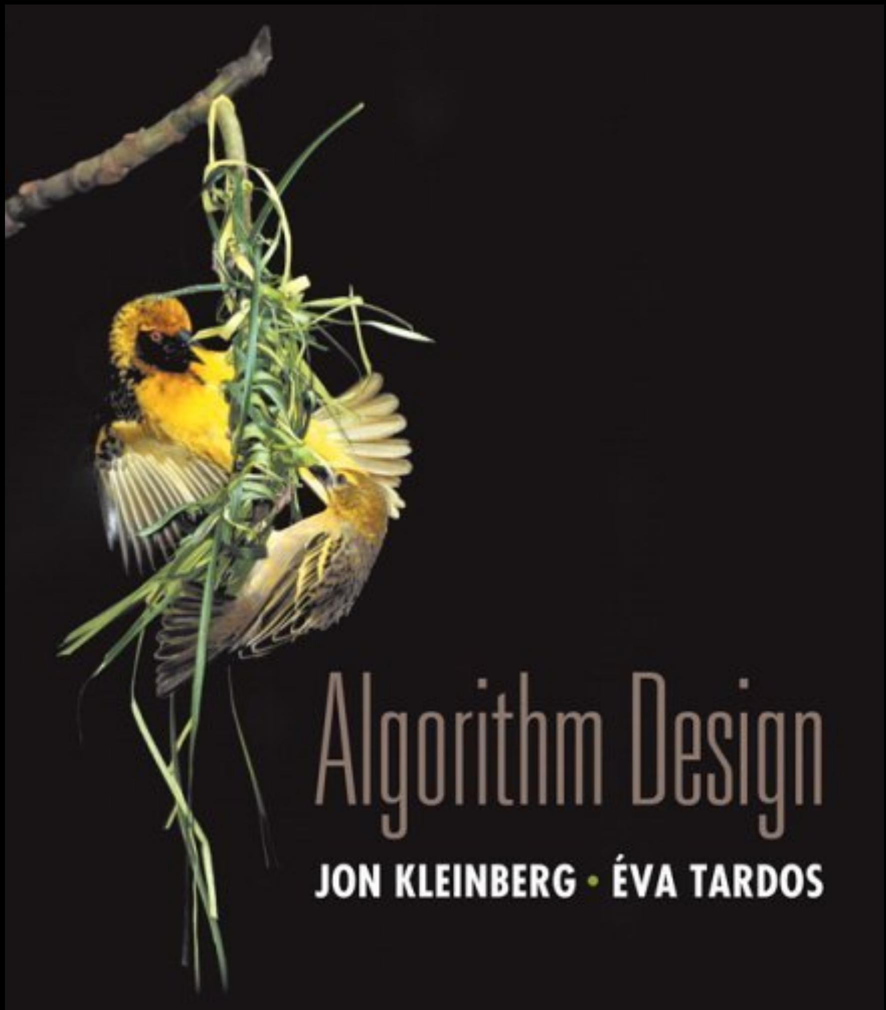


Chapter 6

Dynamic Programming

Slides by Kevin Wayne.

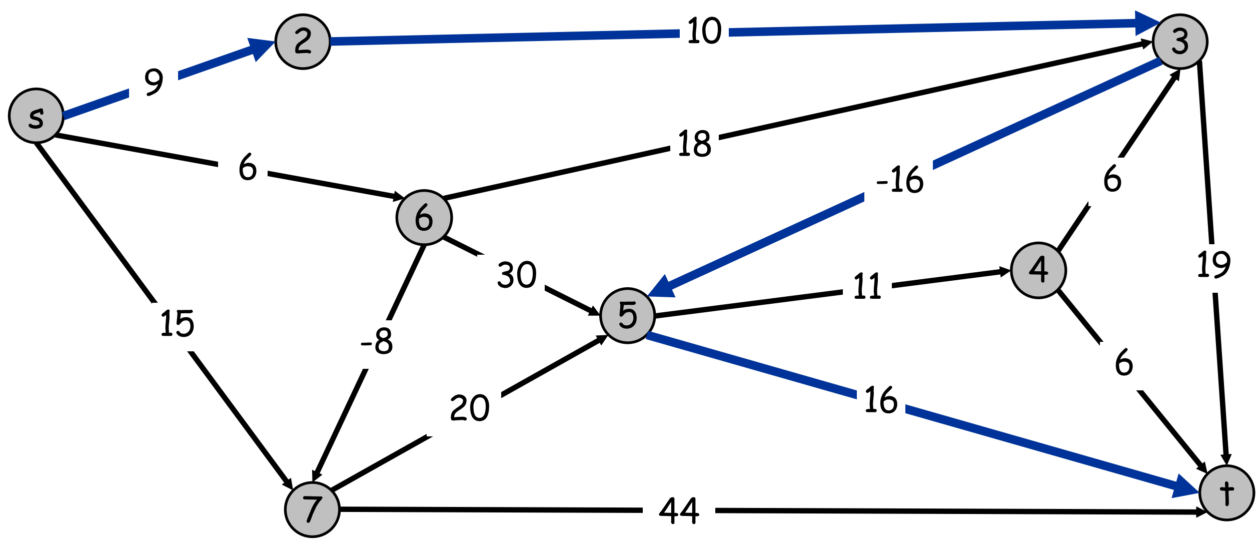
6.8 Shortest Paths

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.

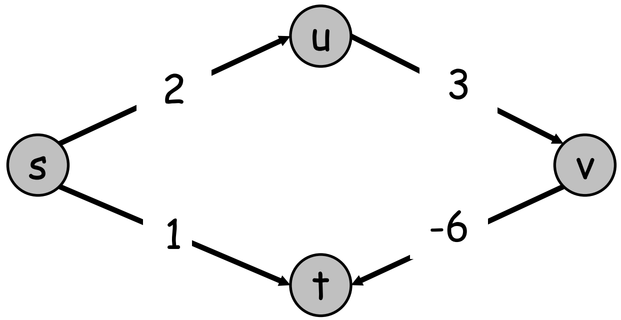
allow negative weights

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.

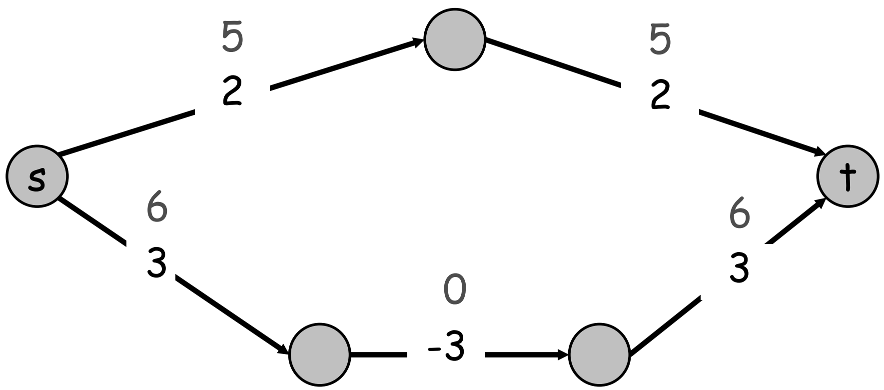


Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs.

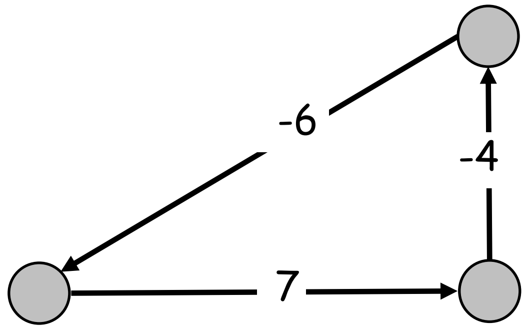


Re-weighting. Adding a constant to every edge weight can fail.

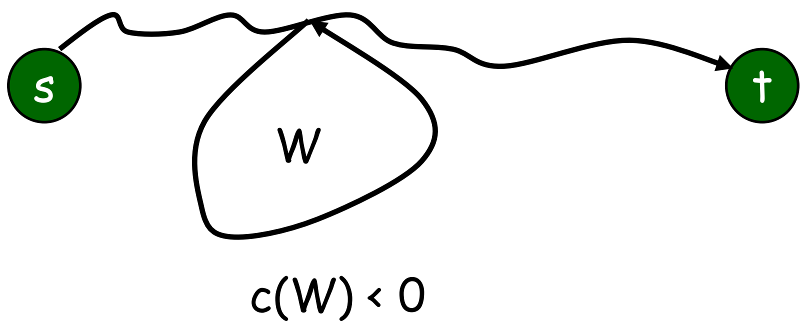


Shortest Paths: Negative Cost Cycles

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.



Shortest Paths: Dynamic Programming

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

- Case 1: P uses at most i-1 edges.
  - OPT(i, v) = OPT(i-1, v)

- Case 2: P uses exactly i 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

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

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-1, v]
      foreach edge (v, w) ∈ E
         M[i, v] ← 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.

Work out the algorithm on the previous graph example

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

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

6.10 Negative Cycles in a Graph

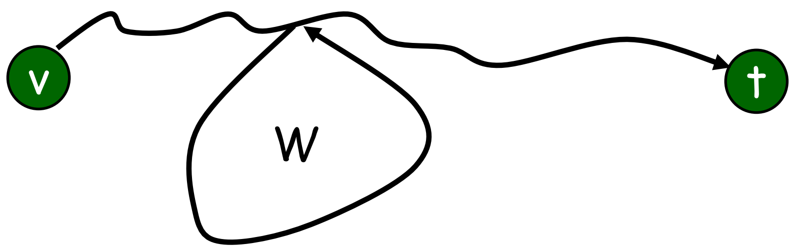
Detecting Negative Cycles

Lemma. If OPT(n,v) = OPT(n-1,v) for all v, then no negative cycles.
Pf. Bellman-Ford algorithm.

Lemma. If OPT(n,v) < 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 OPT(n,v) < 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.

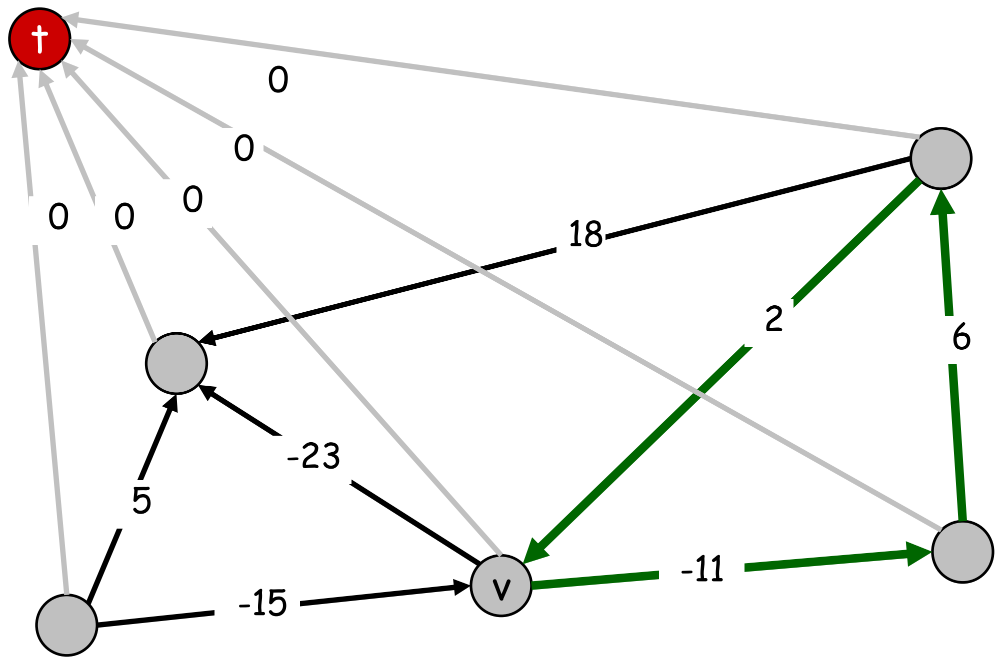


c(W) < 0

Detecting Negative Cycles

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

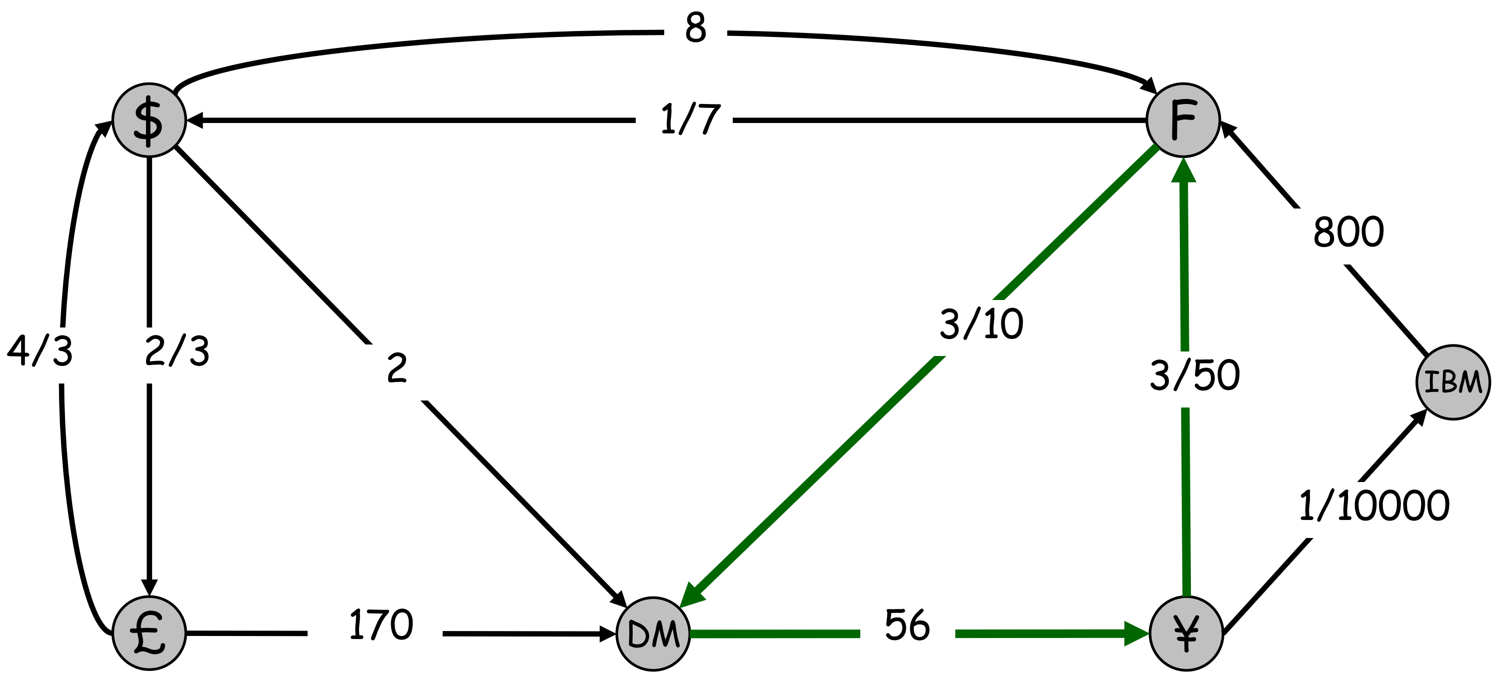
- Add new node t and connect all nodes to t with 0-cost edge.
- Check if OPT(n, v) = OPT(n-1, v) for all nodes v.
  - if yes, then no negative cycles
  - if no, then extract cycle from shortest path from v to t



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!

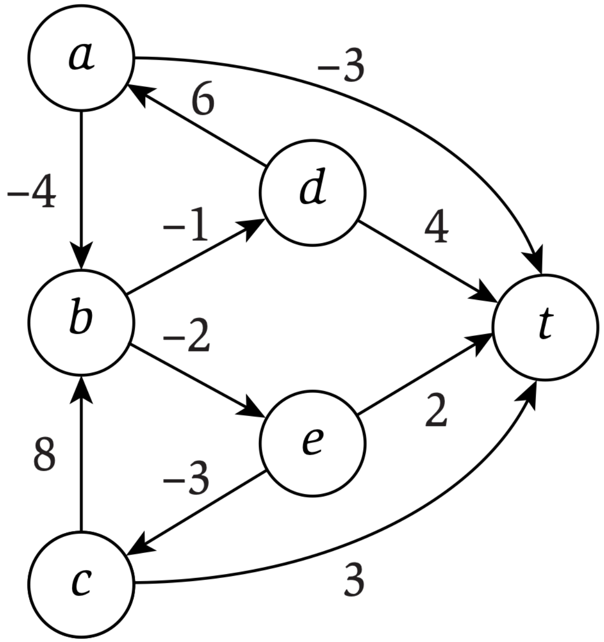


Idea here total profit along the path is a product of exchange rates: 8*1/7 , so for a path it would be a product of all the costs.
If we take the logarithm of the final cost and use the rule of logs, we can write the log of product as sum of logsand transform the problem to the shortest path problem on the graph where costs are longs of original weights

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.



(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