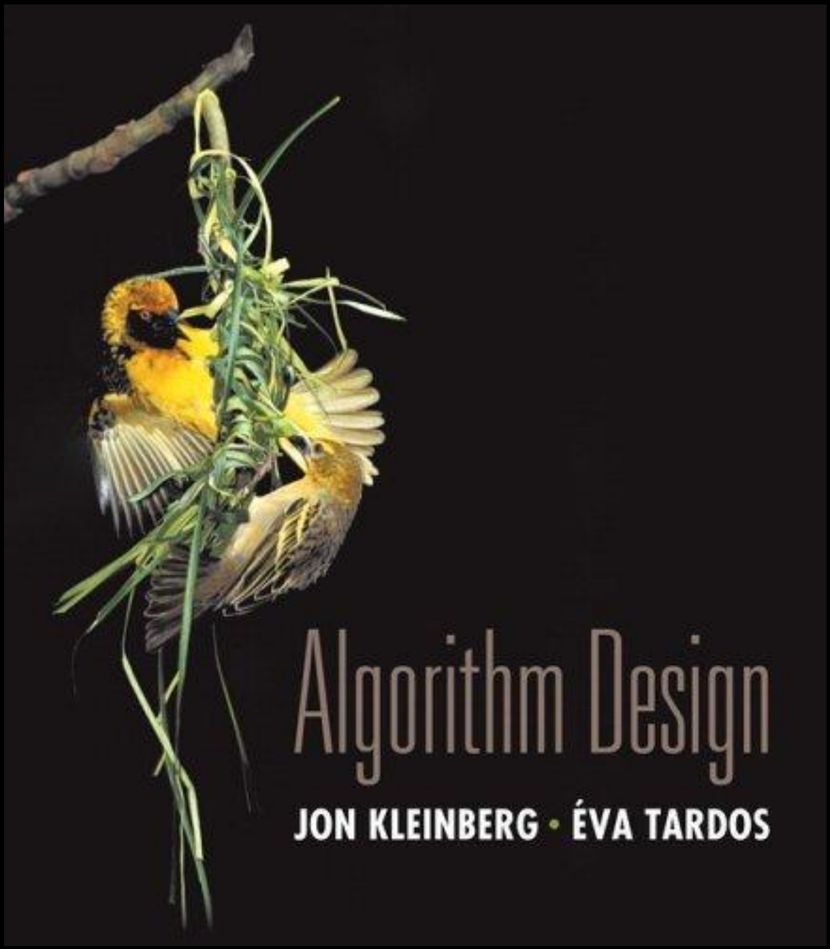


Chapter 6

Dynamic Programming

Slides by Kevin Wayne.

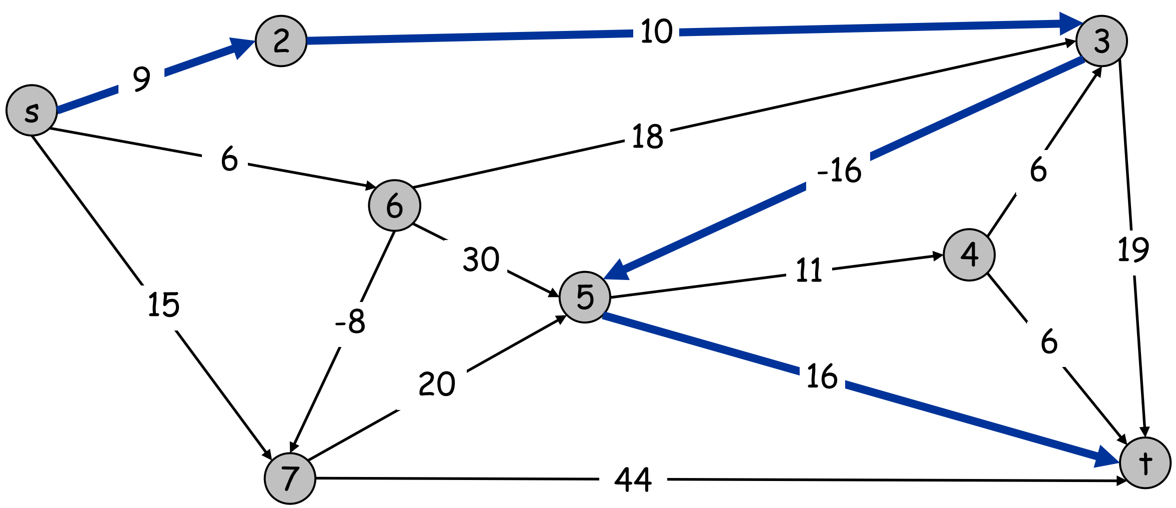
6.8 Shortest Paths

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.

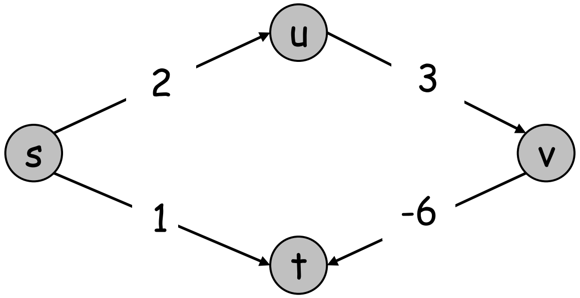
allow negative weights

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.

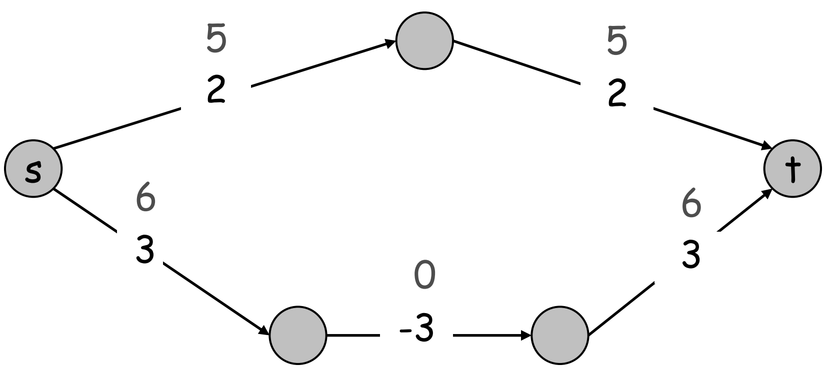


Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs.

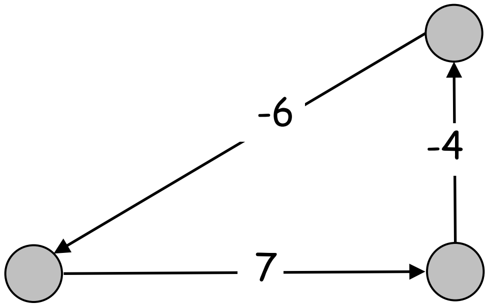


Re-weighting. Adding a constant to every edge weight can fail.

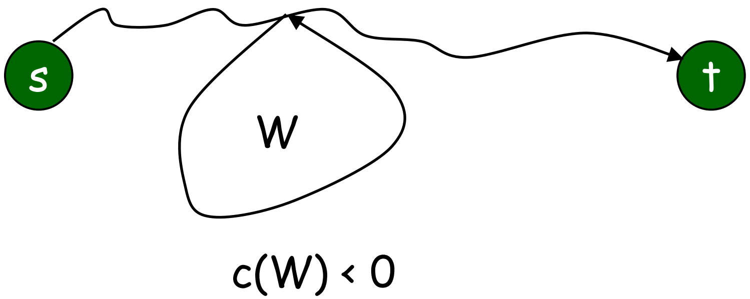


Shortest Paths: Negative Cost Cycles

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 (no cycles and at most n-1 edges).



Shortest Paths: Dynamic Programming

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

- Case 1: v does not have out edges
  - OPT(i, v) = infinite

- Case 2: P has out edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best w-t path using at most i-1 edges

$$OPT(i,v) = \begin{cases} \infty & \text{if } i = 0 \\ \min_{(v,w) \in E} \{ c_{vw} + OPT(i-1, v) \} & \text{otherwise} \end{cases}$$

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

Shortest Paths: Implementation

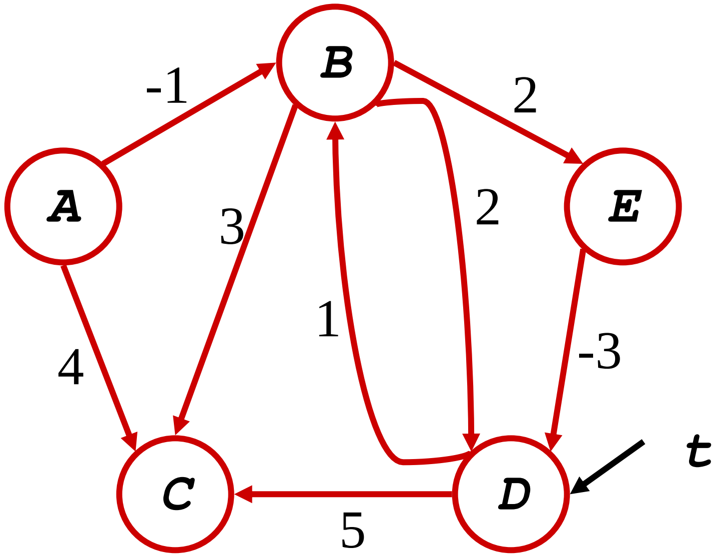
```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] ← ∞
   M[0, t] ← 0

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

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

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

Shortest Paths: Implementation



Ex: work on board

Shortest Paths: Example

Custo Iteração	**A**	**B**	**C**	**D=t**	**E**
0	Inf	Inf	Inf	0	inf
1	Inf	2	Inf	0	-3
2	1	-1	Inf	0	-3
3	-2	-1	Inf	0	-3
4	-2	-1	Inf	0	-3

Custos ao logo do algoritmo

Sucessor Iteração	**A**	**B**	**C**	**D=t**	**E**
0					
1		D			D
2	B	E			D
3	B	E			D
4	B	E			D

Sucessores ao longo do algoritmo.
Vazio=NULL

Shortest Paths: Practical Improvements

Practical improvements.

- Maintain only one array M[v] = shortest v-t path that we have found so far.
- If no M[v] is changed during an iteration we can stop because it will not change in the next iterations as well

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 ∈ V {
      M[v] ← ∞
      successor[v] ← φ
   }

   M[t] = 0
   for i = 1 to n-1 {
      foreach node v ∈ V {
        foreach node w such that (v, w) ∈ E {
            if (M[v] > M[w] + c_vw) {
               M[v] ← M[w] + c_vw
               successor[v] ← w
            }
        }
       If no M[v] value changed in iteration i, stop.
   }
}
```

6.10 Negative Cycles in a Graph

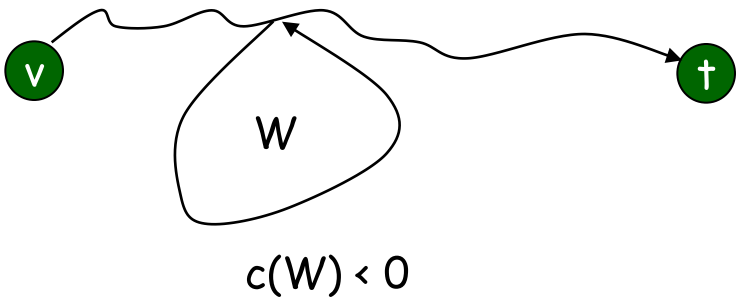
Detecting Negative Cycles

Lemma. If OPT(n,v) = OPT(n-1,v) for all v, then no negative cycles.
Pf. If there is a negative cycle, we would have OPT(j,v) < OPT(n-1,v) for a large enough integer j. However, OPT(n,v)=OPT(n-1,v) for all v implies that OPT(j,v) for all v and for every j larger than n-1.

Lemma. If OPT(n,v) < OPT(n-1,v) for some node v, then there is a negative cycle W in the graph.

Pf. (by contradiction)

- Since OPT(n,v) < OPT(n-1,v), we know that the shortest path P from v to t, among those that use at most n edges, 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: 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.

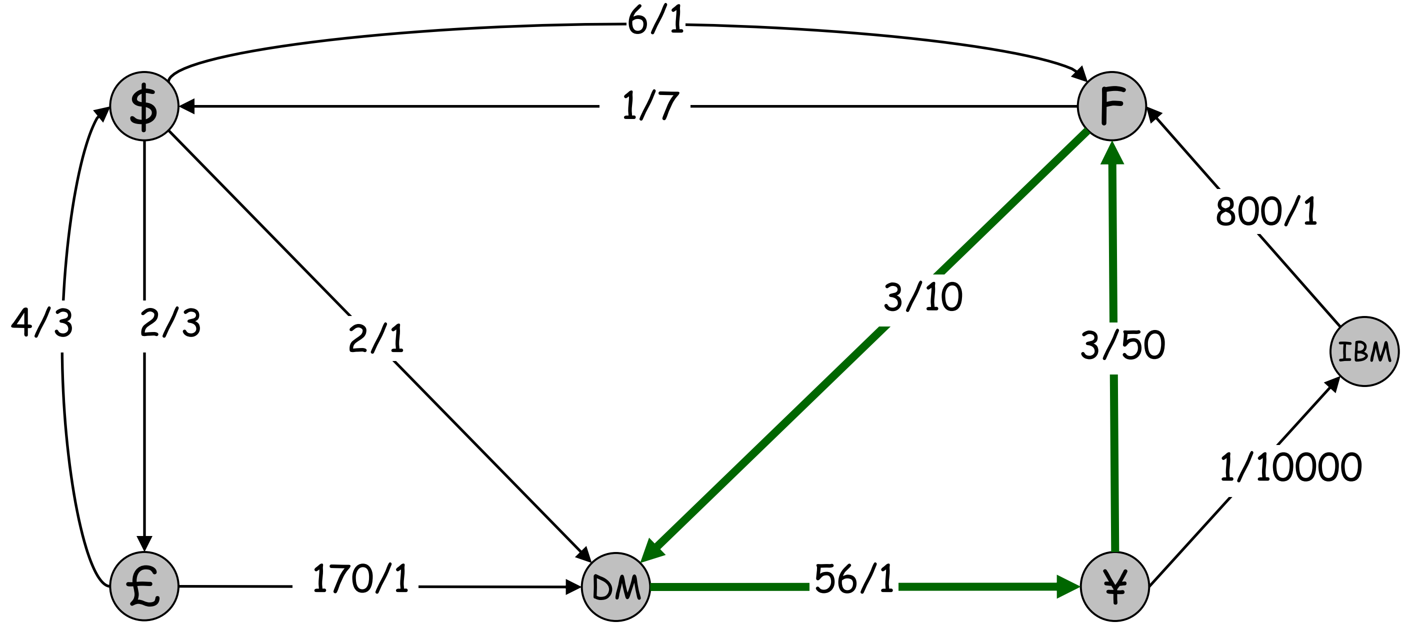
Detecting Negative Cycles

```
Push-Based-Shortest-Path(G, s, t) {
   foreach node v ∈ V {
      M[v] ← ∞
      successor[v] ← φ
   }
   M[t] = 0
   for i = 1 to n-1 {
      foreach node v ∈ V {
        foreach node w such that (v, w) ∈ E {
            if (M[v] > M[w] + c_vw) {
               M[v] ← M[w] + c_vw
               successor[v] ← w
            }
         }
    }
       foreach node v ∈ V {
        foreach node w such that (v, w) ∈ E {
            if (M[v] > M[w] + c_vw)
               Return 'there is a negative cycle'
       }
```

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!



6.9 Distance Vector Protocol

Distance Vector Protocol

Communication network.

- Nodes ≈ routers.
- Edges ≈ direct communication link.
- Cost of edge ≈ delay on link. ← 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 in not important. Moreover, algorithm still converges even if updates are asynchronous.

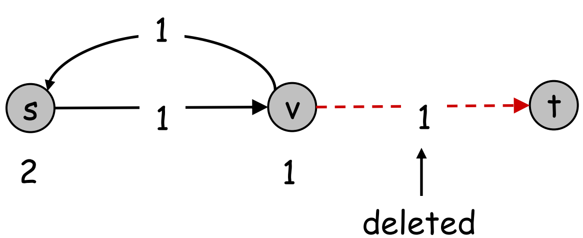
Distance Vector Protocol

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).



"counting to infinity"

Path Vector Protocols

Link state routing.

- 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).

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 OPT(n, v) = 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