CSE 431/531: Algorithm Analysis and Design (Fall 2024)
Graph Algorithms

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

Outline

Outline

Outline

Outline

Outline

Improved Running Time using Priority Queue

Dijkstra(G, w, s)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
$Q \leftarrow$ empty queue, for each $v \in V$: Q.insert$(v, d[v])$
while $S \neq V$ **do**
    $u \leftarrow Q$.extract_min()
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $d[u] + w(u, v) < d[v]$ **then**
            $d[v] \leftarrow d[u] + w(u, v)$, Q.decrease_key$(v, d[v])$
            $\pi[v] \leftarrow u$
return (π, d)

Recall: Prim's Algorithm for MST

MST-Prim(G, w)

$s \leftarrow$ arbitrary vertex in G
$S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
$Q \leftarrow$ empty queue, for each $v \in V$: Q.insert$(v, d[v])$
while $S \neq V$ **do**
    $u \leftarrow Q$.extract_min()
    $S \leftarrow S \cup \{u\}$
    for each $v \in V \setminus S$ such that $(u, v) \in E$ **do**
        if $w(u, v) < d[v]$ **then**
            $d[v] \leftarrow w(u, v)$, Q.decrease_key$(v, d[v])$
            $\pi[v] \leftarrow u$
return $\big\{(u, \pi[u]) | u \in V \setminus \{s\}\big\}$

Improved Running Time

Running time:
$O(n) \times$ (time for extract_min) $+ O(m) \times$ (time for decrease_key)

Priority-Queue	extract_min	decrease_key	Time
Heap	$O(\log n)$	$O(\log n)$	$O(m \log n)$
Fibonacci Heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

Outline

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

- In transition graphs, negative weights make sense

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

- In transition graphs, negative weights make sense
- If we sell a item: 'having the item' $\to$ 'not having the item', weight is negative (we gain money)

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

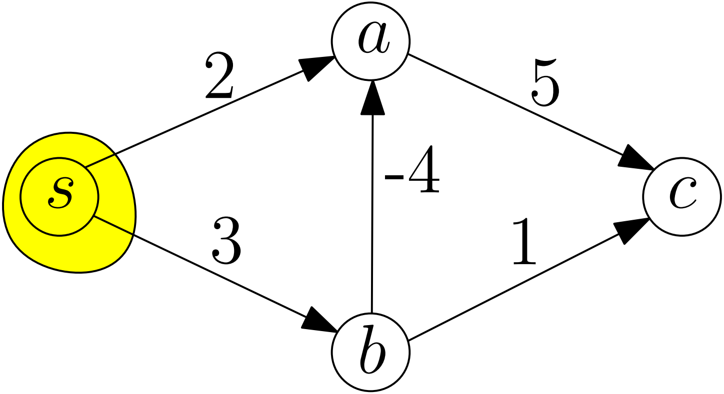
assume all vertices are reachable from s

$w : E \to \mathbb{R}$

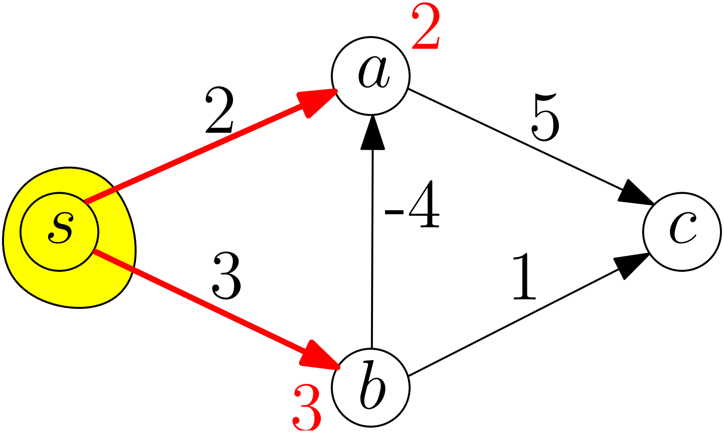
Output: shortest paths from s to all other vertices $v \in V$

- In transition graphs, negative weights make sense
- If we sell a item: 'having the item' $\to$ 'not having the item', weight is negative (we gain money)
- Dijkstra's algorithm does not work any more!

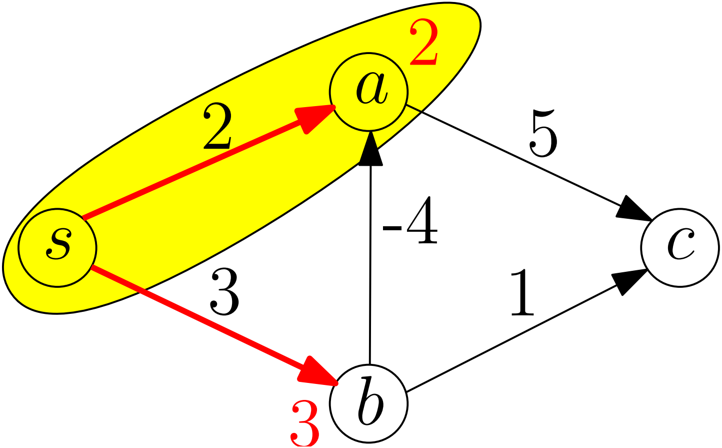
Dijkstra's Algorithm Fails if We Have Negative Weights



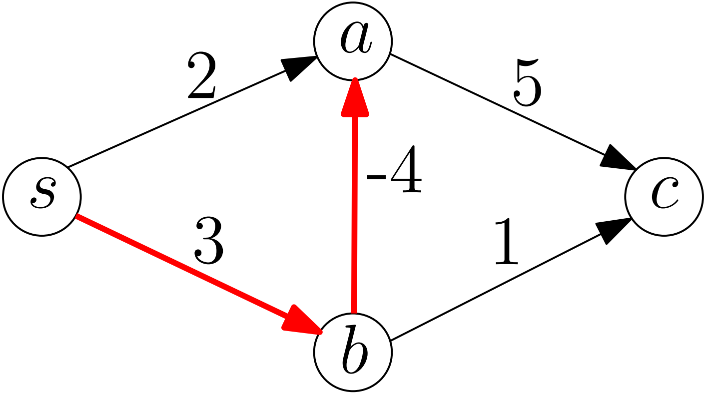
Dijkstra's Algorithm Fails if We Have Negative Weights

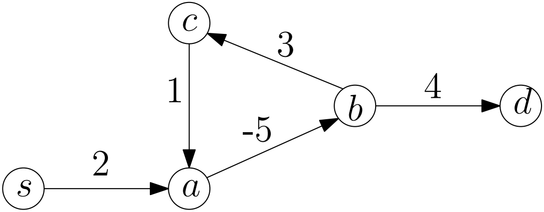


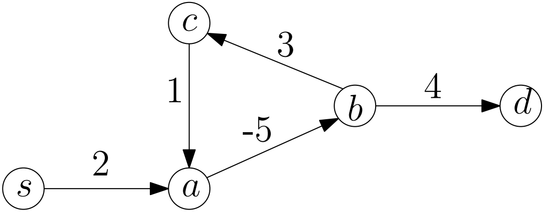
Dijkstra's Algorithm Fails if We Have Negative Weights



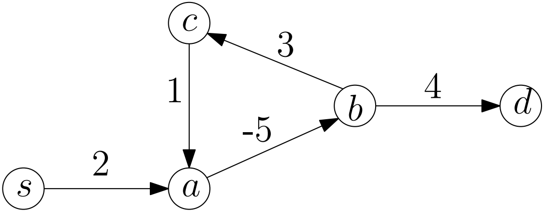
Dijkstra's Algorithm Fails if We Have Negative Weights




c
3
1
b
4
d
-5
2
s
a

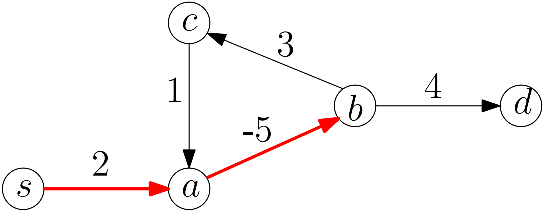


Q: What is the length of the shortest path from s to d?



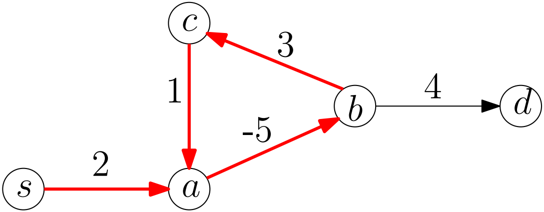
Q: What is the length of the shortest path from s to d?

A: $-\infty$



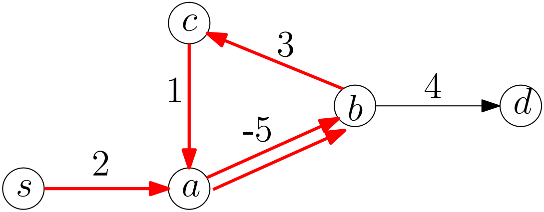
Q: What is the length of the shortest path from s to d?

A: $-\infty$



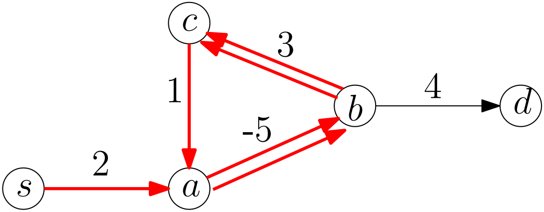
Q: What is the length of the shortest path from s to d?

A: $-\infty$



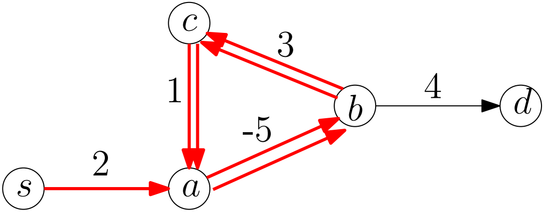
Q: What is the length of the shortest path from s to d?

A: $-\infty$



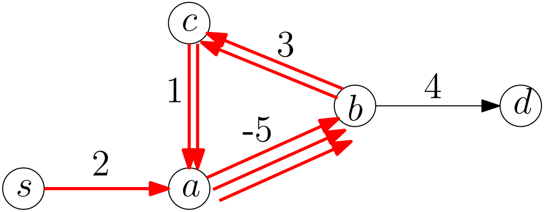
Q: What is the length of the shortest path from s to d?

A: $-\infty$



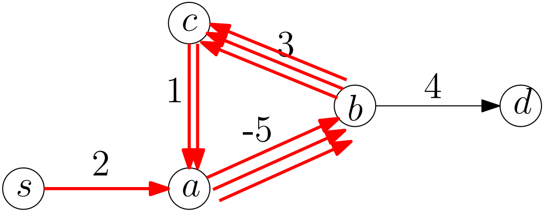
Q: What is the length of the shortest path from s to d?

A: $-\infty$



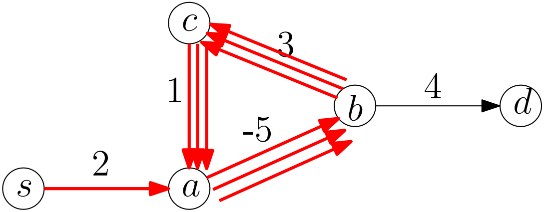
Q: What is the length of the shortest path from s to d?

A: $-\infty$



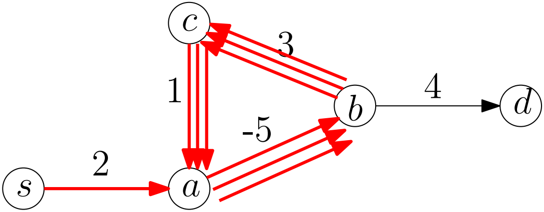
Q: What is the length of the shortest path from s to d?

A: $-\infty$



Q: What is the length of the shortest path from s to d?

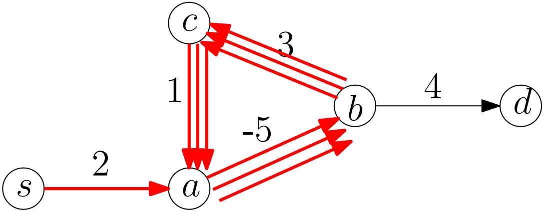
A: $-\infty$



Q: What is the length of the shortest path from s to d?

A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

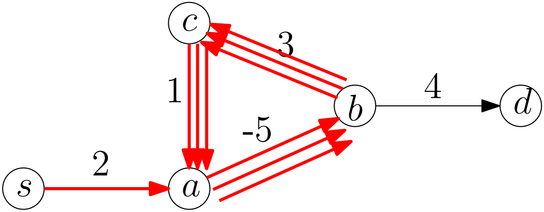


Q: What is the length of the shortest path from s to d?

A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

Dealing with Negative Cycles



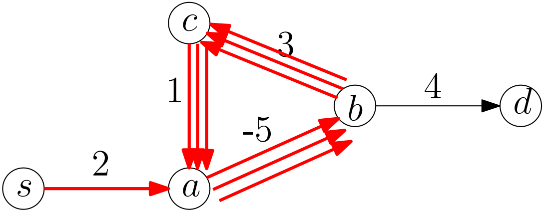
Q: What is the length of the shortest path from s to d?

A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

Dealing with Negative Cycles

- assume the input graph does not contain negative cycles, or



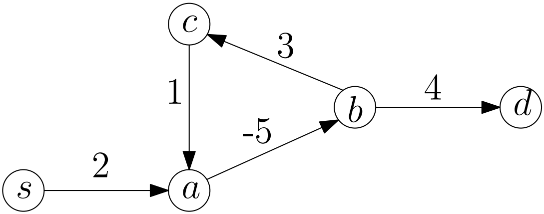
Q: What is the length of the shortest path from s to d?

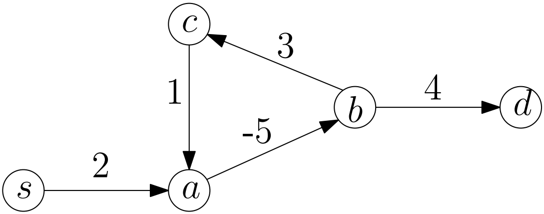
A: $-\infty$

Def. A negative cycle is a cycle in which the total weight of edges is negative.

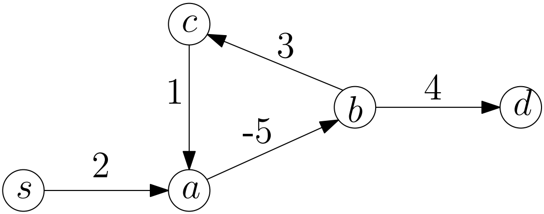
Dealing with Negative Cycles

- assume the input graph does not contain negative cycles, or
- allow algorithm to report "negative cycle exists"


c
3
1
b
4
d
-5
2
s
a

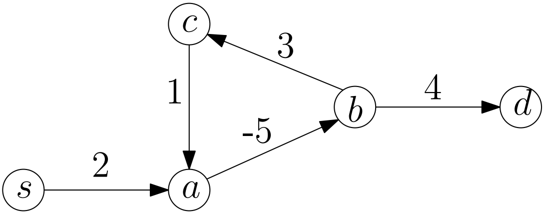


Q: What is the length of the shortest simple path from s to d?



Q: What is the length of the shortest simple path from s to d?

A: 1



Q: What is the length of the shortest simple path from s to d?

A: 1

- Unfortunately, computing the shortest simple path between two vertices is an NP-hard problem.

algorithm	graph	weights	SS?	running time
Simple DP	DAG	$\mathbb{R}$	SS	$O(n+m)$
Dijkstra	U/D	$\mathbb{R}_{\geq 0}$	SS	$O(n \log n + m)$
Bellman-Ford	U/D	$\mathbb{R}$	SS	$O(nm)$
Floyd-Warshall	U/D	$\mathbb{R}$	AP	$O(n^3)$

- DAG = directed acyclic graph U = undirected D = directed
- SS = single source AP = all pairs

Defining Cells of Table

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

Defining Cells of Table

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

- first try: $f[v]$: length of shortest path from s to v

Defining Cells of Table

Single Source Shortest Paths, Weights May be Negative

Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

- first try: $f[v]$: length of shortest path from s to v
- issue: do not know in which order we compute $f[v]$'s

Defining Cells of Table

Single Source Shortest Paths, Weights May be Negative

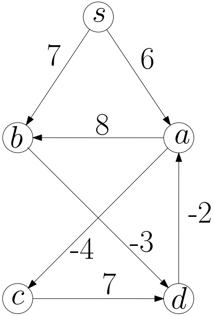
Input: directed graph $G = (V, E)$, $s \in V$

assume all vertices are reachable from s

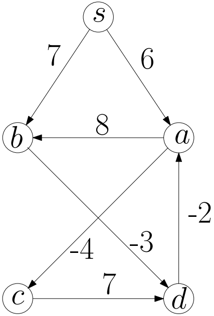
$w : E \to \mathbb{R}$

Output: shortest paths from s to all other vertices $v \in V$

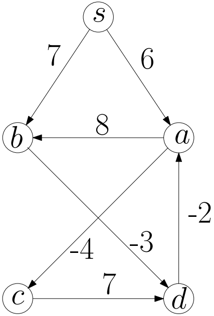
- first try: $f[v]$: length of shortest path from s to v
- issue: do not know in which order we compute $f[v]$'s
- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges



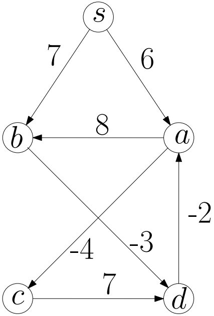
- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges



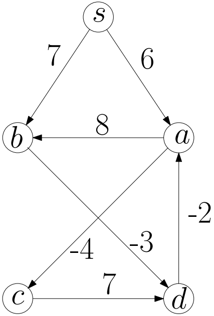
- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] =$



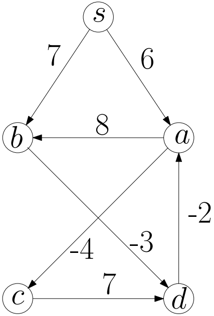
- $f^{\ell}[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$



- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] =$

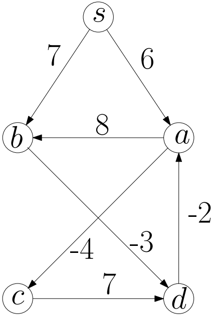


- $f^{\ell}[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$



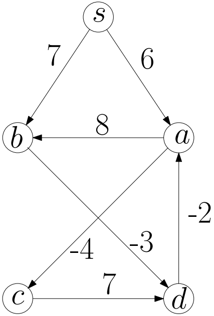
- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$

$$f^\ell[v] = \begin{cases} & \ell = 0, v = s \\ & \ell = 0, v \neq s \\ & \ell > 0 \end{cases}$$



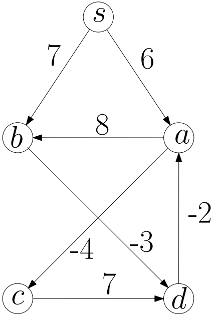
- $f^{\ell}[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$

$$f^{\ell}[v] = \begin{cases} 0 & \ell = 0, v = s \\ & \ell = 0, v \neq s \\ & \ell > 0 \end{cases}$$



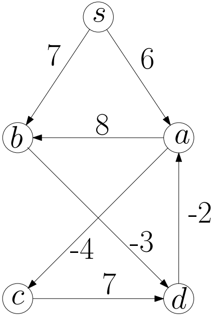
- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$

$$f^\ell[v] = \begin{cases} 0 & \ell = 0, v = s \\ \infty & \ell = 0, v \neq s \\ & \ell > 0 \end{cases}$$



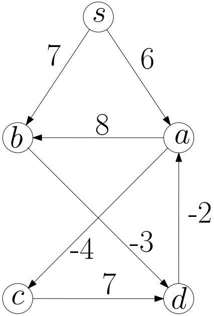
- $\color{red}{f^\ell[v]}$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$

$$f^\ell[v] = \begin{cases} 0 & \ell = 0, v = s \\ \infty & \ell = 0, v \neq s \\ \min \Big\{ & \ell > 0 \end{cases}$$



- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$

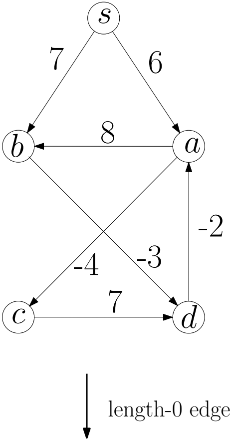
$$f^\ell[v] = \begin{cases} 0 & \ell = 0, v = s \\ \infty & \ell = 0, v \neq s \\ \min \left\{ \quad f^{\ell-1}[v] \right. & \ell > 0 \end{cases}$$



- $f^\ell[v]$, $\ell \in \{0, 1, 2, 3 \cdots, n-1\}$, $v \in V$: length of shortest path from s to v that uses at most ℓ edges
- $f^2[a] = 6$
- $f^3[a] = 2$

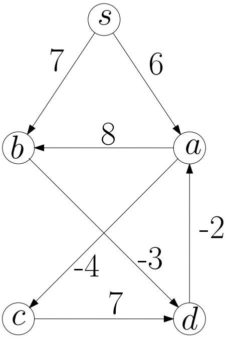
$$f^\ell[v] = \begin{cases} 0 & \ell = 0, v = s \\ \infty & \ell = 0, v \neq s \\ \min \begin{cases} f^{\ell-1}[v] \\ \min_{u:(u,v)\in E} \left(f^{\ell-1}[u] + w(u,v)\right) \end{cases} & \ell > 0 \end{cases}$$

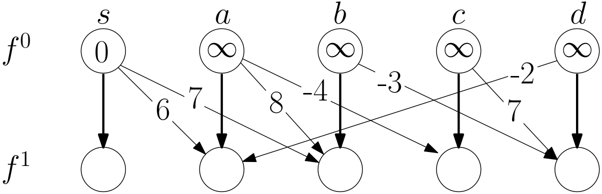
Dynamic Programming: Example



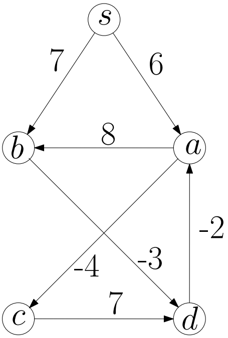
	s	a	b	c	d
f^0	0	∞	∞	∞	∞

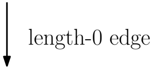
Dynamic Programming: Example

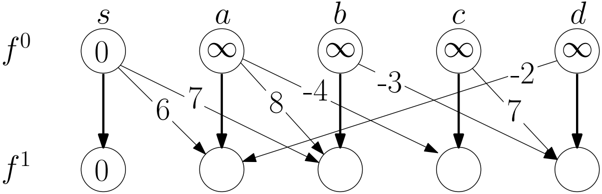




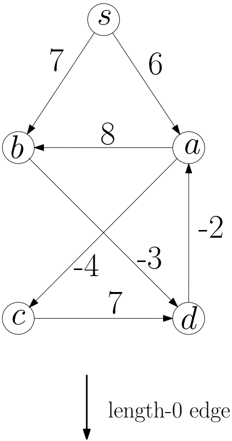
Dynamic Programming: Example

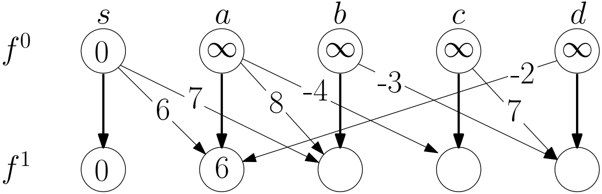




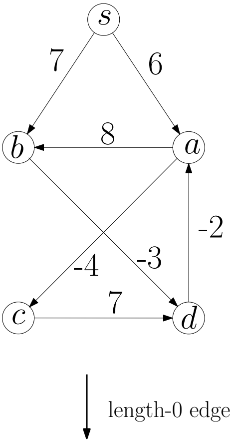


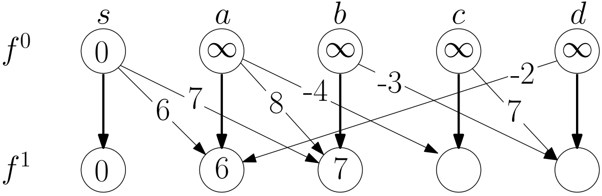
Dynamic Programming: Example



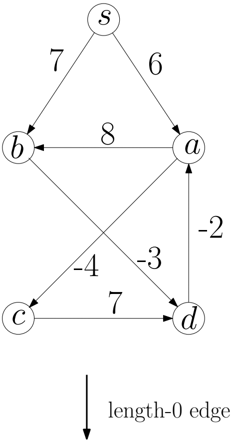


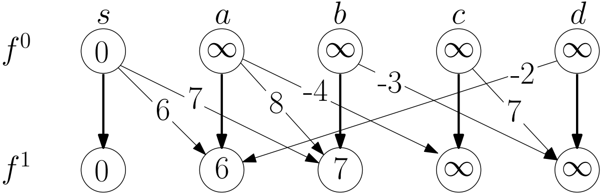
Dynamic Programming: Example



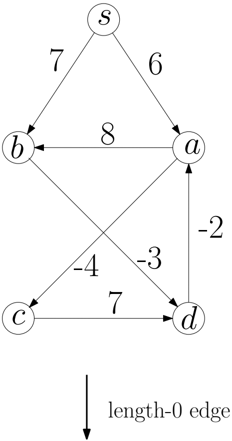


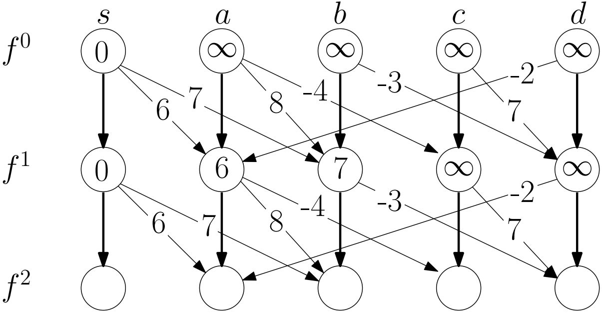
Dynamic Programming: Example



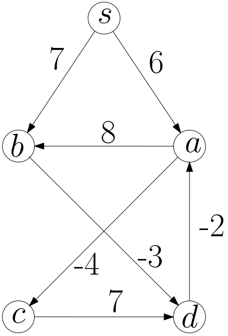


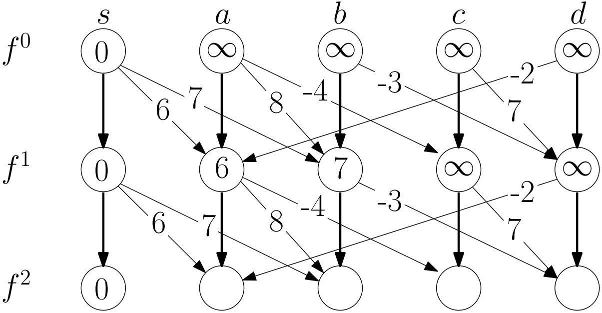
Dynamic Programming: Example





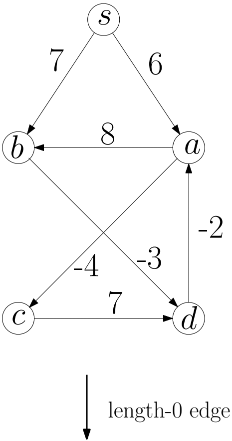
Dynamic Programming: Example

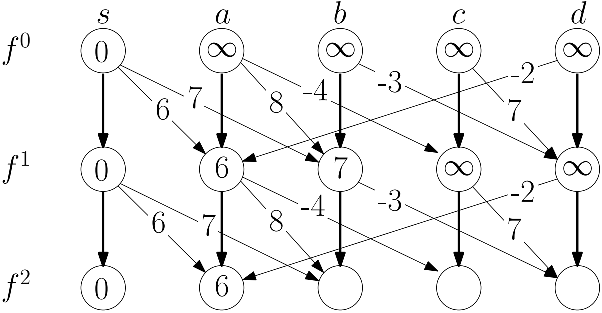




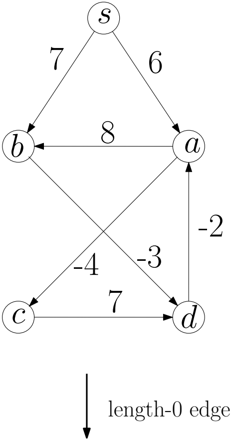
length-0 edge

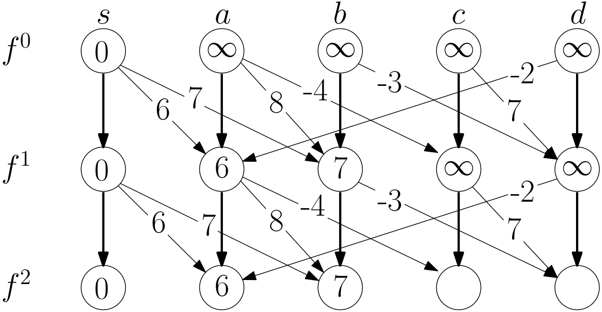
Dynamic Programming: Example



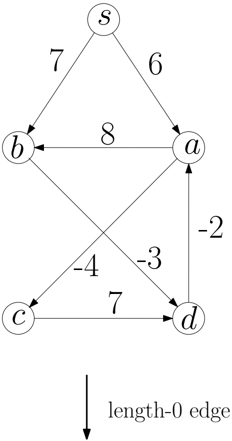


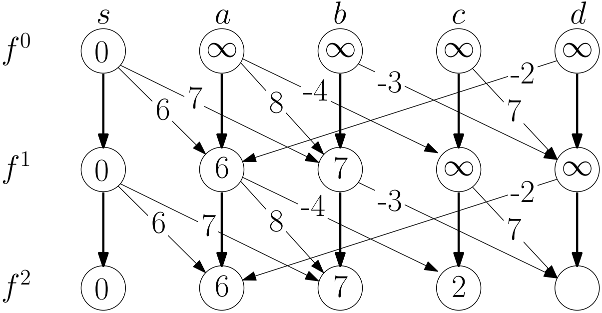
Dynamic Programming: Example



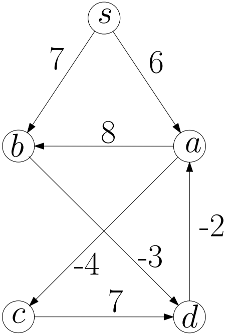


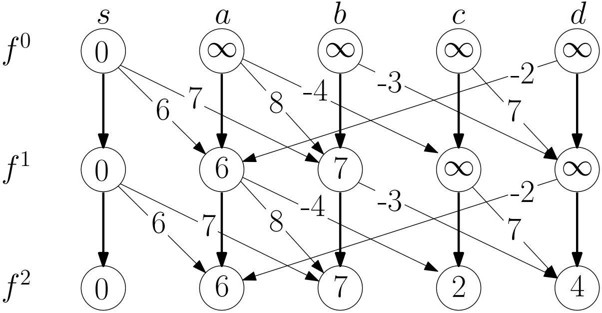
Dynamic Programming: Example





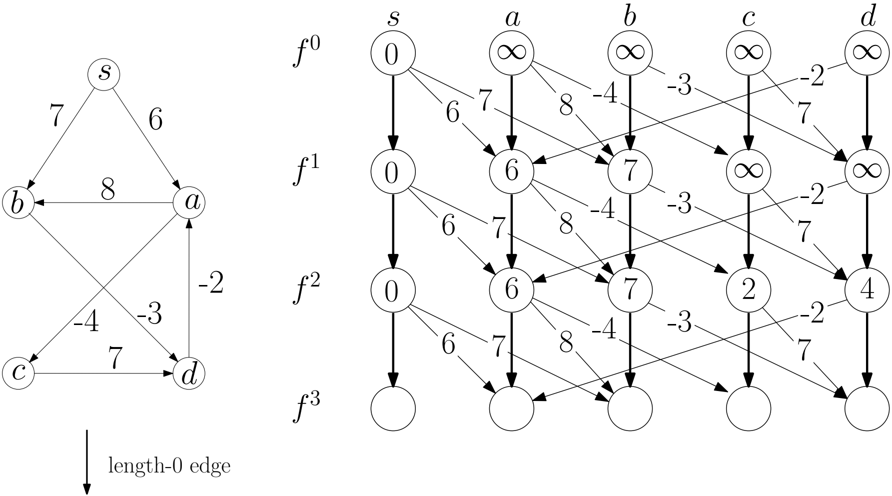
Dynamic Programming: Example



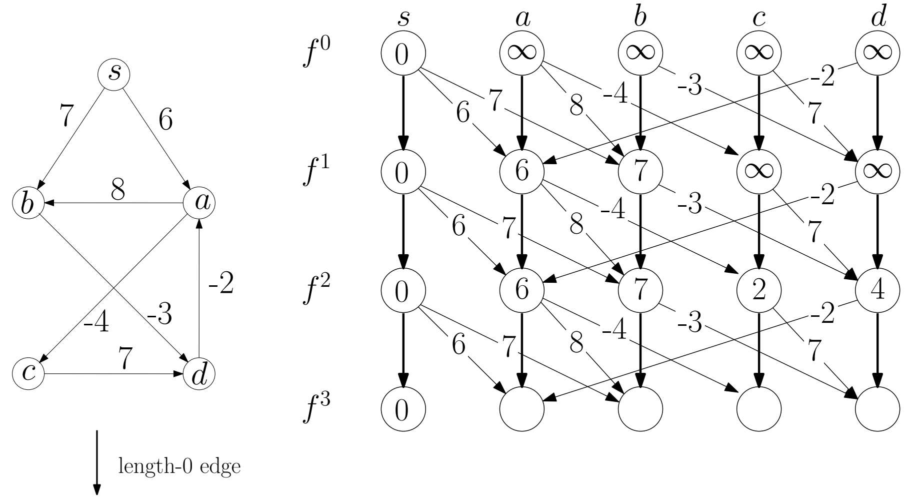


length-0 edge

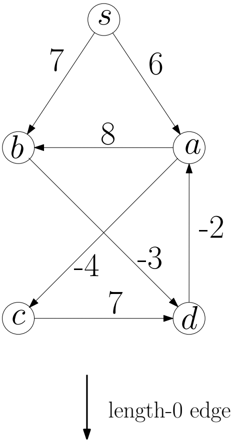
Dynamic Programming: Example

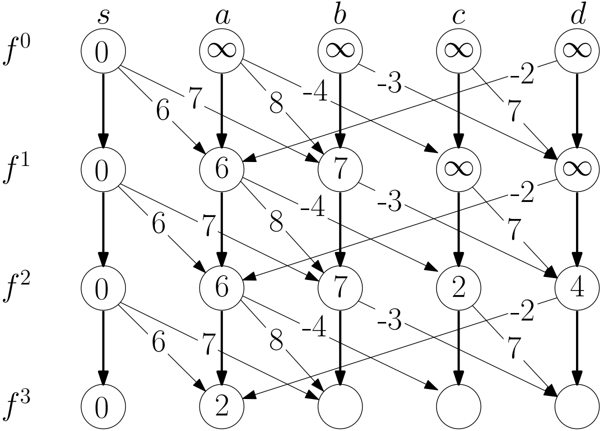


Dynamic Programming: Example

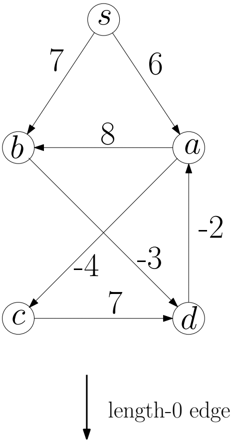


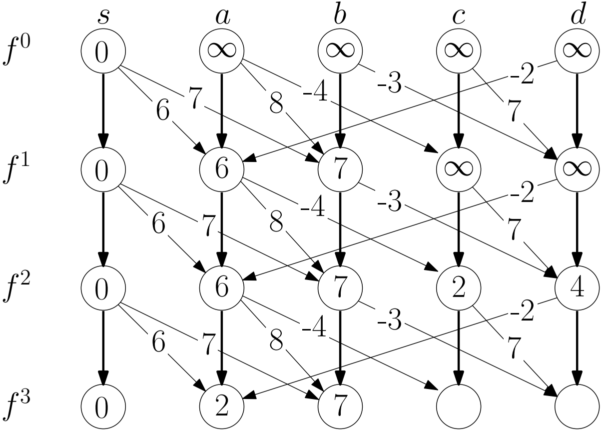
Dynamic Programming: Example



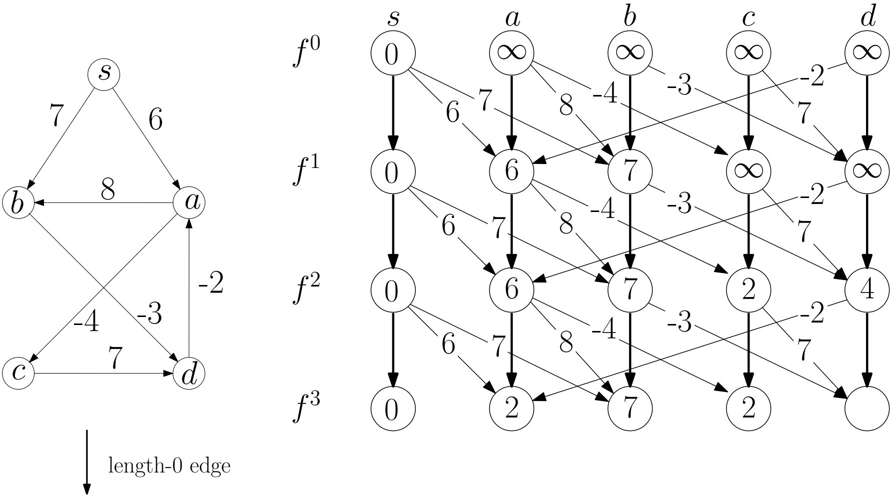


Dynamic Programming: Example

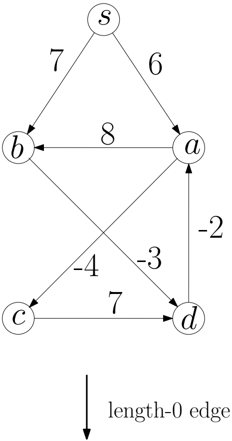


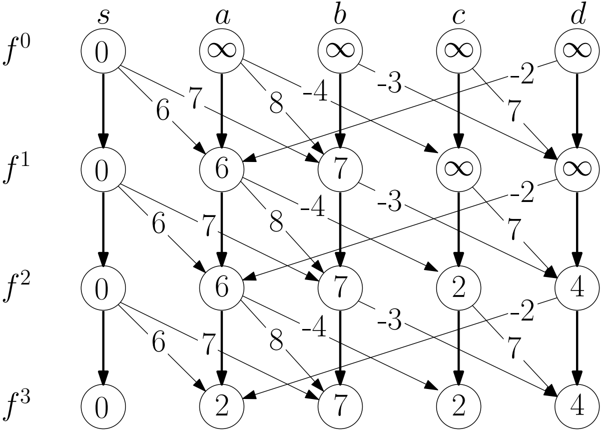


Dynamic Programming: Example

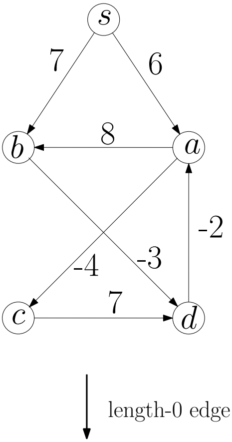


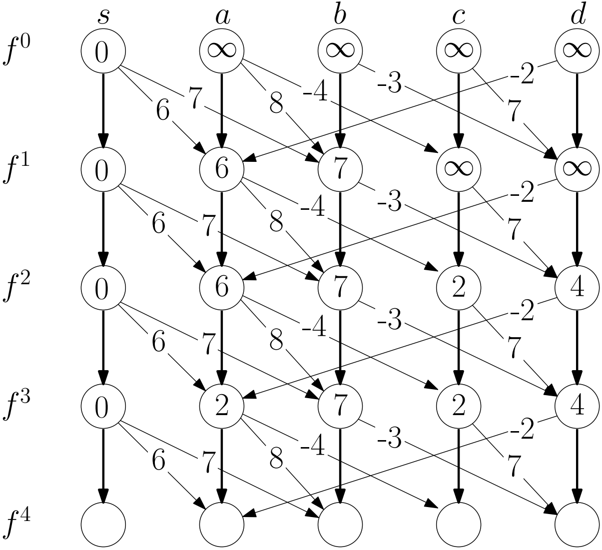
Dynamic Programming: Example



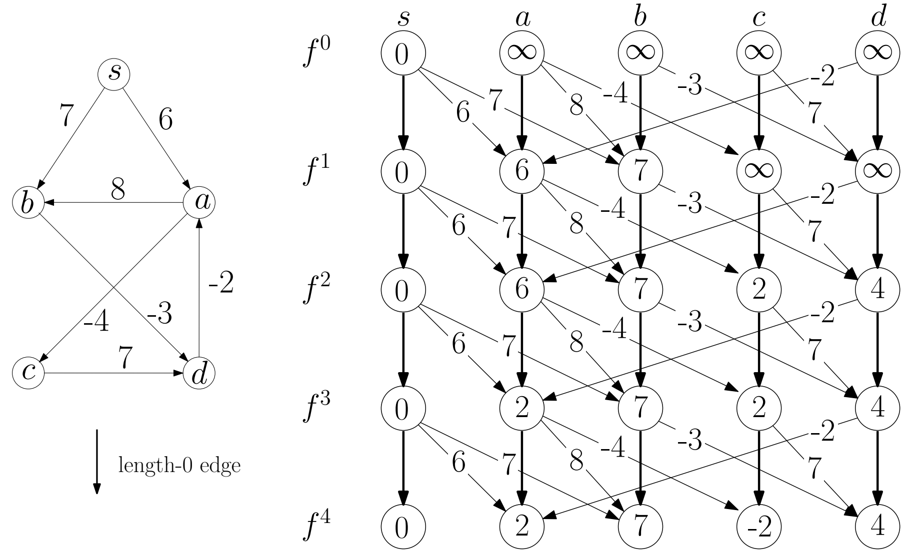


Dynamic Programming: Example





Dynamic Programming: Example



dynamic-programming(G, w, s)

```
f^0[s] ← 0 and f^0[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    copy f^{ℓ−1} → f^ℓ
    for each (u, v) ∈ E do
        if f^{ℓ−1}[u] + w(u, v) < f^ℓ[v] then
            f^ℓ[v] ← f^{ℓ−1}[u] + w(u, v)
return (f^{n−1}[v])_{v∈V}
```

dynamic-programming(G, w, s)

```
f^0[s] ← 0 and f^0[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    copy f^{ℓ−1} → f^ℓ
    for each (u, v) ∈ E do
        if f^{ℓ−1}[u] + w(u, v) < f^ℓ[v] then
            f^ℓ[v] ← f^{ℓ−1}[u] + w(u, v)
return (f^{n−1}[v])_{v∈V}
```

Obs. Assuming there are no negative cycles, then a shortest path contains at most $n - 1$ edges

dynamic-programming(G, w, s)

$f^0[s] \leftarrow 0$ and $f^0[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    copy $f^{\ell-1} \rightarrow f^\ell$
    for each $(u, v) \in E$ **do**
        if $f^{\ell-1}[u] + w(u, v) < f^\ell[v]$ **then**
            $f^\ell[v] \leftarrow f^{\ell-1}[u] + w(u, v)$
return $(f^{n-1}[v])_{v \in V}$

Obs. Assuming there are no negative cycles, then a shortest path contains at most $n-1$ edges

Proof.

If there is a path containing at least n edges, then it contains a cycle. Removing the cycle gives a path with the same or smaller length. $\square$

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

```
f^old[s] ← 0 and f^old[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    copy f^old → f^new
    for each (u, v) ∈ E do
        if f^old[u] + w(u, v) < f^new[v] then
            f^new[v] ← f^old[u] + w(u, v)
    copy f^new → f^old
return f^old
```

- f^ℓ only depends on $f^{\ell-1}$: only need 2 vectors

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

$f^{\text{old}}[s] \leftarrow 0$ and $f^{\text{old}}[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    copy $f^{\text{old}} \to f^{\text{new}}$
    for each $(u, v) \in E$ **do**
        if $f^{\text{old}}[u] + w(u, v) < f^{\text{new}}[v]$ **then**
            $f^{\text{new}}[v] \leftarrow f^{\text{old}}[u] + w(u, v)$
    copy $f^{\text{new}} \to f^{\text{old}}$
return f^{old}

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

```
f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    copy f → f
    for each (u, v) ∈ E do
        if f[u] + w(u, v) < f[v] then
            f[v] ← f[u] + w(u, v)
    copy f → f
return f
```

- f^{ℓ} only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Dynamic Programming with Better Space Usage

dynamic-programming(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    for each $(u, v) \in E$ **do**
        if $f[u] + w(u, v) < f[v]$ **then**
            $f[v] \leftarrow f[u] + w(u, v)$
return f

- f^ℓ only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

```
f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    for each (u, v) ∈ E do
        if f[u] + w(u, v) < f[v] then
            f[v] ← f[u] + w(u, v)
return f
```

- f^ℓ only depends on $f^{\ell-1}$: only need 2 vectors
- only need 1 vector!

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    for each $(u, v) \in E$ **do**
        if $f[u] + w(u, v) < f[v]$ **then**
            $f[v] \leftarrow f[u] + w(u, v)$
return f

- Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
  for each $(u,v) \in E$ **do**
    if $f[u] + w(u,v) < f[v]$ **then**
      $f[v] \leftarrow f[u] + w(u,v)$
return f

- Issue: when we compute $f[u] + w(u,v)$, $f[u]$ may be changed since the end of last iteration
- This is OK: it can only "accelerate" the process!

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

```
f[s] ← 0 and f[v] ← ∞ for any v ∈ V \ {s}
for ℓ ← 1 to n − 1 do
    for each (u, v) ∈ E do
        if f[u] + w(u, v) < f[v] then
            f[v] ← f[u] + w(u, v)
return f
```

- Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration
- This is OK: it can only "accelerate" the process!
- After iteration ℓ, $f[v]$ is at most the length of the shortest path from s to v that uses at most ℓ edges

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to $n-1$ **do**
    for each $(u, v) \in E$ **do**
        if $f[u] + w(u, v) < f[v]$ **then**
            $f[v] \leftarrow f[u] + w(u, v)$
return f

- Issue: when we compute $f[u] + w(u, v)$, $f[u]$ may be changed since the end of last iteration
- This is OK: it can only "accelerate" the process!
- After iteration ℓ, $f[v]$ is at most the length of the shortest path from s to v that uses at most ℓ edges
- $f[v]$ is always the length of some path from s to v

Bellman-Ford Algorithm

- After iteration ℓ:

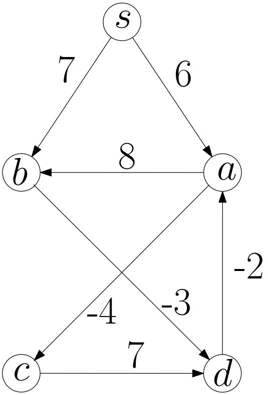
$$\begin{aligned} &\text{length of shortest } s\text{-}v \text{ path} \\ &\leq f[v] \\ &\leq \text{length of shortest } s\text{-}v \text{ path using at most } \ell \text{ edges} \end{aligned}$$

- Assuming there are no negative cycles:

$$\begin{aligned} &\text{length of shortest } s\text{-}v \text{ path} \\ &= \text{length of shortest } s\text{-}v \text{ path using at most } n-1 \text{ edges} \end{aligned}$$

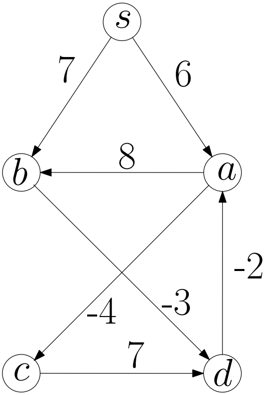
- So, assuming there are no negative cycles, after iteration $n-1$:

$$f[v] = \text{length of shortest } s\text{-}v \text{ path}$$



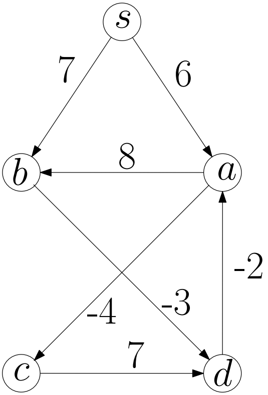
- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	∞	∞	∞	∞



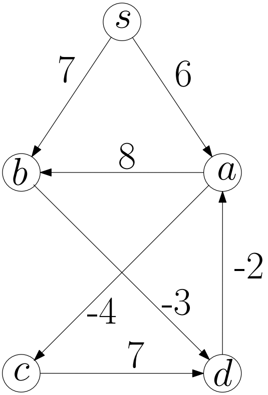
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	∞	∞	∞	∞



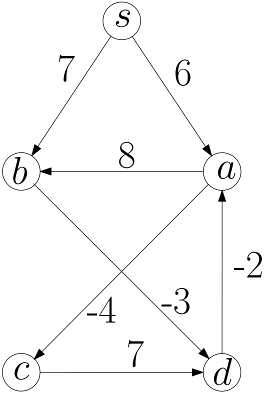
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	∞	∞	∞



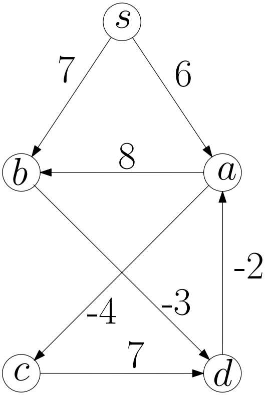
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	∞	∞	∞



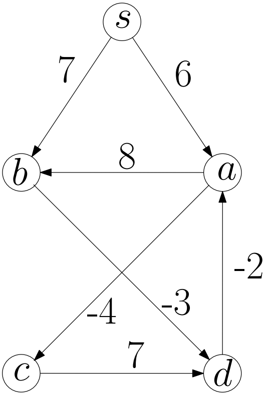
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	∞	∞



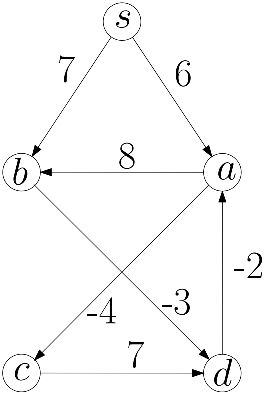
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	∞	∞



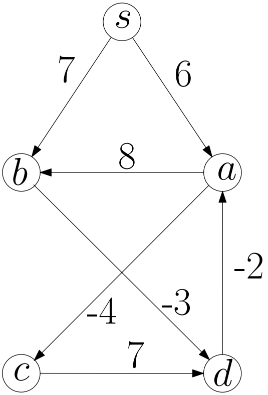
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	∞	∞



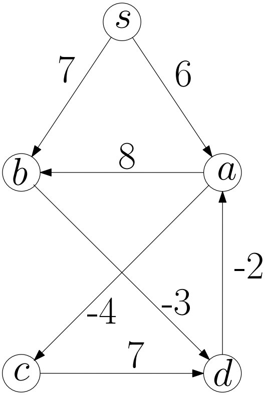
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	∞



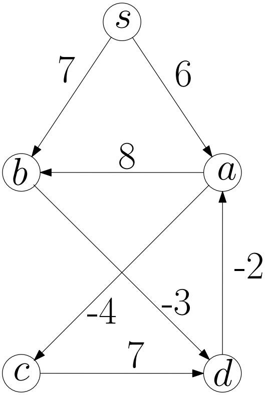
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	∞



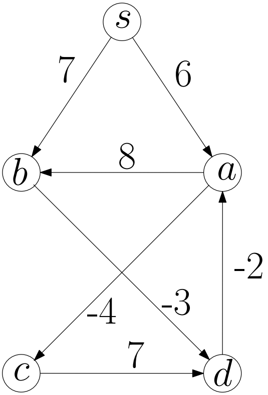
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	4



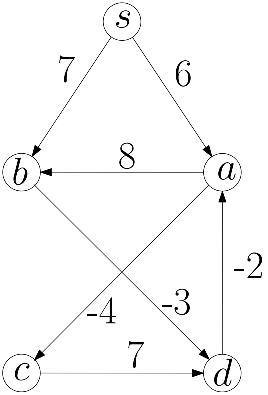
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	4



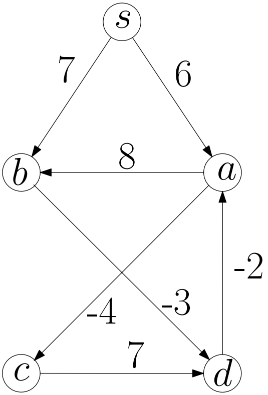
- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	6	7	2	4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

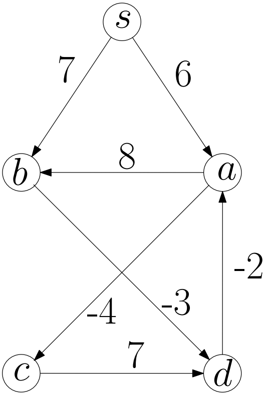
vertices	s	a	b	c	d
f	0	2	7	2	4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	2	4

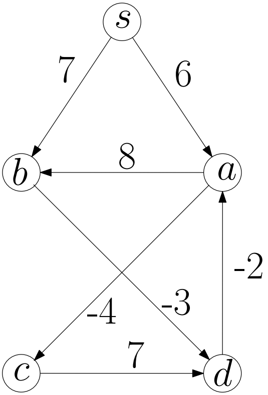
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	2	4

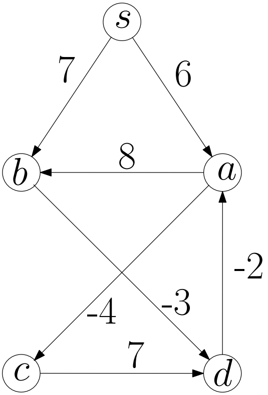
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	2	4

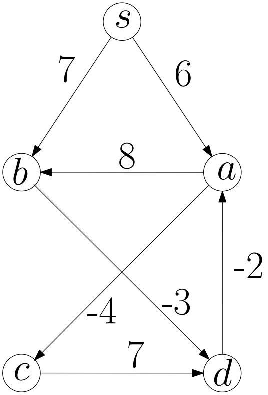
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	2	4

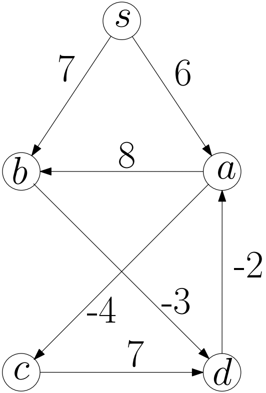
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	2	4

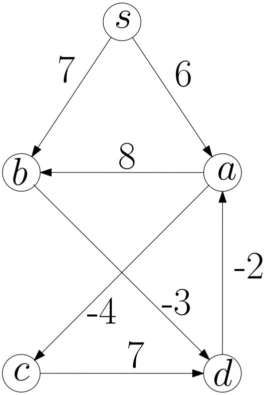
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

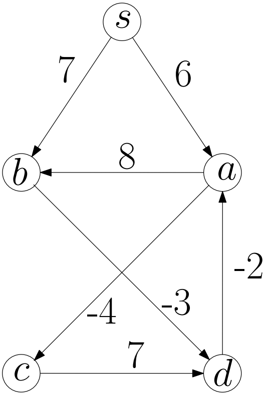
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

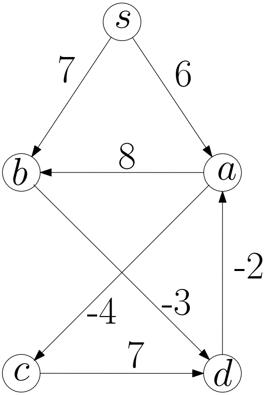
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

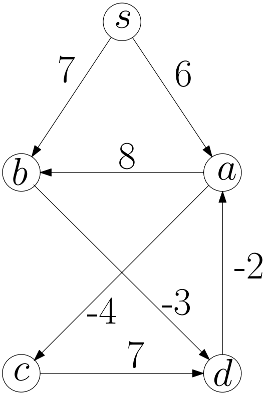
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

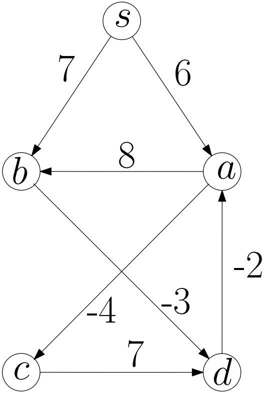
- end of iteration 1: 0, 2, 7, 2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

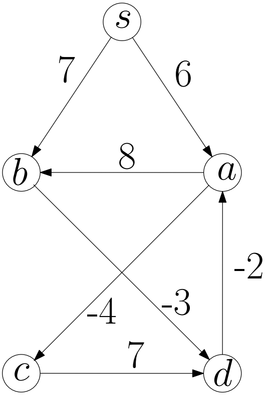
- end of iteration 1: 0, 2, 7, 2, 4
- end of iteration 2: 0, 2, 7, -2, 4



- order in which we consider edges: (s, a), (s, b), (a, b), (a, c), (b, d), (c, d), (d, a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

- end of iteration 1: 0, 2, 7, 2, 4
- end of iteration 2: 0, 2, 7, -2, 4
- end of iteration 3: 0, 2, 7, -2, 4



- order in which we consider edges: (s,a), (s,b), (a,b), (a,c), (b,d), (c,d), (d,a)

vertices	s	a	b	c	d
f	0	2	7	-2	4

- end of iteration 1: 0, 2, 7, 2, 4
- end of iteration 2: 0, 2, 7, -2, 4
- end of iteration 3: 0, 2, 7, -2, 4
- Algorithm terminates in 3 iterations, instead of 4.

Bellman-Ford Algorithm

Bellman-Ford(G, w, s)

$f[s] \leftarrow 0$ and $f[v] \leftarrow \infty$ for any $v \in V \setminus \{s\}$
for $\ell \leftarrow 1$ to n **do**
    $updated \leftarrow$ false
    for each $(u, v) \in E$ **do**
        if $f[u] + w(u, v) < f[v]$ **then**
            $f[v] \leftarrow f[u] + w(u, v)$
            $updated \leftarrow$ true
    if not $updated$, then return f
output "negative cycle exists"