation.

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Example of Bellman-Ford

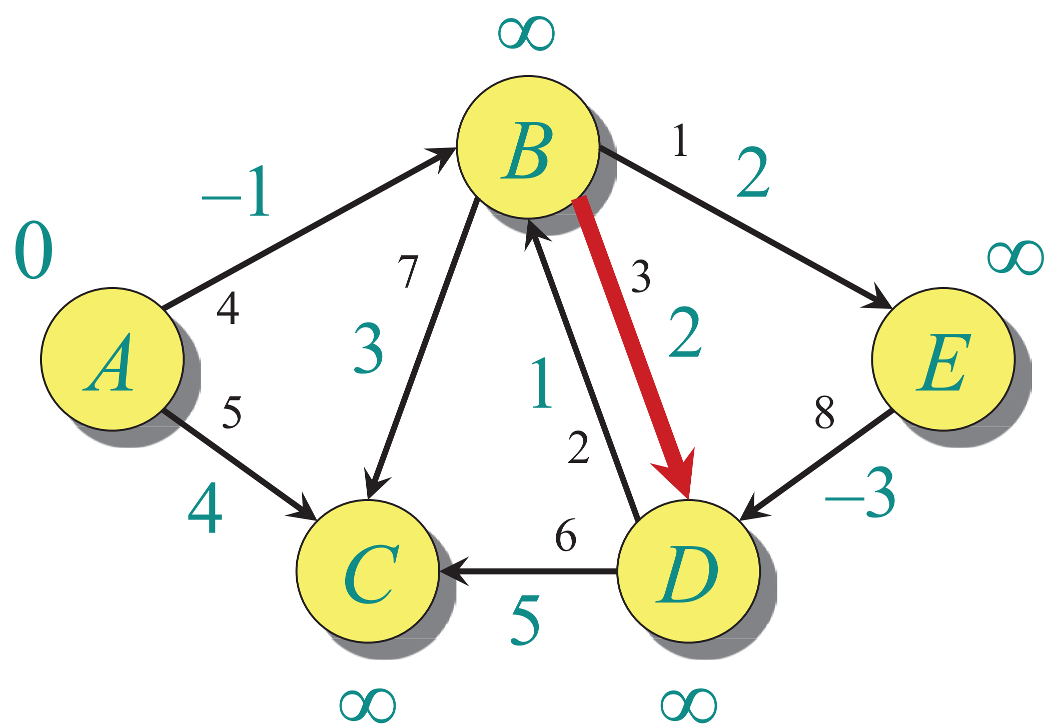


Example of Bellman-Ford

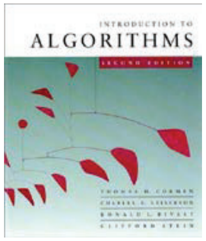


 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

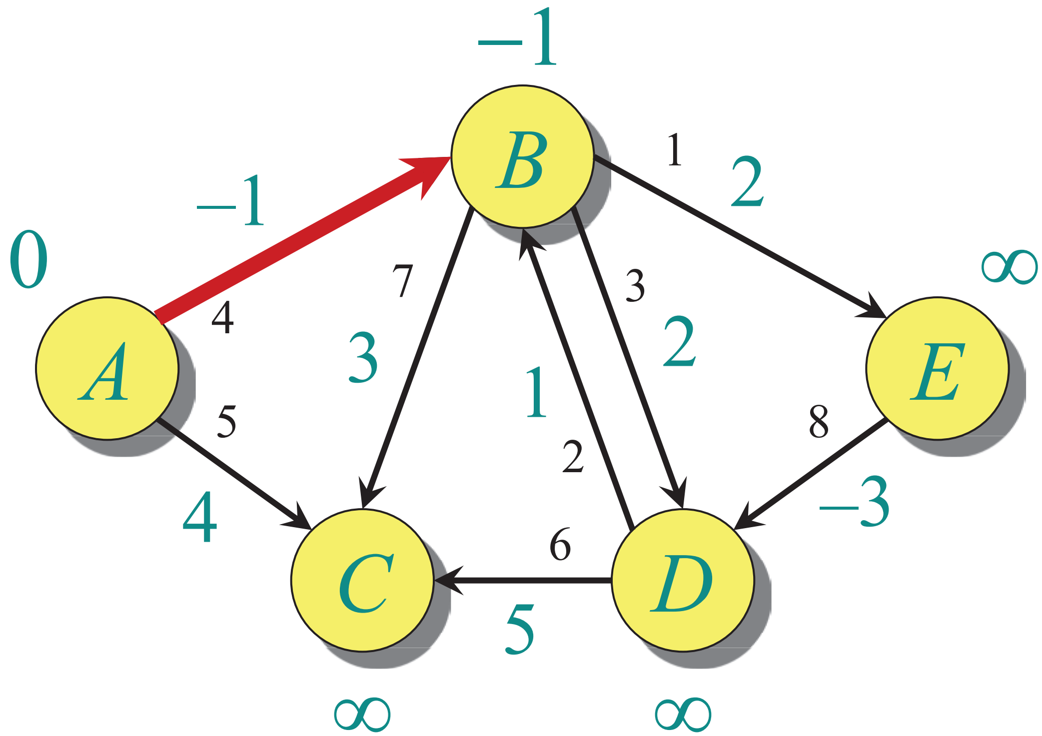
Example of Bellman-Ford



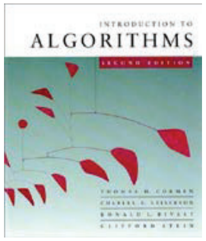
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



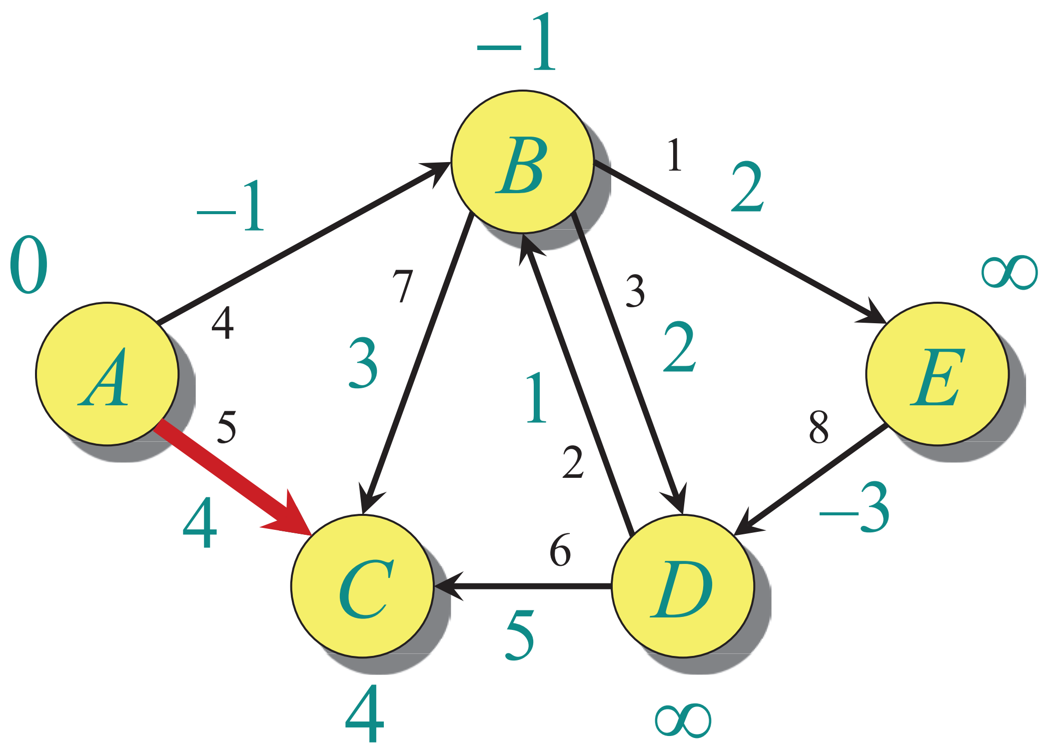
Example of Bellman-Ford



 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

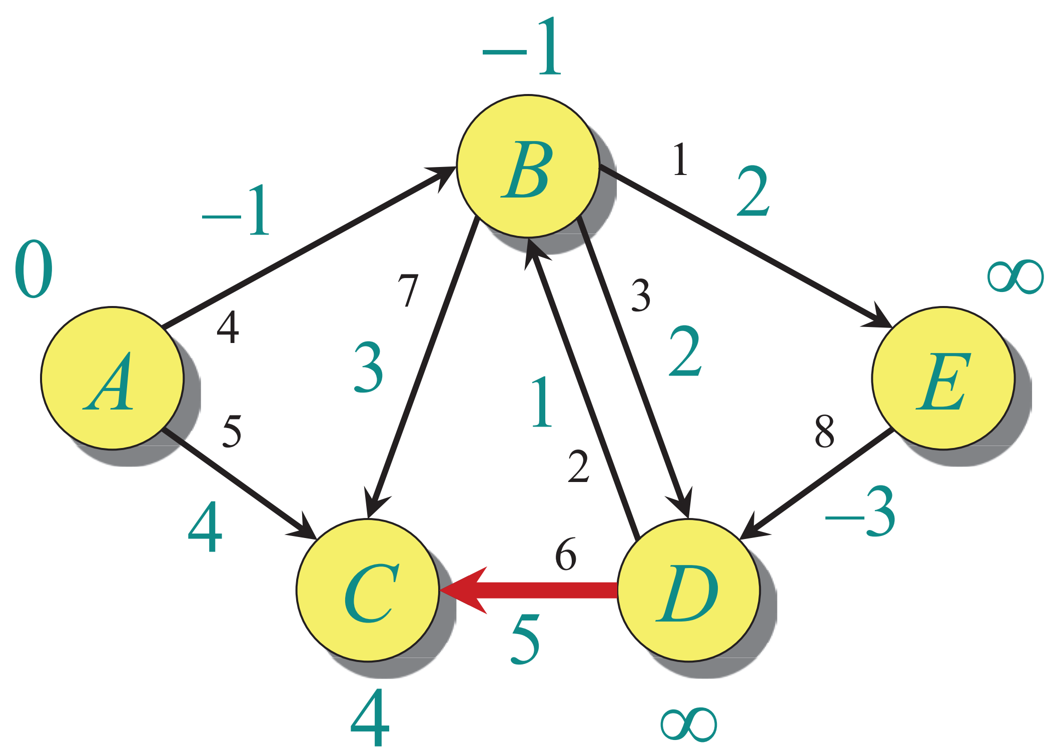


Example of Bellman-Ford

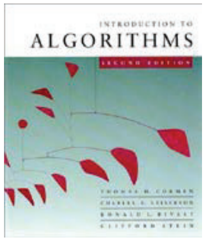


 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

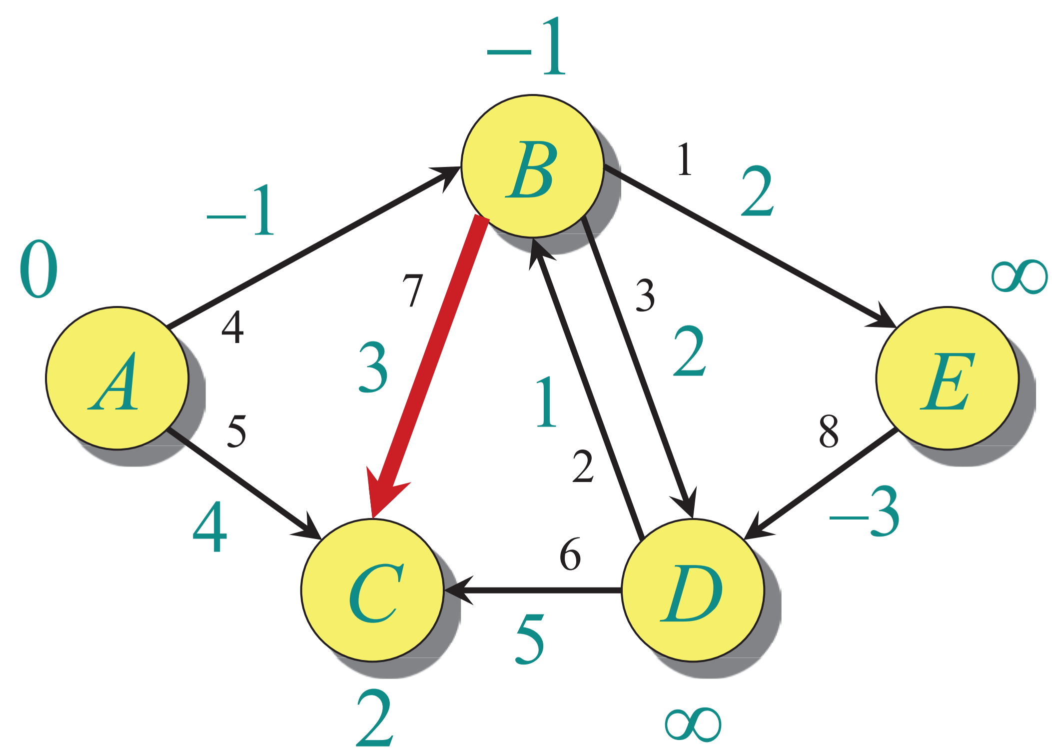
Example of Bellman-Ford



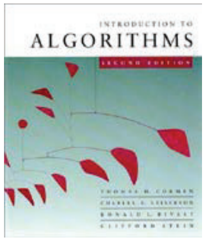
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



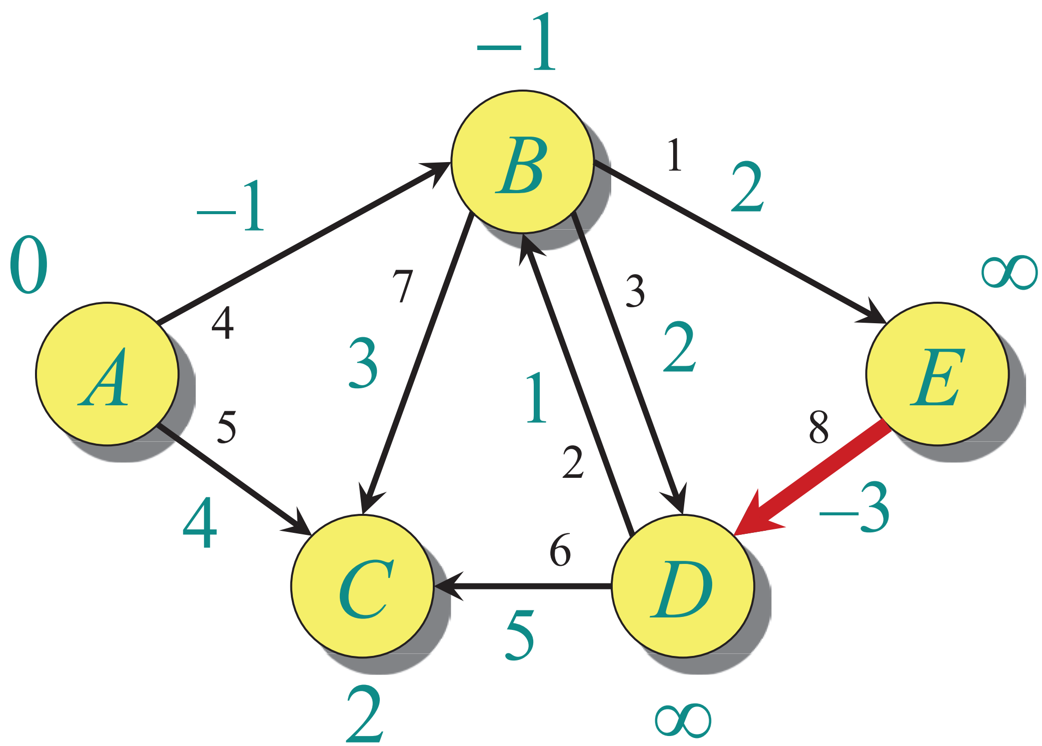
Example of Bellman-Ford

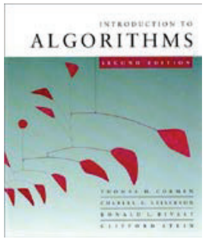


Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

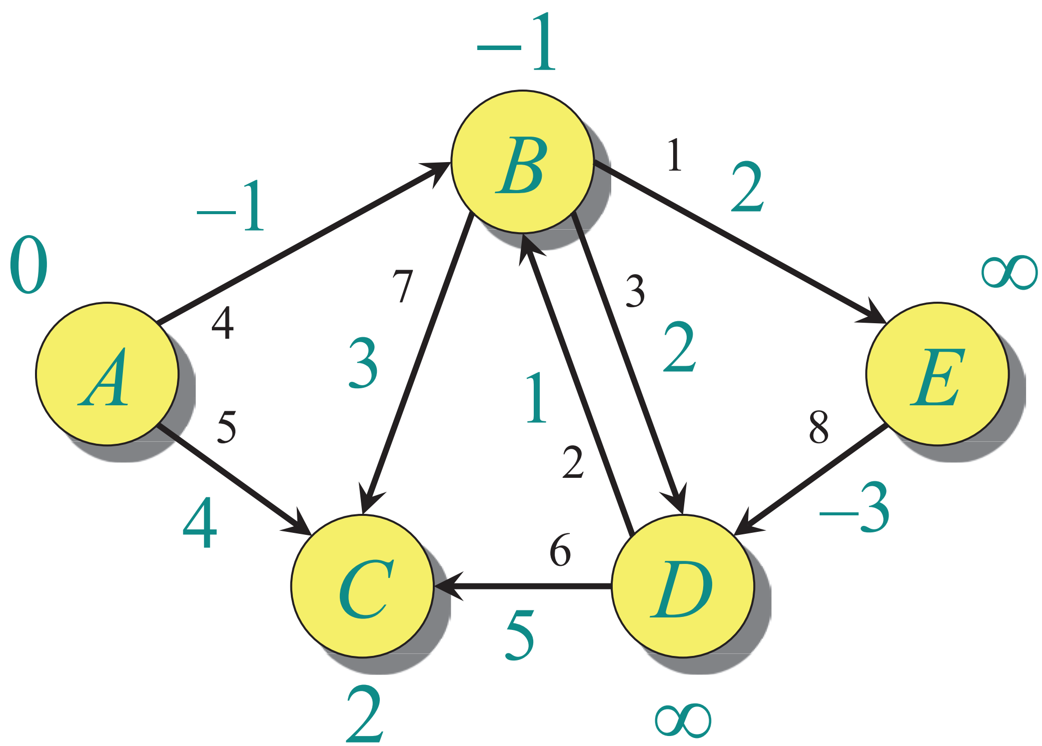


Example of Bellman-Ford



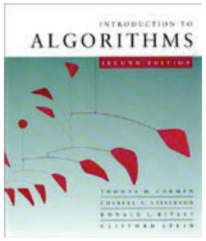


Example of Bellman-Ford

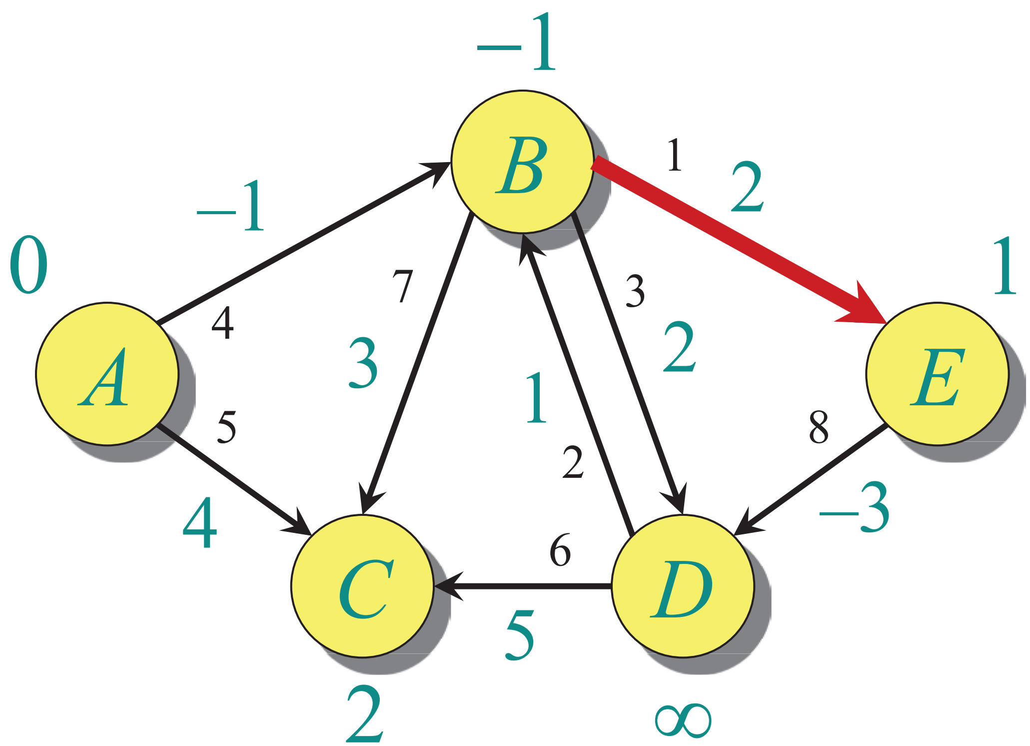


End of pass 1.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

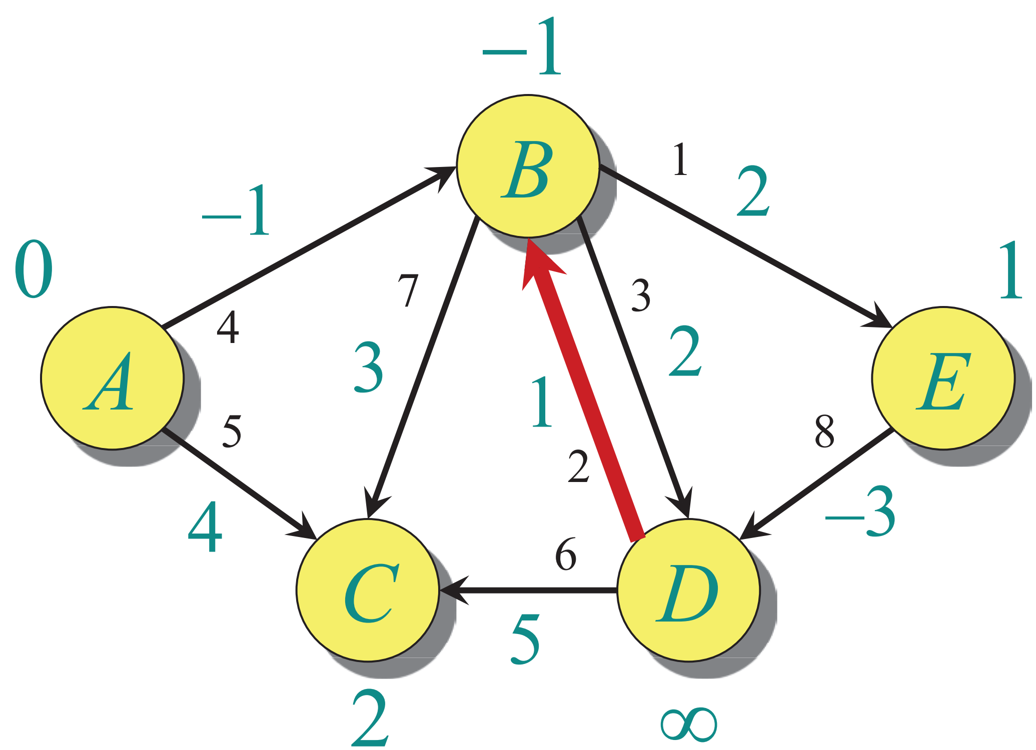


Example of Bellman-Ford

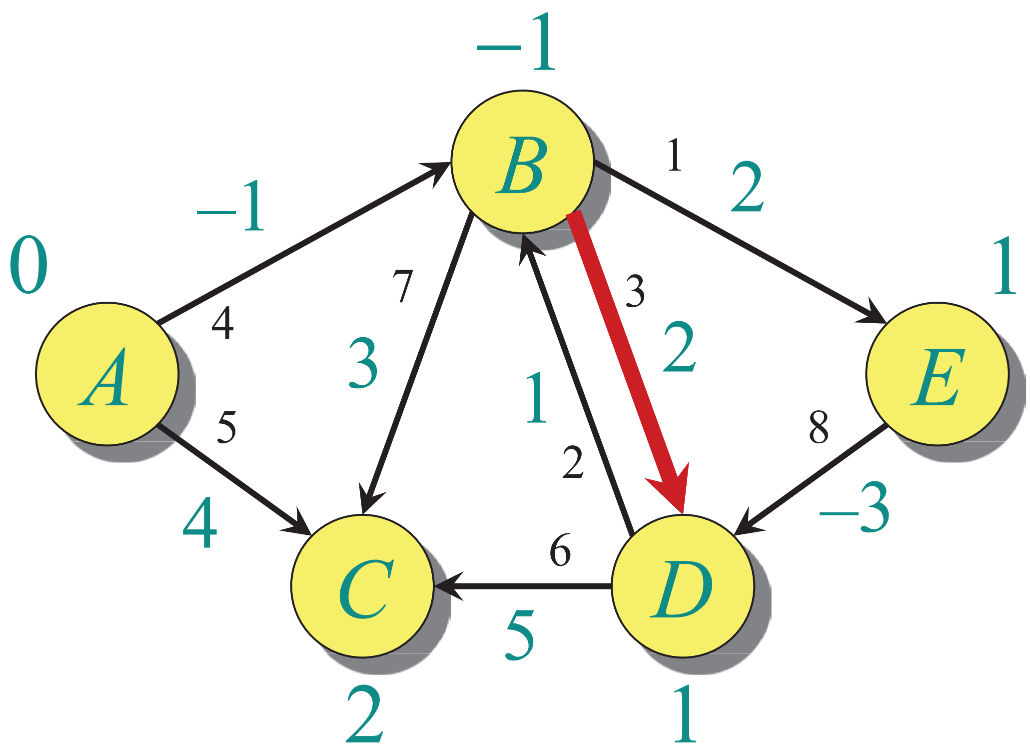


 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

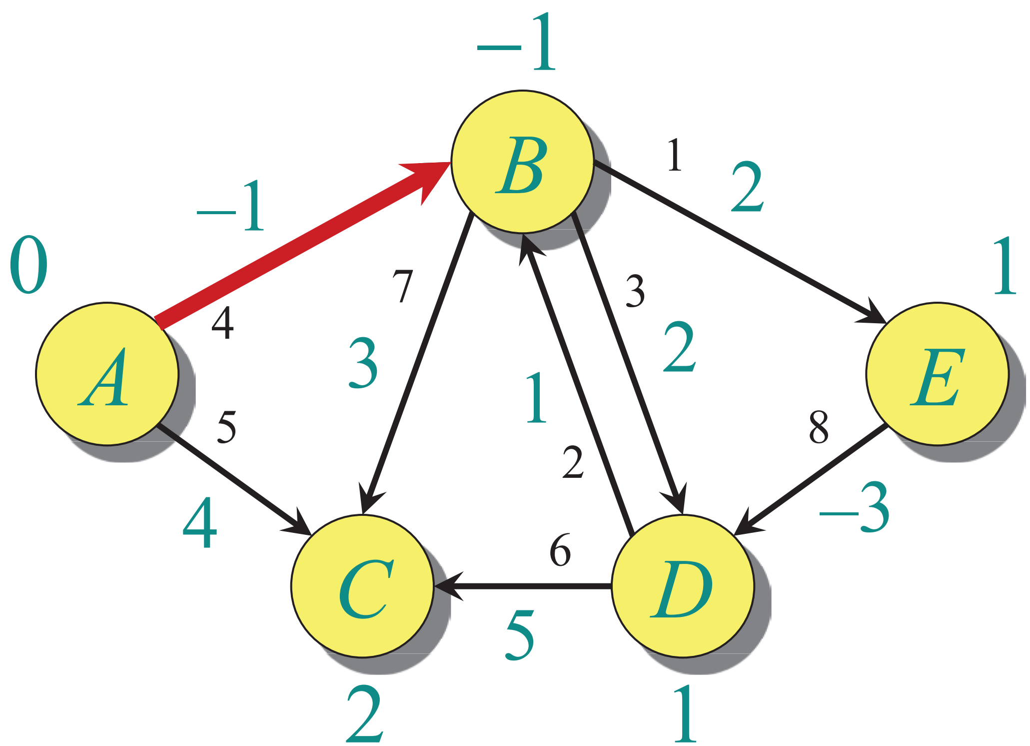
Example of Bellman-Ford



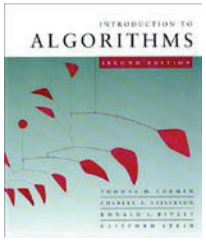
Example of Bellman-Ford



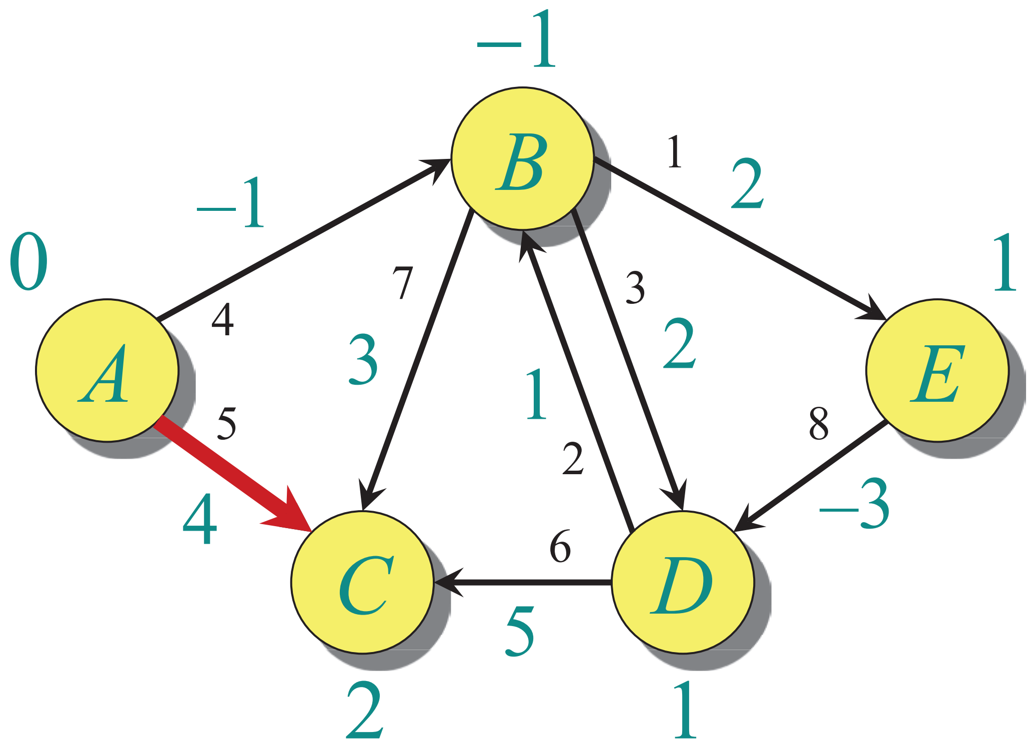
Example of Bellman-Ford



 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

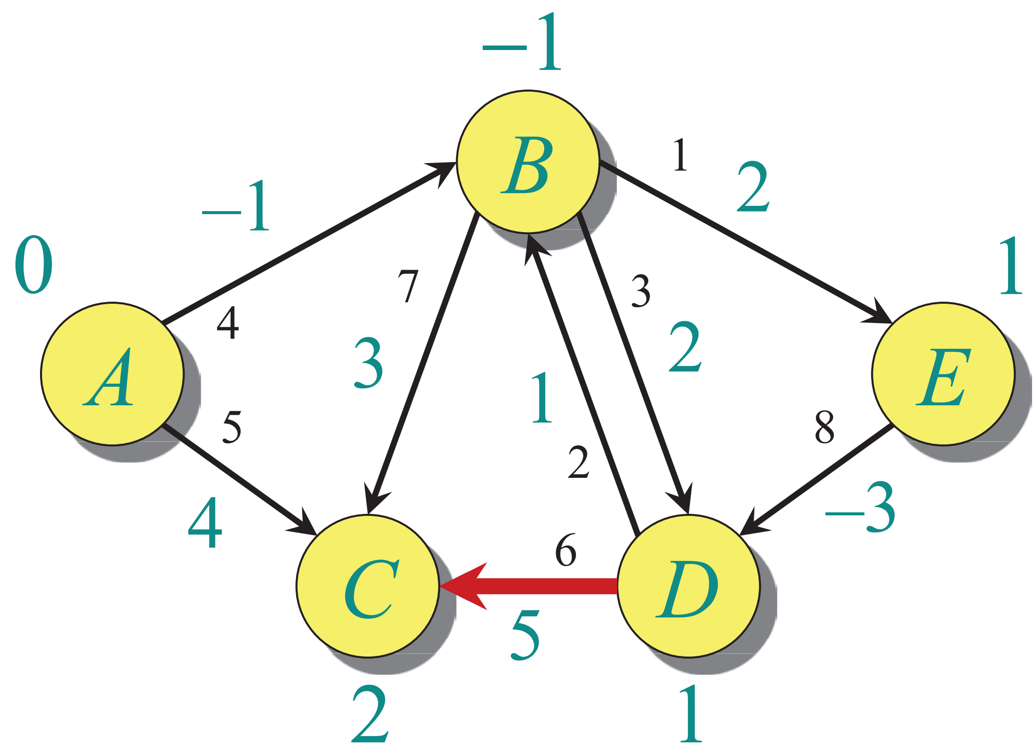


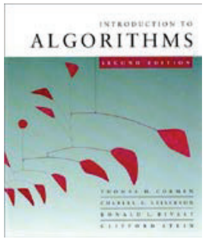
Example of Bellman-Ford



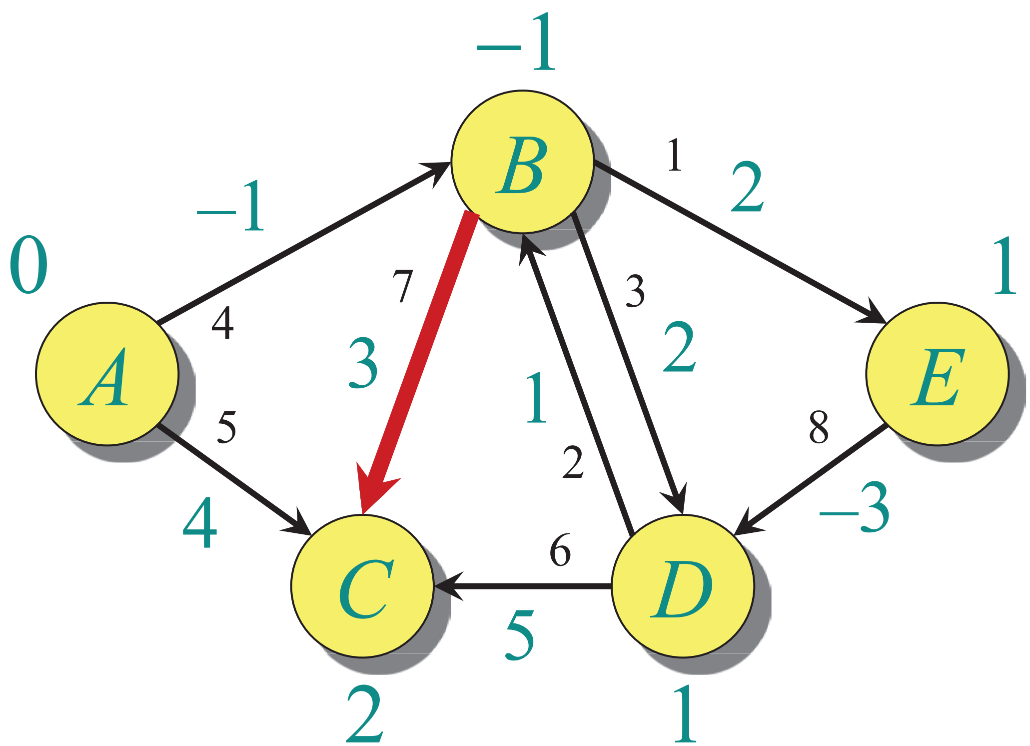
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Example of Bellman-Ford

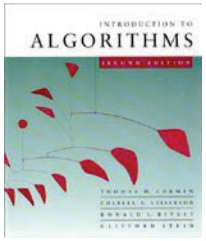




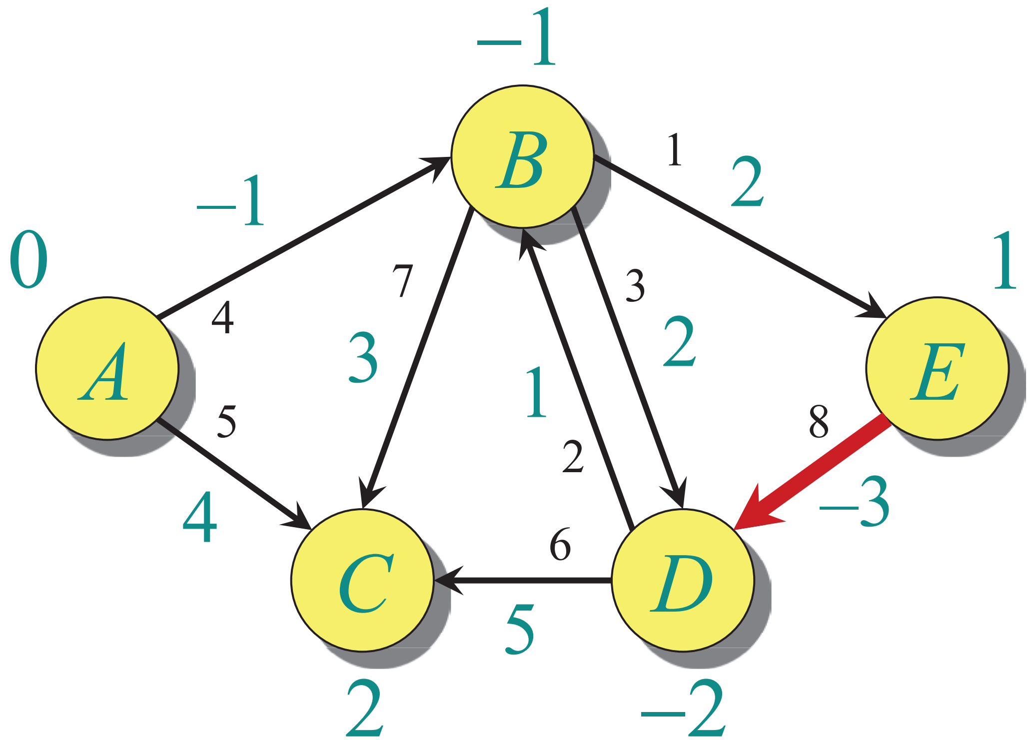
Example of Bellman-Ford



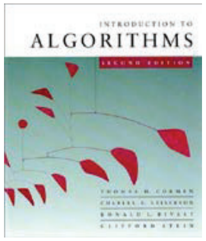
Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



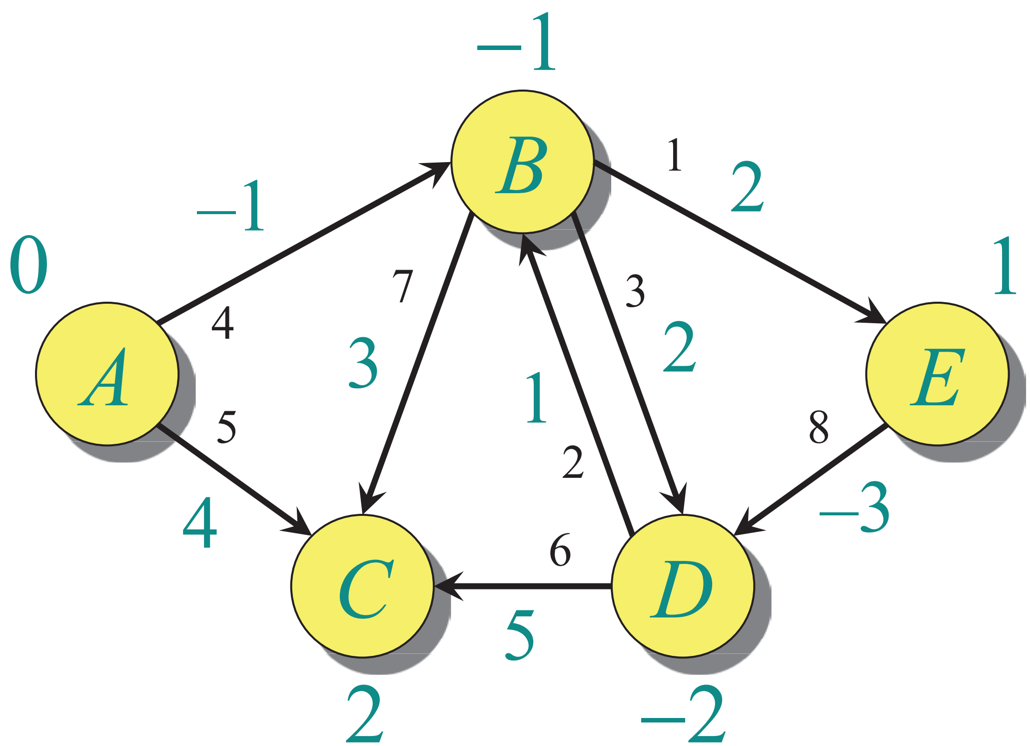
Example of Bellman-Ford



 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

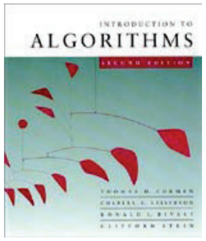


Example of Bellman-Ford



End of pass 2 (and 3 and 4).

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Correctness

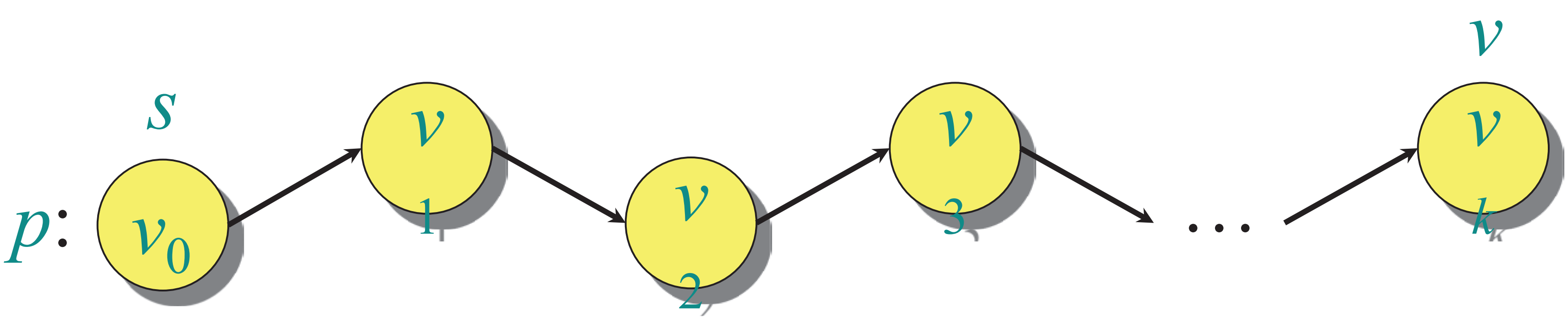
Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Correctness

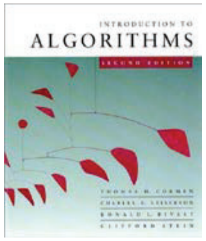
Theorem. If $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$.

Proof. Let $v \in V$ be any vertex, and consider a shortest path p from s to v with the minimum number of edges.

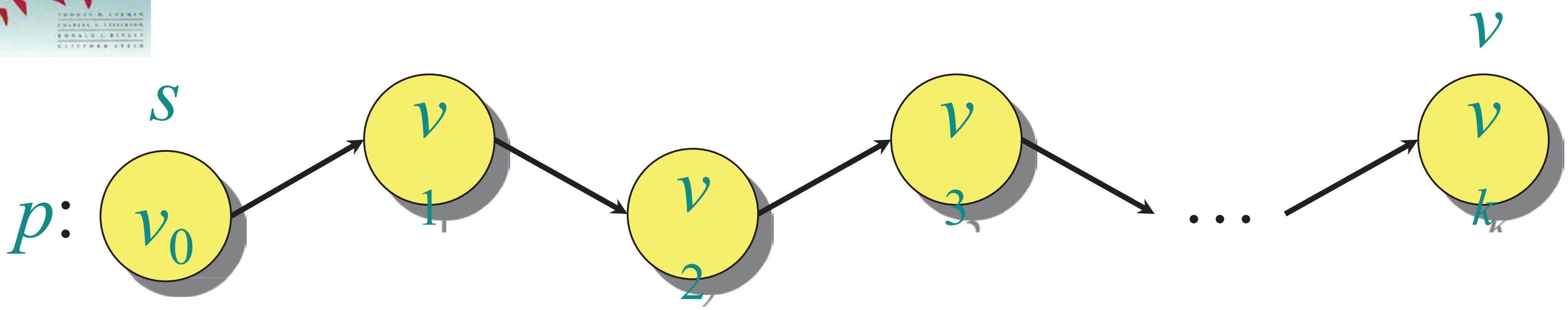


Since p is a shortest path, we have

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i) \,.$$



Correctness (continued)



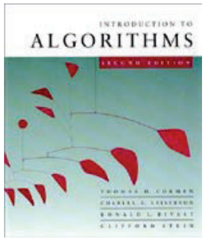
Initially, $d[v_0] = 0 = \delta(s, v_0)$, and $d[v_0]$ is unchanged by subsequent relaxations (because of the lemma from *Shortest Paths I* that $d[v] \geq \delta(s, v)$).

- After 1 pass through E, we have $d[v_1] = \delta(s, v_1)$.
- After 2 passes through E, we have $d[v_2] = \delta(s, v_2)$.

  ⋮
- After k passes through E, we have $d[v_k] = \delta(s, v_k)$.

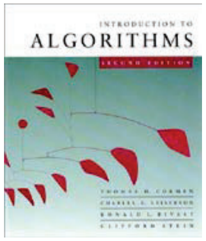
Since G contains no negative-weight cycles, p is simple. Longest simple path has $\leq |V| - 1$ edges. ◻

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Detection of negative-weight cycles

Corollary. If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle in G reachable from s. $\square$

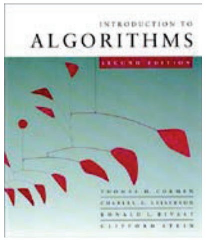


Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford algorithm: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Shortest paths

Single-source shortest paths

- Nonnegative edge weights
  - Dijkstra's algorithm: $O(E + V \lg V)$
- General
  - Bellman-Ford algorithm: $O(VE)$
- DAG
  - One pass of Bellman-Ford: $O(V + E)$

All-pairs shortest paths

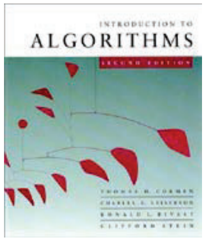
- Nonnegative edge weights
  - Dijkstra's algorithm $|V|$ times: $O(VE + V^2 \lg V)$
- General
  - Three algorithms today.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.



All-pairs shortest paths

Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

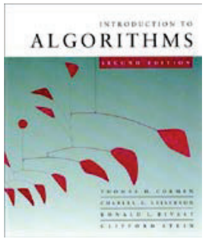
Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

IDEA:

- Run Bellman-Ford once from each vertex.
- Time $= O(V^2E)$.
- Dense graph ($\Theta(n^2)$ edges) $\Rightarrow$ $\Theta(n^4)$ time in the worst case.

Good first try!

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Dynamic programming

Consider the $n \times n$ weighted adjacency matrix $A = (a_{ij})$, where $a_{ij} = w(i, j)$ or ∞, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

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

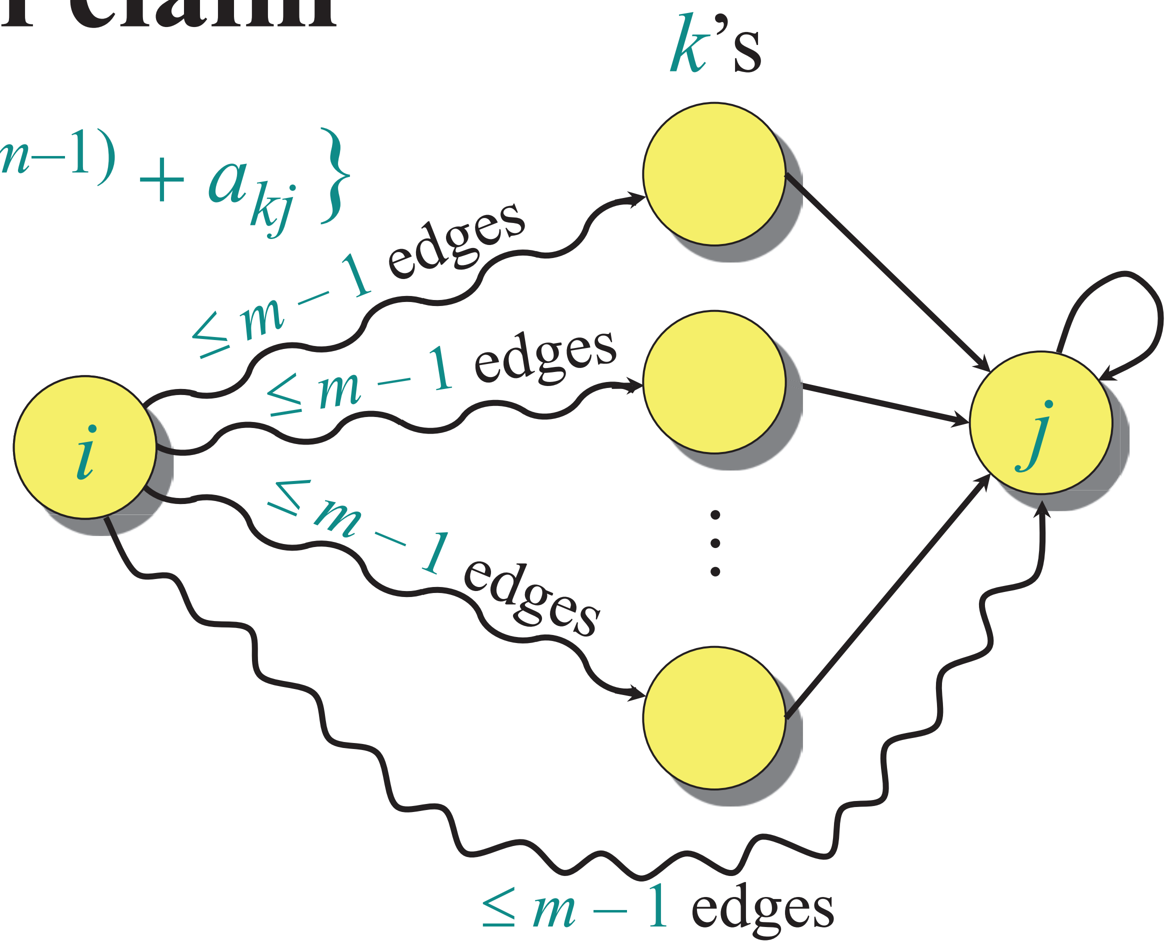
and for $m = 1, 2, \ldots, n - 1$,

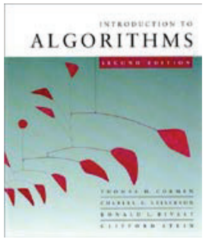
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Proof of claim

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$





Proof of claim

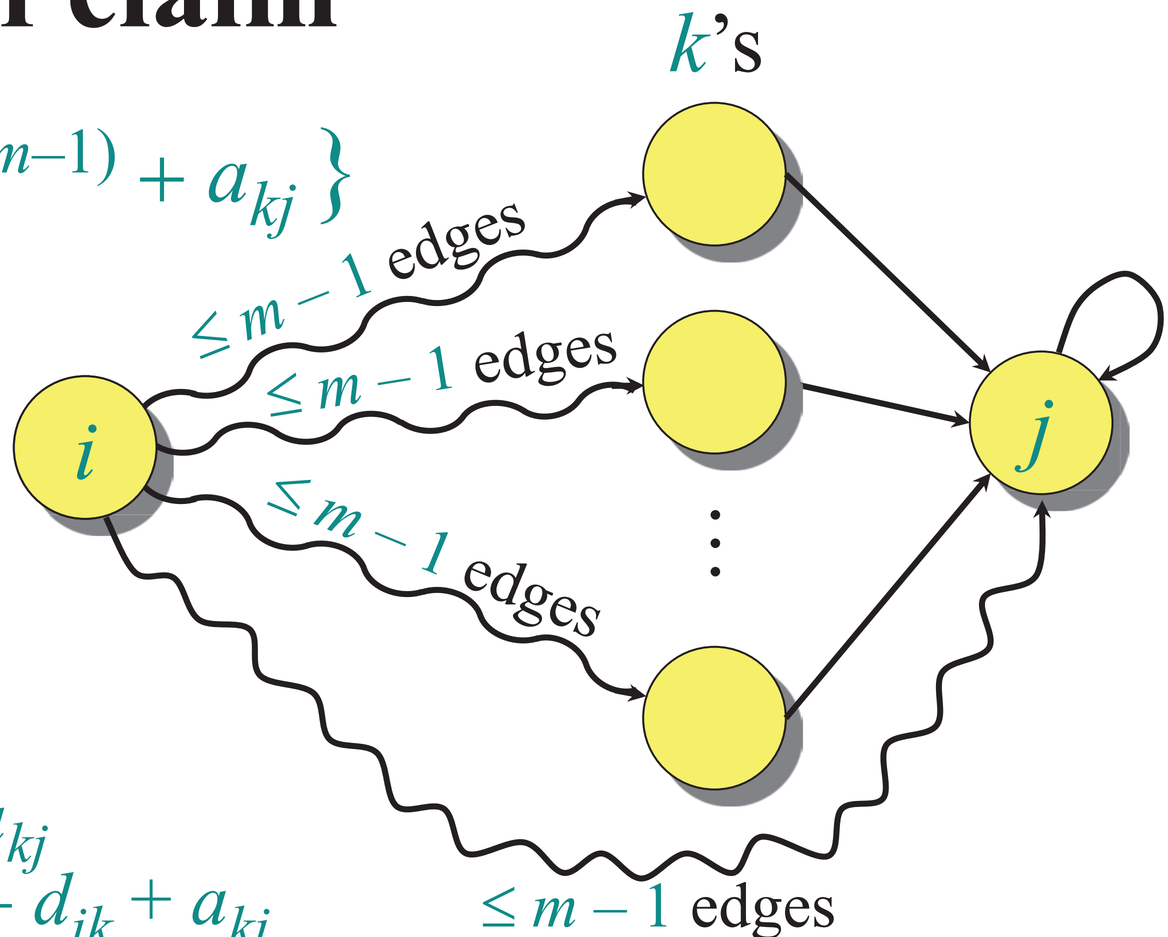
$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$

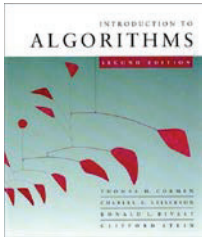
Relaxation!

for $k \leftarrow 1$ **to** n

  do if $d_{ij} > d_{ik} + a_{kj}$

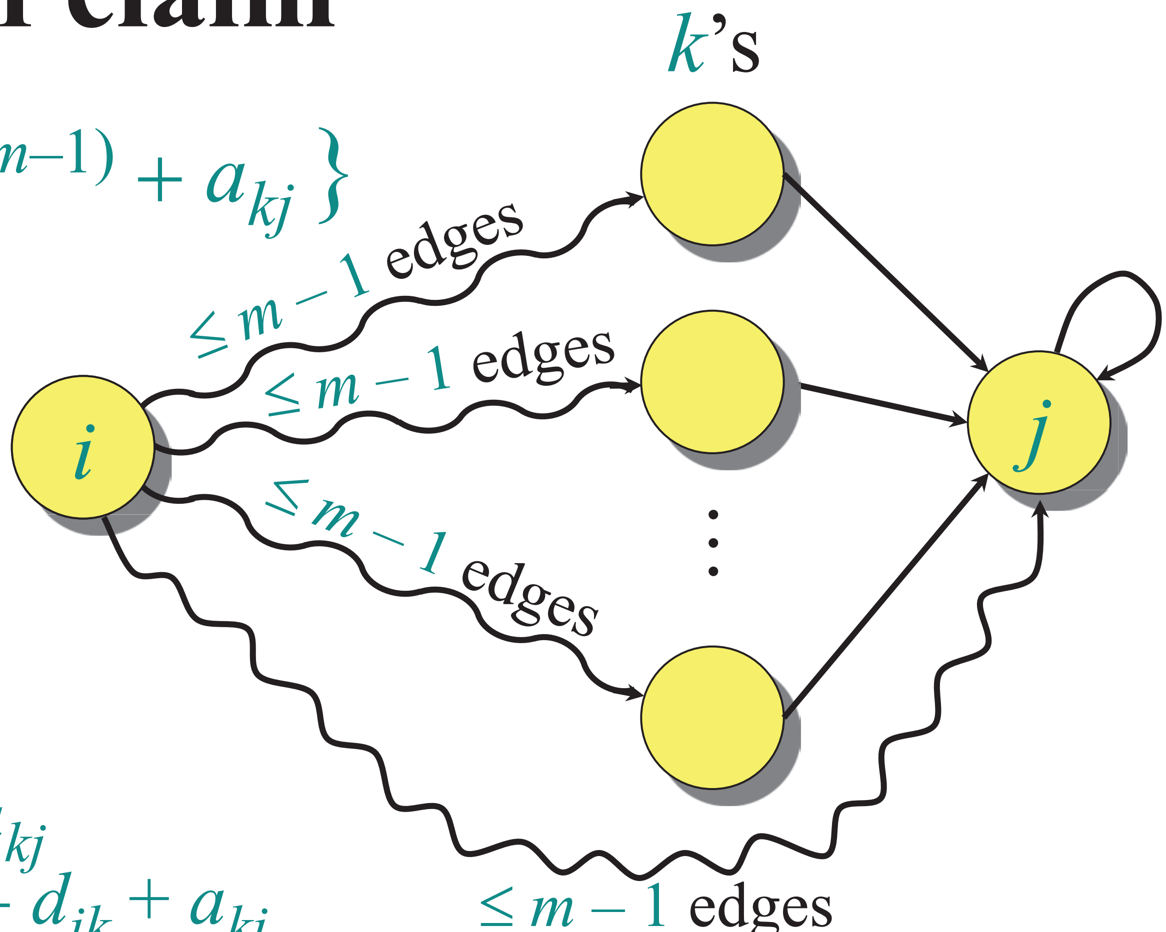
    then $d_{ij} \leftarrow d_{ik} + a_{kj}$





Proof of claim

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$



Relaxation!

for $k \leftarrow 1$ **to** n
    do if $d_{ij} > d_{ik} + a_{kj}$
        then $d_{ij} \leftarrow d_{ik} + a_{kj}$

Note: No negative-weight cycles implies

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \ldots$$

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

Time $= \Theta(n^3)$ using the standard algorithm.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson

Matrix multiplication

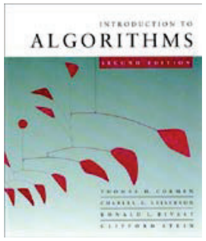
Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

Time $= \Theta(n^3)$ using the standard algorithm.

What if we map "$+$" $\rightarrow$ "min" and "$\cdot$" $\rightarrow$ "$+$"?

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \,.$$

Time $= \Theta(n^3)$ using the standard algorithm.

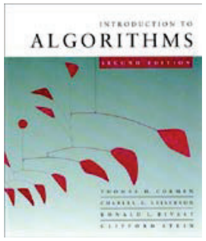
What if we map "$+$" $\to$ "min" and "$\cdot$" $\to$ "$+$"?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}.$$

Thus, $D^{(m)} = D^{(m-1)}$ "$\times$" A.

$$\text{Identity matrix} = \mathrm{I} = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}{}^{(0)}).$$

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Matrix multiplication (continued)

The (min, +) multiplication is ***associative***, and with the real numbers, it forms an algebraic structure called a ***closed semiring***.

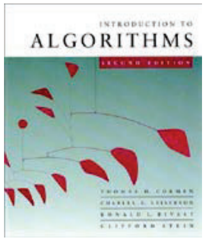
Consequently, we can compute

$$
\begin{aligned}
D^{(1)} &= D^{(0)} \cdot A = A^1 \\
D^{(2)} &= D^{(1)} \cdot A = A^2 \\
&\ \ \vdots \qquad\qquad\quad \vdots \\
D^{(n-1)} &= D^{(n-2)} \cdot A = A^{n-1},
\end{aligned}
$$

yielding $D^{(n-1)} = (\delta(i, j))$.

Time $= \Theta(n \cdot n^3) = \Theta(n^4)$. No better than $n \times$ B-F.

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Improved matrix multiplication algorithm

Repeated squaring: $A^{2k} = A^k \times A^k$.

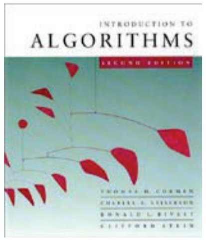
Compute $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n) \text{ squarings}}$.

Note: $A^{n-1} = A^n = A^{n+1} = \cdots$.

Time $= \Theta(n^3 \lg n)$.

To detect negative-weight cycles, check the diagonal for negative values in $O(n)$ additional time.

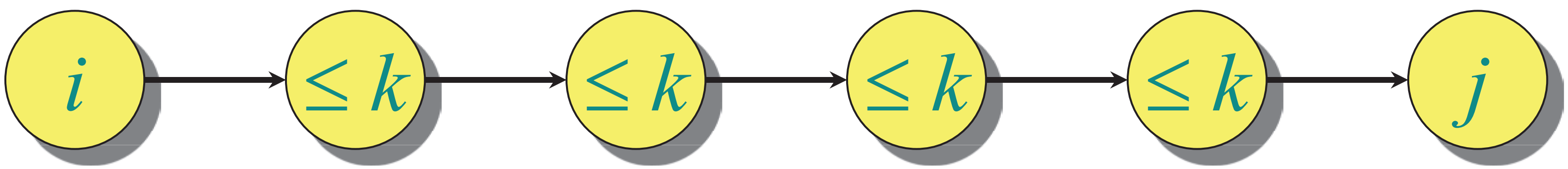
 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



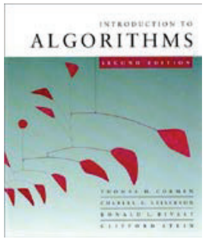
Floyd-Warshall algorithm

Also dynamic programming, but faster!

Define $c_{ij}^{(k)}$ = weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$.

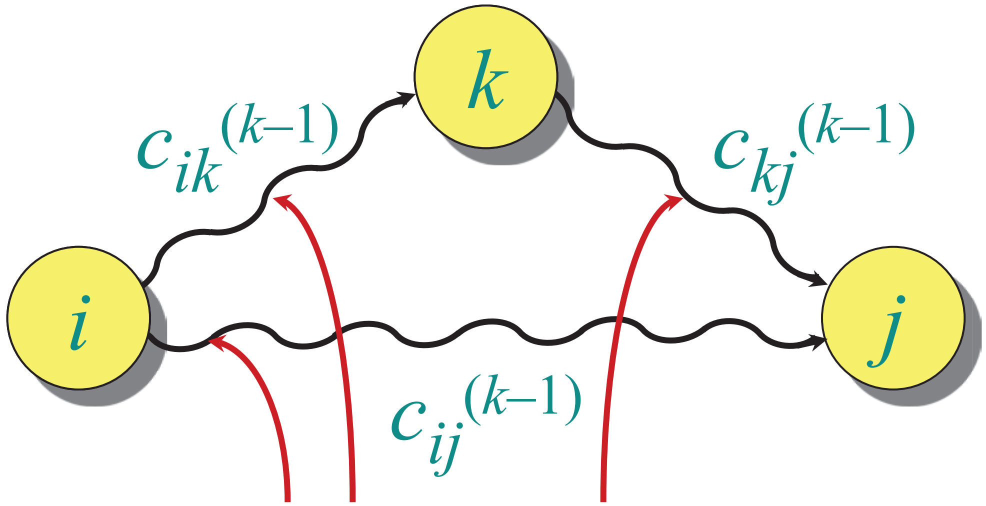


Thus, $\delta(i, j) = c_{ij}^{(n)}$. Also, $c_{ij}^{(0)} = a_{ij}$.

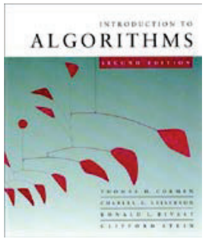


Floyd-Warshall recurrence

$$c_{ij}^{(k)} = \min \{c_{ij}^{(k-1)},\ c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$



intermediate vertices in $\{1, 2, \ldots, k-1\}$



Pseudocode for Floyd-Warshall

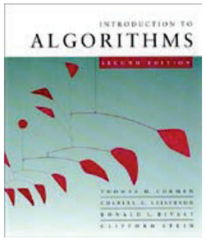
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do if $c_{ij} > c_{ik} + c_{kj}$
        then $c_{ij} \leftarrow c_{ik} + c_{kj}$

$$\left.\begin{array}{l} \textbf{if } c_{ij} > c_{ik} + c_{kj} \\ \textbf{then } c_{ij} \leftarrow c_{ik} + c_{kj} \end{array}\right\} \textit{relaxation}$$

Notes:

- Okay to omit superscripts, since extra relaxations can't hurt.
- Runs in $\Theta(n^3)$ time.
- Simple to code.
- Efficient in practice.

 Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson



Transitive closure of a directed graph

Compute $t_{ij} = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$

IDEA: Use Floyd-Warshall, but with $(\vee, \wedge)$ instead of $(\min, +)$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}).$$

Time $= \Theta(n^3)$.