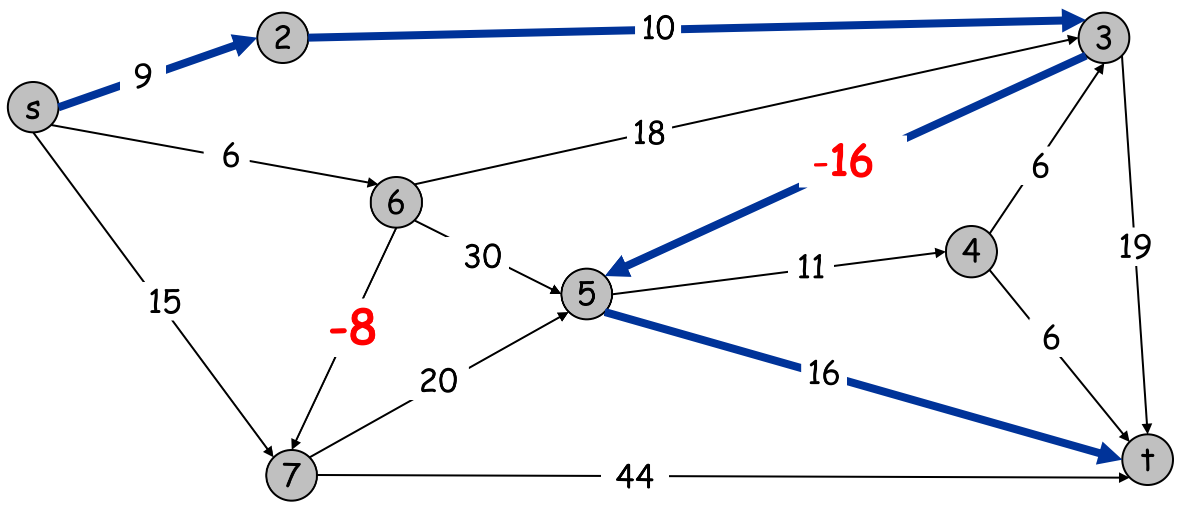
Shortest Paths with Arbitrary Edge Weights

Cormen et. al.24.1

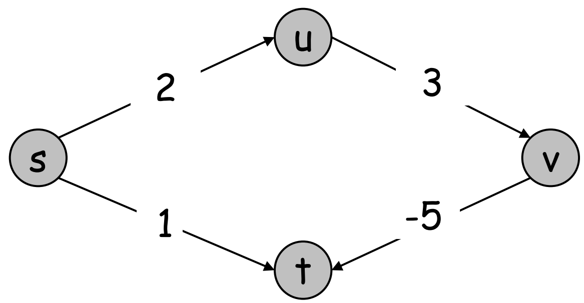
Shortest Path Problem

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.

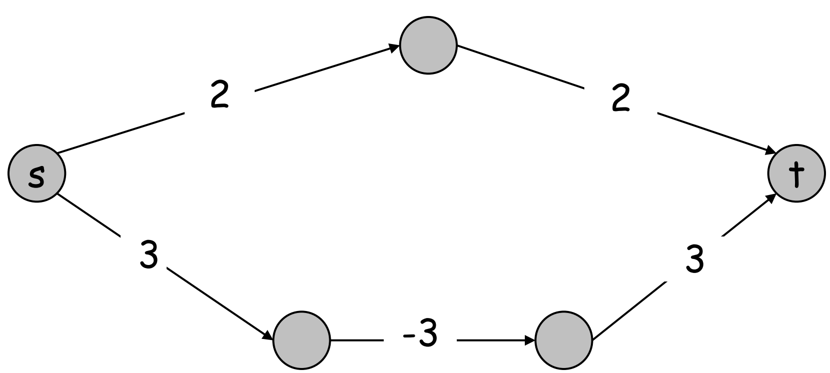


Shortest Paths: Failed Attempts

Dijkstra can fail with negative edge costs. Shortest path s to t is not s → t

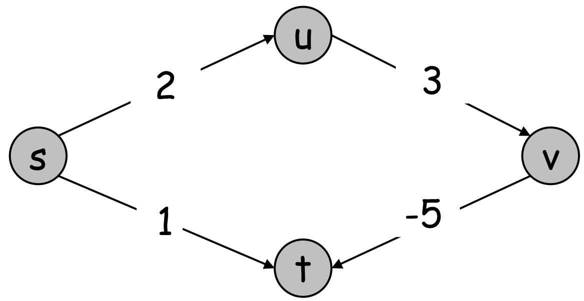


Re-weighting: what if we add large enough value to each edge weight so all weights>0?

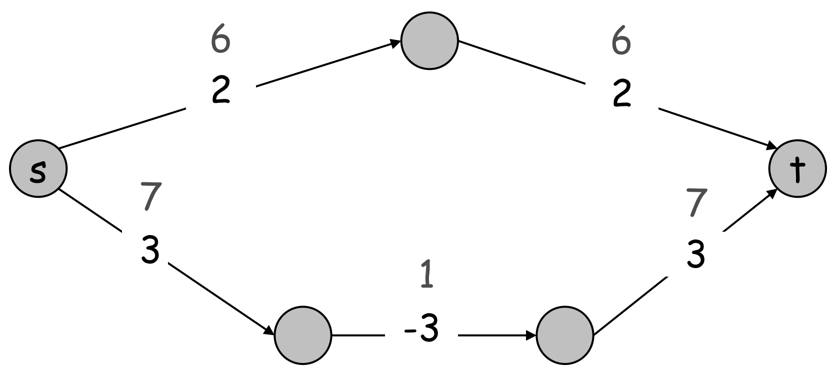


Shortest Paths: Failed Attempts

Dijkstra can fail with negative edge costs. Shortest path s to t is not s → t

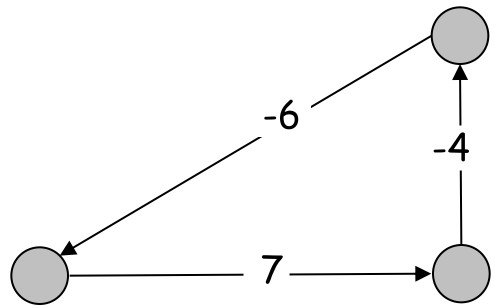


Re-weighting. Adding a constant to every edge weight can fail. Shortest path does not have the minimum number of edges.

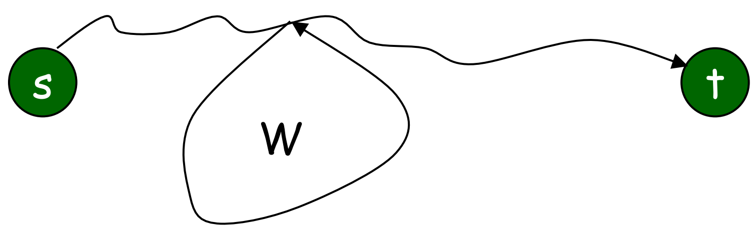


Negative Cost Cycles

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 with no negative cycles



$c(W) < 0$

If there is no negative cycle the shortest path is simple (no nodes repeated). **What about 0 sum cycles?**

Bellman Ford SDSP

Observation: as there are no negative cycles, and a zero sum cycle does 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.

Therefore any shortest path has at most n-1 = |V|-1 edges.

Objective: shortest path from node s to node t

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

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

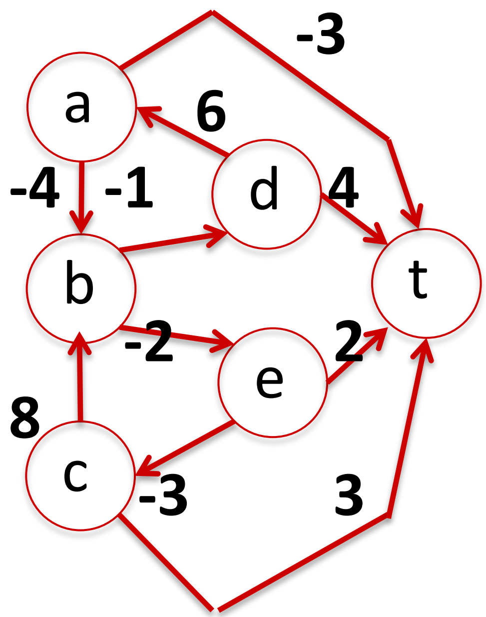
- Case 1: path uses at most i-1 edges.
  - OPT(i, v) = OPT(i-1, v)

- Case 2: path uses up to i edges.
  - use edge (v,w), and then best w-t path using i-1 edges

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

What is the length of the optimal s-t path? OPT(n-1, s) , n = |V|

Bellman Ford

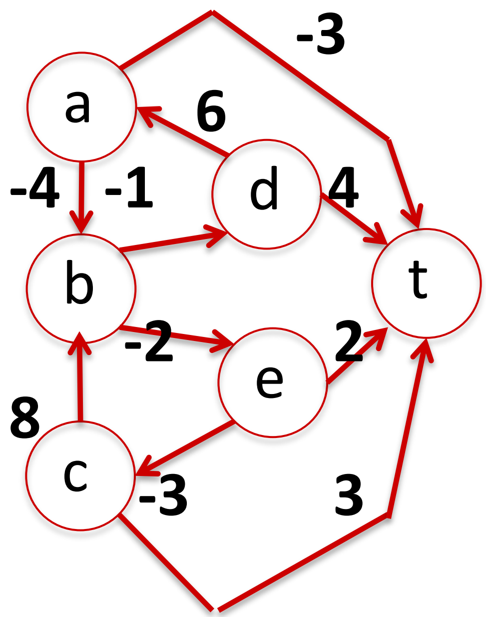


v \ i	0	1	2	3	4	5
t	0					
a	∞					
b	∞					
c	∞					
d	∞					
e	∞					

```
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 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	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

O(mn) time, O(n^2) space. n=|V|, m=|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)
```

Practical Improvements

Practical improvements.

- Since we only refer to the previous column, we only need to maintain M[v] = length shortest v-t path that we have found so far.

- Update is now:

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

- The role of i is only as a counter

Bellman-Ford: Efficient Implementation

```
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 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.
   }
}
```

Detecting Negative Cycles

Comment. 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 path length (in edges) is at most n-1

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 node t.
  - if so, then no negative cycles
  - if not, then extract cycle from shortest path from v to t