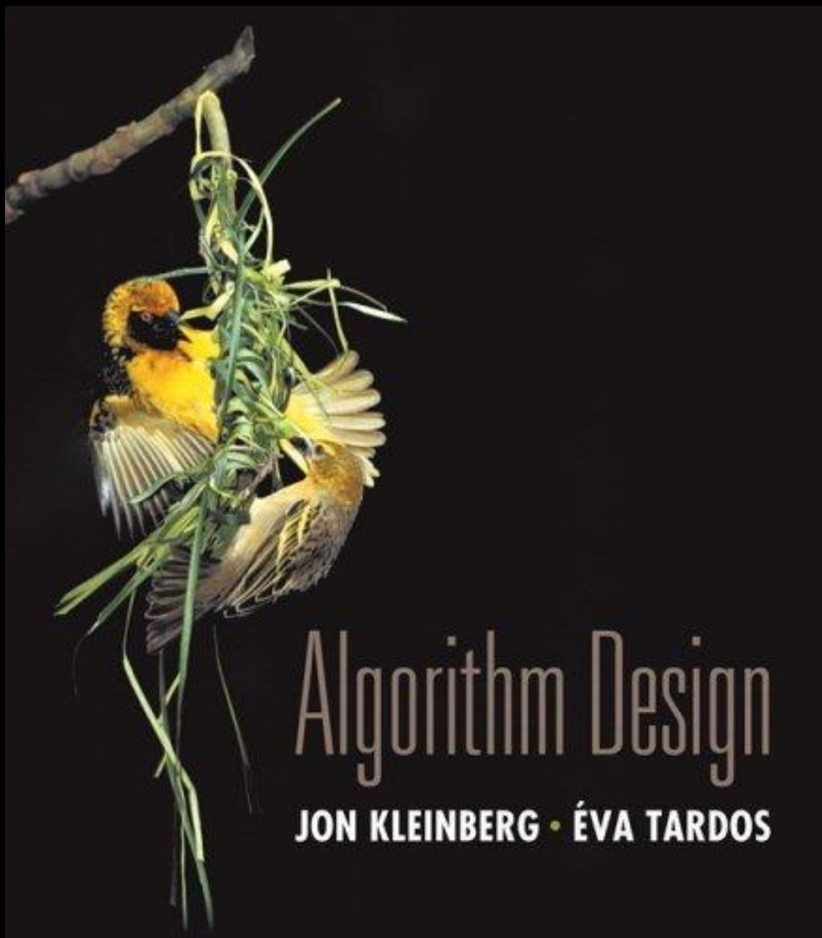


Chapter 6

Dynamic Programming

Slides by Kevin Wayne.

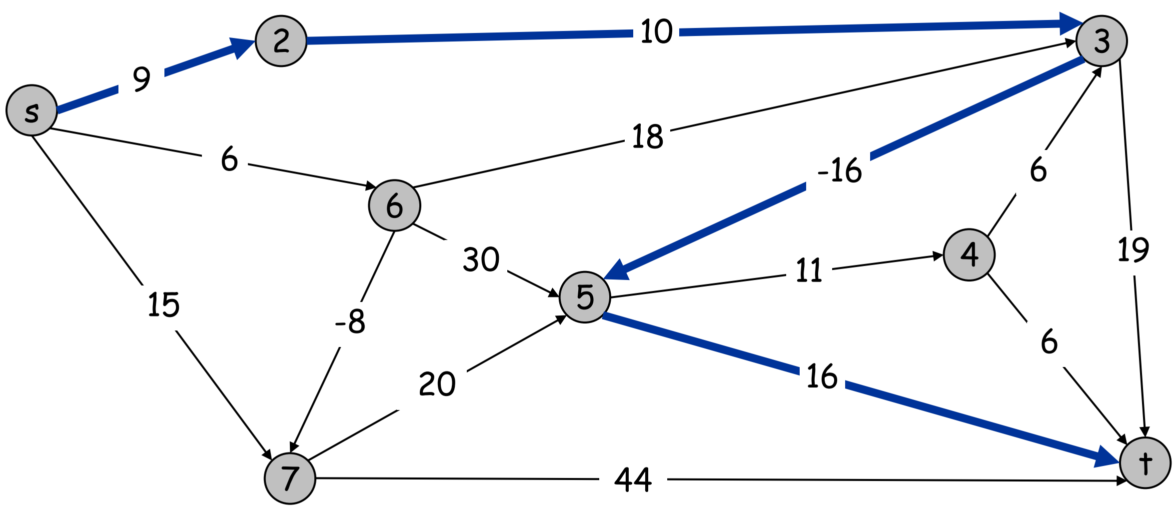
6.8 Shortest Paths

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.

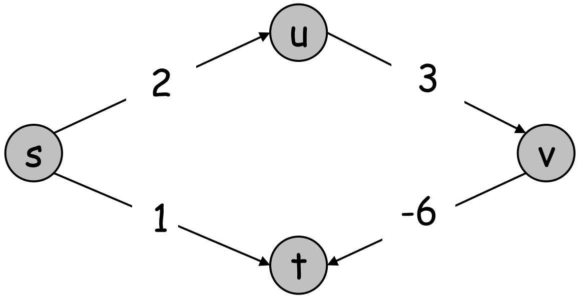
allow negative weights

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.

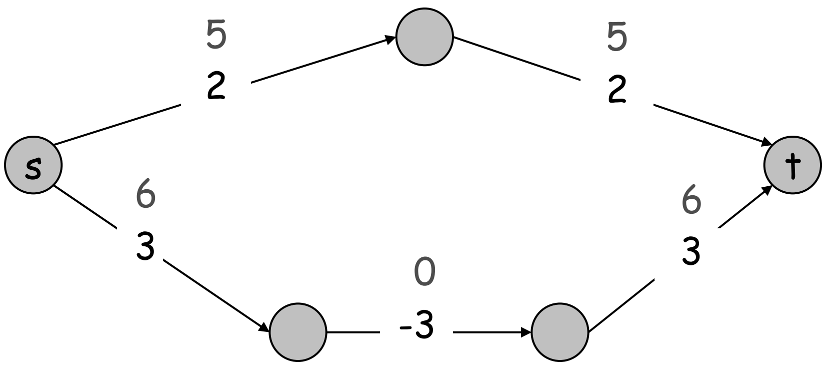


Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs.

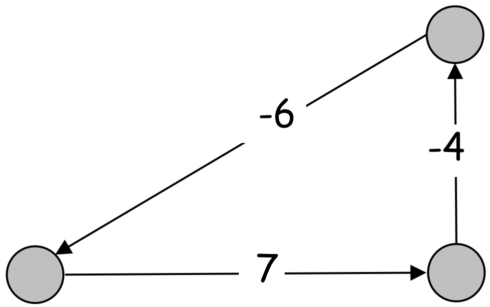


Re-weighting. Adding a constant to every edge weight can fail.

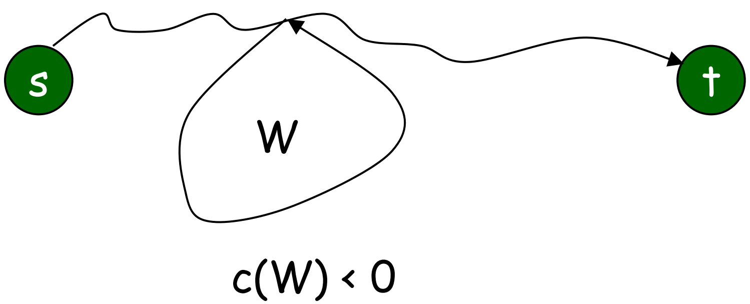


Shortest Paths: Negative Cost Cycles

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 0: v=t,
  - OPT(i, v) =0
- Case 1: v<>t, i=0 or v has outdegree 0
  - OPT(i, v) = ∞
- Case 2: v<>t, i>0 and v has outdegree > 0.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best w-t path using at most i-1 edges

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

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

Shortest Paths: Implementation

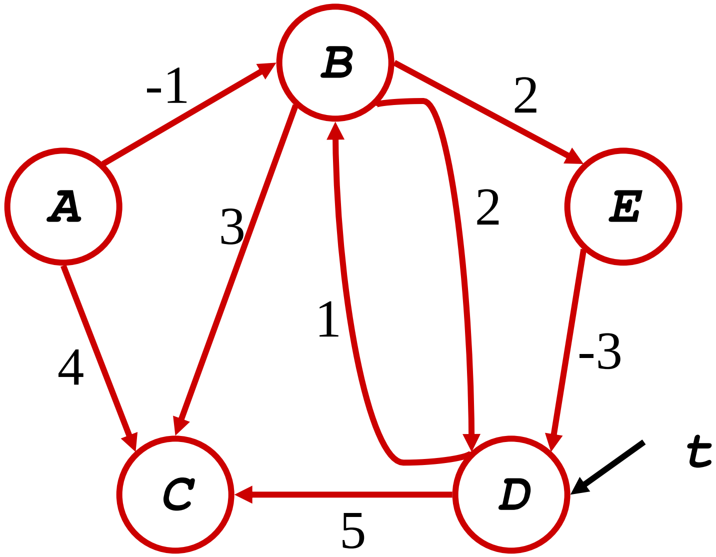
Bellman-Ford algorithm:

```
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
                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

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

Custos ao longo do algoritmo

Sucessores ao longo do algoritmo.
Vazio=NULL

Practical improvements - Part I

- We just maintain the last two lines of the matrix M
  - Memory consumption reduces to O(m+n)
- If no M[i,v] is changed during an iteration we can stop because it will not change in the next iterations as well

Pf

We have

$$M[i,v] = \min\{M[i-1,v], min_u\{c_{uv} + M[i-1,u] | (u,v) \in E\}\}$$
$$M[i+1,v] = \min\{M[i,v], min_u\{c_{uv} + M[i,u] | (u,v) \in E\}\}$$

Thus, if M[i,v]=M[i-1,v] for all v then M[i+1,v] = M[i,v] for all v

Shortest Paths: Practical Improvements

Practical improvements – Part II

- Maintain only one array M[v] = shortest v-t path that we have found so far.

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 that includes a node that reaches t.

Pf. If there is a negative cycle, we would have OPT(j,v) < OPT(n-1,v) for some v and 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

Detecting Negative Cycles

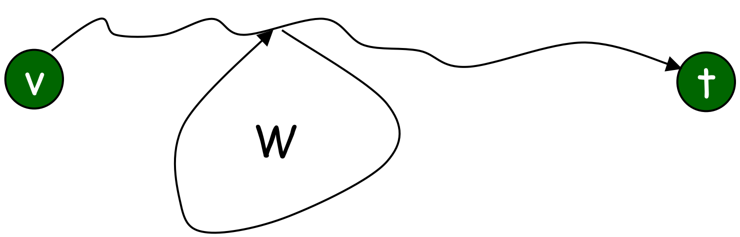
Lemma. If OPT(n,v) < OPT(n-1,v) for some node v, then there is a negative cycle W in the graph that includes a node that reaches t .

Pf.

- 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.

- Since only has n nodes in the graph, some node is visited twice in P

- Deleting W yields a v-t path with < n edges
  $\Rightarrow$ has cost >= OPT(n-1,v) > OPT(n,v) = cost(P)
  $\Rightarrow$ W has negative cost.



c(W) < 0

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.

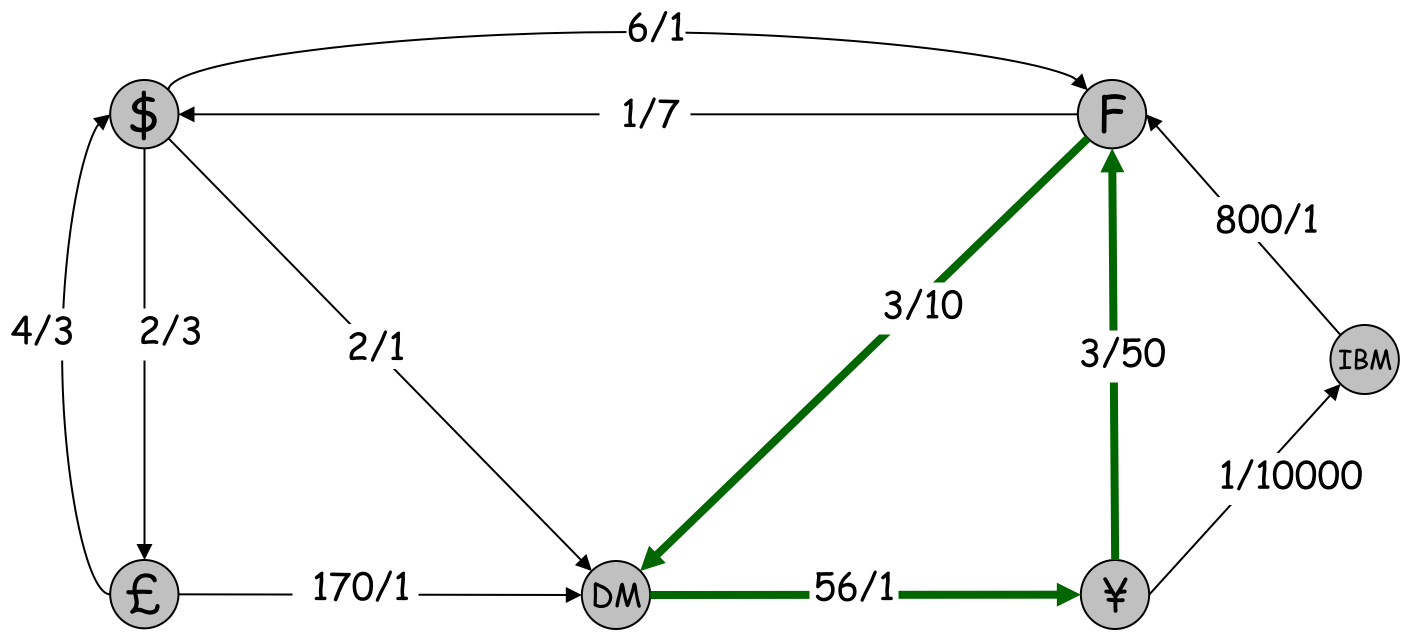
Detecting Negative Cycles

```
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
             }
          }
         if no M[v] value changed in iteration i, stop.
    }
    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?

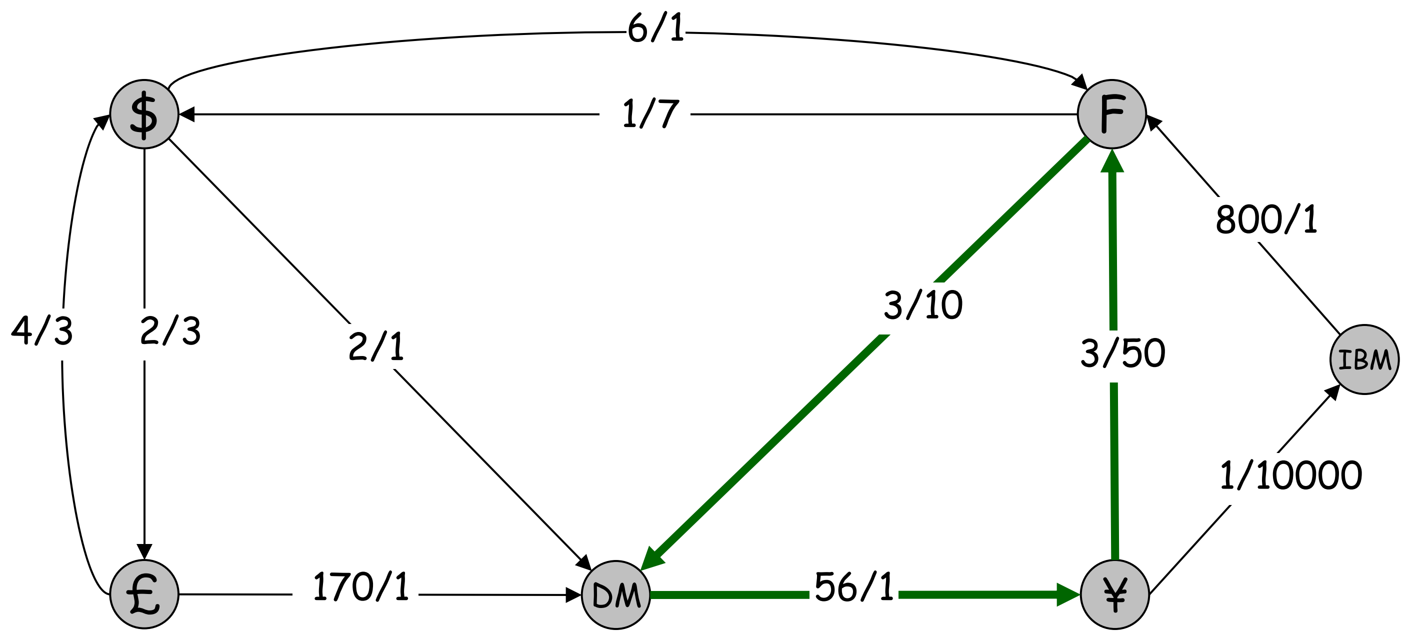
Remark. Fastest algorithm very valuable!



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!



cost(e) = - log e

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