

## 6. DYNAMIC PROGRAMMING II

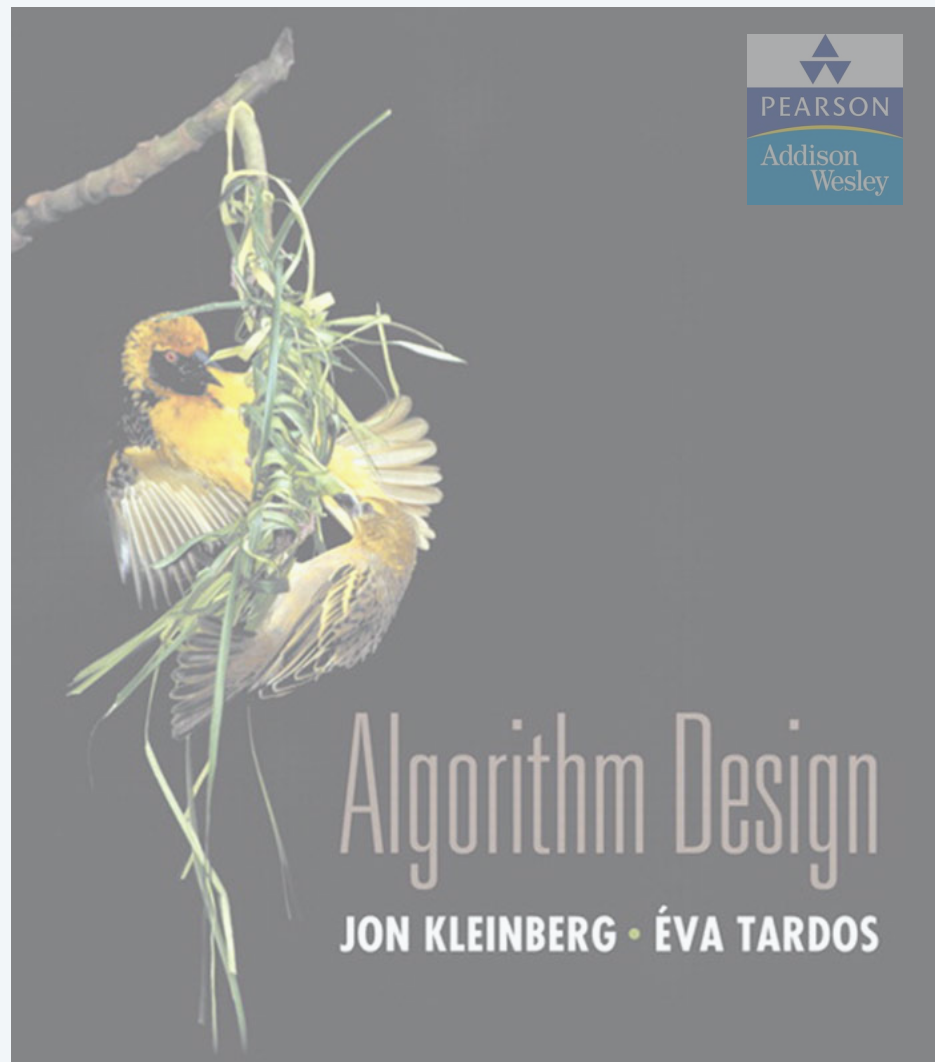
---

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*
- ▶ *distance-vector protocols*
- ▶ *negative cycles*

Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>



## SECTION 6.8

# 6. DYNAMIC PROGRAMMING II

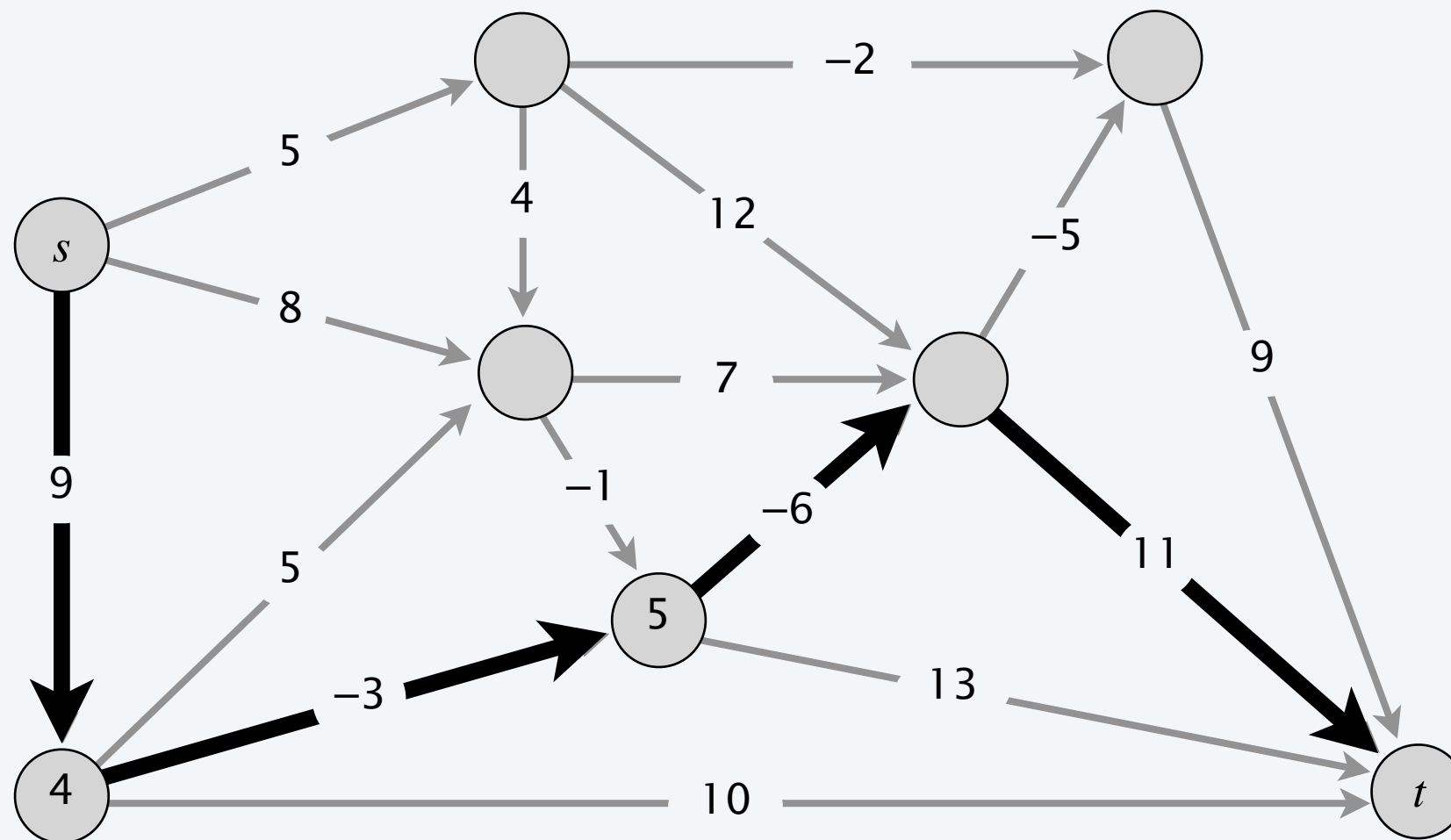
---

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ ***Bellman–Ford–Moore algorithm***
- ▶ *distance-vector protocols*
- ▶ *negative cycles*

# Shortest paths with negative weights

**Shortest-path problem.** Given a digraph  $G = (V, E)$ , with arbitrary edge lengths  $\ell_{vw}$ , find shortest path from source node  $s$  to destination node  $t$ .

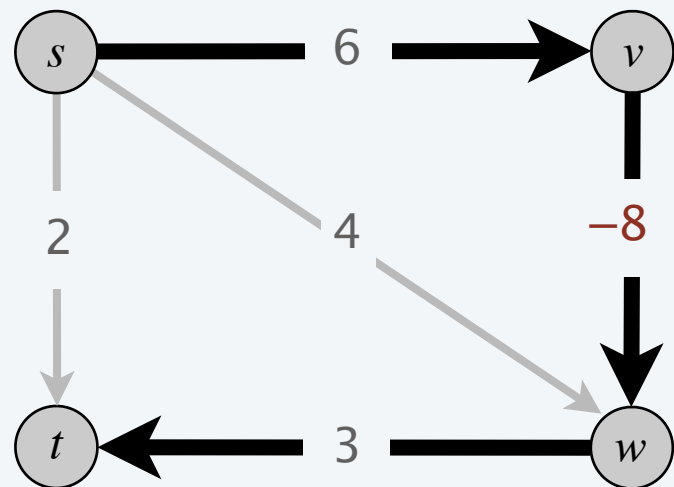
assume there exists a path  
from every node to  $t$



length of shortest path from  $s$  to  $t = 9 - 3 - 6 + 11 = 11$

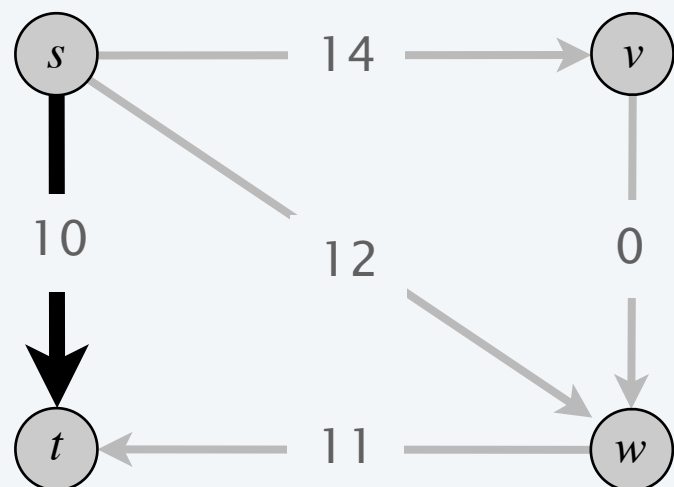
# Shortest paths with negative weights: failed attempts

**Dijkstra.** May not produce shortest paths when edge lengths are negative.



Dijkstra selects the vertices in the order  $s, t, w, v$   
But shortest path from  $s$  to  $t$  is  $s \rightarrow v \rightarrow w \rightarrow t$ .

**Reweighting.** Adding a constant to every edge length does not necessarily make Dijkstra's algorithm produce shortest paths.

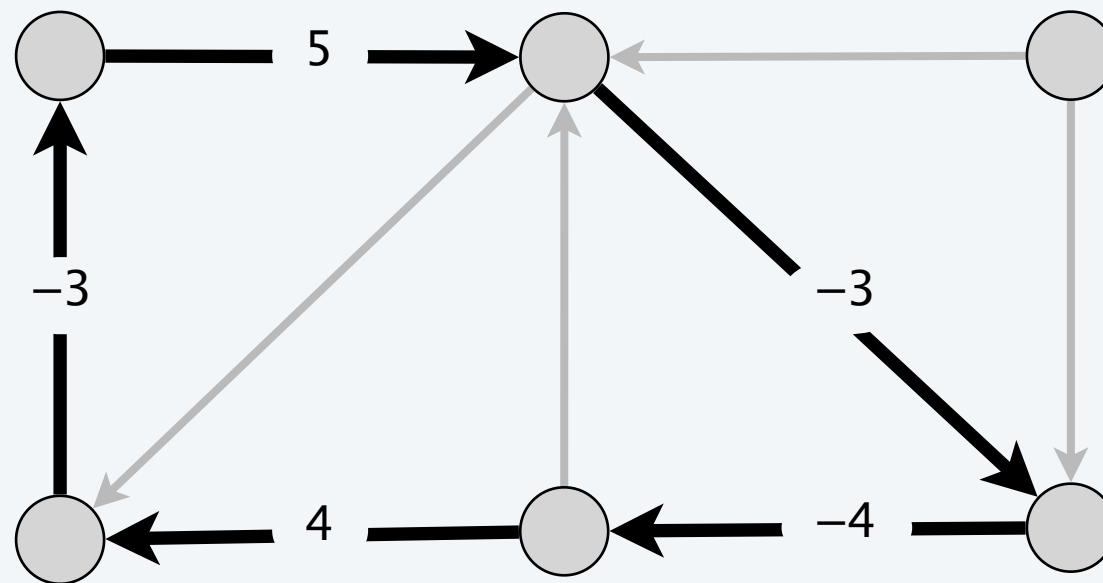


Adding 8 to each edge weight changes the shortest path from  $s \rightarrow v \rightarrow w \rightarrow t$  to  $s \rightarrow t$ .

# Negative cycles

---

**Def.** A **negative cycle** is a directed cycle for which the sum of its edge lengths is negative.



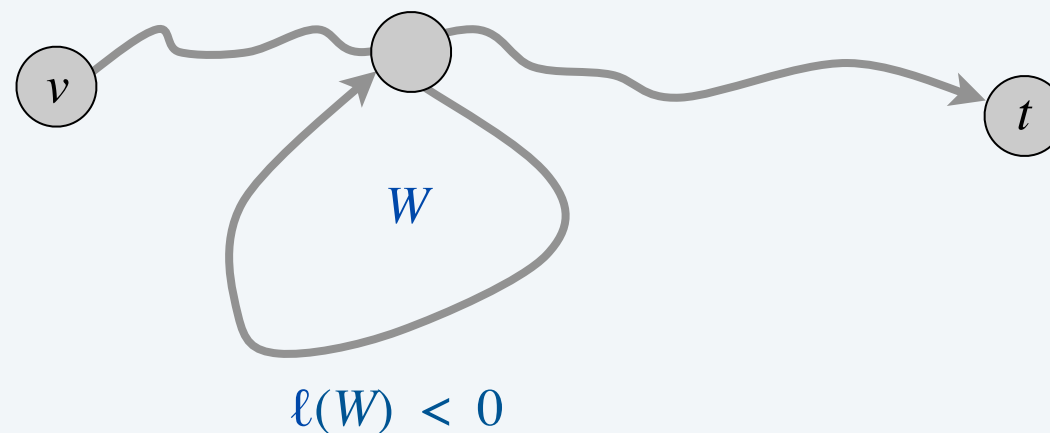
a negative cycle  $W$ :  $\ell(W) = \sum_{e \in W} \ell_e < 0$

# Shortest paths and negative cycles

---

**Lemma 1.** If some  $v \rightsquigarrow t$  path contains a negative cycle, then there does not exist a shortest  $v \rightsquigarrow t$  path.

**Pf.** If there exists such a cycle  $W$ , then can build a  $v \rightsquigarrow t$  path of arbitrarily negative length by detouring around  $W$  as many times as desired. ■



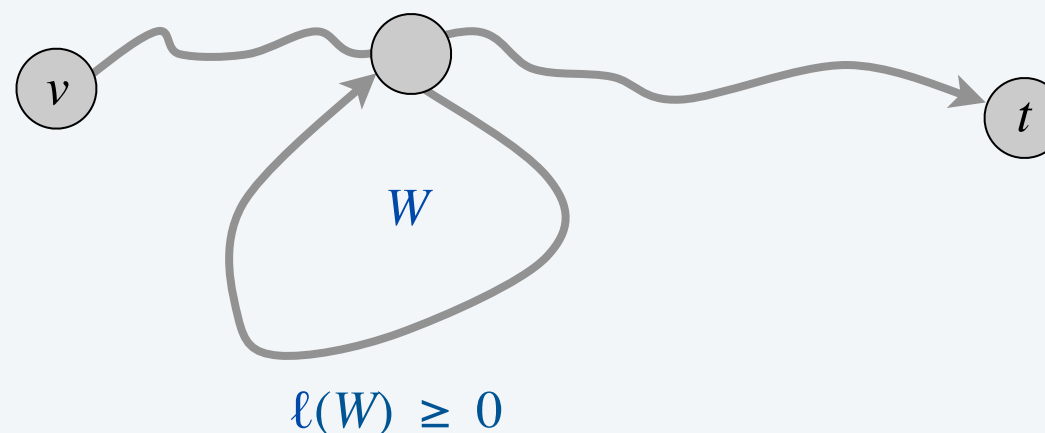
# Shortest paths and negative cycles

---

**Lemma 2.** If  $G$  has no negative cycles, then there exists a shortest  $v \rightsquigarrow t$  path that is simple (and has  $\leq n - 1$  edges).

**Pf.**

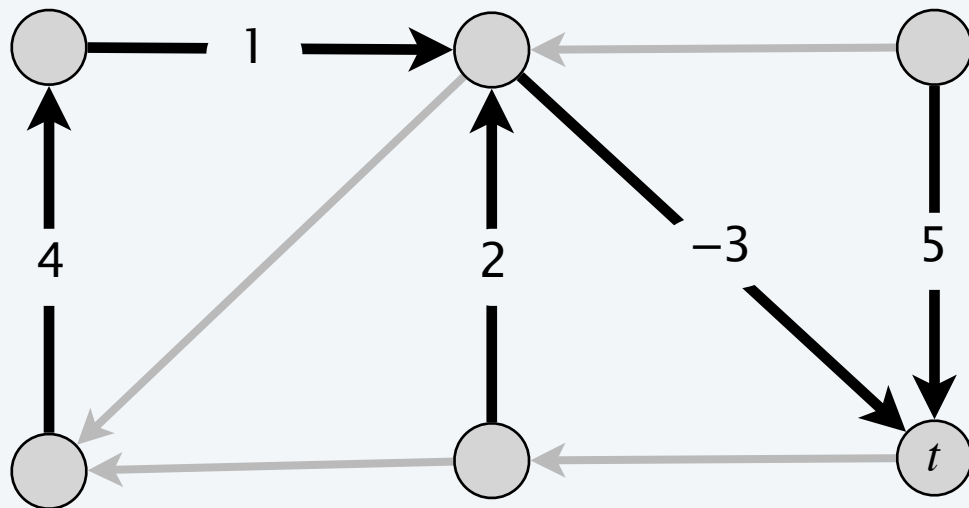
- Among all shortest  $v \rightsquigarrow t$  paths, consider one that uses the fewest edges.
- If that path  $P$  contains a directed cycle  $W$ , can remove the portion of  $P$  corresponding to  $W$  without increasing its length. ▀



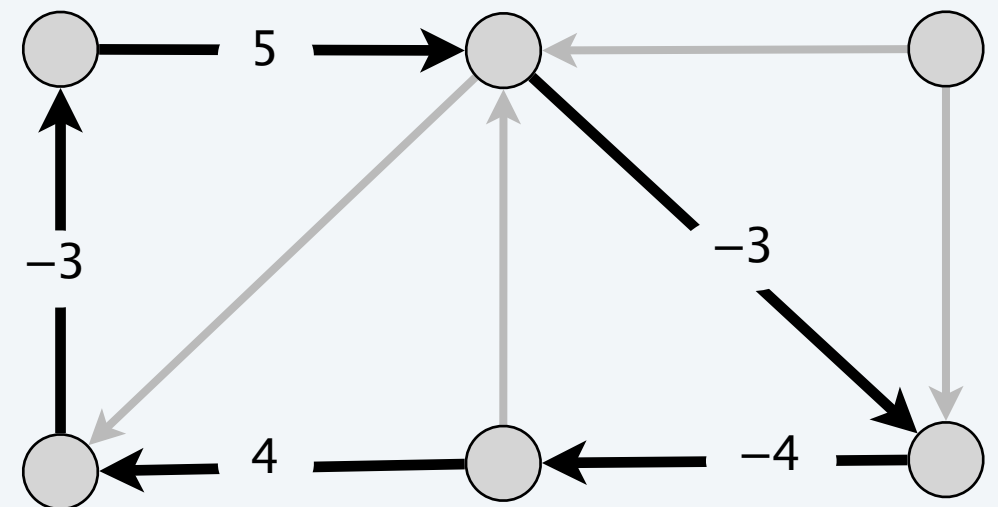
# Shortest-paths and negative-cycle problems

**Single-destination shortest-paths problem.** Given a digraph  $G = (V, E)$  with edge lengths  $\ell_{vw}$  (but no negative cycles) and a distinguished node  $t$ , find a shortest  $v \rightsquigarrow t$  path for every node  $v$ .

**Negative-cycle problem.** Given a digraph  $G = (V, E)$  with edge lengths  $\ell_{vw}$ , find a negative cycle (if one exists).



shortest-paths tree



negative cycle



# Shortest paths with negative weights: dynamic programming


---

**Def.**  $OPT(i, v)$  = length of shortest  $v \rightsquigarrow t$  path that uses  $\leq i$  edges.

**Goal.**  $OPT(n - 1, v)$  for each  $v$ .  by Lemma 2, if no negative cycles,  
there exists a shortest  $v \rightsquigarrow t$  path that is simple

**Case 1.** Shortest  $v \rightsquigarrow t$  path uses  $\leq i - 1$  edges.

- $OPT(i, v) = OPT(i - 1, v)$ .

 optimal substructure property  
(proof via exchange argument)

**Case 2.** Shortest  $v \rightsquigarrow t$  path uses exactly  $i$  edges.

- if  $(v, w)$  is first edge in shortest such  $v \rightsquigarrow t$  path, incur a cost of  $\ell_{vw}$ .
- Then, select best  $w \rightsquigarrow t$  path using  $\leq i - 1$  edges.

**Bellman equation.**

$$OPT(i, v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = t \\ \infty & \text{if } i = 0 \text{ and } v \neq t \\ \min \left\{ OPT(i - 1, v), \min_{(v, w) \in E} \{OPT(i - 1, w) + \ell_{vw}\} \right\} & \text{if } i > 0 \end{cases}$$

# Shortest paths with negative weights: implementation

SHORTEST-PATHS( $V, E, \ell, t$ )

FOREACH node  $v \in V$ :

$$M[0, v] \leftarrow \infty.$$

$$M[0, t] \leftarrow 0.$$

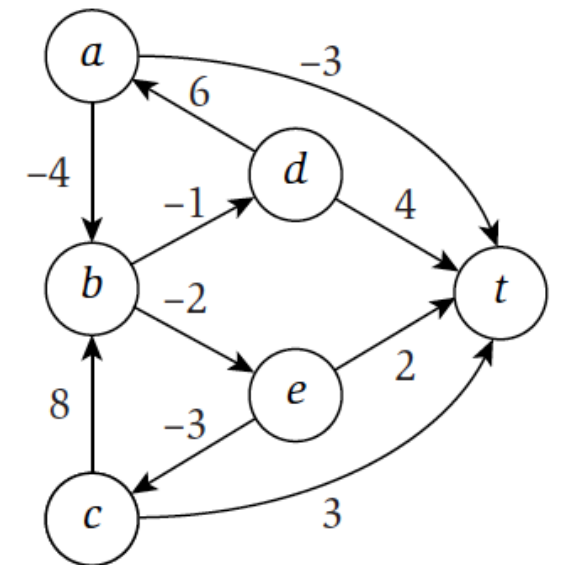
FOR  $i = 1$  TO  $n - 1$

FOREACH node  $v \in V$ :

$$M[i, v] \leftarrow M[i-1, v].$$

FOREACH edge  $(v, w) \in E$ :

$$M[i, v] \leftarrow \min \{ M[i, v], M[i-1, w] + \ell_{vw} \}.$$



(a)

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	$\infty$	-3	-3	-4	-6	-6
b	$\infty$	$\infty$	0	-2	-2	-2
c	$\infty$	3	3	3	3	3
d	$\infty$	4	3	3	2	0
e	$\infty$	2	0	0	0	0

(b)

**Figure 6.23** For the directed graph in (a), the Shortest-Path Algorithm constructs the dynamic programming table in (b).

# Shortest paths with negative weights: implementation

---

**Theorem 1.** Given a digraph  $G = (V, E)$  with no negative cycles, the DP algorithm computes the length of a shortest  $v \rightsquigarrow t$  path for every node  $v$  in  $\Theta(mn)$  time and  $\Theta(n^2)$  space.

**Pf.**

- Table requires  $\Theta(n^2)$  space.
- Each iteration  $i$  takes  $\Theta(m)$  time since we examine each edge once. ■

**Finding the shortest paths.**

- Approach 1: Maintain  $successor[i, v]$  that points to next node on a shortest  $v \rightsquigarrow t$  path using  $\leq i$  edges.
- Approach 2: Compute optimal lengths  $M[i, v]$  and consider only edges with  $M[i, v] = M[i - 1, w] + \ell_{vw}$ . Any directed path in this subgraph is a shortest path.



It is easy to modify the DP algorithm for shortest paths to...

- A. Compute lengths of shortest paths in  $O(mn)$  time and  $O(m + n)$  space.
- B. Compute shortest paths in  $O(mn)$  time and  $O(m + n)$  space.
- C. Both A and B.
- D. Neither A nor B.

# Shortest paths with negative weights: practical improvements

---

**Space optimization.** Maintain two 1D arrays (instead of 2D array).

- $d[v]$  = length of a shortest  $v \rightsquigarrow t$  path that we have found so far.
- $successor[v]$  = next node on a  $v \rightsquigarrow t$  path.

**Performance optimization.** If  $d[w]$  was not updated in iteration  $i - 1$ , then no reason to consider edges entering  $w$  in iteration  $i$ .

# Bellman–Ford–Moore: efficient implementation

---

BELLMAN–FORD–MOORE( $V, E, c, t$ )

---

FOREACH node  $v \in V$ :

$d[v] \leftarrow \infty$ .

$successor[v] \leftarrow null$ .

$d[t] \leftarrow 0$ .

FOR  $i = 1$  TO  $n - 1$

FOREACH node  $w \in V$ :

IF ( $d[w]$  was updated in previous pass)

FOREACH edge  $(v, w) \in E$ :

IF ( $d[v] > d[w] + \ell_{vw}$ )

$d[v] \leftarrow d[w] + \ell_{vw}$ .

$successor[v] \leftarrow w$ .

IF (no  $d[\cdot]$  value changed in pass  $i$ ) STOP.

---

pass  $i$   
 $O(m)$  time



Which properties must hold after pass  $i$  of Bellman–Ford–Moore?

- A.  $d[v]$  = length of a shortest  $v \rightsquigarrow t$  path using  $\leq i$  edges.
- B.  $d[v]$  = length of a shortest  $v \rightsquigarrow t$  path using exactly  $i$  edges.
- C. Both A and B.
- D. Neither A nor B.

# Bellman–Ford–Moore: analysis

---

**Lemma 3.** For each node  $v$  :  $d[v]$  is the length of some  $v \rightsquigarrow t$  path.

**Lemma 4.** For each node  $v$  :  $d[v]$  is monotone non-increasing.

**Lemma 5.** After pass  $i$ ,  $d[v] \leq$  length of a shortest  $v \rightsquigarrow t$  path using  $\leq i$  edges.

**Pf.** [ by induction on  $i$  ]

- Base case:  $i = 0$ .
- Assume true after pass  $i$ .
- Let  $P$  be any  $v \rightsquigarrow t$  path with  $\leq i + 1$  edges.
- Let  $(v, w)$  be first edge in  $P$  and let  $P'$  be subpath from  $w$  to  $t$ .
- By inductive hypothesis, at the end of pass  $i$ ,  $d[w] \leq c(P')$   
because  $P'$  is a  $w \rightsquigarrow t$  path with  $\leq i$  edges.
- After considering edge  $(v, w)$  in pass  $i + 1$ :

and by Lemma 4,  
 $d[w]$  does not increase

$$\begin{aligned} d[v] &\leq \ell_{vw} + d[w] \\ &\leq \ell_{vw} + c(P') \\ &= \ell(P) \quad \blacksquare \end{aligned}$$

and by Lemma 4,  
 $d[v]$  does not increase



# Bellman–Ford–Moore: analysis

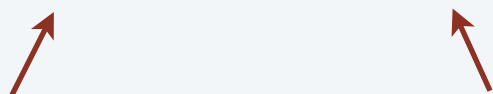
---

**Theorem 2.** Assuming no negative cycles, Bellman–Ford–Moore computes the lengths of the shortest  $v \rightsquigarrow t$  paths in  $O(mn)$  time and  $\Theta(n)$  extra space.

**Pf.** Lemma 2 + Lemma 5. ■

shortest path exists and  
has at most  $n-1$  edges

after  $i$  passes,  
 $d[v] \leq$  length of shortest path  
that uses  $\leq i$  edges



**Remark.** Bellman–Ford–Moore is typically faster in practice.

- Edge  $(v, w)$  considered in pass  $i + 1$  only if  $d[w]$  updated in pass  $i$ .
- If shortest path has  $k$  edges, then algorithm finds it after  $\leq k$  passes.



Assuming no negative cycles, which properties must hold throughout Bellman-Ford-Moore?

- A. Following  $successor[v]$  pointers gives a directed  $v \rightsquigarrow t$  path.
- B. If following  $successor[v]$  pointers gives a directed  $v \rightsquigarrow t$  path, then the length of that  $v \rightsquigarrow t$  path is  $d[v]$ .
- C. Both A and B.
- D. Neither A nor B.

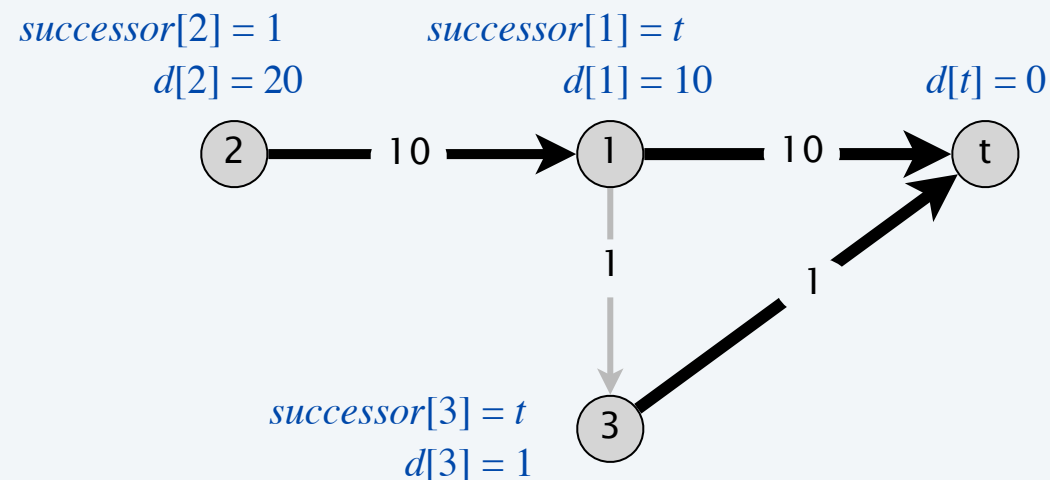
## Bellman–Ford–Moore: analysis

**Claim.** Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].

**Counterexample.** Claim is false!

- Length of successor  $v \rightsquigarrow t$  path may be strictly shorter than  $d[v]$ .

consider nodes in order: *t*, 1, 2, 3



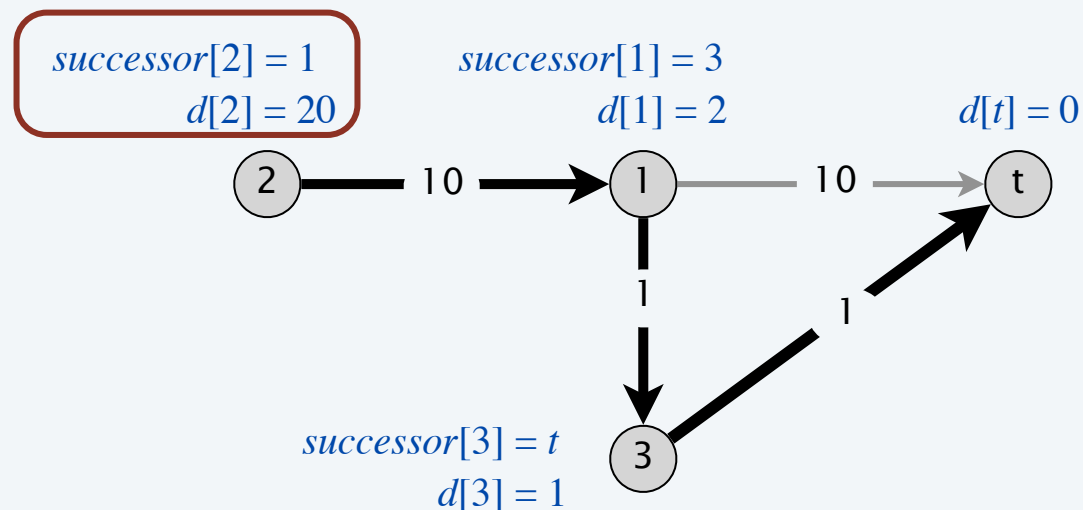
## Bellman-Ford-Moore: analysis

**Claim.** Throughout Bellman-Ford-Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].

**Counterexample.** Claim is false!

- Length of successor  $v \rightsquigarrow t$  path may be strictly shorter than  $d[v]$ .

consider nodes in order: *t*, 1, 2, 3



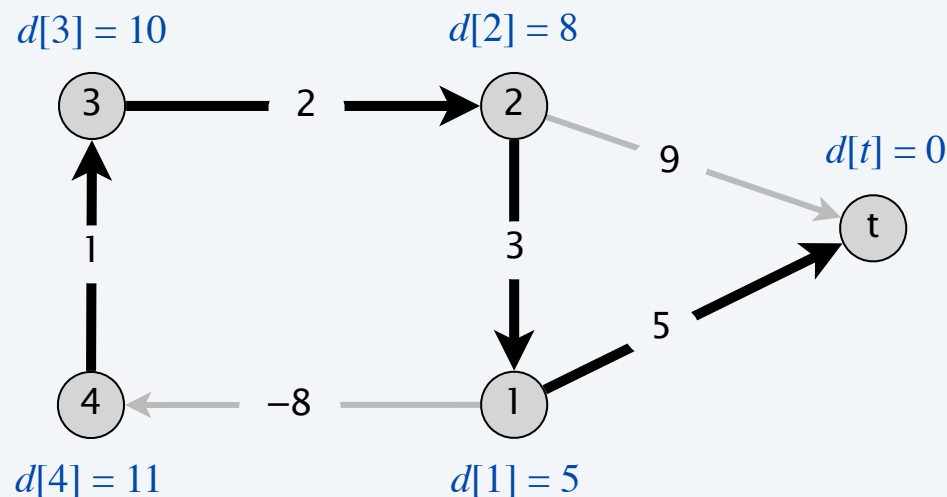
# Bellman-Ford-Moore: analysis

**Claim.** Throughout Bellman-Ford-Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length  $d[v]$ .

**Counterexample.** Claim is false!

- Length of successor  $v \rightsquigarrow t$  path may be strictly shorter than  $d[v]$ .
- If negative cycle, successor graph may have directed cycles.

consider nodes in order: *t*, 1, 2, 3, 4



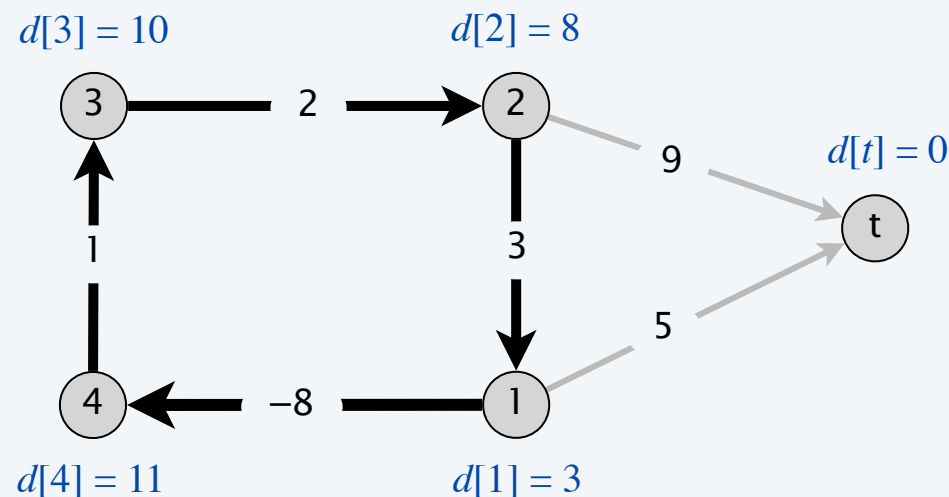
## Bellman-Ford-Moore: analysis

**Claim.** Throughout Bellman-Ford-Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length  $d[v]$ .

Counterexample. Claim is false!

- Length of successor  $v \rightsquigarrow t$  path may be strictly shorter than  $d[v]$ .
- If negative cycle, successor graph may have directed cycles.

**consider nodes in order: t, 1, 2, 3, 4**



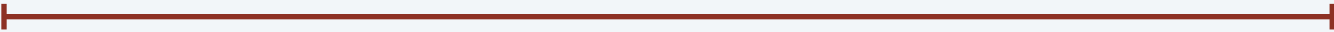
# Bellman–Ford–Moore: finding the shortest paths

---

**Lemma 6.** Any directed cycle  $W$  in the successor graph is a negative cycle.  
**Pf.**

- If  $\text{successor}[v] = w$ , we must have  $d[v] \geq d[w] + \ell_{vw}$ .  
(LHS and RHS are equal when  $\text{successor}[v]$  is set;  $d[w]$  can only decrease;  $d[v]$  decreases only when  $\text{successor}[v]$  is reset)
- Let  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$  be the sequence of nodes in a directed cycle  $W$ .
- Assume that  $(v_k, v_1)$  is the last edge in  $W$  added to the successor graph.
- Just prior to that:
$$\begin{array}{rclcl} d[v_1] & \geq & d[v_2] & + & \ell(v_1, v_2) \\ d[v_2] & \geq & d[v_3] & + & \ell(v_2, v_3) \\ \vdots & & \vdots & & \vdots \\ d[v_{k-1}] & \geq & d[v_k] & + & \ell(v_{k-1}, v_k) \\ d[v_k] & > & d[v_1] & + & \ell(v_k, v_1) \end{array}$$

← holds with strict inequality  
since we are updating  $d[v_k]$
- Adding inequalities yields  $\ell(v_1, v_2) + \ell(v_2, v_3) + \dots + \ell(v_{k-1}, v_k) + \ell(v_k, v_1) < 0$ . ■  

  
 $W$  is a negative cycle

# Bellman–Ford–Moore: finding the shortest paths

---

**Theorem 3.** Assuming no negative cycles, Bellman–Ford–Moore finds shortest  $v \rightsquigarrow t$  paths for every node  $v$  in  $O(mn)$  time and  $\Theta(n)$  extra space.

**Pf.**

- The successor graph cannot have a directed cycle. [Lemma 6]
- Thus, following the successor pointers from  $v$  yields a directed path to  $t$ .
- Let  $v = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k = t$  be the nodes along this path  $P$ .
- Upon termination, if  $\text{successor}[v] = w$ , we must have  $d[v] = d[w] + \ell_{vw}$ .  
(LHS and RHS are equal when  $\text{successor}[v]$  is set;  $d[\cdot]$  did not change)

• Thus,

$$\begin{array}{rcl} d[v_1] & = & d[v_2] + \ell(v_1, v_2) \\ d[v_2] & = & d[v_3] + \ell(v_2, v_3) \\ \vdots & & \vdots \\ d[v_{k-1}] & = & d[v_k] + \ell(v_{k-1}, v_k) \end{array}$$

since algorithm  
terminated

- Adding equations yields  $d[v] = d[t] + \ell(v_1, v_2) + \ell(v_2, v_3) + \dots + \ell(v_{k-1}, v_k)$ . ■

min length of any  $v \rightsquigarrow t$  path  
(Theorem 2)

0

length of path  $P$

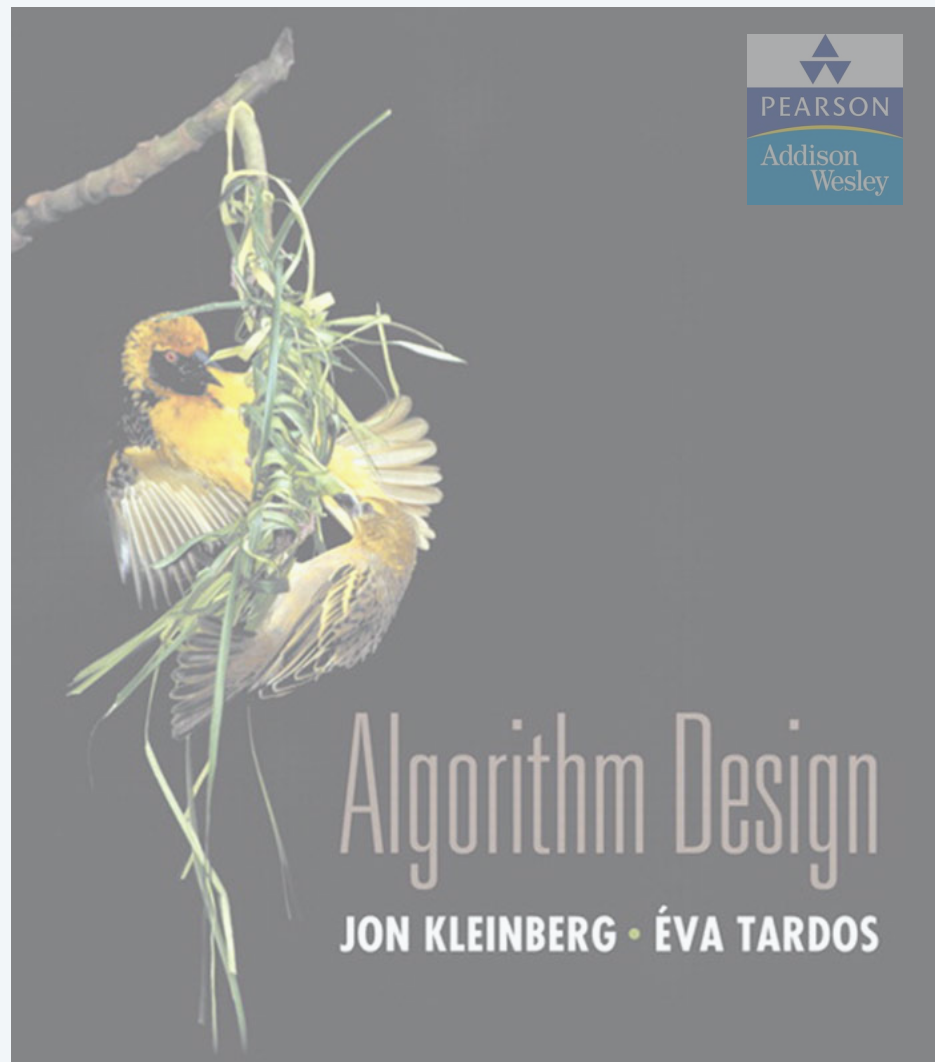


# Single-source shortest paths with negative weights

---

year	worst case	discovered by
1955	$O(n^4)$	Shimbel
1956	$O(m n^2 W)$	Ford
1958	$O(m n)$	Bellman, Moore
1983	$O(n^{3/4} m \log W)$	Gabow
1989	$O(m n^{1/2} \log(nW))$	Gabow–Tarjan
1993	$O(m n^{1/2} \log W)$	Goldberg
2005	$O(n^{2.38} W)$	Sankowski, Yuster–Zwick
2016	$\tilde{O}(n^{10/7} \log W)$	Cohen–Mądry–Sankowski–Vladu
20xx	???	

single-source shortest paths with weights between  $-W$  and  $W$



## SECTION 6.9

# 6. DYNAMIC PROGRAMMING II


---

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*
- ▶ ***distance-vector protocols***
- ▶ *negative cycles*

# Distance-vector routing protocols

---

## Communication network.

- Node  $\approx$  router.
- Edge  $\approx$  direct communication link.
- Length of edge  $\approx$  latency of link.  non-negative, but Bellman–Ford–Moore used anyway!

**Dijkstra's algorithm.** Requires global information of network.

**Bellman–Ford–Moore.** Uses only local knowledge of neighboring nodes.

**Synchronization.** We don't expect routers to run in lockstep. The order in which each edges are processed in Bellman–Ford–Moore is not important. Moreover, algorithm converges even if updates are asynchronous.

# Distance-vector routing protocols

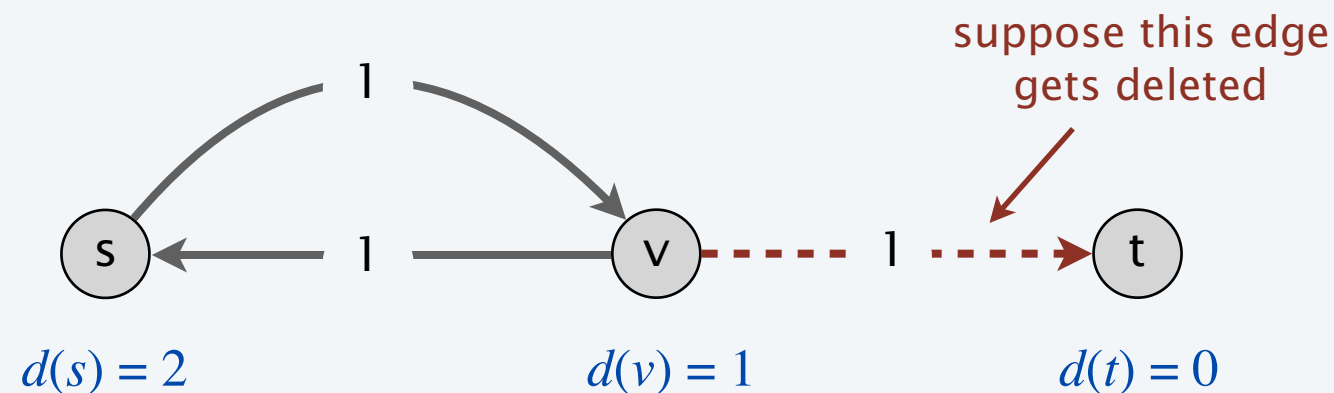
---

## Distance-vector routing protocols. [ “routing by rumor” ]

- Each router maintains a vector of shortest-path lengths to every other node (distances) and the first hop on each path (directions).
- Algorithm: each router performs  $n$  separate computations, one for each potential destination node.

**Ex.** RIP, Xerox XNS RIP, Novell’s IPX RIP, Cisco’s IGRP, DEC’s DNA Phase IV, AppleTalk’s RTMP.

**Caveat.** Edge lengths may **change** during algorithm (or fail completely).




“counting to infinity”

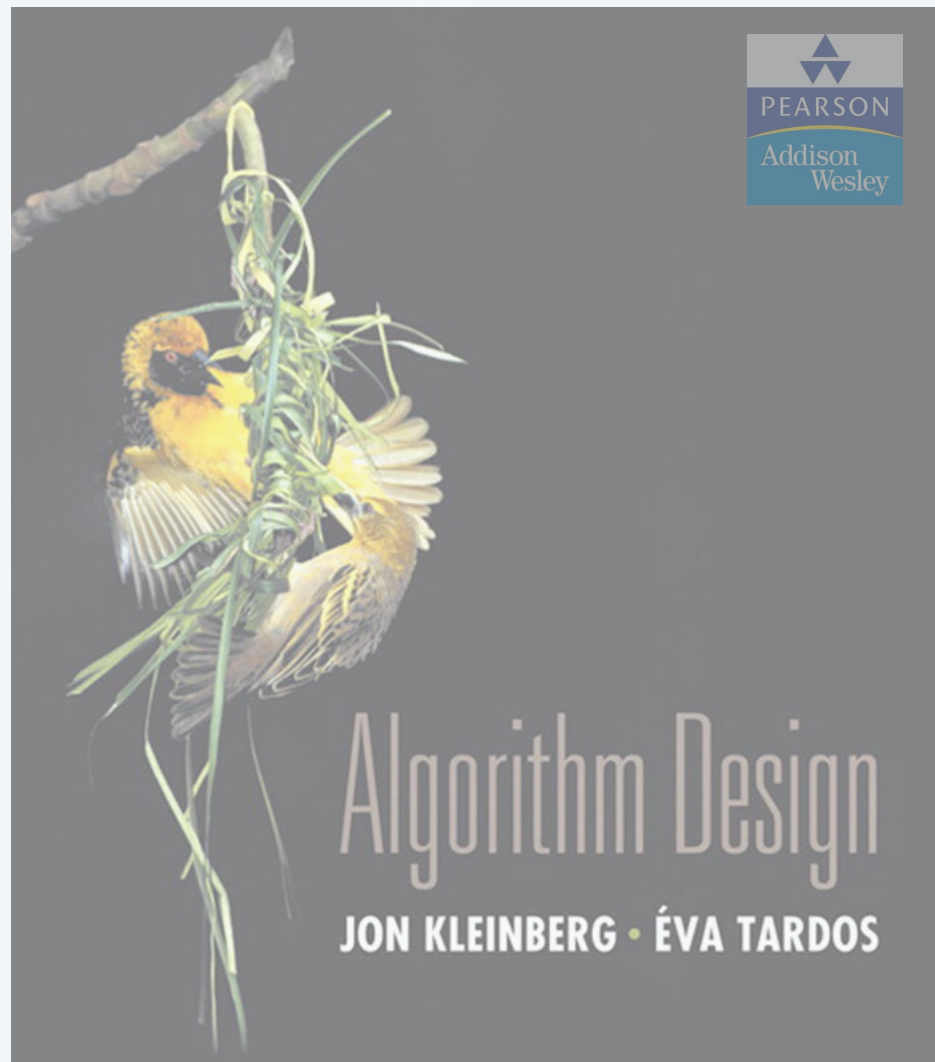
# Path-vector routing protocols

---

## Link-state routing protocols.

- Each router stores the whole network topology.
  - Based on Dijkstra's algorithm.
  - Avoids “counting-to-infinity” problem and related difficulties.
  - Requires significantly more storage.
- 
- not just the distance  
and first hop

**Ex.** Border Gateway Protocol (BGP), Open Shortest Path First (OSPF).



## SECTION 6.10

# 6. DYNAMIC PROGRAMMING II

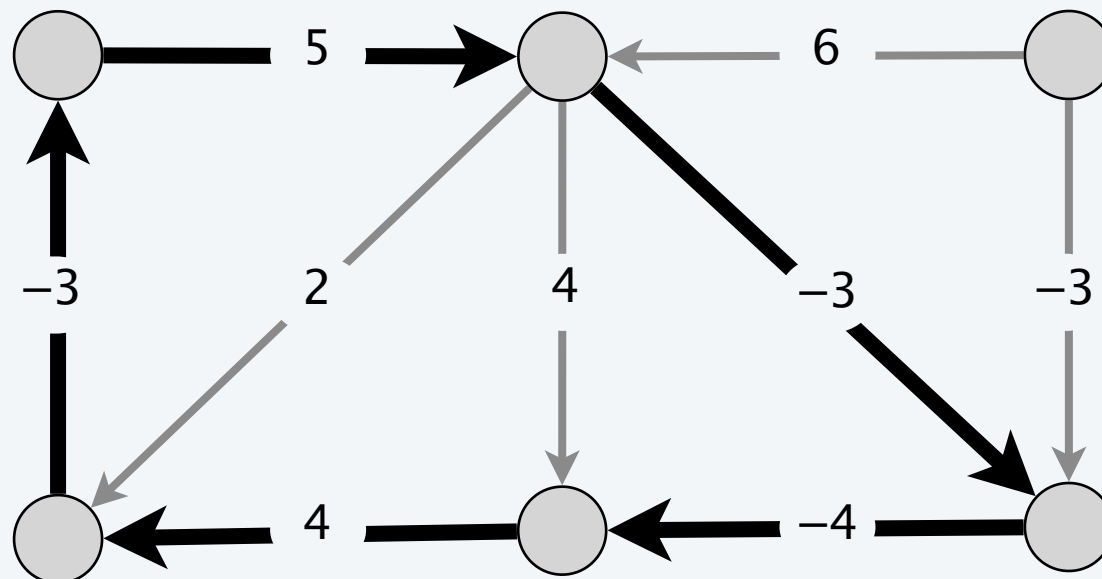
---

- ▶ *sequence alignment*
- ▶ *Hirschberg's algorithm*
- ▶ *Bellman–Ford–Moore algorithm*
- ▶ *distance vector protocol*
- ▶ ***negative cycles***

# Detecting negative cycles

---

**Negative cycle detection problem.** Given a digraph  $G = (V, E)$ , with edge lengths  $\ell_{vw}$ , find a negative cycle (if one exists).

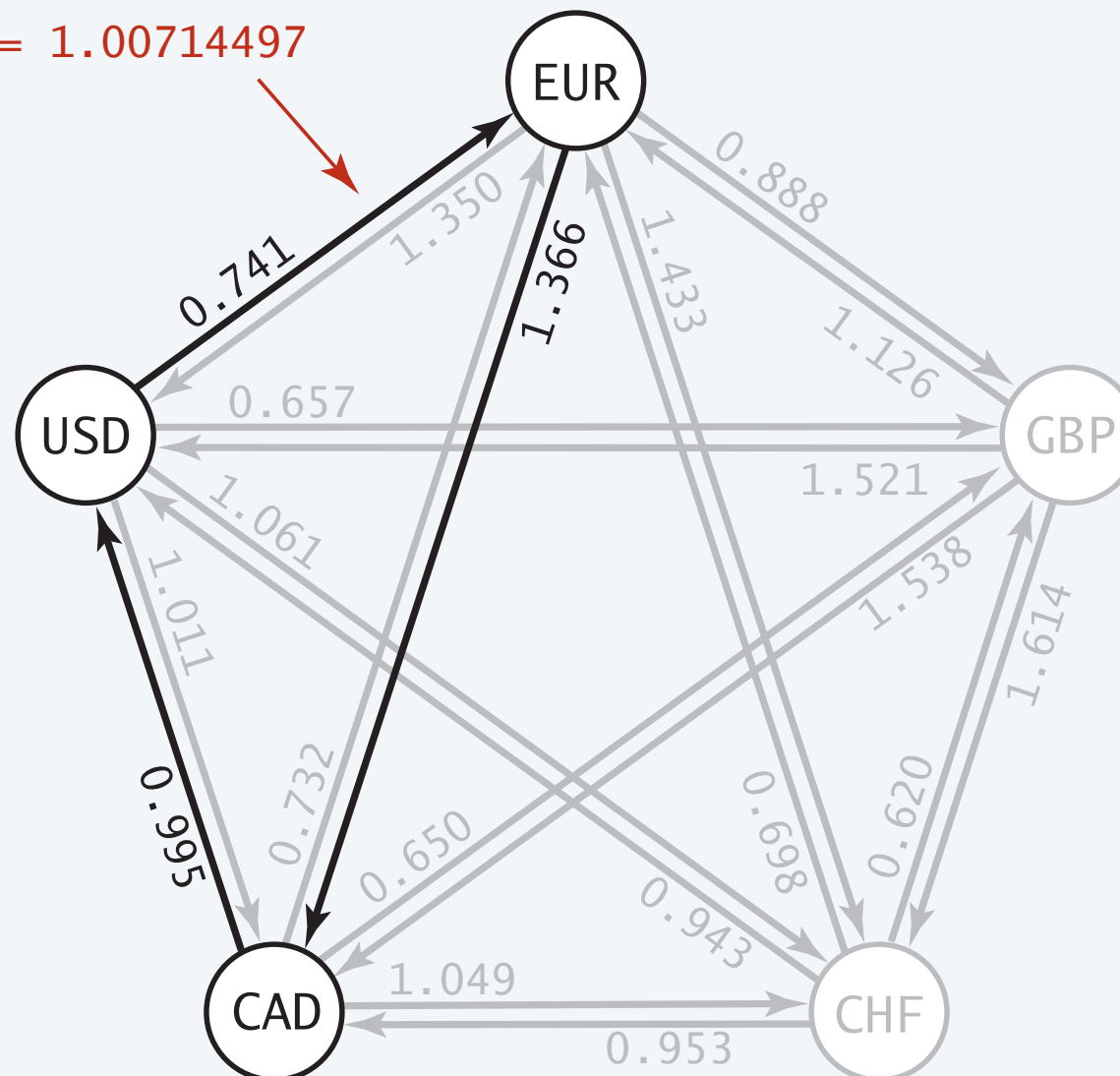


# Detecting negative cycles: application

**Currency conversion.** Given  $n$  currencies and exchange rates between pairs of currencies, is there an arbitrage opportunity?

**Remark.** Fastest algorithm very valuable!

$$0.741 * 1.366 * .995 = 1.00714497$$





# Detecting negative cycles

---

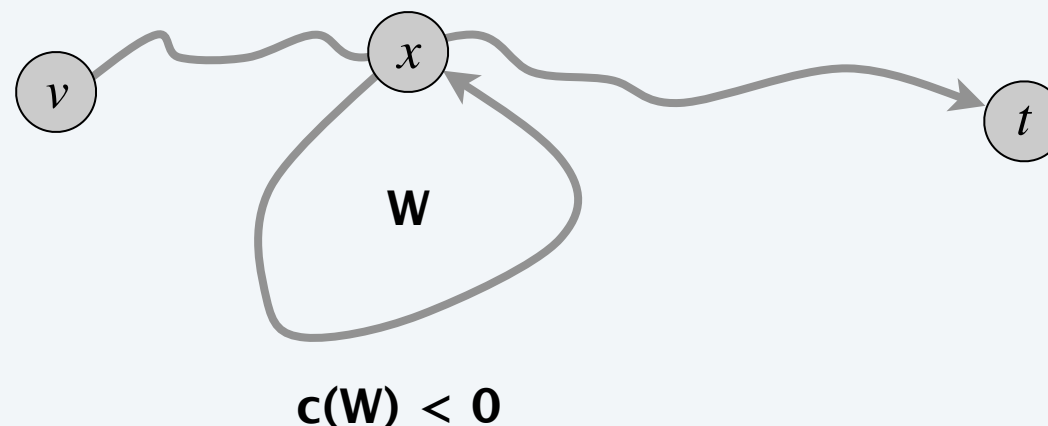
**Lemma 7.** If  $OPT(n, v) = OPT(n - 1, v)$  for every node  $v$ , then no negative cycles.

**Pf.** The  $OPT(n, v)$  values have converged  $\Rightarrow$  shortest  $v \rightsquigarrow t$  path exists. ■

**Lemma 8.** If  $OPT(n, v) < OPT(n - 1, v)$  for some node  $v$ , then (any) shortest  $v \rightsquigarrow t$  path of length  $\leq n$  contains a cycle  $W$ . Moreover  $W$  is a negative cycle.

**Pf.** [by contradiction]

- Since  $OPT(n, v) < OPT(n - 1, v)$ , we know that shortest  $v \rightsquigarrow t$  path  $P$  has exactly  $n$  edges.
- By pigeonhole principle, the path  $P$  must contain a repeated node  $x$ .
- Let  $W$  be any cycle in  $P$ .
- Deleting  $W$  yields a  $v \rightsquigarrow t$  path with  $< n$  edges  $\Rightarrow W$  is a negative cycle. ■



# Detecting negative cycles

**Theorem 4.** Can find a negative cycle in  $\Theta(mn)$  time and  $\Theta(n^2)$  space.

**Pf.**

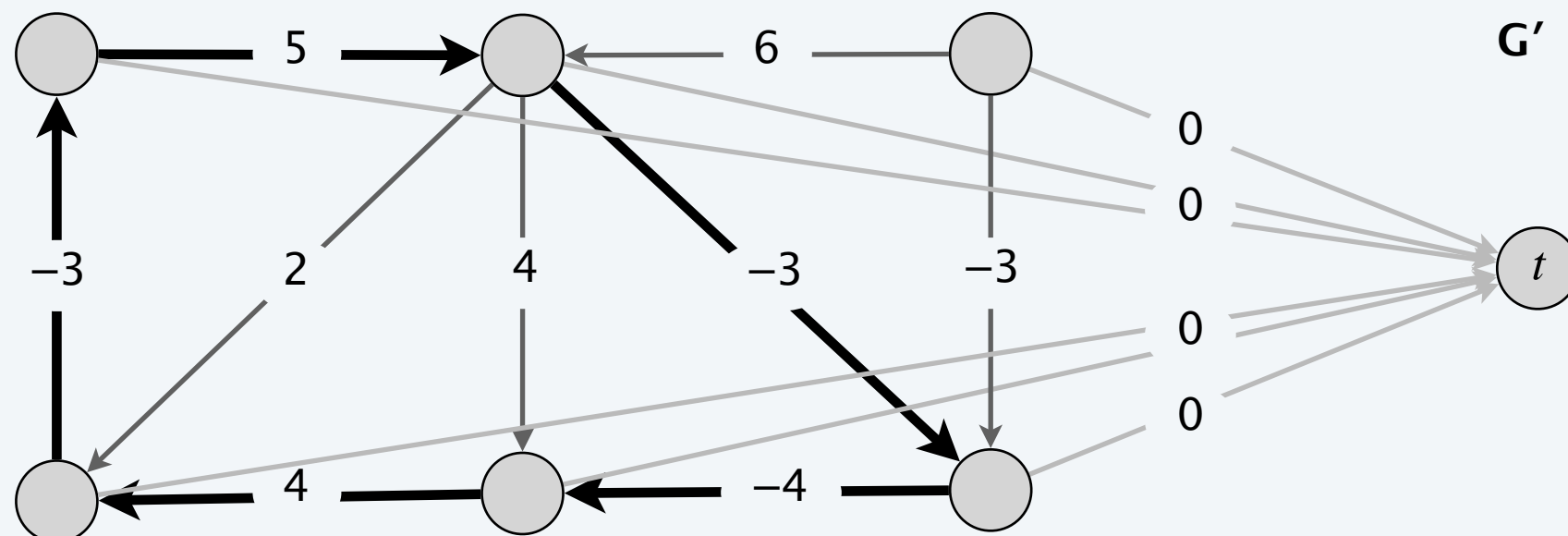
- Add new sink node  $t$  and connect all nodes to  $t$  with 0-length edge.
- $G$  has a negative cycle iff  $G'$  has a negative cycle.
- Case 1. [  $OPT(n, v) = OPT(n - 1, v)$  for every node  $v$  ]

By Lemma 7, no negative cycles.

- Case 2. [  $OPT(n, v) < OPT(n - 1, v)$  for some node  $v$  ]

Using proof of Lemma 8, can extract negative cycle from  $v \rightsquigarrow t$  path.

(cycle cannot contain  $t$  since no edge leaves  $t$ ) ■



# Detecting negative cycles

---

**Theorem 5.** Can find a negative cycle in  $O(mn)$  time and  $O(n)$  extra space.

**Pf.**

- Run Bellman–Ford–Moore on  $G'$  for  $n' = n + 1$  passes (instead of  $n' - 1$ ).
- If no  $d[v]$  values updated in pass  $n'$ , then no negative cycles.
- Otherwise, suppose  $d[s]$  updated in pass  $n'$ .
- Define  $pass(v) =$  last pass in which  $d[v]$  was updated.
- Observe  $pass(s) = n'$  and  $pass(successor[v]) \geq pass(v) - 1$  for each  $v$ .
- Following successor pointers, we must eventually repeat a node.
- Lemma 6  $\Rightarrow$  the corresponding cycle is a negative cycle. ■

**Remark.** See p. 304 for improved version and early termination rule.

(Tarjan's subtree disassembly trick)



How difficult to find a negative cycle in an undirected graph? 

- A.  $O(m \log n)$
- B.  $O(mn)$
- C.  $O(mn + n^2 \log n)$
- D.  $O(n^{2.38})$
- E. No poly-time algorithm is known.