Bellman Ford Algorithm

→ Return shortest path if no negative weight cycle present

→ Else identify the presence of negative weight cycle

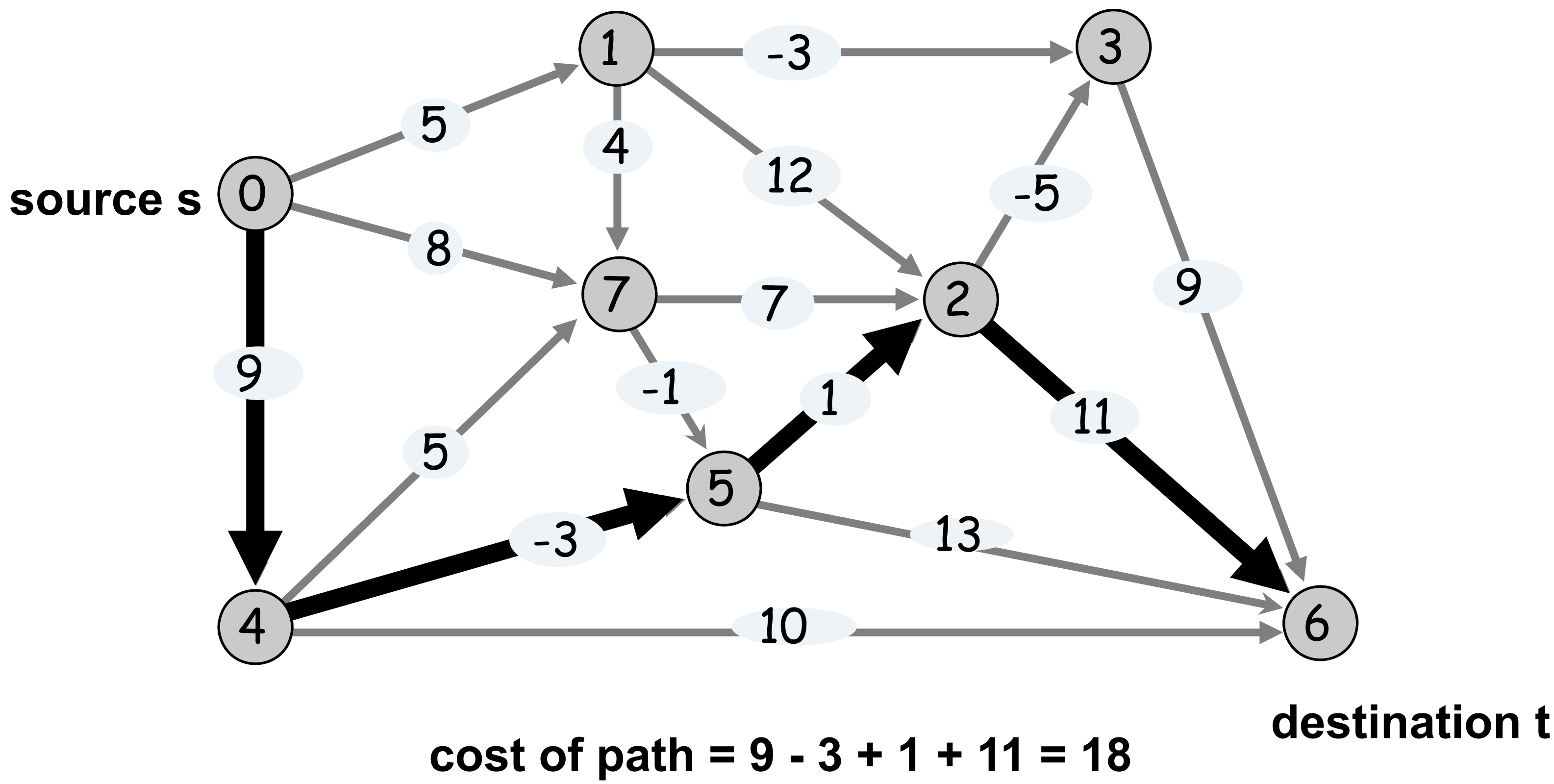
6.8 Shortest Paths

WARNING:
Watch for ambiguous use of "shortest"
in the sense of least weight/cost, and
in the sense of fewest edges.

Shortest paths

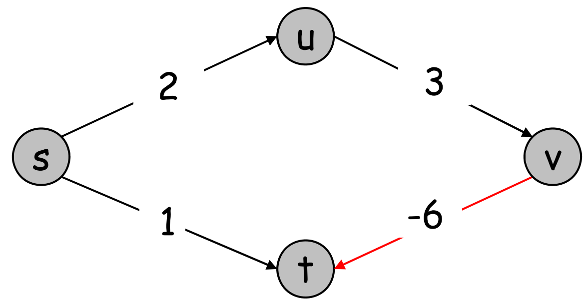
Shortest path problem. Given a digraph $G = (V, E)$, with arbitrary edge weights or costs c_{vw}, find cheapest path from node s to node t.

- Allow negative weights

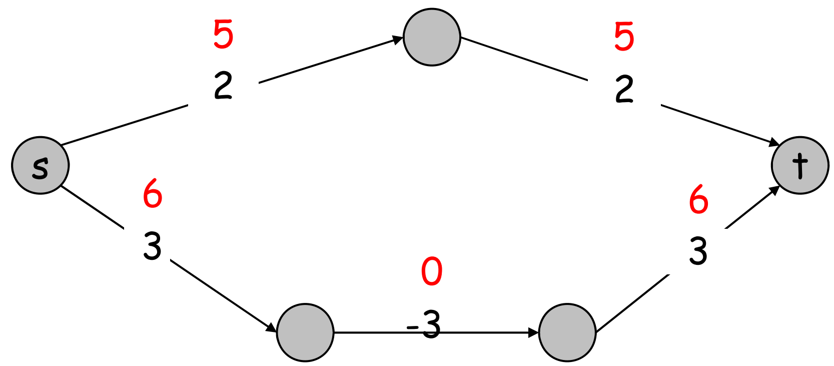


Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs/weights are present.

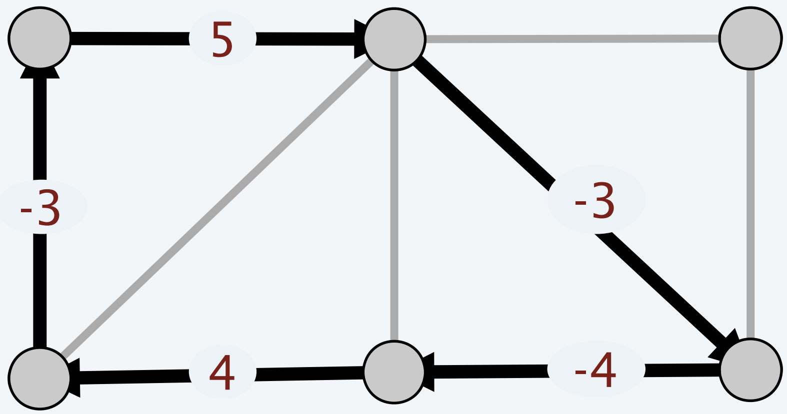


Re-weighting. Adding a constant to every edge weight can fail.



Negative cycles

Def. A negative cycle is a directed cycle such that the sum of its edge weights is negative.

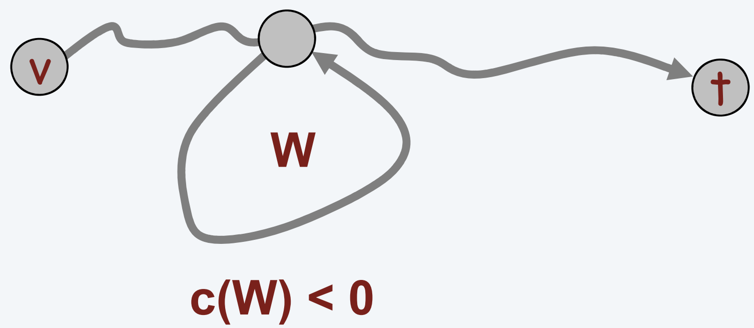


a negative cycle W : $$c(W) = \sum_{e \in W} c_e < 0$$

Shortest paths and negative cycles

Lemma 1. If some path from v to t contains a negative cycle, then there does not exist a cheapest path from v to t.

Pf. If there exists such a cycle W, then we can build a $v \leadsto t$ path of arbitrarily negative weight by detouring around the cycle as many times as desired. ▪

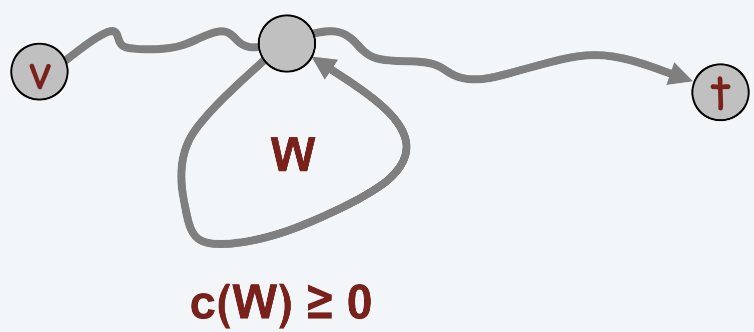


Shortest paths and negative cycles

Lemma 2. If G has no negative cycle, then there exists a cheapest path from v to t that is *simple* (and that has $\leq n-1$ edges).

Pf.

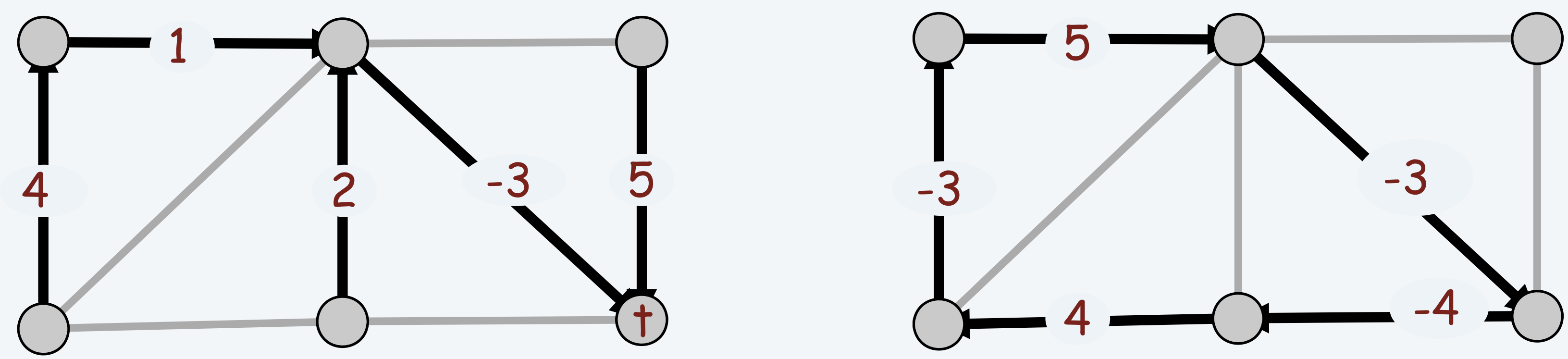
- Consider a cheapest $v \leadsto t$ path P that uses the fewest number of edges.
- If P contains a cycle W, we can remove portion of P corresponding to W, *obtaining a path with fewer edges,* without increasing the cost. ▪



Shortest path and negative cycle problems

Shortest path problem. Given a digraph $G = (V, E)$ with edge weights c_{vw} and no negative cycles, find cheapest $v \rightsquigarrow t$ path for each node v.

Negative cycle problem. Given a digraph $G = (V, E)$ with edge weights c_{vw}, find a negative cycle (if one exists).



shortest-paths tree **negative cycle**

Shortest paths: dynamic programming

Def. $OPT(i, v)$ = cost of cheapest $v \leadsto t$ path that uses $\leq i$ edges. $(v \neq t)$

- **Case 1:** Cheapest $v \leadsto t$ path uses $\leq i - 1$ edges.
  - $OPT(i, v) = OPT(i - 1, v)$

optimal substructure property
(proof via exchange argument)

- **Case 2:** Cheapest $v \leadsto t$ path uses exactly i edges.
  - if (v, w) is first edge, then OPT uses (v, w), and then selects best $w \leadsto t$ path using $i - 1$ edges (**equivalent to using $\leq i - 1$ edges given** Case 1).

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

Observation. If no negative cycles,

$$OPT(n - 1, v) = \text{cost of cheapest } v \leadsto t \text{ path.}$$

Pf. By Lemma 2, cheapest $v \leadsto t$ path is simple. ▪

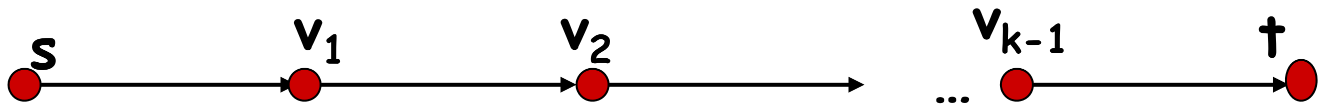
Shortest Paths: Implementation

```
Shortest-Path(G =(V,E,c), 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], c_vw + M[i-1, w]}
}
```

Analysis. $\Theta(mn)$ time, $\Theta(n^2)$ space.

Proof of Correctness : A Sketch



Induction Hypothesis: A cheapest path of length k from any node to node t can be discovered in at most k steps.

Basis Case: k = 0. (t -> t) , k = 1 (single edges v->t).

Induction Step: To construct the cheapest path of length k from some s ($s \to v_1 \to^* \to t$), use an edge ($s \to v_1$) and the cheapest path of length k-1 ($v_1 \to^* \to t$), discovered using at most k-1 steps [Ind. Hyp.]. Note that the second foreach-loop of the nested-loops is run for each edge.

Shortest paths: implementation

Theorem 1. Given a digraph $G = (V, E)$ with no negative cycles, the dynamic programming algorithm computes the cost of the cheapest $v \leadsto t$ path for each node v in $\Theta(mn)$ time and $\Theta(n^2)$ space.

Pf.

- Table requires $\Theta(n^2)$ space.
- Each iteration i takes $\Theta(m)$ time since we examine each edge once, and the number of iterations is n. ▪

Finding the shortest paths.

- Approach 1: Maintain a $successor(i, v)$ that points to next node on cheapest $v \leadsto t$ path using at most i edges.
- Approach 2: Compute optimal costs $M[i, v]$ and consider only edges with $M[i, v] = M[i - 1, w] + c_{vw}$.

Space optimization. Maintain two 1D arrays (instead of 2D array).

- $M(v)$ = cost of cheapest $v \leadsto t$ path that we have found so far.
- $successor(v)$ = next node on a $v \leadsto t$ path.

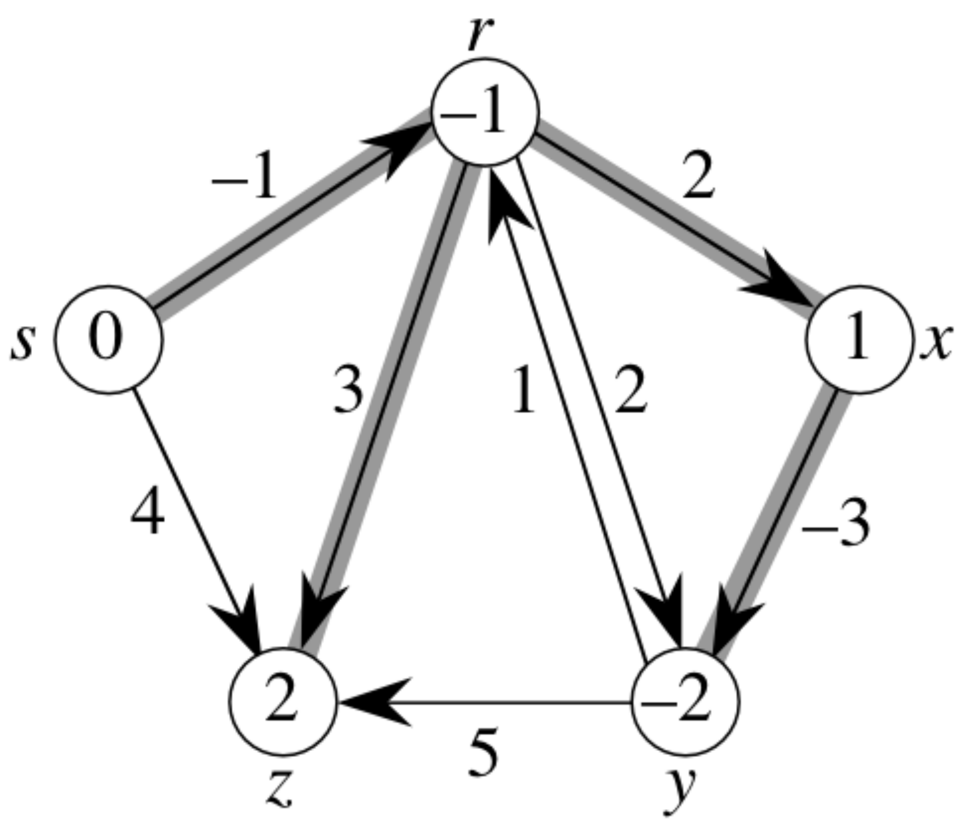
Performance optimization. If $M(w)$ was not updated in iteration $i - 1$, then no reason to consider edges entering w in iteration i.

Bellman-Ford: Efficient Implementation

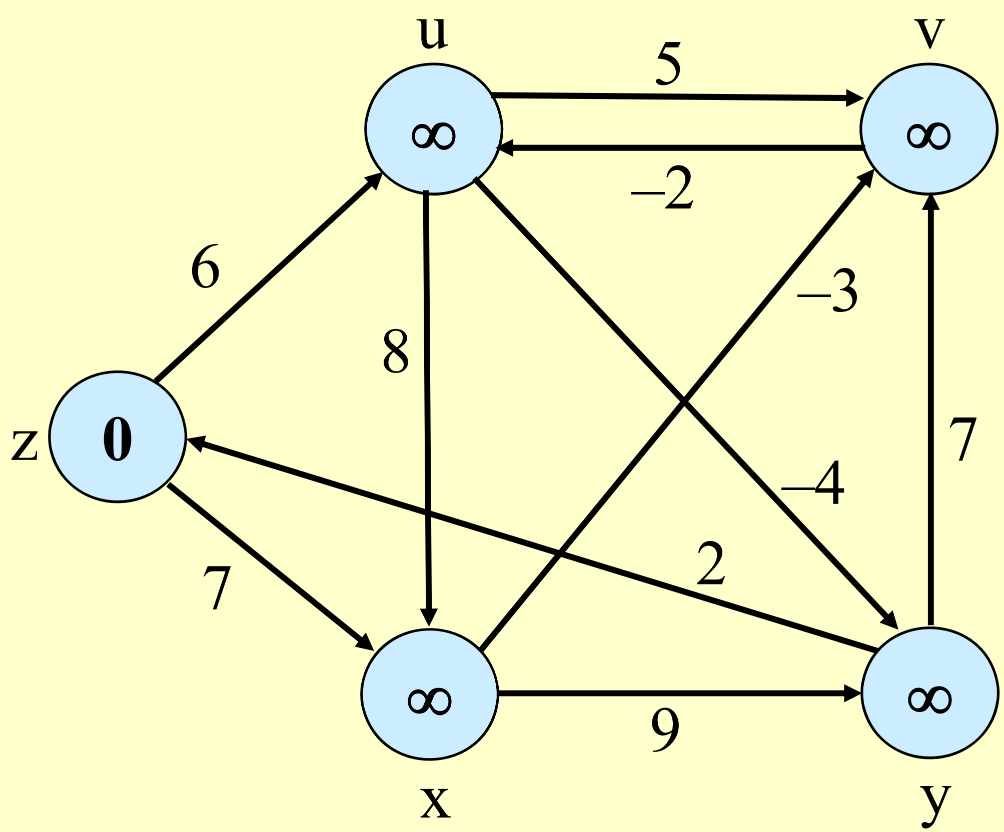
```
Bellman-Ford(G=(V,E,c), s, t) {
   foreach node v ∈ V {
      M[v] ← ∞
      successor[v] ← φ
   }

   M[t] = 0
   for i = 1 to n-1 {
     foreach node w ∈ V {
       if (M[w] was updated in previous iteration) {
         foreach edge (v, w) ∈ E {   (w, v) ∈ 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.
   }
}
```

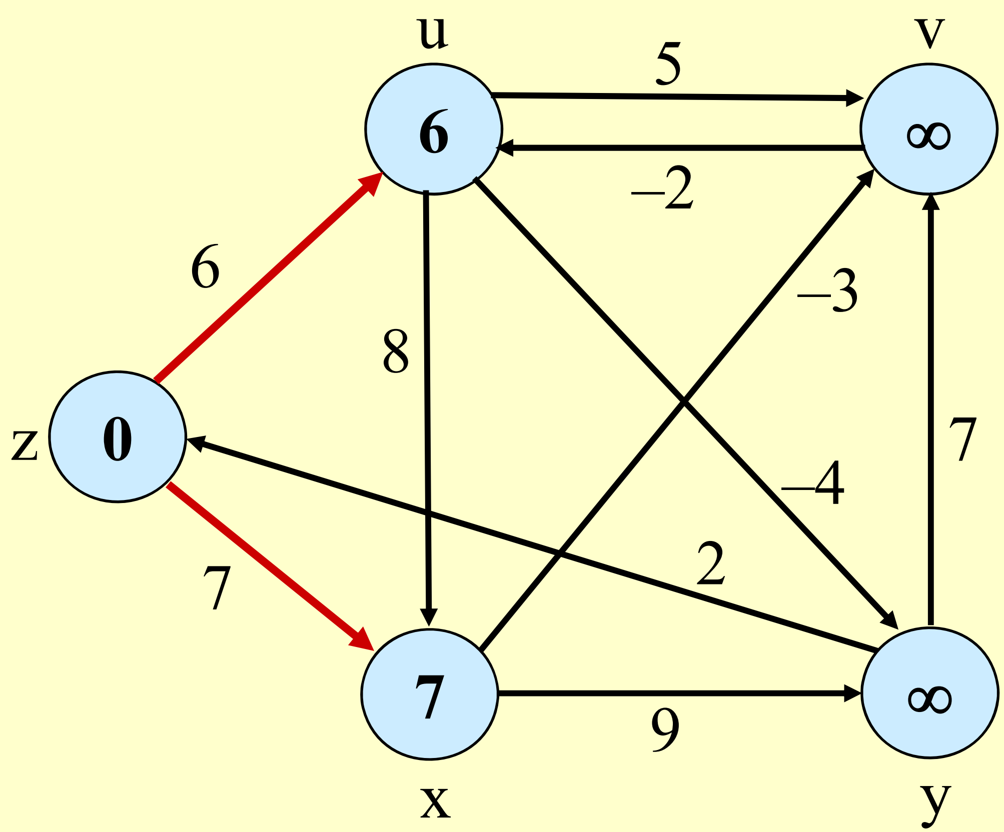
Example (shortest paths from the source s)



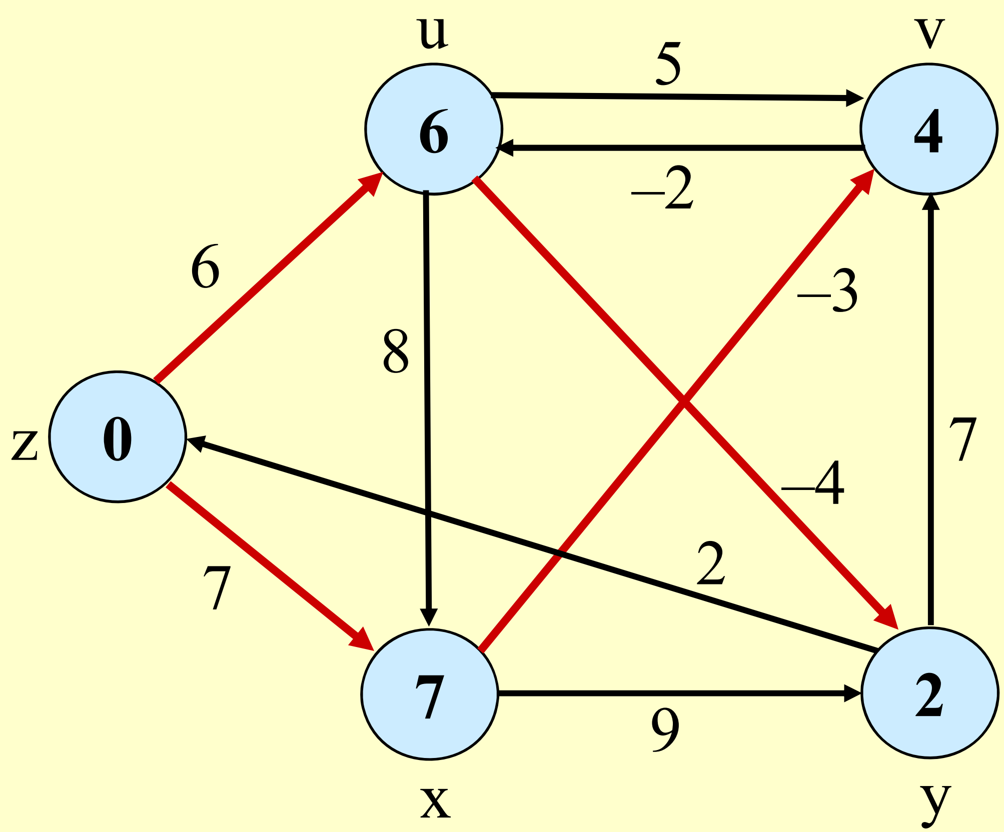
Example



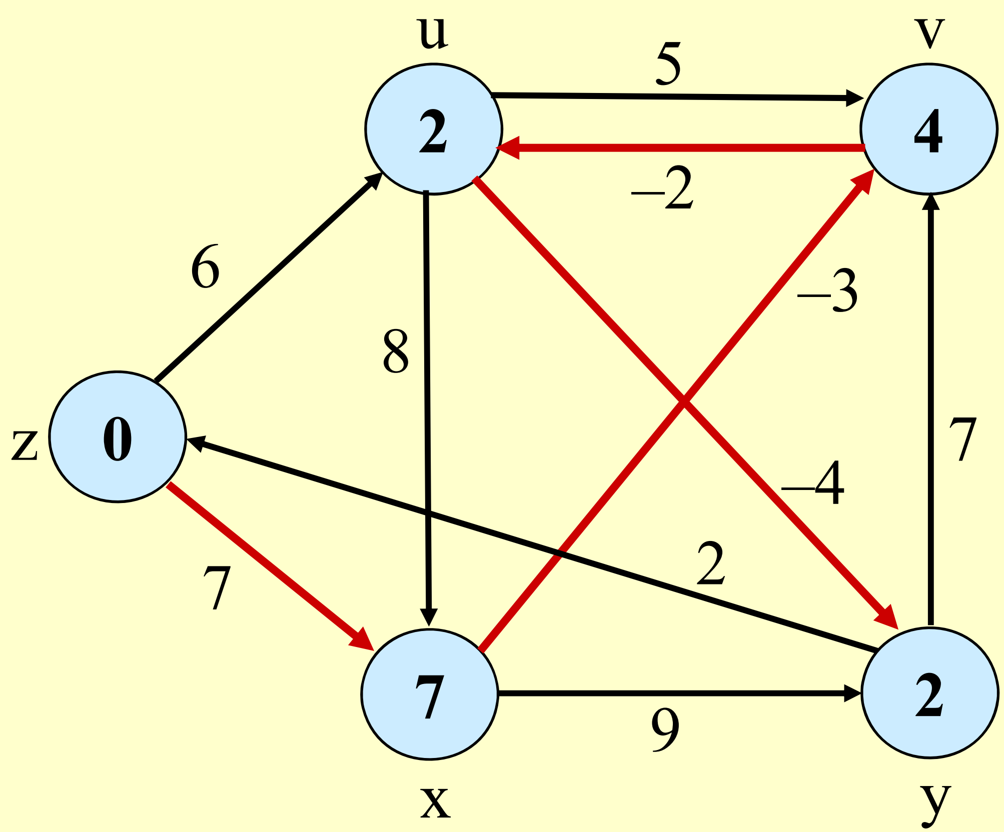
Example



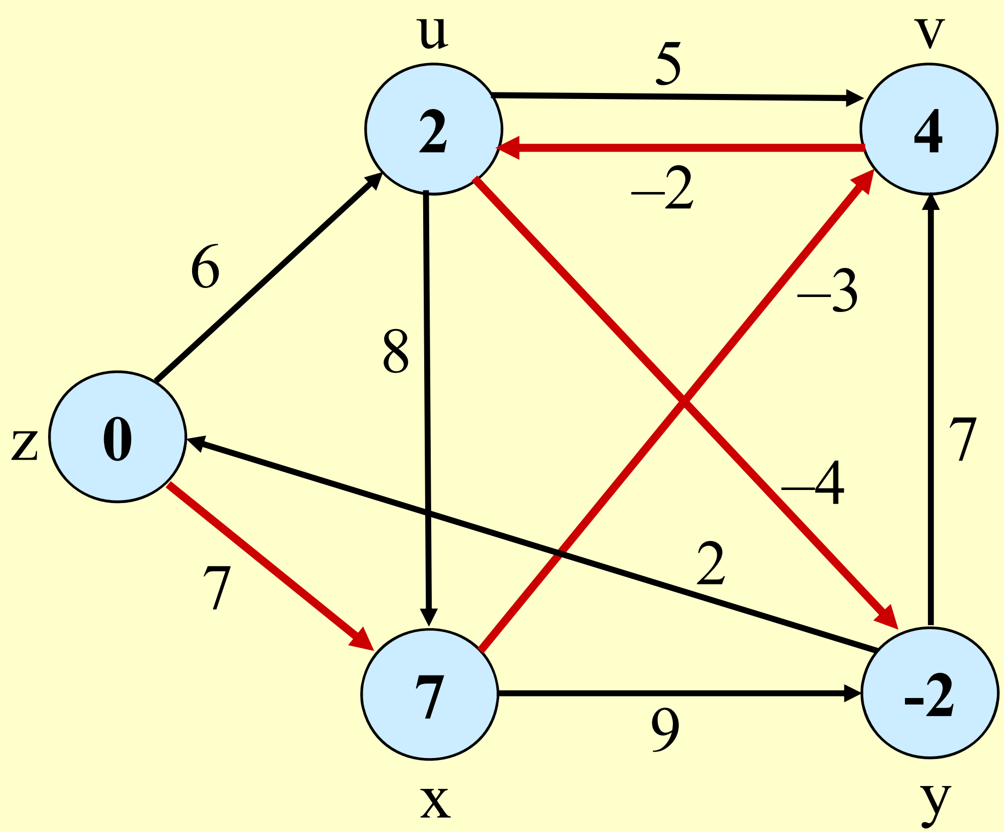
Example



Example



Example



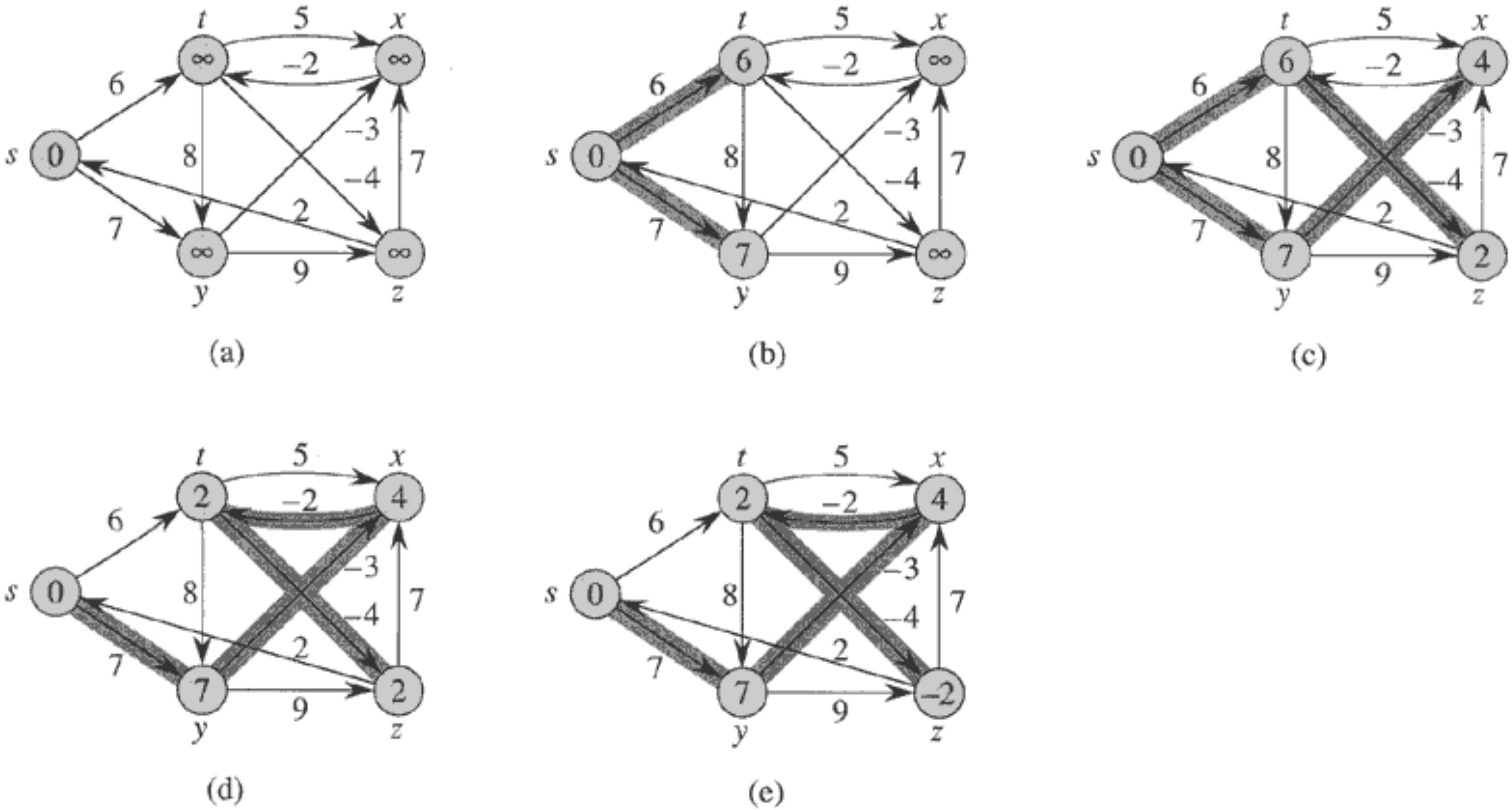


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

- Bellman-Ford returns a compact representation of the set of shortest paths from source s (resp. to destination t) to (resp. from) all other vertices in the graph reachable from s (resp. to t). This is contained in the predecessor (resp. successor) subgraph.
- If Bellman-Ford has not converged after $|V| - 1$ iterations, then there cannot be a shortest path tree, and there must be a negative weight cycle.

Another Look

Note: Bellman-Ford is essentially **dynamic programming**.

Let d(i, j) = cost of the shortest path from s to i that is at most j hops.

$$d(i, j) = \begin{cases} 0 & \textbf{if } i = s \wedge j = 0 \\ \infty & \textbf{if } i \neq s \wedge j = 0 \\ \min(\{d(k, j-1) + w(k, i) : (i,k) \in E\} \\ \qquad \cup \{d(i, j-1)\}) & \textbf{if } j > 0 \end{cases}$$

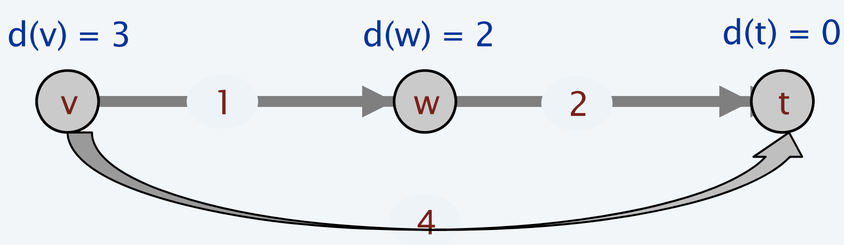
i →

j ↓	z 1	u 2	v 3	x 4	y 5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

Bellman-Ford: analysis

Claim. ~~ONLY after the i^{th} pass of Bellman-Ford IS $d(v)$ equal to the cost of the cheapest $v \leadsto t$ path using i edges.~~

Counterexample. Claim is false!



if node w is considered before node v in the 1st pass, then d(v) = 3 after the 1st pass itself

Bellman-Ford: analysis

Lemma 3. Throughout Bellman-Ford algorithm, $d(v)$ is the cost of some $v \rightsquigarrow t$ path; after the i^{th} pass, $d(v)$ is no larger than the cost of the cheapest $v \rightsquigarrow t$ path using $\leq i$ edges.

Pf. [by induction on i]

- Assume true after i^{th} pass.
- Let P be any $v \rightsquigarrow t$ path (trivially includes the cheapest) with $i + 1$ edges.
- Let (v, w) be first edge on path and let P' be subpath from w to t.
- By inductive hypothesis, $d(w) \leq c(P')$ since P' is a $w \rightsquigarrow t$ path with i edges.
- After considering v in pass i+1:

$$\begin{aligned} d(v) &\leq c_{vw} + d(w) \\ &\leq c_{vw} + c(P') \\ &= c(P) \quad \blacksquare \end{aligned}$$

Theorem 2. Given a digraph with no negative cycles, Bellman-Ford computes the costs of the cheapest $v \rightsquigarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf. Lemmas 2 + 3. ▪

can be substantially faster in practice

Bellman-Ford: analysis

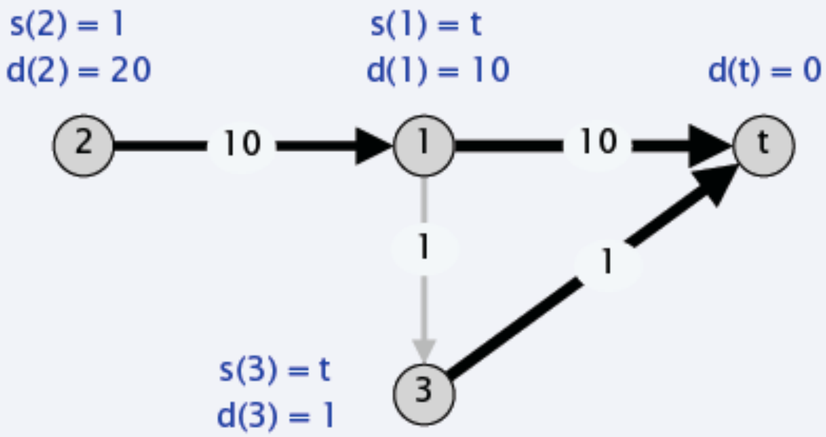
Claim. ~~Throughout the Bellman-Ford algorithm, following *successor*(v) pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \to t$ path may have strictly lower cost than $d(v)$.

Illustrating plausibility of the claim

consider nodes in order: t, 1, 2, 3



Bellman-Ford: analysis

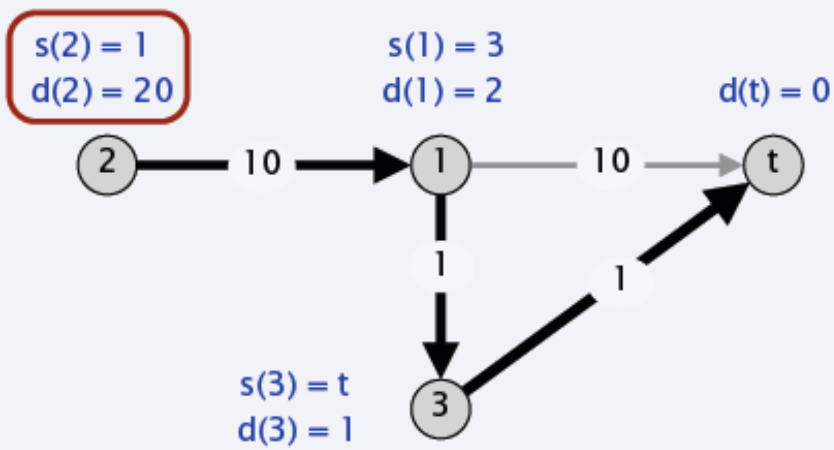
Claim. ~~Throughout the Bellman-Ford algorithm, following *successor*(v) pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \to t$ path may have strictly lower cost than $d(v)$.

consider nodes in order: t, 1, 2, 3

Nodes 2 and 3 are updated before and after node 1
Order: t, 1, 2, 3, 1, ...



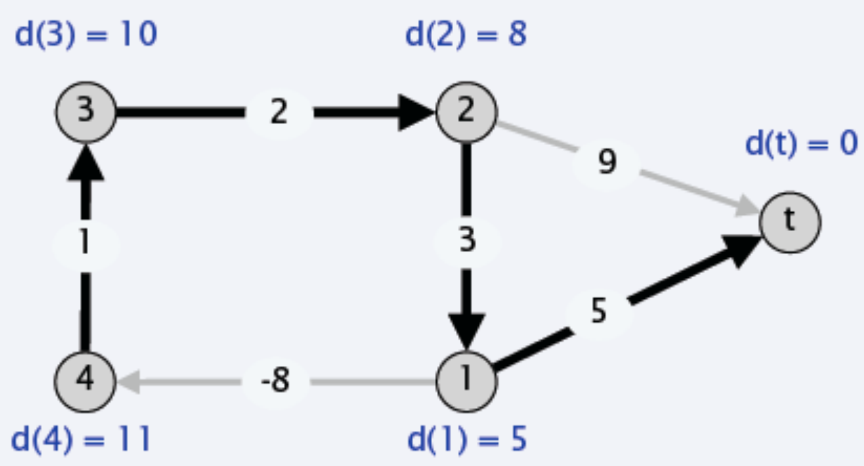
Bellman-Ford: analysis

Claim. ~~Throughout the Bellman-Ford algorithm, following $successor(v)$ pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \rightarrow t$ path may have strictly lower cost than $d(v)$.
- Successor graph may have cycles.

consider nodes in order: t, 1, 2, 3, 4



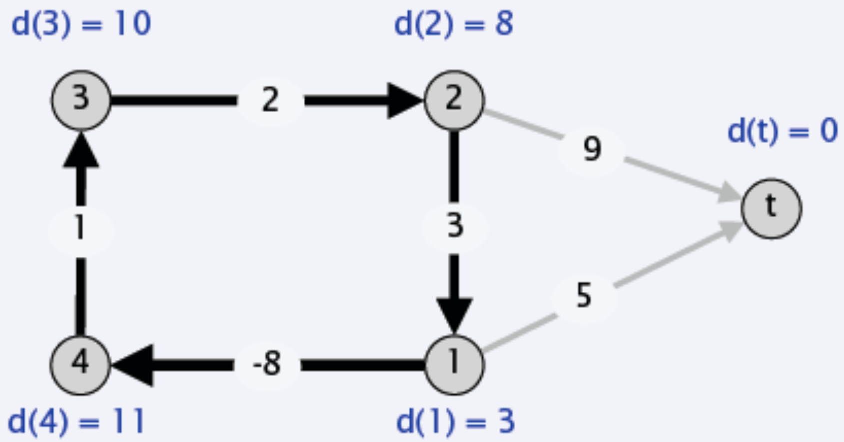
Bellman-Ford: analysis

Claim. ~~Throughout the Bellman-Ford algorithm, following $successor(v)$ pointers gives a directed path from v to t of cost $d(v)$.~~

Counterexample. Claim is false!

- Cost of successor $v \to t$ path may have strictly lower cost than $d(v)$.
- Successor graph may have cycles.

consider nodes in order: t, 1, 2, 3, 4 ,1



Bellman-Ford: finding the shortest path

Lemma 4. If the successor graph contains a directed cycle W, then W is a negative cycle.

Pf.

- If $successor(v) = w$, we must have $d(v) \geq d(w) + c_{vw}$.
  (LHS and RHS are equal when $successor(v)$ is set; $d(w)$ can only decrease; $d(v)$ decreases only when $successor(v)$ is reset)
- Let $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ be the nodes along the cycle W.
- Assume that (v_k, v_1) is the last edge added to the successor graph.
- Just prior to that:

$$\begin{array}{lcll}
d(v_1) & \geq & d(v_2) & + \; c(v_1, v_2) \\
d(v_2) & \geq & d(v_3) & + \; c(v_2, v_3) \\
\vdots & \vdots & & \vdots \\
d(v_{k-1}) & \geq & d(v_k) & + \; c(v_{k-1}, v_k) \\
d(v_k) & > & d(v_1) & + \; c(v_k, v_1)
\end{array}$$

holds with strict inequality since we are updating d(v_k)

- Adding inequalities yields $\underbrace{c(v_1, v_2) + c(v_2, v_3) + \ldots + c(v_{k-1}, v_k) + c(v_k, v_1) < 0}_{\text{W is a negative cycle}}$. ▪

Bellman-Ford: finding the shortest path

Theorem 3. Given a digraph with no negative cycles, Bellman-Ford finds the cheapest $s \leadsto t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf.

- The successor graph cannot have a negative cycle. [Lemma 4]
- Thus, following the successor pointers from s yields a directed path to t.
- Let $s = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ be the nodes along this path P.
- Upon termination, if $successor(v) = w$, we must have $d(v) = d(w) + c_{vw}$. (LHS and RHS are equal when $successor(v)$ is set; $d(\cdot)$ did not change)

since algorithm terminated

- Thus,

$$\begin{aligned} d(v_1) &= d(v_2) + c(v_1, v_2) \\ d(v_2) &= d(v_3) + c(v_2, v_3) \\ &\vdots \\ d(v_{k-1}) &= d(v_k) + c(v_{k-1}, v_k) \end{aligned}$$

Adding equations yields $d(s) = d(t) + \underbrace{c(v_1, v_2) + c(v_2, v_3) + \ldots + c(v_{k-1}, v_k)}_{\text{cost of path P}}$. ▪

min cost of any s⤳t path (Theorem 2)

$\boxed{0}$

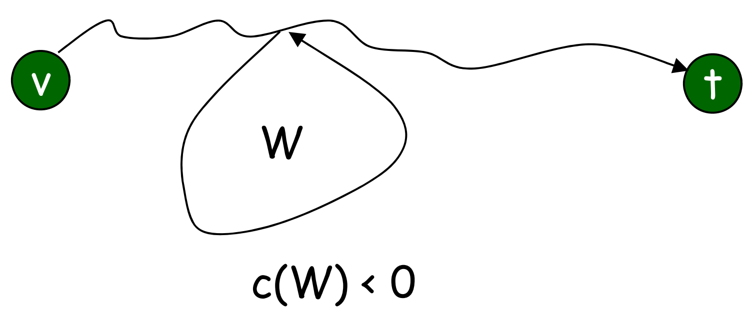
Detecting Negative Cycles

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) cheapest 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.



Detecting Negative Cycles

Theorem. Can detect any 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