Shortest Paths with Dynamic Programming

Bellman-Ford Algorithm

Paul Beame

Shortest Path Problem

- Dijkstra's Single Source Shortest Paths Algorithm
  - O(mlog n) time, positive cost edges
- Bellman-Ford Algorithm
  - O(mn) time for graphs with negative cost edges

Shortest paths with negative cost edges

- Dijsktra's algorithm failed with negative-cost edges
  - What can we do in this case?
  - Negative-cost cycles could result in shortest paths with length **-∞**
    - but these would be infinitely long...

- What if we just wanted shortest paths of exactly **i** edges?

Shortest paths with negative cost edges (Bellman-Ford)

- We want to grow paths from **s** to **t** based on the # of edges in the path
- Let Cost(**s**,**w**,**i**)=cost of minimum-length path from **s** to **w** using exactly **i** edges.
  - $\text{Cost}(\mathbf{s},\mathbf{w},\mathbf{0}) = \begin{cases} \mathbf{0} \text{ if } \mathbf{w}=\mathbf{s} \\ \infty \text{ otherwise} \end{cases}$
  - $\text{Cost}(\mathbf{s},\mathbf{w},\mathbf{i}) = \min_{(\mathbf{v},\mathbf{w})\in \mathbf{E}}(\text{Cost}(\mathbf{s},\mathbf{v},\mathbf{i-1})+\mathbf{c}_{\mathbf{vw}})$

Bellman-Ford

- Observe that the recursion for Cost(**s**,**w**,**i**) doesn't change **s**
  - Only store an entry for each **w** and **i**
    - $OPT_i(\mathbf{w})$
  - $OPT_0(\mathbf{w}) = \begin{cases} \mathbf{0} \text{ if } \mathbf{w}=\mathbf{s} \\ \infty \text{ otherwise} \end{cases}$
  - $OPT_i(\mathbf{w}) = \min_{(\mathbf{v},\mathbf{w}) \in \mathbf{E}}(OPT_{i-1}(\mathbf{v}) + \mathbf{c}_{\mathbf{vw}})$

Shortest paths with negative cost edges (Bellman-Ford)

- Suppose no negative-cost cycles in G
  - Shortest path from **s** to **t** has at most **n-1** edges
    - If not, there would be a repeated vertex which would create a cycle that could be removed since cycle can't have –ve cost

Algorithm, Version 1

```
foreach w
    M[0, w] = infinity;
M[0, s] = 0;
for i = 1 to n-1
    foreach w
        M[i, w] = min_v(M[i-1,v] + cost[v,w]);
```

What if we want to allow **up to** **i** edges rather than require exactly **i** edges?

Algorithm, Version 2

```
foreach w
    M[0, w] = infinity;
M[0, s] = 0;
for i = 1 to n-1
    foreach w
        M[i, w] = min(M[i-1, w], min_v(M[i-1,v] + cost[v,w]))
```

Now $M[i,w] \leq M[i-1,w] \leq \ldots \leq M[0,w]$.
If all we only care about is finding short paths we can use the shortest length we have found and forget # of hops

Algorithm, Version 3

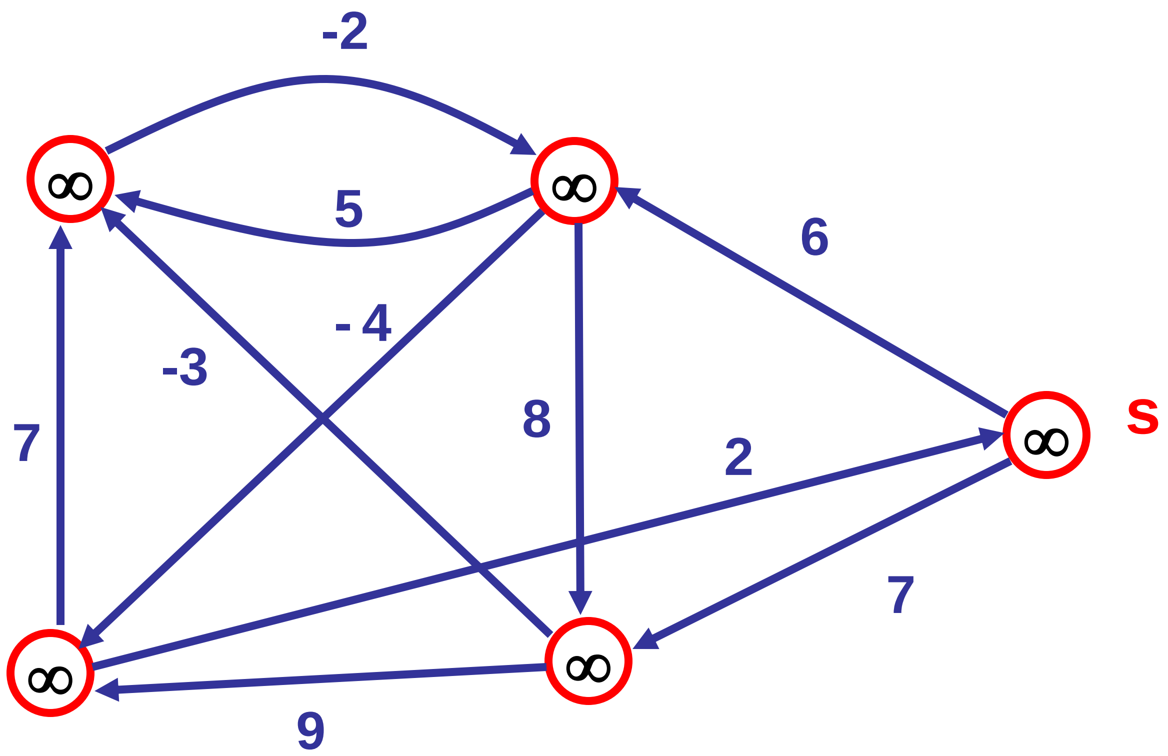
```
foreach w
    M[w] = infinity;
M[s] = 0;
for i = 1 to n-1
    foreach w
        M[w] = min(M[w], min_v(M[v] + cost[v,w]))
```

Correctness Proof for Algorithm 3

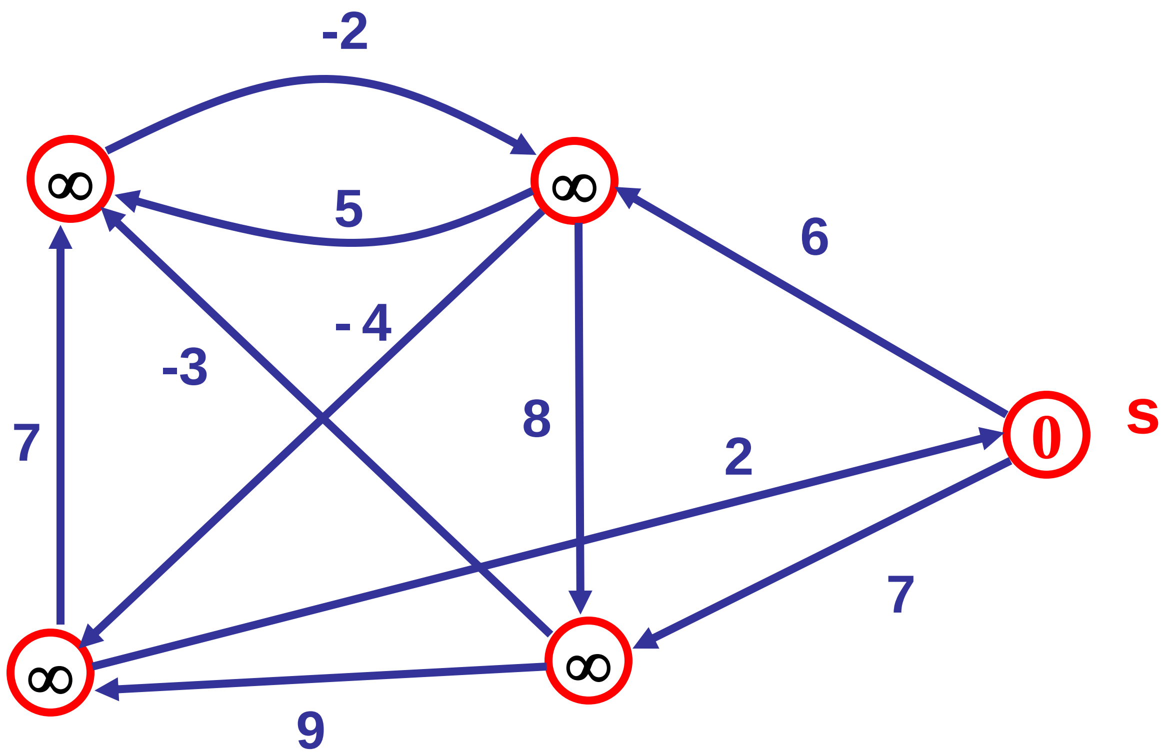
- Key lemma – at the end of iteration i, for all w, M[w] ≤ M[i, w];

- Reconstructing the path:
  - Set P[w] = v, whenever M[w] is updated from vertex v

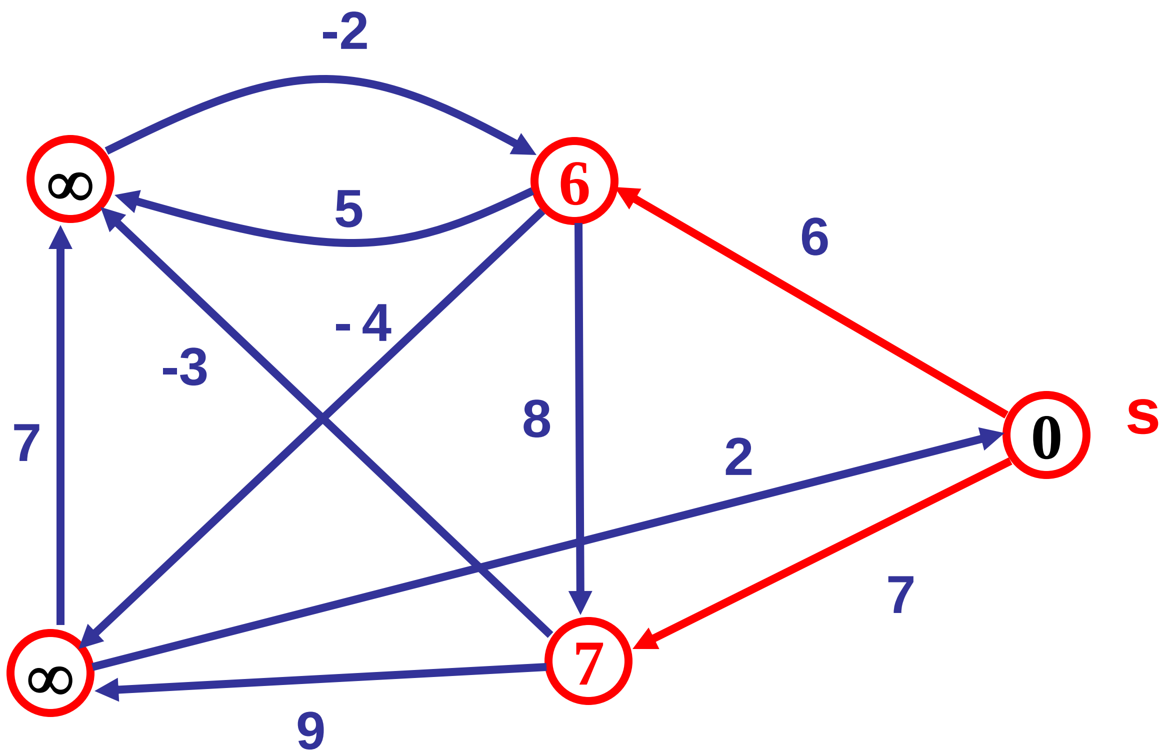
Bellman-Ford



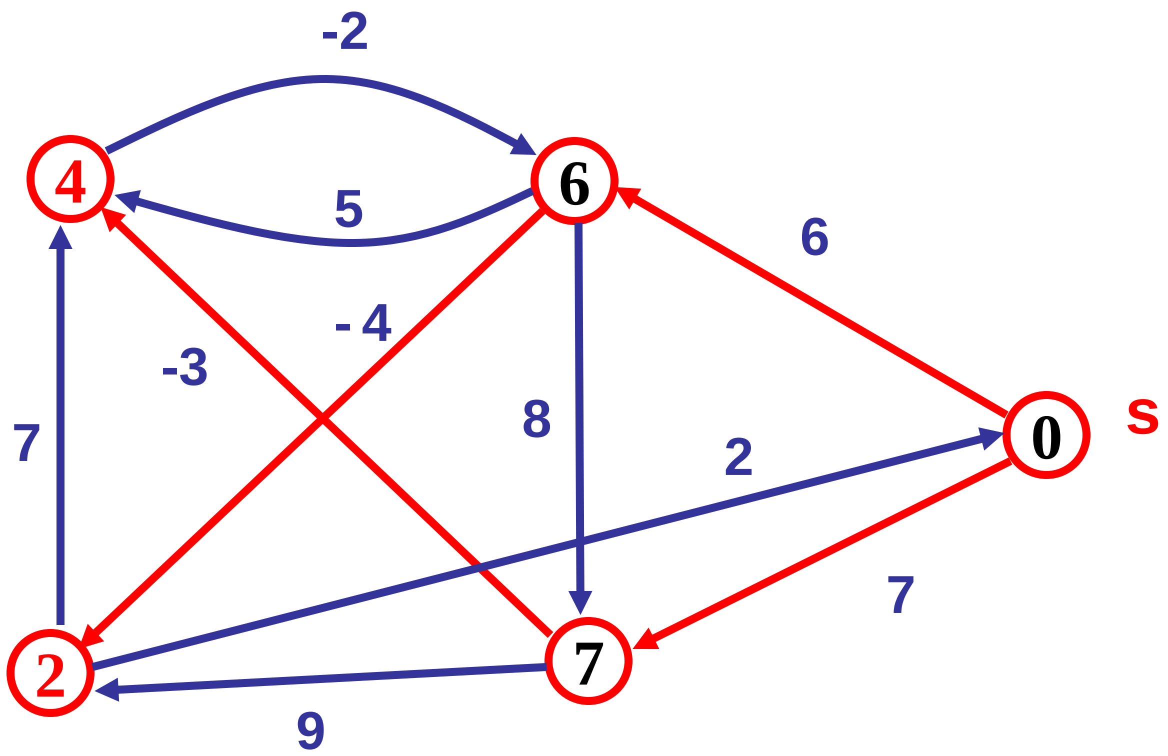
Bellman-Ford



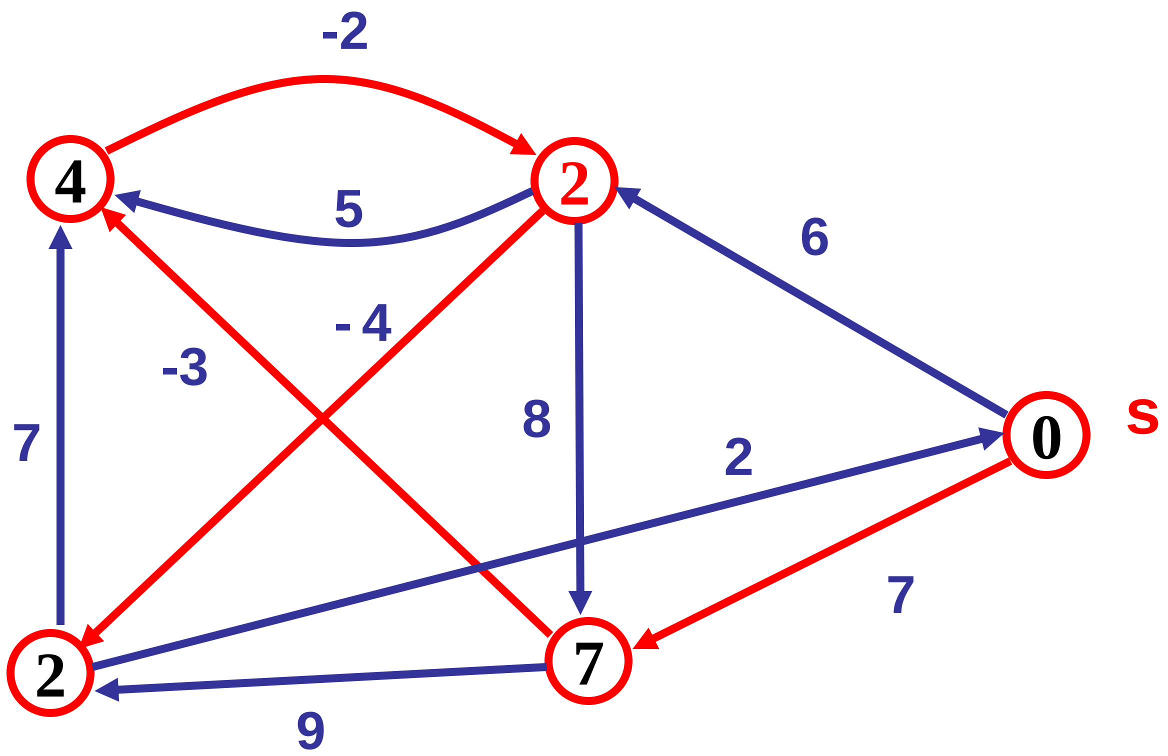
Bellman-Ford



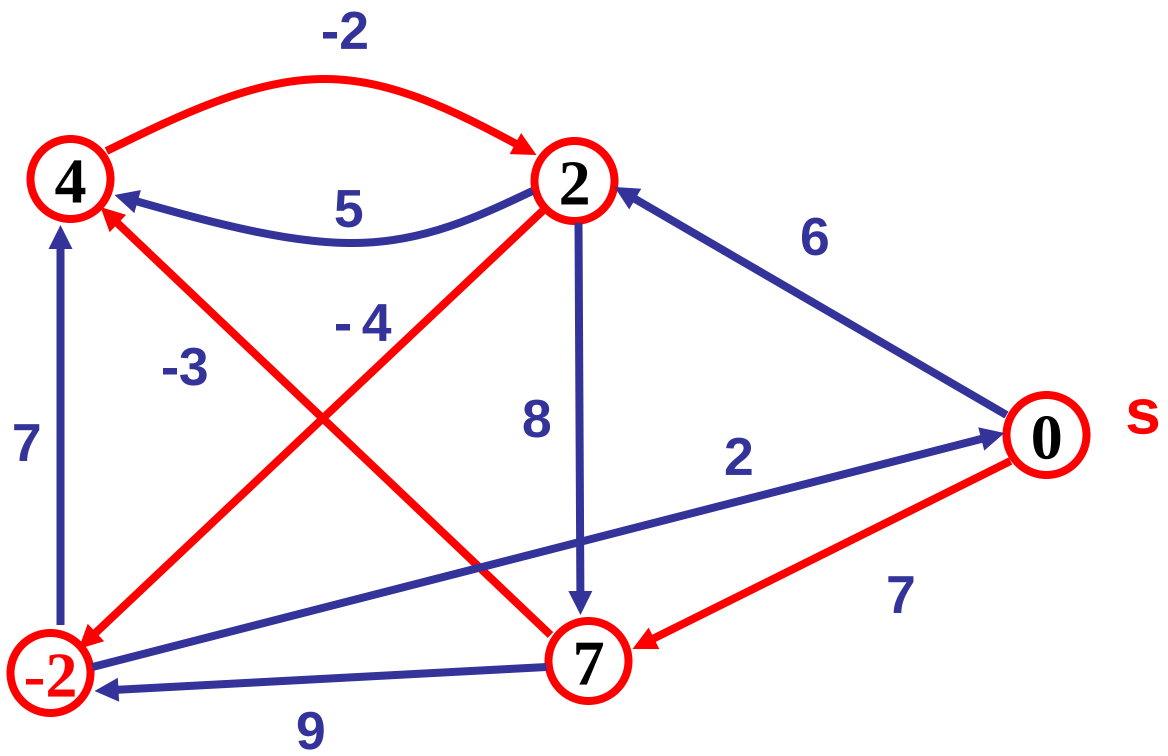
Bellman-Ford



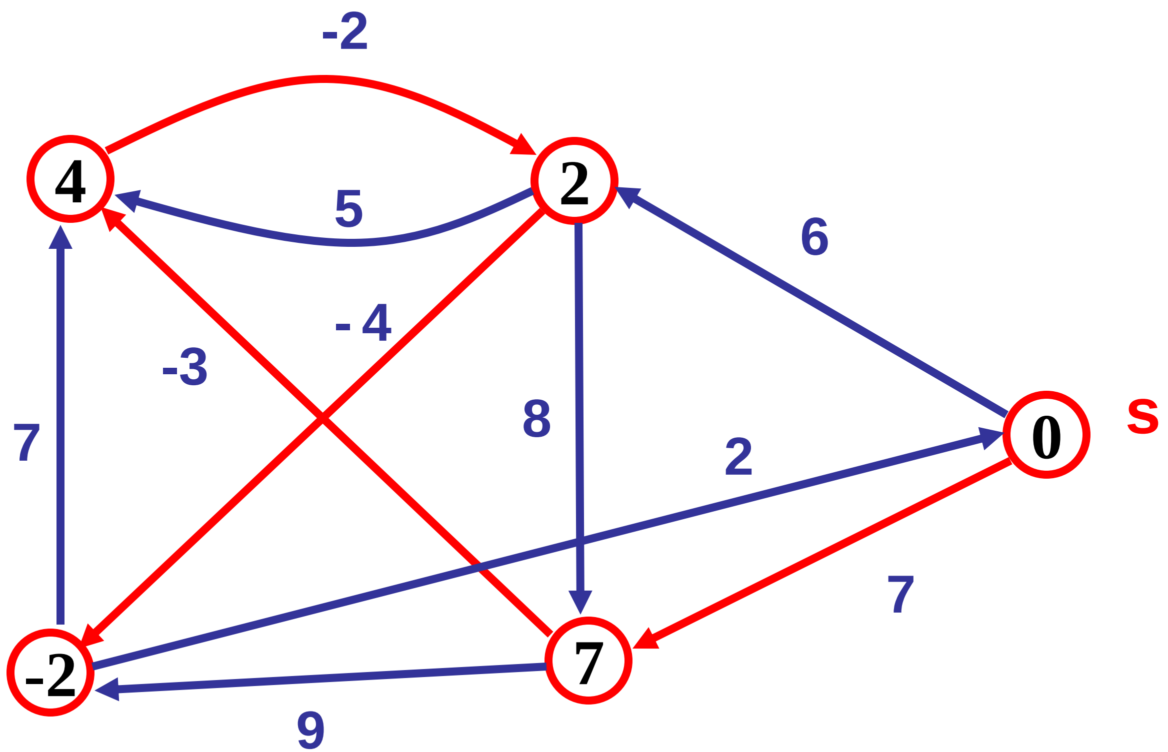
Bellman-Ford



Bellman-Ford



Bellman-Ford

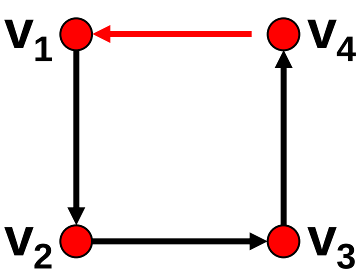


Other details

- Can run algorithm and stop early if M doesn't change in an iteration
  - Even better, one can update only neighbors **x** of vertices **w** whose **M** value changed in an iteration

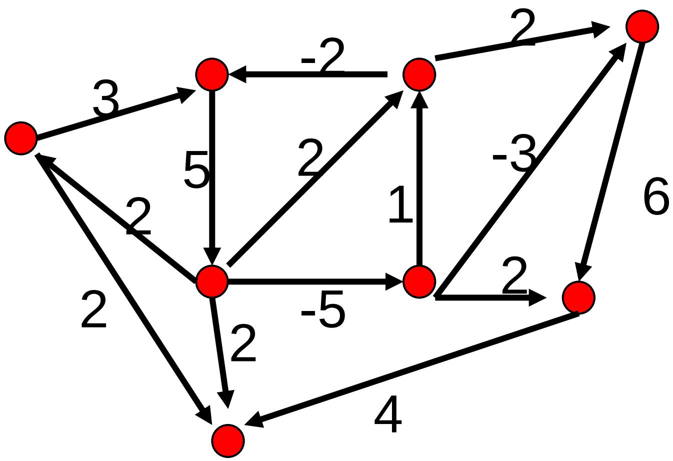
If the pointer graph has a cycle, then the graph has a negative cost cycle

- If P[w] = v then M[w] ≥ M[v] + cost(v,w)
  - Equal when w is updated
  - M[v] could later be reduced after update
- Let $\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v_k}$ be a cycle in the pointer graph with $(\mathbf{v_k}, \mathbf{v_1})$ the **last** edge added
  - Just before the update
    - $M[\mathbf{v_j}] \geq M[\mathbf{v_{j+1}}] + cost(\mathbf{v_{j+1}}, \mathbf{v_j})$ for j < k
    - $M[\mathbf{v_k}] > M[\mathbf{v_1}] + cost(\mathbf{v_1}, \mathbf{v_k})$
  - Adding everything up
    - $0 > cost(\mathbf{v_1},\mathbf{v_2}) + cost(\mathbf{v_2},\mathbf{v_3}) + \ldots + cost(\mathbf{v_k}, \mathbf{v_1})$

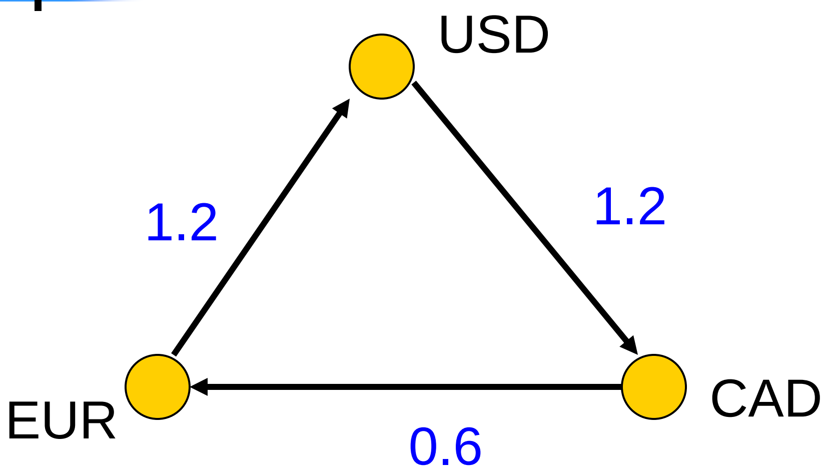


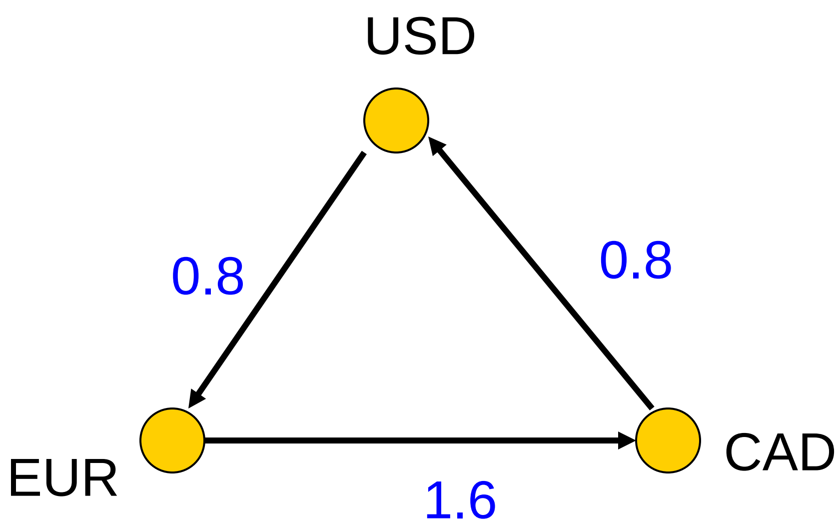
Finding negative cost cycles

- What if you want to find negative cost cycles?



Foreign Exchange Arbitrage





	USD	EUR	CAD
USD	------	0.8	1.2
EUR	1.2	------	1.6
CAD	0.8	0.6	-----

Bellman-Ford with a DAG

Edges only go from lower to higher-numbered vertices

- Update distances in order of topological sort
- Only one pass through vertices required
- **O(n+m)** time