

Bellman-Ford and Dynamic Programming on Graphs

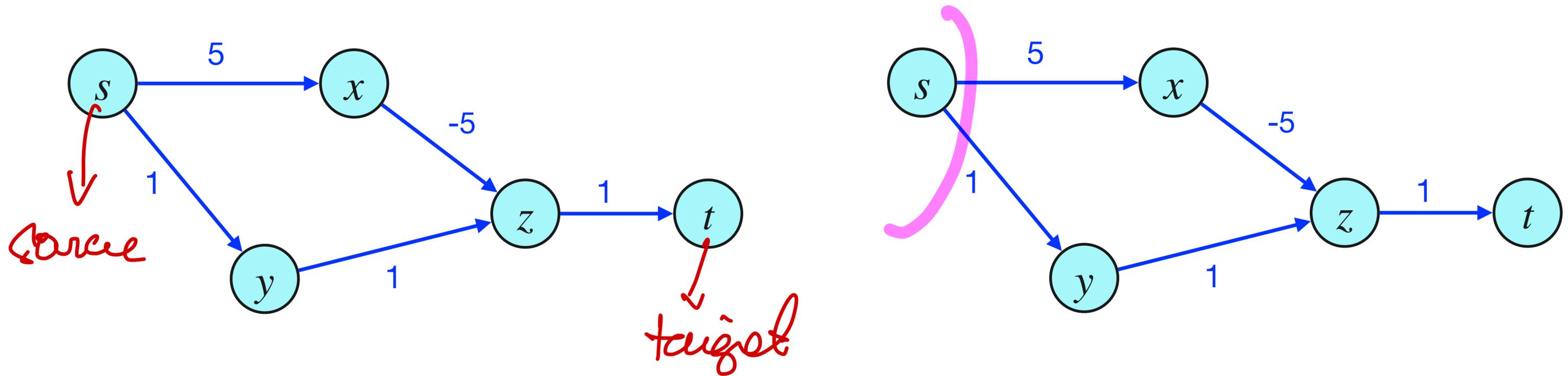
Sides based on material by Kani, Erickson, Chekuri, et. al.

All mistakes are my own! - Ivan Abraham (Fall 2024)

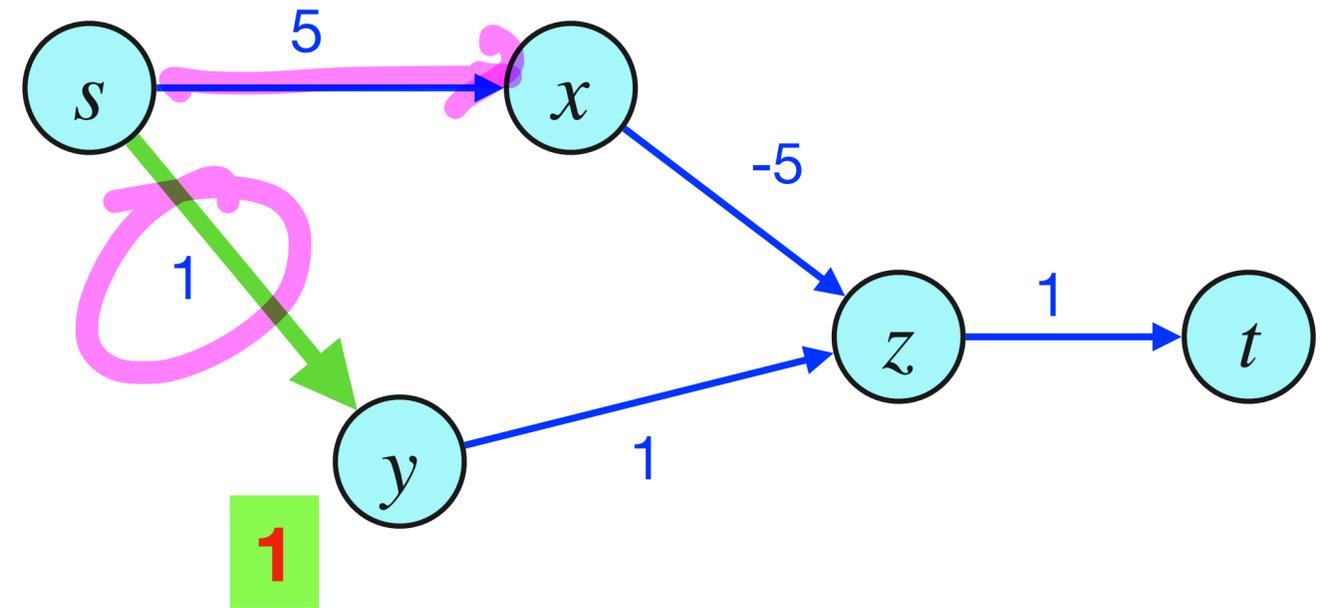
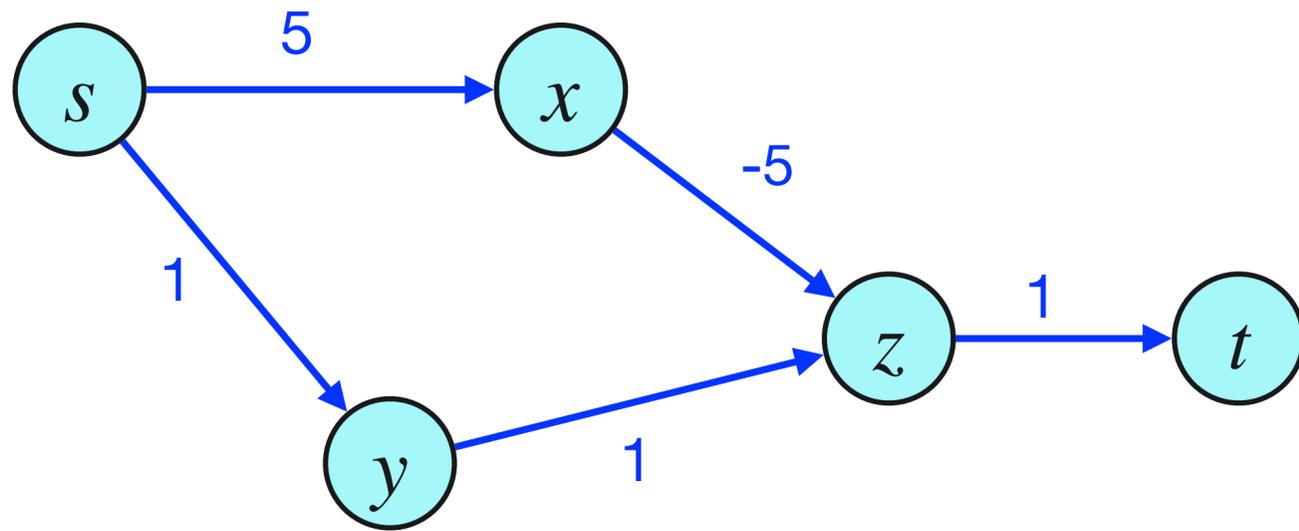
Image by ChatGPT (probably collaborated with DALL-E)

Why Dijkstra's algorithm fails with negative edges

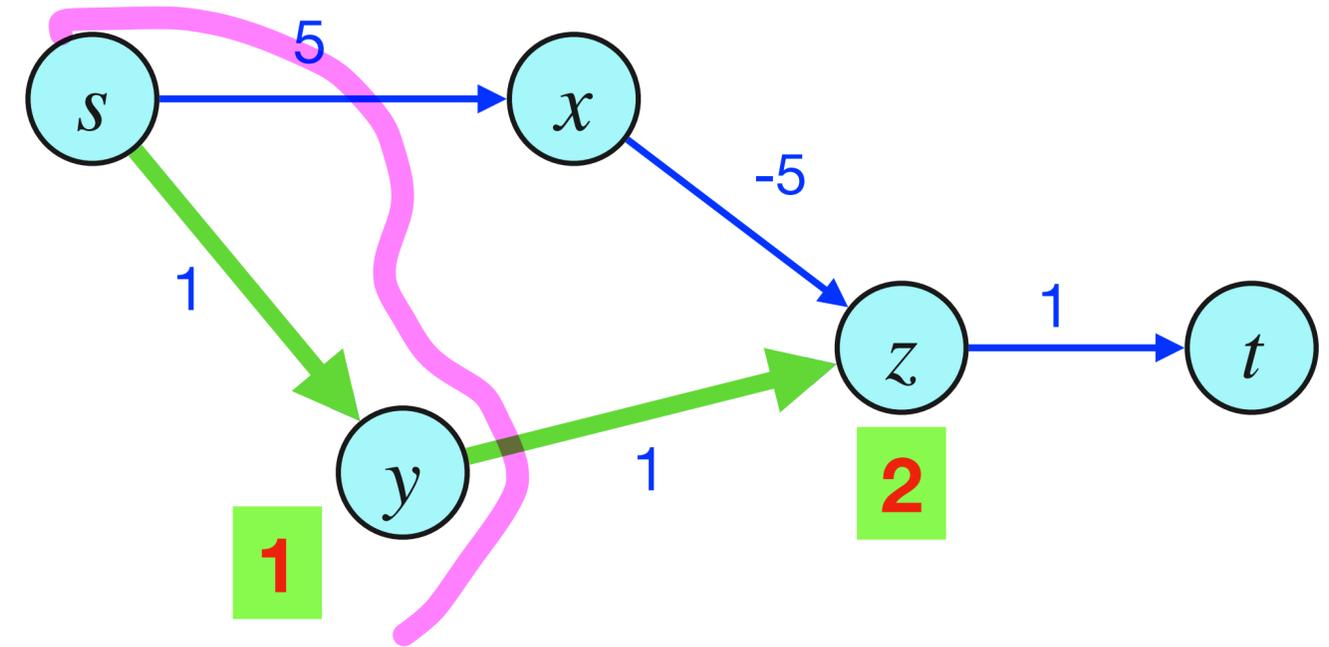
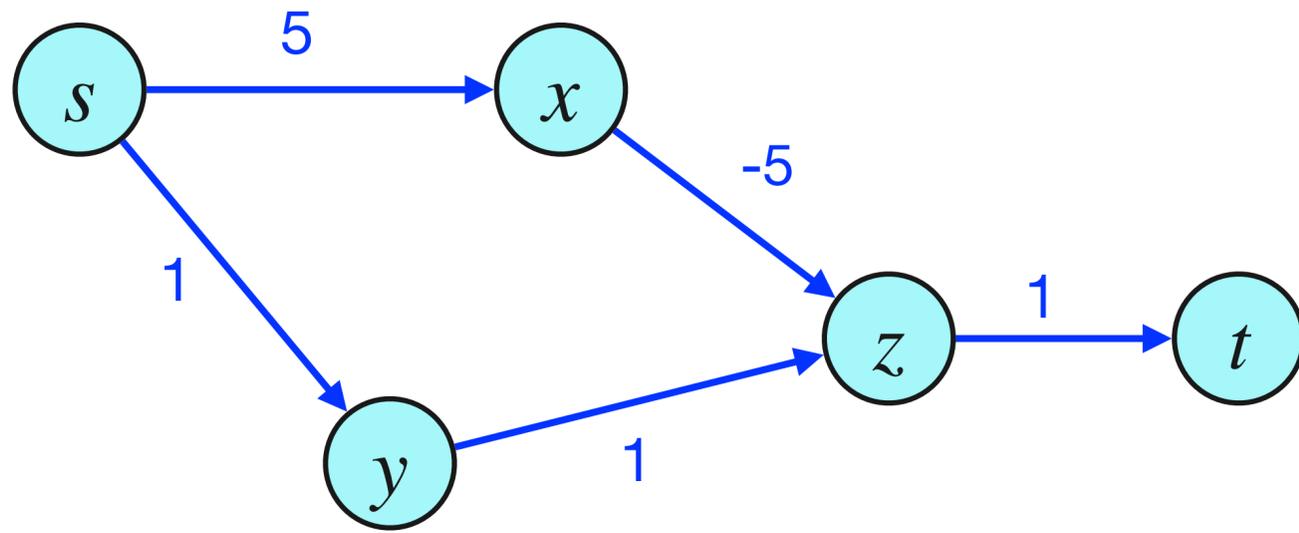
What are the distances computed by Dijkstra's algorithm?



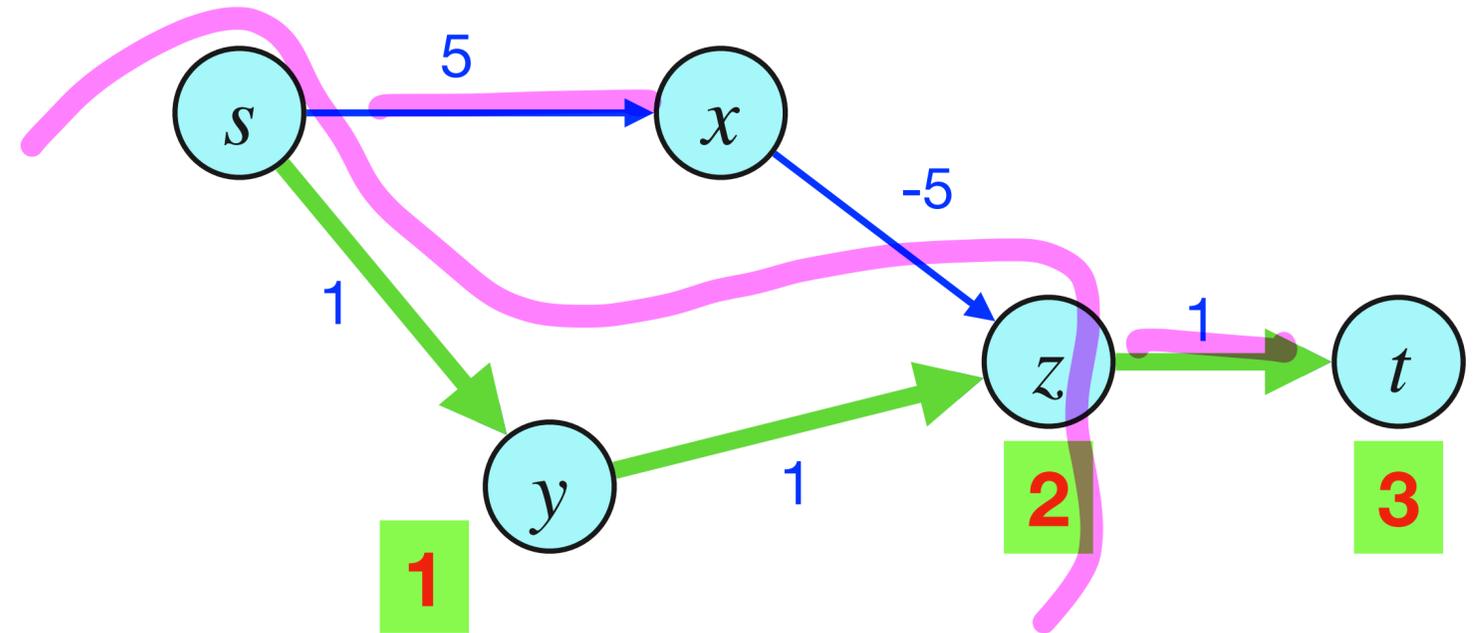
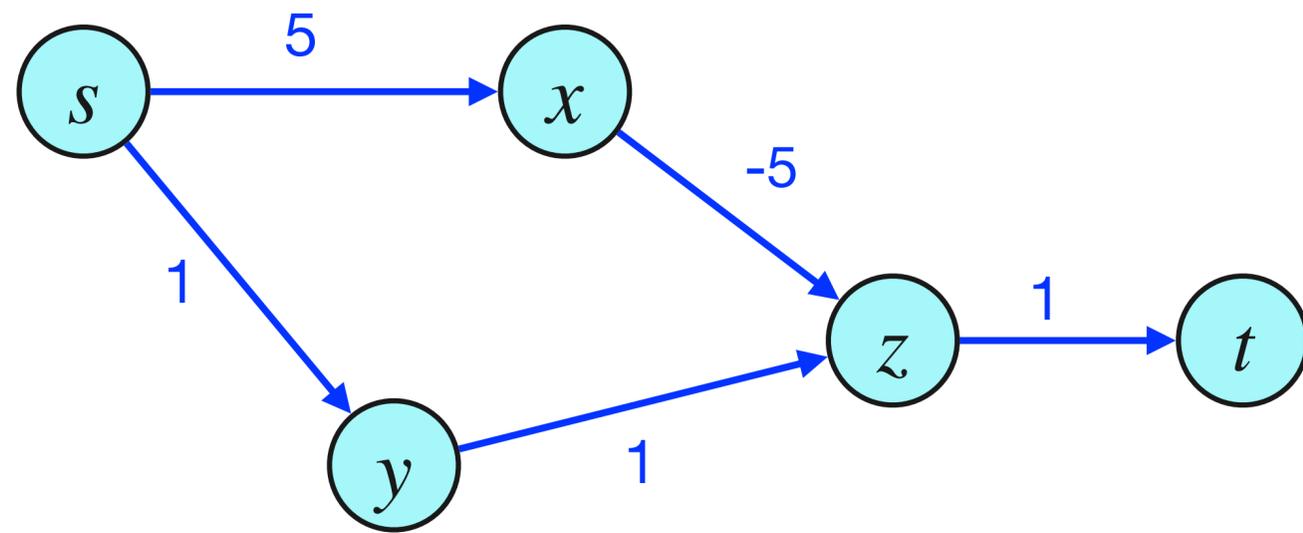
What are the distances computed by Dijkstra's algorithm?



What are the distances computed by Dijkstra's algorithm?

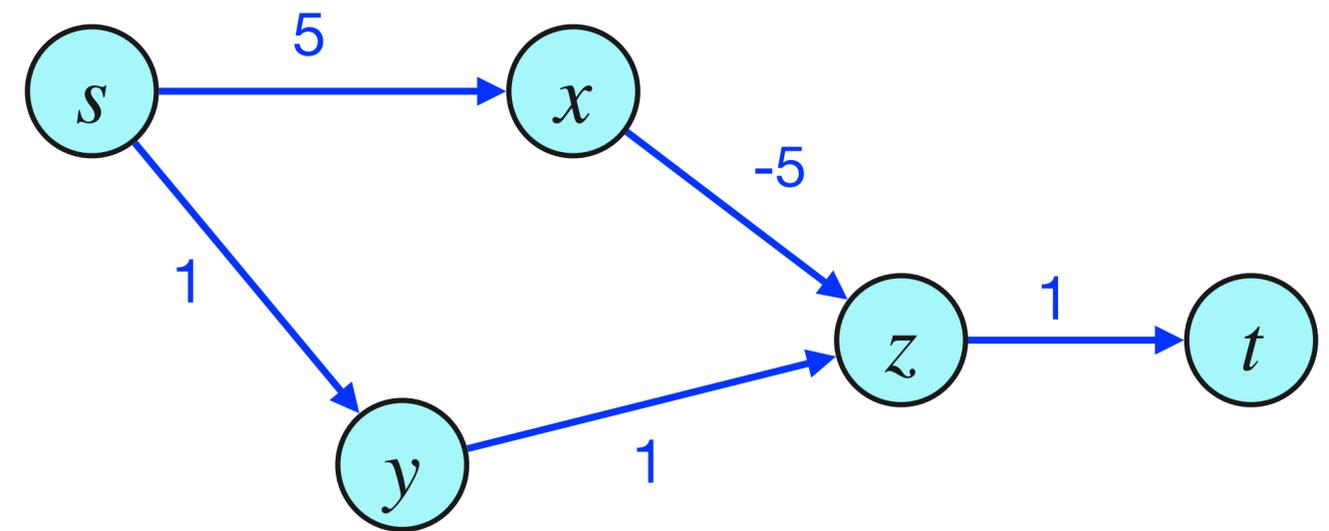
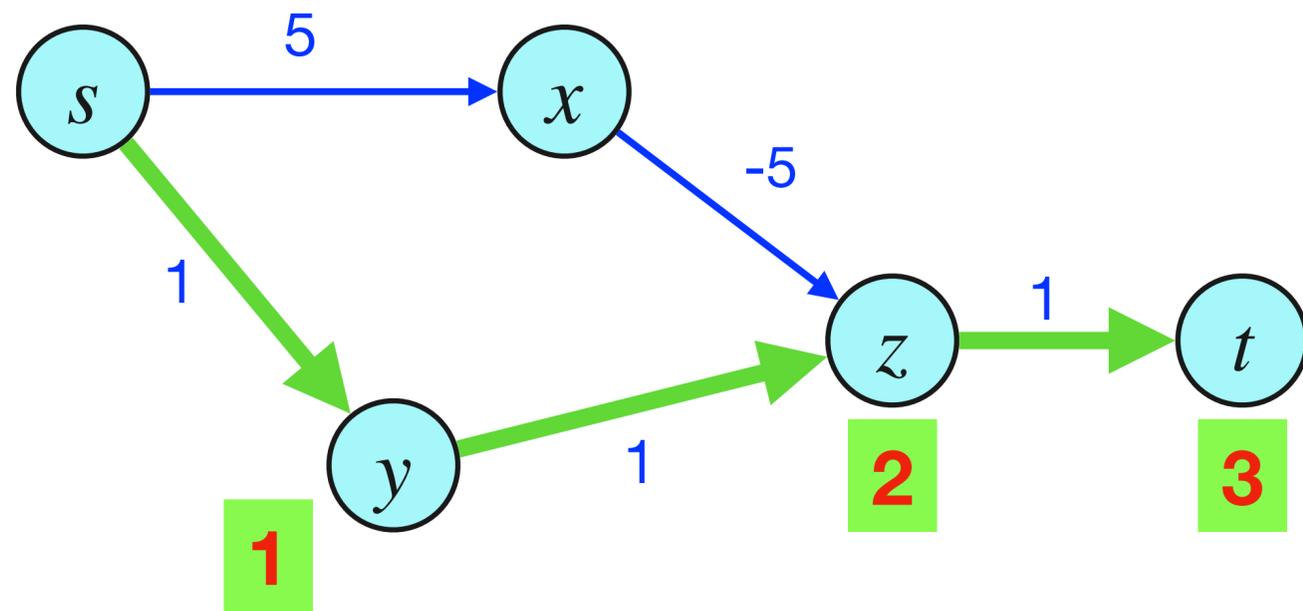


What are the distances computed by Dijkstra's algorithm?



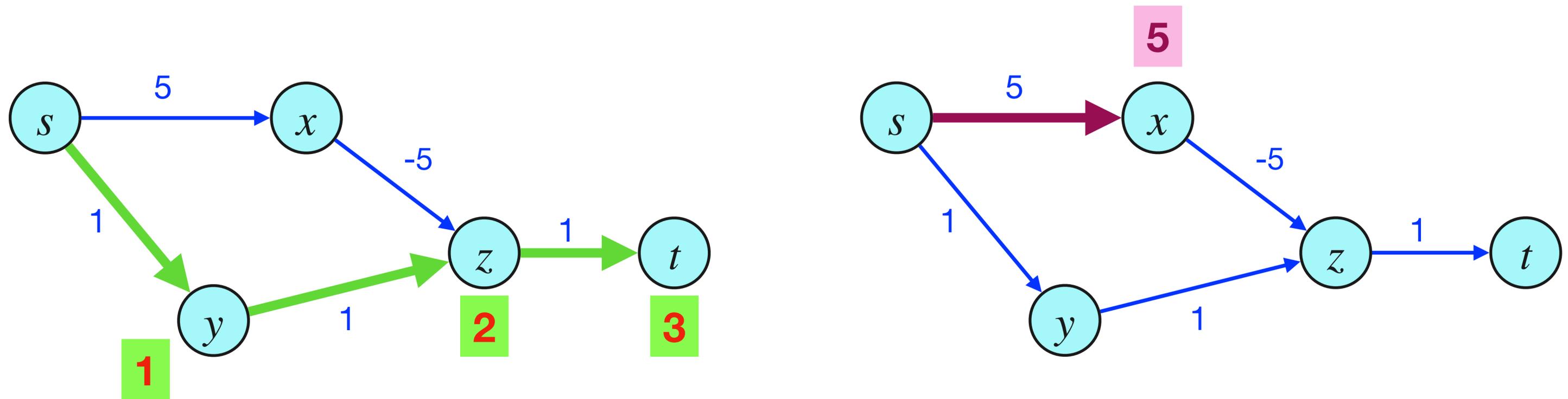
What are the distances computed by Dijkstra's algorithm?

But that is not the shortest path!



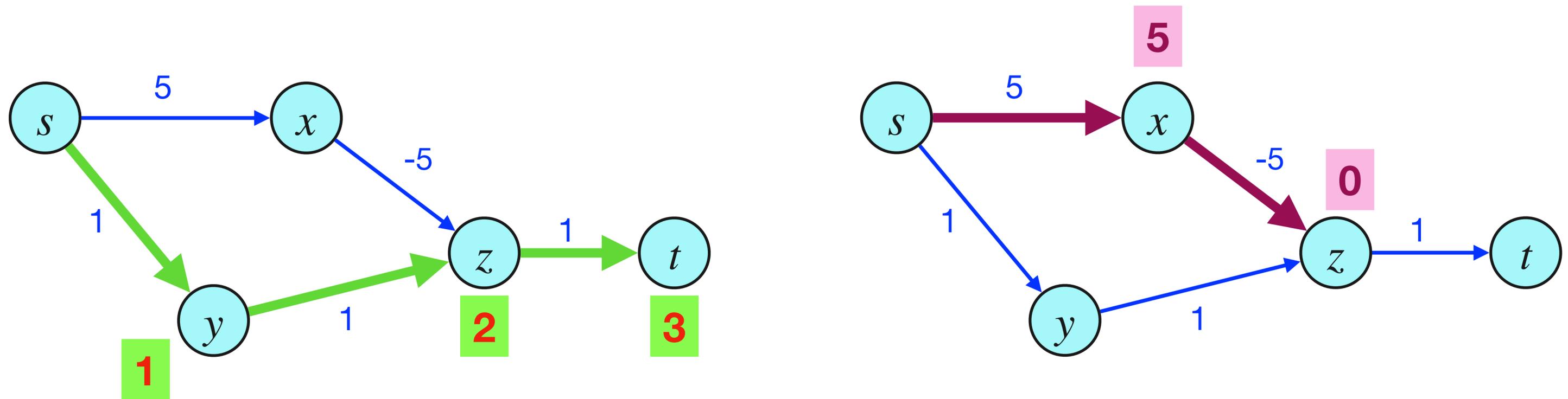
What are the distances computed by Dijkstra's algorithm?

But that is not the shortest path!



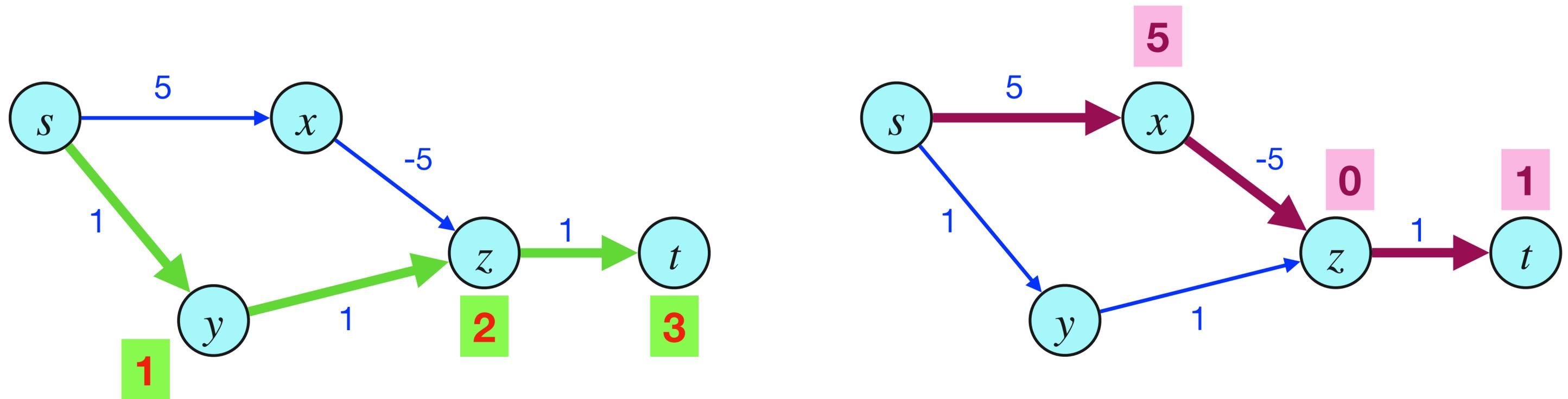
What are the distances computed by Dijkstra's algorithm?

But that is not the shortest path!



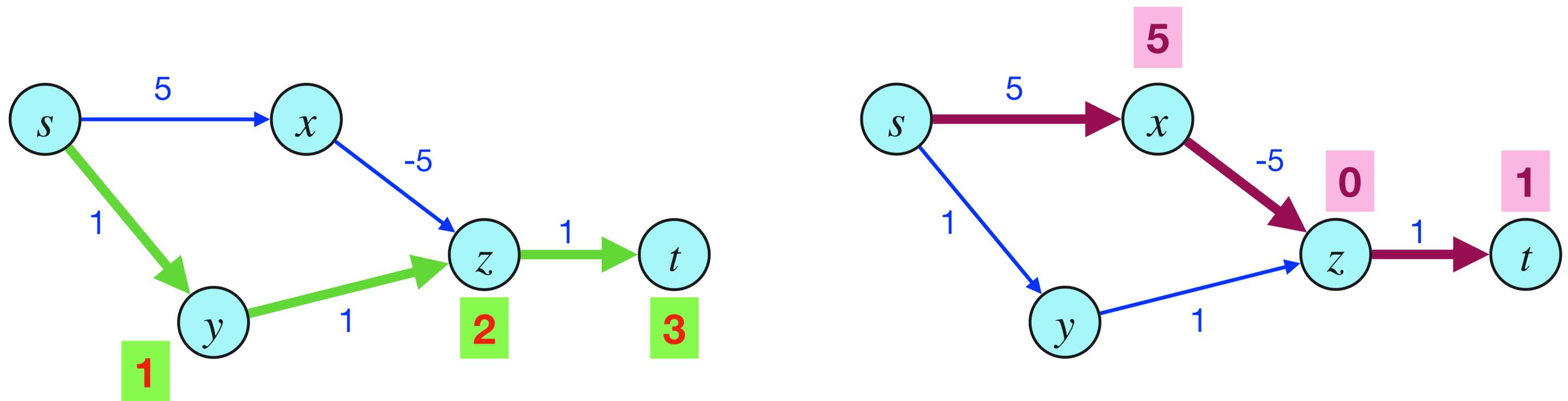
What are the distances computed by Dijkstra's algorithm?

But that is not the shortest path!



What are the distances computed by Dijkstra's algorithm?

But that is not the shortest path!



Problem: False assumption that if $s \rightarrow v_0 \rightarrow v_1 \rightarrow v_2 \dots \rightarrow v_k$ is a shortest path from s to v_k then $\text{dist}(s, v_i) \leq \text{dist}(s, v_{i+1})$ for $0 \leq i < k$. Holds true only for non-negative edge lengths.

Shortest paths with negative lengths

Lemma: Let G be a directed graph with *arbitrary* edge lengths and let

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \dots \rightarrow v_k = t$$

be a shortest path from s to t then for $1 \leq i < k$:

- $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots v_i$ is a shortest path from s to v_i

Shortest paths with negative lengths

Lemma: Let G be a directed graph with *arbitrary* edge lengths and let

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \dots \rightarrow v_k = t$$

be a shortest path from s to t then for $1 \leq i < k$:

- $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots v_i$ is a shortest path from s to v_i
- $\text{dist}(s, v_i) \leq \text{dist}(s, v_k)$ for $1 \leq i < k$.

Shortest paths with negative lengths

Lemma: Let G be a directed graph with *arbitrary* edge lengths and let

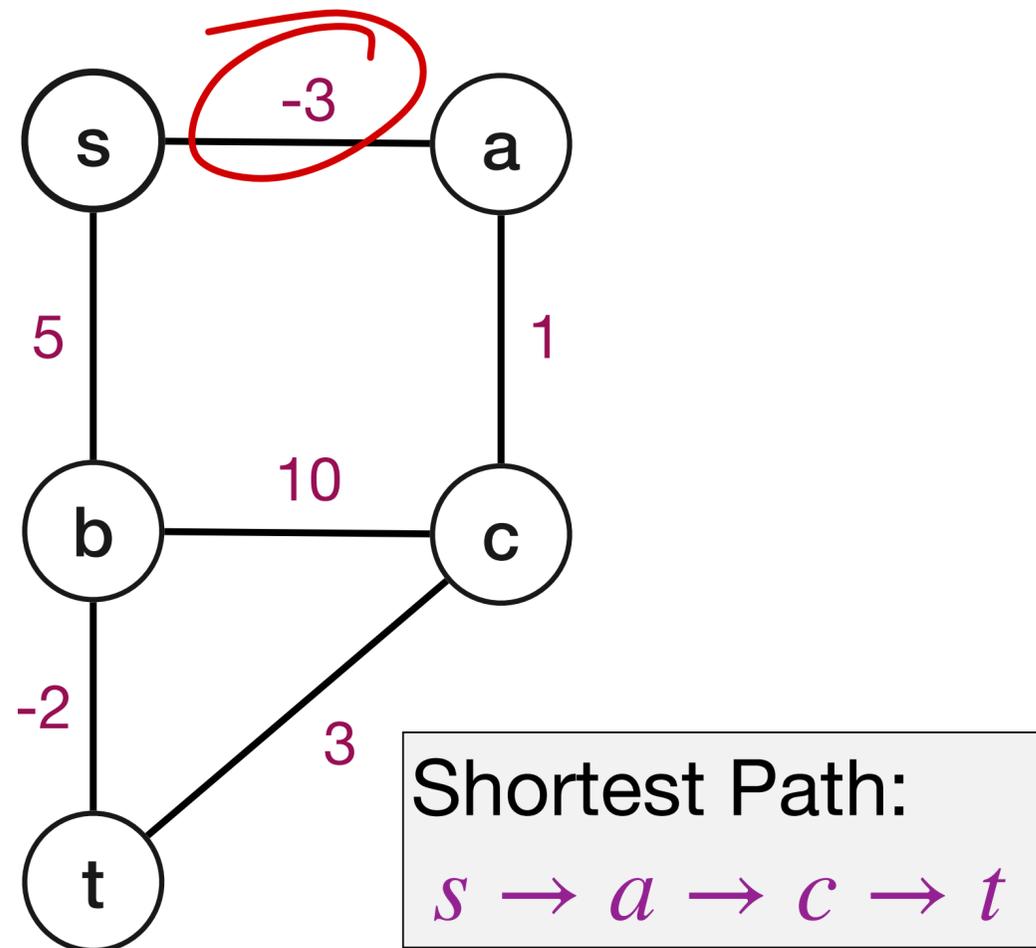
$$s = v_0 \rightarrow v_1 \rightarrow v_2 \dots \rightarrow v_k = t$$

be a shortest path from s to t then for $1 \leq i < k$:

- $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots v_i$ is a shortest path from s to v_i ✓
- $\text{dist}(s, v_i) \leq \text{dist}(s, v_k)$ for $1 \leq i < k$. \Rightarrow False!

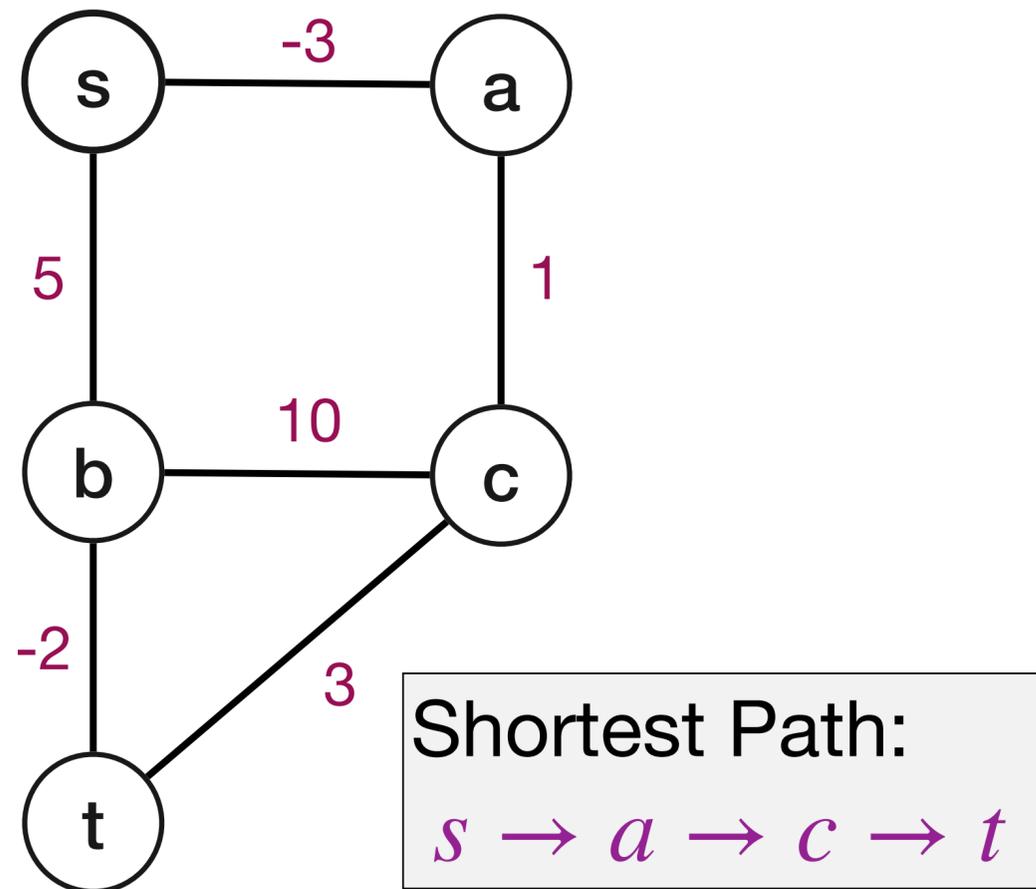
Why can't we just re-normalize the edge lengths?

Instinctual thought



Why can't we just re-normalize the edge lengths?

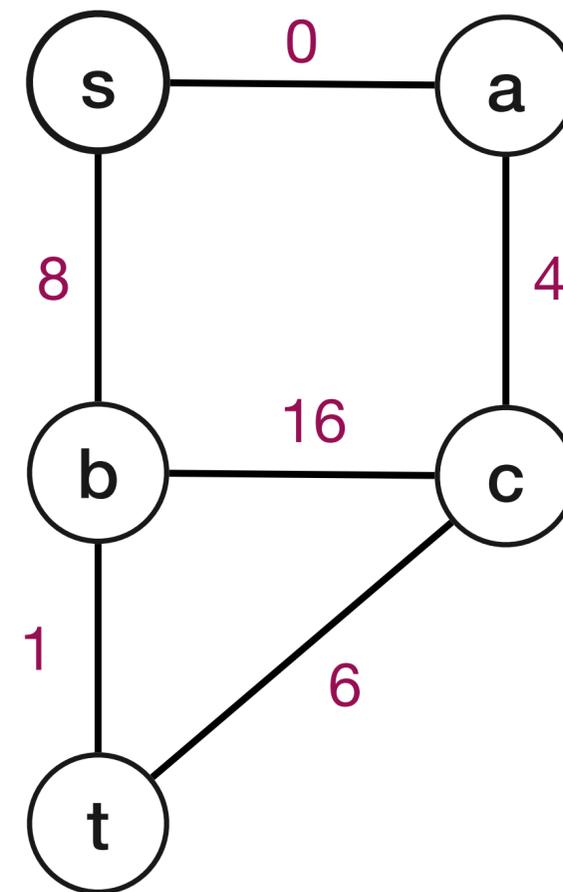
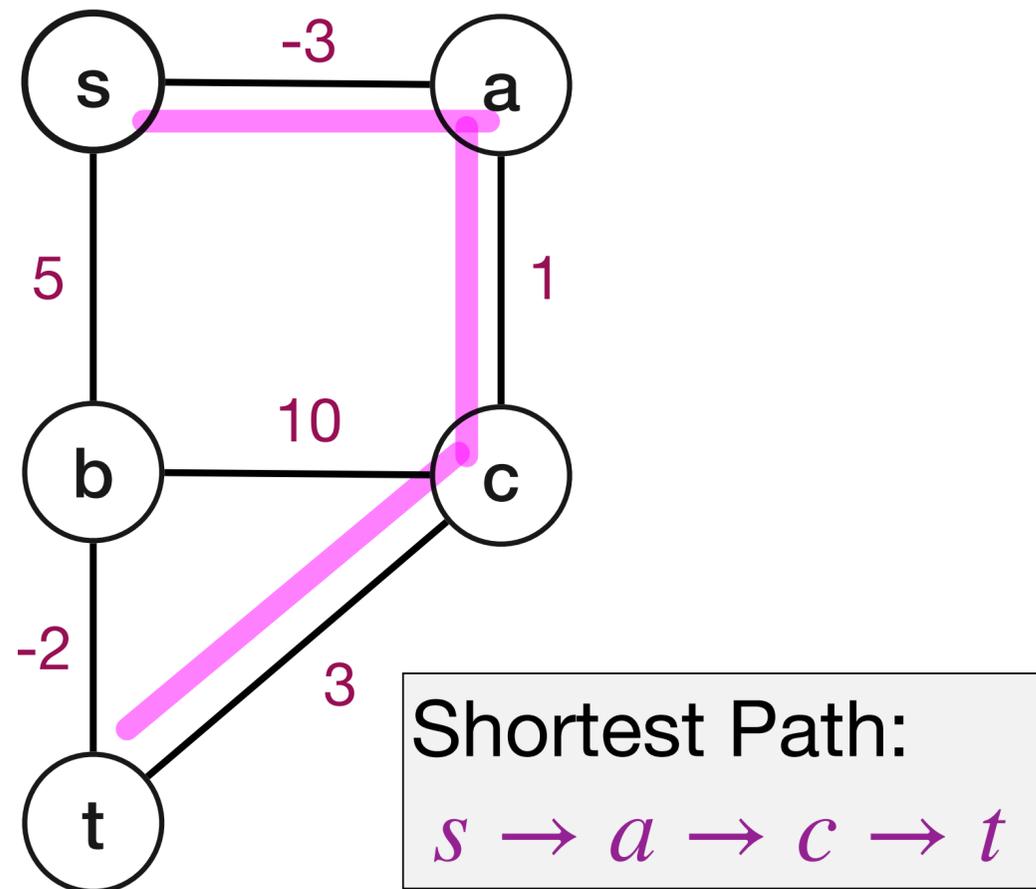
Instinctual thought



Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive)?

Why can't we just re-normalize the edge lengths?

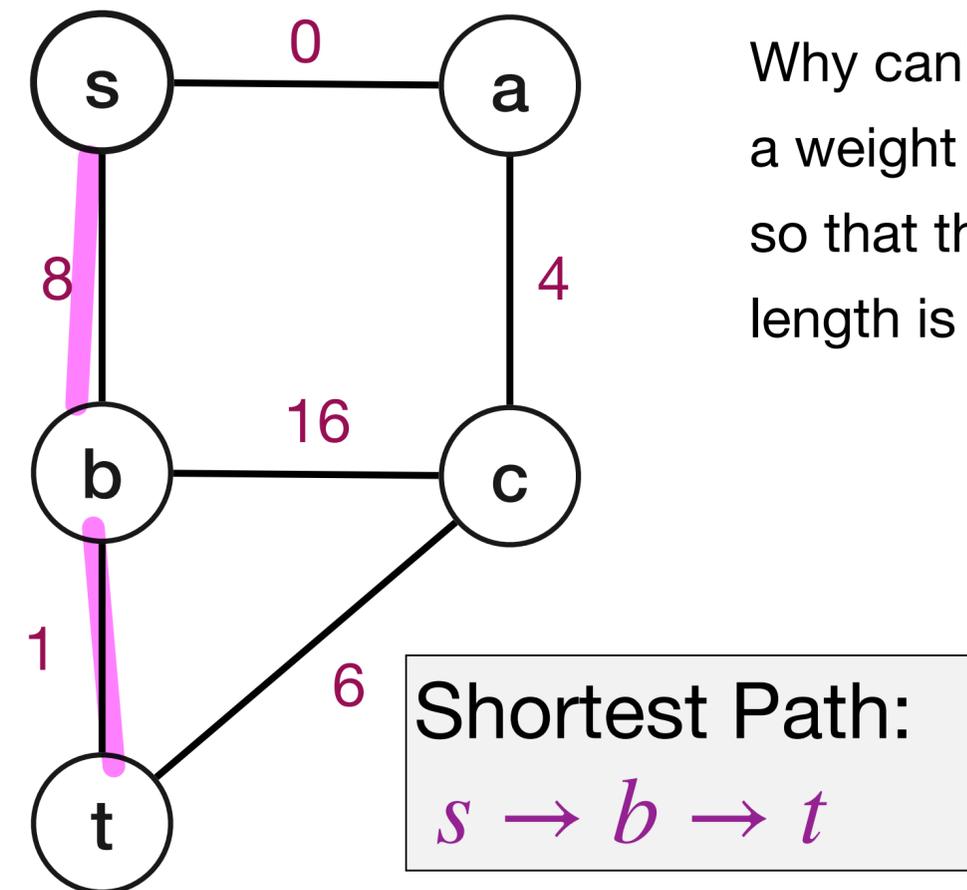
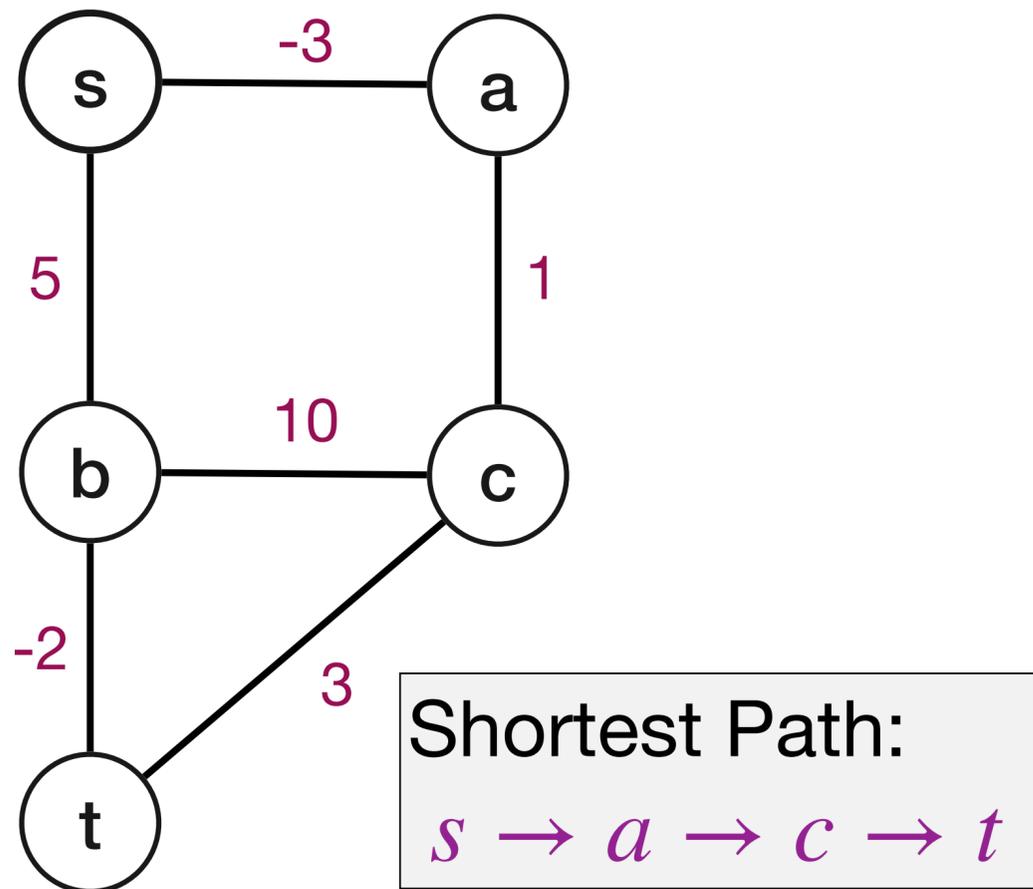
Instinctual thought



Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive)?

Why can't we just re-normalize the edge lengths?

Instinctual thought

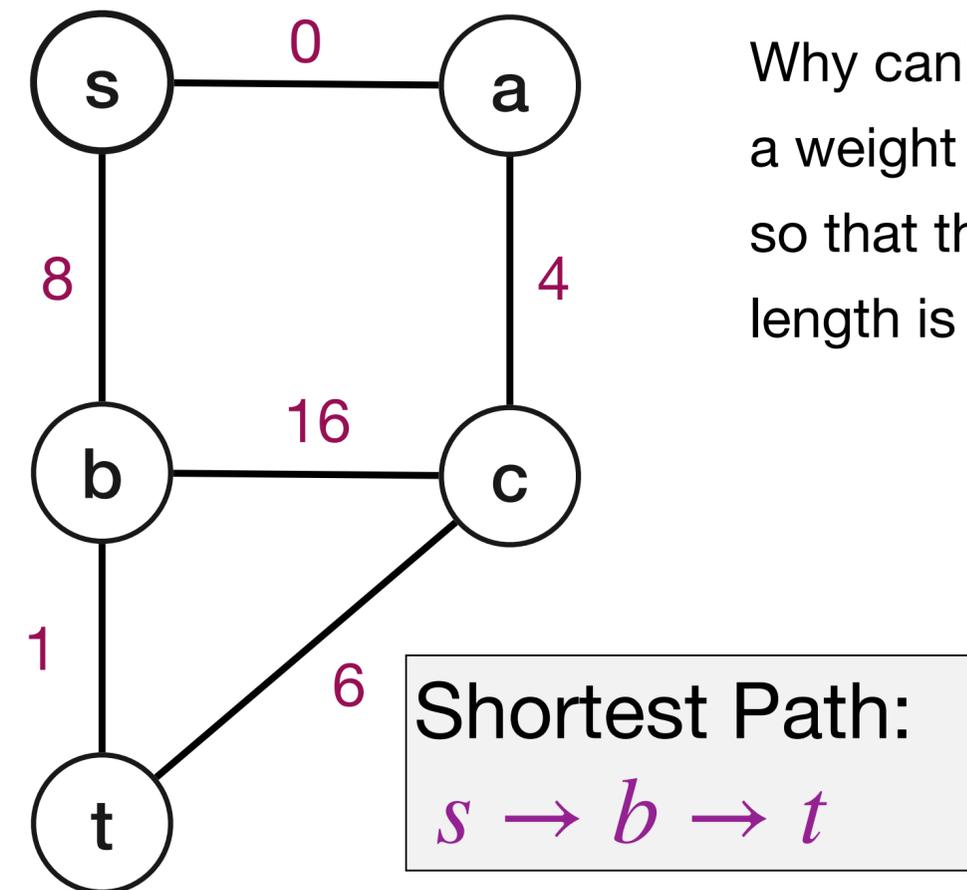
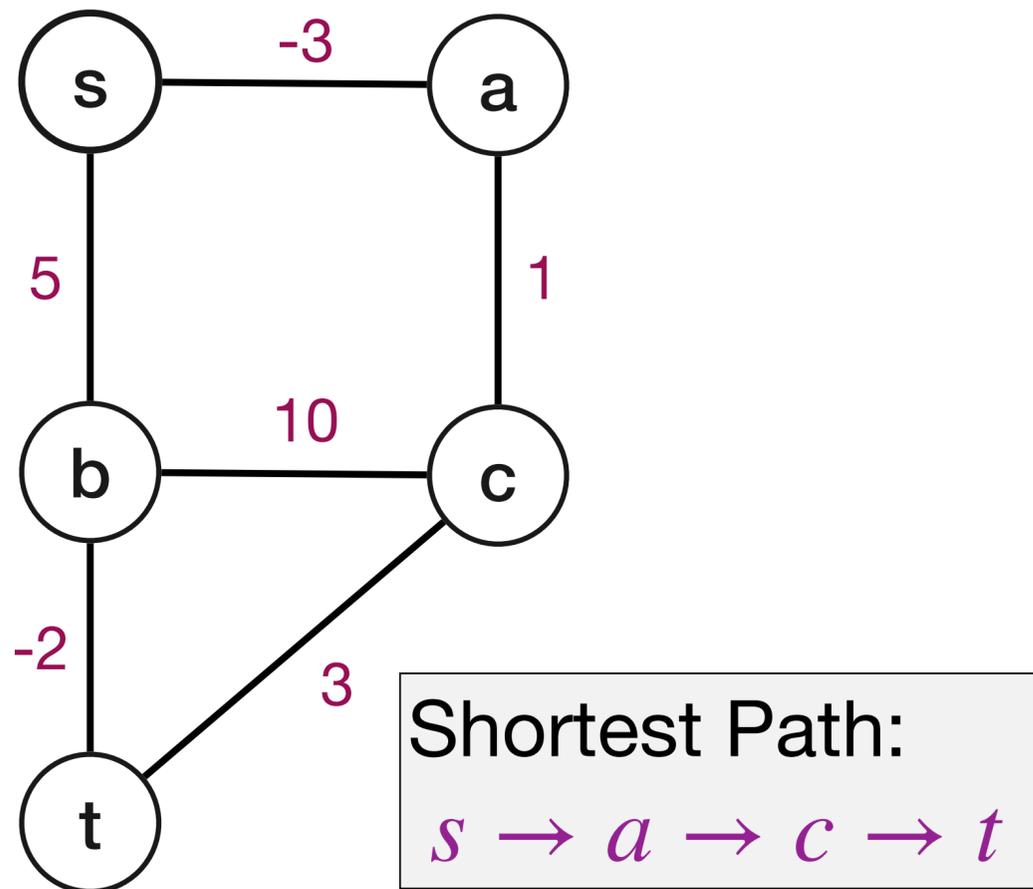


Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive)?

Why can't we just re-normalize the edge lengths?

Instinctual thought

Adding weights to edges penalizes paths with more edges, gives wrong path on original graph.

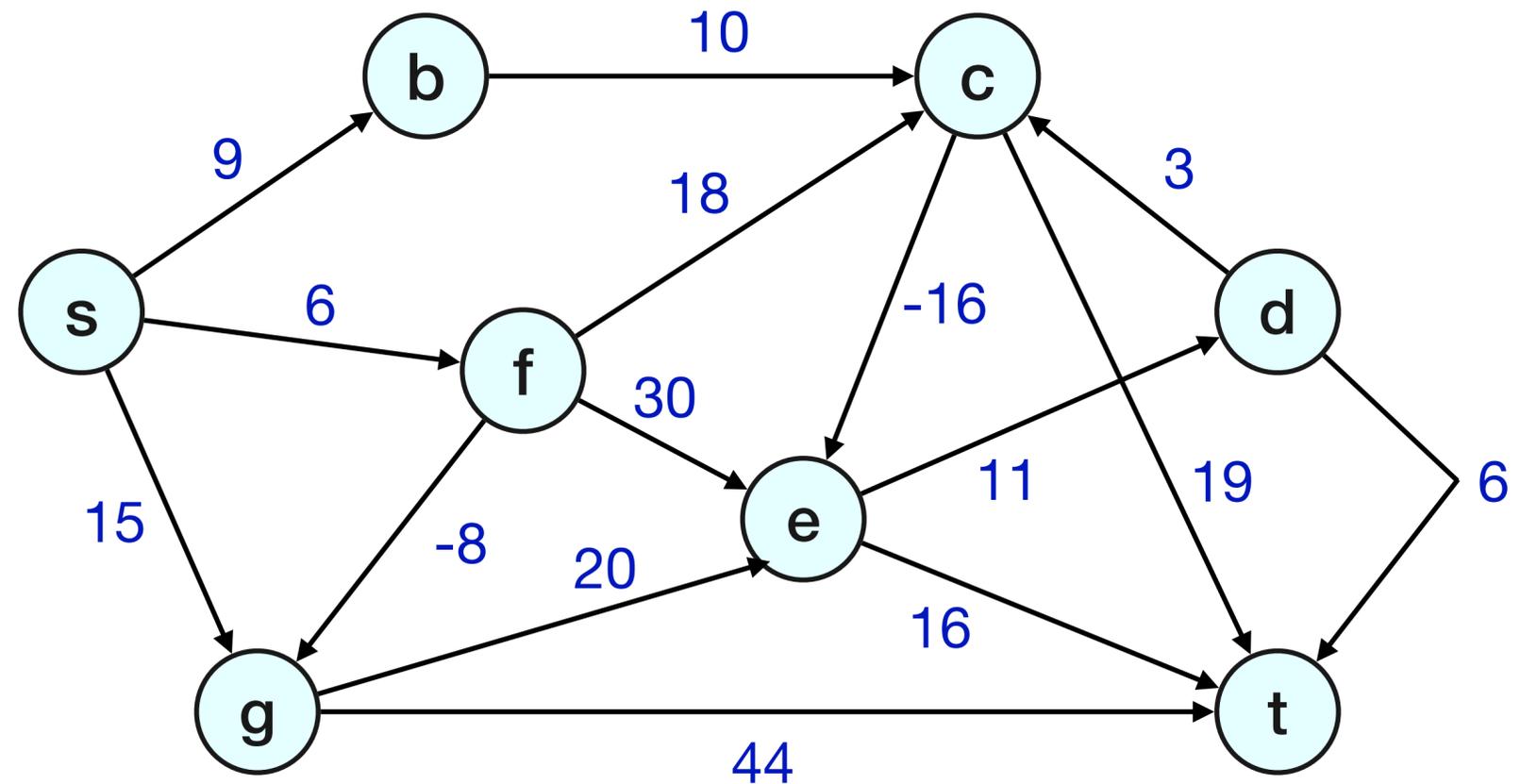


Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive)?

Negative length cycles

Definition

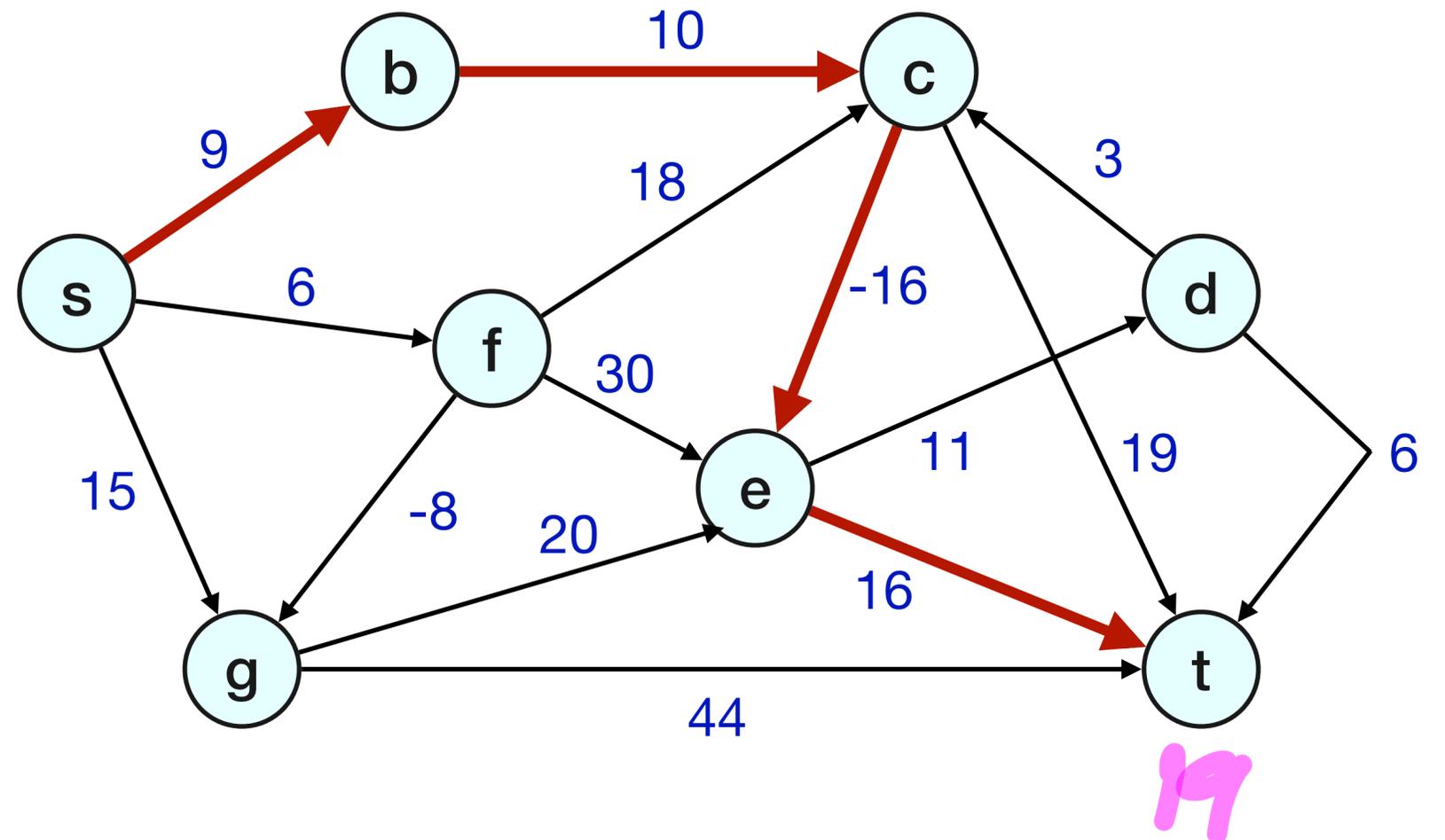
What is the shortest path distance between s and t ?



Negative length cycles

Definition

What is the shortest path distance between s and t ?

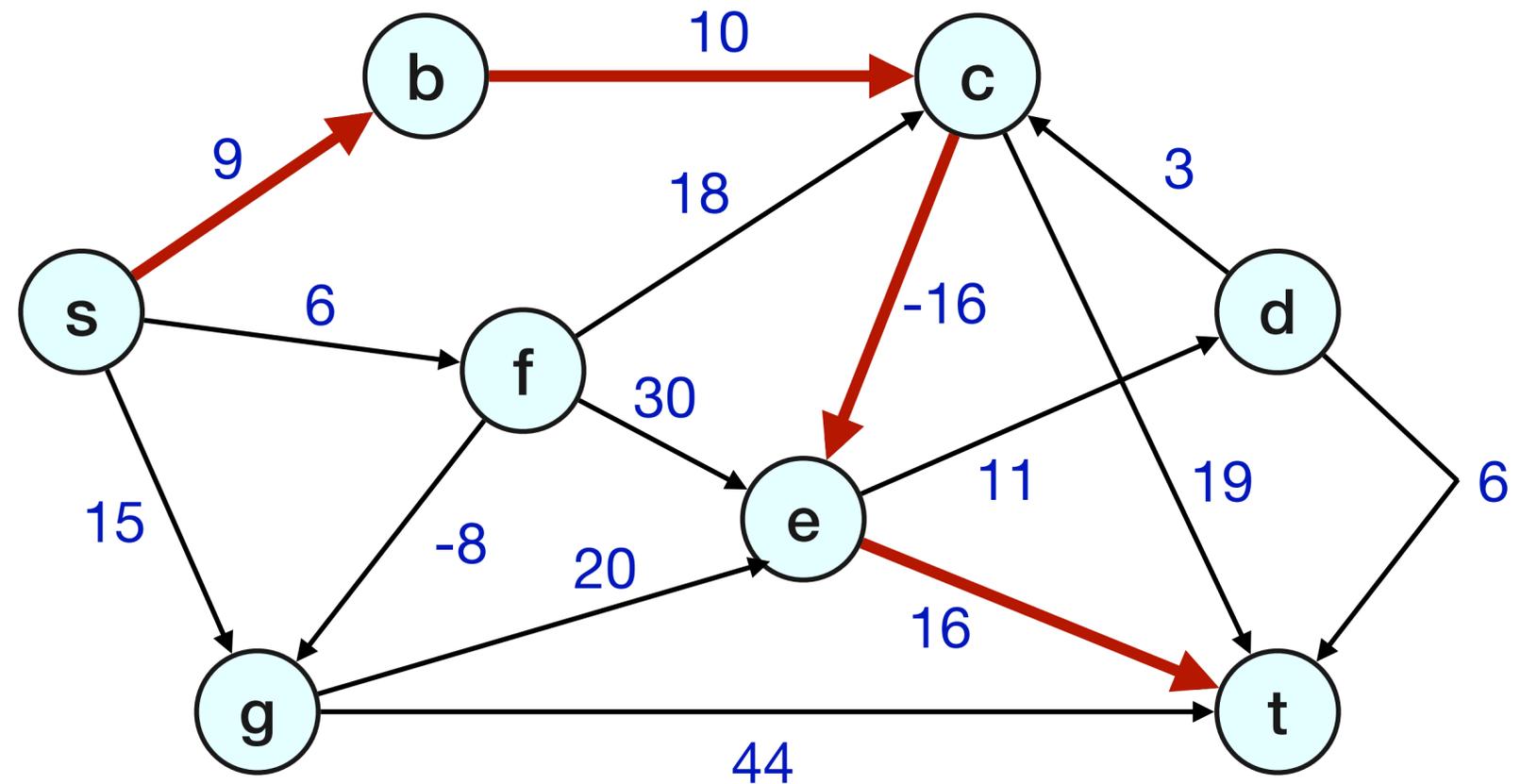


Negative length cycles

Definition

What is the shortest path distance between s and t ?

What about adding red path?

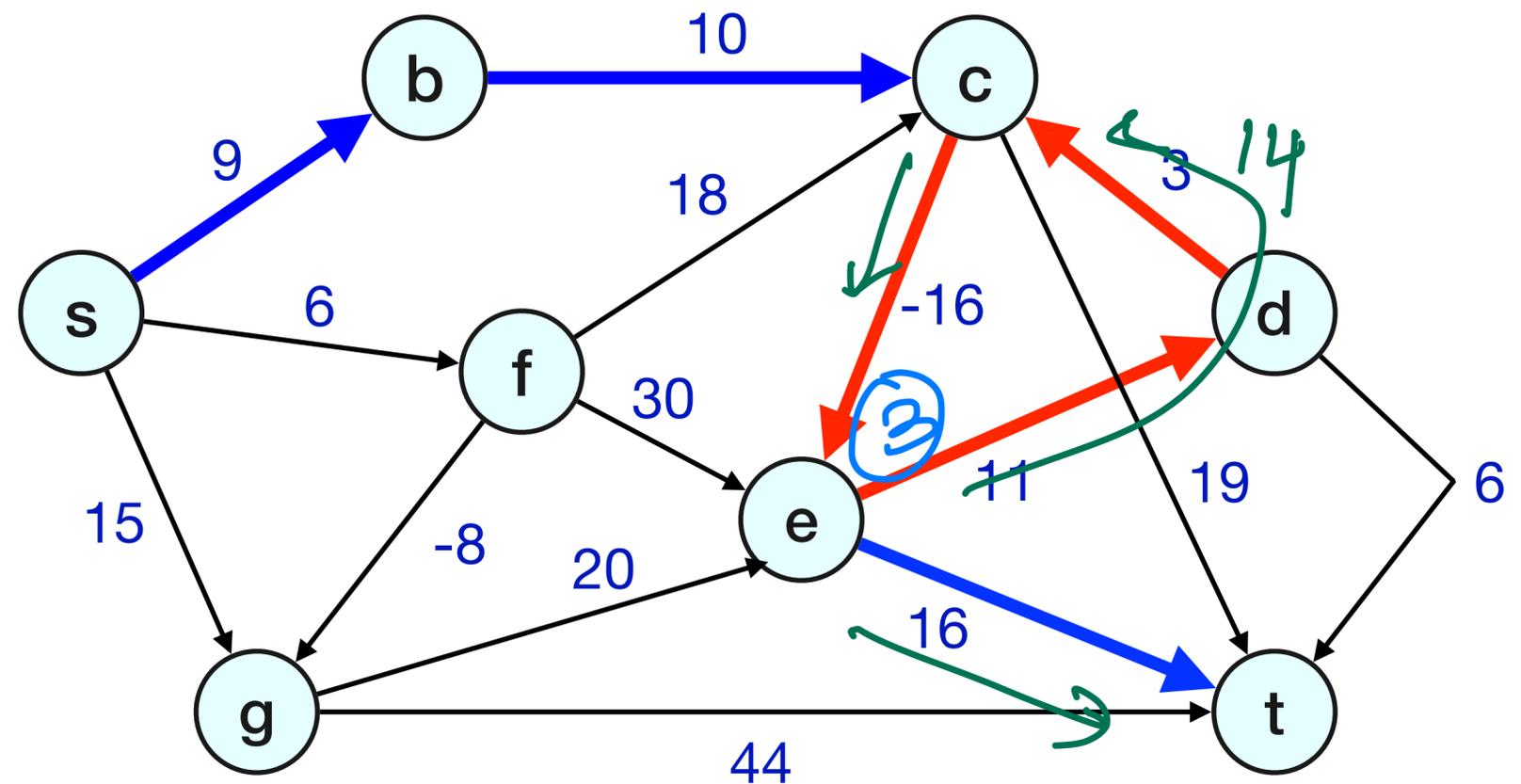


Negative length cycles

Definition

What is the shortest path distance between s and t ?

What about adding red path?



Negative length cycles

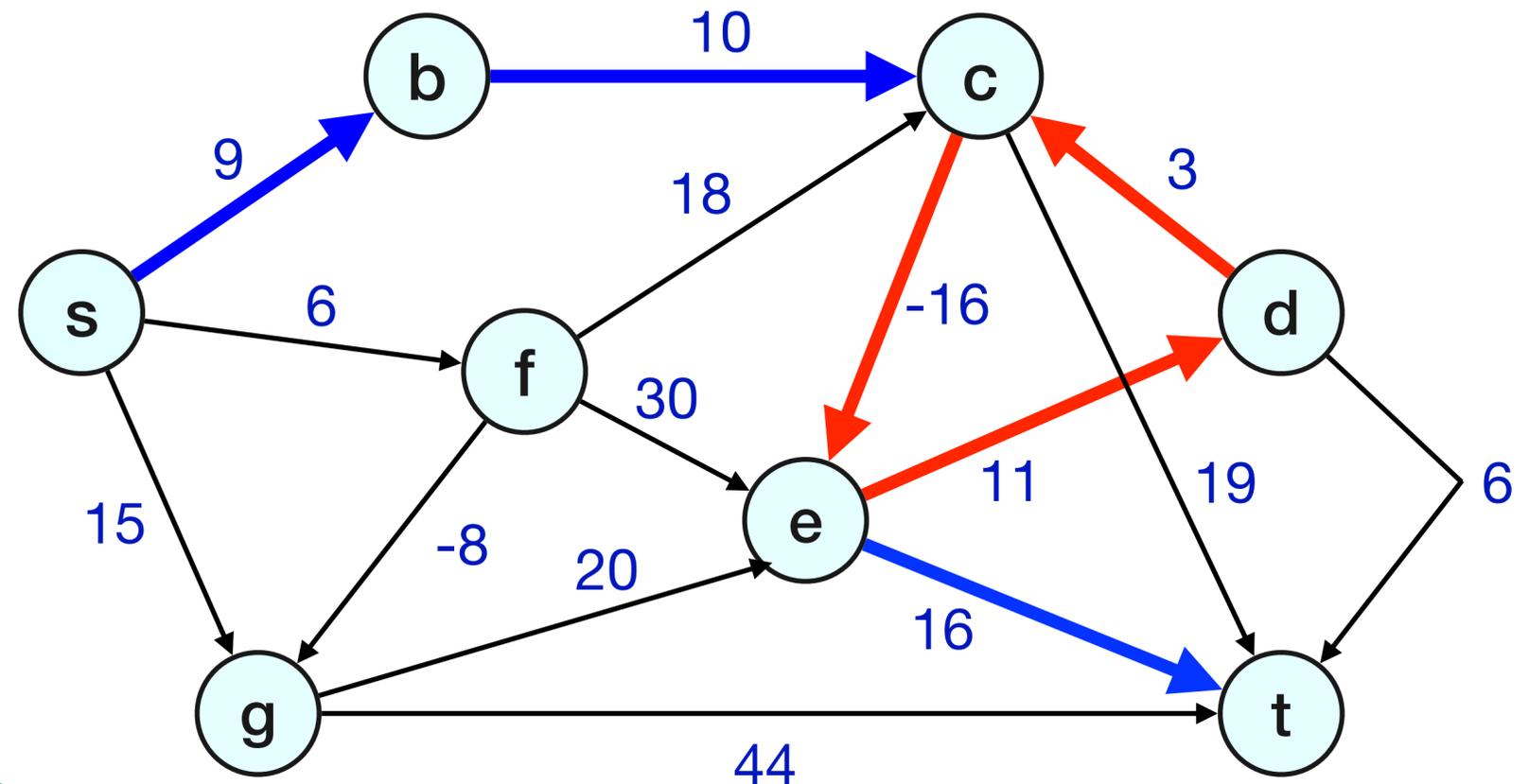
Definition

What is the shortest path distance between s and t ?

What about adding red path?

Fix: Restrict paths to be *simple* ...

no repeated



Negative length cycles

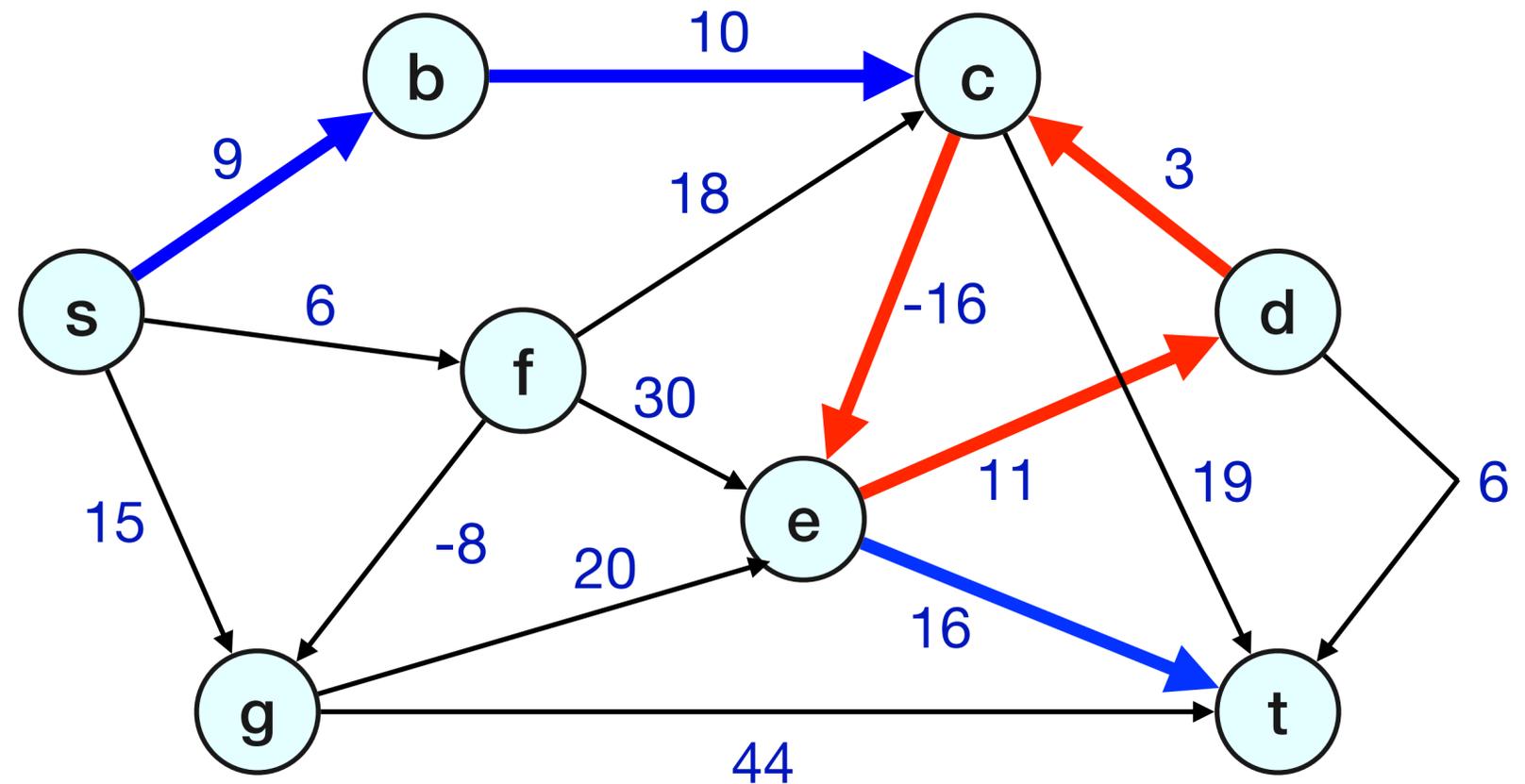
Definition

A cycle C is a negative length cycle if the sum of the edge lengths of C is negative.

What is the shortest path distance between s and t ?

What about adding red path?

Fix: Restrict paths to be *simple* ...



Shortest paths and negative cycles

Given $G = (V, E)$ with edge lengths and s, t . Suppose

- G has a negative length cycle C , and s can reach C and C can reach t .

Question: What is the shortest distance from s to t ?

Shortest paths and negative cycles

Given $G = (V, E)$ with edge lengths and s, t . Suppose

- G has a negative length cycle C , and s can reach C and C can reach t .

Question: What is the shortest distance from s to t ?

Possible answers:

Shortest paths and negative cycles

Given $G = (V, E)$ with edge lengths and s, t . Suppose

- G has a negative length cycle C , and s can reach C and C can reach t .

Question: What is the shortest distance from s to t ?

Possible answers:

- undefined, that is $-\infty$, OR

Shortest paths and negative cycles

Given $G = (V, E)$ with edge lengths and s, t . Suppose

- G has a negative length cycle C , and s can reach C and C can reach t .

Question: What is the shortest distance from s to t ?

Possible answers:

- undefined, that is $-\infty$, OR 
- the length of a shortest simple path from s to t . 

Restating problem of shortest path with negative edges

Alternatively: Finding shortest walks

Recall that given a graph $G = (V, E)$:

- A (simple) path is a sequence of distinct vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
- A walk is a sequence of vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.

Restating problem of shortest path with negative edges

Alternatively: Finding shortest walks

Recall that given a graph $G = (V, E)$:

- A (simple) **path** is a sequence of distinct vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
- A **walk** is a sequence of vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.

Define $\text{dist}(u, v)$ to be the length of a shortest walk from u to v .

Restating problem of shortest path with negative edges

Alternatively: Finding shortest walks

Recall that given a graph $G = (V, E)$:

- A (simple) **path** is a sequence of distinct vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
- A **walk** is a sequence of vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.

Define $\text{dist}(u, v)$ to be the length of a shortest walk from u to v .

- If there is a walk from u to v that contains negative length cycle then $\text{dist}(u, v) = -\infty$ 

Restating problem of shortest path with negative edges

Alternatively: Finding shortest walks

Recall that given a graph $G = (V, E)$:

- A (simple) **path** is a sequence of distinct vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
- A **walk** is a sequence of vertices v_1, v_2, \dots, v_k such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.

Define $\text{dist}(u, v)$ to be the length of a shortest walk from u to v .

- If there is a walk from u to v that contains negative length cycle then $\text{dist}(u, v) = -\infty$
- Else, there is a path with at most $n - 1$ edges whose length is equal to the length of a shortest walk and $\text{dist}(u, v)$ is finite

Shortest paths with negative edges

Algorithmic problems

Input: A directed graph $G = (V, E)$ with edge lengths (could be negative). For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

Questions:

Shortest paths with negative edges

Algorithmic problems

Input: A directed graph $G = (V, E)$ with edge lengths (could be negative). For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

Questions:

- Given nodes s, t either find a negative length cycle C that s can reach or find a shortest path from s to t .

directed

reachable

Shortest paths with negative edges

Algorithmic problems

Input: A directed graph $G = (V, E)$ with edge lengths (could be negative). For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

Questions:

- Given nodes s, t either find a negative length cycle C that s can reach or find a shortest path from s to t . 
- Given node s , either find a negative length cycle C that s can reach or find shortest path distances from s to all reachable nodes. 

Shortest paths with negative edges

Algorithmic problems

Input: A directed graph $G = (V, E)$ with edge lengths (could be negative). For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

Questions:

- Given nodes s, t either find a negative length cycle C that s can reach or find a shortest path from s to t .
- Given node s , either find a negative length cycle C that s can reach or find shortest path distances from s to all reachable nodes.
- Check if G has a negative length cycle or not.

→ tell parent shortest G
path → Floyd
Warshall.

Bellman Ford Algorithm

Shortest paths and recursion

- Is it possible to compute the shortest path distance from s to t recursively?
- If yes, what are the smaller sub-problems?

Shortest paths and recursion

- Is it possible to compute the shortest path distance from s to t *recursively*?
- If yes, what are the *smaller* sub-problems?

Lemma: Let G be a directed graph with arbitrary edge lengths. If

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$$

Shortest paths and recursion

- Is it possible to compute the shortest path distance from s to t *recursively*?
- If yes, what are the *smaller* sub-problems?

Lemma: Let G be a directed graph with arbitrary edge lengths. If

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$$

is a shortest path from s to v_k then for $1 \leq i < k$:

Shortest paths and recursion

- Is it possible to compute the shortest path distance from s to t *recursively*?
- If yes, what are the *smaller* sub-problems?

Lemma: Let G be a directed graph with arbitrary edge lengths. If

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$$

is a shortest path from s to v_k then for $1 \leq i < k$:

$s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_i$ is a shortest path from s to v_i

Shortest paths and recursion

- Is it possible to compute the shortest path distance from s to t *recursively*?
- If yes, what are the *smaller* sub-problems?

Lemma: Let G be a directed graph with arbitrary edge lengths. If

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$$

is a shortest path from s to v_k then for $1 \leq i < k$:

$$s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_i \text{ is a shortest path from } s \text{ to } v_i$$

Sub-problem idea: paths of *fewer hops/edges*

Hop-based recursion

Bellman-Ford Algorithm

Single-source problem: Fix source s .

Assumptions: All nodes can be reached from s in G . Assume G has no negative-length cycle (for now).

Hop-based recursion

Bellman-Ford Algorithm

Single-source problem: Fix source s .

Assumptions: All nodes can be reached from s in G . Assume G has no negative-length cycle (for now).

Define, $d(v, k)$ as the shortest *walk* length from s to v using at most k edges.

Then note, $\text{dist}(s, v) = d(v, n - 1)$. Recursion for $d(v, k)$:

\uparrow \uparrow
different.

Hop-based recursion

Bellman-Ford Algorithm

Single-source problem: Fix source s .

Assumptions: All nodes can be reached from s in G . Assume G has no negative-length cycle (for now).

Define, $d(v, k)$ as the shortest *walk* length from s to v using at most k edges.

Then note, $\text{dist}(s, v) = d(v, n - 1)$. Recursion for $d(v, k)$:

$$d(v, k) = \min \begin{cases} \min_{u \in V} (d(u, k - 1) + l(u, v)) \\ d(v, k - 1) \end{cases}$$

Handwritten annotations:
A blue circle around $d(u, k - 1)$ in the first term of the min, with an arrow pointing to the text "intermediate node u ".
A green arrow points to $d(v, k - 1)$ in the second term of the min.
A green arrow points to $d(v, k)$ in the left-hand side of the equation.

Hop-based recursion

Bellman-Ford Algorithm

Single-source problem: Fix source s .

Assumptions: All nodes can be reached from s in G . Assume G has no negative-length cycle (for now).

Define, $d(v, k)$ as the shortest *walk* length from s to v using at most k edges. Then note, $\text{dist}(s, v) = d(v, n - 1)$. Recursion for $d(v, k)$:

$$d(v, k) = \min \begin{cases} \min_{u \in V} (d(u, k - 1) + l(u, v)) \\ d(v, k - 1) \end{cases}$$

Base case: $d(s, 0) = 0$ and $d(v, 0) = \infty$ for all $v \neq s$

Bellman-Ford Algorithm

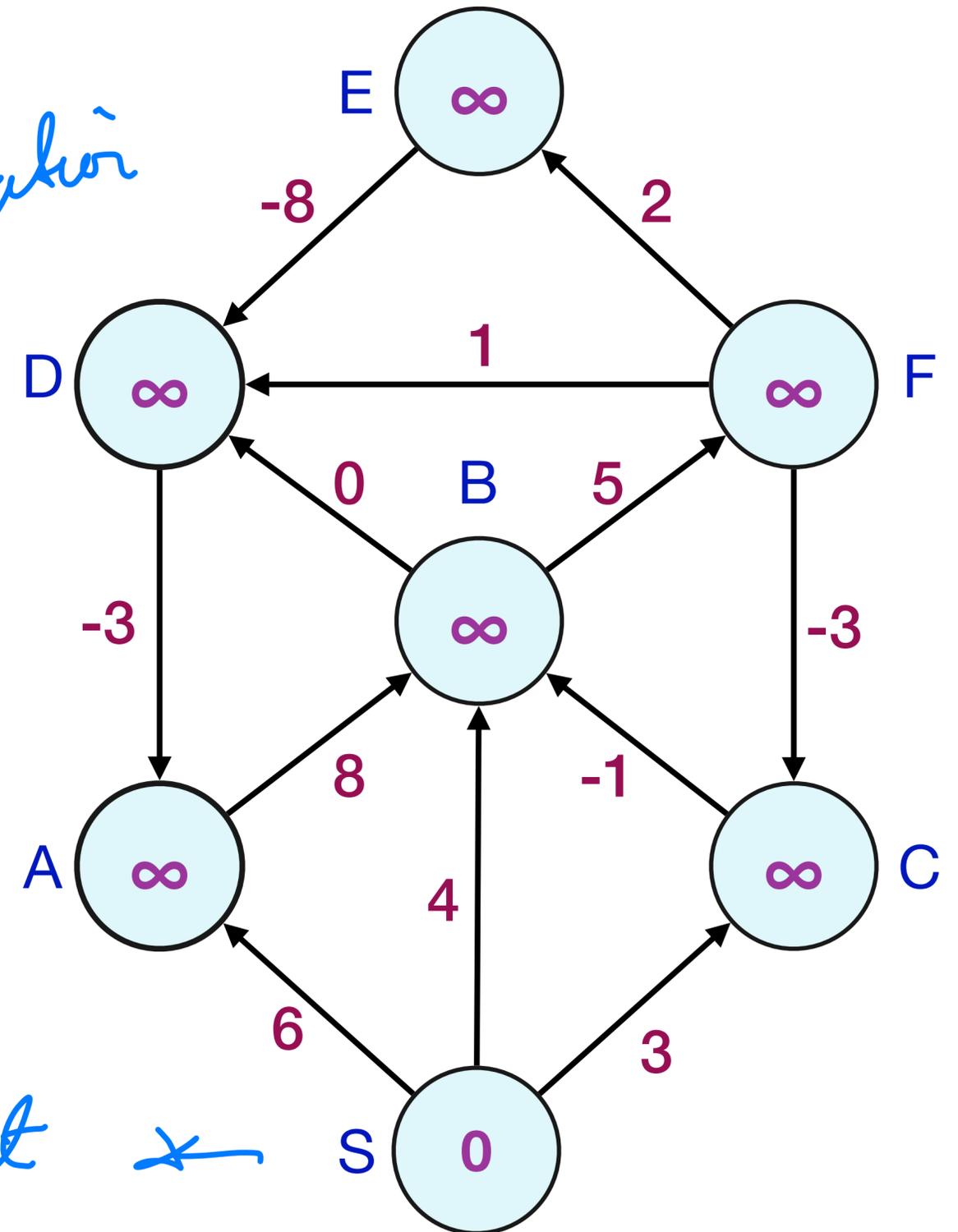
Example

Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1							
2							
3							
4							
5							
6							

Initialization



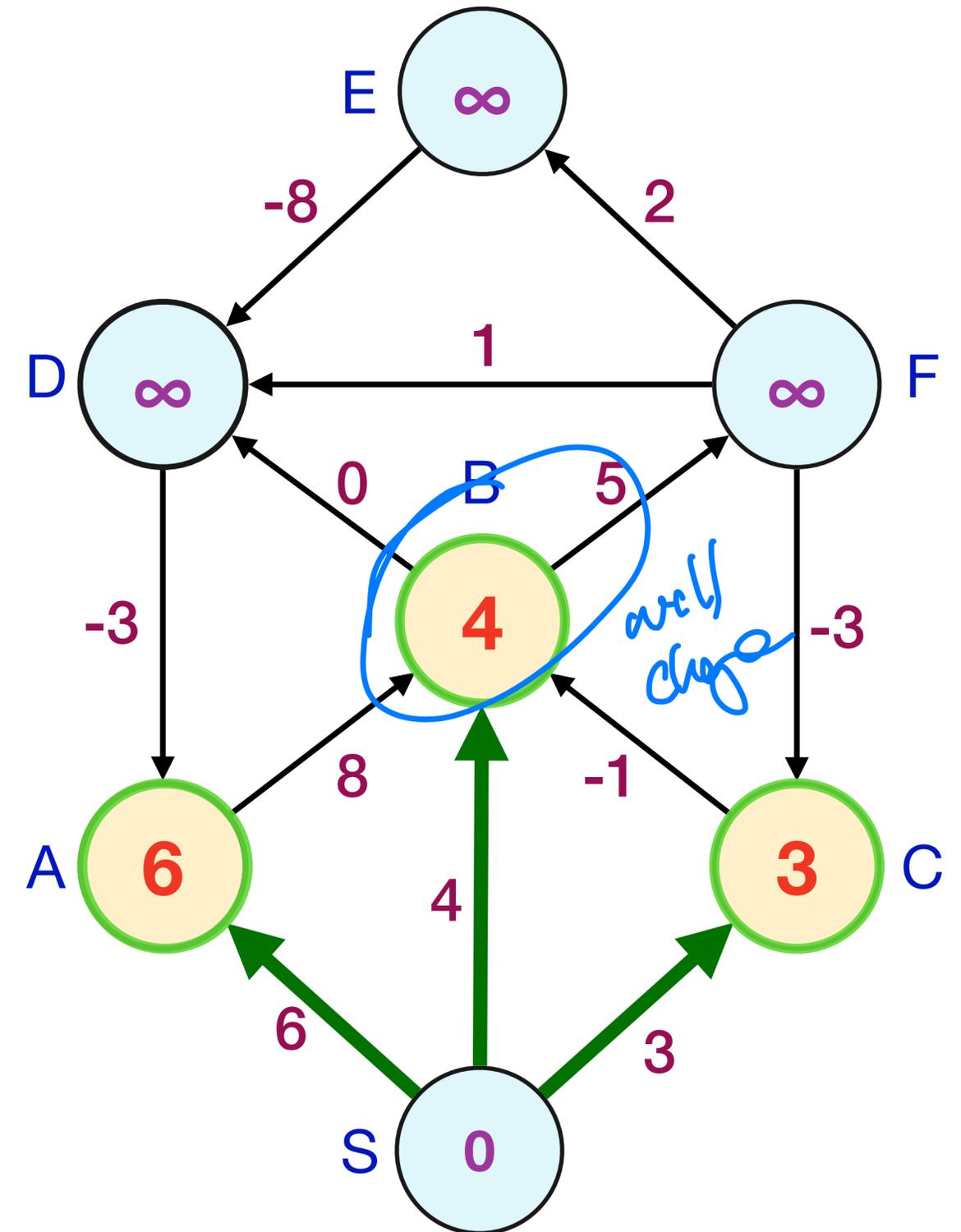
Start



Bellman-Ford Algorithm

Example

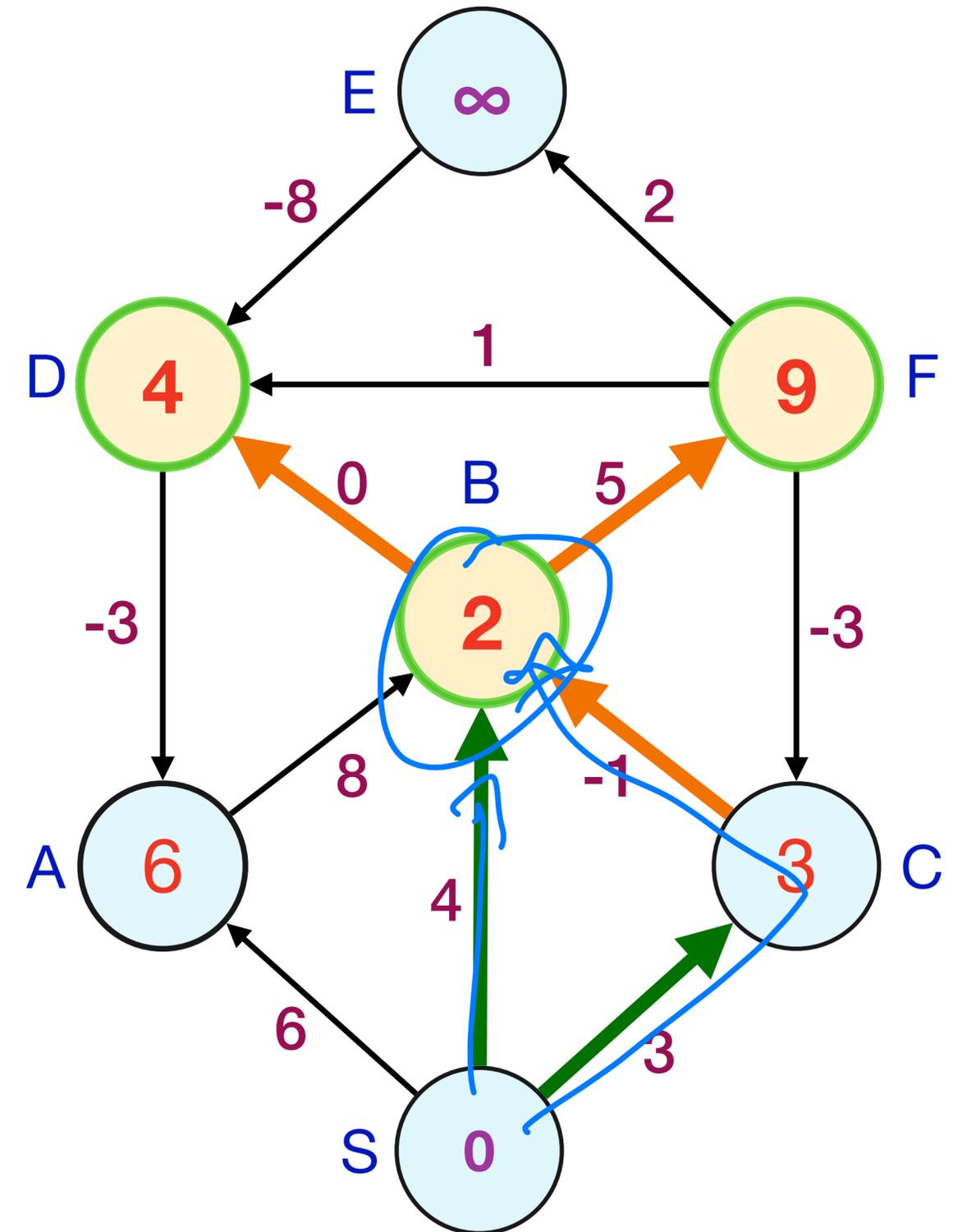
Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2							
3							
4							
5							
6							



Bellman-Ford Algorithm

Example

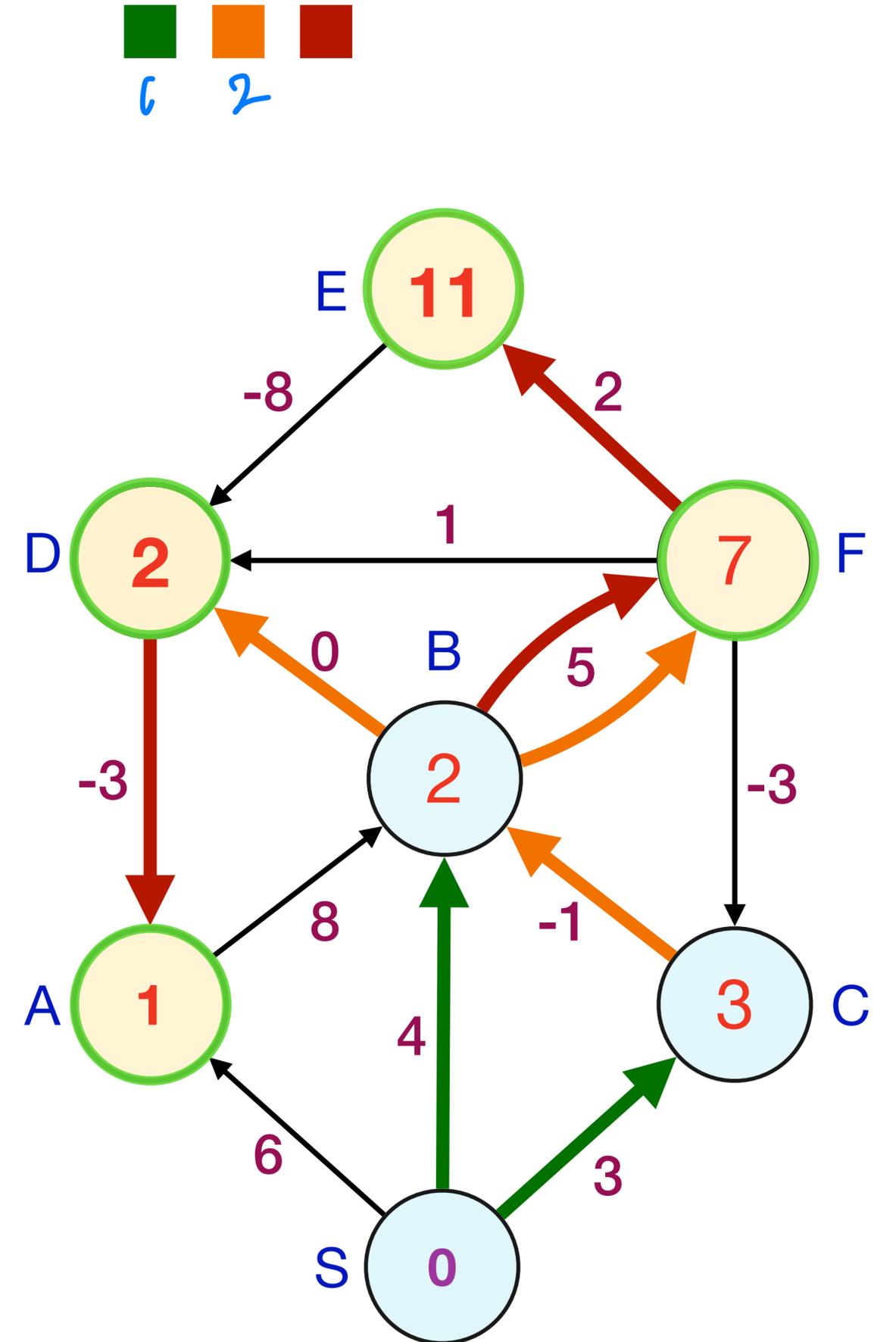
Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3							
4							
5							
6							



Bellman-Ford Algorithm

Example

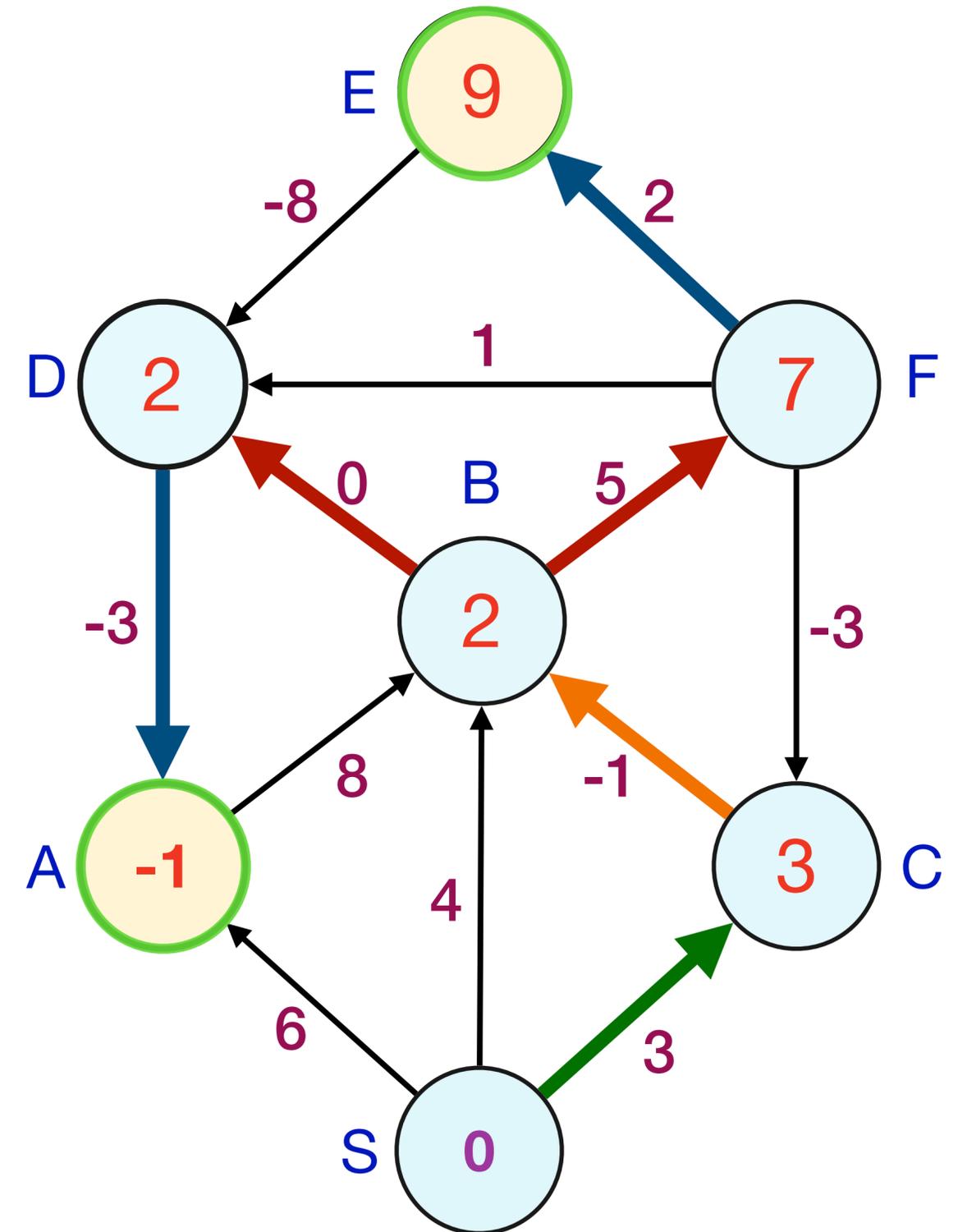
Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4							
5							
6							



Bellman-Ford Algorithm

Example

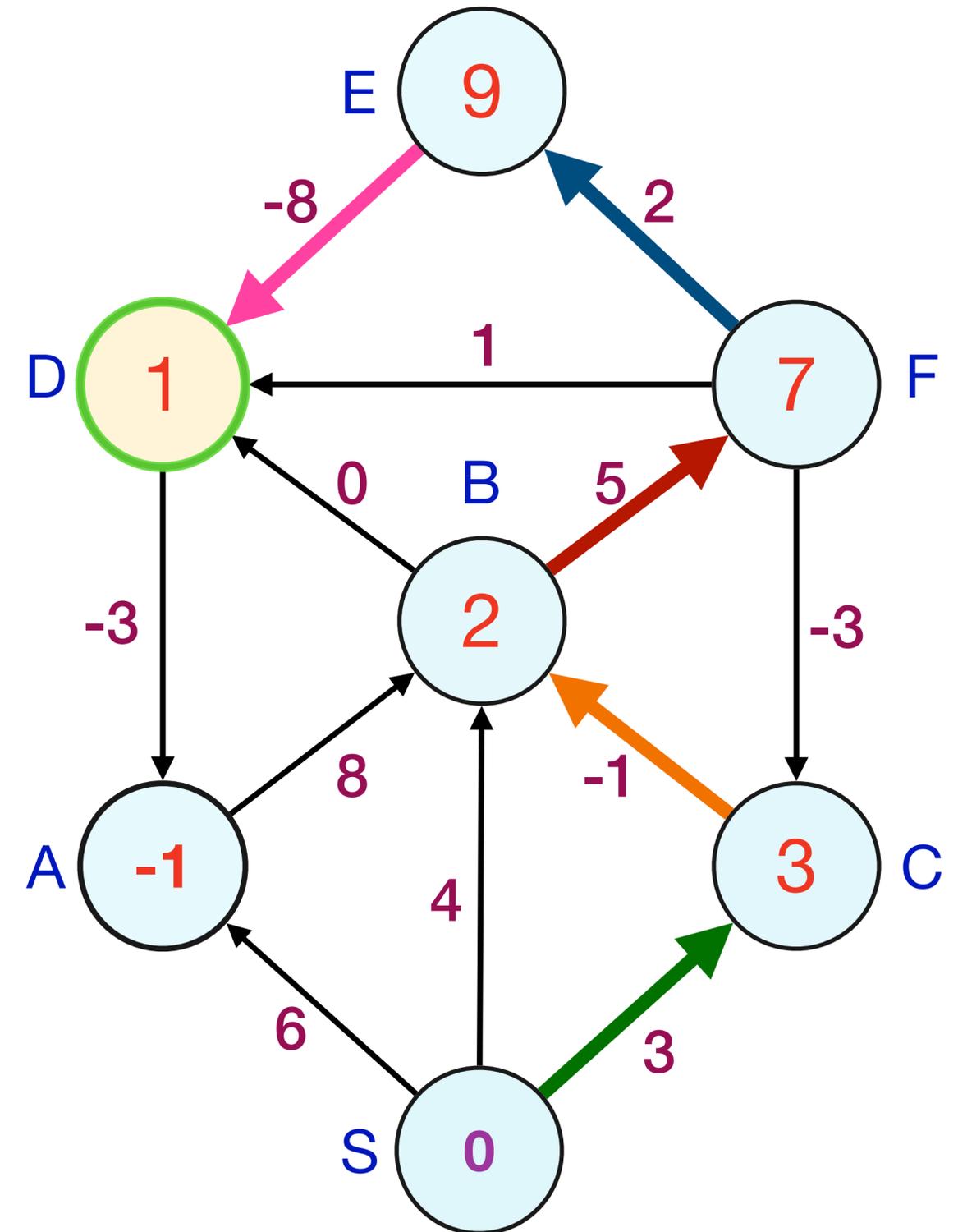
Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5							
6							



Bellman-Ford Algorithm

Example

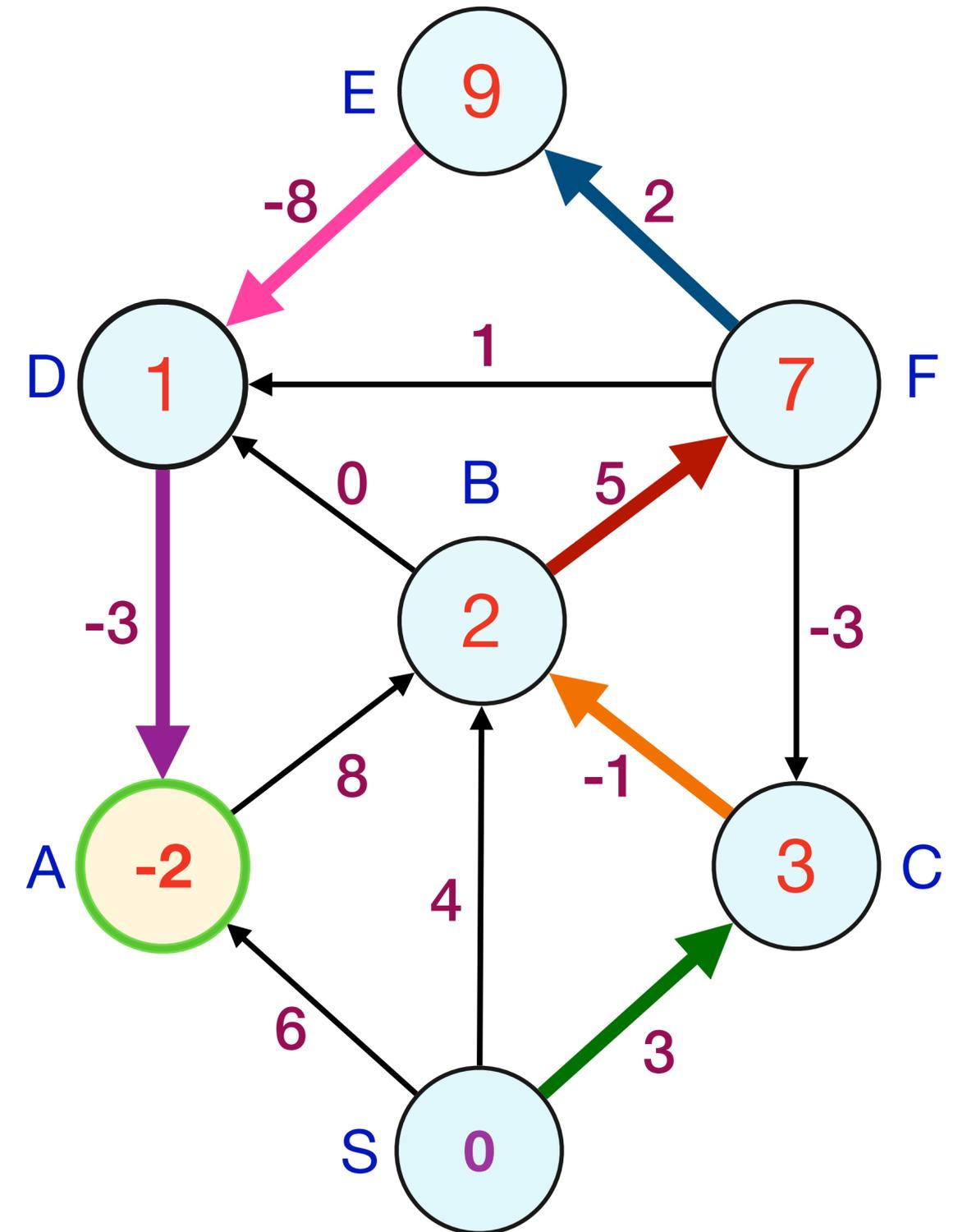
Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5	0	-1	2	3	1	9	7
6							



Bellman-Ford Algorithm

Example

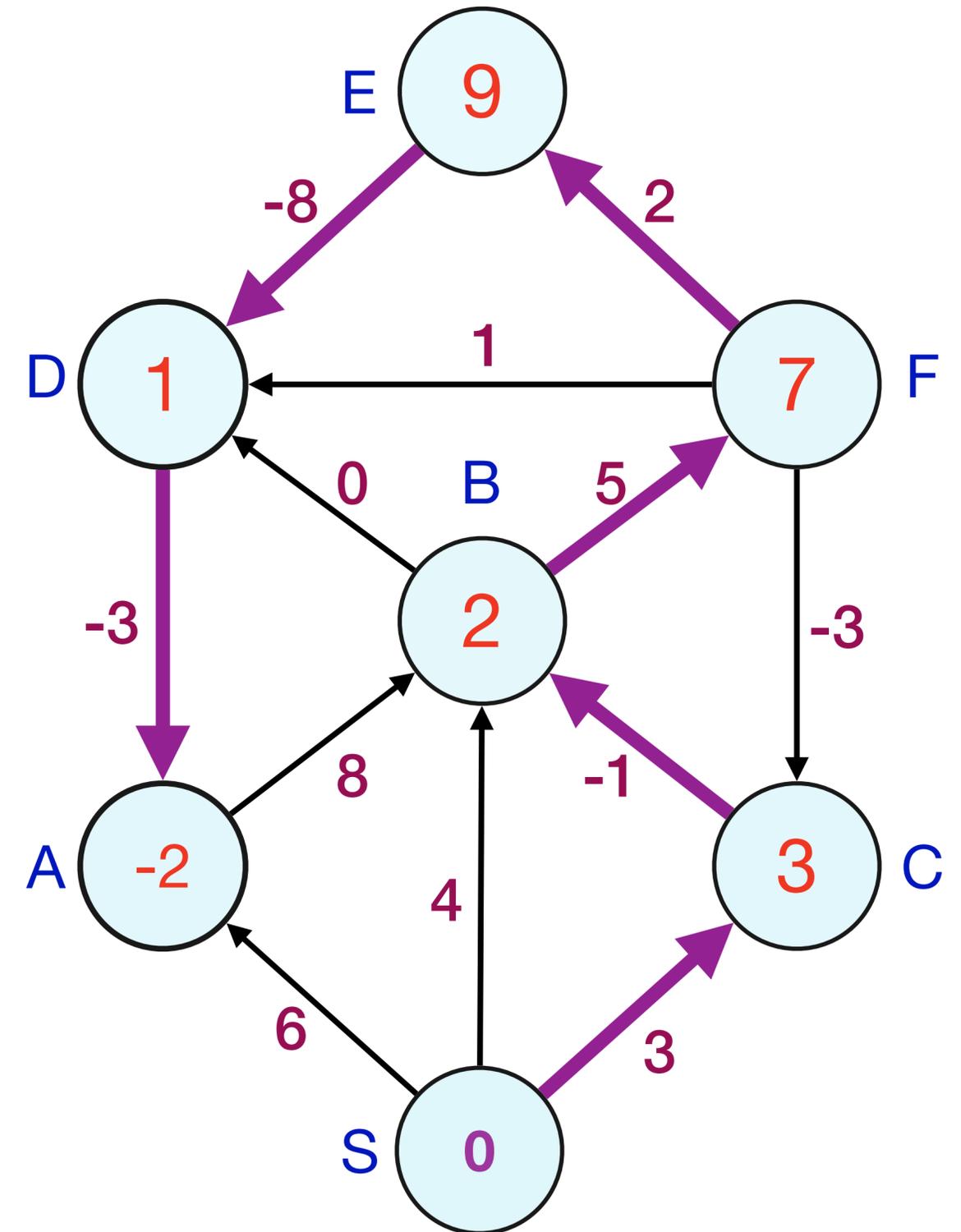
Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5	0	-1	2	3	1	9	7
6	0	-2	2	3	1	9	7



Bellman-Ford Algorithm

Example

Round	S	A	B	C	D	E	F
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5	0	-1	2	3	1	9	7
6	0	-2	2	3	1	9	7



Bellman-Ford Algorithm

Algorithm

Create $In(G)$ list from $adj(G)$

← creating a list at the beginning

for each $u \in V$ **do**

$d(u,0) \leftarrow \infty$

$d(s,0) \leftarrow 0$

for $k = 1$ to $n - 1$ **do**

for each $v \in V$ **do**

$d(v, k) \leftarrow d(v, k - 1)$

for each edge $(u,v) \in In(v)$ **do**

$d(v, k) = \min\{d(v, k), d(u, k - 1) + l(u, v)\}$

→ copying values from last round.

for each $v \in V$ **do**

$\text{dist}(s, v) \leftarrow d(v, n - 1)$

Running time: $O(n(n + m))$

Space: $O(m + n^2)$

Bellman-Ford Algorithm

Algorithm

Create $\text{In}(G)$ list from $\text{adj}(G)$

for each $u \in V$ **do**

$d(u,0) \leftarrow \infty$

$d(s,0) \leftarrow 0$

for $k = 1$ to $n - 1$ **do**

for each $v \in V$ **do**

$d(v, k) \leftarrow d(v, k - 1)$

for each edge $(u,v) \in \text{In}(v)$ **do**

$d(v, k) = \min\{d(v, k), d(u, k - 1) + l(u, v)\}$

for each $v \in V$ **do**

$\text{dist}(s, v) \leftarrow d(v, n - 1)$

Running time: $O(n(n + m))$

Space: $O(m + n^2)$

Space can be reduced to
 $O(m + n)$

Bellman-Ford Algorithm

Algorithm

Create $\text{In}(G)$ list from $\text{adj}(G)$ 

for each $u \in V$ **do**

$d(u,0) \leftarrow \infty$

$d(s,0) \leftarrow 0$

for $k = 1$ to $n - 1$ **do**

for each $v \in V$ **do**

$d(v, k) \leftarrow d(v, k - 1)$ 

for each edge $(u,v) \in \text{In}(v)$ **do**

$d(v, k) = \min\{d(v, k), d(u, k - 1) + l(u, v)\}$

for each $v \in V$ **do**

$\text{dist}(s, v) \leftarrow d(v, n - 1)$

Do we need the $\text{In}(G)$ list?

Running time: $O(n(n + m))$

Space: $O(m + n^2)$

Space can be reduced to
 $O(m + n)$

Bellman-Ford Algorithm

Algorithm - cleaner version

Do we need the $\text{In}(G)$ list?

for each $u \in V$ **do**

$d(u,0) \leftarrow \infty$

$d(s,0) \leftarrow 0$

for $k = 1$ to $n - 1$ **do**

for each $v \in V$ **do**

 **for** each edge $(u,v) \in \text{In}(v)$ **do**

$d(v) = \min\{d(v), d(u) + l(u, v)\}$

for each $v \in V$ **do**

$\text{dist}(s, v) \leftarrow d(v, n - 1)$

Bellman-Ford Algorithm

Algorithm - cleaner version

```
for each  $u \in V$  do  
     $d(u,0) \leftarrow \infty$   
 $d(s,0) \leftarrow 0$   
  
for  $k = 1$  to  $n - 1$  do  
    for each  $v \in V$  do  
        for each edge  $(u,v) \in \text{In}(v)$  do  
             $d(v) = \min\{d(v), d(u) + l(u, v)\}$   
  
for each  $v \in V$  do  
     $\text{dist}(s, v) \leftarrow d(v, n - 1)$ 
```

Do we need the $\text{In}(G)$ list?

Running time: $O(mn)$

Space: $O(m + n)$

Bellman-Ford Algorithm

Algorithm - cleaner version

```
for each  $u \in V$  do  
     $d(u,0) \leftarrow \infty$   
 $d(s,0) \leftarrow 0$   
  
for  $k = 1$  to  $n - 1$  do  
    for each  $v \in V$  do  
        for each edge  $(u, v) \in \text{In}(v)$  do  
             $d(v) = \min\{d(v), d(u) + l(u, v)\}$   
  
for each  $v \in V$  do  
     $\text{dist}(s, v) \leftarrow d(v, n - 1)$ 
```

Do we need the $\text{In}(v)$ list?

Running time: $O(mn)$

Space: $O(m + n)$

Bellman-Ford Algorithm

Algorithm - optimized

```
for each  $u \in V$  do  
     $d(u,0) \leftarrow \infty$   
 $d(s,0) \leftarrow 0$   
  
for  $k = 1$  to  $n - 1$  do  
    for each edge  $(u,v) \in G$  do  
         $d(v) = \min\{d(v), d(u) + l(u, v)\}$   
  
for each  $v \in V$  do  
     $\text{dist}(s, v) \leftarrow d(v, n - 1)$ 
```

Do we need the $\text{In}(v)$ list?

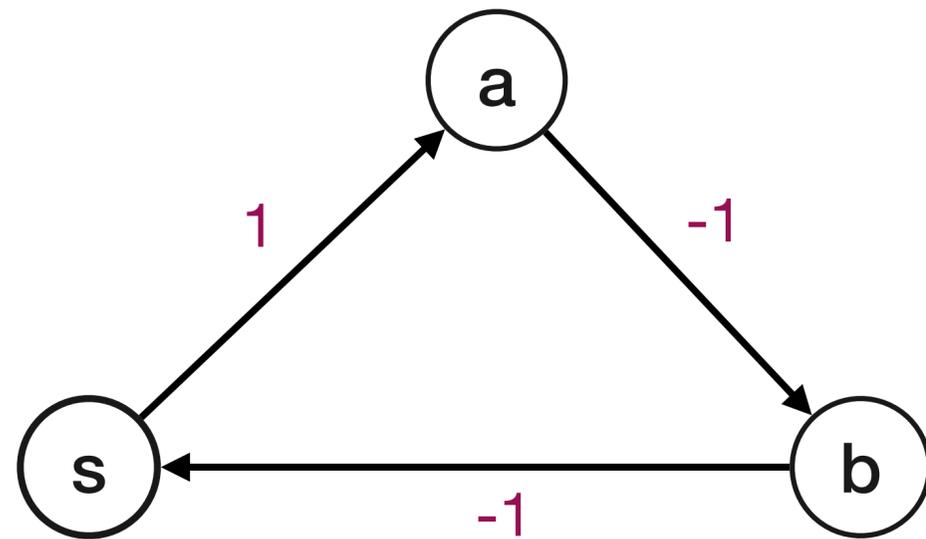
Running time: $O(mn)$

Space: $O(n)$ ←

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?

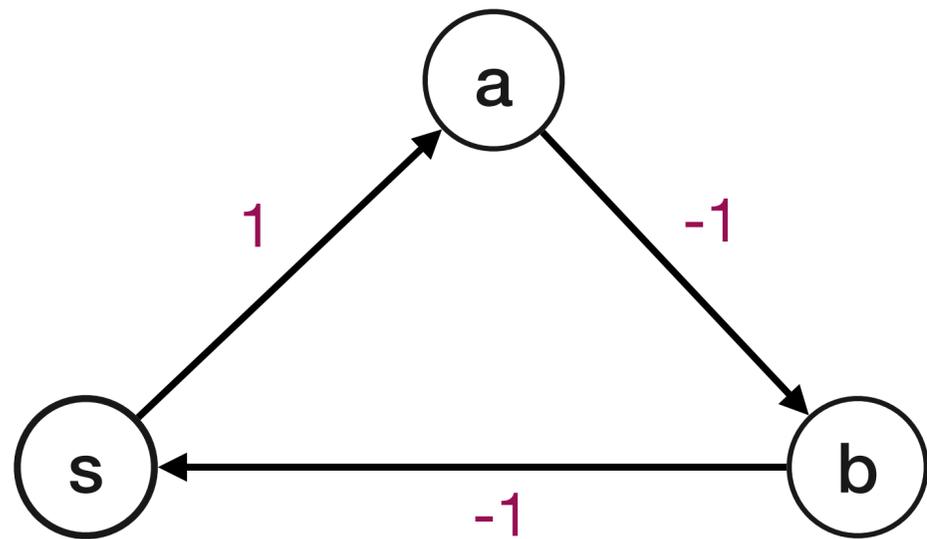


Round	s	a	b

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?

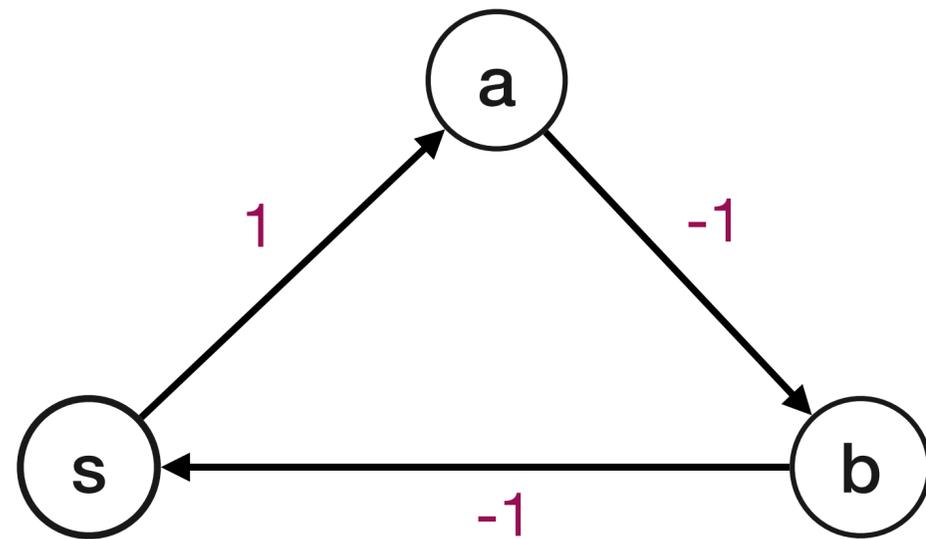


Round	s	a	b
0	0	∞	∞

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?

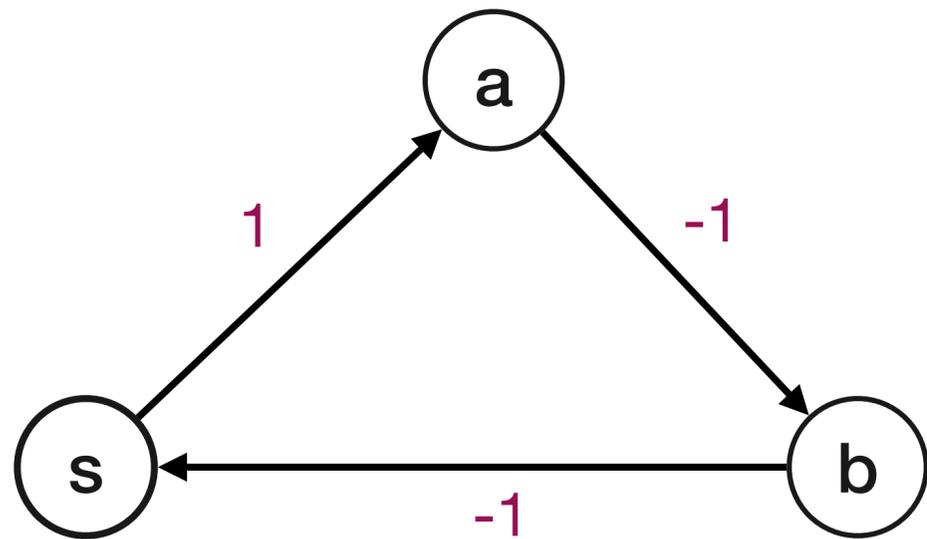


Round	s	a	b
0	0	∞	∞
1	0	1	∞

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?

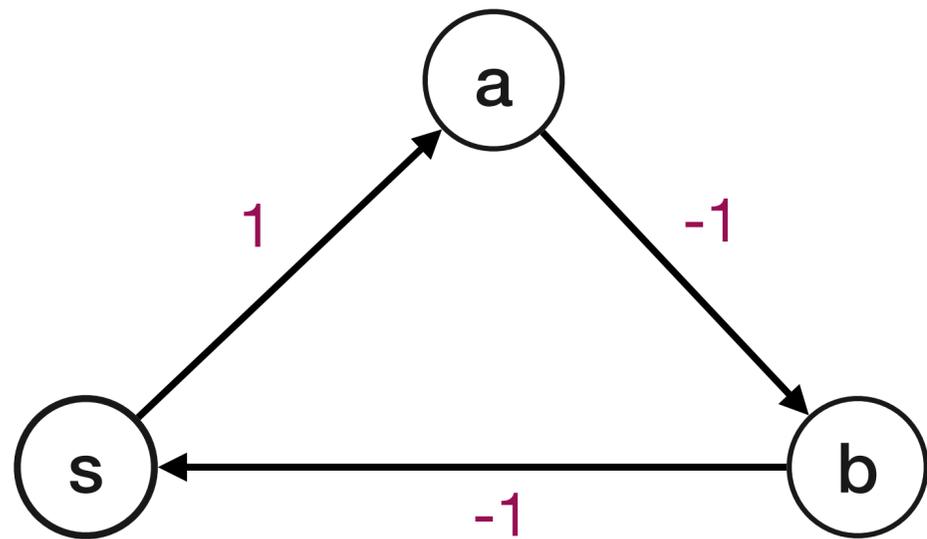


Round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?

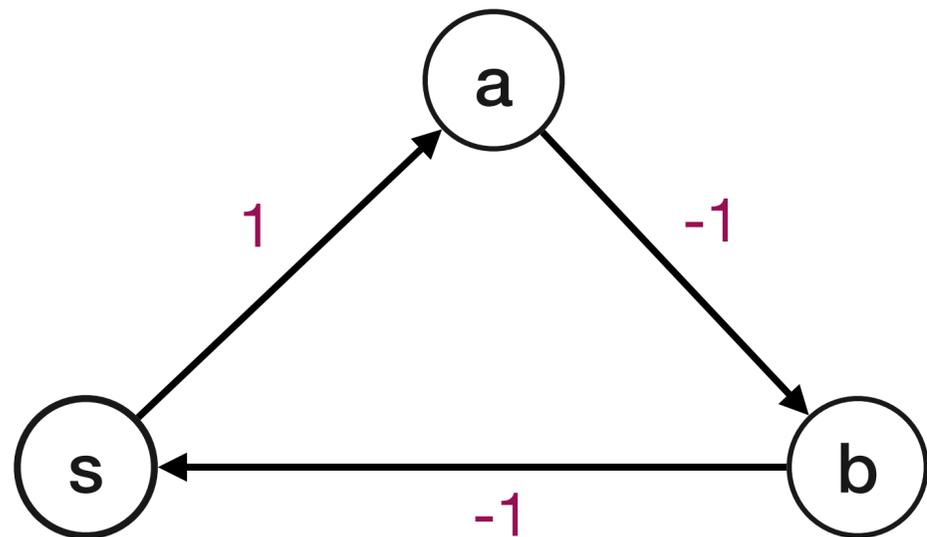


Round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0
3	-1	1	0

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?

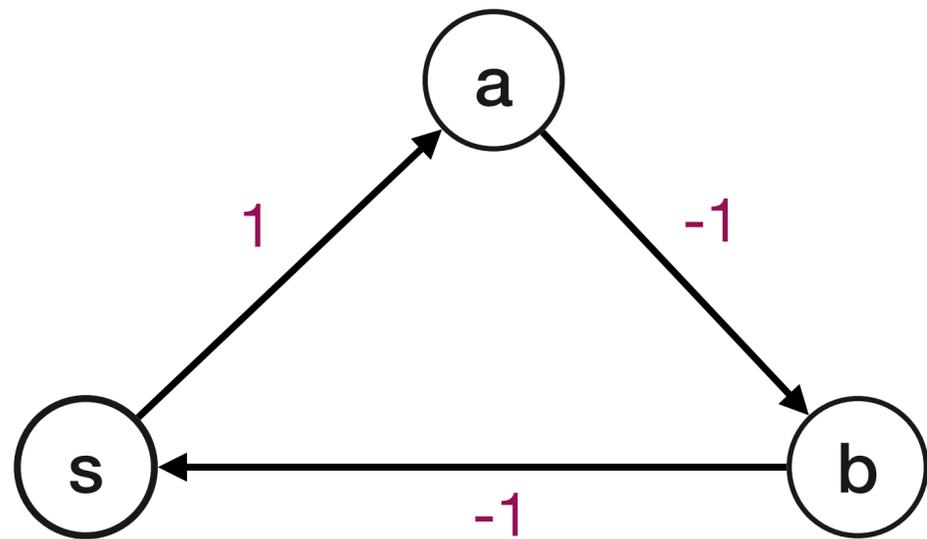


Round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0
3	-1	1	0
4	-1	0	0

Bellman-Ford Algorithm

Negative cycles

What happens if we run this on a graph with negative cycles?



Round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0
3	-1	1	0
4	-1	0	0
5	-1	0	-1

-2

Correctness: detecting negative length cycle

Lemma: Suppose G has a negative cycle C reachable from s . Then there is some node $v \in C$ such that $d(v, n) < d(v, n - 1)$.

Correctness: detecting negative length cycle

Lemma: Suppose G has a negative cycle C reachable from s . Then there is some node $v \in C$ such that $d(v, n) < d(v, n - 1)$. *so assume $d(v, n-1) \leq d(v, n)$*

Proof: Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ be negative length cycle reachable from s . Then $d(v_i, n - 1)$ is finite for $1 \leq i \leq k$ since C is reachable from s .

Correctness: detecting negative length cycle

Lemma: Suppose G has a negative cycle C reachable from s . Then there is some node $v \in C$ such that $d(v, n) < d(v, n - 1)$.

Proof: Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ be negative length cycle reachable from s . Then $d(v_i, n - 1)$ is finite for $1 \leq i \leq k$ since C is reachable from s .

By assumption $d(v, n) \geq d(v, n - 1)$ for all $v \in C$; implies no change in n^{th} iteration; $d(v_i, n - 1) = d(v_i, n)$ for $1 \leq i \leq k$.

Correctness: detecting negative length cycle

Lemma: Suppose G has a negative cycle C reachable from s . Then there is some node $v \in C$ such that $d(v, n) < d(v, n - 1)$.

Proof: Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ be negative length cycle reachable from s . Then $d(v_i, n - 1)$ is finite for $1 \leq i \leq k$ since C is reachable from s .

By assumption $d(v, n) \geq d(v, n - 1)$ for all $v \in C$; implies no change in n^{th} iteration; $d(v_i, n - 1) = d(v_i, n)$ for $1 \leq i \leq k$.

This means $d(v_i, n - 1) \leq d(v_{i-1}, n - 1) + l(v_{i-1}, v_i)$ for $2 \leq i \leq k$ and $d(v_1, n - 1) \leq d(v_k, n - 1) + l(v_k, v_1)$.

Correctness: detecting negative length cycle

Lemma: Suppose G has a negative cycle C reachable from s . Then there is some node $v \in C$ such that $d(v, n) < d(v, n - 1)$.

Proof: Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$ be negative length cycle reachable from s . Then $d(v_i, n - 1)$ is finite for $1 \leq i \leq k$ since C is reachable from s .

By assumption $d(v, n) \geq d(v, n - 1)$ for all $v \in C$; implies no change in n^{th} iteration; $d(v_i, n - 1) = d(v_i, n)$ for $1 \leq i \leq k$.

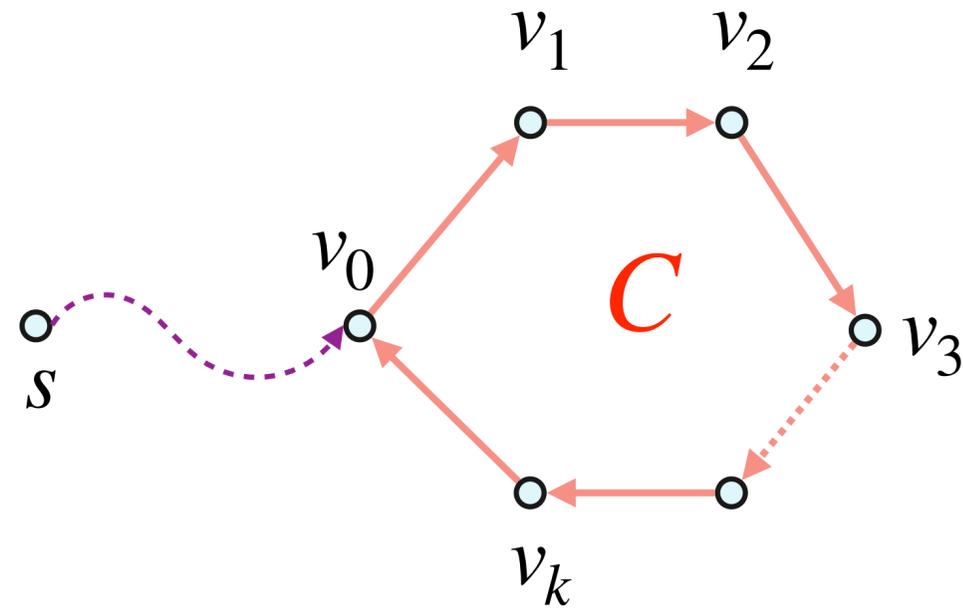
This means $d(v_i, n - 1) \leq d(v_{i-1}, n - 1) + l(v_{i-1}, v_i)$ for $2 \leq i \leq k$ and $d(v_1, n - 1) \leq d(v_k, n - 1) + l(v_k, v_1)$.

Summing/telescoping these inequalities results in $0 \leq l(C)$ which contradicts the assumption that $l(C) < 0$!

For proof by contradiction

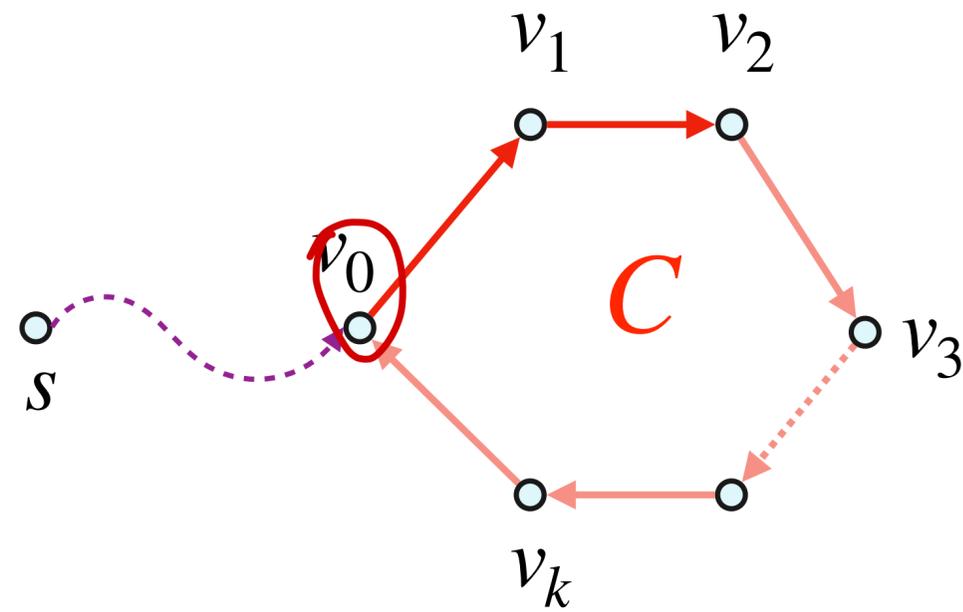
Assumed that for all v , $d(v, n) \geq d(v, n - 1)$

Proof of lemma ...



Assumed that for all v , $d(v, n) \geq d(v, n - 1)$

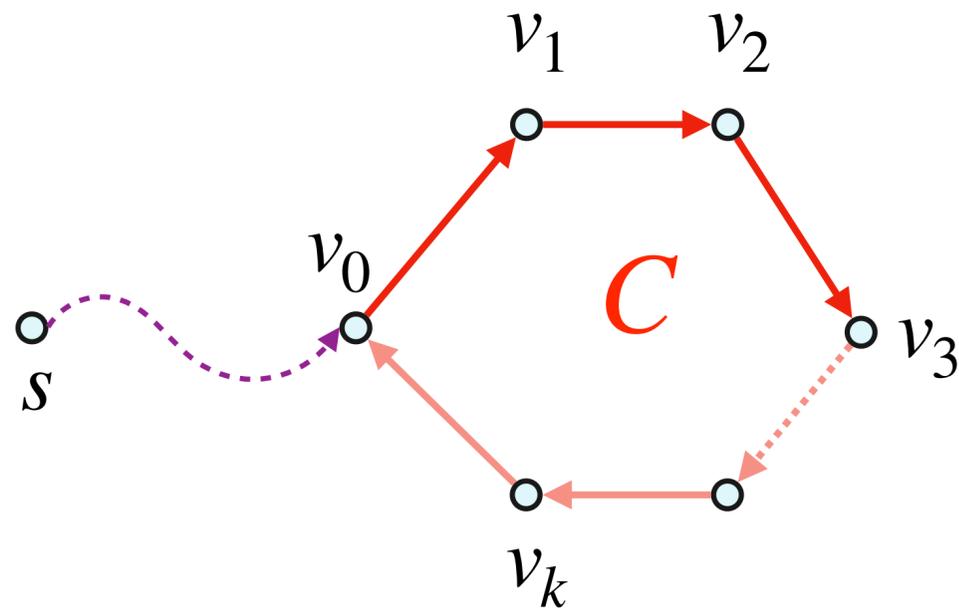
Proof of lemma ...



$$d(v_1, n - 1) \leq d(v_0, n - 1) + \underline{l(v_0, v_1)}$$

Assumed that for all v , $d(v, n) \geq d(v, n - 1)$

Proof of lemma ...

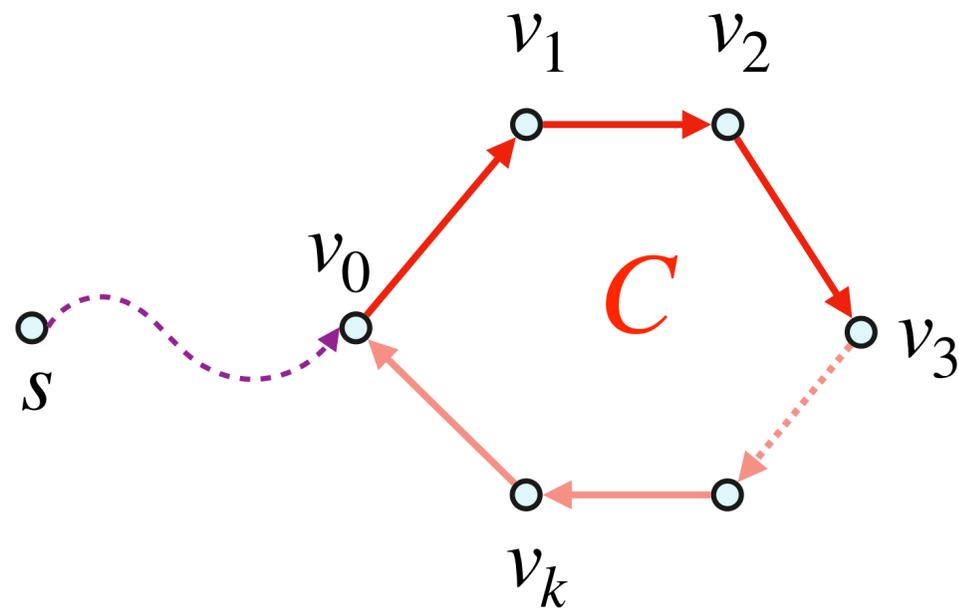


$$d(v_1, n - 1) \leq d(v_0, n - 1) + l(v_0, v_1)$$

$$\underline{d(v_2, n - 1)} \leq d(v_1, n - 1) + l(v_1, v_2)$$

Assumed that for all v , $d(v, n) \geq d(v, n - 1)$

Proof of lemma ...



$$d(v_1, n - 1) \leq d(v_0, n - 1) + l(v_0, v_1)$$

$$d(v_2, n - 1) \leq d(v_1, n - 1) + l(v_1, v_2)$$

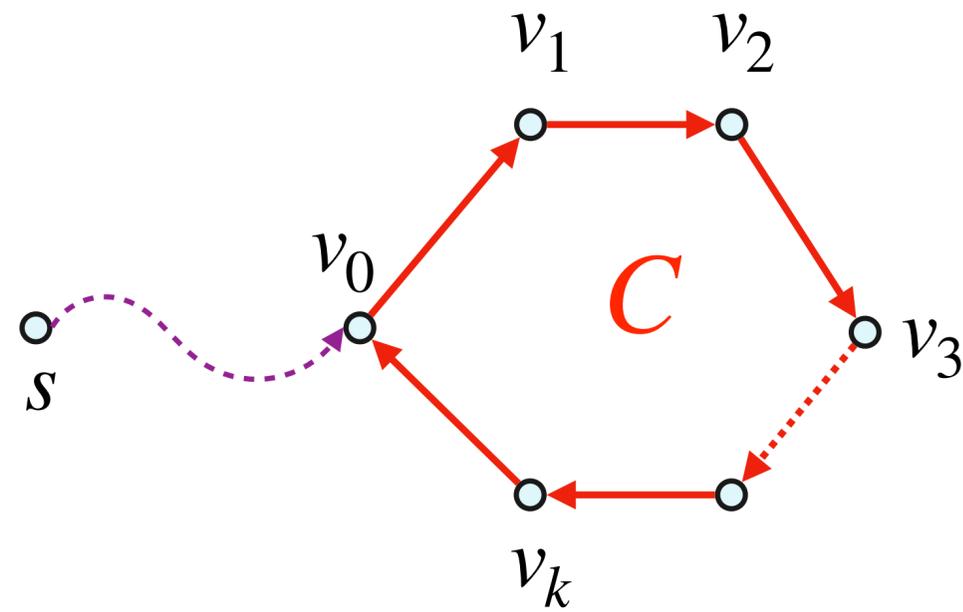
...

$$d(v_i, n - 1) \leq d(v_{i-1}, n - 1) + l(v_{i-1}, v_i)$$

...

Assumed that for all v , $d(v, n) \geq d(v, n - 1)$

Proof of lemma ...



$$d(v_1, n - 1) \leq d(v_0, n - 1) + l(v_0, v_1)$$

$$d(v_2, n - 1) \leq d(v_1, n - 1) + l(v_1, v_2)$$

...

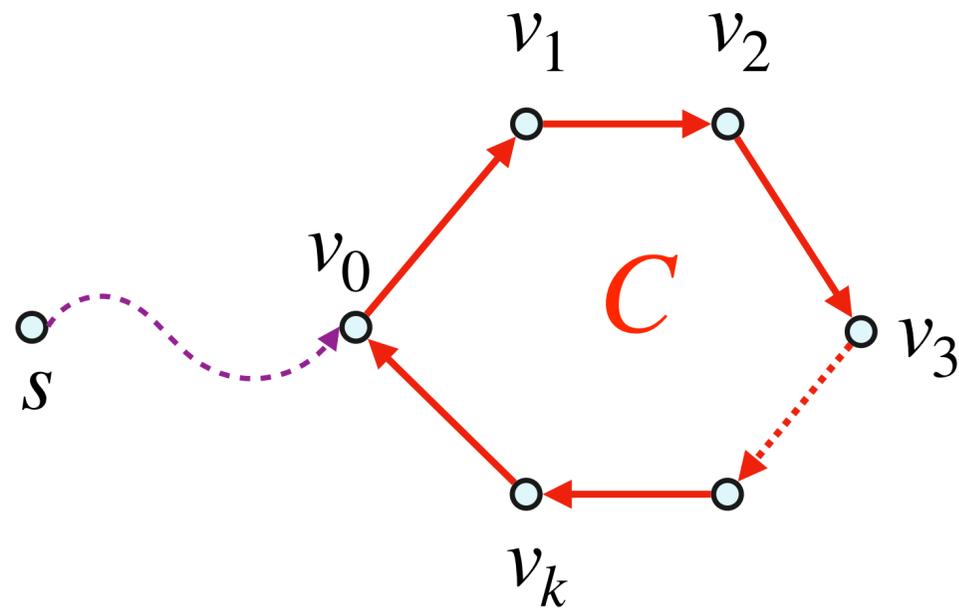
$$\leq d(v_i, n - 1) \leq d(v_{i-1}, n - 1) + l(v_{i-1}, v_i)$$

...

$$d(v_k, n - 1) \leq d(v_{k-1}, n - 1) + l(v_{k-1}, v_k)$$

$$d(v_0, n - 1) \leq d(v_k, n - 1) + l(v_k, v_0)$$

Proof of lemma ...

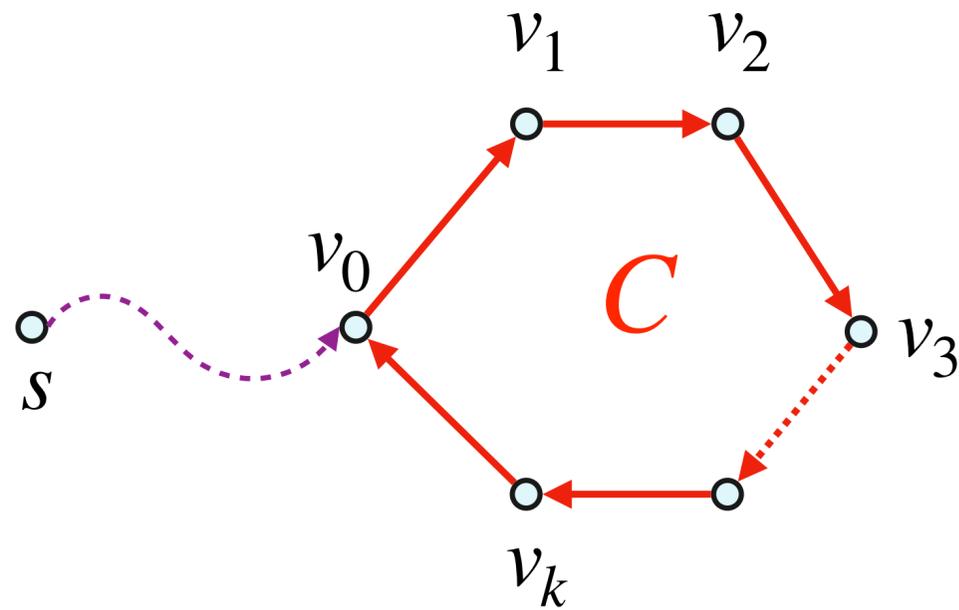


$$\sum_{i=0}^k d(v_i, n-1) \leq \sum_{i=0}^k d(v_i, n-1) + \sum_{i=1}^k l(v_{i-1}, v_i) + l(v_k, v_0)$$

$$0 \leq \sum_{i=1}^k l(v_{i-1}, v_i) + l(v_k, v_0) = \text{len}(C)$$

C is a **not** a negative cycle. Contradiction!

Proof of lemma ...



$$\sum_{i=0}^k d(v_i, n-1) \leq \sum_{i=0}^k d(v_i, n-1) + \sum_{i=1}^k l(v_{i-1}, v_i) + l(v_k, v_0)$$

$$0 \leq \sum_{i=1}^k l(v_{i-1}, v_i) + l(v_k, v_0) = \text{len}(C)$$

C is a **not** a negative cycle. Contradiction!

Essence of the lemma: If G has a negative cycle reachable from s , then it can be detected in n iterations of the Bellman-Ford algorithm.

Bellman-Ford: negative cycle detection

Final version

```
for each  $u \in V$  do
```

```
     $d(u) \leftarrow \infty$ 
```

```
 $d(s) \leftarrow 0$ 
```

```
for  $k = 1$  to  $n - 1$  do
```

```
    for each  $v \in V$  do
```

```
        for each edge  $(u, v) \in \text{In}(v)$  do
```

```
             $d(v) = \min\{d(v), d(u) + l(u, v)\}$ 
```

```
(* One more iteration to check if distances change *)
```

```
for each  $v \in V$  do
```

```
    for each edge  $(u, v) \in \text{In}(v)$  do
```

```
        if  $(d(v) > d(u) + l(u, v))$ 
```

```
            Output "Negative Cycle"
```

```
for each  $v \in V$  do
```

```
     $\text{dist}(s, v) \leftarrow d(v, n-1)$ 
```

Bellman-Ford: negative cycle detection

Final version

```
for each  $u \in V$  do
```

```
     $d(u) \leftarrow \infty$ 
```

```
 $d(s) \leftarrow 0$ 
```

```
for  $k = 1$  to  $n - 1$  do
```

```
    for each  $v \in V$  do
```

```
        for each edge  $(u, v) \in \text{In}(v)$  do
```

```
             $d(v) = \min\{d(v), d(u) + l(u, v)\}$ 
```

```
(* One more iteration to check if distances change *)
```

```
for each  $v \in V$  do
```

```
    for each edge  $(u, v) \in \text{In}(v)$  do
```

```
        if  $(d(v) > d(u) + l(u, v))$ 
```

```
            Output "Negative Cycle"
```

```
for each  $v \in V$  do
```

```
     $\text{dist}(s, v) \leftarrow d(v, n-1)$ 
```

All lines colored in
red

where already
present previously.

Variants on Bellman-Ford

Finding the shortest path tree

How do we find a shortest path tree in addition to distances?

Finding the shortest path tree

How do we find a shortest path tree in addition to distances?

- For each v the $d(v)$ can only get smaller as the algorithm proceeds.

Finding the shortest path tree

How do we find a shortest path tree in addition to distances?

- For each v the $d(v)$ can only get smaller as the algorithm proceeds.
- If $d(v)$ becomes smaller it is because we found a vertex u such that $d(v) > d(u) + l(u, v)$ and we update $d(v) = d(u) + l(u, v)$. That is, we found a shorter path to v through u .

Finding the shortest path tree

How do we find a shortest path tree in addition to distances?

- For each v the $d(v)$ can only get smaller as the algorithm proceeds.
- If $d(v)$ becomes smaller it is because we found a vertex u such that $d(v) > d(u) + l(u, v)$ and we update $d(v) = d(u) + l(u, v)$. That is, we found a shorter path to v through u .
- For each v have a $\text{prev}(v)$ pointer and update it to point to u if v finds a shorter path via u .

Finding the shortest path tree

How do we find a shortest path tree in addition to distances?

- For each v the $d(v)$ can only get smaller as the algorithm proceeds.
- If $d(v)$ becomes smaller it is because we found a vertex u such that $d(v) > d(u) + l(u, v)$ and we update $d(v) = d(u) + l(u, v)$. That is, we found a shorter path to v through u .
- For each v have a $\text{prev}(v)$ pointer and update it to point to u if v finds a shorter path via u .
- At the end of the algorithm $\text{prev}(v)$ pointers give a shortest path tree oriented towards the source s .

Negative cycle detection

Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

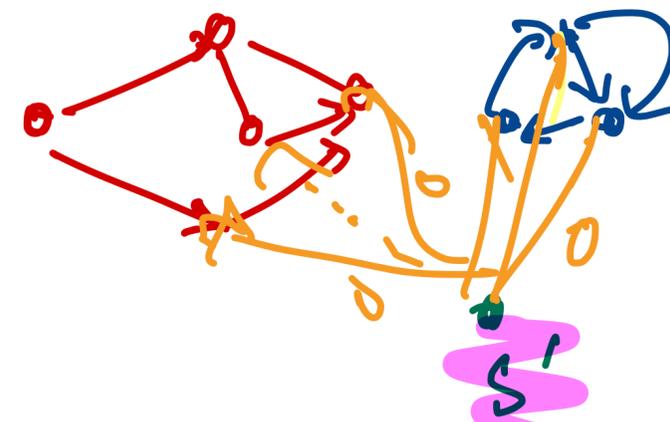
- Bellman-Ford checks whether there is a negative cycle C that is reachable from a specific vertex s . There may be negative cycles ***not*** reachable from s .
- 

Negative cycle detection

Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

- Bellman-Ford checks whether there is a negative cycle C that is reachable from a specific vertex s . There may be negative cycles **not** reachable from s .
- Run Bellman-Ford $|V|$ times, once from each node u ?

Negative cycle detection



Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

- Bellman-Ford checks whether there is a negative cycle C that is reachable from a specific vertex s . There may be negative cycles **not** reachable from s .
- Run Bellman-Ford $|V|$ times, once from each node u ?
- Add a new node s' and connect it to all nodes of G with zero length edges. Bellman-Ford from s' will find a negative length cycle if there is one.
Exercise: why does this work?

Negative cycle detection

Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

- Bellman-Ford checks whether there is a negative cycle C that is reachable from a specific vertex s . There may be negative cycles **not** reachable from s .
 - Run Bellman-Ford $|V|$ times, once from each node u ?
- Add a new node s' and connect it to all nodes of G with zero length edges. Bellman-Ford from s' will find a negative length cycle if there is one.
Exercise: why does this work?
- \implies negative cycle detection can be done with one Bellman-Ford invocation.



Shortest paths in a DAG

Input: A directed acyclic graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

no cycles

- Given nodes s, t find shortest path from s to t .
- Given node s find shortest path from s to all other nodes.

Shortest paths in a DAG

Input: A directed **acyclic** graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

- Given nodes s, t find shortest path from s to t .
- Given node s find shortest path from s to all other nodes.

Simplification of algorithms for **DAGs**

Shortest paths in a DAG

Input: A directed **acyclic** graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

- Given nodes s, t find shortest path from s to t .
- Given node s find shortest path from s to all other nodes.

Simplification of algorithms for **DAGs**

- No cycles and hence no negative length cycles! Hence can find shortest paths even for negative length edges.

Shortest paths in a DAG

Input: A directed **acyclic** graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

- Given nodes s, t find shortest path from s to t .
- Given node s find shortest path from s to all other nodes.

Simplification of algorithms for **DAGs**

- No cycles and hence no negative length cycles! Hence can find shortest paths even for negative length edges.
- Can order nodes using topological sort.

Algorithm for DAGs

- Want to find shortest paths from s . Ignore nodes not reachable from s .
- Let $s = v_1, v_2, v_{i+1}, \dots, v_n$ be a topological sort of G .

Observations:

Algorithm for DAGs

- Want to find shortest paths from s . Ignore nodes not reachable from s .
- Let $s = v_1, v_2, v_{i+1}, \dots, v_n$ be a topological sort of G .

Observations:

- Shortest path from s to v_i cannot use any node from v_{i+1}, \dots, v_n .

Algorithm for DAGs

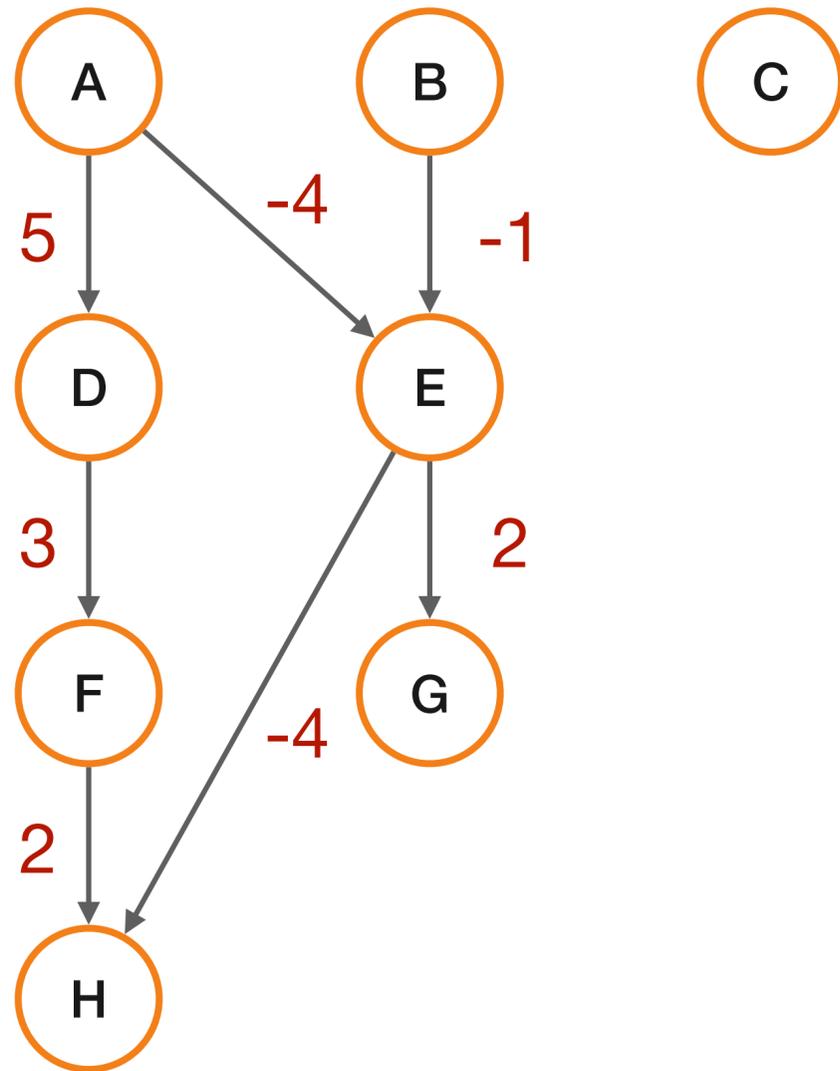
- Want to find shortest paths from s . Ignore nodes not reachable from s .
- Let $s = v_1, v_2, v_{i+1}, \dots, v_n$ be a topological sort of G .

Observations:

- Shortest path from s to v_i cannot use any node from v_{i+1}, \dots, v_n .
- Can also find shortest paths in topologically sorted ordering.

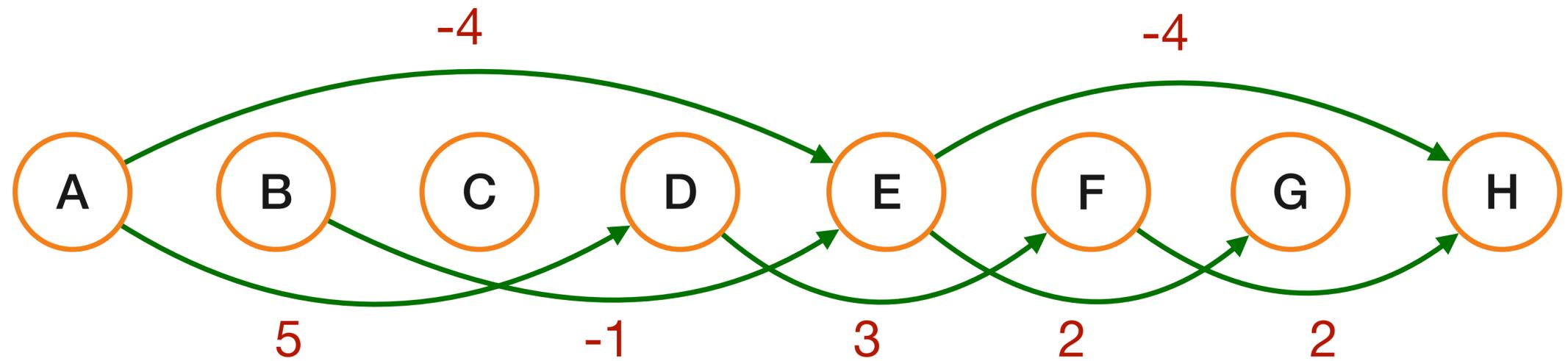
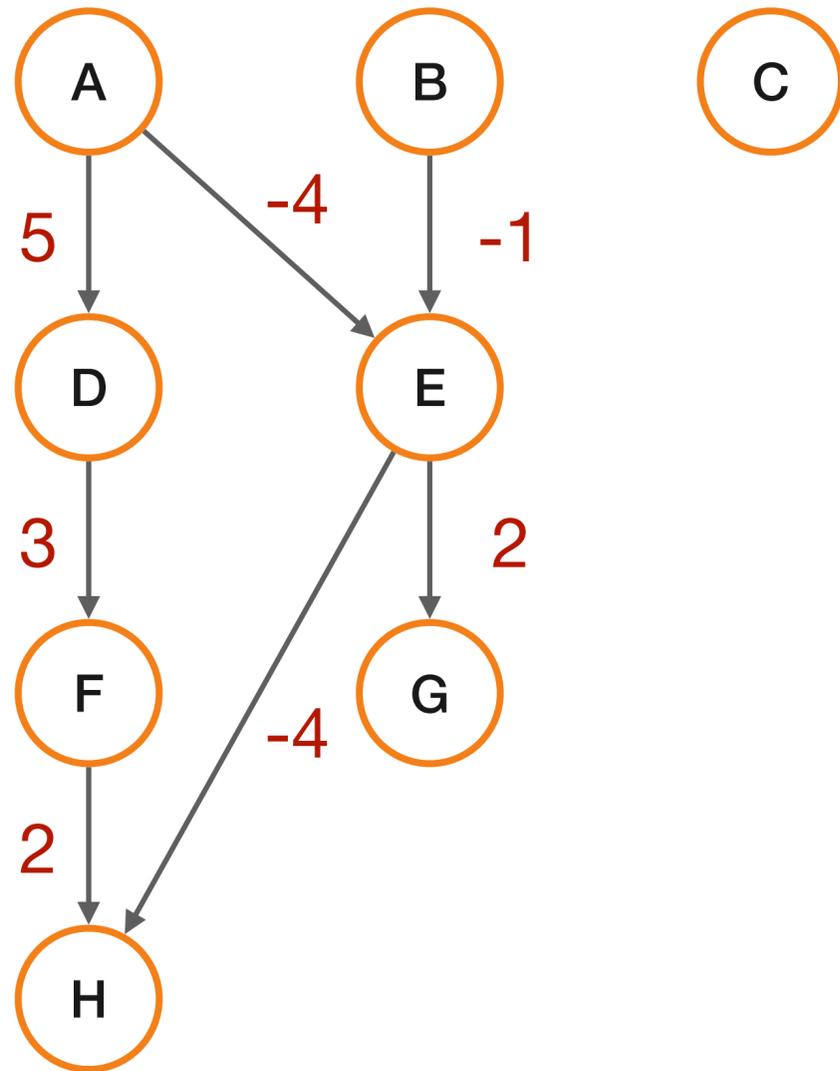
Shortest Paths for DAGs

Example



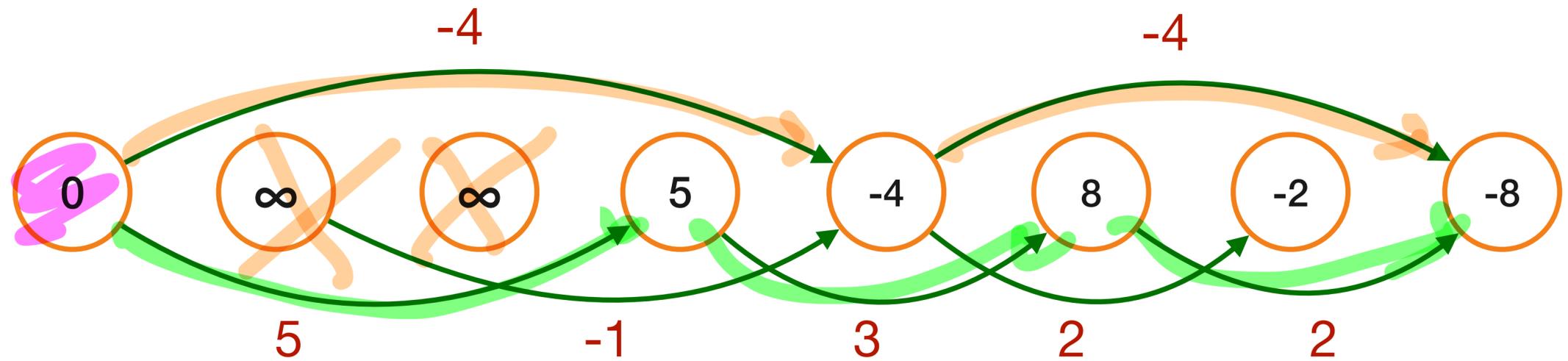
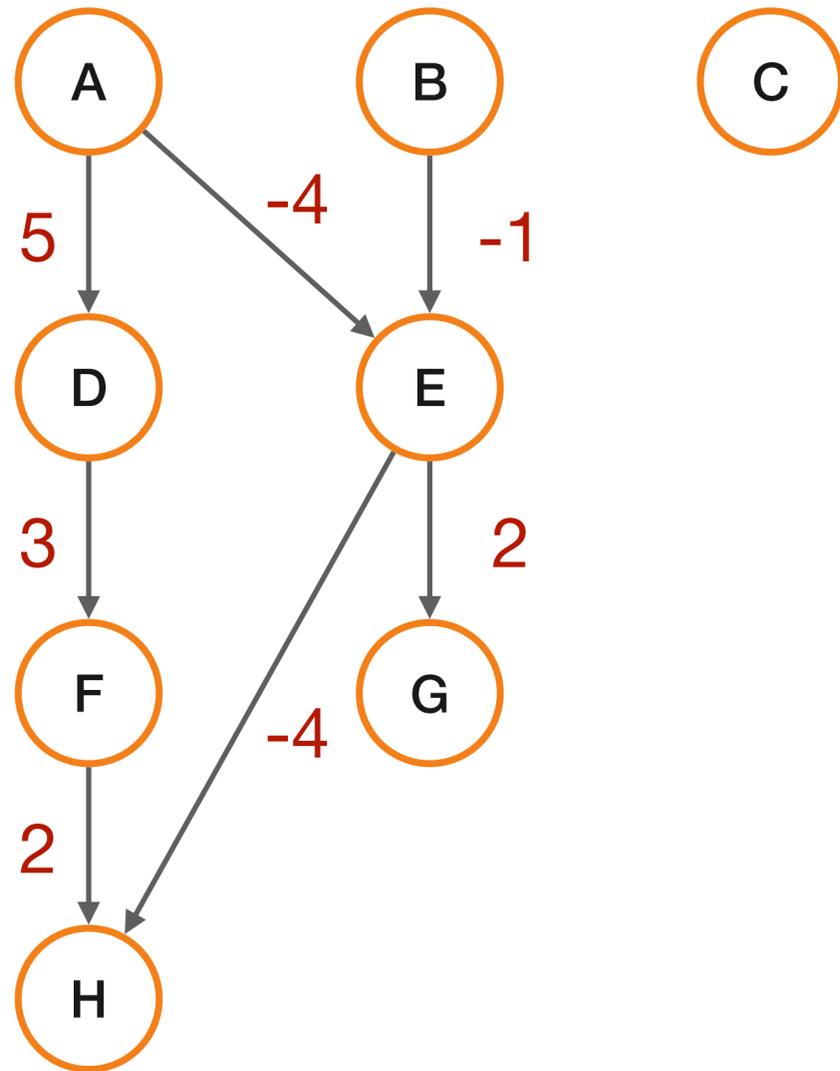
Shortest Paths for DAGs

Example



Shortest Paths for DAGs

Example



Algorithm for DAGs

```
for  $i = 1$  to  $n$  do
```

```
     $d(s, v_i) = \infty$ 
```

```
 $d(s, s) = 0$ 
```

```
for  $i = 1$  to  $n - 1$  do
```

```
    for each edge  $(v_i, v_j)$  in  $Adj(v_i)$  do
```

```
         $d(s, v_j) = \min\{d(s, v_j), d(s, v_i) + l(v_i, v_j)\}$ 
```

```
return  $d(s, \cdot)$  values computed
```

Algorithm for DAGs

```
for  $i = 1$  to  $n$  do  
     $d(s, v_i) = \infty$   
  
 $d(s, s) = 0$   
for  $i = 1$  to  $n - 1$  do  
    for each edge  $(v_i, v_j)$  in  $Adj(v_i)$  do  
         $d(s, v_j) = \min\{d(s, v_j), d(s, v_i) + l(v_i, v_j)\}$   
  
return  $d(s, \cdot)$  values computed
```

Correctness: Induction on i and observations in previous slide.

Running time: $O(m + n)$ time algorithm! Works for negative edge lengths and hence can find longest paths in a DAG.

All pairs shortest paths

Shortest Path Problems

Input: A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

- Given nodes s, t find shortest path from s to t .
- Given node s find shortest path from s to all other nodes.

All pairs shortest paths

Shortest Path Problems

Input: A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length.

- Given nodes s, t find shortest path from s to t .
- Given node s find shortest path from s to all other nodes.
- **Find shortest paths for all pairs of nodes.** \rightarrow left over to discuss

All-Pairs Shortest Paths - Using known algorithms...

All-Pairs Shortest Path Problem

Input: A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length. Find shortest paths for all pairs of nodes.

If we apply single-source algorithms n times, once for each vertex.

All-Pairs Shortest Paths - Using known algorithms...

All-Pairs Shortest Path Problem

Input: A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length. Find shortest paths for all pairs of nodes.

If we apply single-source algorithms n times, once for each vertex.

- Non-negative lengths. $O(nm \log n)$ with heaps and $O(nm + n^2 \log n)$ using advanced priority queues. \rightarrow Dijkstra

All-Pairs Shortest Paths - Using known algorithms...

All-Pairs Shortest Path Problem

Input: A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length. Find shortest paths for all pairs of nodes.

If we apply single-source algorithms n times, once for each vertex.

- Non-negative lengths. $O(nm \log n)$ with heaps and $O(nm + n^2 \log n)$ using advanced priority queues.
- Arbitrary edge lengths: $O(n^2 m)$ and $\Theta(n^4)$ if $m = \Omega(n^2)$

Behren Farel.

All-Pairs Shortest Paths - Using known algorithms...

All-Pairs Shortest Path Problem

Input: A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $l(e) = l(u, v)$ is its length. Find shortest paths for all pairs of nodes.

If we apply single-source algorithms n times, once for each vertex.

- Non-negative lengths. $O(nm \log n)$ with heaps and $O(nm + n^2 \log n)$ using advanced priority queues.
- Arbitrary edge lengths: $O(n^2m)$ and $\Theta(n^4)$ if $m = \Omega(n^2)$

Can we do better? *Yes (obviously).*

All Pairs Shortest Paths: A recursive solution

All-Pairs Shortest Paths

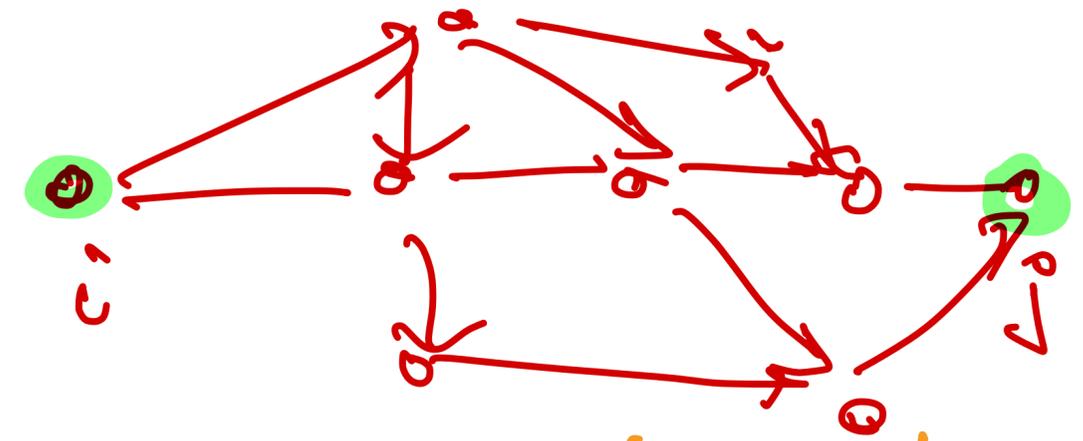
Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n

All-Pairs Shortest Paths

Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the **largest** index of an **intermediate** node is **at most k** (could be $-\infty$ if there is a negative length cycle).

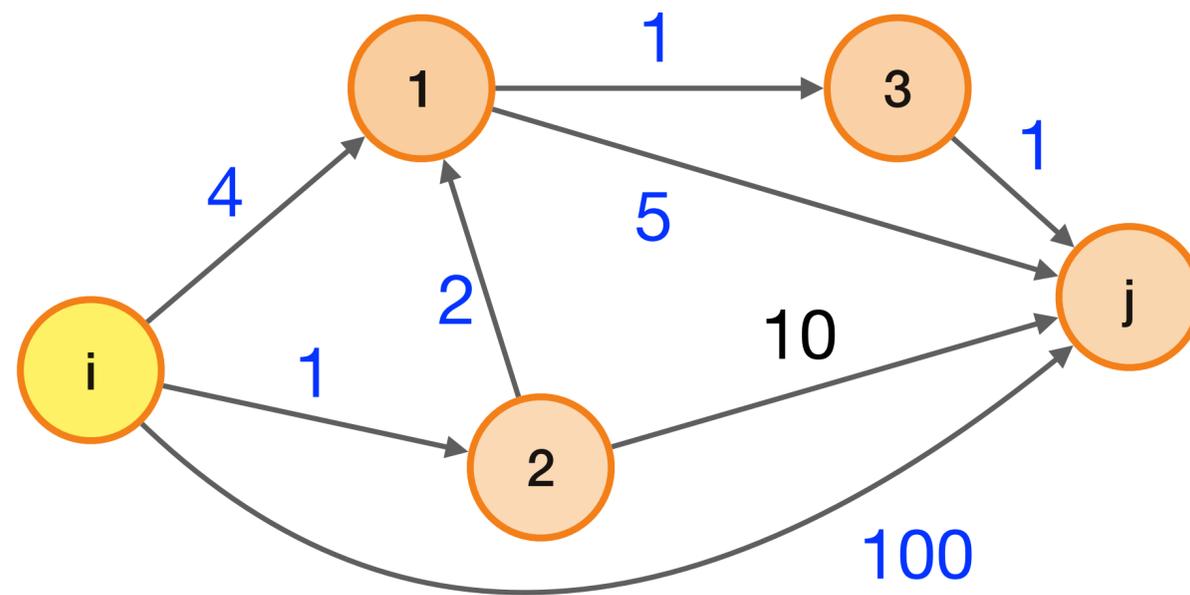


many paths but
restrict all intermediate
nodes to be $\leq k$

All-Pairs Shortest Paths

Recursion on index of intermediate nodes

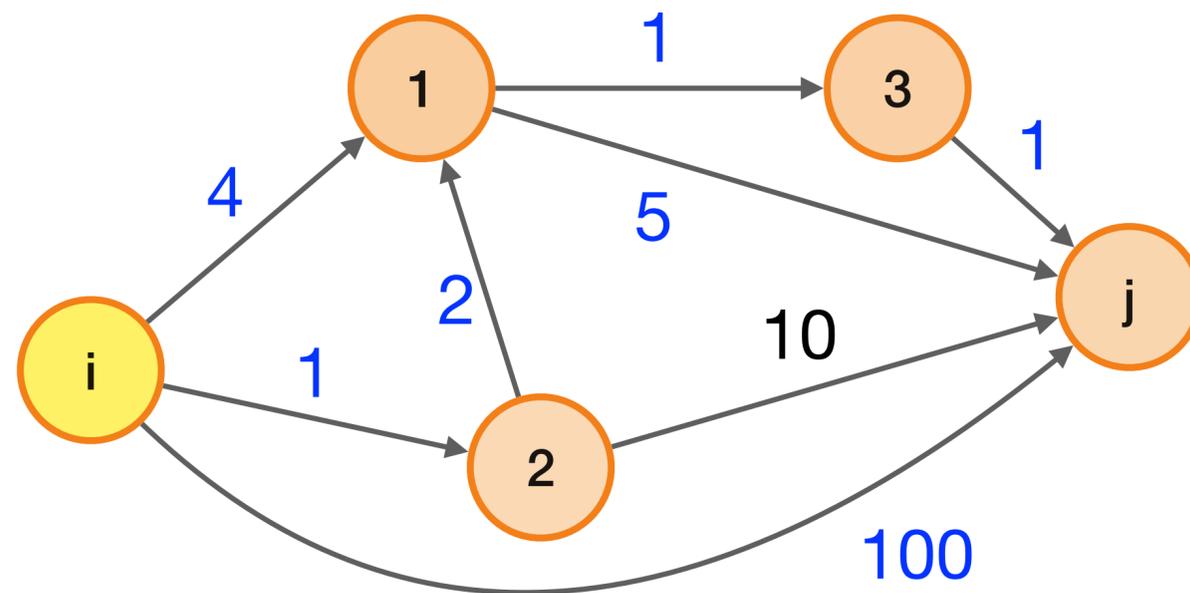
- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the **largest index** of an **intermediate** node is **at most** k (could be $-\infty$ if there is a negative length cycle).



All-Pairs Shortest Paths

Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the **largest index** of an **intermediate** node is **at most** k (could be $-\infty$ if there is a negative length cycle).

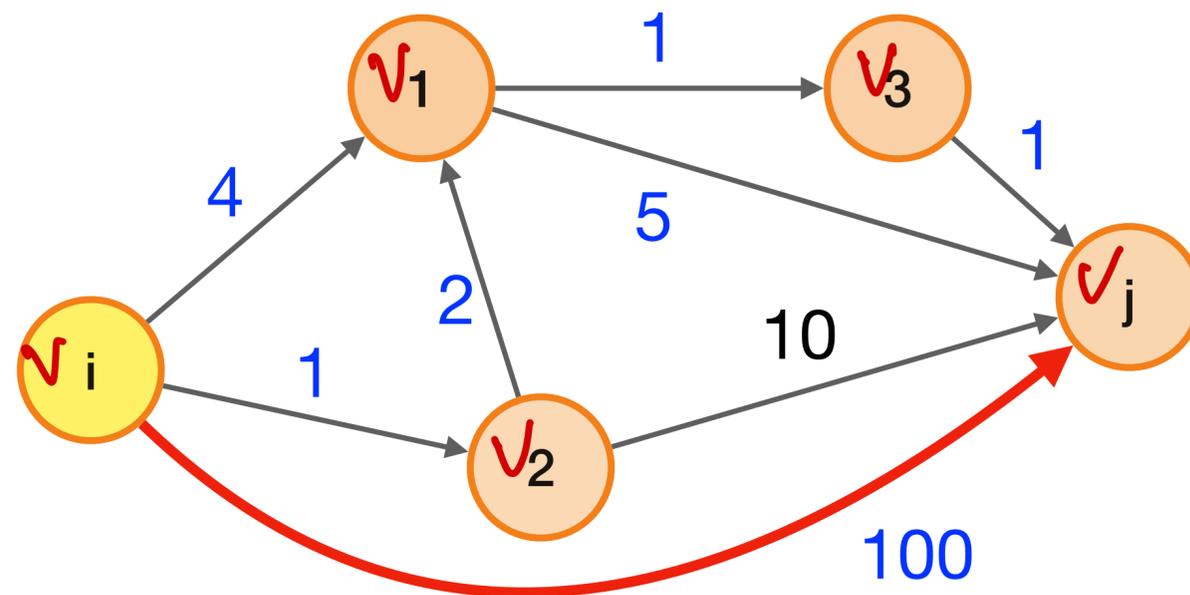


$$\begin{aligned} \text{dist}(i, j, 0) &= \\ \text{dist}(i, j, 1) &= \\ \text{dist}(i, j, 2) &= \\ \text{dist}(i, j, 3) &= \end{aligned}$$

All-Pairs Shortest Paths

Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



$$\text{dist}(i, j, 0) = 100$$

$$\text{dist}(i, j, 1) =$$

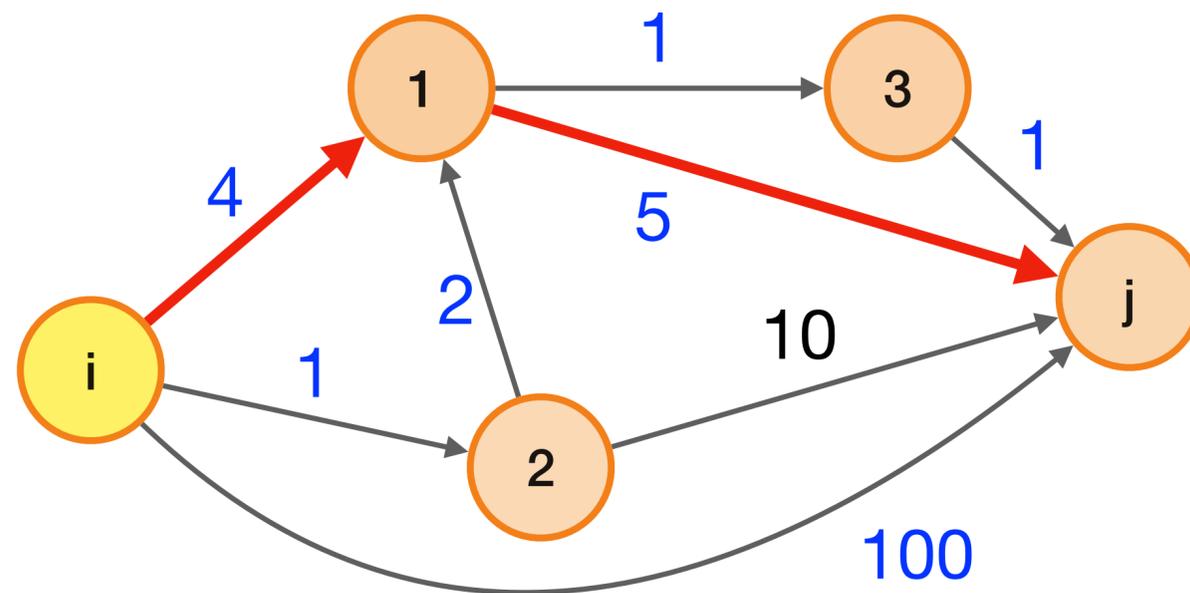
$$\text{dist}(i, j, 2) =$$

$$\text{dist}(i, j, 3) =$$

All-Pairs Shortest Paths

Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



$$\text{dist}(i, j, 0) = 100$$

$$\text{dist}(i, j, 1) = 9$$

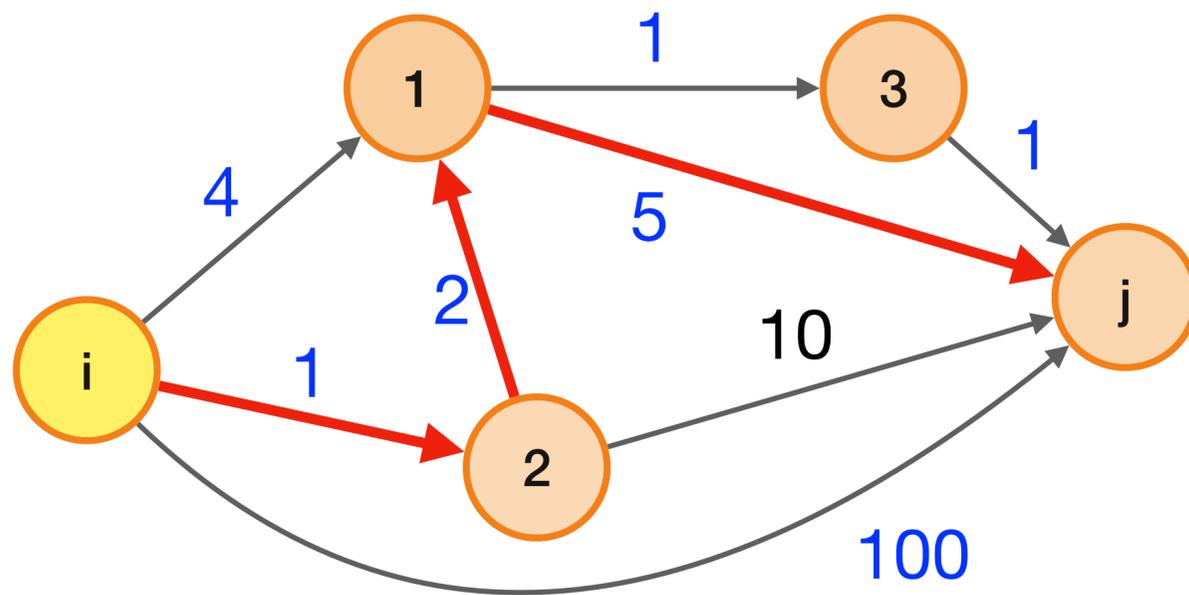
$$\text{dist}(i, j, 2) =$$

$$\text{dist}(i, j, 3) =$$

All-Pairs Shortest Paths

Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



$$\text{dist}(i, j, 0) = 100$$

$$\text{dist}(i, j, 1) = 9$$

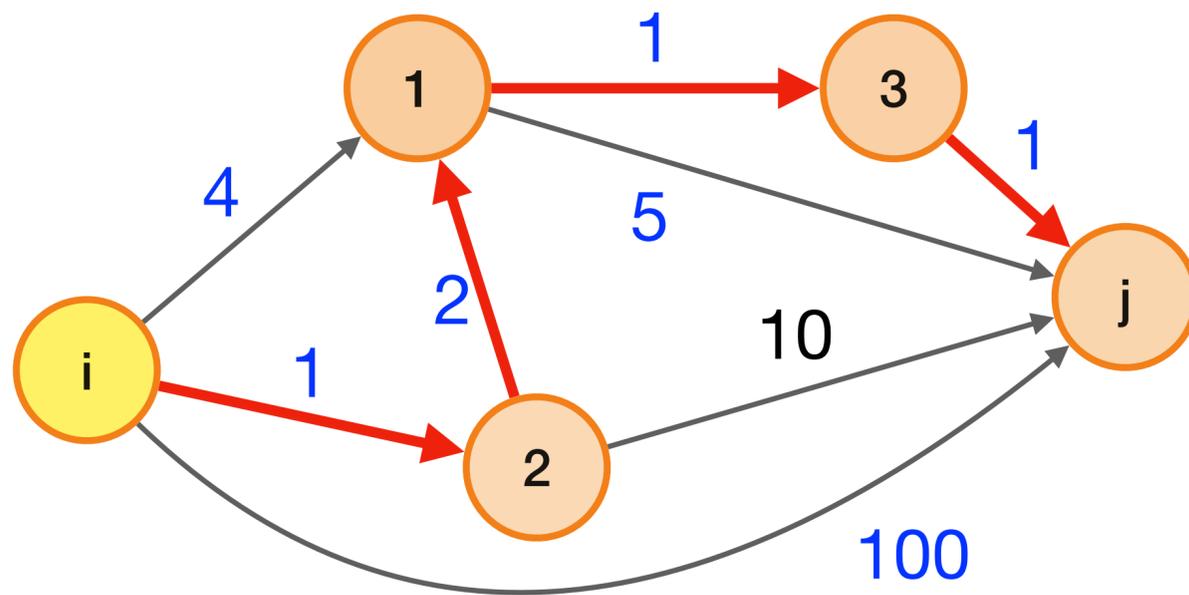
$$\text{dist}(i, j, 2) = 8$$

$$\text{dist}(i, j, 3) =$$

All-Pairs Shortest Paths

Recursion on index of intermediate nodes

- Number vertices arbitrarily as v_1, v_2, \dots, v_n
- $\text{dist}(i, j, k)$: length of **shortest** walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



$$\text{dist}(i, j, 0) = 100$$

$$\text{dist}(i, j, 1) = 9$$

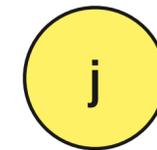
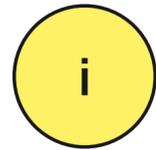
$$\text{dist}(i, j, 2) = 8$$

$$\text{dist}(i, j, 3) = 5$$

↓
note
going
down

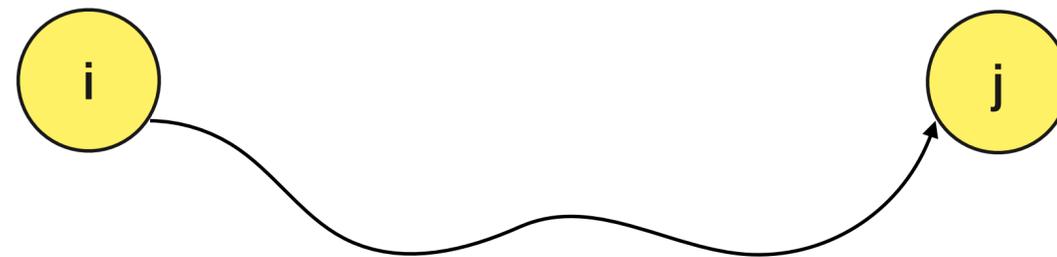
All-Pairs Shortest Paths

Recursion on index of intermediate nodes



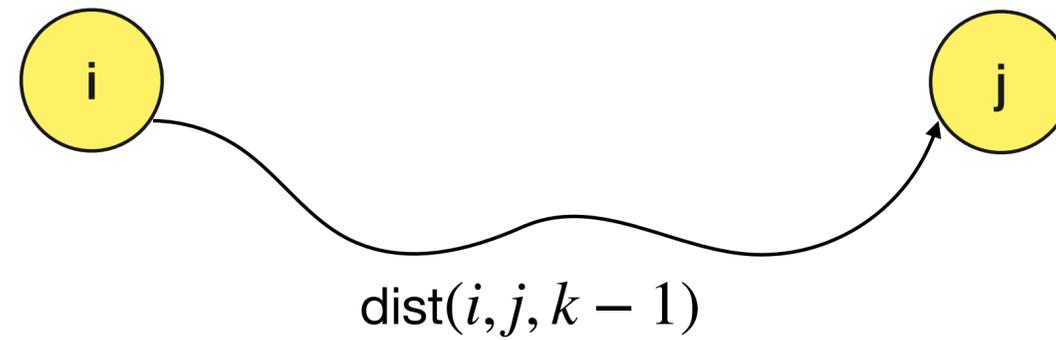
All-Pairs Shortest Paths

Recursion on index of intermediate nodes



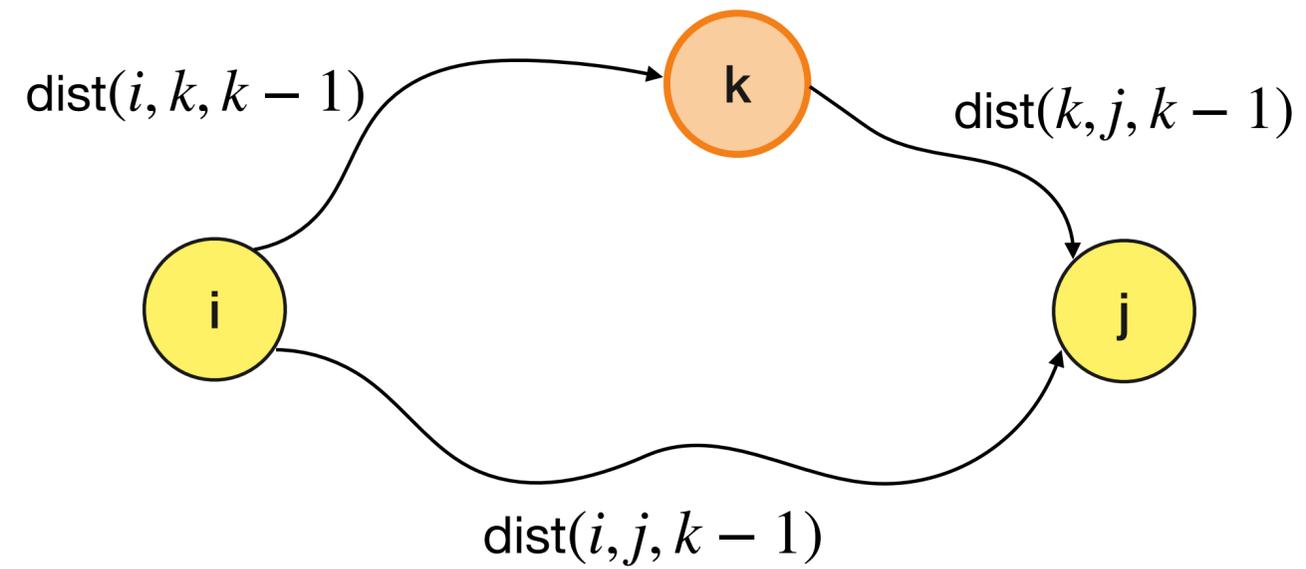
All-Pairs Shortest Paths

Recursion on index of intermediate nodes



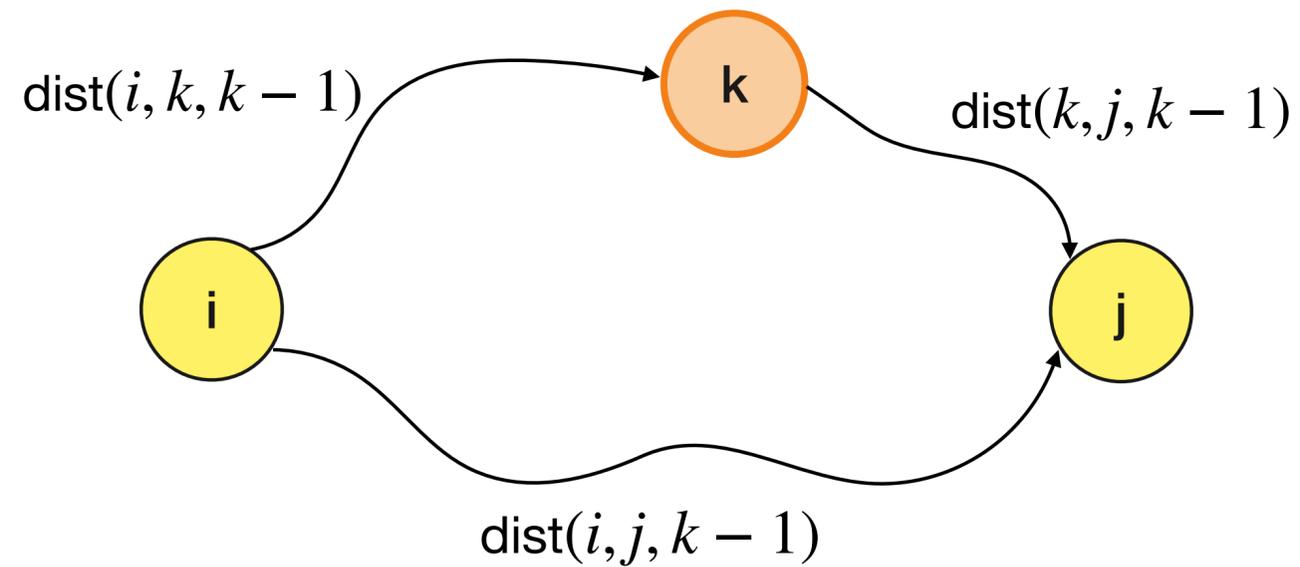
All-Pairs Shortest Paths

Recursion on index of intermediate nodes



All-Pairs Shortest Paths

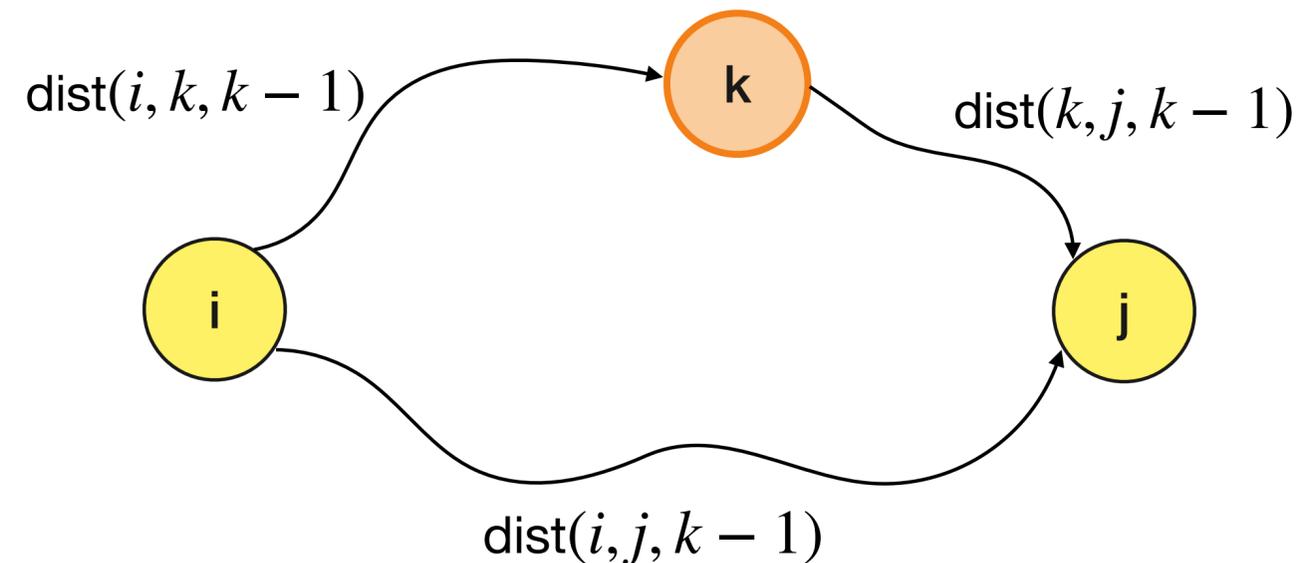
Recursion on index of intermediate nodes



$$\text{dist}(i, j, k) = \min \begin{cases} \text{dist}(i, j, k - 1) \\ \text{dist}(i, k, k - 1) + \text{dist}(k, j, k - 1) \end{cases}$$

All-Pairs Shortest Paths

Recursion on index of intermediate nodes



$$\text{dist}(i, j, k) = \min \begin{cases} \text{dist}(i, j, k-1) \\ \text{dist}(i, k, k-1) + \text{dist}(k, j, k-1) \end{cases}$$

- Base case: $\text{dist}(i, j, 0) = l(i, j)$ if $(i, j) \in E$, otherwise ∞
- **Correctness:** If $i \rightarrow j$ shortest walk goes through k then k occurs only once on the path — otherwise there is a negative length cycle

All-Pairs: Recursion on index of intermediate nodes

If i can reach k and k can reach j and $\text{dist}(k, k, k - 1) < 0$ then G has a negative length cycle containing k and $\text{dist}(i, j, k) = -\infty$.

All-Pairs: Recursion on index of intermediate nodes

If i can reach k and k can reach j and $\text{dist}(k, k, k - 1) < 0$ then G has a negative length cycle containing k and $\text{dist}(i, j, k) = -\infty$.

Recursion below is valid only if $\text{dist}(k, k, k - 1) \geq 0$. We can detect this during the algorithm or wait till the end.

All-Pairs: Recursion on index of intermediate nodes

If i can reach k and k can reach j and $\text{dist}(k, k, k - 1) < 0$ then G has a negative length cycle containing k and $\text{dist}(i, j, k) = -\infty$.

Recursion below is valid only if $\text{dist}(k, k, k - 1) \geq 0$. We can detect this during the algorithm or wait till the end.

$$\text{dist}(i, j, k) = \min \begin{cases} \text{dist}(i, j, k - 1) \\ \text{dist}(i, k, k - 1) + \text{dist}(k, j, k - 1) \end{cases}$$

Floyd-Warshall Algorithm

Floyd - Warshall Algorithm

For All-Pairs Shortest Paths

```
for  $i = 1$  to  $n$  do
    for  $j = 1$  to  $n$  do
         $d(i, j, 0) = l(i, j)$ 

(*  $l(i, j) = \infty$  if  $(i, j) \notin E$ ,  $0$  if  $i = j$  *)
for  $k = 1$  to  $n$  do
    for  $i = 1$  to  $n$  do
        for  $j = 1$  to  $n$  do
             $\text{dist}(i, j, k) = \min \begin{cases} \text{dist}(i, j, k - 1) \\ \text{dist}(i, k, k - 1) + \text{dist}(k, j, k - 1) \end{cases}$ 

for  $i = 1$  to  $n$  do
    if  $(\text{dist}(i, i, n) < 0)$  then
        Output  $\exists$  negative cycle in  $G$ 
```

Floyd - Warshall Algorithm

For All-Pairs Shortest Paths

```
for i = 1 to n do
  for j = 1 to n do
    d(i,j,0) = l(i,j)

(* l(i, j) = ∞ if (i, j) ∉ E, 0 if i = j *)
for k = 1 to n do
  for i = 1 to n do
    for j = 1 to n do
      dist(i, j, k) = min { dist(i, j, k - 1)
                          { dist(i, k, k - 1) + dist(k, j, k - 1)

for i = 1 to n do
  if (dist(i, i, n) < 0) then
    Output ∃ negative cycle in G
```

Running time: $\Theta(n^3)$

Space: $\Theta(n^3)$

Correctness: via induction
and recursive definition

Floyd - Warshall Algorithm

Finding the Paths

Question: Can we find the paths in addition to the distances?

- Create a $n \times n$ array **Next** that stores the next vertex on shortest path for each pair of vertices
- With array **Next**, for any pair of given vertices i, j can compute a shortest path in $O(n)$ time.

Floyd - Warshall Algorithm

Finding the Paths

```
for i = 1 to n do
  for j = 1 to n do
    d(i,j,0) = l(i,j)

(* l(i, j) = ∞ if (i, j) ∉ E, 0 if i = j *)
  Next(i, j) = -1
for k = 1 to n do
  for i = 1 to n do
    for j = 1 to n do
      if (d(i,j,k-1) > d(i,k,k-1) + d(k,j,k-1)) then
        d(i,j,k) = d(i,k,k-1) + d(k,j,k-1)
        Next(i, j) = k

for i = 1 to n do
  if (dist(i,i,n) < 0) then
    Output ∃ negative cycle in G
```

Summary of shortest path algorithms

$m \rightarrow \# \text{ edges}$
 $n \rightarrow \# \text{ vertices}$

Summary of results on shortest paths

Single source		
No negative edges	Dijkstra	$O(n \log n + m)$
Edge lengths can be negative	Bellman Ford	$O(nm)$

(1) The algorithm for the case that there are no negative cycles, and doing all shortest paths, works by computing a potential function using **Bellman-Ford** and then doing **Dijkstra**. It is mentioned for the sake of completeness, but it is outside the scope of the class.

All Pairs Shortest Paths		
No negative edges	$n * \text{Dijkstra}$	$O(n^2 \log n + nm)$
No negative cycles	$n * \text{Bellman Ford}$	$O(n^2 m) = O(n^4)$
No negative cycles	Johnson's ¹	$O(nm + n^2 \log n)$
No negative cycles	Floyd-Warsh <i>all</i>	$O(n^3)$
Unweighted	Matrix multiplication ²	$O(n^{2.38}), O(n^{2.58})$

(2) <https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss12/AdvancedGraphAlgorithms/Slides14.pdf>