

# Chapter 6

## Dynamic Programming



Slides by Kevin Wayne.  
Copyright © 2005 Pearson-Addison Wesley.  
All rights reserved.

## 6.8 Shortest Paths

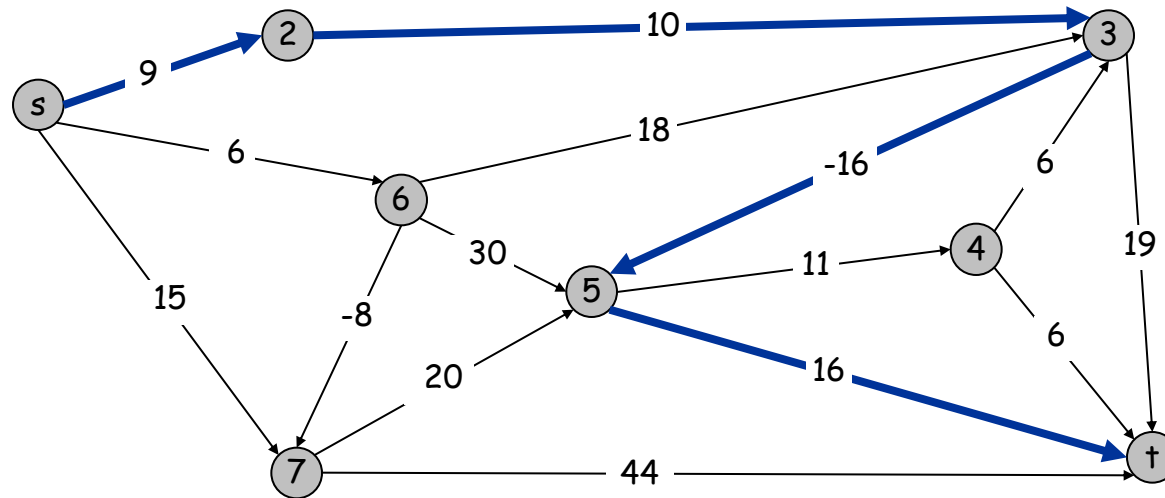
---

# Shortest Paths

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

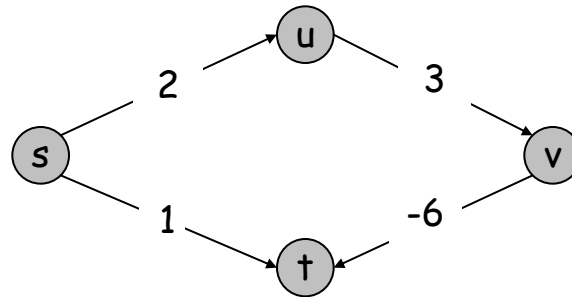
↖ allow negative weights

**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  $w$ .

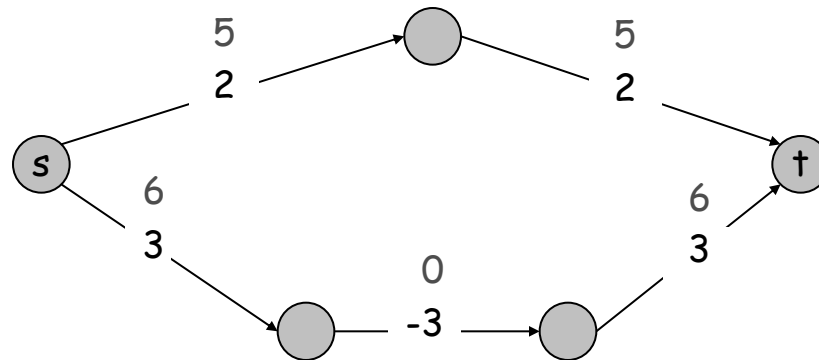


# Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs.

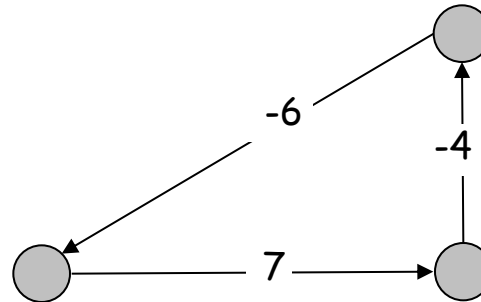


Re-weighting. Adding a constant to every edge weight can fail.

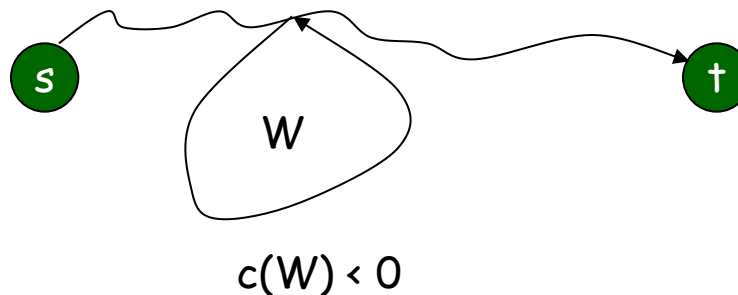


# Shortest Paths: Negative Cost Cycles

Negative cost cycle.



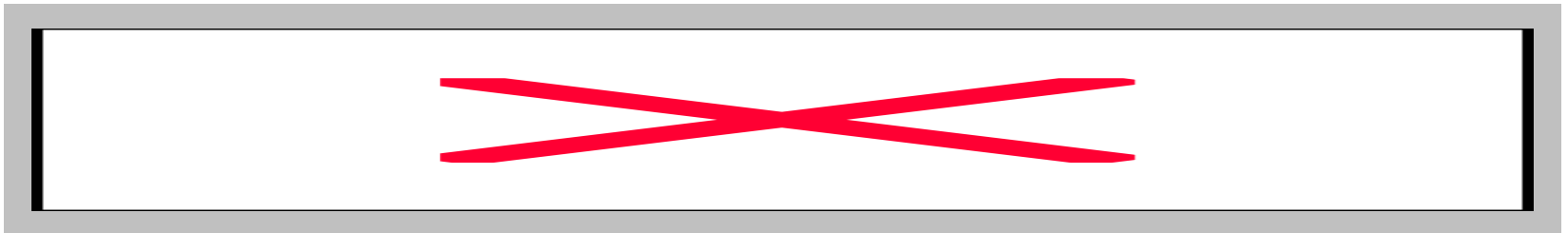
**Observation.** If some path from  $s$  to  $t$  contains a negative cost cycle, there does not exist a shortest  $s$ - $t$  path; otherwise, there exists one that is simple.



# Shortest Paths: Dynamic Programming

**Def.**  $\text{OPT}(i, v)$  = length of shortest  $v$ - $t$  path  $P$  using at most  $i$  edges.

- Case 1:  $P$  uses at most  $i-1$  edges.
  - $\text{OPT}(i, v) = \text{OPT}(i-1, v)$
- Case 2:  $P$  uses exactly  $i$  edges.
  - if  $(v, w)$  is first edge, then  $\text{OPT}$  uses  $(v, w)$ , and then selects best  $w$ - $t$  path using at most  $i-1$  edges



**Remark.** By previous observation, if no negative cycles, then  $\text{OPT}(n-1, v)$  = length of shortest  $v$ - $t$  path.

# Shortest Paths: Implementation

```
Shortest-Path( $G, t$ ) {  
    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  $(v, w) \in E$   
                 $M[i, v] \leftarrow \min \{ M[i, v], M[i-1, w] + c_{vw} \}$   
}
```

**Analysis.**  $\Theta(mn)$  time,  $\Theta(n^2)$  space.

**Finding the shortest paths.** Maintain a "successor" for each table entry.

# Shortest Paths: Practical Improvements

## Practical improvements.

- Maintain only one array  $M[v]$  = shortest  $v$ - $t$  path that we have found so far.
- No need to check edges of the form  $(v, w)$  unless  $M[w]$  changed in previous iteration.

**Theorem.** Throughout the algorithm,  $M[v]$  is length of some  $v$ - $t$  path, and after  $i$  rounds of updates, the value  $M[v]$  is no larger than the length of shortest  $v$ - $t$  path using  $\leq i$  edges.

## Overall impact.

- Memory:  $O(m + n)$ .
- Running time:  $O(mn)$  worst case, but substantially faster in practice.



# Bellman-Ford: Efficient Implementation

```
Push-Based-Shortest-Path( $G, s, t$ ) {  
    foreach node  $v \in V$  {  
         $M[v] \leftarrow \infty$   
         $\text{successor}[v] \leftarrow \phi$   
    }  
  
     $M[t] = 0$   
    for  $i = 1$  to  $n-1$  {  
        foreach node  $w \in V$  {  
            if ( $M[w]$  has been updated in previous iteration) {  
                foreach node  $v$  such that  $(v, w) \in E$  {  
                    if ( $M[v] > M[w] + c_{vw}$ ) {  
                         $M[v] \leftarrow M[w] + c_{vw}$   
                         $\text{successor}[v] \leftarrow w$   
                    }  
                }  
            }  
        }  
        If no  $M[w]$  value changed in iteration  $i$ , stop.  
    }  
}
```

## 6.9 Distance Vector Protocol

---

# Distance Vector Protocol

## Communication network.

- Nodes  $\approx$  routers.
- Edges  $\approx$  direct communication link.
- Cost of edge  $\approx$  delay on link.  $\leftarrow$  naturally nonnegative, but Bellman-Ford used anyway!

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

**Bellman-Ford.** Uses only local knowledge of neighboring nodes.

**Synchronization.** We don't expect routers to run in lockstep. The order in which each `foreach` loop executes is not important. Moreover, algorithm still converges even if updates are asynchronous.

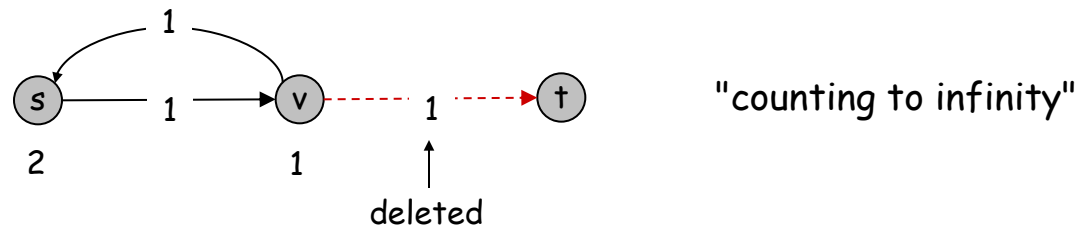
# Distance Vector Protocol

## Distance vector protocol.

- 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.
- "Routing by rumor."

**Ex.** 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).



# Path Vector Protocols

## Link state routing.

- Each router also stores the entire path.
- 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).

## 6.10 Negative Cycles in a Graph

---

## Detecting Negative Cycles

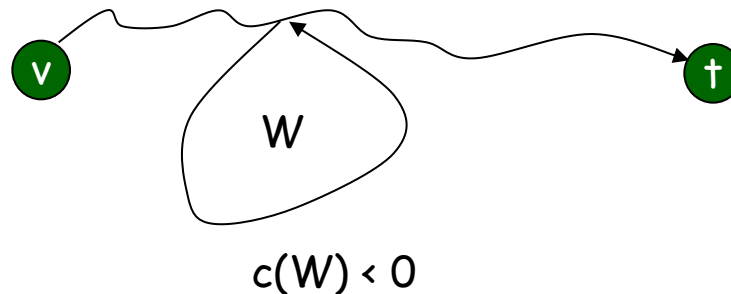
**Lemma.** If  $\text{OPT}(n,v) = \text{OPT}(n-1,v)$  for all  $v$ , then no negative cycles.

**Pf.** Bellman-Ford algorithm.

**Lemma.** If  $\text{OPT}(n,v) < \text{OPT}(n-1,v)$  for some node  $v$ , then (any) shortest path from  $v$  to  $t$  contains a cycle  $W$ . Moreover  $W$  has negative cost.

**Pf.** (by contradiction)

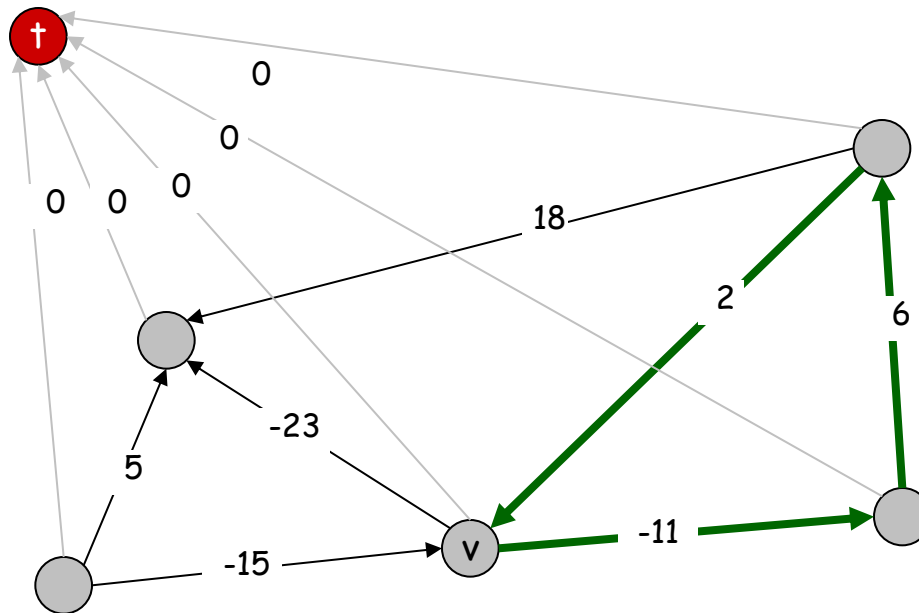
- Since  $\text{OPT}(n,v) < \text{OPT}(n-1,v)$ , we know  $P$  has exactly  $n$  edges.
- By pigeonhole principle,  $P$  must contain a directed cycle  $W$ .
- Deleting  $W$  yields a  $v$ - $t$  path with  $< n$  edges  $\Rightarrow W$  has negative cost.



# Detecting Negative Cycles

**Theorem.** Can detect negative cost cycle in  $O(mn)$  time.

- Add new node  $t$  and connect all nodes to  $t$  with 0-cost edge.
- Check if  $\text{OPT}(n, v) = \text{OPT}(n-1, v)$  for all nodes  $v$ .
  - if yes, then no negative cycles
  - if no, then extract cycle from shortest path from  $v$  to  $t$

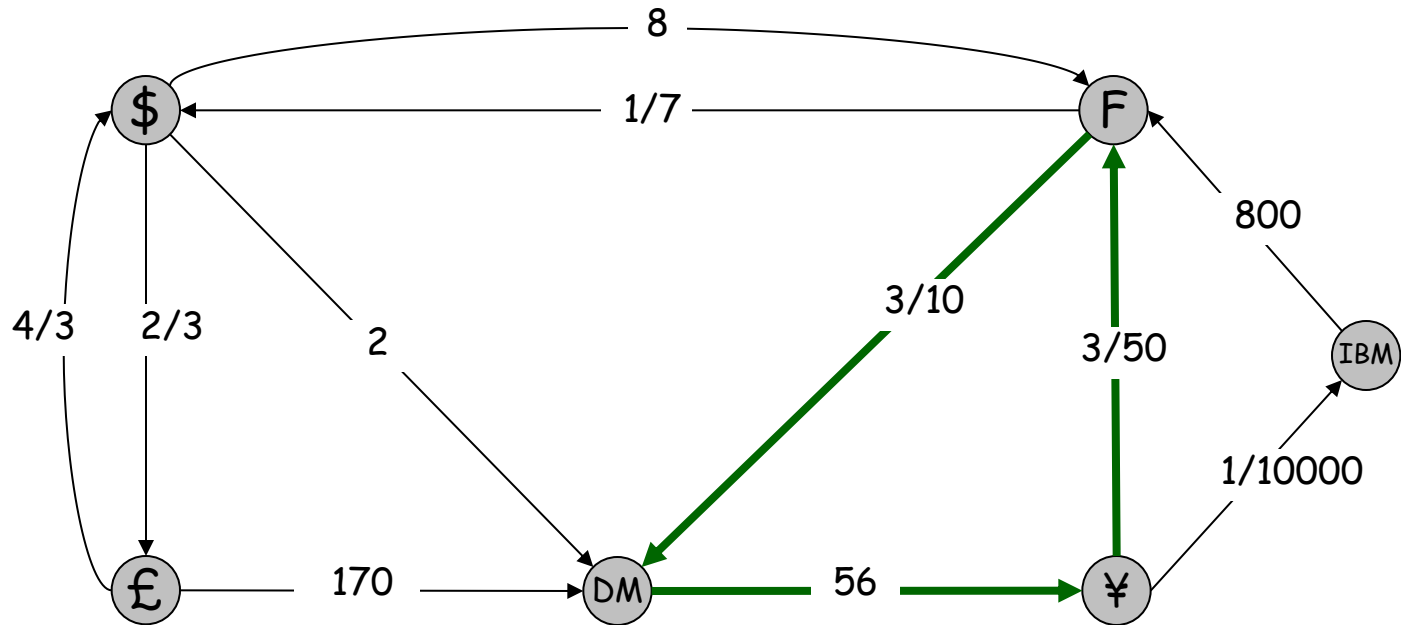




## Detecting Negative Cycles: Application

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

**Remark.** Fastest algorithm very valuable!



## Detecting Negative Cycles: Summary

**Bellman-Ford.**  $O(mn)$  time,  $O(m + n)$  space.

- Run Bellman-Ford for  $n$  iterations (instead of  $n-1$ ).
- Upon termination, Bellman-Ford successor variables trace a negative cycle if one exists.
- See p. 288 for improved version and early termination rule.