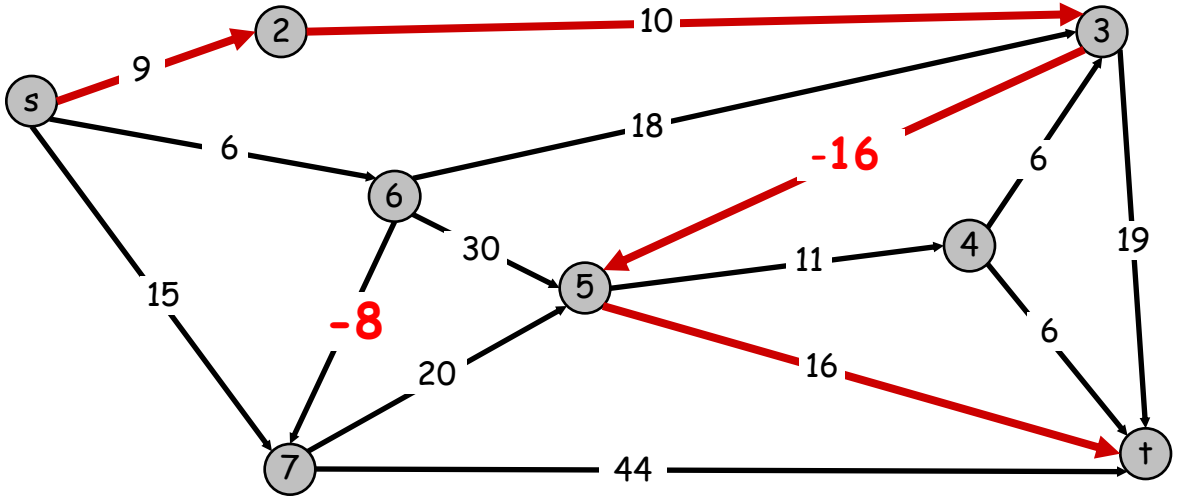
Shortest Paths in graphs with arbitrary edge weights

Cormen et. al. 24.1

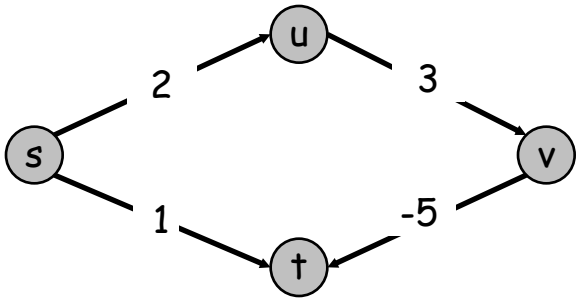
Shortest Path Problem

Shortest path problem. Given a directed graph $G = (V, E)$, with edge weights c_{vw}, find shortest path from node s to node t.
This time we allow zero and negative edge weights.



Shortest Paths: Dijkstra fails for negative edges

Dijkstra SSSP can fail with negative edges. Shortest path s to t is not s → t (length 1)

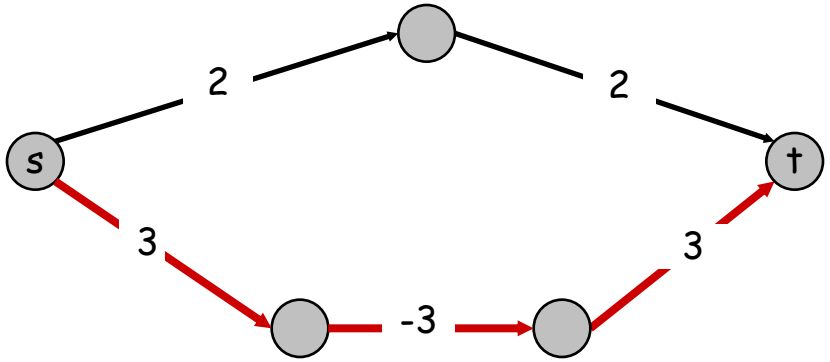


Shortest path s to t is s → u → v → t (length 0)

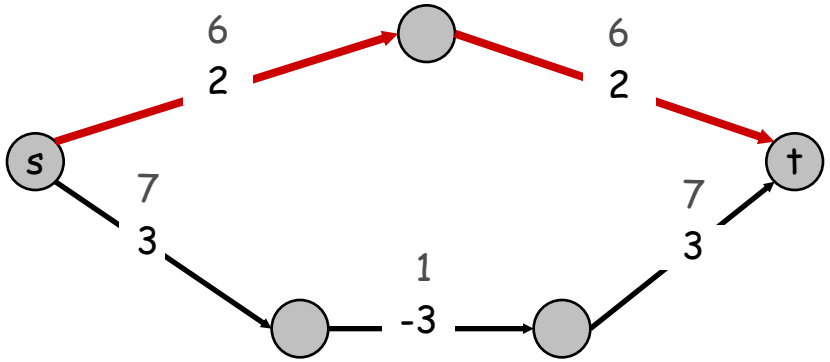
Greedy approach does NOT work here.

Shortest Paths: Failed Attempt

Re-weighting: what if we add large enough value to each edge weight so all weights>0?

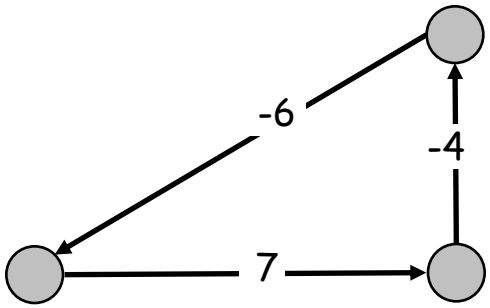


Re-weighting. Adding a constant to every edge weight fails, as the shortest path does not need to have the minimum number of edges. x

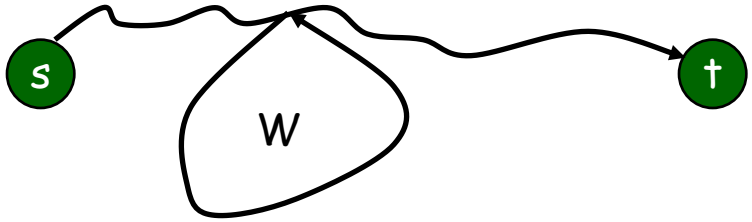


Negative Cost Cycles

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; therefore we consider only graphs without negative cycles



c(W) < 0

If there is no negative cycle the shortest path is simple (no nodes repeated).

What about 0 sum cycles?

We can ignore those, as they do not change the path length.

Bellman Ford SDSP

Goal: **SDSP, single destination shortest path**
Determine the shortest paths of all nodes to target node t

Observation: as we consider only graphs without negative cycles, and zero sum cycles do not add to the path length, we can ignore cycles in our algorithm, searching for simple shortest paths, altogether.

Therefore, a shortest path does not repeat any node. Hence, any shortest path has at most n-1 = |V|-1 edges.

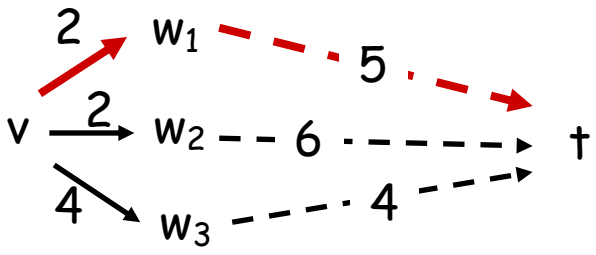
Objective: BF(G,s,t): for all nodes s and destination t, find the shortest simple path from s to t

A Dynamic Programming Approach

OPT(i, v) = length of shortest v-t path using at most i edges. We want to create a recurrence, i.e. express OPT(i,v) in some Opt(j,w) j<i:

- Case 1: path uses at most i-1 edges.
  - OPT(i, v) = OPT(i-1, v)
- Case 2: path uses at most i edges.
  - use some edge (v,w), and then thebest w-t path using at most i-1 edges

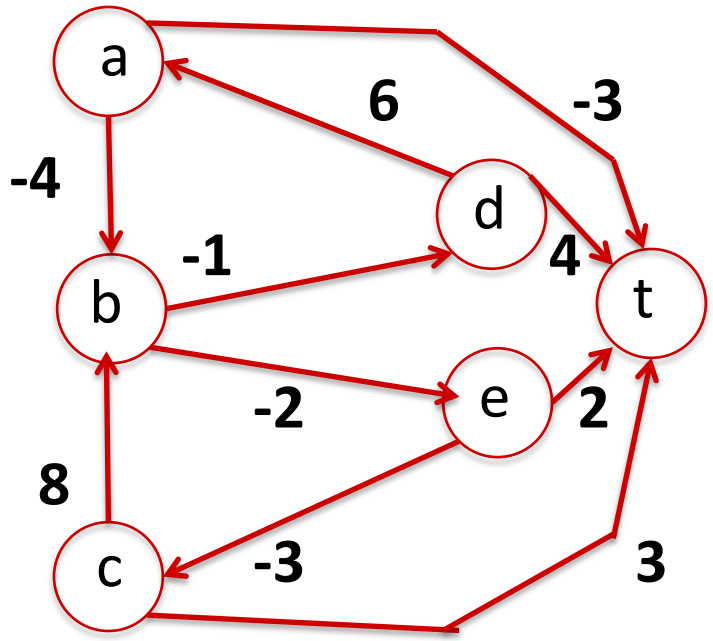
$$OPT(i, v) = \begin{cases} 0 \text{ if } v = t, \text{ otherwise } \infty & \text{if } i = 0 \\ \min\left\{ OPT(i-1, v), \min_{(v,w) \in E} \{ OPT(i-1, w) + c_{vw} \} \right\} & \text{otherwise} \end{cases}$$



What is the length of the optimal s-t path?

OPT(**n-1**, s) , n = |V|

Bellman Ford

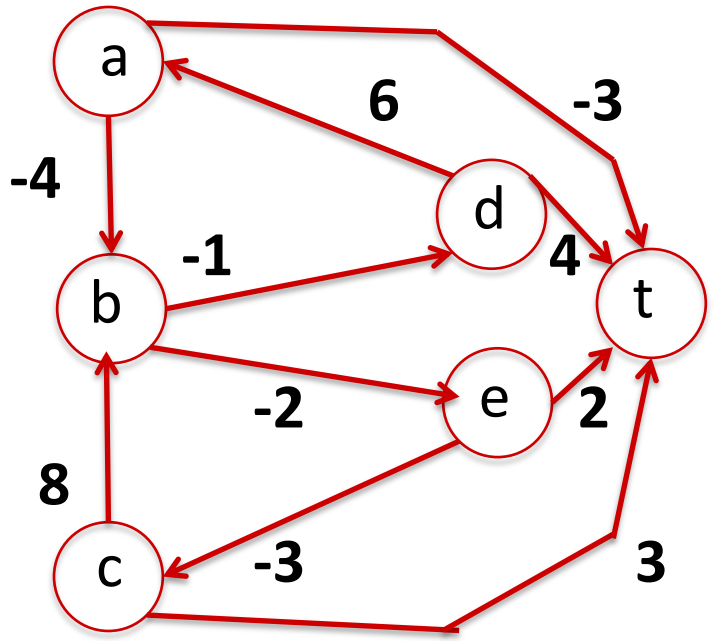


v \ i	0	1	2	3	4	5
t	0					
a	∞					
b	∞					
c	∞					
d	∞					
e	∞					

```
BF(G,s,t)
  n = |V|
  array M[0..n-1,V]
  M[0,t]=0    M[0,v]=∞ for all v != t
  for i = 1 to n-1
     compute M[i,v] using the recurrence
  return M[n-1,s]
```

```
foreach node v
   foreach edge (v, w) ∈ E
    M[i,v] ← min(M[i-1,v],
                 M[i-1,w]+c_vw)
```

Bellman Ford

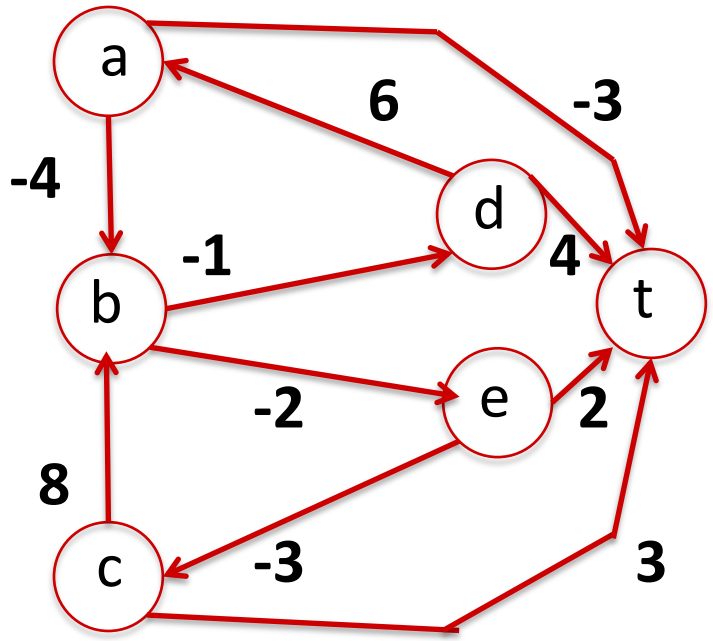


v \ i	0	1	2	3	4	5
t	0	0				
a	∞	-3				
b	∞	∞				
c	∞	3				
d	∞	4				
e	∞	2				

```
foreach node v
  foreach edge (v, w) ∈ E
   M[i,v] ← min(M[i-1,v],
                M[i-1,w]+c_vw)
```

i=1: find shortest path with <=1 edge
M[1,t]=0 (∄edge (t,w))
M[1,a]=min(∞,M[0,t]+(-3))= -3 (via b: ∞)
M[1,b]=min(M[0,b]= ∞,
via d: M[0,d]+(-1)=∞-1= ∞,
via e: M[0,e]+(-2)=∞-2= ∞
) = ∞
M[1,c]=min(∞,M[0,t]+(3))= 3 (via b: ∞)
M[1,d]=min(∞,M[0,t]+(4))= 4
M[1,e]=min(∞,M[0,t]+(2))= 2 (via c: ∞)

Bellman Ford

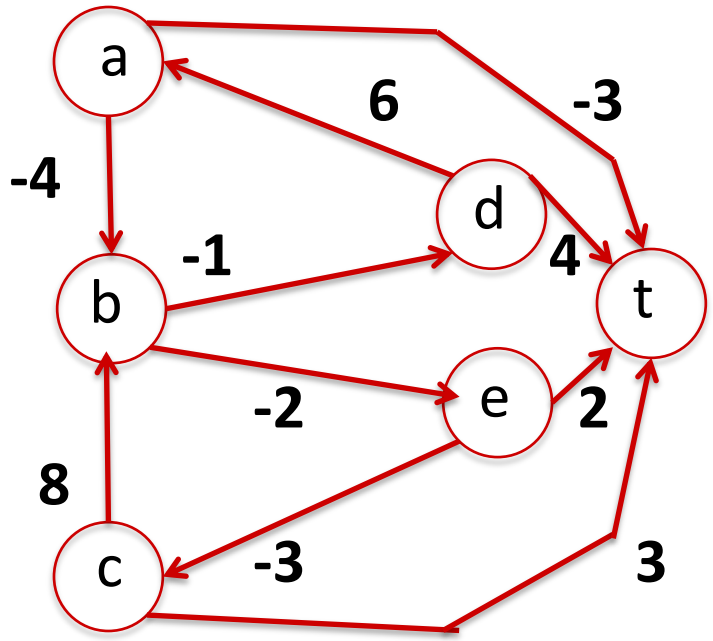


v \ i	0	1	2	3	4	5
t	0	0	0			
a	∞	-3	-3			
b	∞	∞	0			
c	∞	3	3			
d	∞	4	3			
e	∞	2	0			

```
foreach node v
   foreach edge (v, w) ∈ E
    M[i,v] ← min(M[i-1,v],
                 M[i-1,w]+c_vw)
```

i=2: find shortest path with <=2 edges
M[2,t]=0 (∄edge (t,w))
M[2,a]=min(-3,M[1,t]+(-3))= -3 (via b: ∞)
M[2,b]=0 (bet:0, bdt:3)
M[2,c]=min(3,M[1,t]+(3))= 3 (via b: ∞)
M[2,d]=min(4,M[1,a]+(6))= 3
M[2,e]=min(2,M[1,c]+(-3))= 0

Bellman Ford



v \ i	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

For each column all edges are inspected:
$O(nm)$ time, $O(n^2)$ space. $n=|V|$, $m=|E|$

Practical Improvements

Space improvements.

- Since we only refer to the previous column, we can make a space improvement: instead of keeping the whole table we keep previous and current column. At the end of a sweep previous becomes current, and current gets updated in the next sweep.
- We can even use only ONE column, and update nodes using the latest value. This will sometimes get us an update one sweep earlier. Now we only need to maintain M[v] = length shortest v-t path that we have found so far. Update is now:

  $$\texttt{M[v]} \leftarrow \texttt{min } \{ \texttt{M[v]}, \texttt{M[w]} + c_{vw} \}$$

- The role of i is only as a counter
- Space complexity is now $O(n)$

Bellman-Ford: Efficient Implementation

```
// Uses only knowledge of neighboring nodes.
Shortest-Path(G, s, t) {
    foreach node v ∈ V:
        M[v] ← ∞
        successor[v] ← None
    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[w] value changed in iteration i, stop.
```

Detecting Negative Cycles

Bellman-Ford can be used to detect negative cycles by running it one more iteration.

Lemma. If $OPT(n,v) = OPT(n-1,v)$ for all v, then there is no negative cycle on a path to t.

because if there is a negative cycle, we can keep bringing OPT(i,v) down

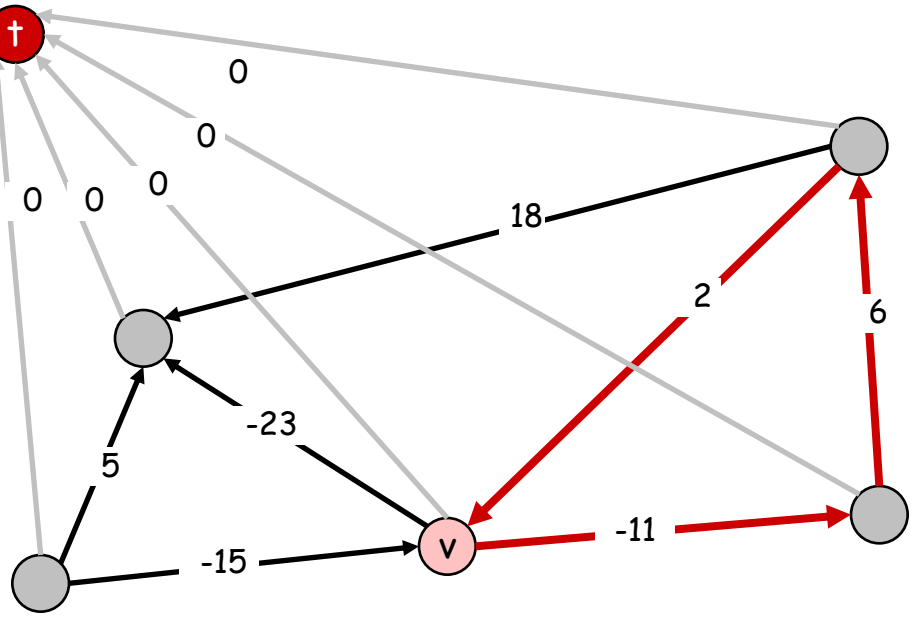
Lemma. If $OPT(n,v) < OPT(n-1,v)$ for some node v, then there is a negative cycle on a path to t.

because, as argued before, without negative cycles the optimum path length has at most n-1 edges

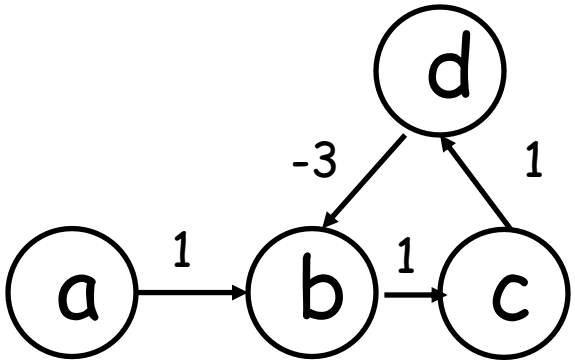
Detecting Negative Cycles

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

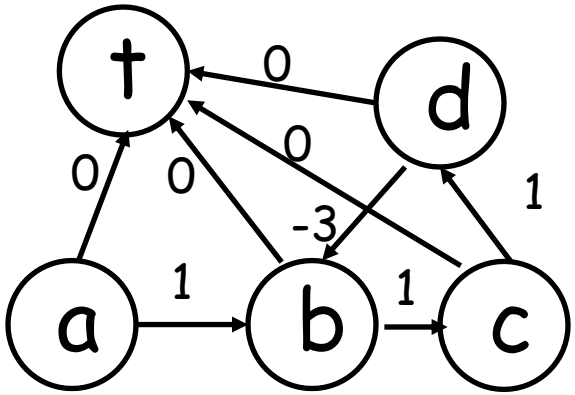
- Add new node t and connect all nodes to t with 0-cost edge.
- Check if OPT(n, v) = OPT(n-1, v) for target node t.
  - if so, then there are no negative cycles
  - if not, then there is a negative cycle



Do Bellman Ford for Negative cycles



Do Bellman Ford for Negative cycles



Augment with t

	i=0	i=1	i=2	i=3	i=4	i=5
t	0	0	0	0	0	0
a	∞	0	0	0	0	0
b	∞	0	0	0	-1 bcdbt	-1
c	∞	0	0	-2 cdbt	-2	-2
d	∞	0	-3 dbt	-3	-3	**-4 dbcdbt**