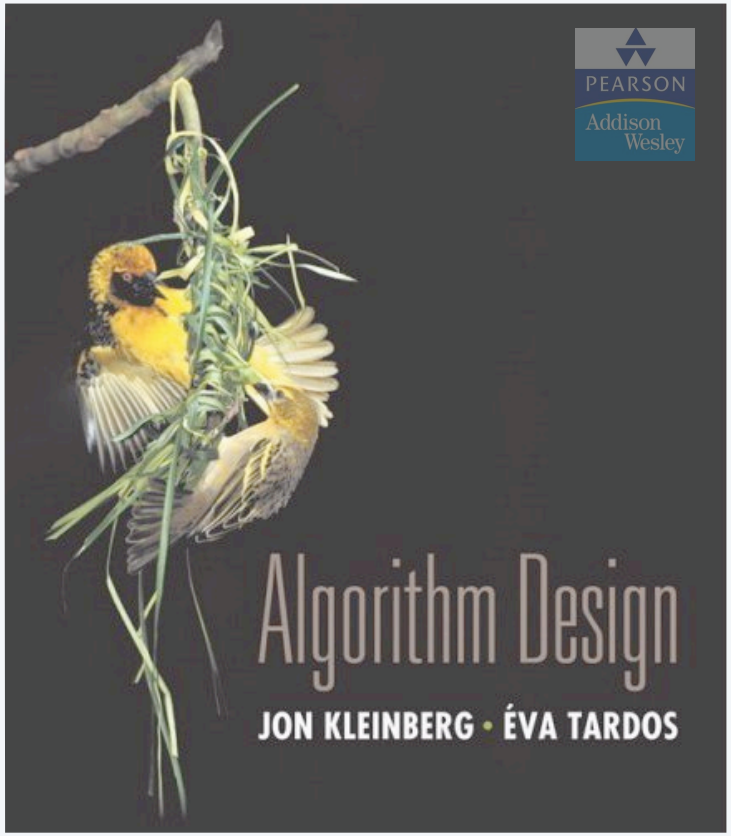


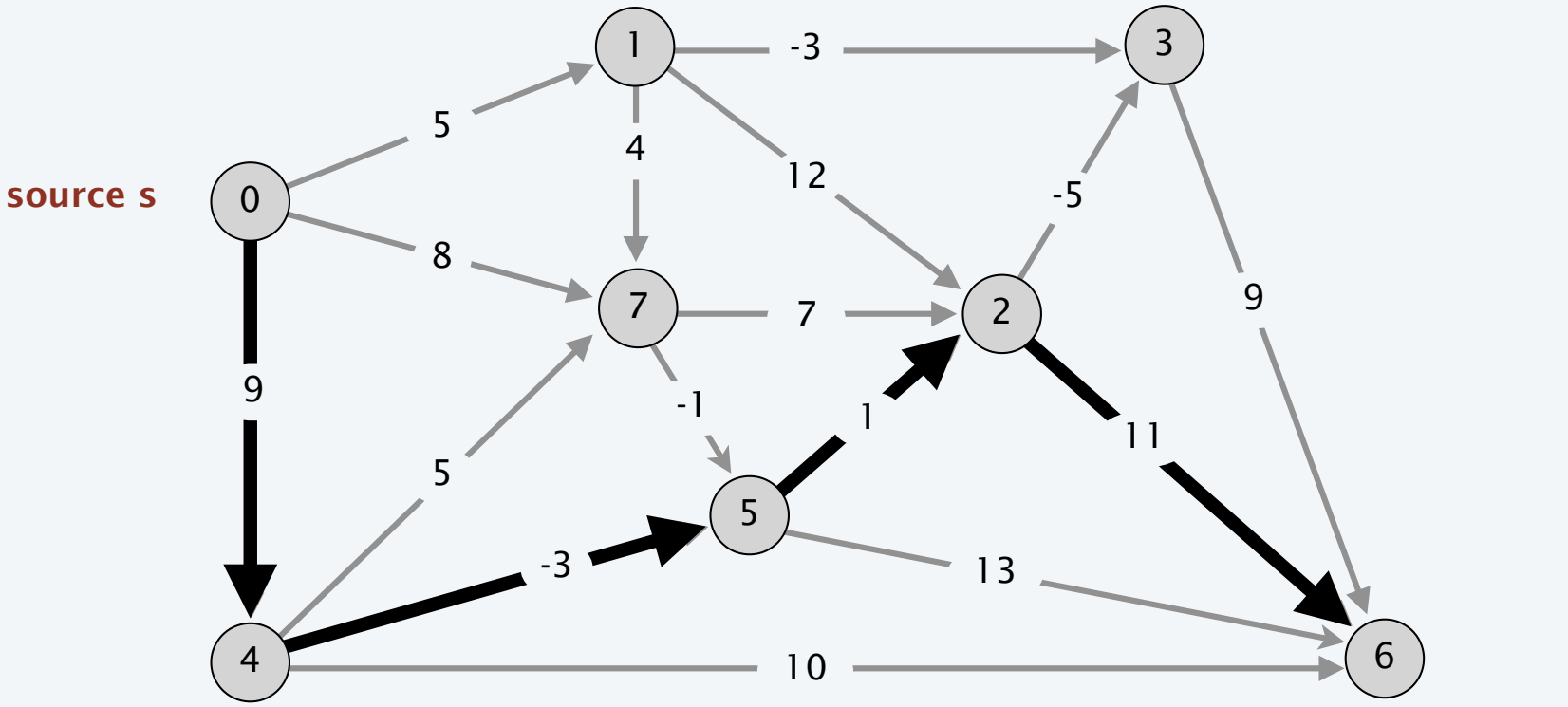
SECTION 6.8

6. DYNAMIC PROGRAMMING II

- *Bellman-Ford*

Shortest paths

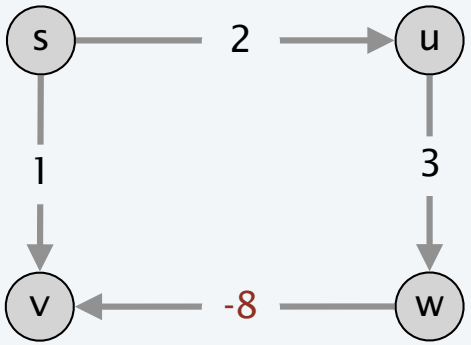
Shortest path problem. Given a digraph $G = (V, E)$, with arbitrary edge weights or costs c_{vw}, find cheapest path from node s to node t.



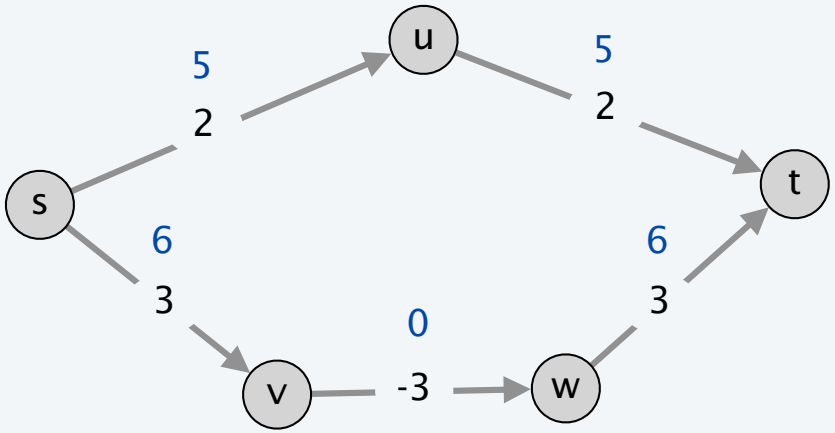
cost of path = 9 − 3 + 1 + 11 = 18

Shortest paths: failed attempts

Dijkstra. Can fail if negative edge weights.

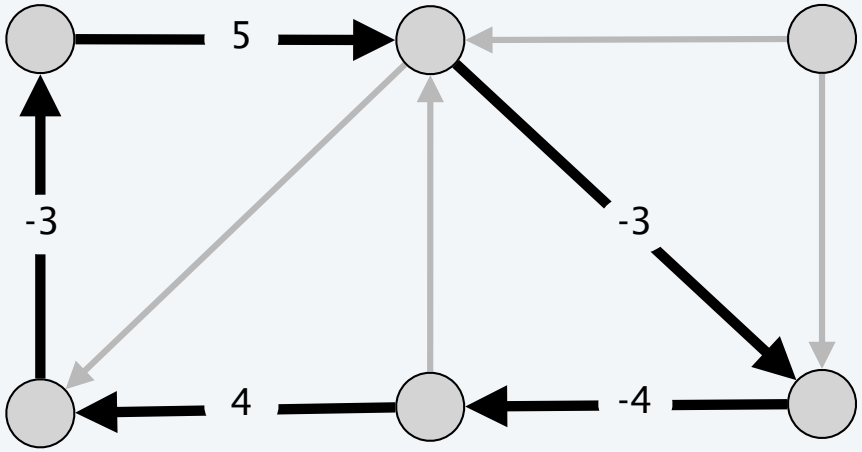


Reweighting. Adding a constant to every edge weight can fail.



Negative cycles

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

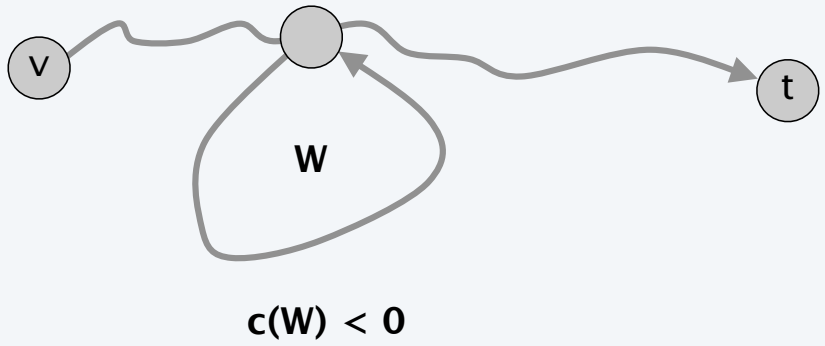


a negative cycle W : $c(W) = \sum_{e \in W} c_e < 0$

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 can build a $v \rightarrow t$ path of arbitrarily negative weight by detouring around cycle as many times as desired. ▪

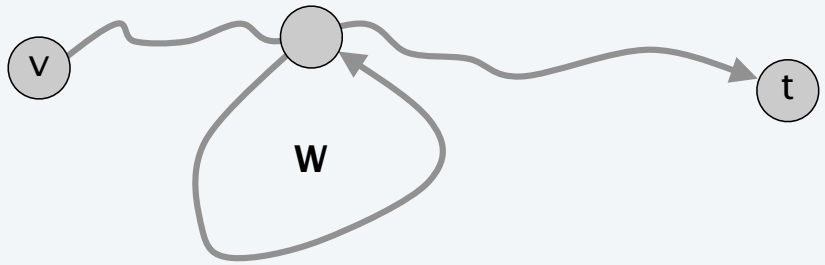


Shortest paths and negative cycles

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

Pf.

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

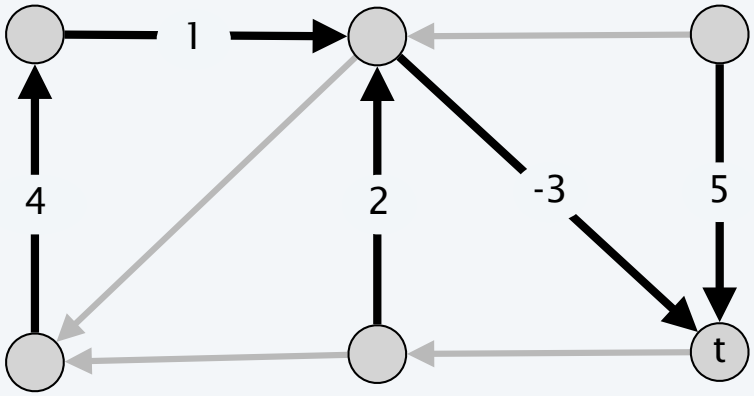


c(W) ≥ 0

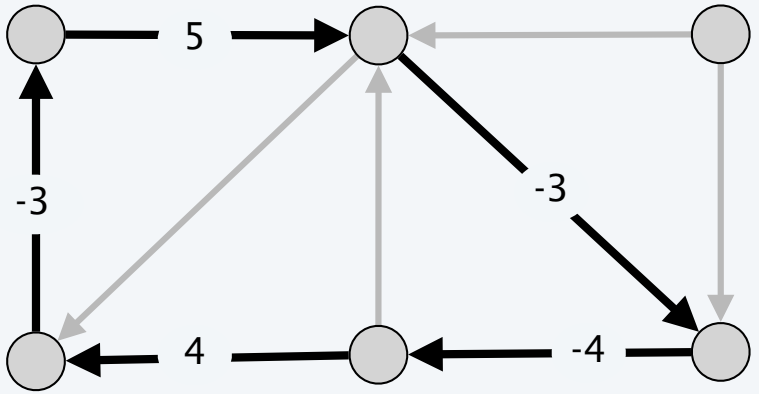
Shortest path and negative cycle problems

Shortest path problem. Given a digraph $G = (V, E)$ with edge weights c_{vw} and no negative cycles, find cheapest $v \rightarrow t$ 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 shortest $v \to t$ path that uses $\le i$ edges.

- Case 1: Cheapest $v \to t$ path uses $\le i - 1$ edges.
  - $OPT(i, v) = \mathrm{OPT}(i - 1, v)$

optimal substructure property
(proof via exchange argument)

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

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

Observation. If no negative cycles, $OPT(n - 1, v)$ = cost of cheapest $v \to t$ path.

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

Shortest paths: implementation

SHORTEST-PATHS (V, E, c, t)

FOREACH node $v \in V$

  $M[0, v] \leftarrow \infty$.

$M[0, t] \leftarrow 0$.

FOR i = 1 TO $n - 1$

  FOREACH node $v \in V$

    $M[i, v] \leftarrow M[i-1, v]$.

    FOREACH edge $(v, w) \in E$

      $M[i, v] \leftarrow \min \{ M[i, v],\ M[i-1, w] + c_{vw} \}$.

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 $v \rightarrow t$ 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. ▪

Finding the shortest paths.

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

Shortest paths: practical improvements

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

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

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

Bellman-Ford: efficient implementation

BELLMAN-FORD (V, E, c, t)

FOREACH node $v \in V$

  $d(v) \leftarrow \infty$.

  $successor(v) \leftarrow null$.

$d(t) \leftarrow 0$.

FOR i = 1 TO $n - 1$

  FOREACH node $w \in V$

    IF ($d(w)$ was updated in previous iteration)

      FOREACH edge $(v, w) \in E$

        IF ($d(v) > d(w) + c_{vw}$)

          $d(v) \leftarrow d(w) + c_{vw}$.

          $successor(v) \leftarrow w$.

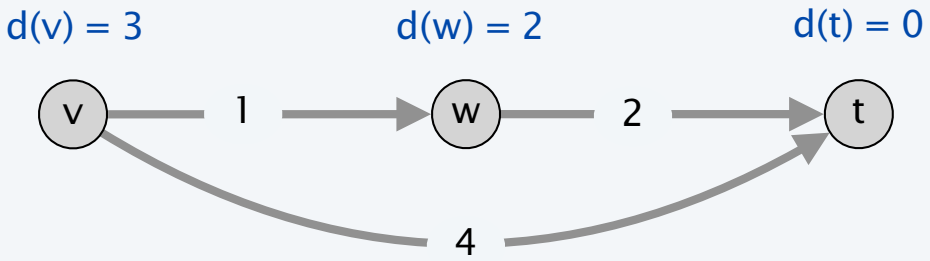
  IF no $d(w)$ value changed in iteration i, STOP.

1 pass

Bellman-Ford: analysis

Claim. After ~~the i^{th} pass of Bellman-Ford, $d(v)$ equals the cost of the cheapest $v \rightarrow t$ path using at most i edges~~.

Counterexample. Claim is false!



if nodes w considered before node v,
then d(v) = 3 after 1 pass

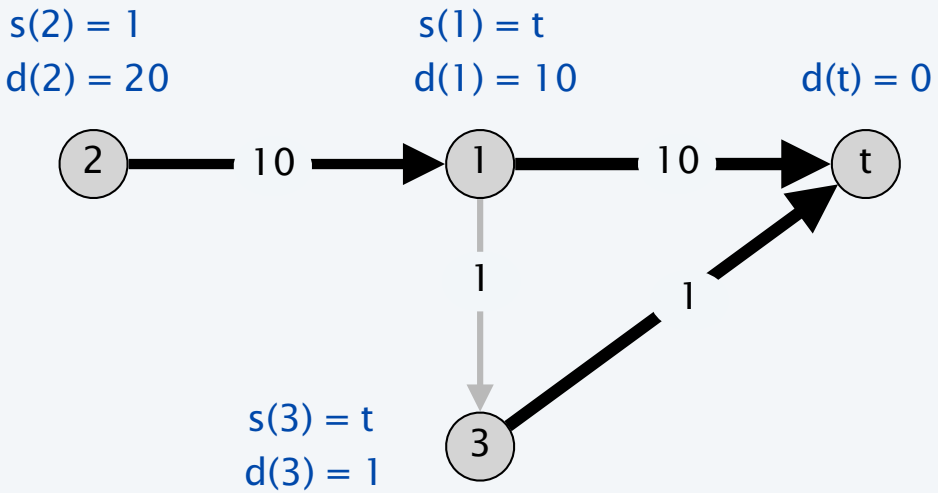
Bellman-Ford: analysis

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 $d(v)$.

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



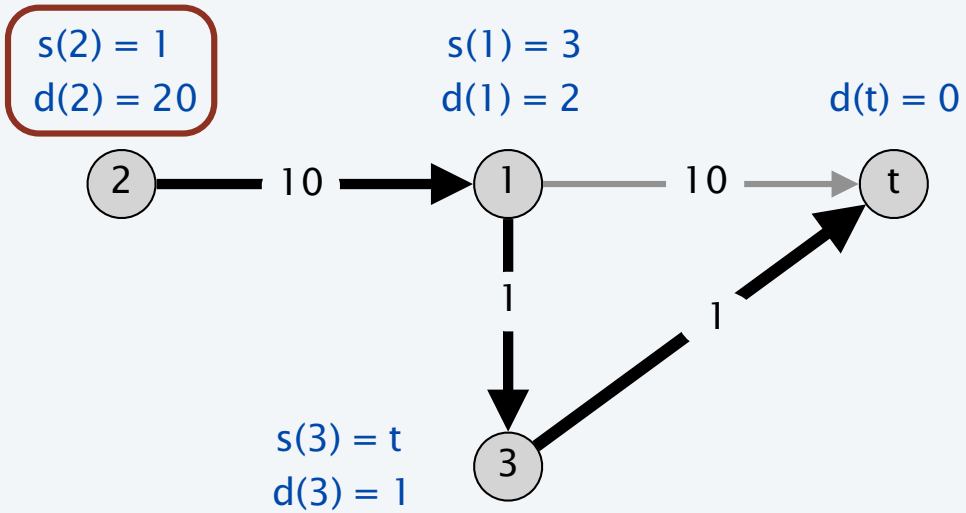
Bellman-Ford: analysis

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 $v \rightarrow t$ path may have strictly lower cost than $d(v)$.

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



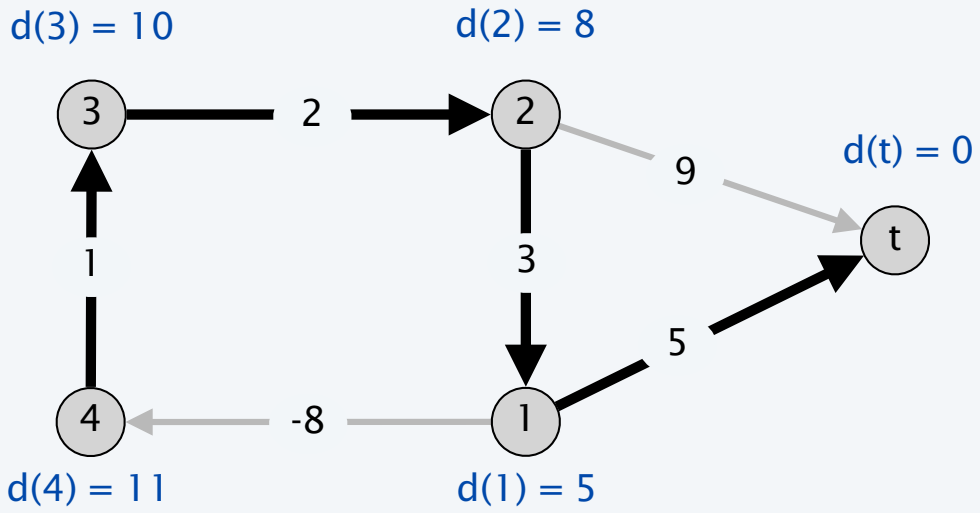
Bellman-Ford: analysis

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

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