Learning Goals

- Minimum-cost Path problem with general edge costs
- Bellman-Ford algorithm
- Running time of Bellman-Ford algorithm
- Polynomial-time algorithm to detect a negative cycle

Finding minimum-cost paths in a graph

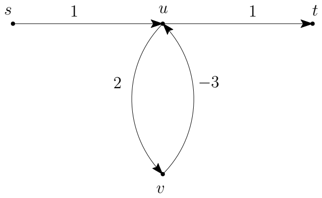
- Input: a directed graph $G = (V, E)$, with cost $c_e \in \mathbb{R}$ for each edge $e \in E$. A node $s \in V$.
- Output: If a negative cycle exists, report so. If not, for each node $v \in V$, a minimum-cost path from s to v, and its cost.

Finding minimum-cost paths in a graph

- Input: a directed graph $G = (V, E)$, with cost $c_e \in \mathbb{R}$ for each edge $e \in E$. A node $s \in V$.
- Output: If a negative cycle exists, report so. If not, for each node $v \in V$, a minimum-cost path from s to v, and its cost.
- The two cases (with negative cycle or not) needs to be separate
  - With a negative cycle, minimum-cost paths are generally not defined.

Finding minimum-cost paths in a graph

- Input: a directed graph $G = (V, E)$, with cost $c_e \in \mathbb{R}$ for each edge $e \in E$. A node $s \in V$.
- Output: If a negative cycle exists, report so. If not, for each node $v \in V$, a minimum-cost path from s to v, and its cost.
- The two cases (with negative cycle or not) needs to be separate
  - With a negative cycle, minimum-cost paths are generally not defined.
  - A path can go around such a cycle indefinitely to reduce its cost!

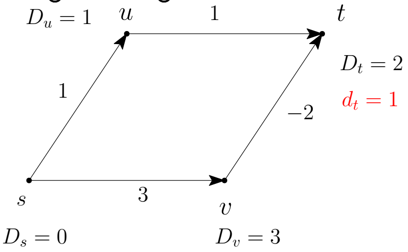


Why Dijkstra fails

- Even without negative cycles, Dijkstra's algorithm may fail when there are negative edge costs.

Why Dijkstra fails

- Even without negative cycles, Dijkstra's algorithm may fail when there are negative edge costs.

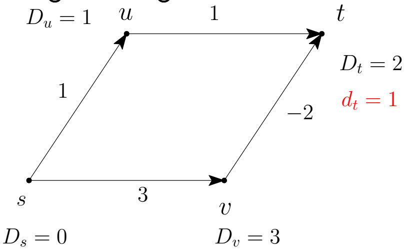


Example of Dijkstra failure

- D: Dijkstra results
- d_t is correct answer

Why Dijkstra fails

- Even without negative cycles, Dijkstra's algorithm may fail when there are negative edge costs.

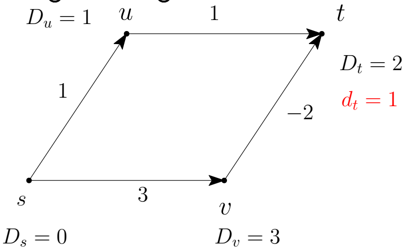


Example of Dijkstra failure

- D: Dijkstra results
- d_t is correct answer
- $D_t = 2$ is incorrect

Why Dijkstra fails

- Even without negative cycles, Dijkstra's algorithm may fail when there are negative edge costs.

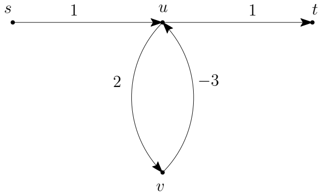


Example of Dijkstra failure

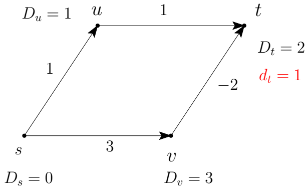
- D: Dijkstra results
- d_t is correct answer
- $D_t = 2$ is incorrect
- Lesson: Nodes "discovered" later may lead to better paths to nodes "discovered" earlier

Review of Last Lecture

Min-cost paths are not well defined when there is a negative cycle.



Dijkstra can fail when there are negative edge costs.



Bellman-Ford Algorithm

- The example suggests: Keep updating everything, always

Bellman-Ford Algorithm

- The example suggests: Keep updating everything, always
- Bellman-Ford Algorithm, when the graph is guaranteed to have no negative cycle:
  - Initialize: $d_0(s) = 0$, $d_0(v) = \infty$ for all $v \neq s$.

Bellman-Ford Algorithm

- The example suggests: Keep updating everything, always
- Bellman-Ford Algorithm, when the graph is guaranteed to have no negative cycle:
  - Initialize: $d_0(s) = 0$, $d_0(v) = \infty$ for all $v \neq s$.
  - Iterate: initialize $i \leftarrow 1$.
    - for each $v \in V$, $d_i(v) \leftarrow \min\{d_{i-1}(v), \min_{(u,v) \in E} d_{i-1}(u) + c_{(u,v)}\}$.
    - If $d_i(v) = d_{i-1}(v)$ for all v, terminate.
    - $i \leftarrow i + 1$.

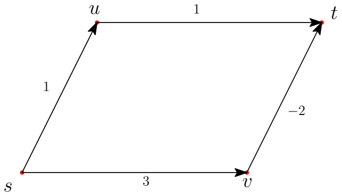
Bellman-Ford Algorithm

- The example suggests: Keep updating everything, always
- Bellman-Ford Algorithm, when the graph is guaranteed to have no negative cycle:
  - Initialize: $d_0(s) = 0$, $d_0(v) = \infty$ for all $v \neq s$.
  - Iterate: initialize $i \leftarrow 1$.
    - for each $v \in V$, $d_i(v) \leftarrow \min\{d_{i-1}(v), \min_{(u,v)\in E} d_{i-1}(u) + c_{(u,v)}\}$.
    - If $d_i(v) = d_{i-1}(v)$ for all v, terminate.
    - $i \leftarrow i + 1$.
- Output $d_i(v)$ for each $v \in V$.

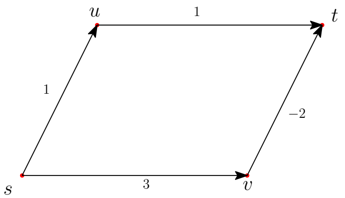
Bellman-Ford Algorithm

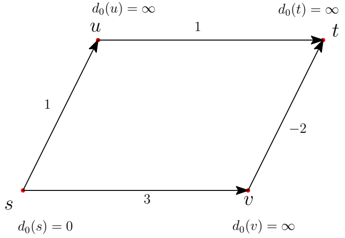
- The example suggests: Keep updating everything, always
- Bellman-Ford Algorithm, when the graph is guaranteed to have no negative cycle:
  - Initialize: $d_0(s) = 0$, $d_0(v) = \infty$ for all $v \neq s$.
  - Iterate: initialize $i \leftarrow 1$.
    - for each $v \in V$, $d_i(v) \leftarrow \min\{d_{i-1}(v), \min_{(u,v)\in E} d_{i-1}(u) + c_{(u,v)}\}$.
    - If $d_i(v) = d_{i-1}(v)$ for all v, terminate.
    - $i \leftarrow i + 1$.
- Output $d_i(v)$ for each $v \in V$.
- Obvious question: Does the algorithm terminate at all?

Bellman-Ford example

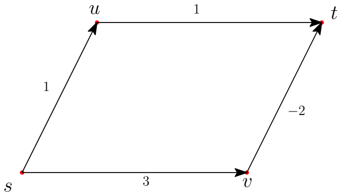


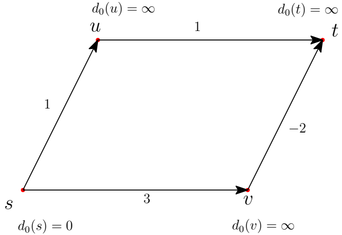
Bellman-Ford example

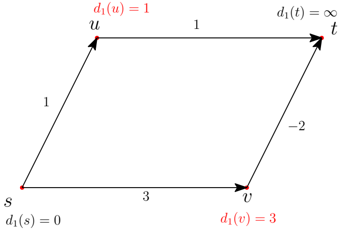




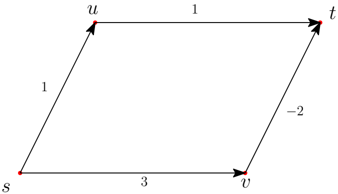
Bellman-Ford example

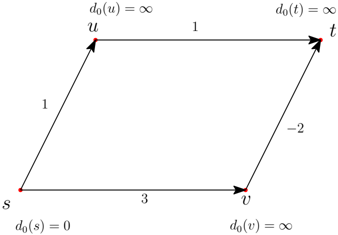


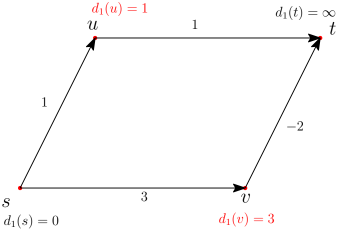


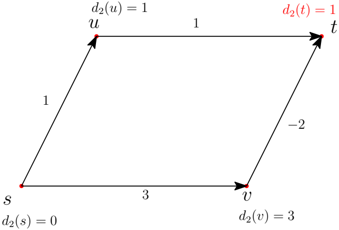


Bellman-Ford example









Proof of Correctness and Termination

The correctness of the algorithm is a consequence of two lemmas.

Lemma

For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is the minimum cost among all paths from s to v using at most i edges.

Proof of Correctness and Termination

The correctness of the algorithm is a consequence of two lemmas.

Lemma

For each $v \in V$, after the i-th iteration, $d_i(v)$ is the minimum cost among all paths from s to v using at most i edges.

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path with at most $n - 1$ edges.

Proof of Correctness and Termination

The correctness of the algorithm is a consequence of two lemmas.

Lemma

For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is the minimum cost among all paths from s to v using at most i edges.

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path with at most $n-1$ edges.

When the graph is guaranteed to have no negative cycle, the algorithm terminates after at most n iterations, and $d_i(v)$ contains the correct answer for each $v \in V$.

Proof of Correctness and Termination

The correctness of the algorithm is a consequence of two lemmas.

Lemma

For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is the minimum cost among all paths from s to v using at most i edges.

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path with at most $n-1$ edges.

When the graph is guaranteed to have no negative cycle, the algorithm terminates after at most n iterations, and $d_i(v)$ contains the correct answer for each $v \in V$.

Running time: $O(m)$ per iteration, $O(n)$ iterations, so $O(mn)$ in total.

Proof of First Lemma

Lemma

For each $v \in V$, after the $i^{\text{-}th}$ iteration, $d_i(v)$ is minimum cost among all paths from s to v using at most i edges.

Proof of First Lemma

Lemma

For each $v \in V$, after the $i^{\text{-}th}$ iteration, $d_i(v)$ is minimum cost among all paths from s to v using at most i edges.

Proof.

By induction. Induction hypothesis is the lemma statement. Base case: $i = 0$, obvious. Inductive step:

Proof of First Lemma

Lemma

For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is minimum cost among all paths from s to v using at most i edges.

Proof.

By induction. Induction hypothesis is the lemma statement. Base case: $i = 0$, obvious. Inductive step:

- For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is the cost of *some* path from s to v with at most i edges.

Proof of First Lemma

Lemma

For each $v \in V$, after the i-th iteration, $d_i(v)$ is minimum cost among all paths from s to v using at most i edges.

Proof.

By induction. Induction hypothesis is the lemma statement. Base case: $i = 0$, obvious. Inductive step:

- For each $v \in V$, after the i-th iteration, $d_i(v)$ is the cost of *some* path from s to v with at most i edges.
- Any path P from s to v with at most i edges either
  - reaches v with $\leq i - 1$ edges, with cost $\geq d_{i-1}(v)$, by IH;

Proof of First Lemma

Lemma

For each $v \in V$, after the i-th iteration, $d_i(v)$ is minimum cost among all paths from s to v using at most i edges.

Proof.

By induction. Induction hypothesis is the lemma statement. Base case: $i = 0$, obvious. Inductive step:

- For each $v \in V$, after the i-th iteration, $d_i(v)$ is the cost of *some* path from s to v with at most i edges.
- Any path P from s to v with at most i edges either
  - reaches v with $\leq i - 1$ edges, with cost $\geq d_{i-1}(v)$, by IH;
  - or first reaches some node u using at most $i - 1$ edges, and then takes edge (u, v), with cost $\geq d_{i-1}(u) + c_{(u,v)}$.

Proof of First Lemma

Lemma

For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is minimum cost among all paths from s to v using at most i edges.

Proof.

By induction. Induction hypothesis is the lemma statement. Base case: $i = 0$, obvious. Inductive step:

- For each $v \in V$, after the $i^{\text{-th}}$ iteration, $d_i(v)$ is the cost of *some* path from s to v with at most i edges.
- Any path P from s to v with at most i edges either
  - reaches v with $\leq i - 1$ edges, with cost $\geq d_{i-1}(v)$, by IH;
  - or first reaches some node u using at most $i - 1$ edges, and then takes edge (u, v), with cost $\geq d_{i-1}(u) + c_{(u,v)}$.
- P has cost at least $\min\{d_{i-1}(v), \min_{(u,v)\in E} d_{i-1}(u) + c_{(u,v)}\} = d_i(v)$.

$\square$

Proof of Second Lemma

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path from s to t with at most $n-1$ edges.

Proof of Second Lemma

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path from s to t with at most $n-1$ edges.

Proof.

Take any path P from s to t. If P passes a node v twice, the part of the path between these is a cycle.

Proof of Second Lemma

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path from s to t with at most $n - 1$ edges.

Proof.

Take any path P from s to t. If P passes a node v twice, the part of the path between these is a cycle. "Skipping" the cycle does not increase the cost of P because the cycle has nonnegative cost.

Proof of Second Lemma

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path from s to t with at most $n-1$ edges.

Proof.

Take any path P from s to t. If P passes a node v twice, the part of the path between these is a cycle. “Skipping” the cycle does not increase the cost of P because the cycle has nonnegative cost. Keep removing cycles this way until P is simple (i.e., passes each node at most once); its cost never increased.

Proof of Second Lemma

Lemma

In a graph containing no negative cycles, if there is a path from node s to node t, then there is a minimum-cost path from s to t with at most $n - 1$ edges.

Proof.

Take any path P from s to t. If P passes a node v twice, the part of the path between these is a cycle. “Skipping” the cycle does not increase the cost of P because the cycle has nonnegative cost. Keep removing cycles this way until P is simple (i.e., passes each node at most once); its cost never increased. A simple path has at most $n - 1$ edges. □

What if there is negative cycle?

Modified Bellman-Ford Algorithm, without guarantee of no negative cycle:

- Initialize: $d_0(s) = 0, d_0(v) = \infty$ for $v \neq s$.

What if there is negative cycle?

Modified Bellman-Ford Algorithm, without guarantee of no negative cycle:

- Initialize: $d_0(s) = 0, d_0(v) = \infty$ for $v \neq s$.
- Iterate: initialize $i \leftarrow 1$.
  - for each $v \in V$, $d_i(v) \leftarrow \min\{d_{i-1}(v), \min_{(u,v)\in E} d_{i-1}(u) + c_{(u,v)}\}$.
  - If $d_i(v) = d_{i-1}(v)$ for all v, terminate and output the $d_i(v)$'s.
  - $i \leftarrow i + 1$.
  - If $i > n$, terminate, report there is a negative cycle.

Correctness of Modified Bellman-Ford Algorithm

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Correctness of Modified Bellman-Ford Algorithm

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Rightarrow$): If the algorithm does not terminate after n iterations, there must be a negative cycle, because

- For some $v \in V$, $d_n(v) < d_{n-1}(v)$;

Correctness of Modified Bellman-Ford Algorithm

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Rightarrow$): If the algorithm does not terminate after n iterations, there must be a negative cycle, because

- For some $v \in V$, $d_n(v) < d_{n-1}(v)$;
- By First Lemma, $d_{n-1}(v)$ is the cost of min-cost path to v with at most $n-1$ edges.

Correctness of Modified Bellman-Ford Algorithm

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Rightarrow$): If the algorithm does not terminate after n iterations, there must be a negative cycle, because

- For some $v \in V$, $d_n(v) < d_{n-1}(v)$;
- By First Lemma, $d_{n-1}(v)$ is the cost of min-cost path to v with at most $n-1$ edges.
- By Second Lemma, if any path has strictly less cost than $d_{n-1}(v)$, there must be a negative cycle.

Correctness of Modified Bellman-Ford Algorithm Cont.

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Leftarrow$): If there is a negative cycle reachable from s, the algorithm cannot terminate before n iterations finish. Proof by contradiction:

Correctness of Modified Bellman-Ford Algorithm Cont.

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Leftarrow$): If there is a negative cycle reachable from s, the algorithm cannot terminate before n iterations finish. Proof by contradiction:

- Key observation: if in some iteration i, $d_i(v) = d_{i-1}(v)$ for all v, then $d_k(v) = d_i(v)$ for any $k > i$, including any $k \geq n$ (if we let the algorithm keep running).

Correctness of Modified Bellman-Ford Algorithm Cont.

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Leftarrow$): If there is a negative cycle reachable from s, the algorithm cannot terminate before n iterations finish. Proof by contradiction:

- Key observation: if in some iteration i, $d_i(v) = d_{i-1}(v)$ for all v, then $d_k(v) = d_i(v)$ for any $k > i$, including any $k \geq n$ (if we let the algorithm keep running).
- By First Lemma, no path to v (of any length) can have cost $< d_i(v)$.

Correctness of Modified Bellman-Ford Algorithm Cont.

Claim

The modified Bellman-Ford algorithm reports a negative cycle if and only if there is one reachable from s.

Proof.

($\Leftarrow$): If there is a negative cycle reachable from s, the algorithm cannot terminate before n iterations finish. Proof by contradiction:

- Key observation: if in some iteration i, $d_i(v) = d_{i-1}(v)$ for all v, then $d_k(v) = d_i(v)$ for any $k > i$, including any $k \geq n$ (if we let the algorithm keep running).
- By First Lemma, no path to v (of any length) can have cost $< d_i(v)$.
- But for any v on the reachable negative cycle, there must be a path to v with cost $< d_i(v)$, by going through the cycle for enough rounds. This is a contradiction.

$\square$

Space Consideration

- In Bellman-Ford, we seem to use an $n \times n$ array: an entry per node for at most n rounds.

Space Consideration

- In Bellman-Ford, we seem to use an $n \times n$ array: an entry per node for at most n rounds.
- This is unnecessary, because after the $i^{\text{-th}}$ iteration, we never use the contents of the array from iterations $1, \ldots, i-1$.

Space Consideration

- In Bellman-Ford, we seem to use an $n \times n$ array: an entry per node for at most n rounds.
- This is unnecessary, because after the i-th iteration, we never use the contents of the array from iterations $1, \ldots, i-1$.
- It suffices to use a $2 \times n$ array, using the first row to keep results from the last iteration, and the second for current iteration computation.

Space Consideration

- In Bellman-Ford, we seem to use an $n \times n$ array: an entry per node for at most n rounds.
- This is unnecessary, because after the i-th iteration, we never use the contents of the array from iterations $1, \ldots, i-1$.
- It suffices to use a $2 \times n$ array, using the first row to keep results from the last iteration, and the second for current iteration computation.
- In fact it suffices to use an array with only n entries, one per each node, and the update rule in each iteration is simply

$$d(v) \leftarrow \min\{d(v), \min_{(u,v)\in E} d(u) + c_{(u,v)}\}.$$

Exercise: Why is this OK?