南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Instructor: Shan CHEN (陈杉)

chens3@sustech.edu.cn

(slides edited from Prof. Shiqi Yu)

Dynamic Programming

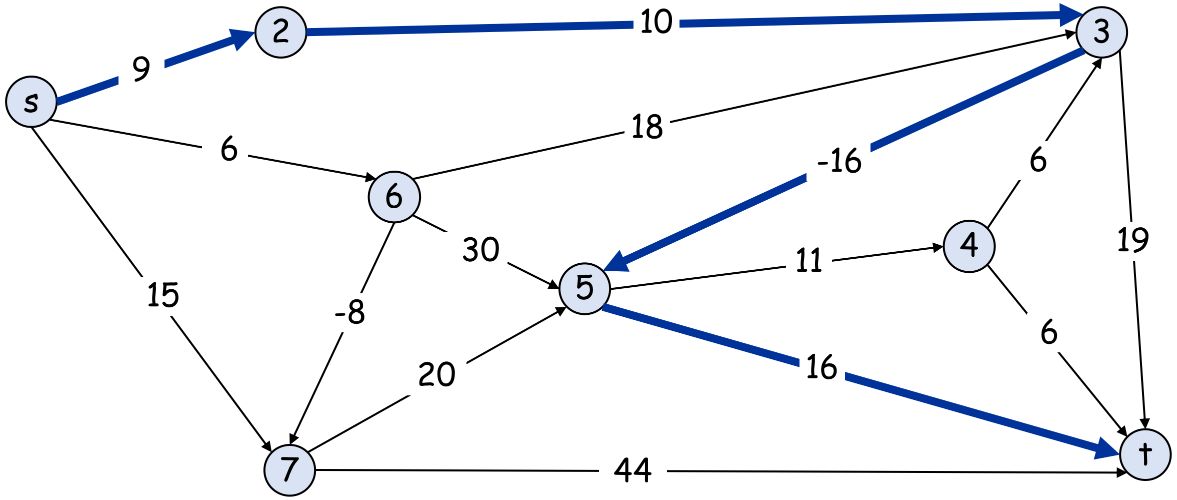
6. Shortest Paths with Negative Weights

Shortest Paths with Negative Weights

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

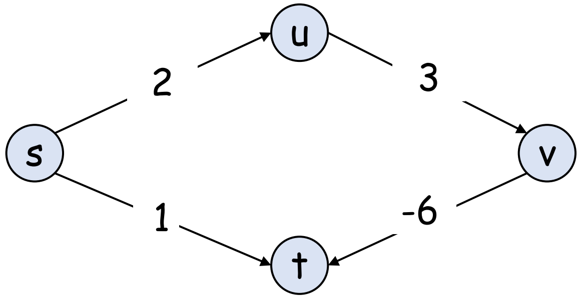
assume there exits a path from every node to t

- **Ex.** Nodes represent agents in a financial setting and c_{vw} is cost of transaction in which we buy from agent v and sell immediately to w.

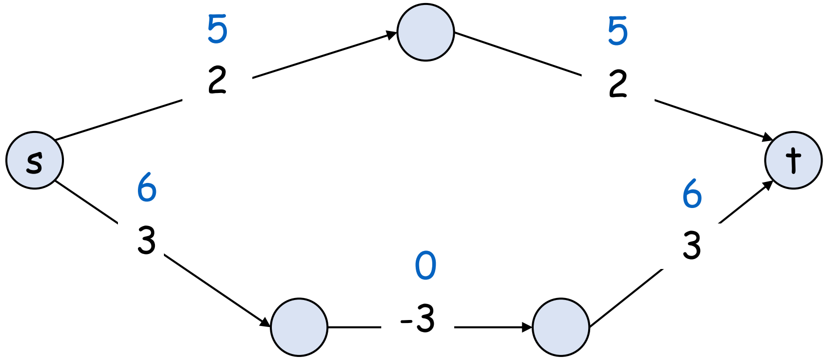


Shortest Paths: Failed Attempts

- **Dijkstra.** Can fail if there exist negative edge weights.

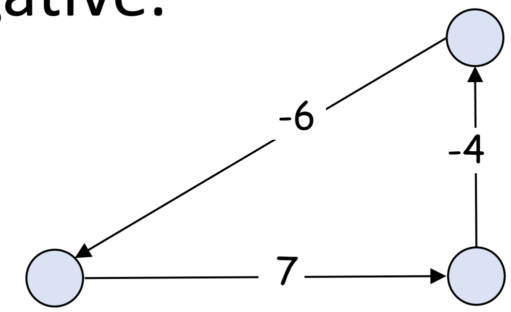


- **Re-weighting.** Adding a constant to every edge weight can still fail.

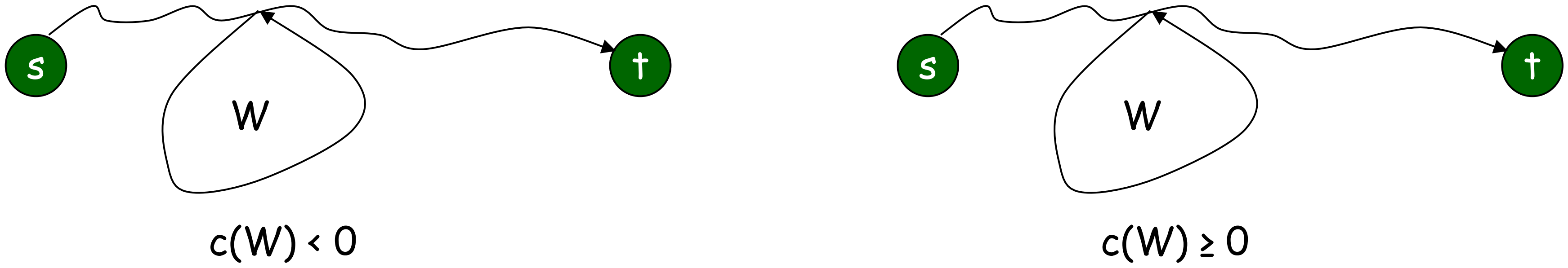


Shortest Paths: Negative Cycles

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



- **Observation.** If some *s*-*t* path contains a negative cycle, then there does not exist a shortest *s*-*t* path; if there exists no negative cycle, there exists a shortest *s*-*t* path that is simple (and has $\leq n - 1$ edges).



Shortest-Paths and Negative-Cycle Problems

- **Single-destination shortest-paths problem.** Given a digraph $G = (V, E)$, with arbitrary edge weights c_{vw} (but no negative cycles) and a destination node t, find v-t shortest paths from every node v to t.

> Single-destination shortest-paths problem is equivalent to single-source shortest-paths problem with edge directions reversed.

- **Negative cycle detection problem.** Given a digraph $G = (V, E)$, with arbitrary edge weights c_{vw}, find a negative cycle (if one exists).

Shortest Paths: Dynamic Programming

- **Def.** OPT(i, v) = length of shortest v-t path P using ≤ i edges.
- **Goal.** OPT(n – 1, v)

space $O(n^2)$

if no neg cycles, there exists a simple shortest path

- **To compute OPT(i,v):**
  - **Case 1:** P uses at most ≤ i – 1 edges.
    - ✓OPT(i, v) = OPT(i – 1, v)
  - **Case 2:** P uses exactly i edges.
    - ✓let (v, w) be the first edge in P: pay the cost of c_{vw} , then select best w-t path using ≤ i – 1 edges

Shortest Paths: Dynamic Programming

- **Def.** OPT(i, v) = length of shortest v-t path P using at most i edges.
- **Goal.** OPT(n – 1, v)

if no neg cycles, there exists a simple shortest path

- **Bellman equation.**

$$\mathrm{OPT}(i,j)=\begin{cases} 0, & \text{if } i=0 \text{ and } v=t \\ \infty, & \text{if } i=0 \text{ and } v\neq t \\ \min\{OPT(i-1,v),\ \min_{w\in V}(OPT(i-1,w)+c_{vw})\} & \text{if } i>0 \end{cases}$$

Shortest Paths: Algorithm

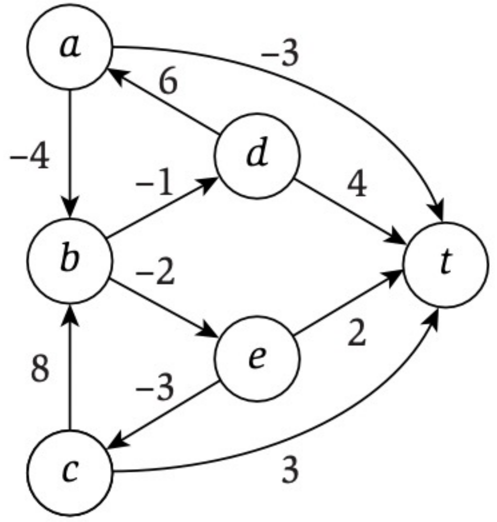
- **Dynamic programming algorithm (bottom-up).**

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] = ∞
   M[0, t] = 0

   for i = 1 to n - 1
      foreach node v ∈ V
         M[i, v] = M[i - 1, v]
         foreach edge (v, w) ∈ E
            M[i, v] = min{ M[i, v], M[i - 1, w] + c_vw }
}
```

- **Finding shortest paths.** Maintain *successor[i, v]* for each M[i, v].

Shortest-Paths Algorithm: Demo



(a)

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	−3	−3	−4	−6	−6
b	∞	∞	0	−2	−2	−2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

(b)

$$\mathrm{OPT}(i,j) = \begin{cases} 0, & \text{if } i = 0 \text{ and } v = t \\ \infty, & \text{if } i = 0 \text{ and } v \neq t \\ \min\{OPT(i-1,v),\ \min\limits_{w \in V}(OPT(i-1,w) + c_{vw})\} & \text{if } i > 0 \end{cases}$$

Shortest Paths: Algorithm

- **Dynamic programming algorithm (bottom-up).**

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] = ∞
   M[0, t] = 0

   for i = 1 to n - 1
      foreach node v ∈ V
         M[i, v] = M[i - 1, v]
         foreach edge (v, w) ∈ E
            M[i, v] = min{ M[i, v], M[i - 1, w] + c_vw }
}
```

only M[i – 1, .] is used!

- **Running time.** $O(mn)$ **Space.** $O(n^2)$ space

can we reduce the required space?

Shortest Paths: Practical Improvements

- **Space optimization.** Maintain two 1-D arrays (instead of 2-D array).
  - $d[v]$ = length of a shortest v-t path that we have found so far
  - $successor[v]$ = next node on a v-t path

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

Bellman-Ford-Moore: Efficient Implementation

- **Dynamic programming algorithm (bottom-up).**

```
Bellman-Ford-Moore(G, s, t) {
   foreach node v ∈ V
      d[v] = ∞
      successor[v] = null

   d[t] = 0
   for i = 1 to n - 1
      foreach node w ∈ V
         if (d[w] has been updated in previous iteration) {
            foreach node v such that (v, w) ∈ E {
               if (d[v] > d[w] + c_vw)
                  d[v] = d[w] + c_vw
                  successor[v] = w
      if (no d[w] value changed in iteration i)
         break
}
```

push-based rather than pull-based

O(n) space

each pass
O(m) time:
O(mn) total

Bellman-Ford-Moore: Analysis

- **Theorem.** After pass i, d[v] = length of a shortest v-t path using $\leq i$ edges.
- **Pf.** (by induction on i)
  - **Base case:** $i = 0$.
  - **Inductive case:** Assume true after pass i. Let P be any v-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, because P' is a w-t path with $\leq i$ edges, at the end of pass i we have $d[w] \leq \ell(P')$.
    - ✓ After considering edge (v, w) in pass $i + 1$ (or in some previous pass $< i + 1$): $d[v] \leq c_{vw} + d[w]$. Then, since $d[w] \leq \ell(P')$ after pass i and $d[w]$ never increases during the algorithm, we have $d[v] \leq c_{vw} + \ell(P') = \ell(P)$.
    - ✓ Obviously, there exists a v-t path of length d[v]. From above, this path is a shortest v-t path using $\leq i + 1$ edges.

Bellman-Ford-Moore: Analysis

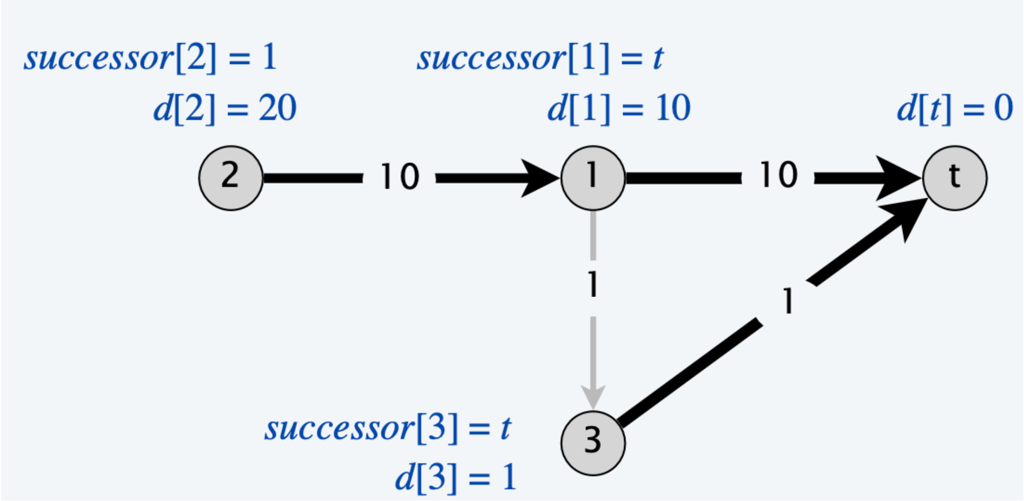
- **Theorem.** Assume no negative cycles, Bellman-Ford-Moore computes the lengths of the shortest v-t paths in $O(mn)$ time and $O(n)$ extra space.
- **Pf.** From previous observation and theorem, we have:
  - If no negative cycles, shortest path exists and has at most $n - 1$ edges.
  - After pass $n - 1$, $d[v]$ = length of a shortest v-t path using $n - 1$ edges.

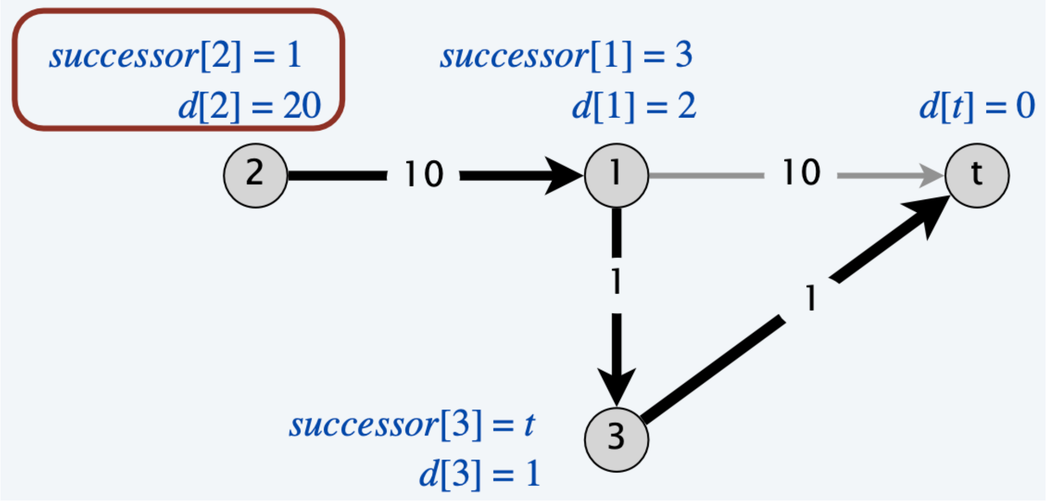
- **Remark.** Bellman–Ford–Moore is typically faster in practice.
  - Edge (v, w) is considered in pass $i + 1$ only if $d[w]$ was updated in pass i.
  - If shortest path is known to have k edges, then algorithm finds it in k passes.

- **Q.** How do we find a shortest v-t path of length $d[v]$ for every node v?

Bellman-Ford-Moore: Finding Shortest Paths

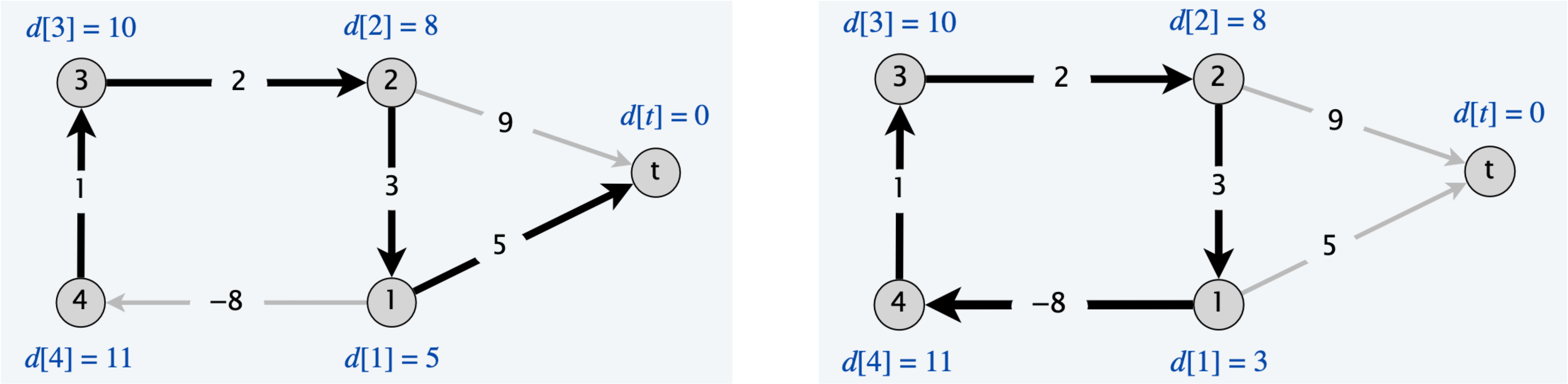
- **Claim.** Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].
- **Counterexamples.** (Claim is false!)
  - Length of successor *v*-*t* path may be strictly shorter than *d*[*v*].
    - ✓Ex. Consider nodes in order: *t*, 1, 2, 3





Bellman-Ford-Moore: Finding Shortest Paths

- **Claim.** Throughout Bellman–Ford–Moore, following the *successor*[*v*] pointers gives a directed path from *v* to *t* of length *d*[*v*].
- **Counterexamples.** (Claim is false!)
  - ➢ Length of successor *v*-*t* path may be strictly shorter than *d*[*v*].
  - ➢ If negative cycles exist, successor graph may have directed cycles.
    - ✓Ex. Consider nodes in order: *t*, 1, 2, 3, 4



Bellman-Ford-Moore: Finding Shortest Paths

- **Lemma.** Any directed cycle W in the successor graph is a negative cycle.
- **Pf.**
  - If *successor*[v] = w, we have $d[v] \geq d[w] + c_{vw}$. (They are equal when *successor*[v] is set; $d[w]$ can only decrease; $d[v]$ decreases only when *successor*[v] is reset.)
  - Let $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_1$ 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:

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

holds with strict inequality since we are updating $d[v_k]$

  - Adding inequalities yields $\ell(v_1, v_2) + \ell(v_2, v_3) + \ldots + \ell(v_{k-1}, v_k) + \ell(v_k, v_1) < 0$. ▪

Bellman-Ford-Moore: Finding Shortest Paths

- **Theorem.** Assuming no negative cycles, Bellman–Ford–Moore finds shortest v-t paths for every node v in $O(mn)$ time and $O(n)$ extra space.
- **Pf.**
  - From previous lemma, the successor graph cannot have a directed cycle. Thus, following the successor pointers from v yields a directed path to t.
  - Let $v = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ be the nodes along this path P.
  - Upon termination, if *successor*[v] = w, we have $d[v] = d[w] + c_{vw}$. (They are equal when *successor*[v] is set; $d[\cdot]$ did not change since algorithm terminates.)
  - Thus,

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

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

Shortest Paths: Asymptotic Complexity

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	$O(n^{2.38}\ W)$	Sankowsi, 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

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

7. Distance-Vector Protocols

Distance-Vector Routing Protocols

- **Communication network.**
  - Node ≈ router.
  - Edge ≈ direct communication link.
  - Cost of edge ≈ latency of link.

non-negative costs, 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.

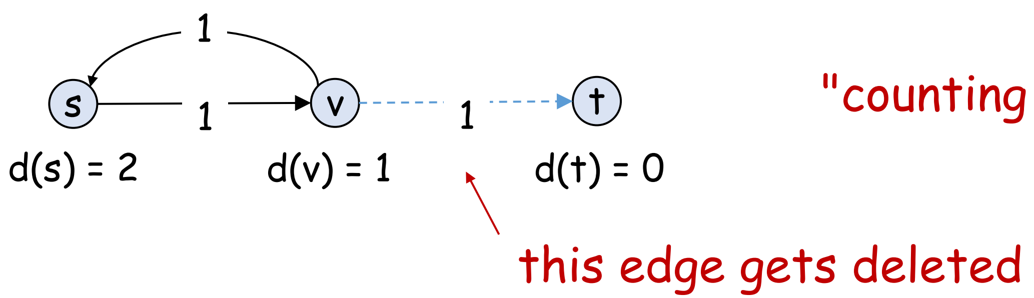
Asynchronous Shortest-Paths Algorithm

```
Asynchronous-Shortest-Path(G, s, t)
  n = number of nodes in G
  Array M[V]
  Initialize M[t] = 0 and M[v] = ∞ for all other v ∈ V
  Declare t to be active and all other nodes inactive
  While there exists an active node
    Choose an active node w
       For all edges (v, w) in any order
         M[v] = min(M[v], c_vw + M[w])
         If this changes the value of M[v], then
           first[v] = w
           v becomes active
       Endfor
       w becomes inactive
  EndWhile
```

no for loop for nodes in asynchronous version

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.
- **Example applications.** RIP, Xerox XNS RIP, Novell's IPX RIP, Cisco's IGRP, DEC's DNA Phase IV, AppleTalk's RTMP.
- **Caveat.** Edge costs may change during algorithm (or fail completely).



"counting to infinity"

Path-Vector Routing Protocols

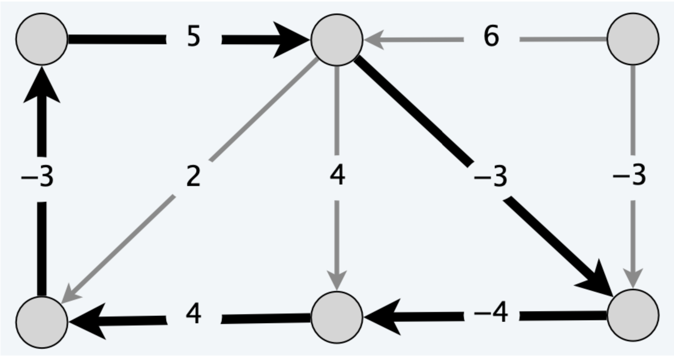
- **Link-state routing protocols.**
  - Each router also stores the entire path. not just the distance and first hop
  - Based on Dijkstra's algorithm.
  - Avoids "counting-to-infinity" problem and related difficulties.
  - Requires significantly more storage.

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

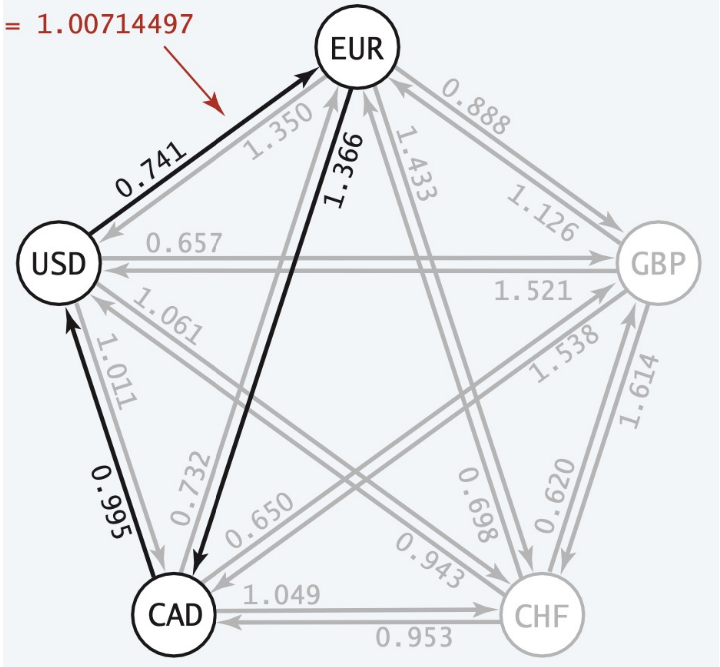
8. Negative Cycles

Detecting Negative Cycles

- **Negative cycle detection problem.** Given a directed graph G = (V, E), with arbitrary edge weights c_{vw}, find a negative cycle (if one exists).

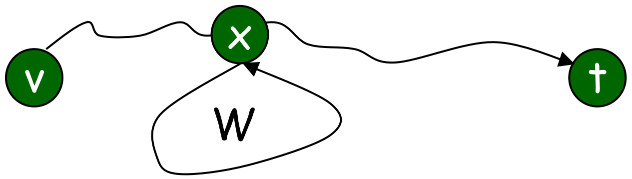


- **Application.** [Currency conversion] Given *n* currencies and exchange rates between them, is there an arbitrage opportunity?
  - ➢ Fastest algorithm very valuable!



Detecting Negative Cycles

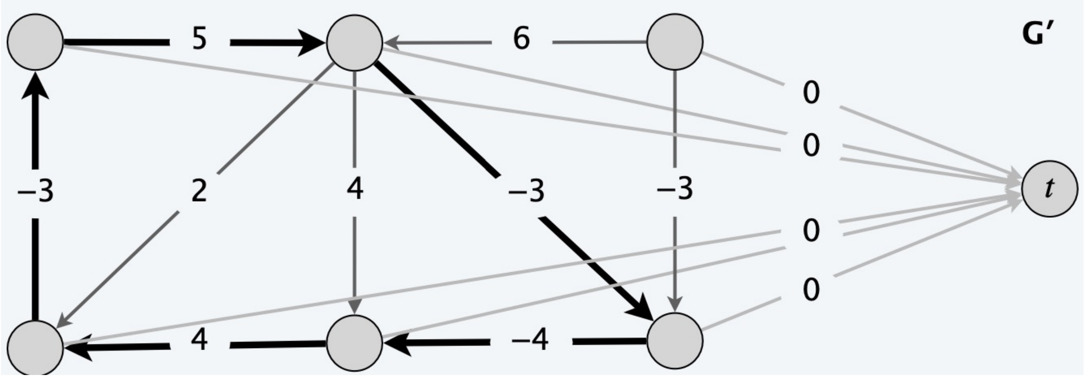
- **Lemma.** If OPT(n, v) = OPT(n – 1, v) for every v, then no negative cycles.
- **Pf.** The OPT(n, v) values have converged ⇒ shortest v-t path exists. ▪
- **Lemma.** If OPT(n, v) < OPT(n – 1, v) for some node v, then (any) shortest v-t path of length ≤ n contains a cycle W. Moreover, W is a negative cycle.
- **Pf.**
  - OPT(n, v) < OPT(n – 1, v) ⇒ shortest v-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-t path with < n edges.
  - Therefore, W is a negative cycle. ▪



c(W) < 0

Detecting Negative Cycles

- **Theorem.** Can find a negative cycle in $O(mn)$ time and $O(n^2)$ space.
- **Pf.** Add new sink node t and connect all nodes to t with 0-length edges. G has a negative cycle if and only if G' has a negative cycle.
  - ➢ **Case 1:** $OPT(n, v) = OPT(n-1, v)$ for every node v
    - ✓By previous lemma, there exist no negative cycles.
  - ➢ **Case 2:** $OPT(n, v) < OPT(n-1, v)$ for some node v
    - ✓Can extract negative cycle from v-t path (cycle cannot contain t since no edge leaves t). ▪



Detecting Negative Cycles

- **Theorem.** 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 $pass(v) - 1 \leq pass(successor[v])$ for each v.
  - Following successor pointers ($\geq n'$ edges), we must eventually repeat a node.
  - Previous lemma shows that the corresponding cycle is a negative cycle. ▪
- **Remark.** See textbook for improved version and early termination rule. (Tarjan's subtree disassembly trick.)