Data Structures and Algorithms – COMS21103

2015/2016

Shortest Paths Revisited

Negative Weights and All-Pairs

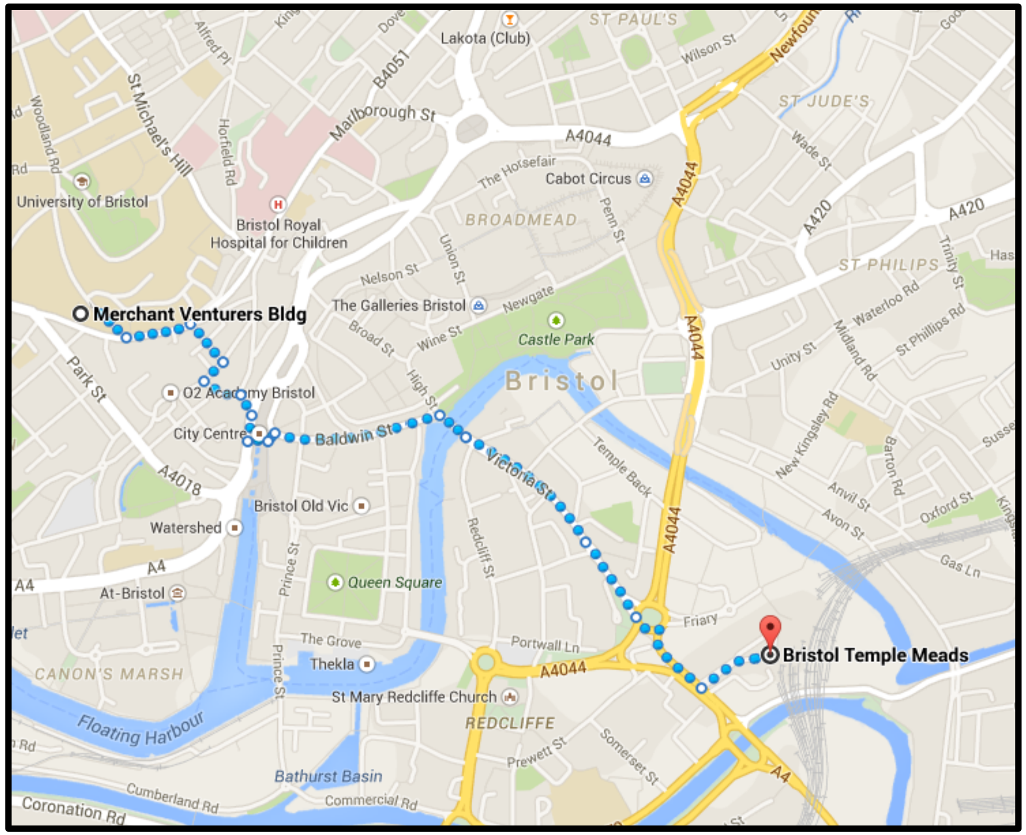
Benjamin Sach

inspired by slides by Ashley Montanaro

University of BRISTOL

In today's lectures we'll be revisiting the **shortest paths** problem

in a weighted, directed graph. . .



The shortest path from MVB to Temple Meads

(according to Google Maps)

In particular we'll be interested in algorithms which allow **negative edge weights**

and algorithms which compute the shortest path between **all pairs** of vertices

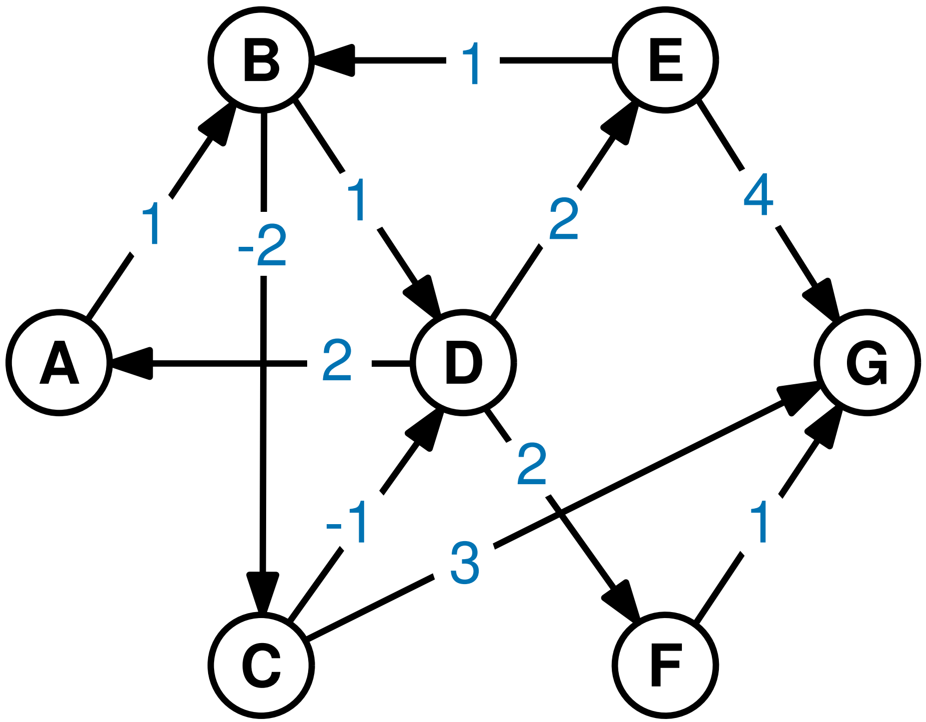
Part one

Single Source Shortest Paths with negative weights

Single source shortest paths with negative weights

Bellman-Ford's algorithm solves the **single source shortest paths** problem

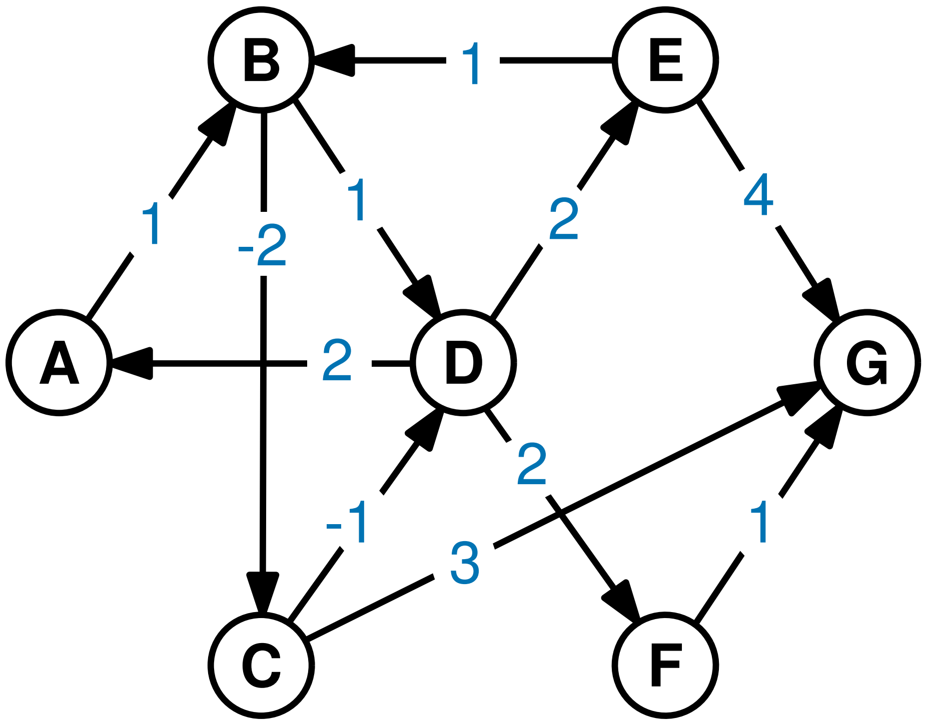
in a **weighted**, directed graph...



Single source shortest paths with negative weights

Bellman-Ford's algorithm solves the **single source shortest paths** problem

in a **weighted**, directed graph...



It finds the shortest path from

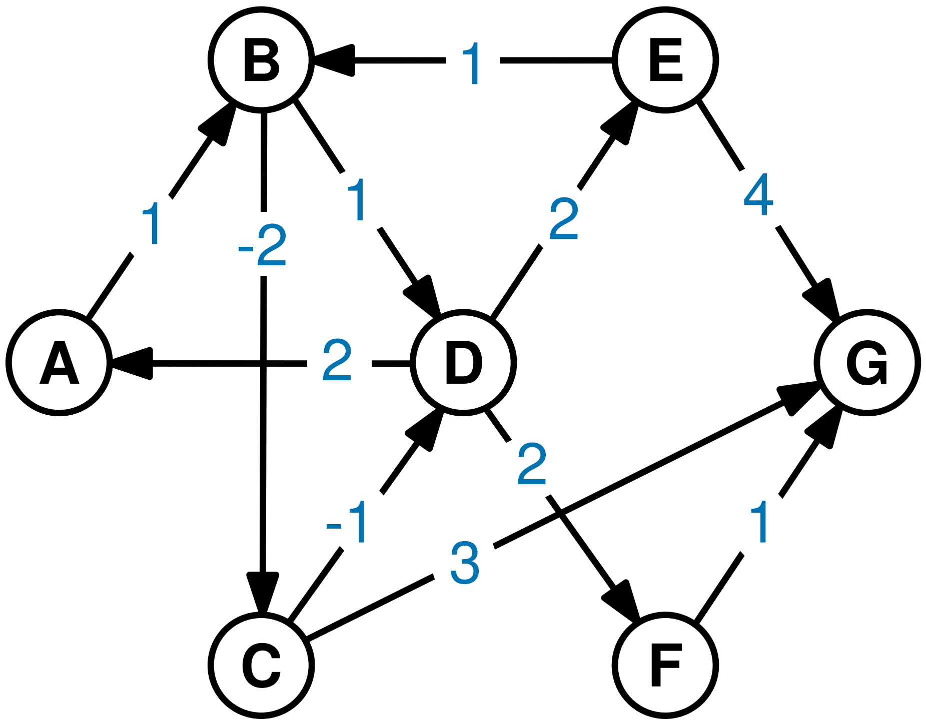
a given *source* vertex

to every other vertex

Single source shortest paths with negative weights

Bellman-Ford's algorithm solves the **single source shortest paths** problem

in a **weighted**, directed graph. . .



It finds the shortest path from

a given *source* vertex

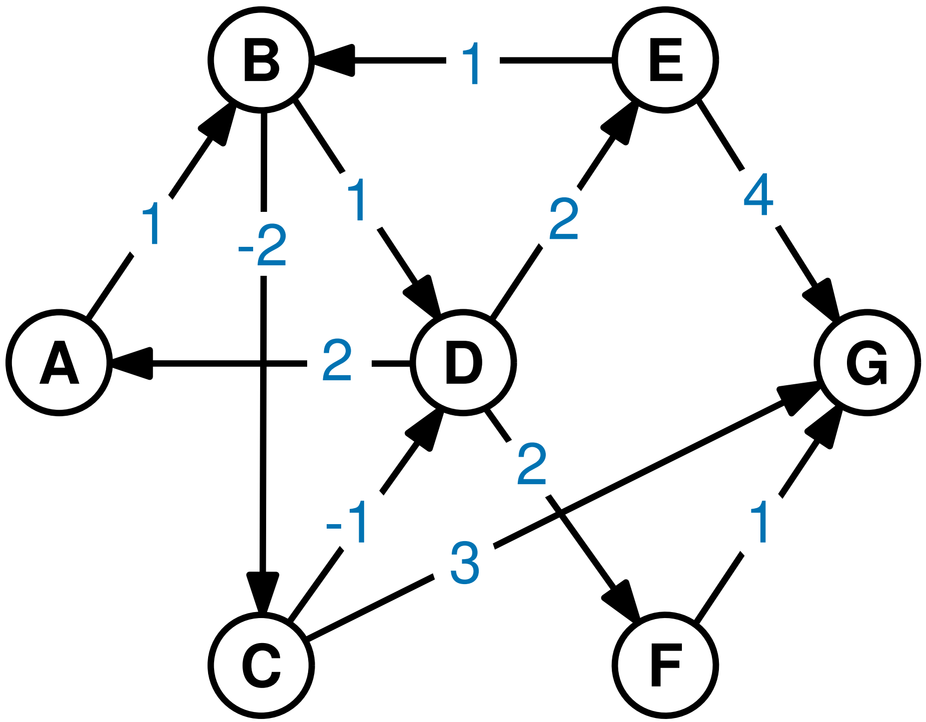
to every other vertex

The weights are allowed to be
positive or negative

Single source shortest paths with negative weights

Bellman-Ford's algorithm solves the **single source shortest paths** problem

in a **weighted**, directed graph. . .



It finds the shortest path from

a given *source* vertex

to every other vertex

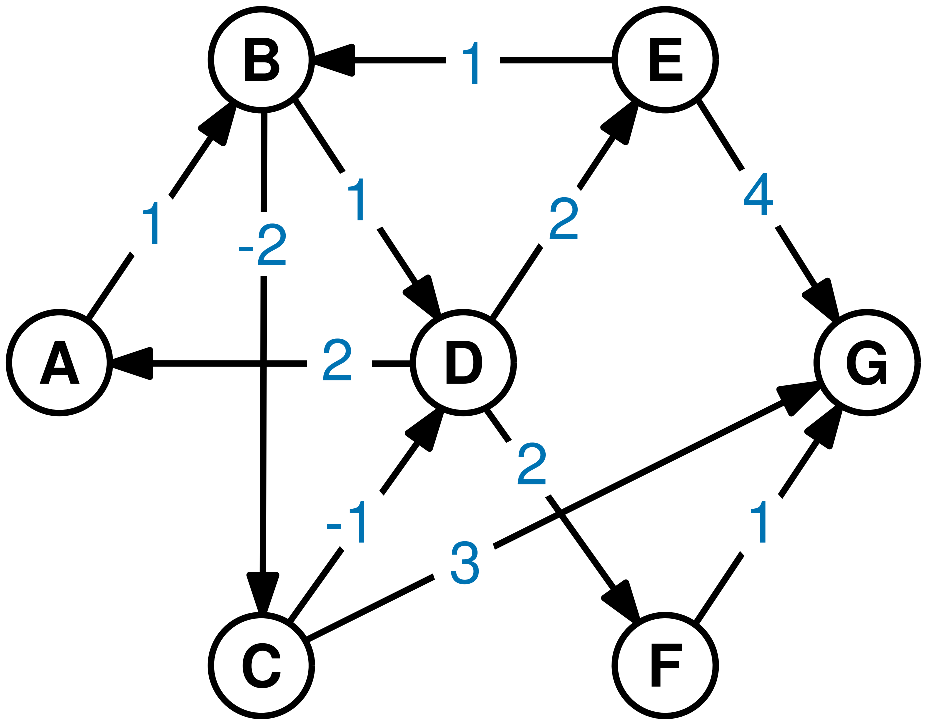
The weights are allowed to be
positive or negative

The graph is stored as an **Adjacency List**

Single source shortest paths with negative weights

Bellman-Ford's algorithm solves the **single source shortest paths** problem

in a **weighted**, directed graph. . .



It finds the shortest path from

a given *source* vertex

to every other vertex

The weights are allowed to be positive or negative

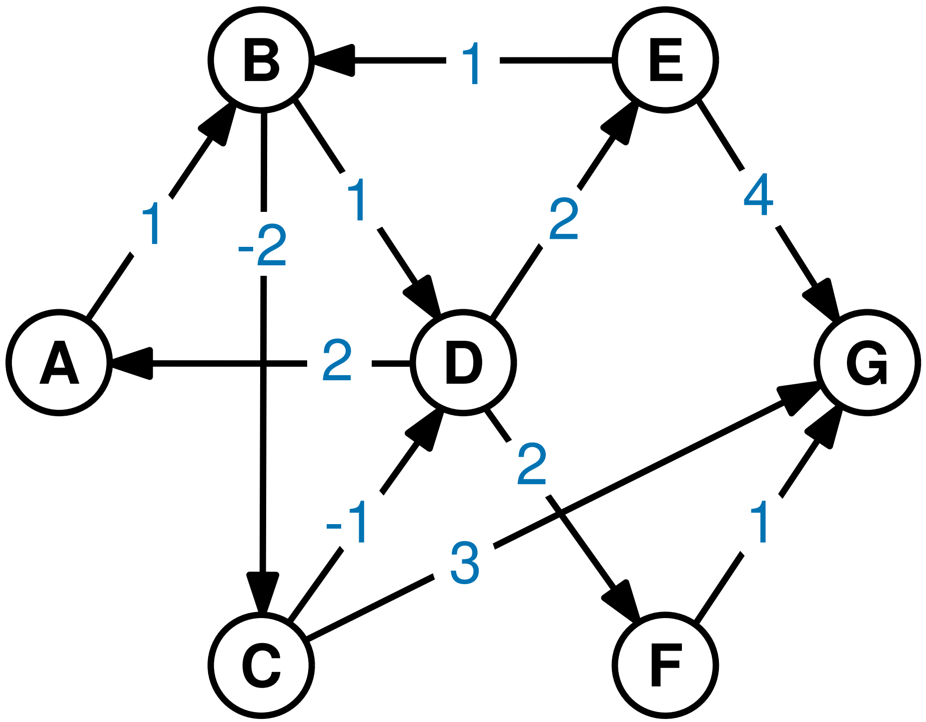
The graph is stored as an **Adjacency List**

We will not need any non-elementary data structures

Single source shortest paths with negative weights

Bellman-Ford's algorithm solves the **single source shortest paths** problem

in a **weighted**, directed graph. . .



It finds the shortest path from

a given *source* vertex

to every other vertex

The weights are allowed to be positive or negative

The graph is stored as an **Adjacency List**

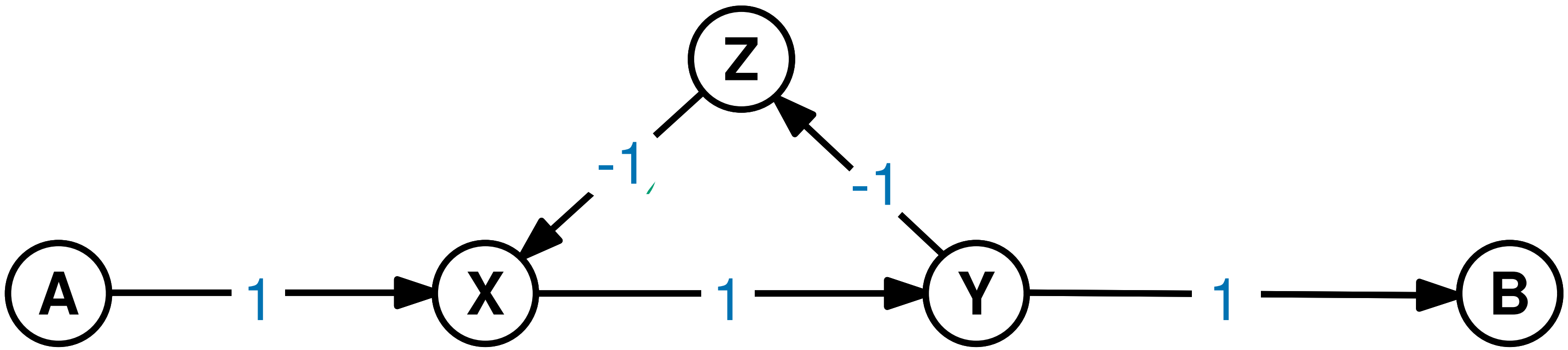
We will not need any non-elementary data structures

Previously we saw that Dijkstra's algorithm *(implemented with a binary heap)* solves this problem in $O((|V| + |E|) \log |V|)$ time when the edges have **non-negative weights**

$|V|$ is the number of vertices and $|E|$ is the number of edges

Negative weight cycles

If some of the edges in the graph have negative weights
the idea of a shortest path might not make sense:



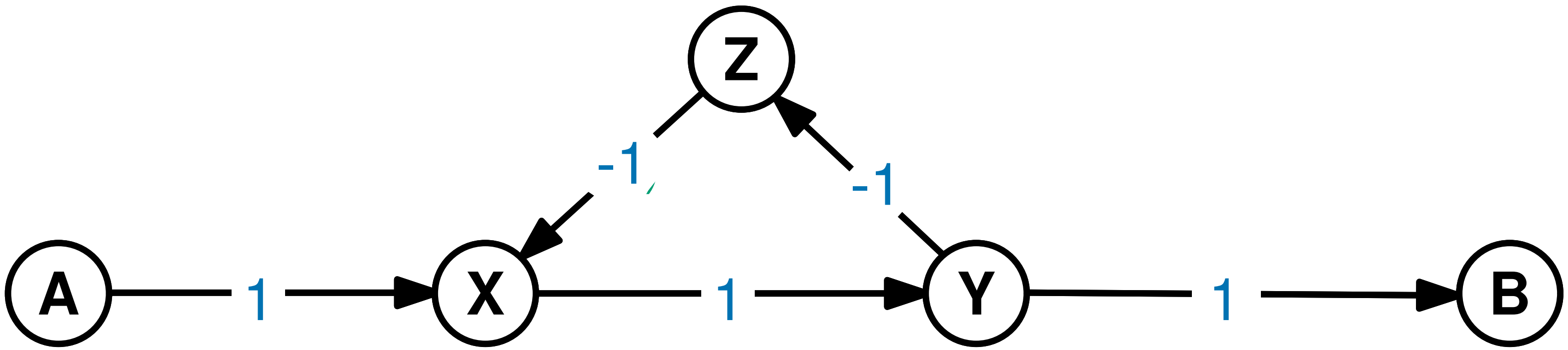
What is the shortest path from **A** to **B**?

A negative weight cycle is a path from a vertex v back to v such that
the sum of the edge weights is **negative**

If there is a path from s to t which includes a negative weight cycle,
there is no shortest path from s to t

Negative weight cycles

If some of the edges in the graph have negative weights
the idea of a shortest path might not make sense:



What is the shortest path from **A** to **B**?

A negative weight cycle is a path from a vertex v back to v such that
the sum of the edge weights is **negative**

If there is a path from s to t which includes a negative weight cycle,
there is no shortest path from s to t

We will first discuss a (slightly) simpler version of Bellman-Ford
that assumes there are no such cycles

Most of the Bellman-Ford algorithm

MOSTOFBELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    If dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)
```

$(u, v) \in E$ iff there is an edge *from* u *to* v

$\texttt{weight}(u, v)$ is the weight of the edge from u to v

$\texttt{dist}(v)$ is the length of the shortest path between s and v, found so far

Most of the Bellman-Ford algorithm

MOSTOFBELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1,2,...,|V|,
  For every edge (u,v) ∈ E
    If dist(v) > dist(u) + weight(u,v)
      dist(v) = dist(u) + weight(u,v)
```

The algorithm repeatedly asks, for each edge (u, v)

"can I find a shorter route to v if I go via u?"

$(u, v) \in E$ iff there is an edge *from* u *to* v

$\texttt{weight}(u, v)$ is the weight of the edge from u to v

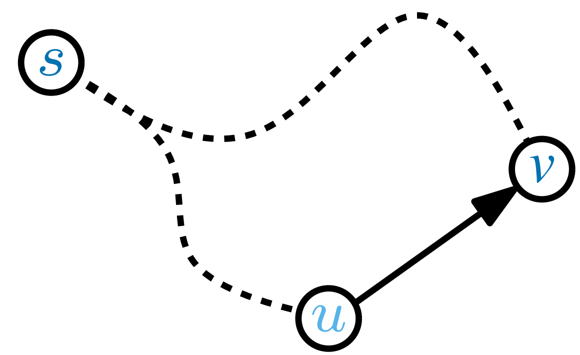
$\texttt{dist}(v)$ is the length of the shortest path between s and v, found so far

---

Most of the Bellman-Ford algorithm

MOSTOFBELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    If dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)
```



The algorithm repeatedly asks, for each edge (u, v)

"can I find a shorter route to v if I go via u?"

$(u, v) \in E$ iff there is an edge *from* u *to* v

$\texttt{weight}(u, v)$ is the weight of the edge from u to v

$\texttt{dist}(v)$ is the length of the shortest path between s and v, found so far

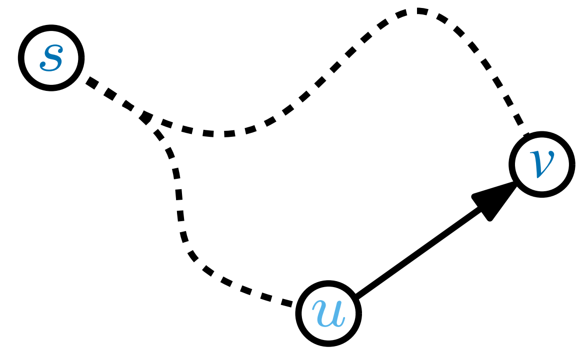
---

Most of the Bellman-Ford algorithm

MostOfBellman-Ford(s)

For all v, set $\texttt{dist}(v) = \infty$
set $\texttt{dist}(s) = 0$
For $i = 1, 2, \ldots, |V|$,
  For every edge $(u, v) \in E$
    If $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
      $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

This is called Relaxing edge (u, v)



The algorithm repeatedly asks, for each edge (u, v)

"can I find a shorter route to v if I go via u?"

$(u, v) \in E$ iff there is an edge *from* u *to* v

$\texttt{weight}(u, v)$ is the weight of the edge from u to v

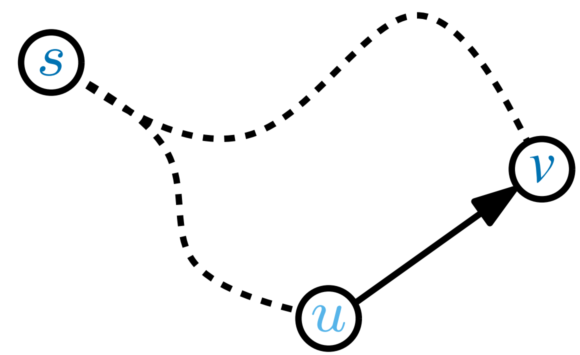
$\texttt{dist}(v)$ is the length of the shortest path between s and v, found so far

---

Most of the Bellman-Ford algorithm

MOSTOFBELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    If dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)
```



This is called RELAXING edge (u, v)

The algorithm repeatedly asks, for each edge (u, v)

"can I find a shorter route to v if I go via u?"

$(u, v) \in E$ iff there is an edge *from* u *to* v

$\texttt{weight}(u, v)$ is the weight of the edge from u to v

$\texttt{dist}(v)$ is the length of the shortest path between s and v, found so far

---

Claim When the MOSTOFBELLMAN-FORD algorithm terminates,

for each vertex v, $\texttt{dist}(v)$ is the length of the shortest path between s and v

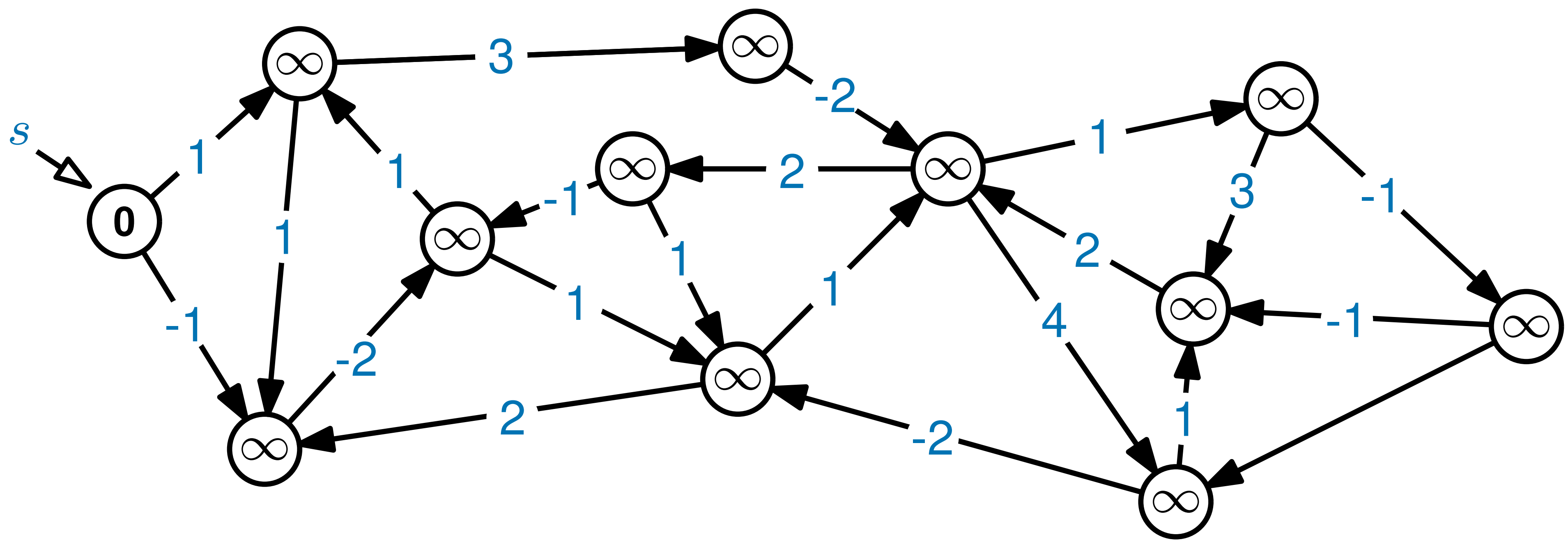
(assuming there are no-negative weight cycles)

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

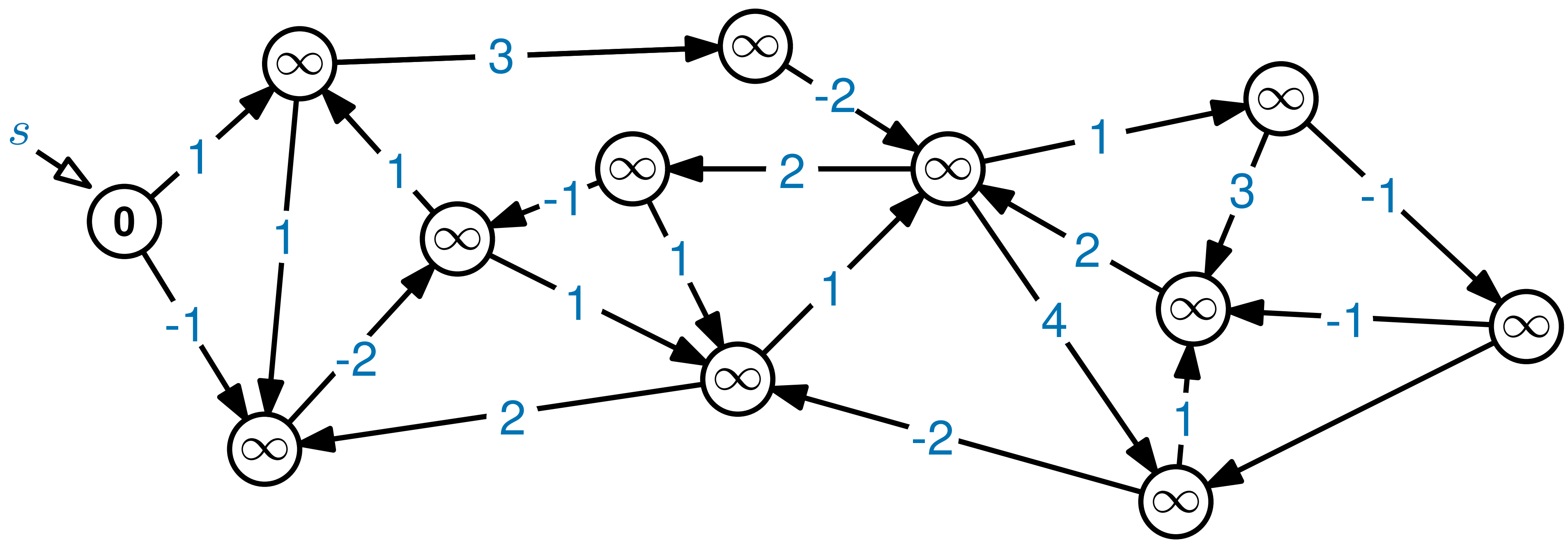
RELAX(u, v)

```
if dist(v) > dist(u) + weight(u,v)
   dist(v) = dist(u) + weight(u,v)
```

- *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We start by setting $\texttt{dist}(s) = 0$ and every other $\texttt{dist}(v) = \infty\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

Relax(u, v)

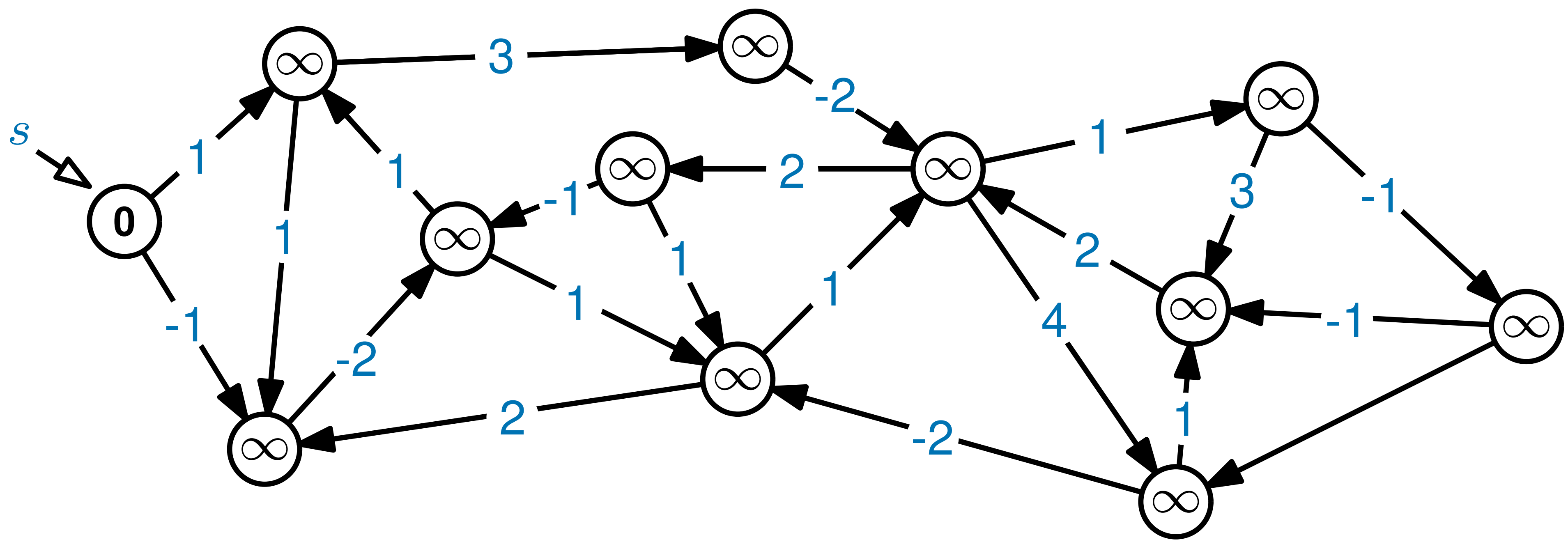
```
if dist(v) > dist(u) + weight(u,v)
   dist(v) = dist(u) + weight(u,v)
```

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

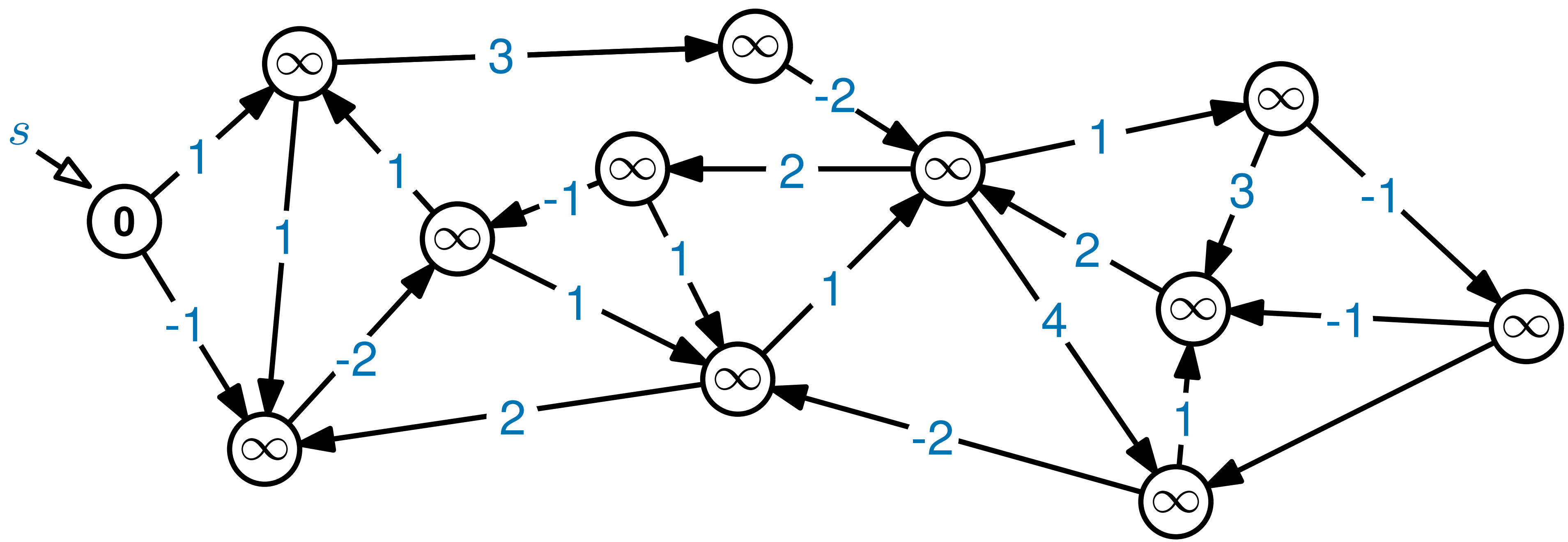
RELAX(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

$\texttt{if}\ \ \texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad \texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

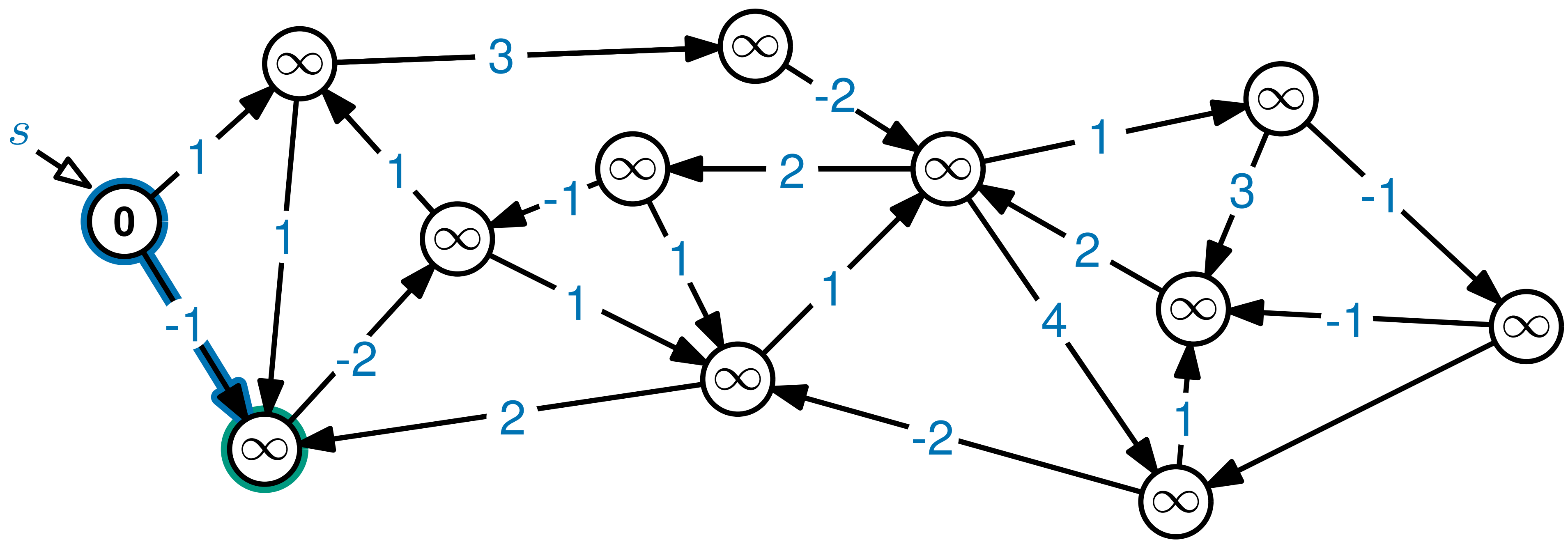
37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

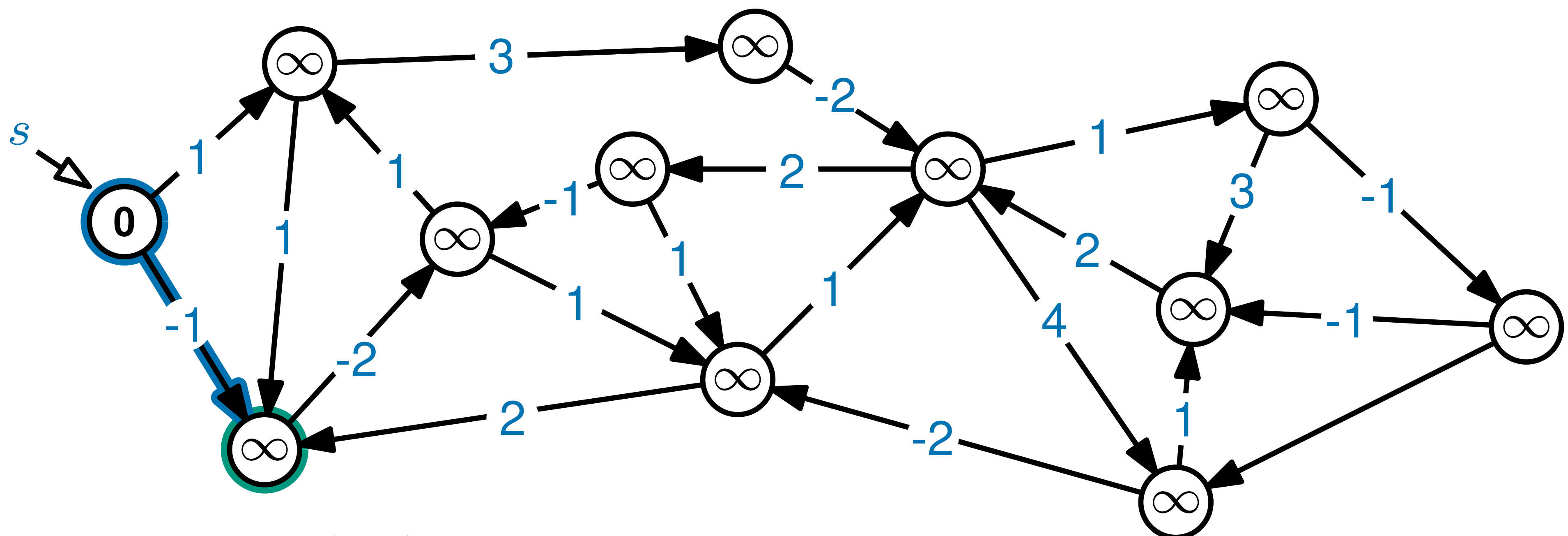
$\texttt{if dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$\infty > 0 + (-1)$

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

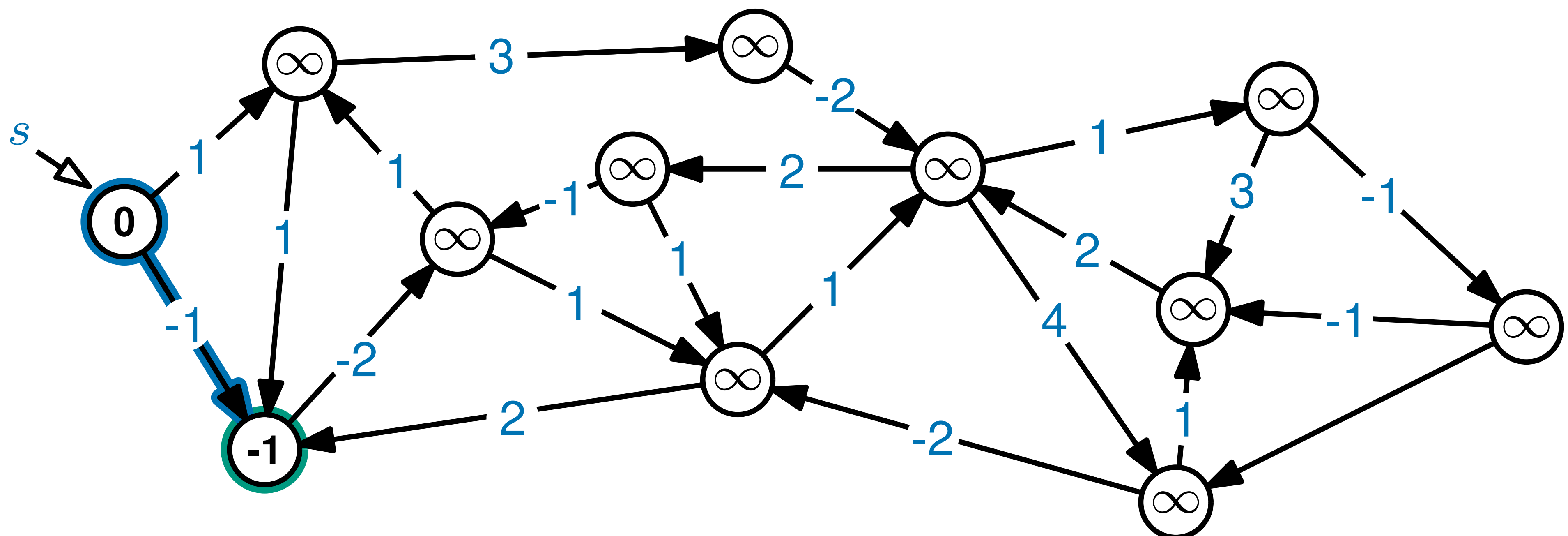
RELAX(u, v)
`if` $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$\infty > 0 + (-1)$

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

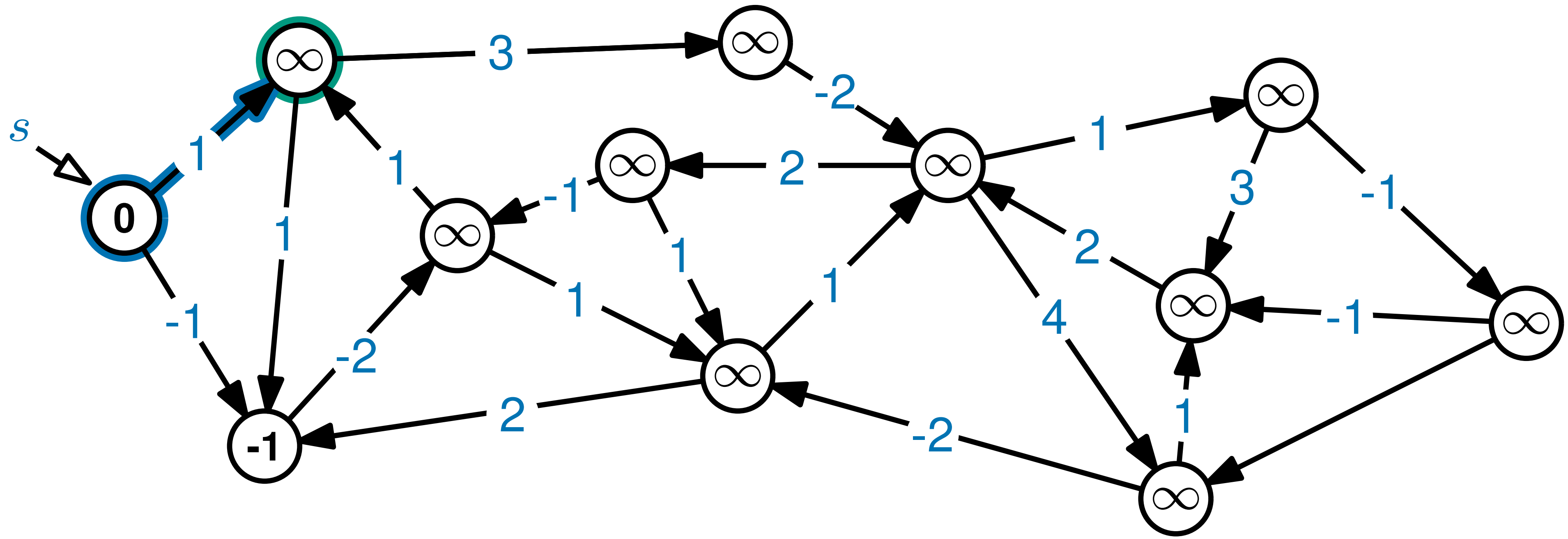
37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

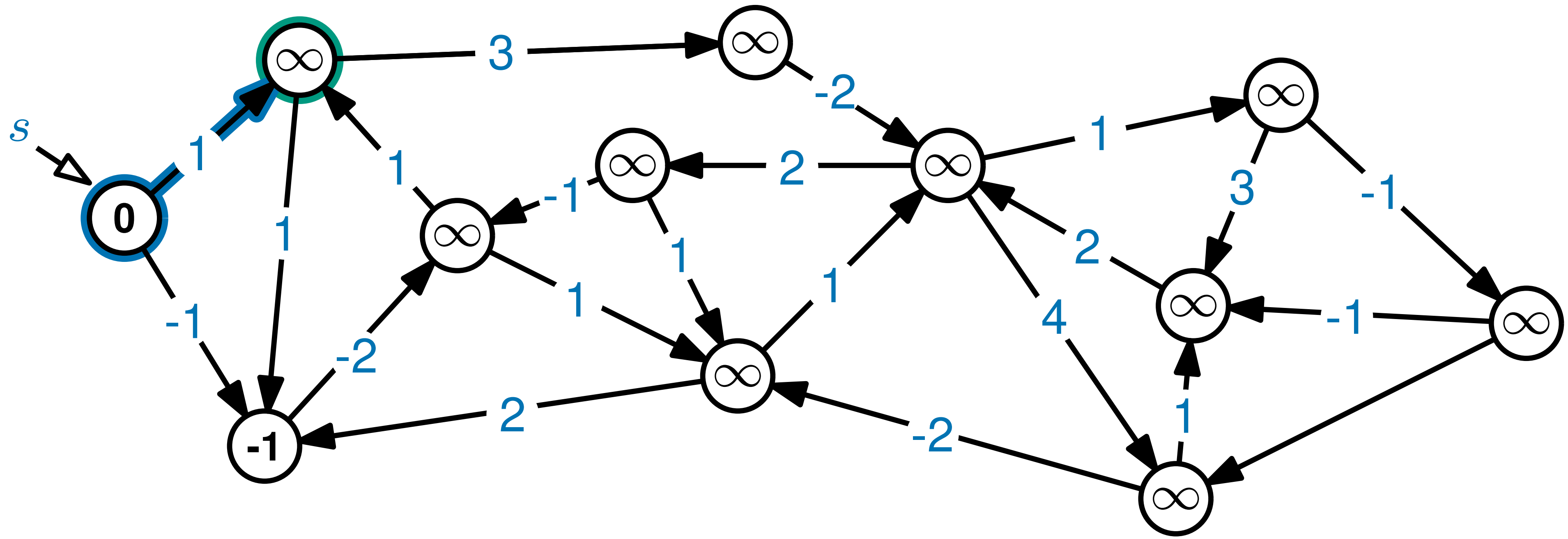
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$

$\infty > 0 + 1$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

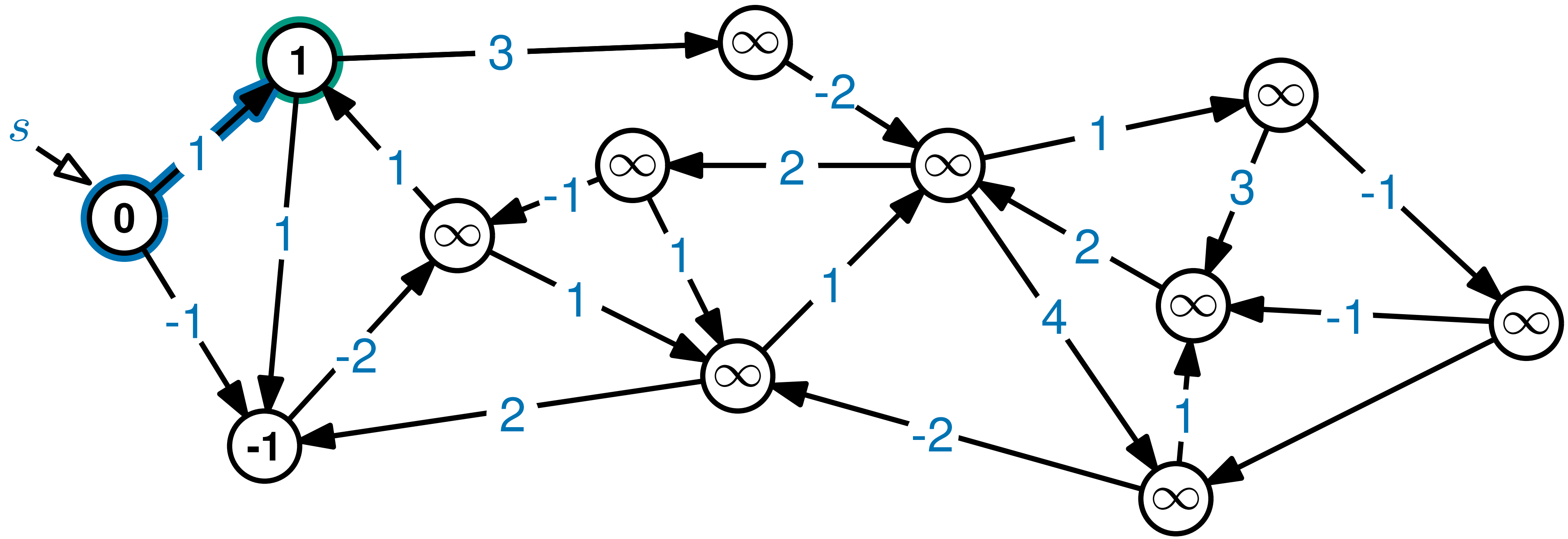
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$

$\infty > 0 + 1$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$

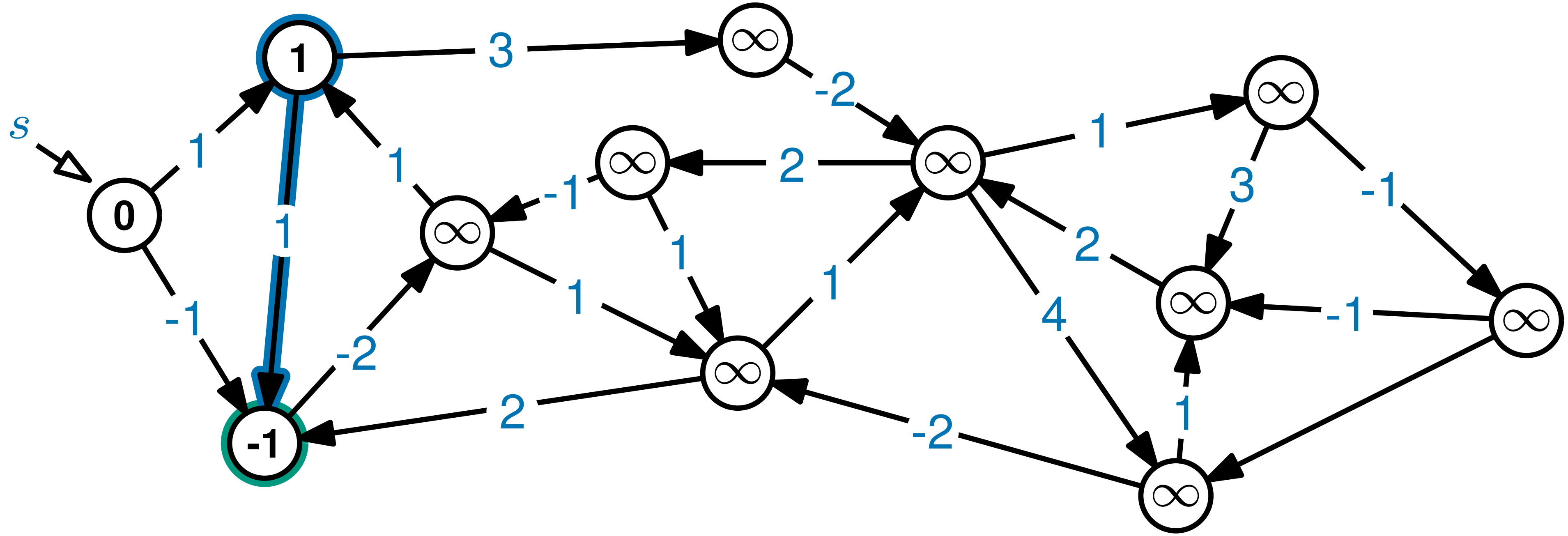
$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

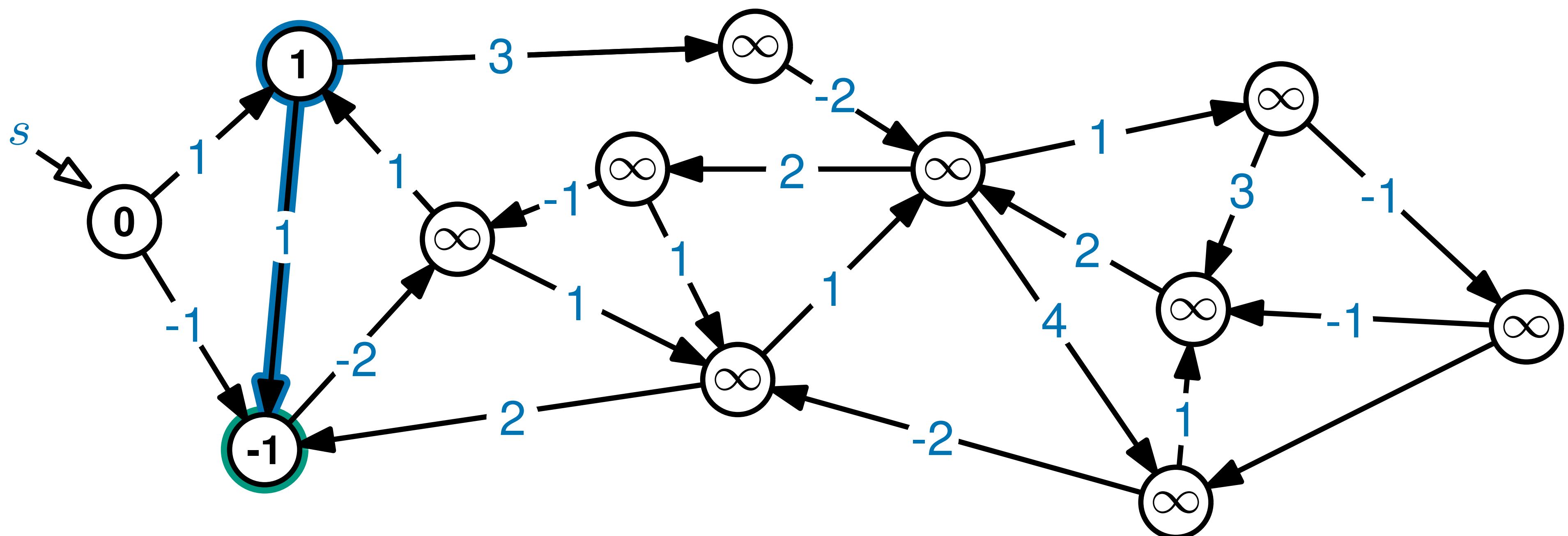
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

vertex v

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$-1 < 1 + 1$

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

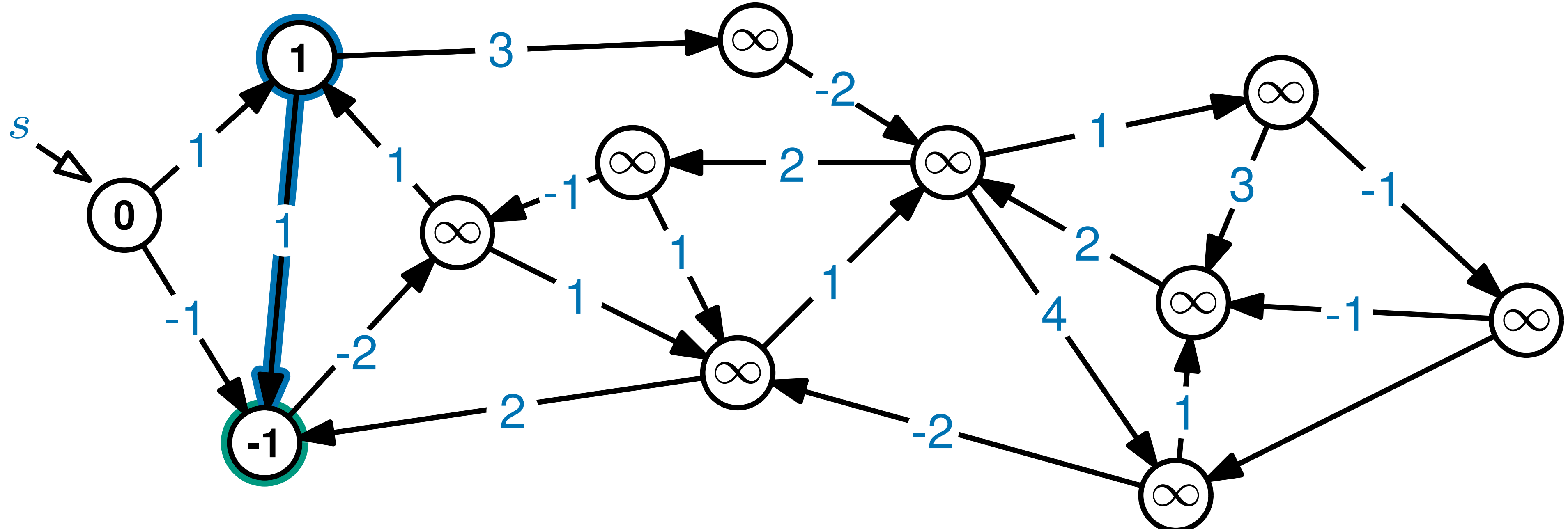
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

$\circled{37}$ ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$-1 < 1 + 1$

(so we don't change $\texttt{dist}$*)*

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

```
if dist(v) > dist(u) + weight(u,v)
  dist(v) = dist(u) + weight(u,v)
```

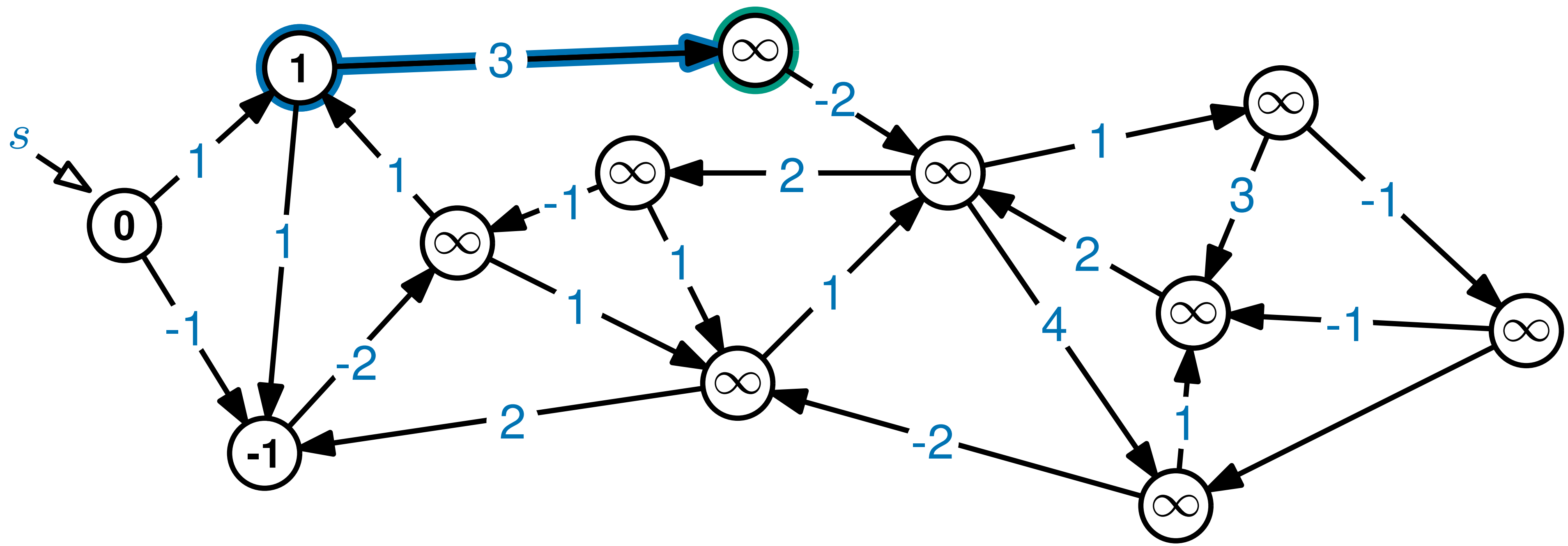
37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

```
if  dist(v) > dist(u) + weight(u,v)
    dist(v) = dist(u) + weight(u,v)
```

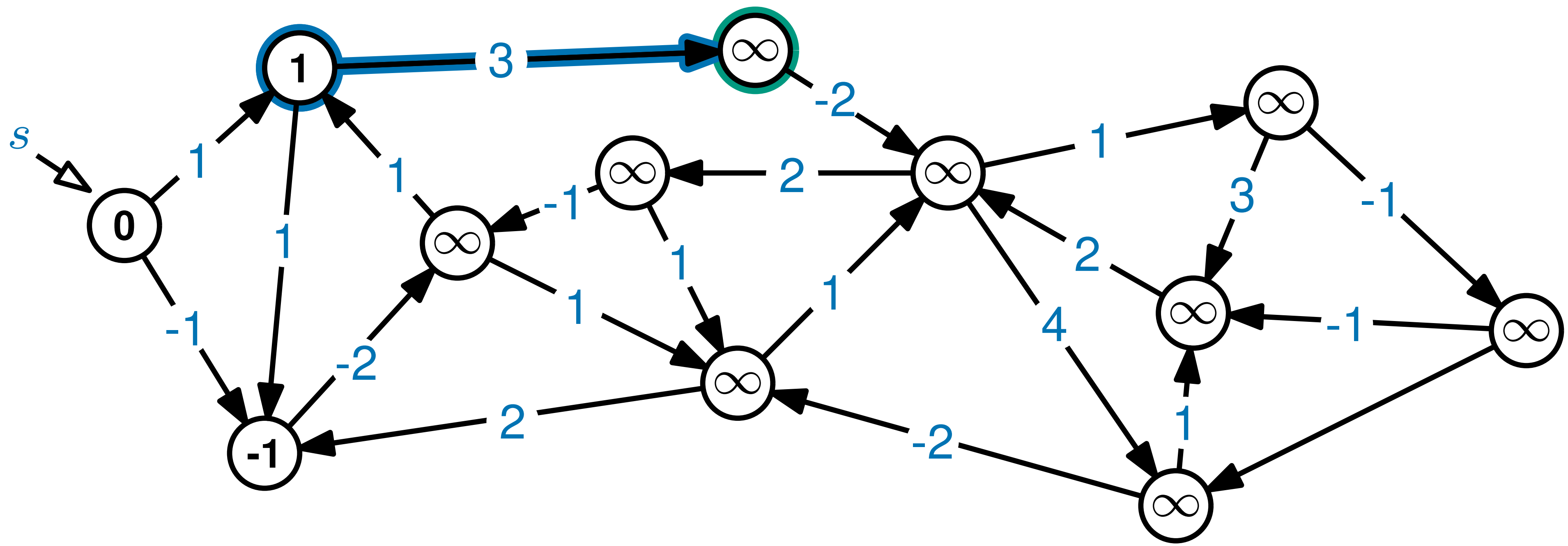
37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1\ldots$

$\infty > 1 + 3$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

Relax(u, v)

```
if dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

37 ← vertex v

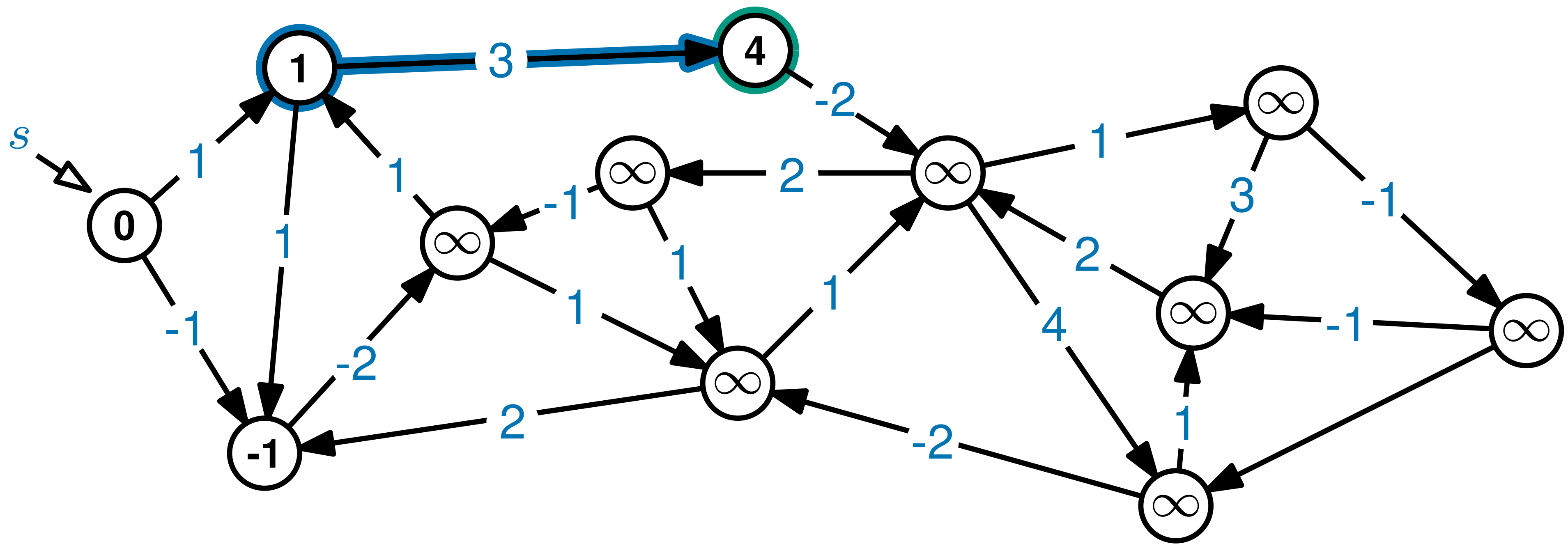
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$

$$\infty > 1 + 3$$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

RELAX(u, v)

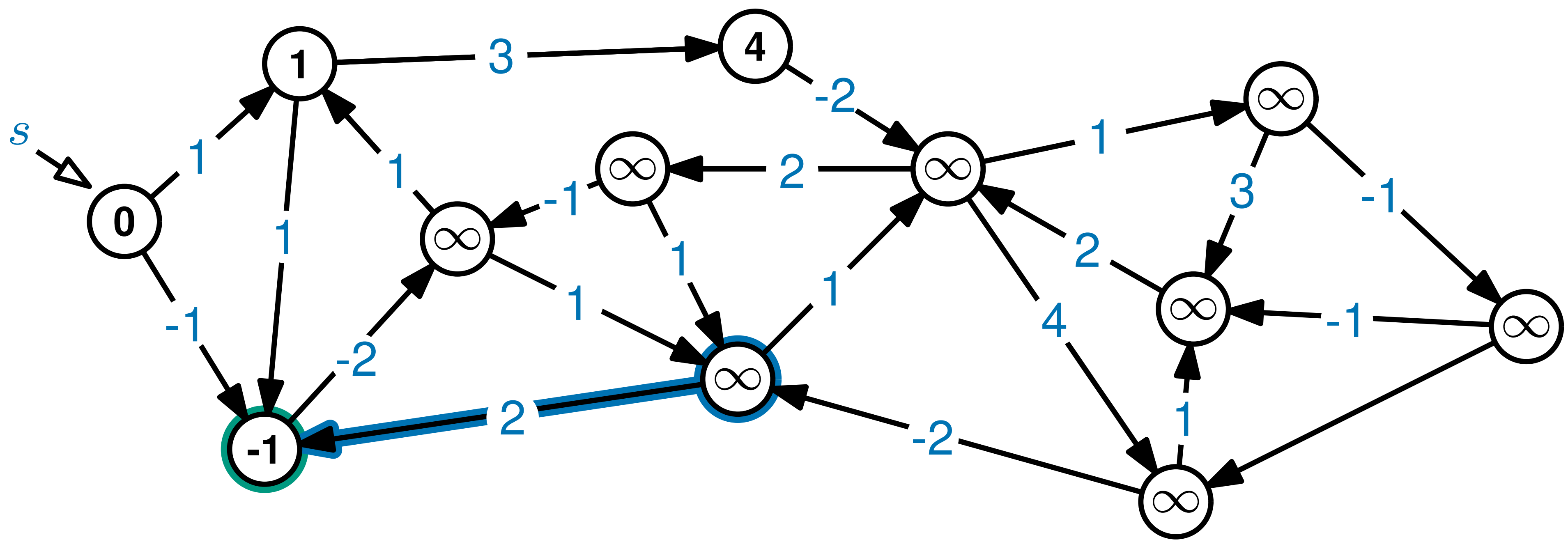
```
if dist(v) > dist(u) + weight(u, v)
   dist(v) = dist(u) + weight(u, v)
```

$\text{dist}(v)$ - *the length of the best path between s and v, found so far*

vertex v

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1 \ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

Relax(u, v)

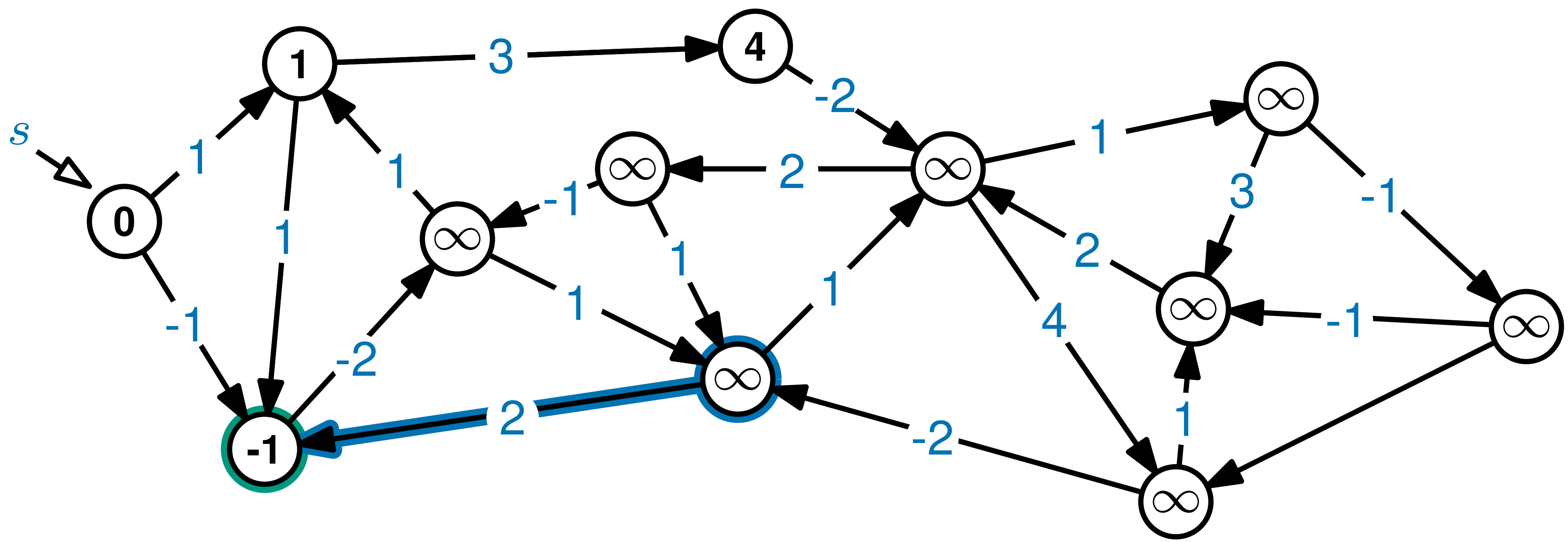
```
if  dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

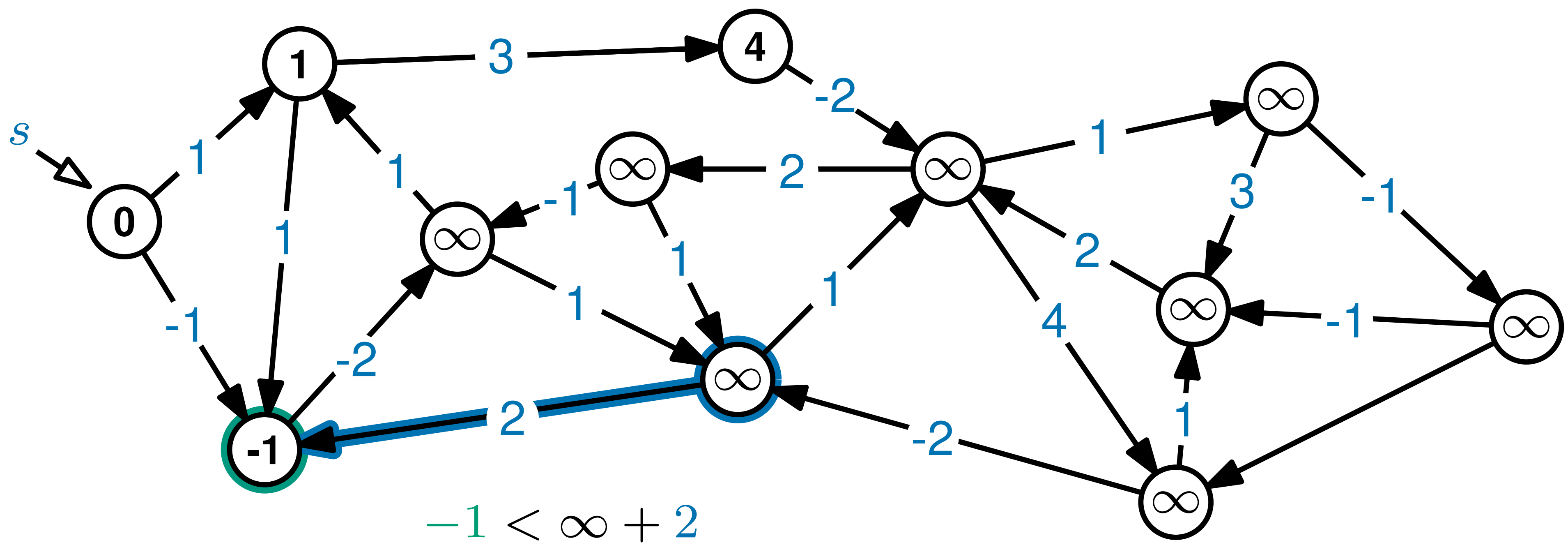
```
if dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$-1 < \infty + 2$

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

```
if dist(v) > dist(u) + weight(u,v)
   dist(v) = dist(u) + weight(u,v)
```

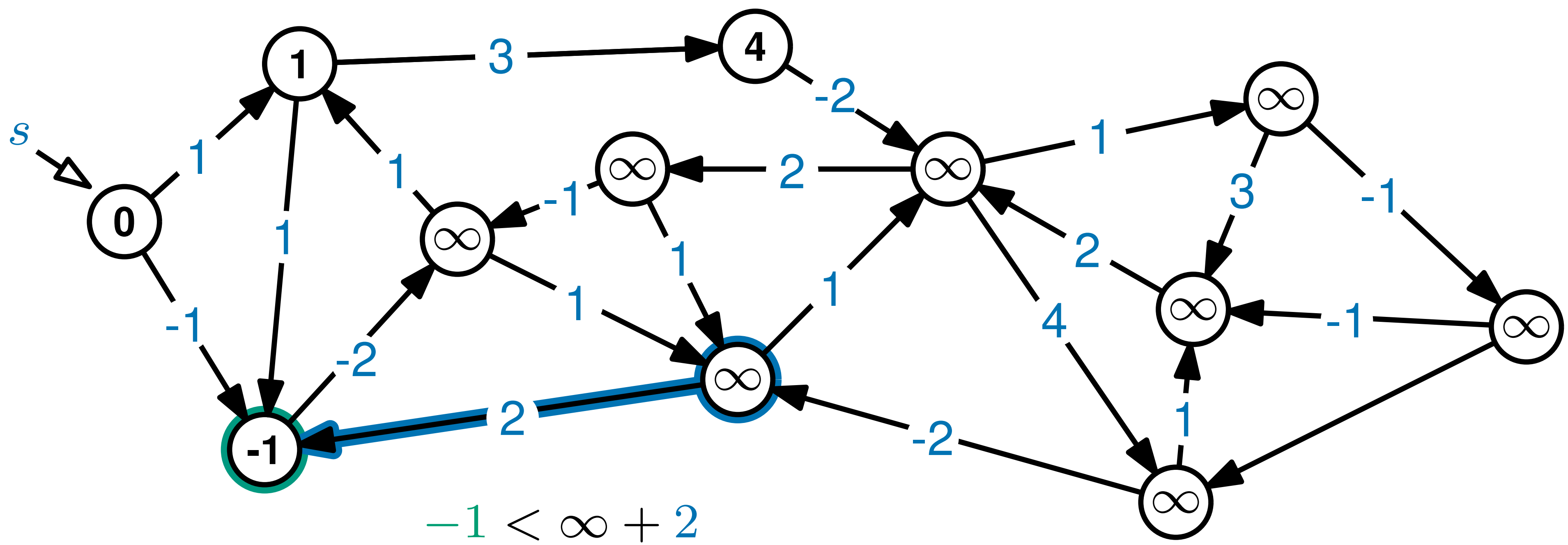
vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$



$-1 < \infty + 2$

(so we don't change $\texttt{dist}$*)*

MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

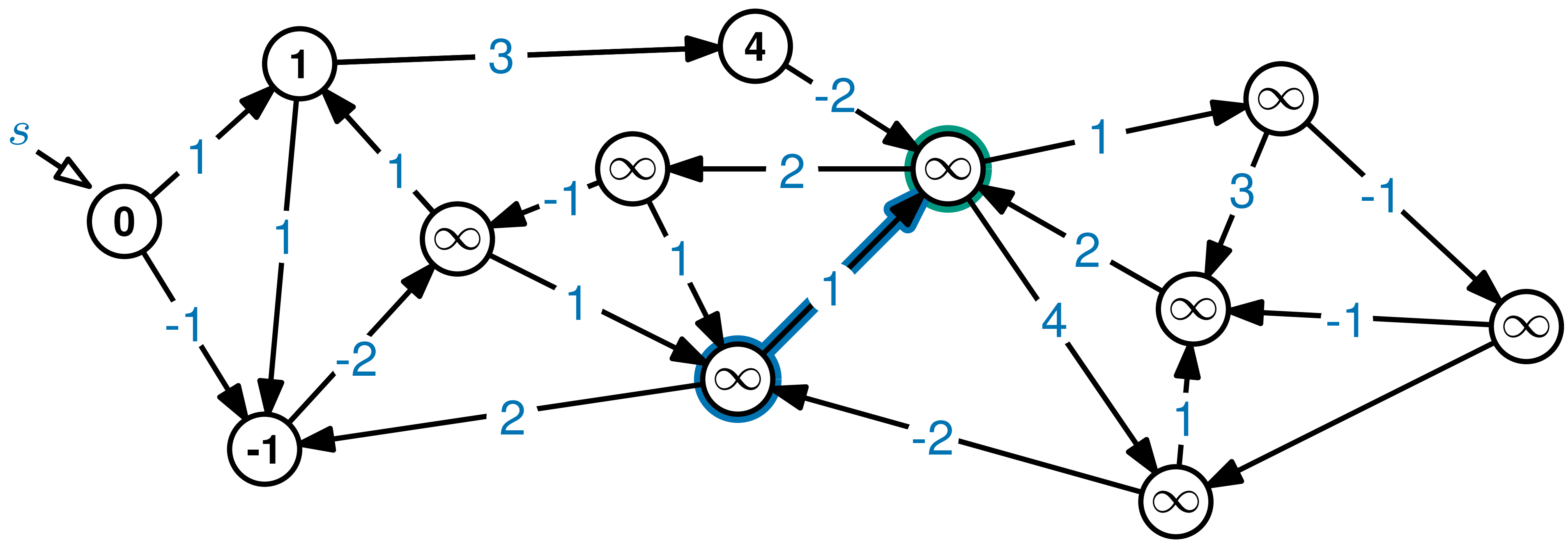
$\text{dist}(v)$ - *the length of the best path between s and v, found so far*

vertex v

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\dots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

$\texttt{if } \texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$

$\quad \texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

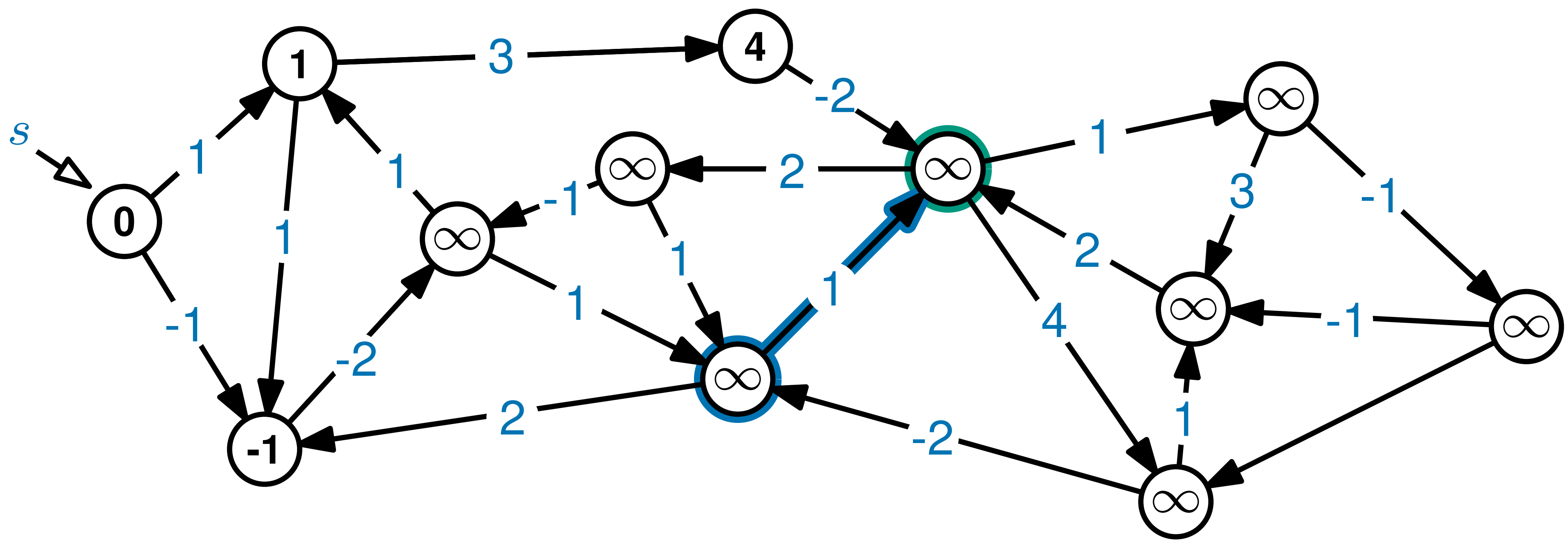
37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1 \ldots$

$\infty < \infty + 2$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

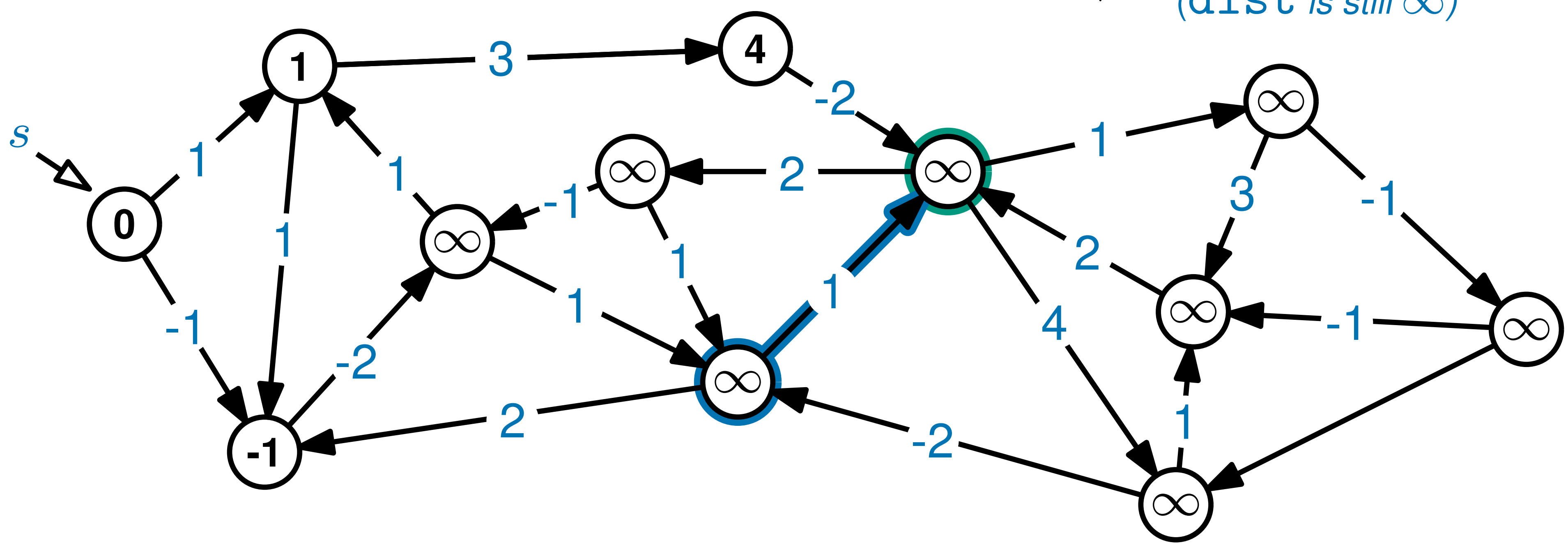
$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$

$\infty < \infty + 2$ *($\texttt{dist}$ is still ∞)*



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

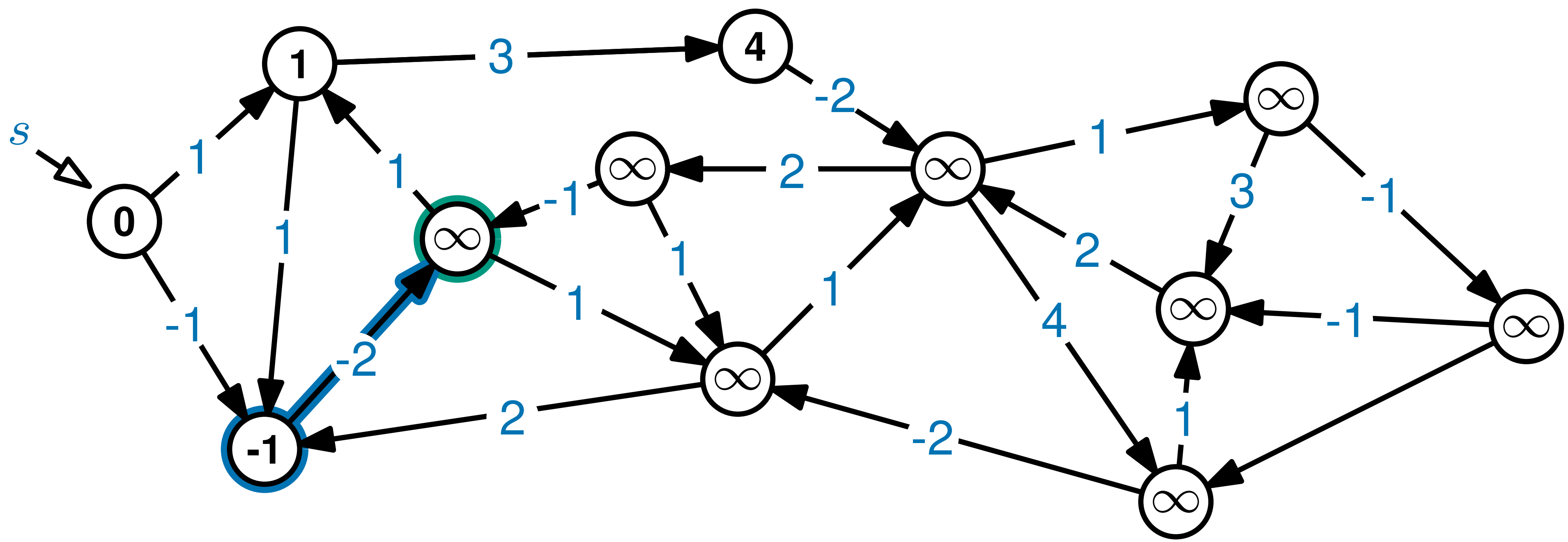
37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

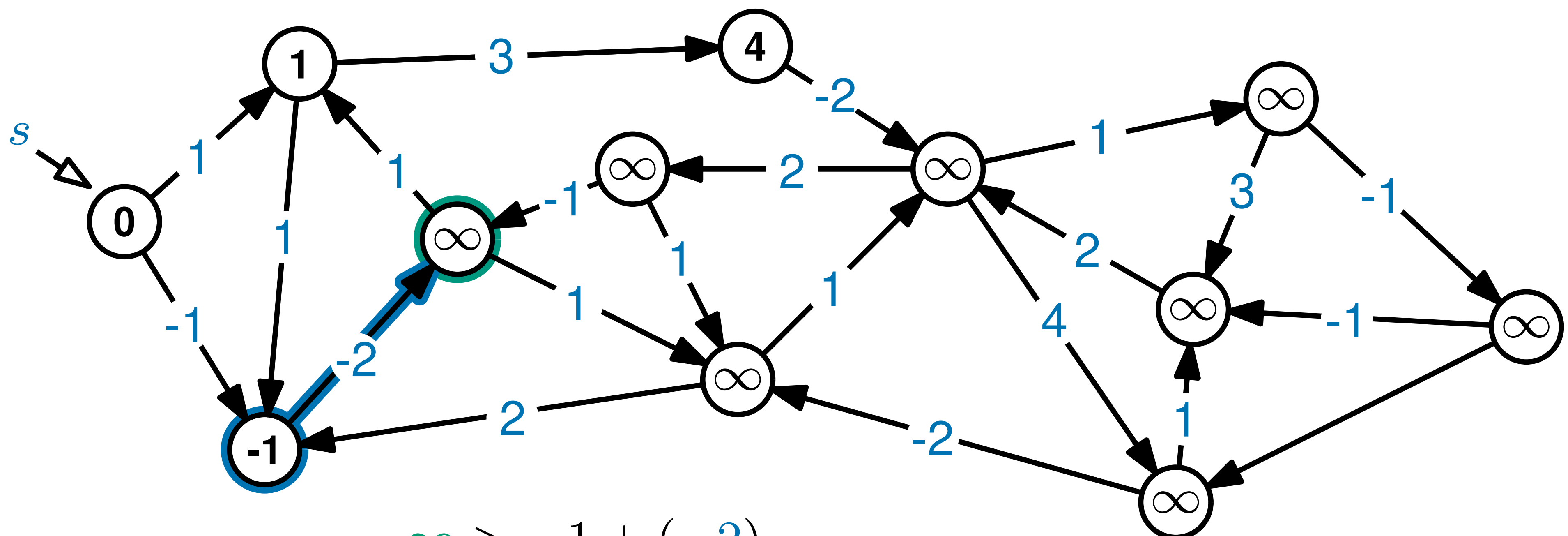
Relax(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$\infty > -1 + (-2)$

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

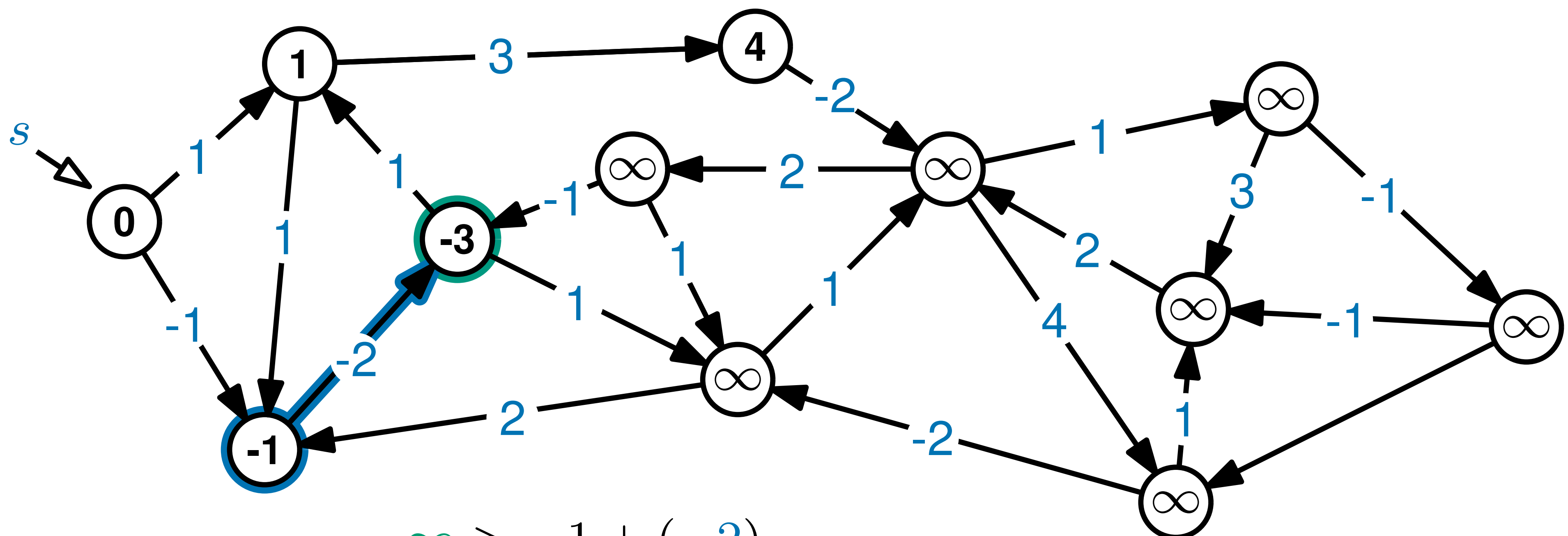
RELAX(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



$\infty > -1 + (-2)$

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

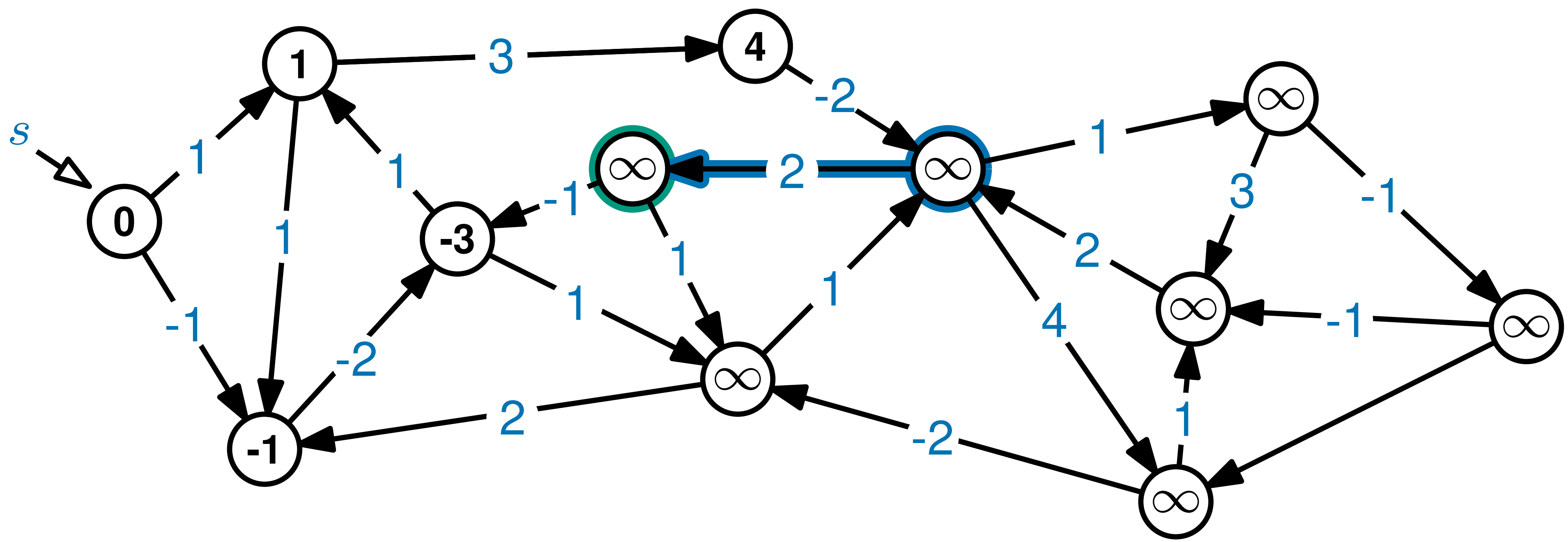
37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

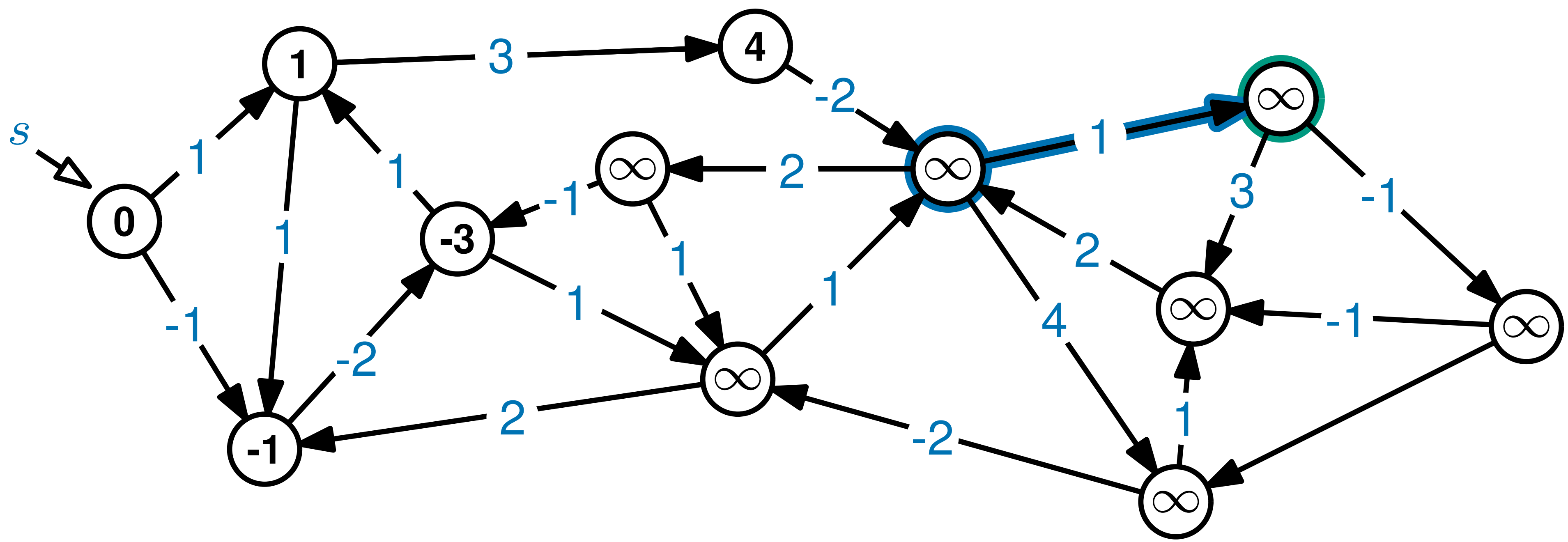
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

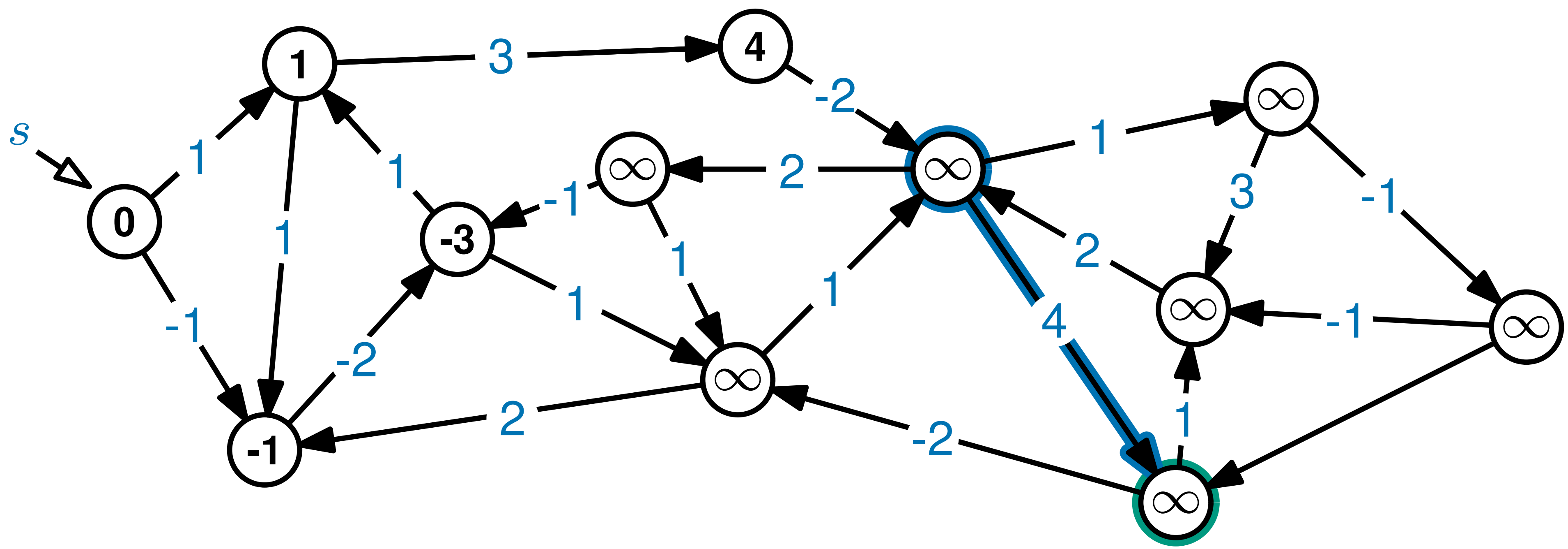
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1 \ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

$\texttt{if dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad \texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

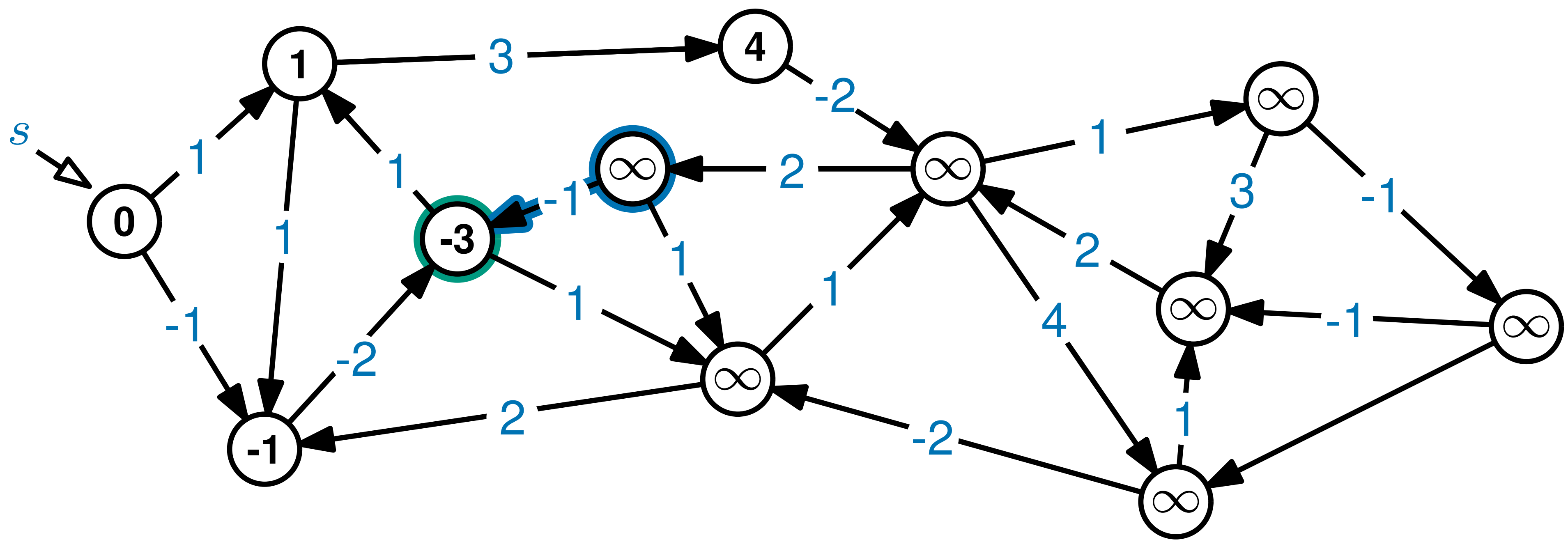
37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

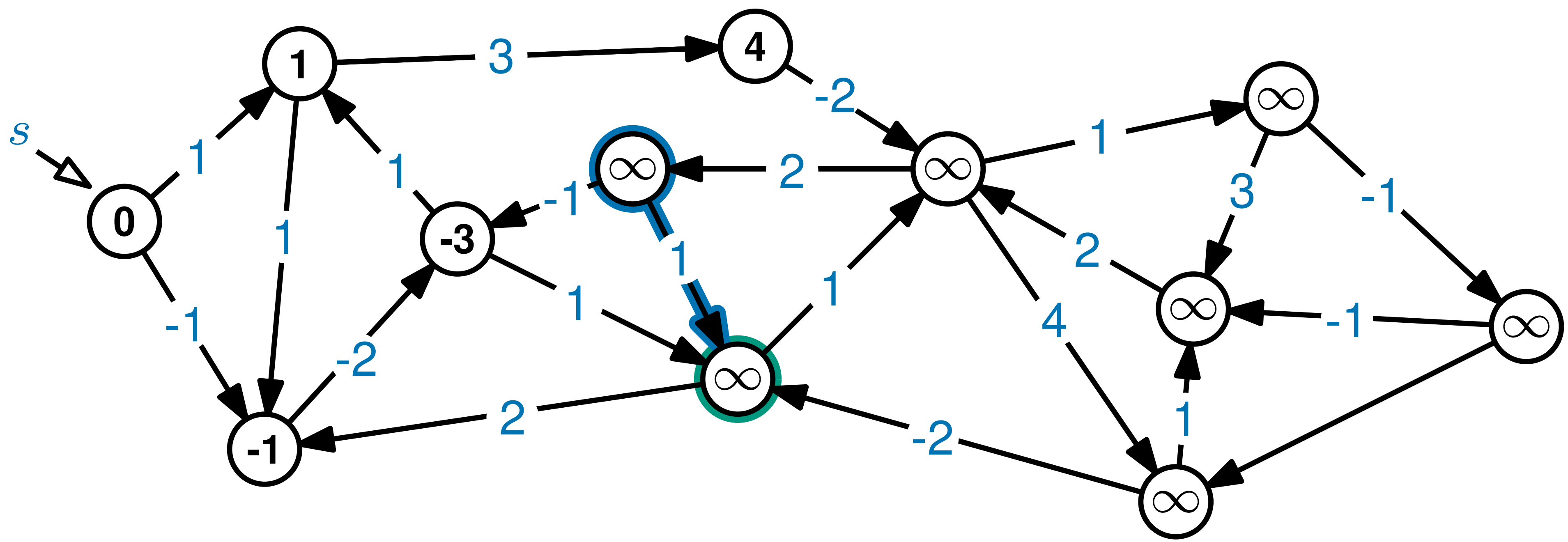
Relax(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

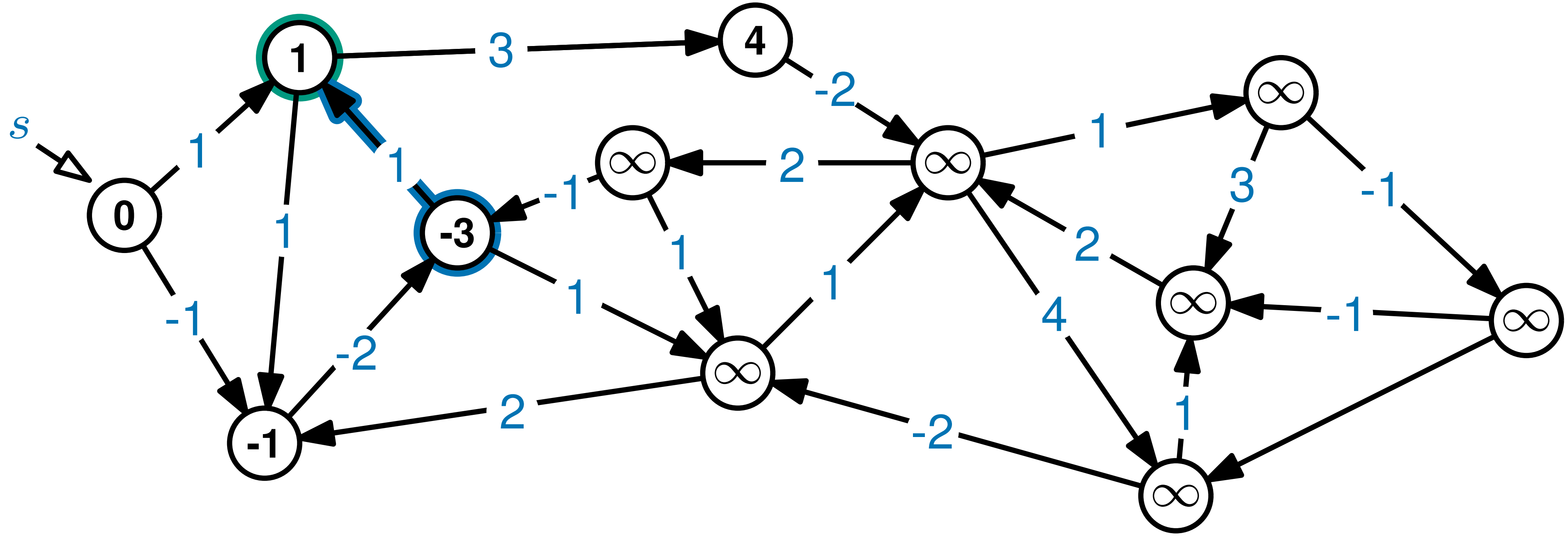
```
if  dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

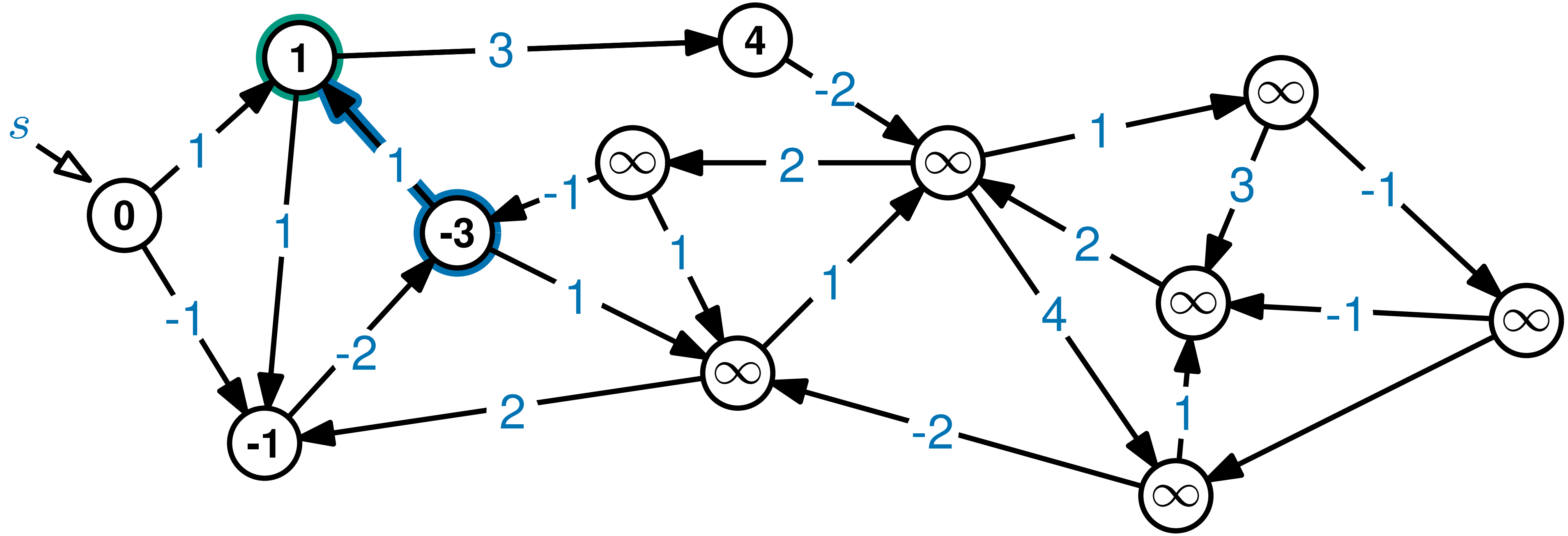
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

We now start iteration $1\ldots$

$$1 > -3 + 1$$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

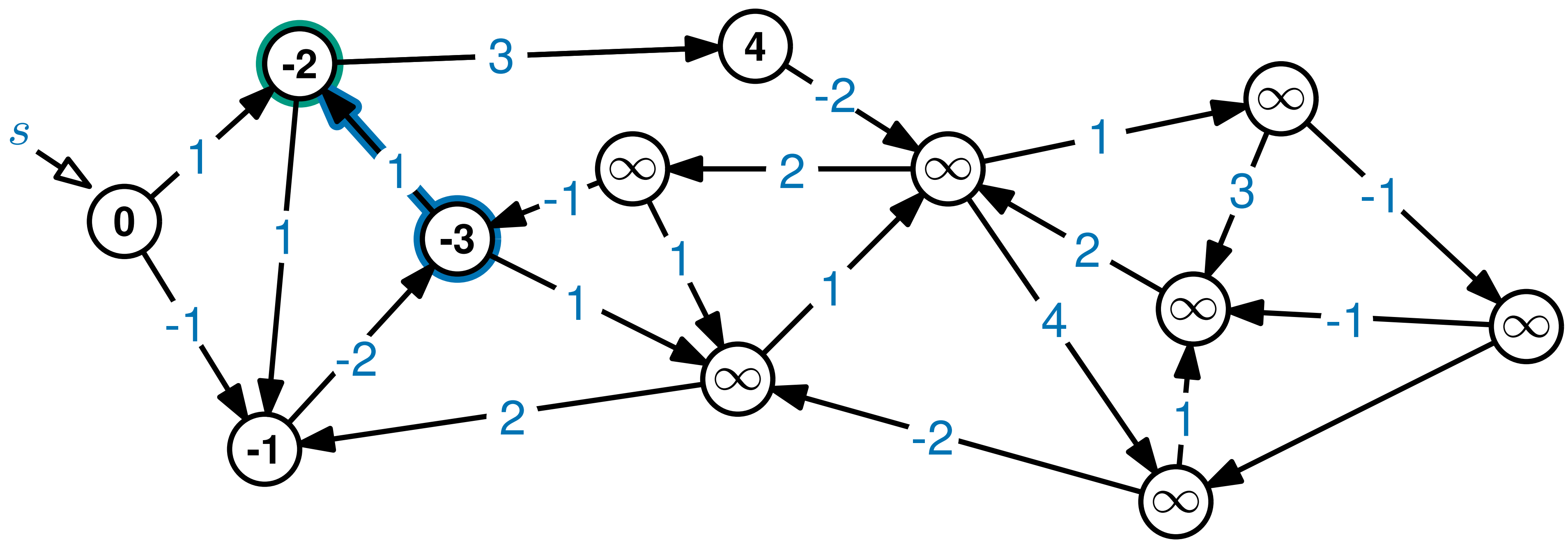
37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$

$1 > -3 + 1$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

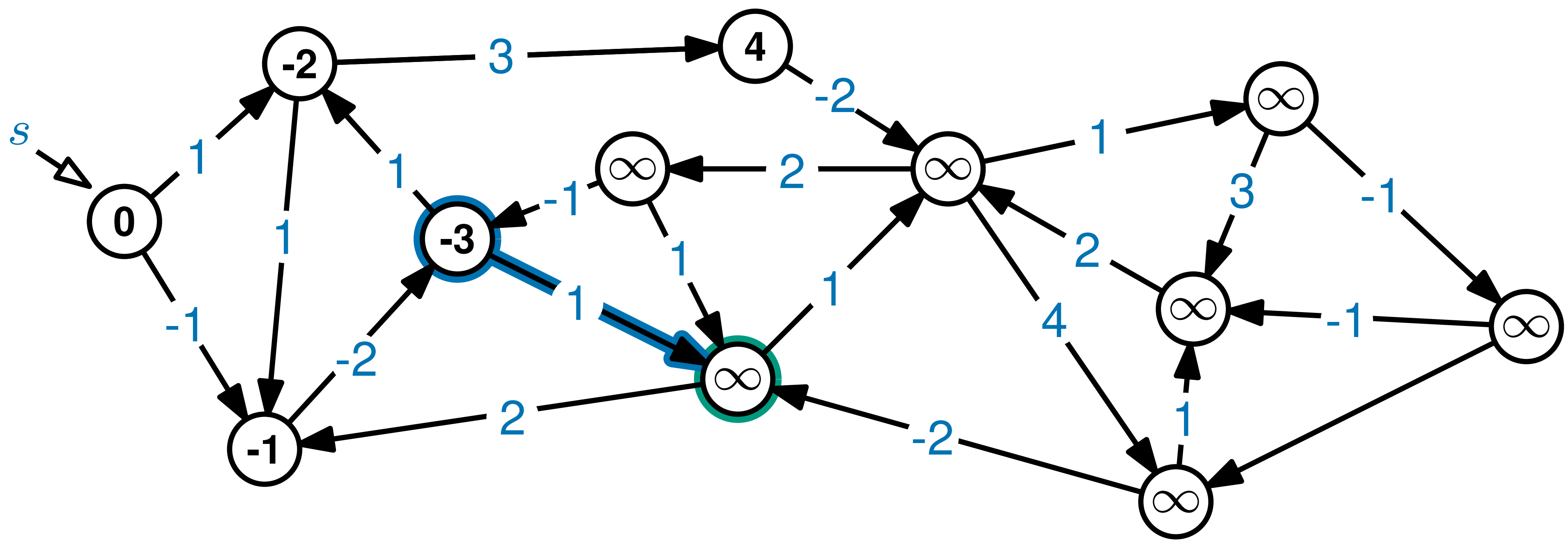
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

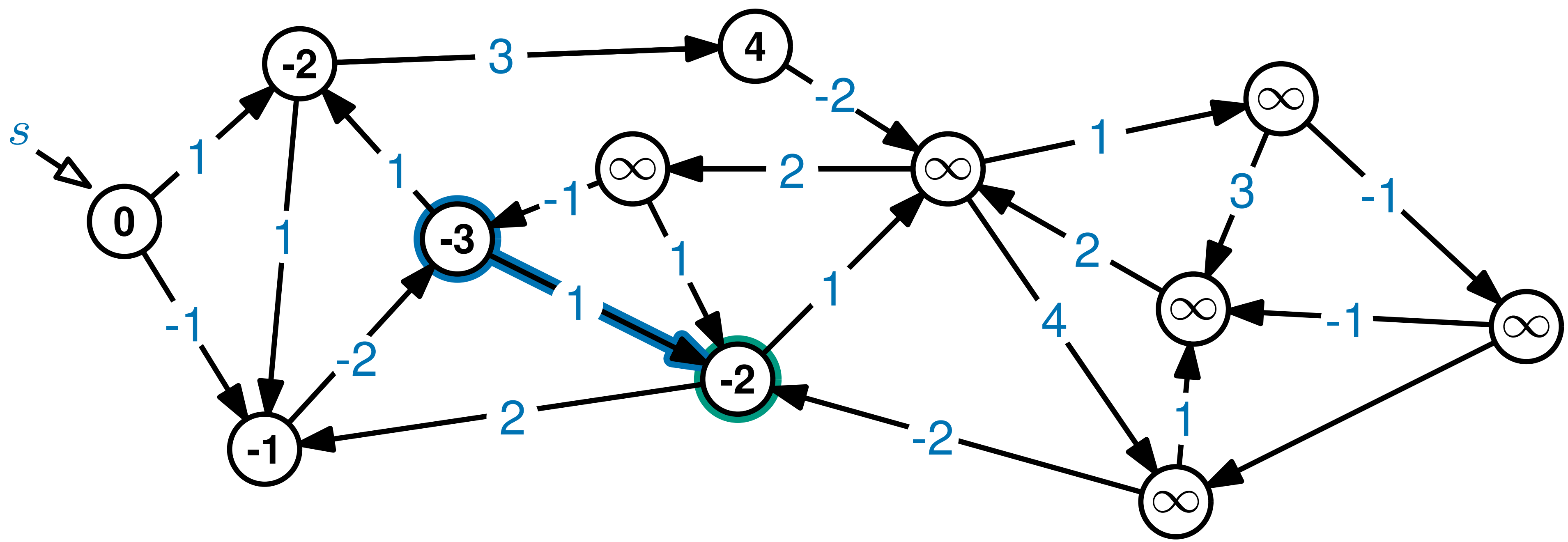
$\textcircled{37}$ ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

```
if  dist(v) > dist(u) + weight(u,v)
    dist(v) = dist(u) + weight(u,v)
```

37 ← vertex v

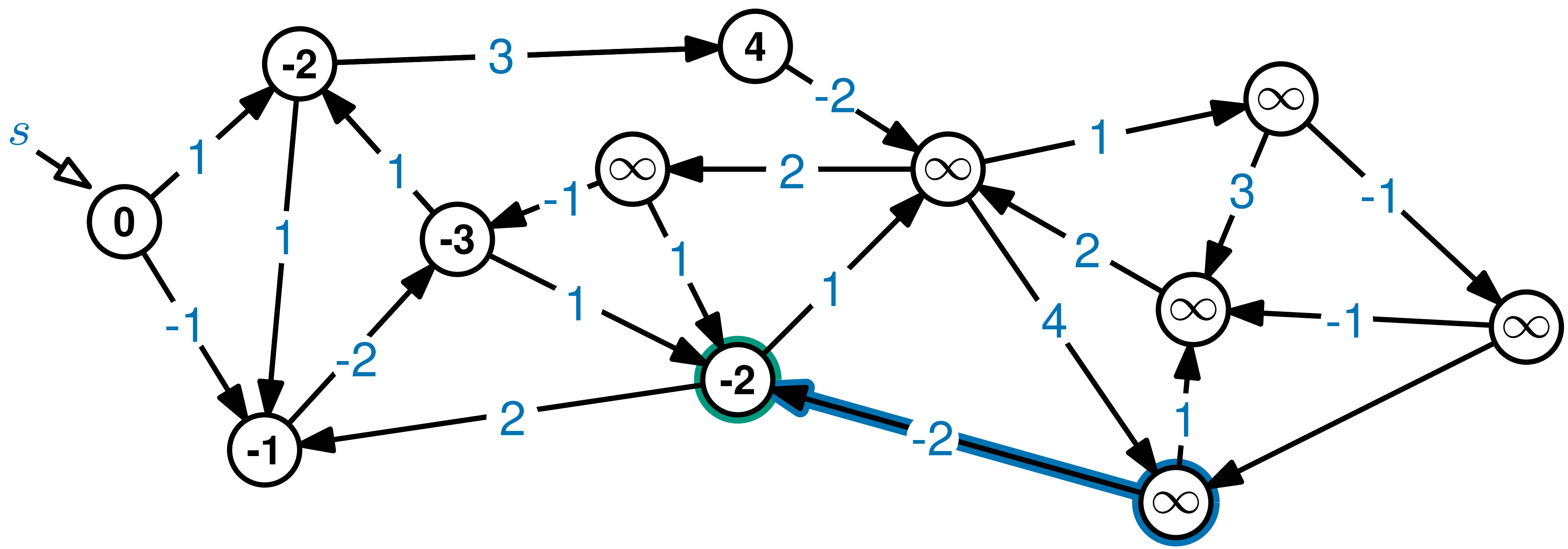
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

University of BRISTOL

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

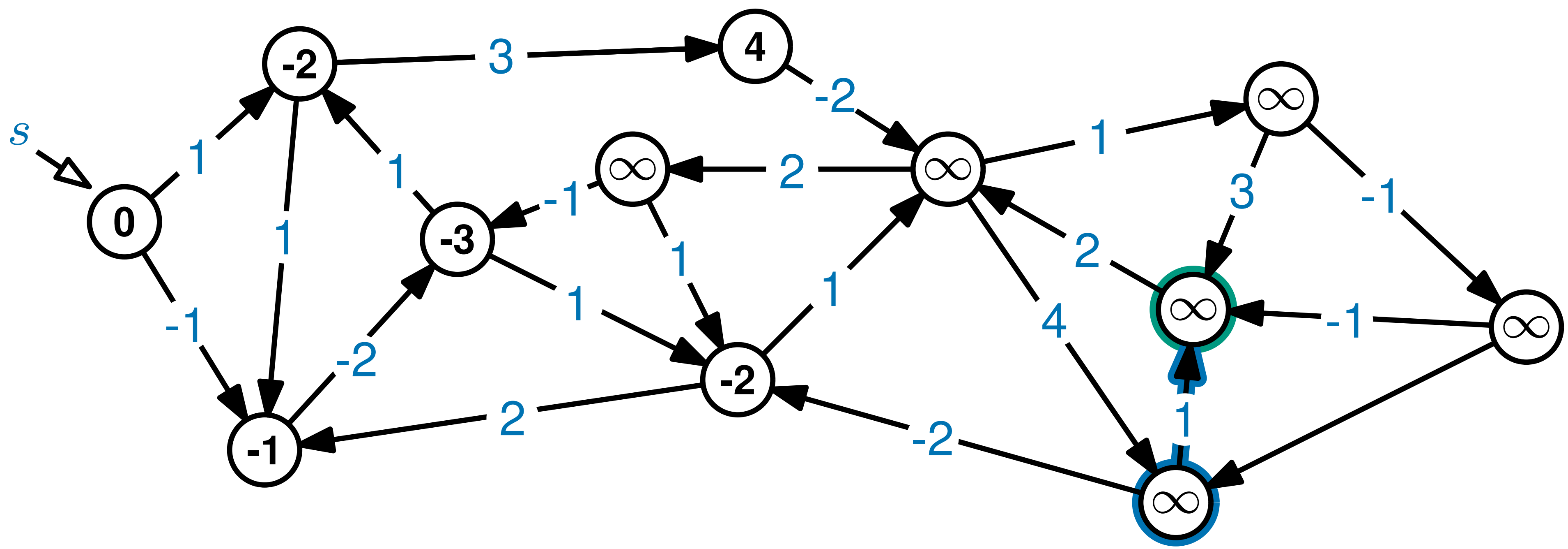
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\dots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

```
if dist(v) > dist(u) + weight(u, v)
   dist(v) = dist(u) + weight(u, v)
```

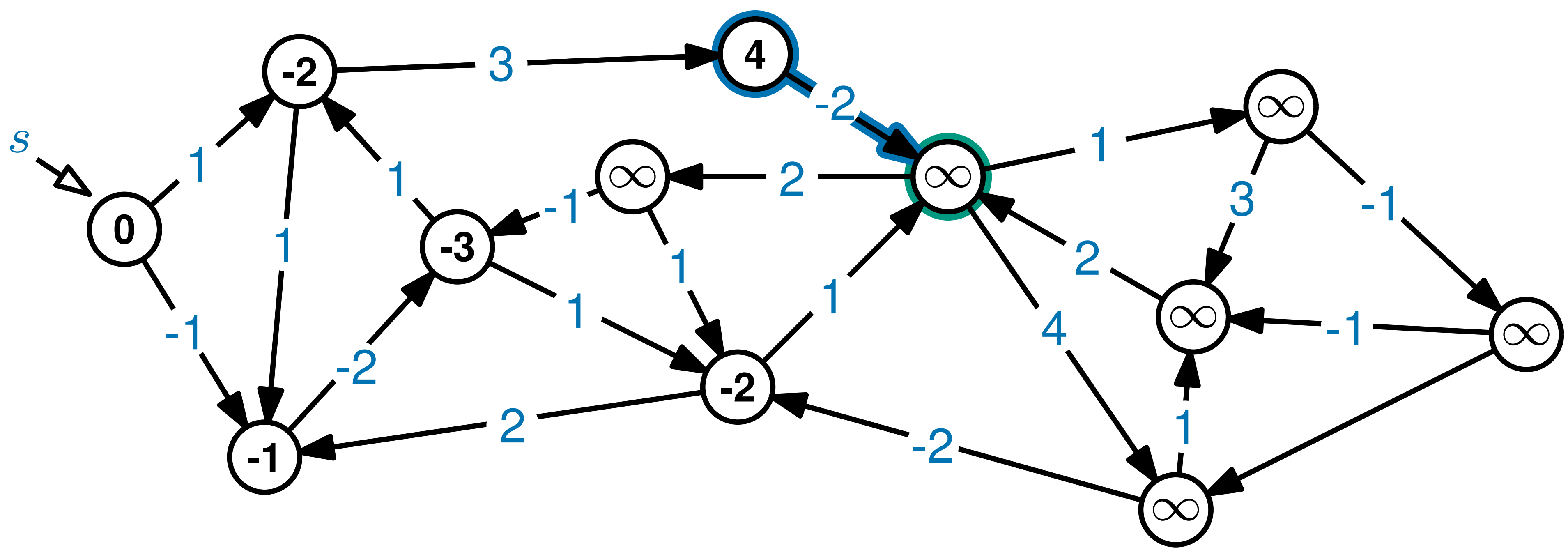
37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

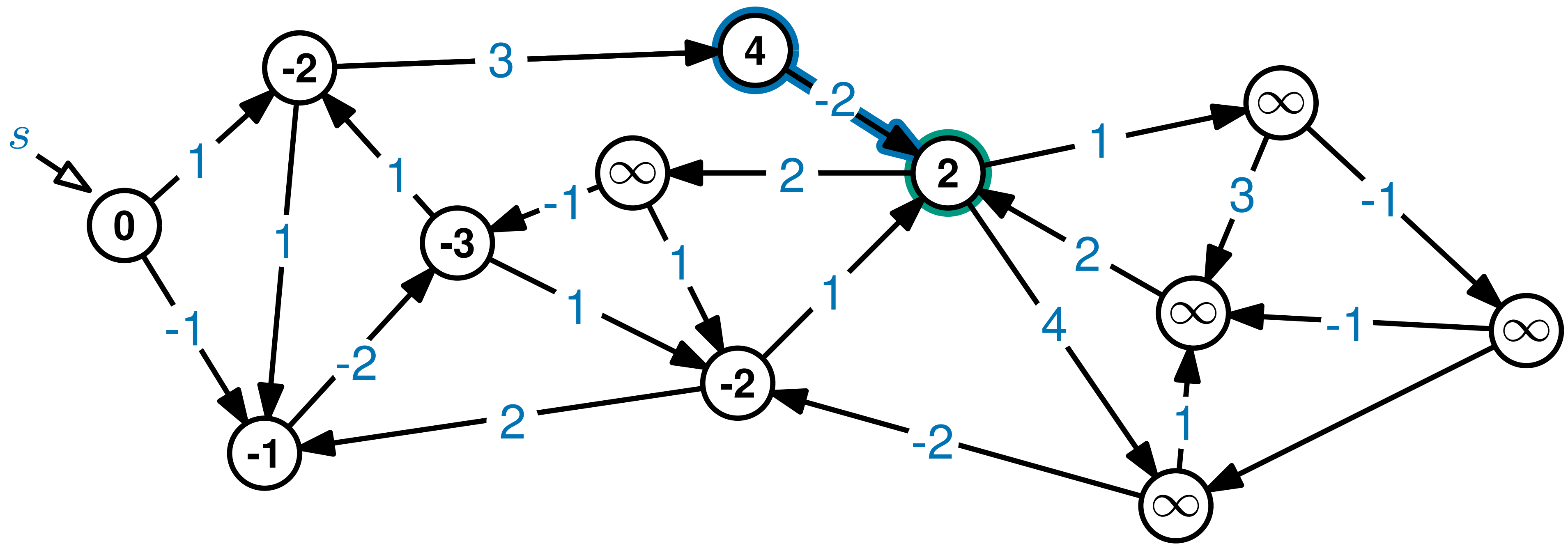
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

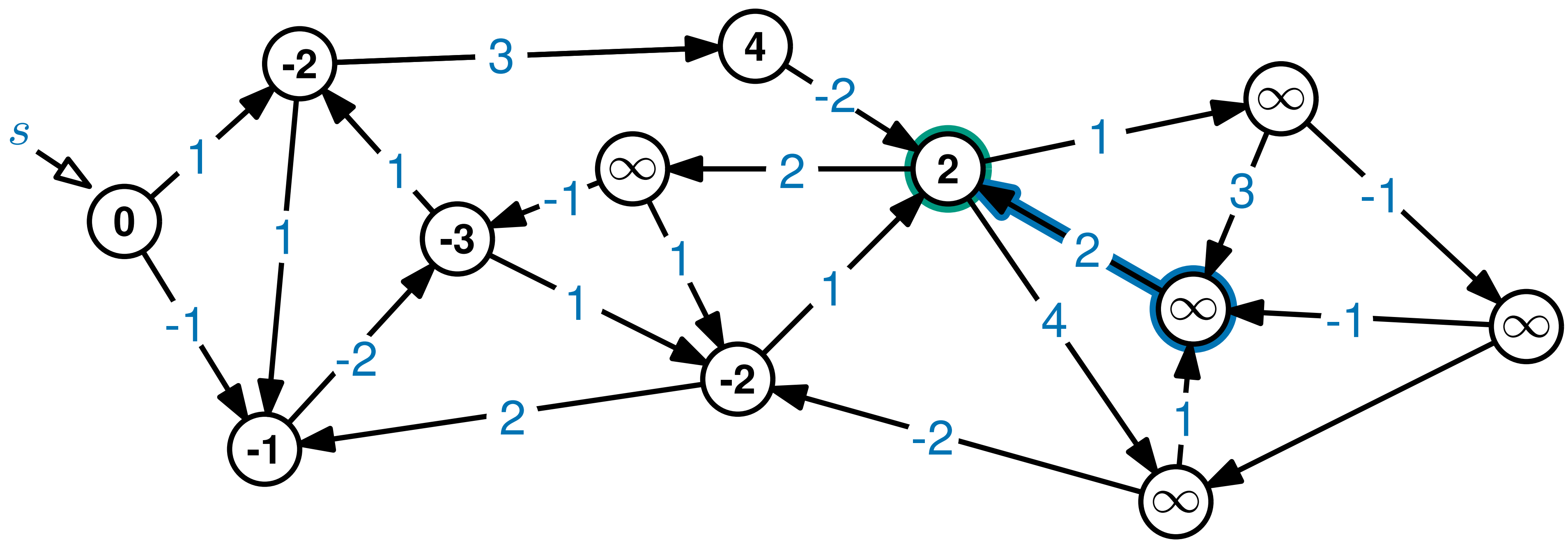
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1\ldots$



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

$\texttt{if}\ \ \texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

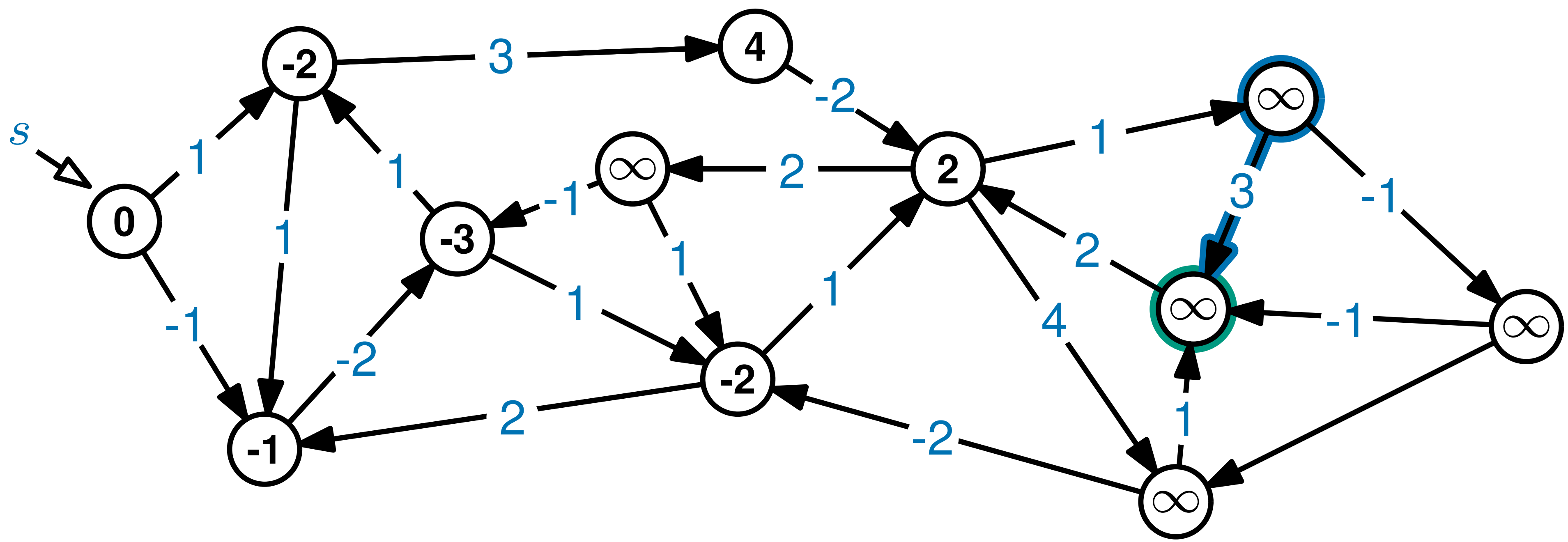
University of BRISTOL

37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

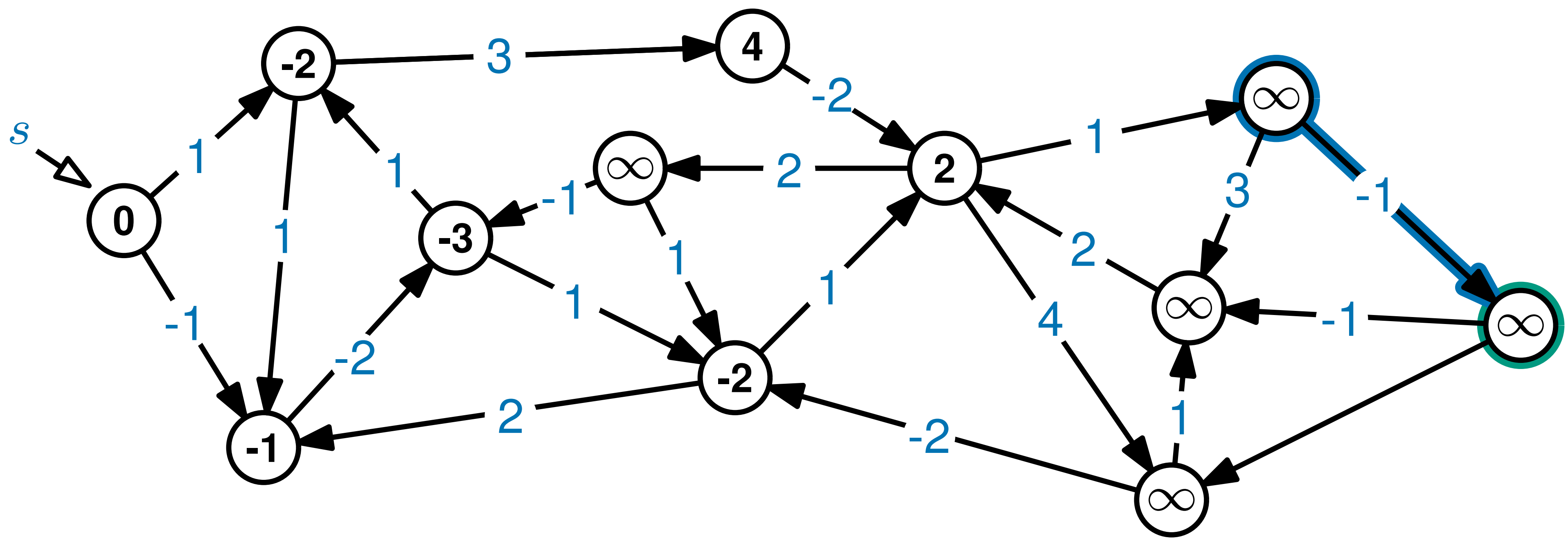
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)

We now start iteration $1\ldots$



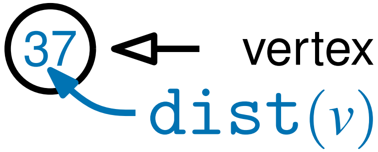
MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

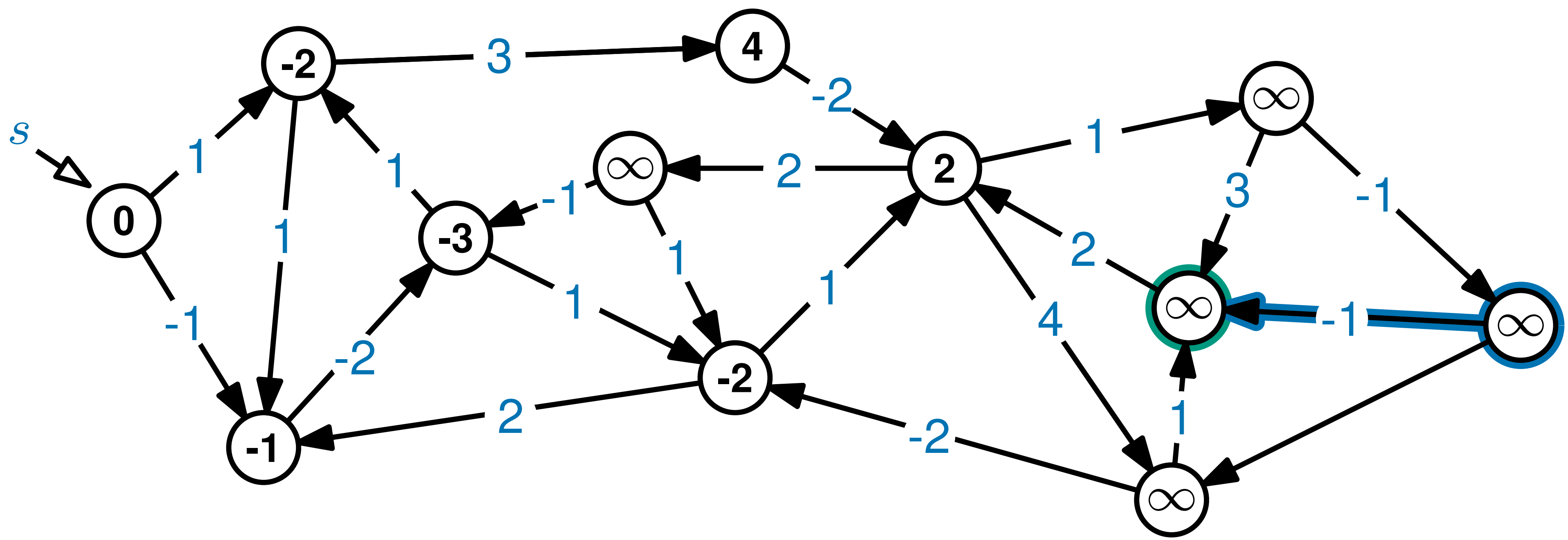
```
if dist(v) > dist(u) + weight(u,v)
   dist(v) = dist(u) + weight(u,v)
```



We're going to simulate

MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

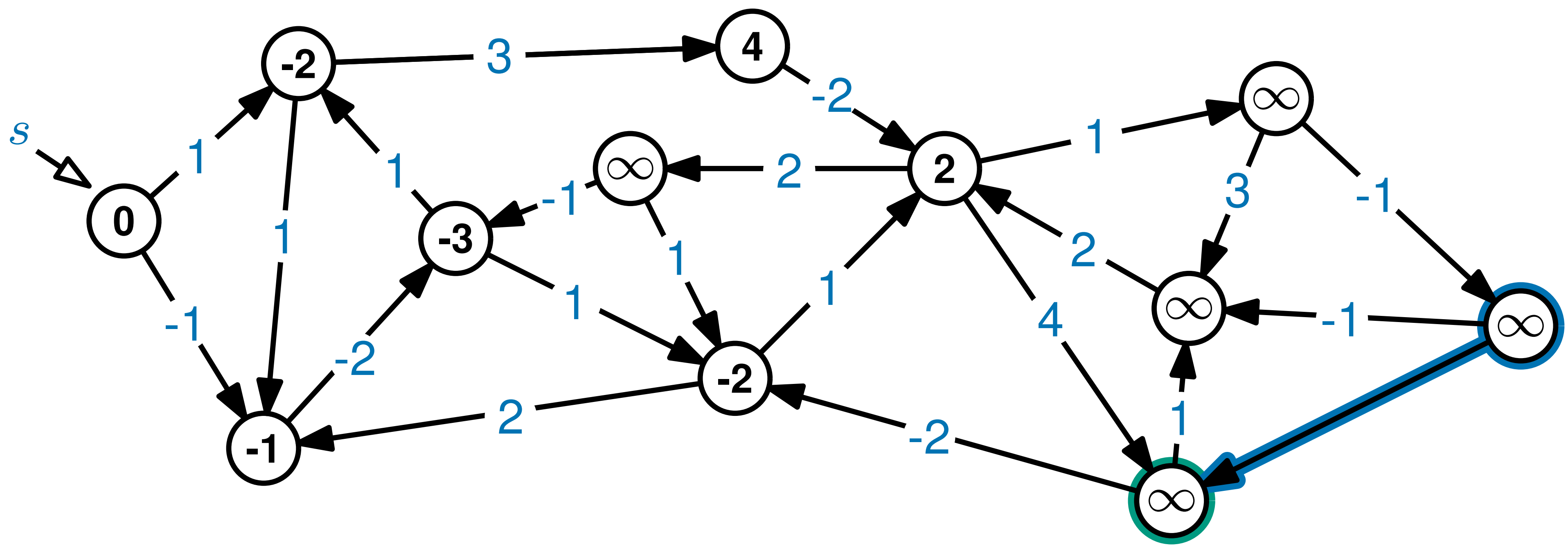
```
if dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

We now start iteration $1\ldots$



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

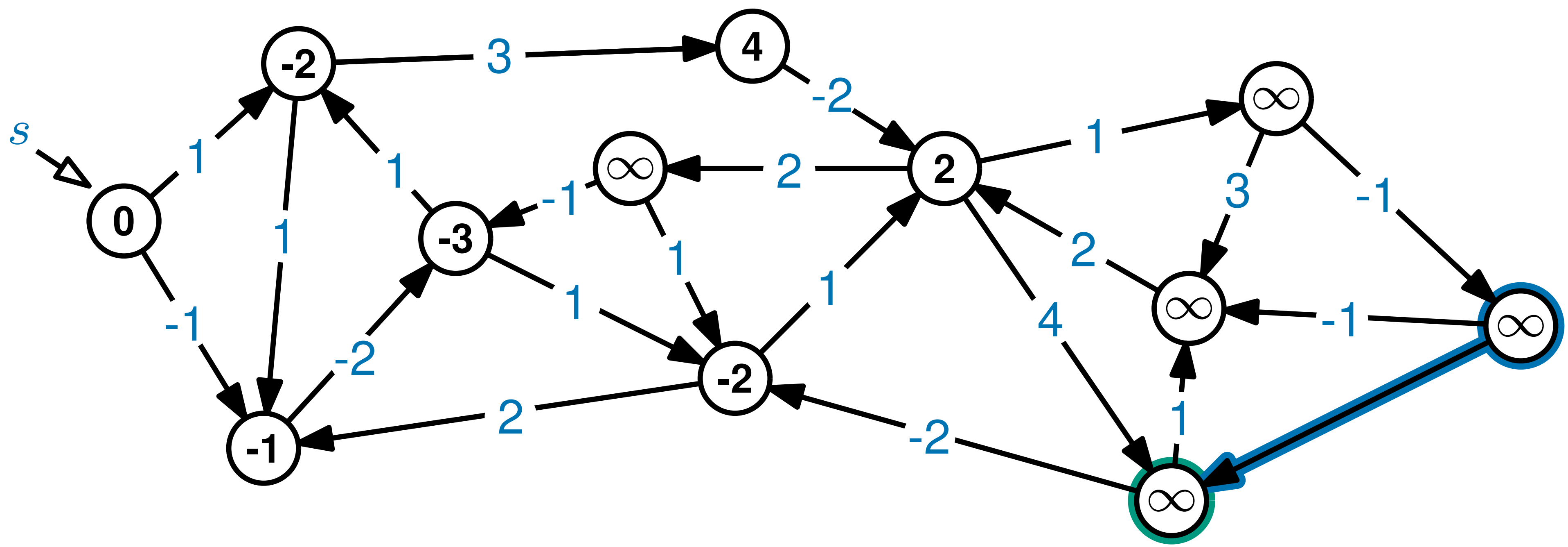
$\texttt{if dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad \texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

This is the end of iteration 1



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

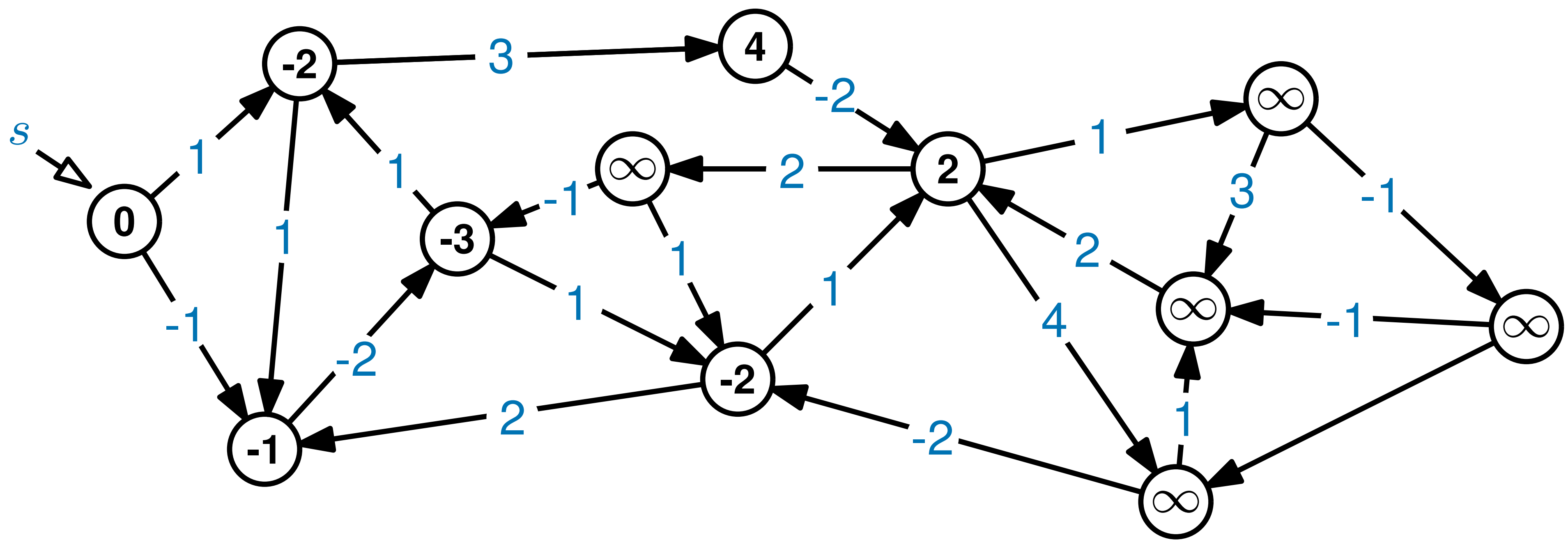
RELAX(u, v)

$\texttt{if dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad \texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MostOfBellman-Ford(s)



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

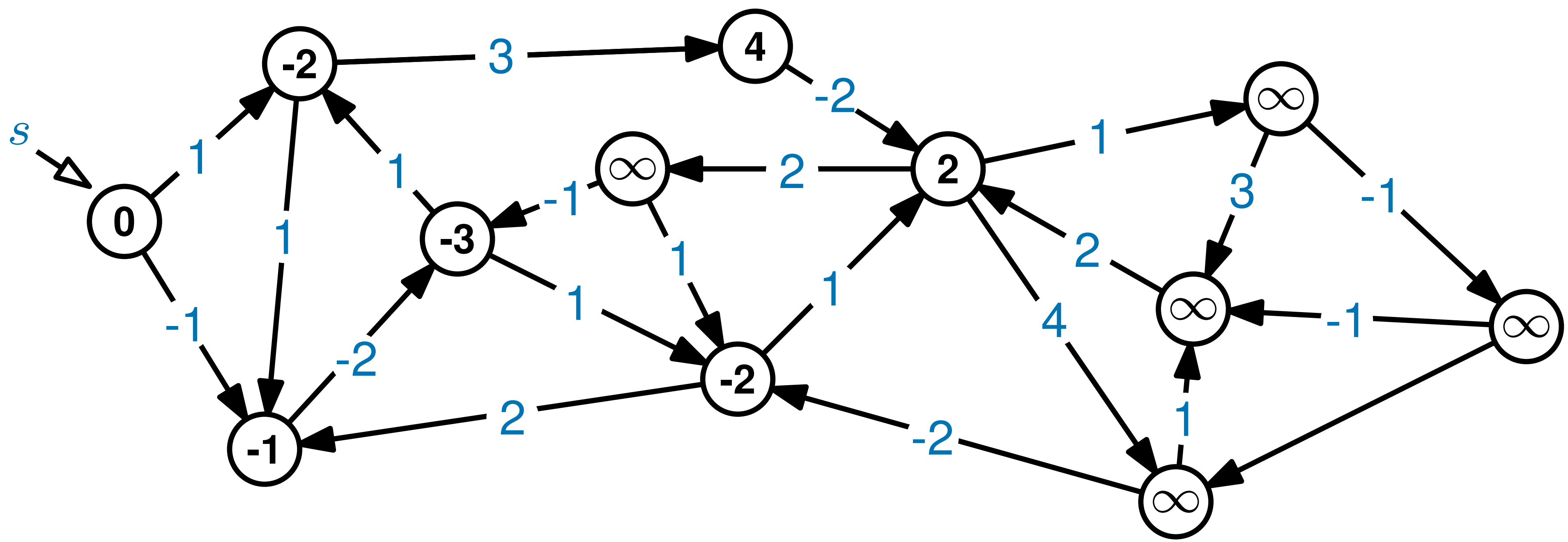
Relax(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

How are things looking?



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

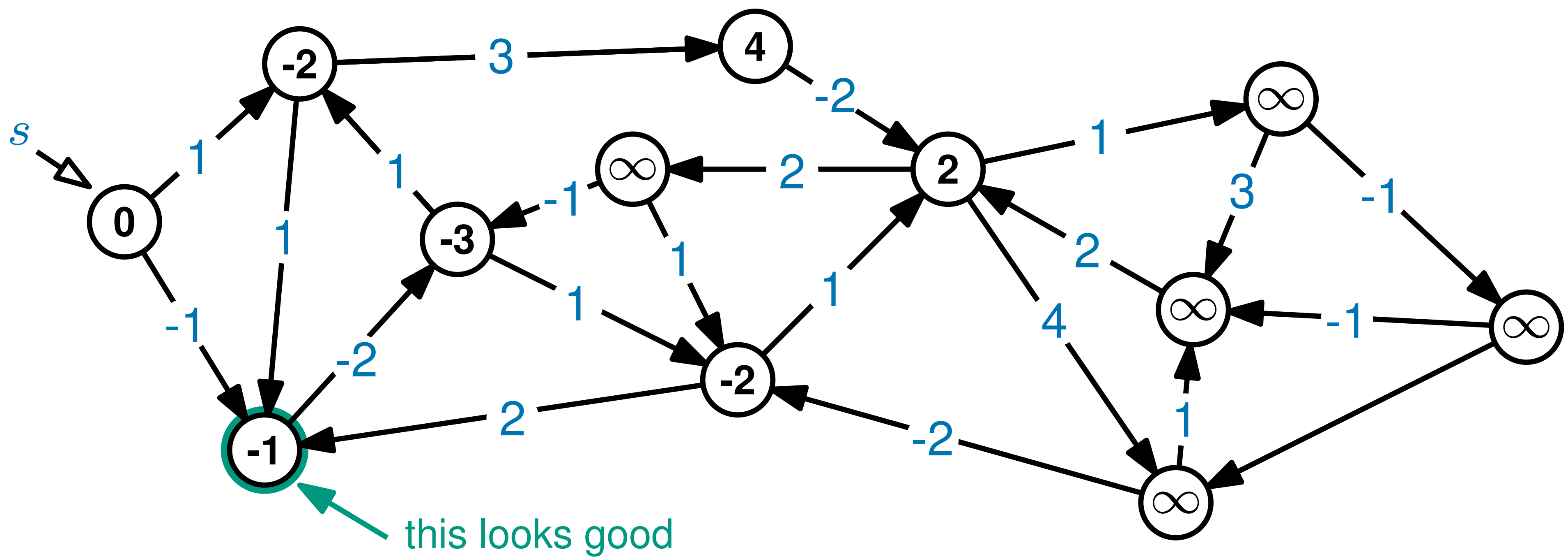
$\texttt{if}\ \ \texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\quad \texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 vertex v

$\mathtt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

How are things looking?



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

if $\mathtt{dist}(v) > \mathtt{dist}(u) + \mathtt{weight}(u, v)$
$\quad\mathtt{dist}(v) = \mathtt{dist}(u) + \mathtt{weight}(u, v)$

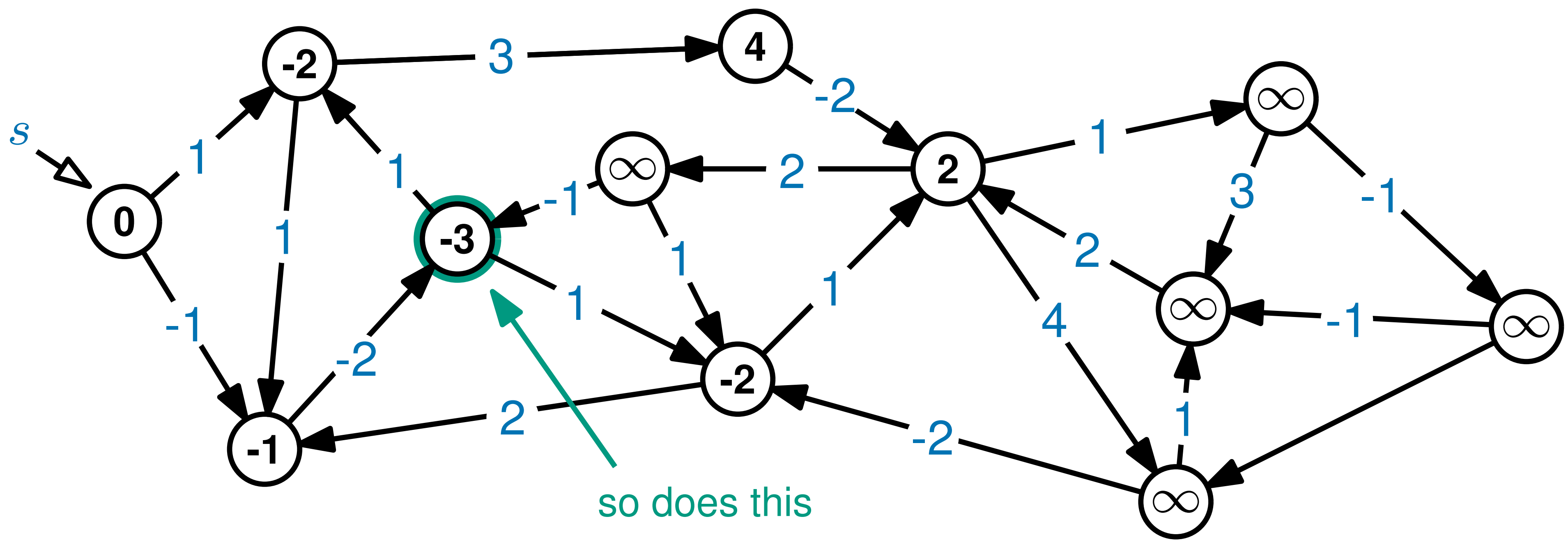
37 ← vertex v

$\mathtt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

if $\mathtt{dist}(v) > \mathtt{dist}(u) + \mathtt{weight}(u, v)$
$\mathtt{dist}(v) = \mathtt{dist}(u) + \mathtt{weight}(u, v)$

37 ← vertex v

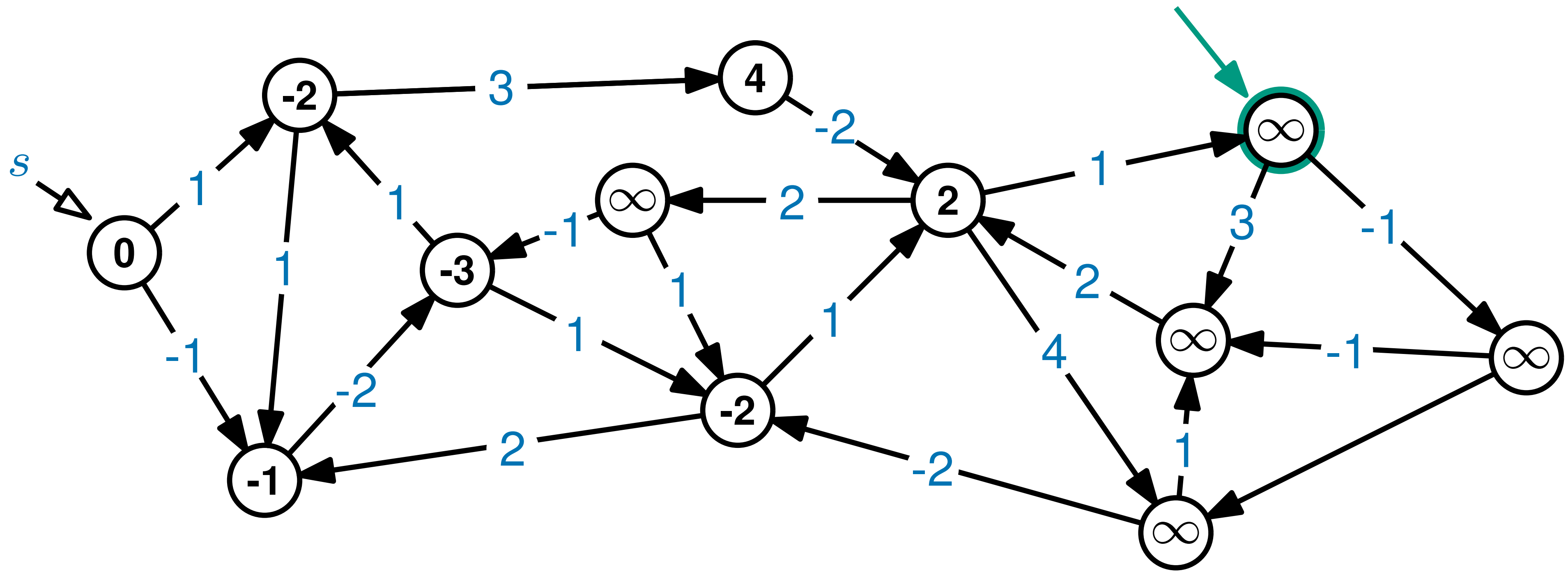
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?

this doesn't look so good



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

```
if  dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

37 ← vertex v

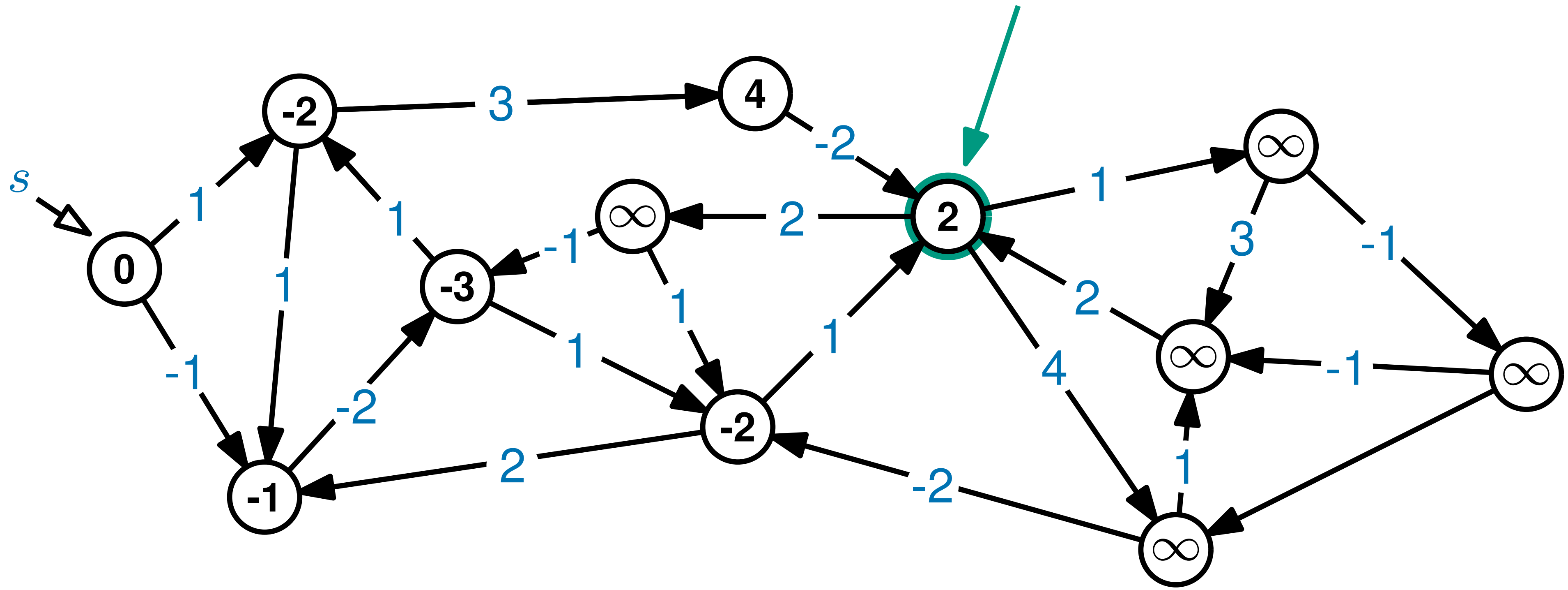
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?

neither does this



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

```
if  dist(v) > dist(u) + weight(u,v)
    dist(v) = dist(u) + weight(u,v)
```

37 ← vertex v

$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

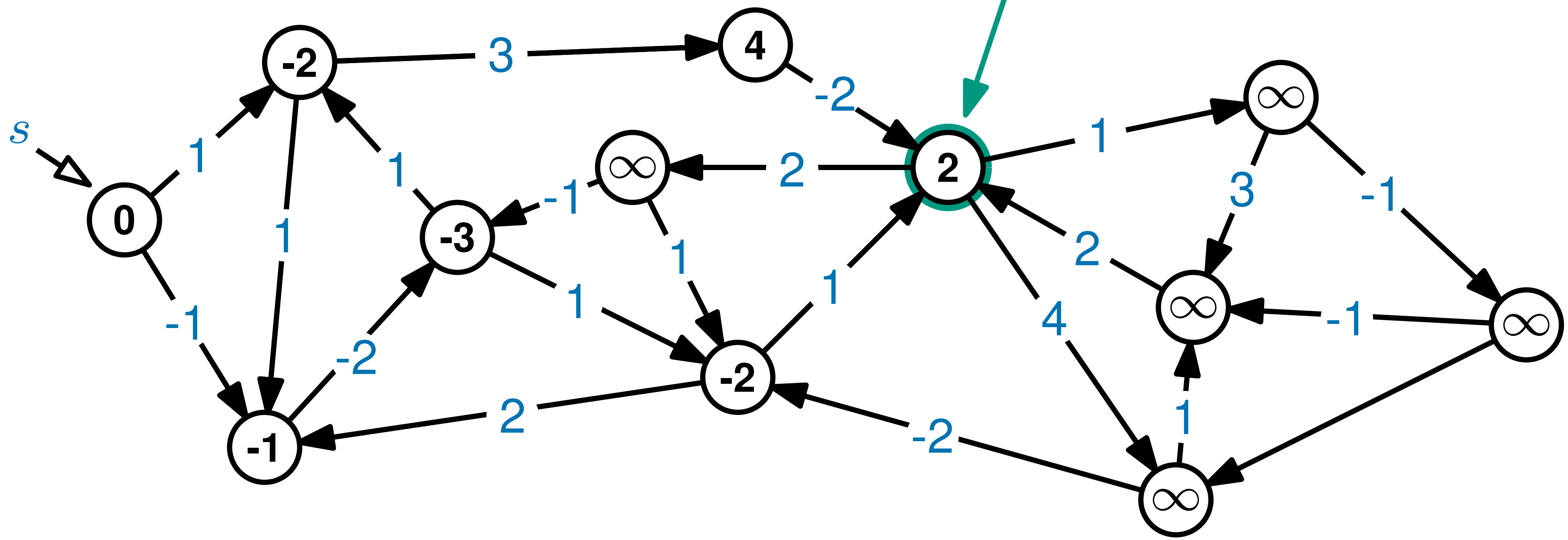
We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?

neither does this

(it's not the shortest path length)



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

```
if dist(v) > dist(u) + weight(u, v)
    dist(v) = dist(u) + weight(u, v)
```

37 ← vertex v

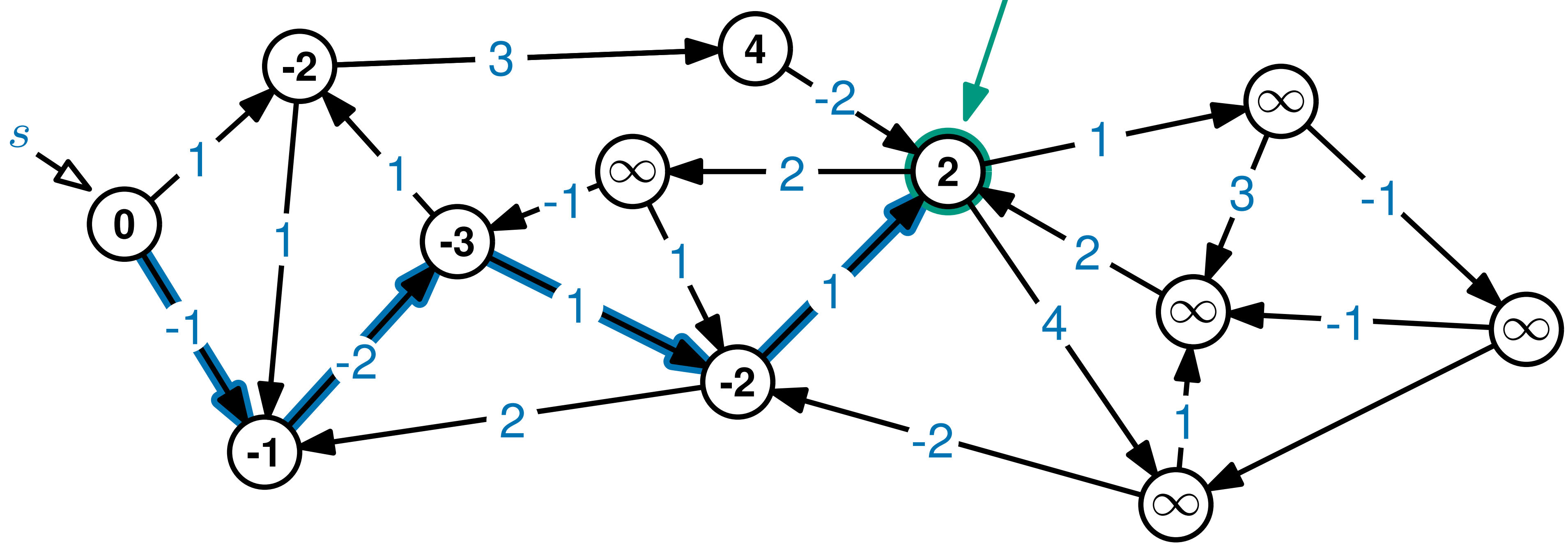
$\texttt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

How are things looking?

neither does this

(it's not the shortest path length)



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
  $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

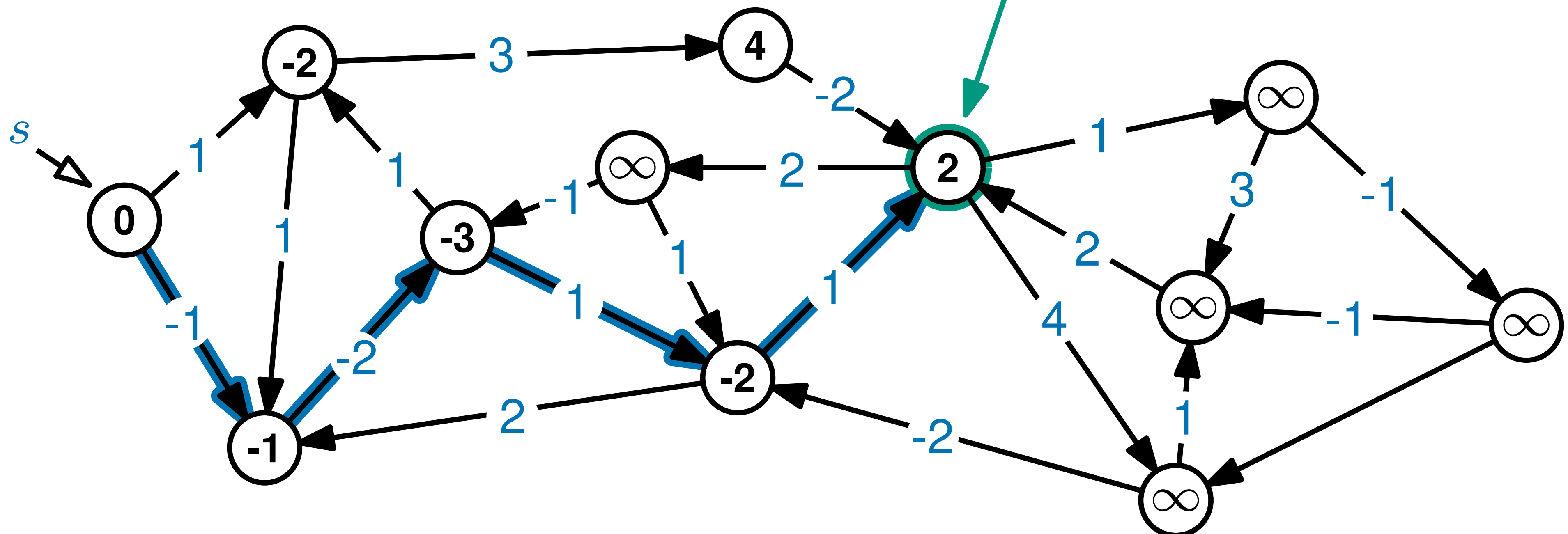
We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?

neither does this

(it's not the shortest path length)



This path has length -1

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)
if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$ $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

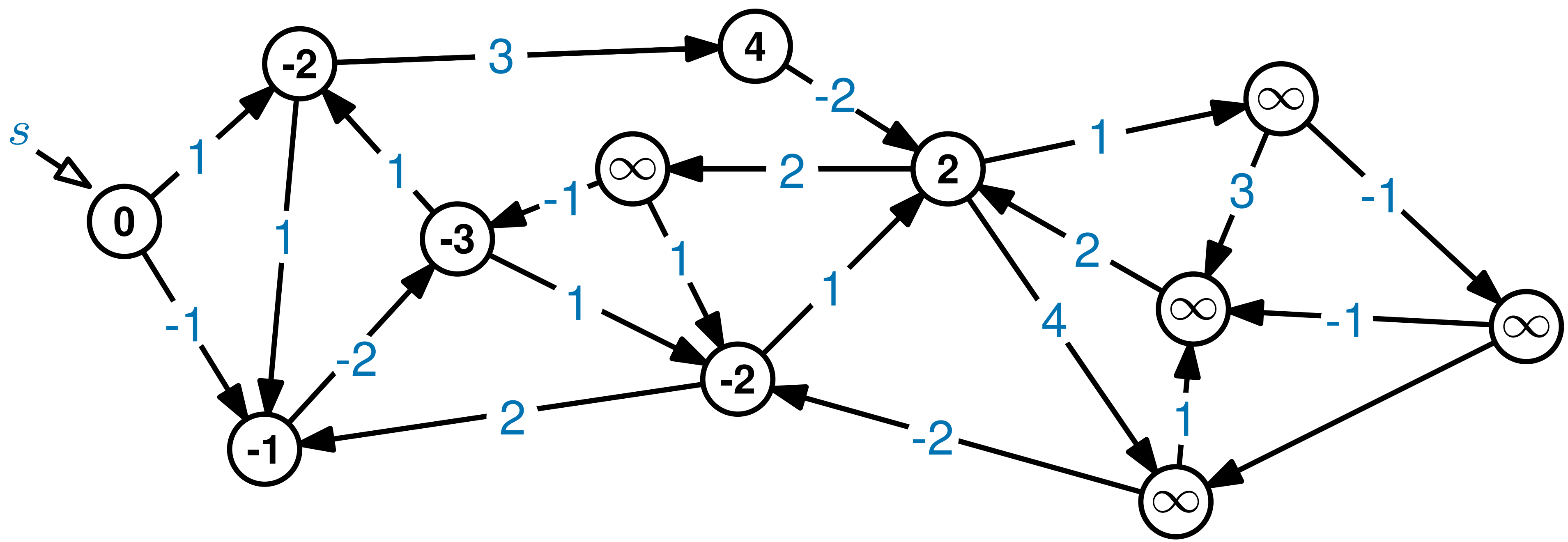
37 ⟵ vertex v

$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?



MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

$\texttt{if}\ \ \texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$

$\quad\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

37 ← vertex v

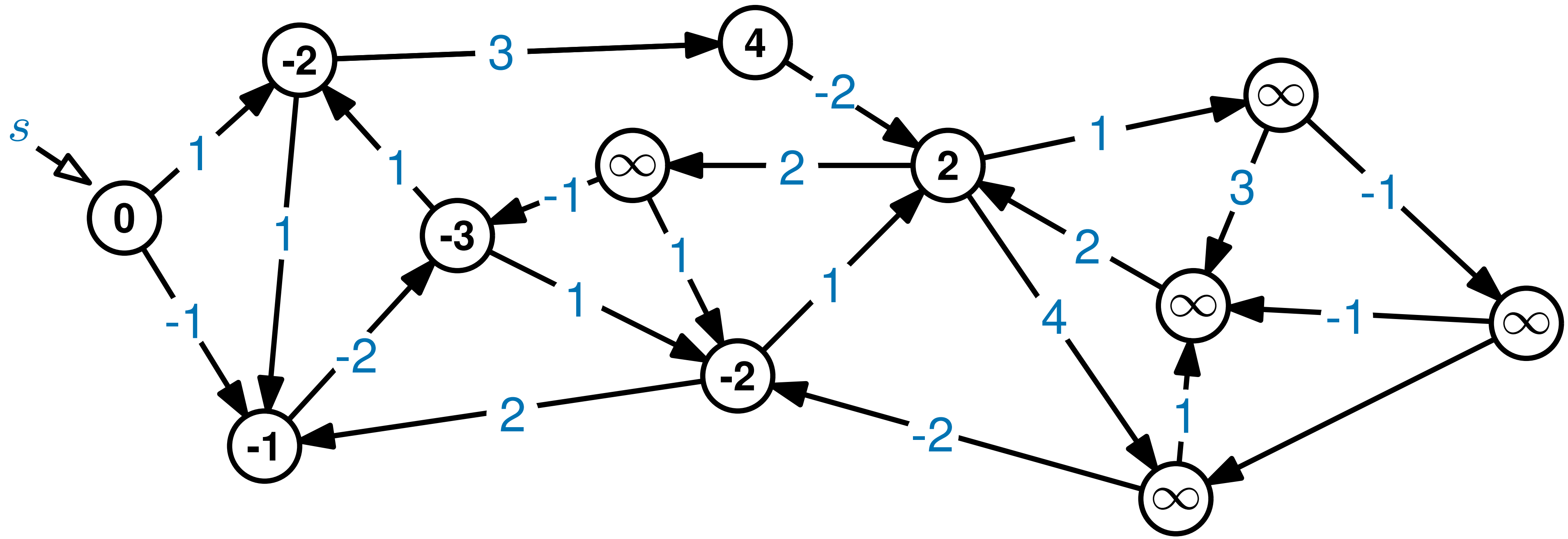
$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MostOfBellman-Ford(s)

How are things looking?

we aren't done



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

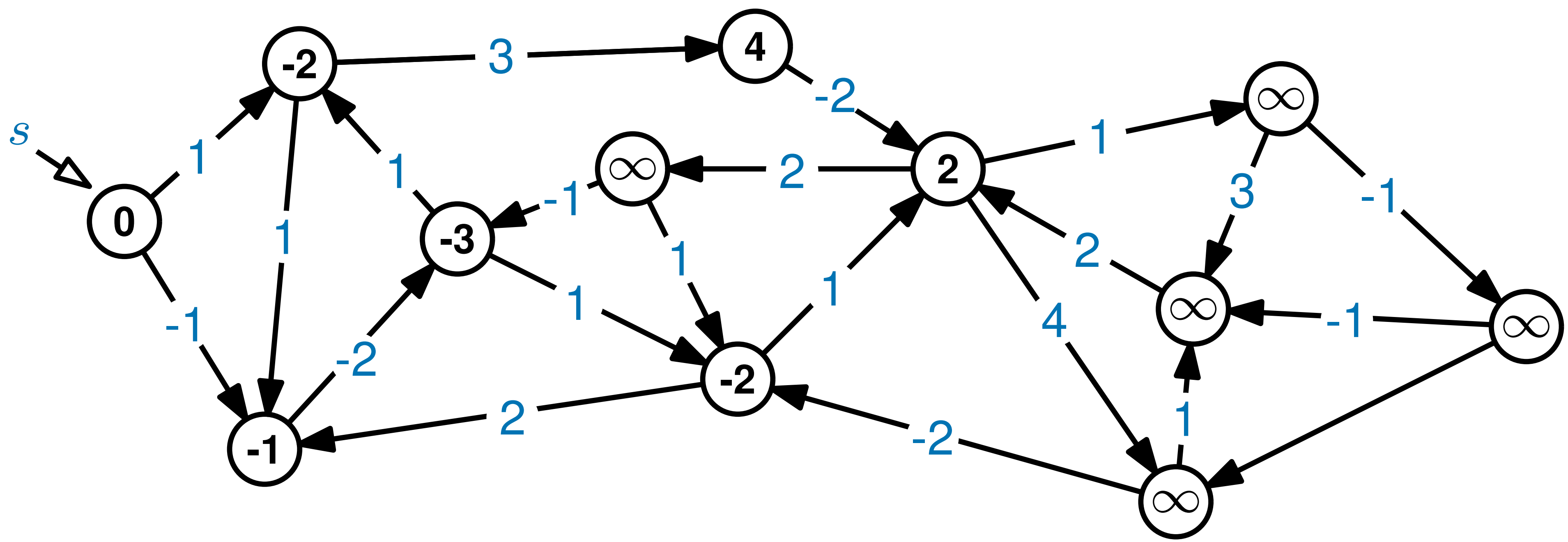
- the length of the best path between s and v, found so far

We're going to simulate

MostOfBellman-Ford(s)

How are things looking?

we aren't done *but it seems like we made progress*



MostOfBellman-Ford runs $|V|$ iterations,

In each iteration we Relax every edge (u, v)

(in the order they occur in the adjacency list)

Relax(u, v)

$$\texttt{if } \texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$$
$$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$$

vertex v

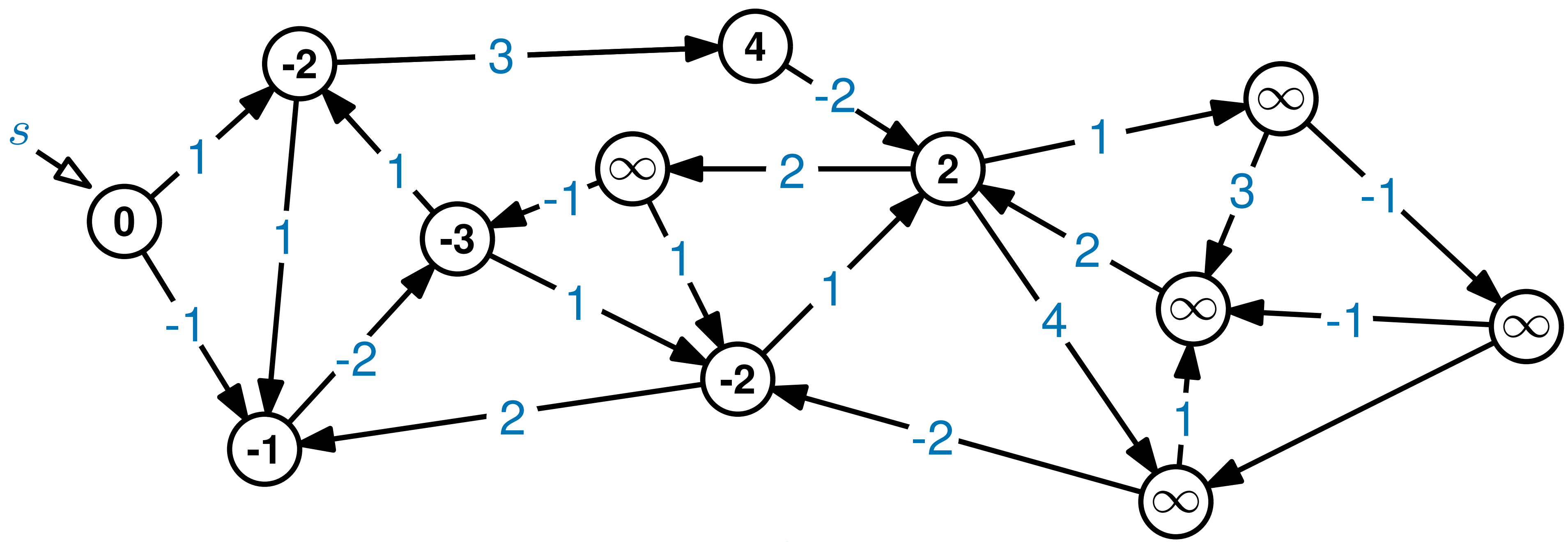
$\texttt{dist}(v)$ *- the length of the best path between s and v, found so far*

We're going to simulate

MOSTOFBELLMAN-FORD(s)

How are things looking?

we aren't done *but it seems like we made progress*



does every iteration make progress?

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

(in the order they occur in the adjacency list)

RELAX(u, v)

if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
$\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

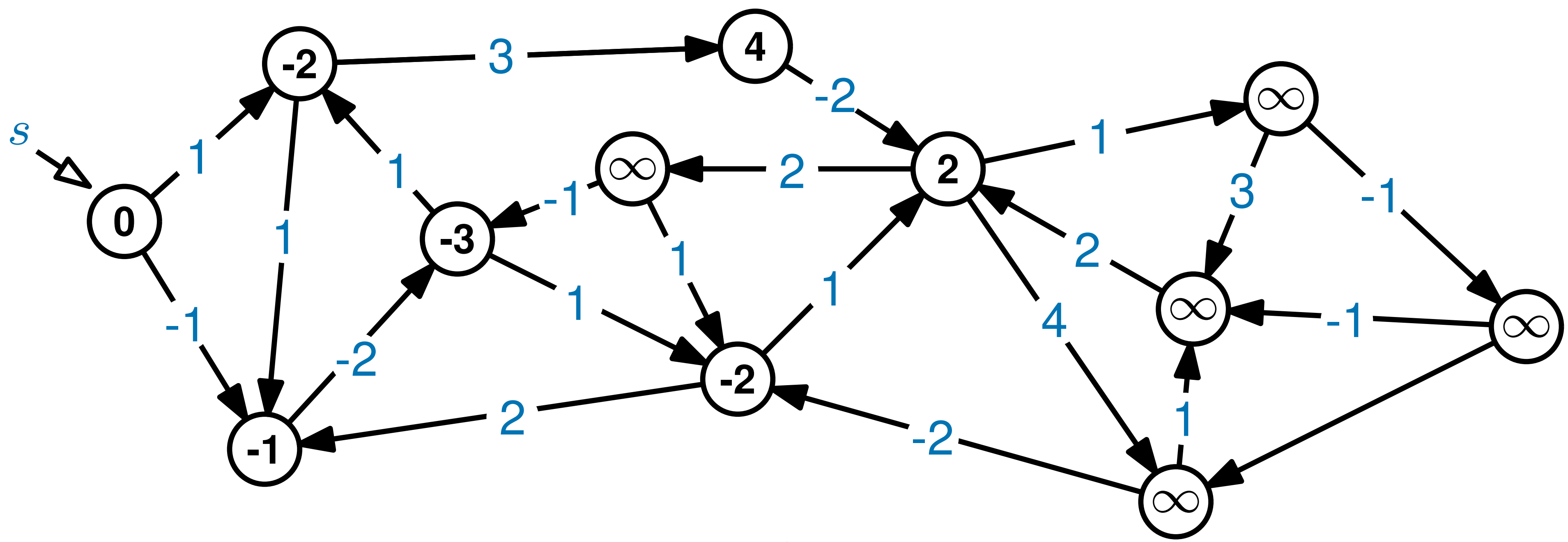
37 ⟵ vertex v

$\mathtt{dist}(v)$ - *the length of the best path between s and v, found so far*

We're going to simulate MOSTOFBELLMAN-FORD(s)

How are things looking?

we aren't done *but it seems like we made progress*



does every iteration make progress?

are $|V|$ iterations enough?

MOSTOFBELLMAN-FORD runs $|V|$ iterations,

In each iteration we RELAX every edge (u, v)

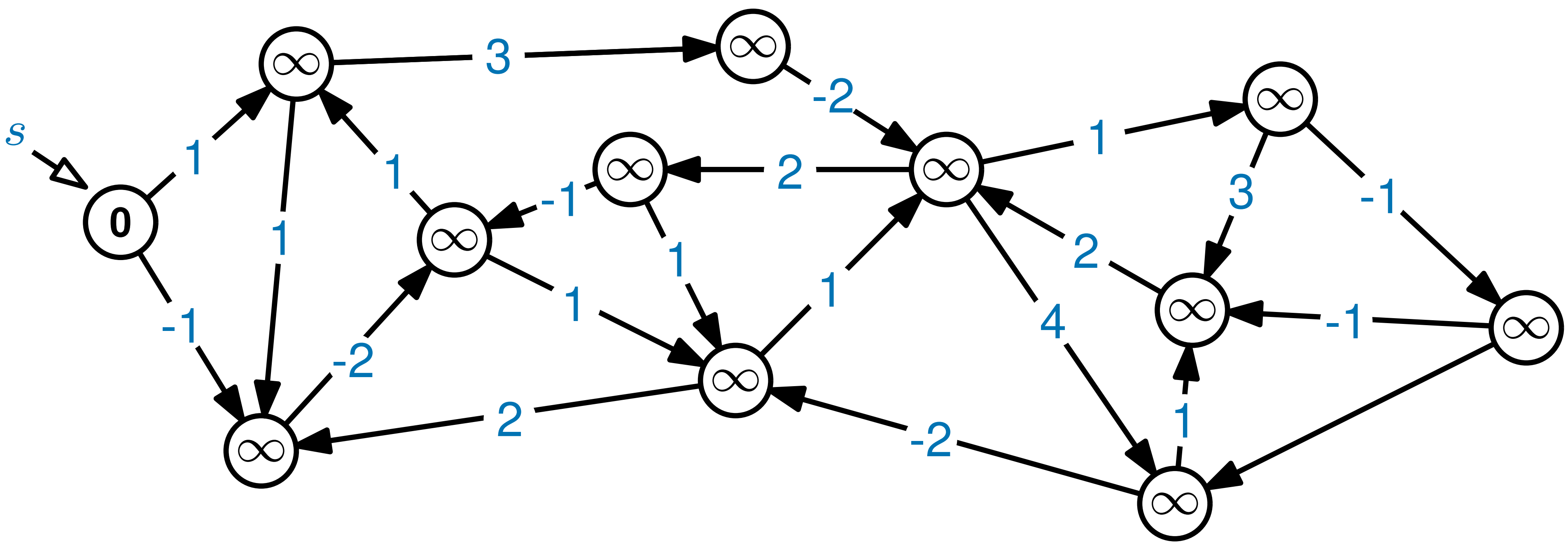
(in the order they occur in the adjacency list)

RELAX(u, v)
`if` $\mathtt{dist}(v) > \mathtt{dist}(u) + \mathtt{weight}(u, v)$ $\quad\mathtt{dist}(v) = \mathtt{dist}(u) + \mathtt{weight}(u, v)$

The proof idea

Imagine a different algorithm where in each iteration...

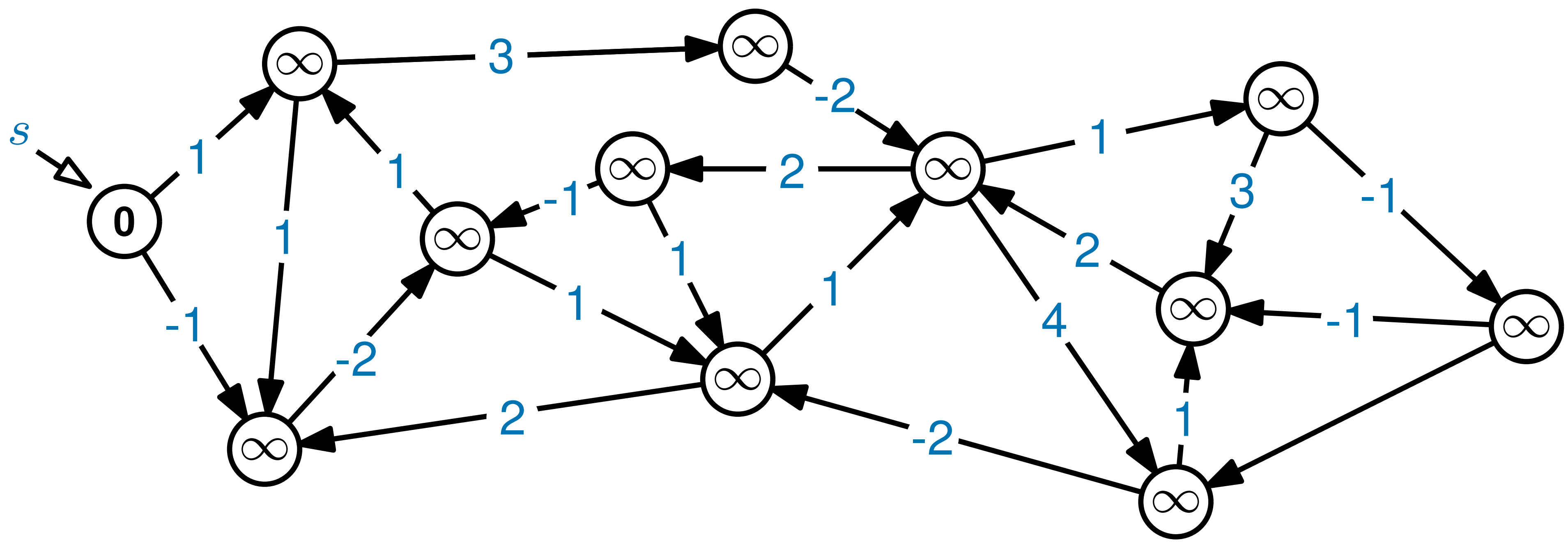
you only relax one edge *(rather than all edges)*



The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

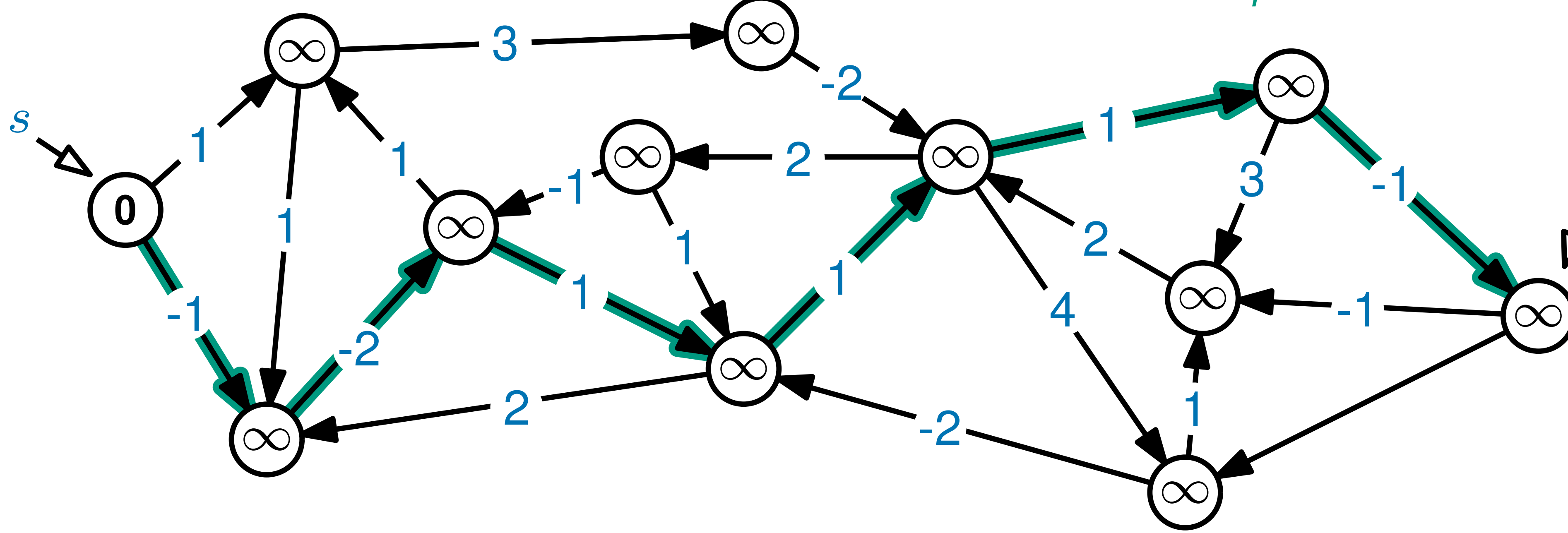
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

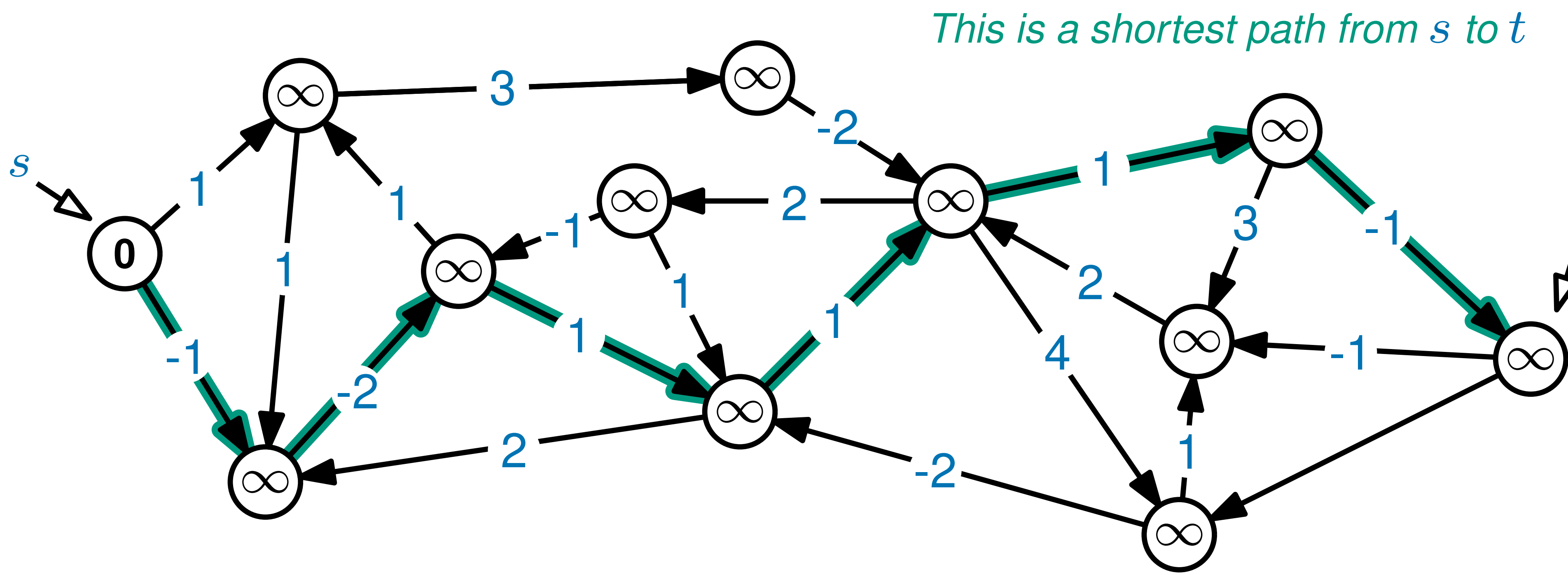
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 1:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

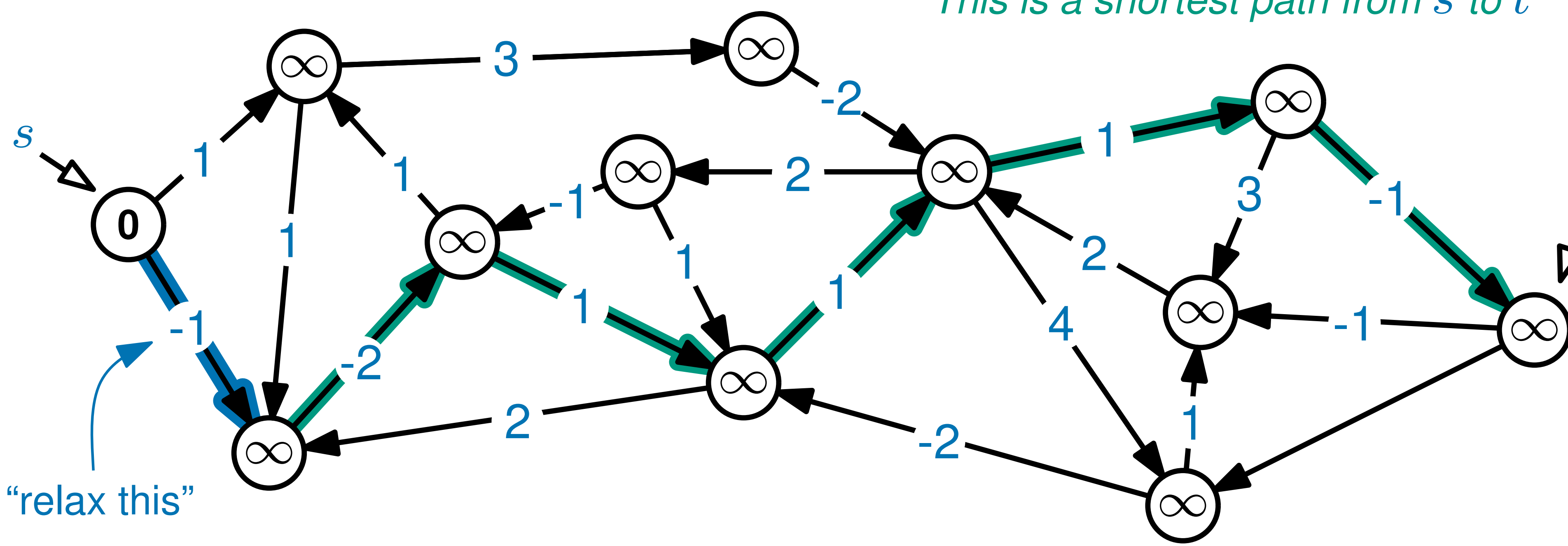
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 1:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

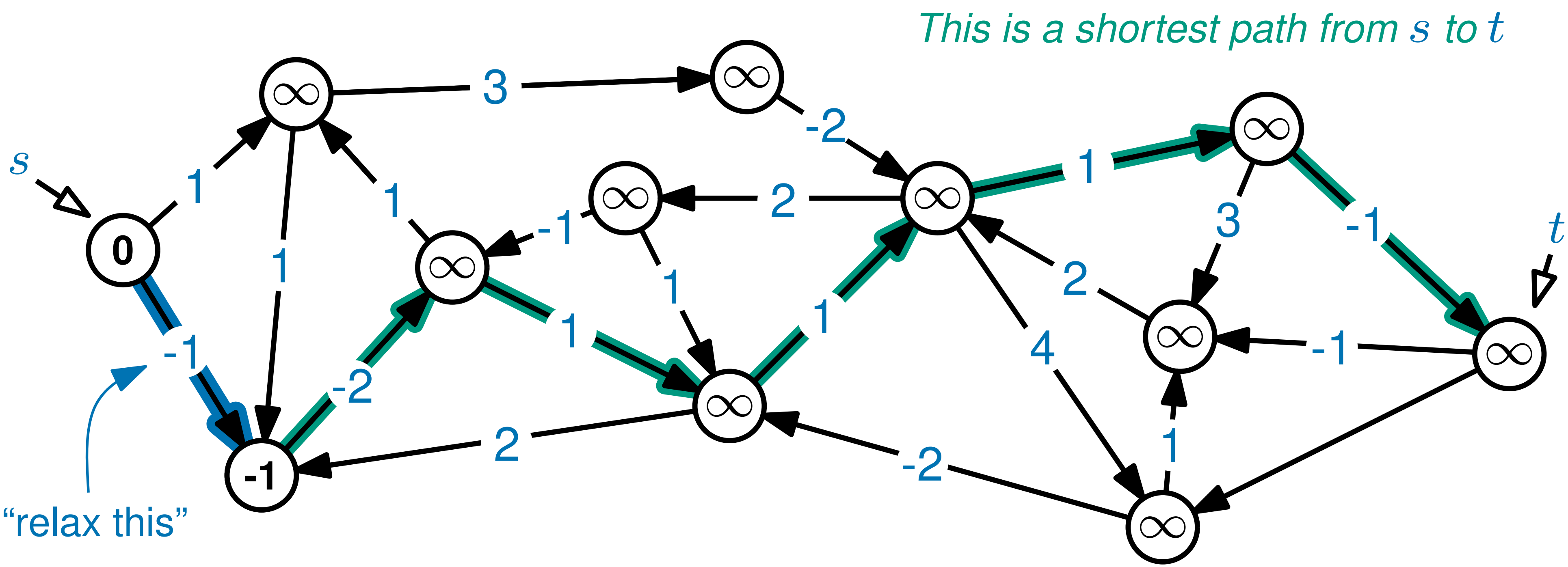
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 1:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i is the i-th edge in a shortest path from s to some vertex t

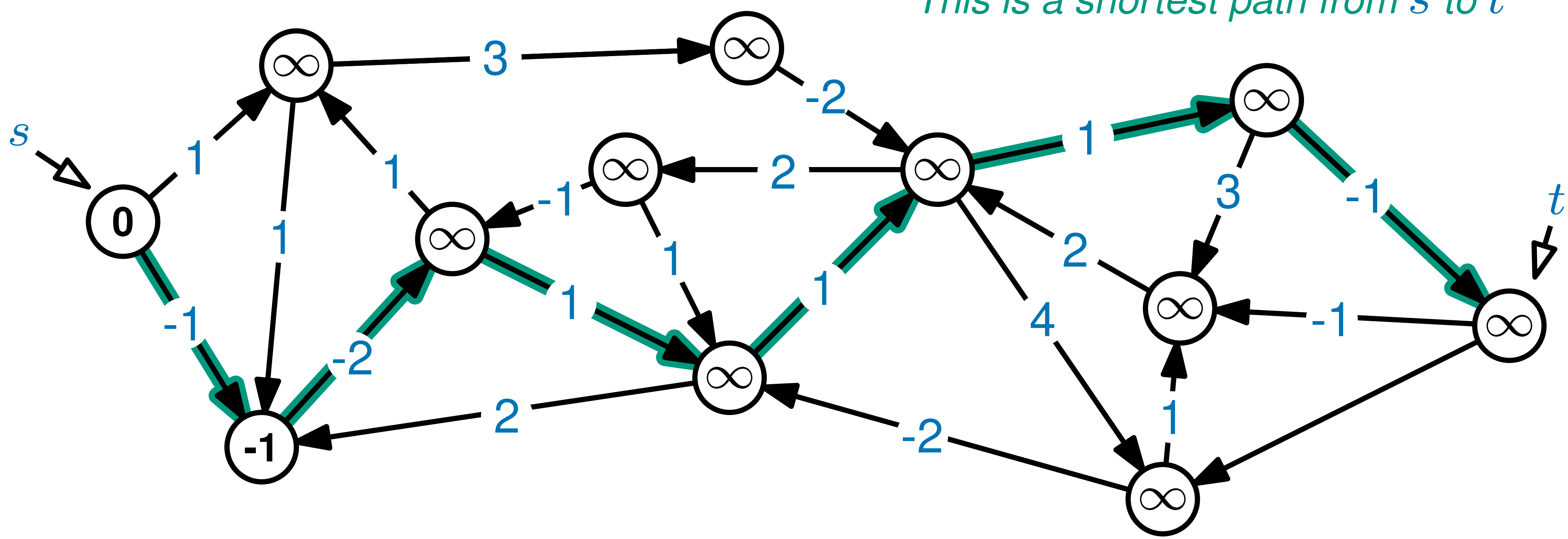
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 1:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

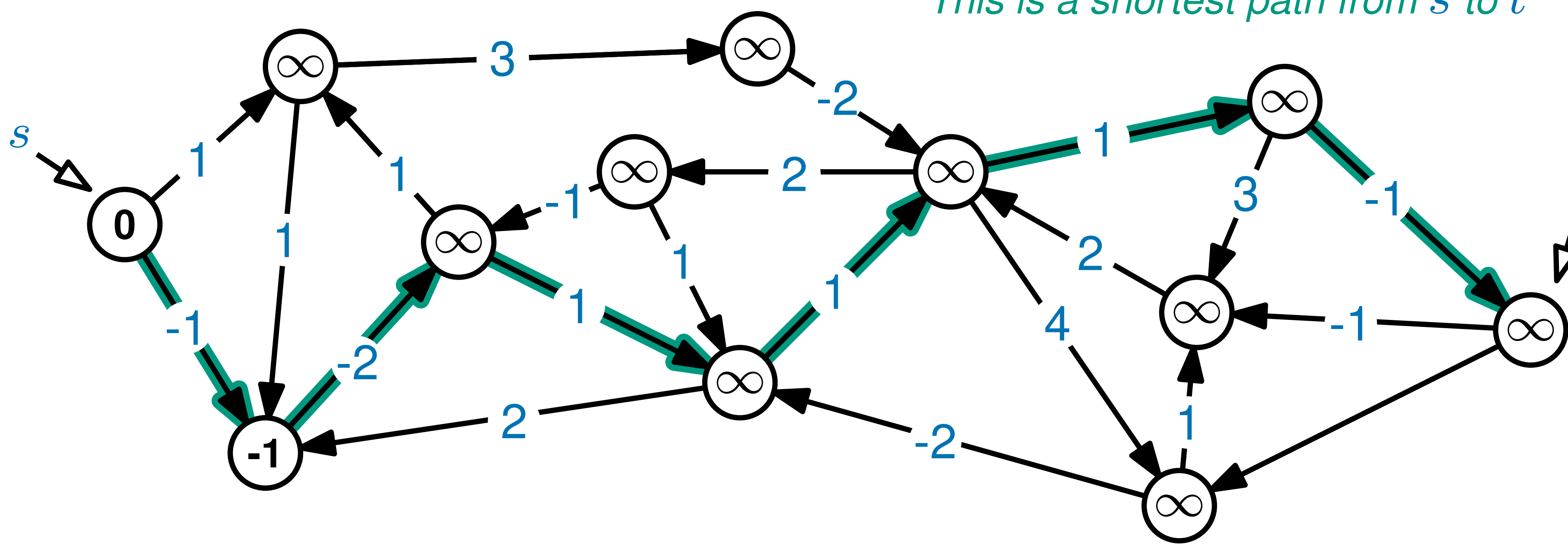
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 2:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

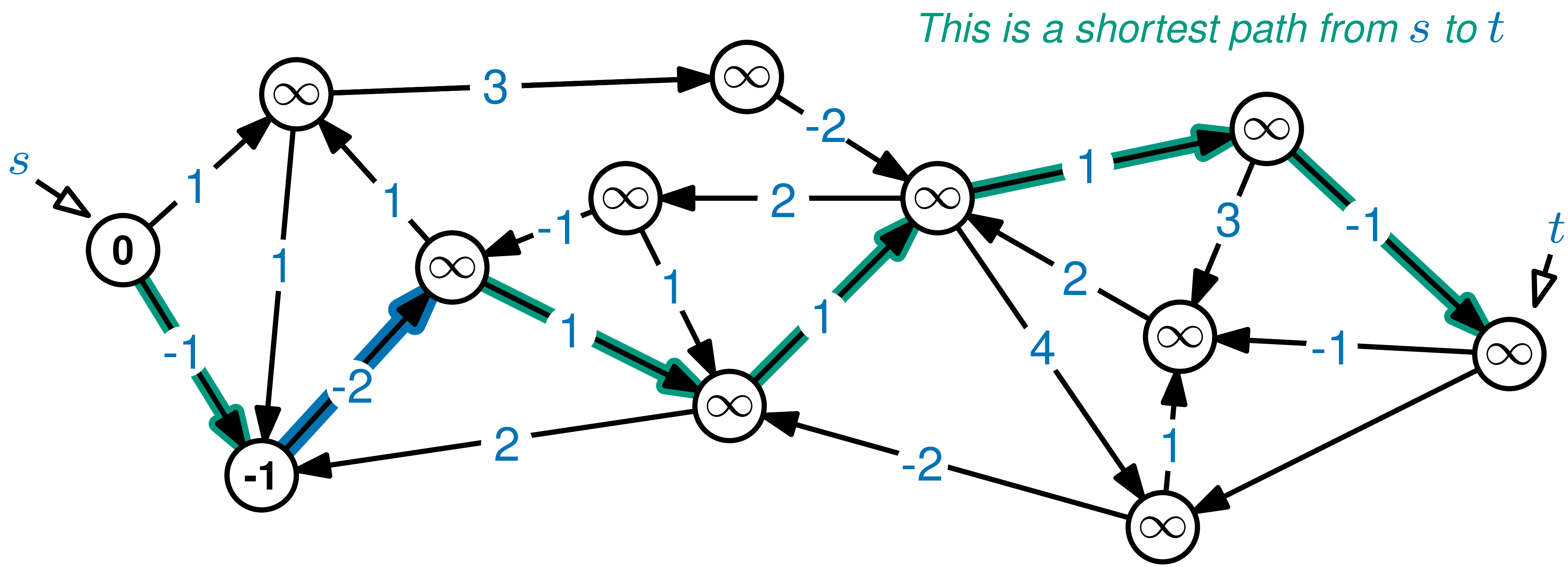
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 2:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

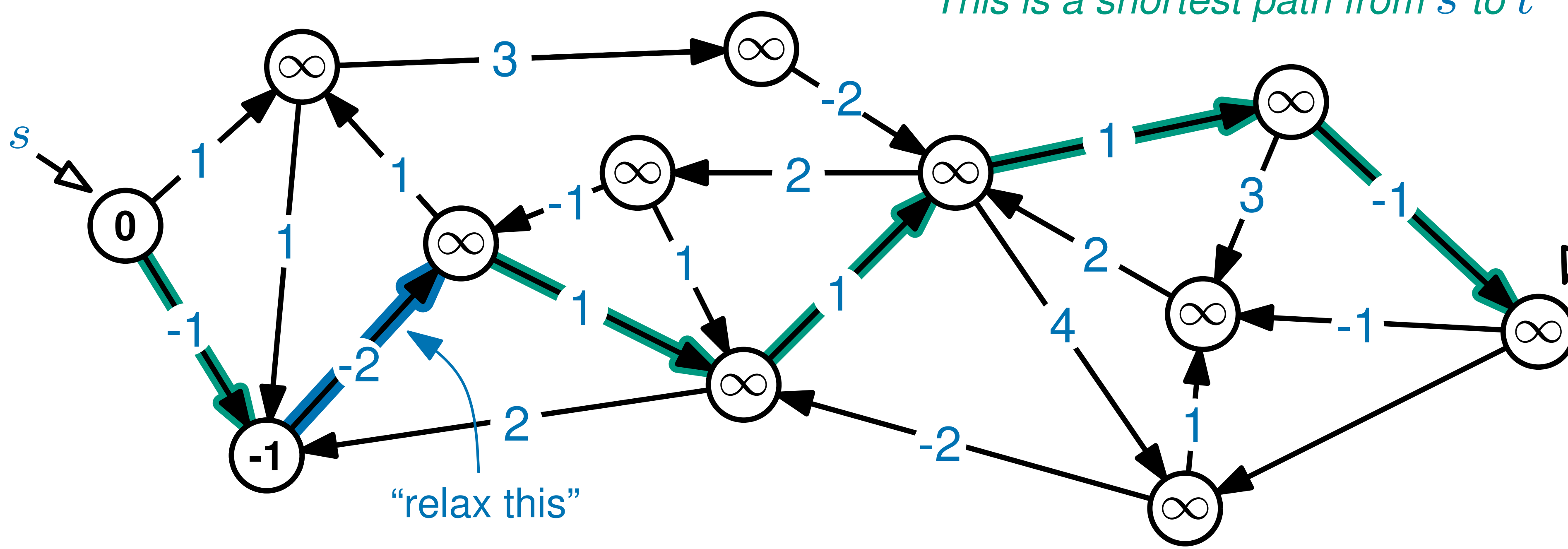
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 2:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

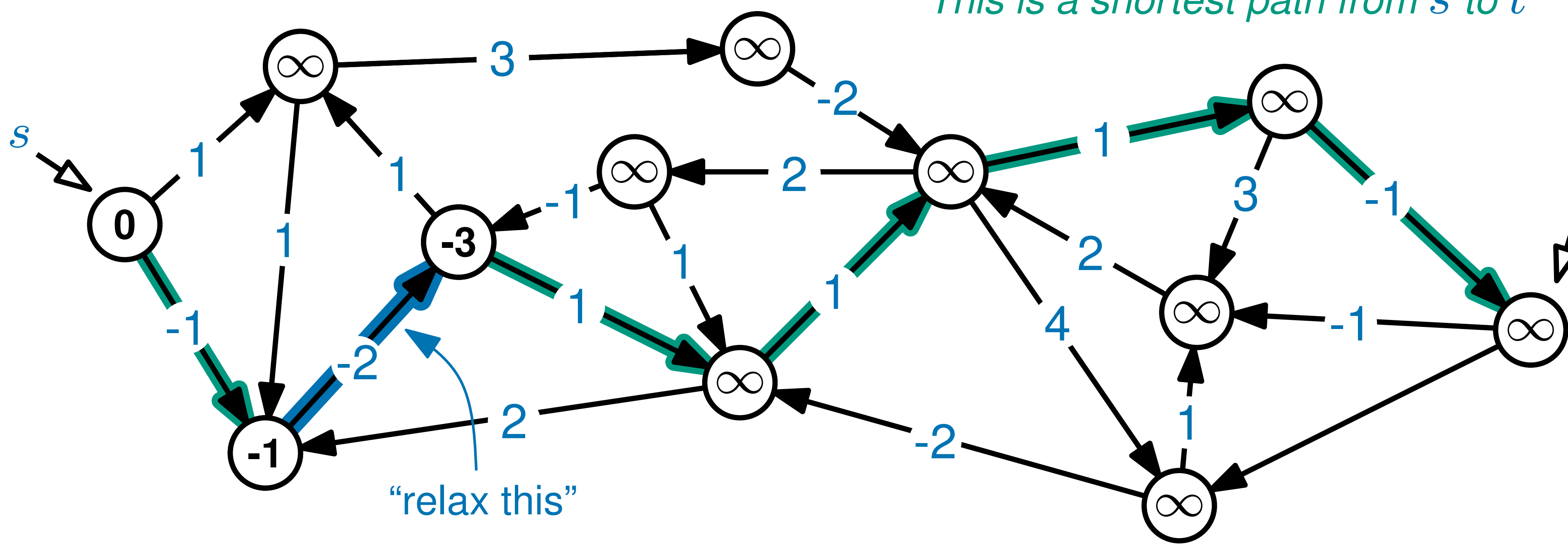
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 2:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

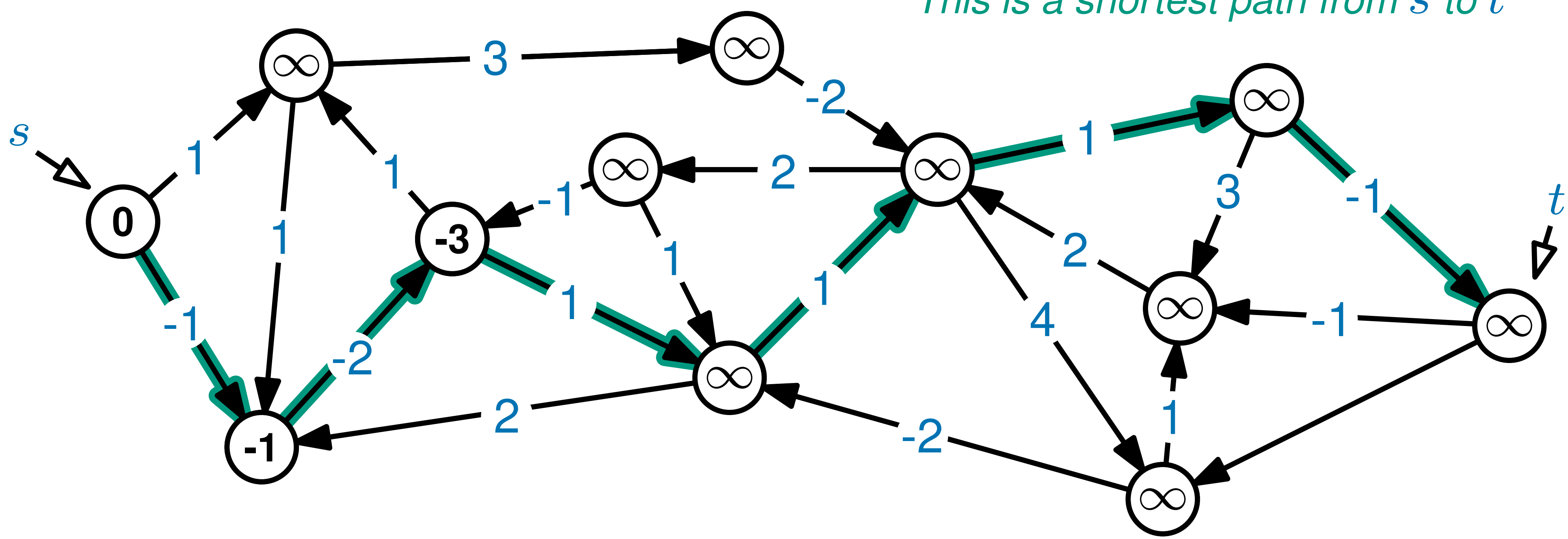
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 2:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

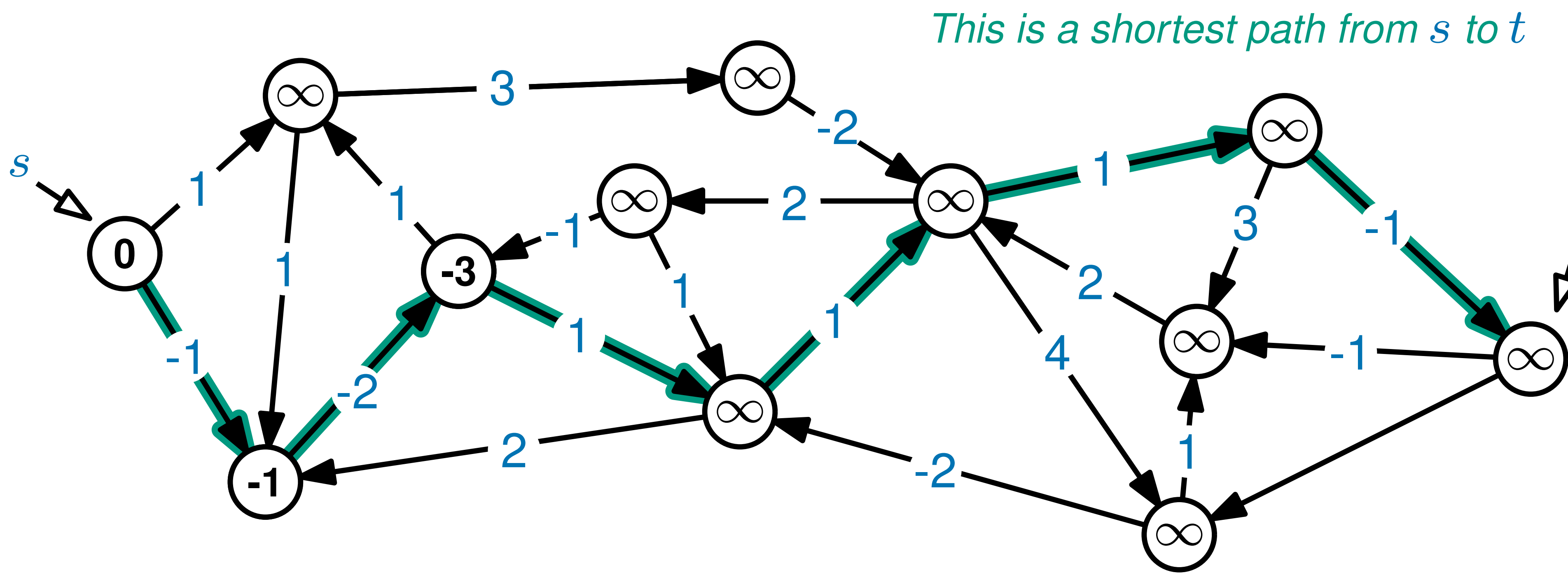
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 3:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

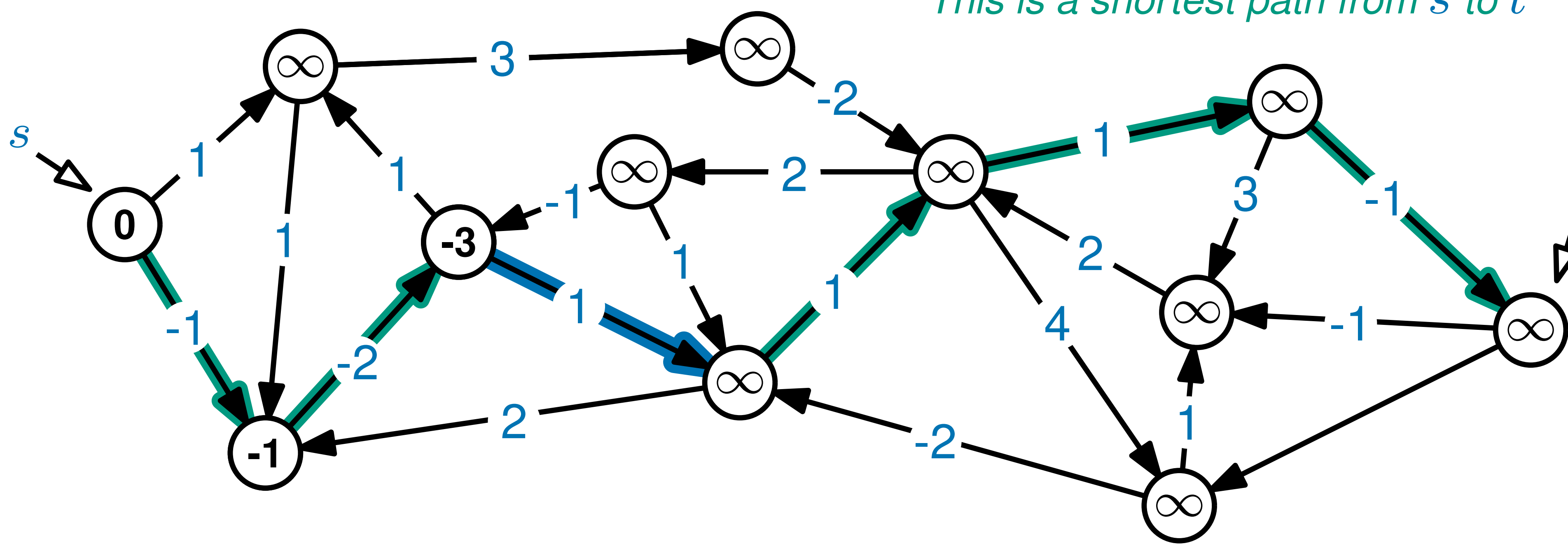
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 3:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

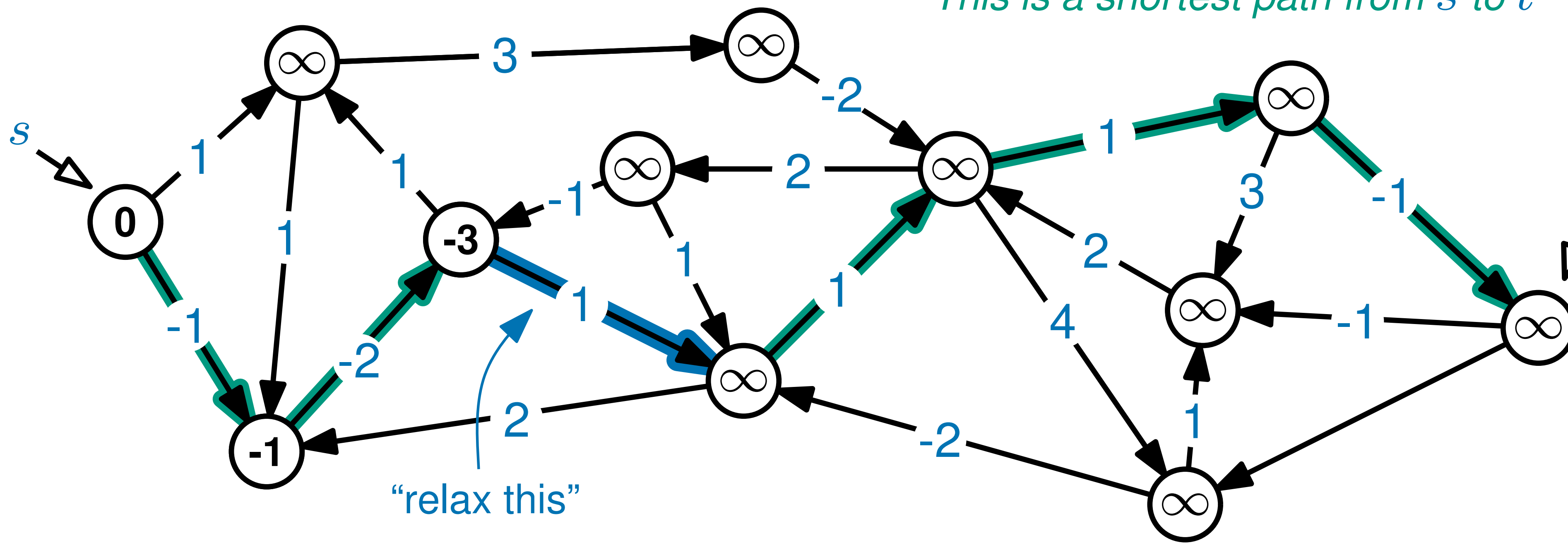
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 3:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

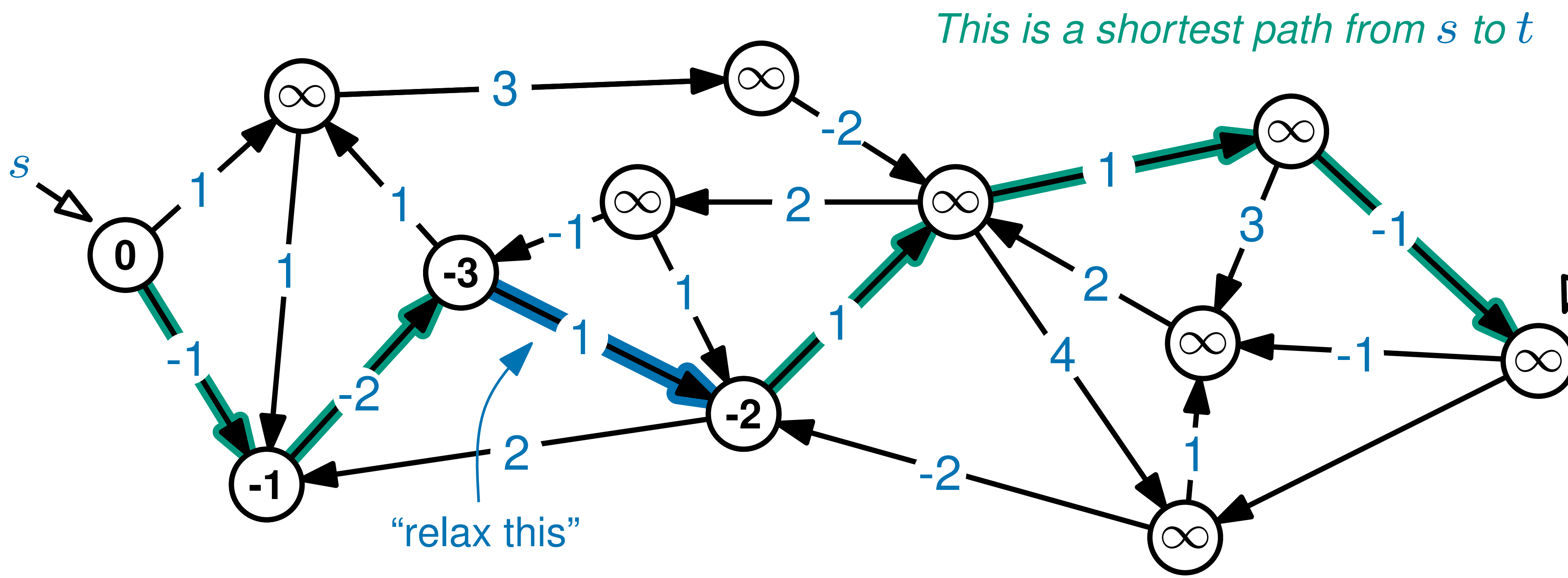
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 3:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

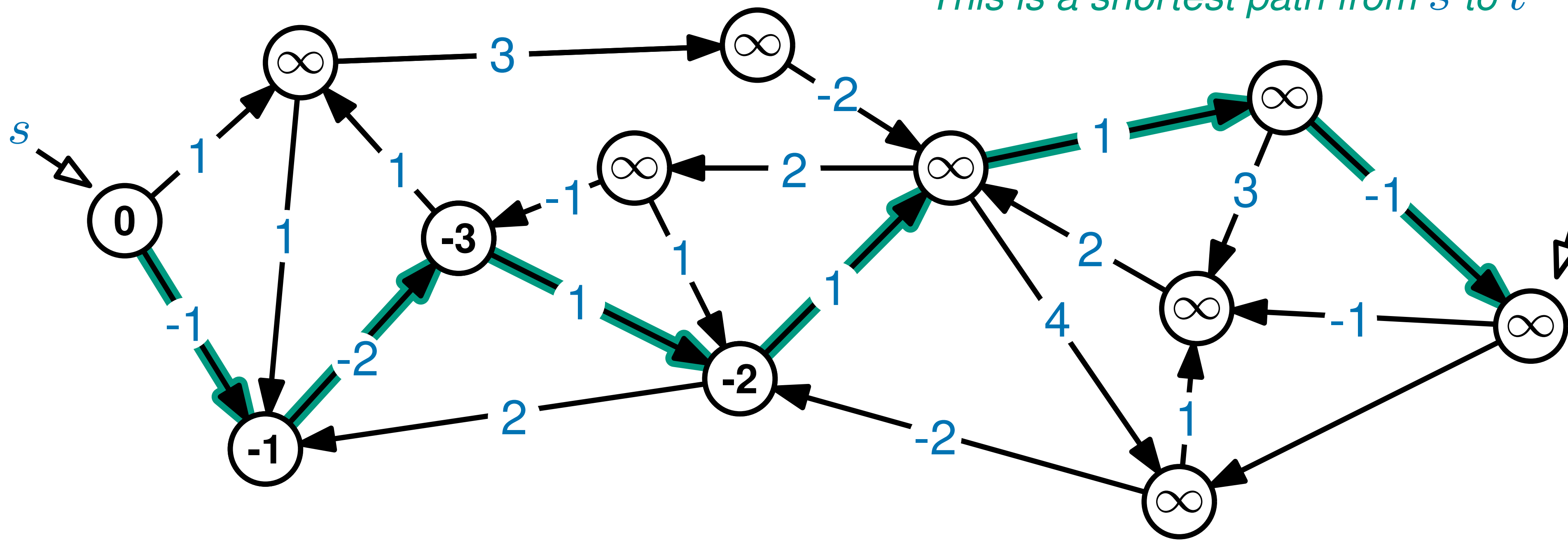
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 3:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

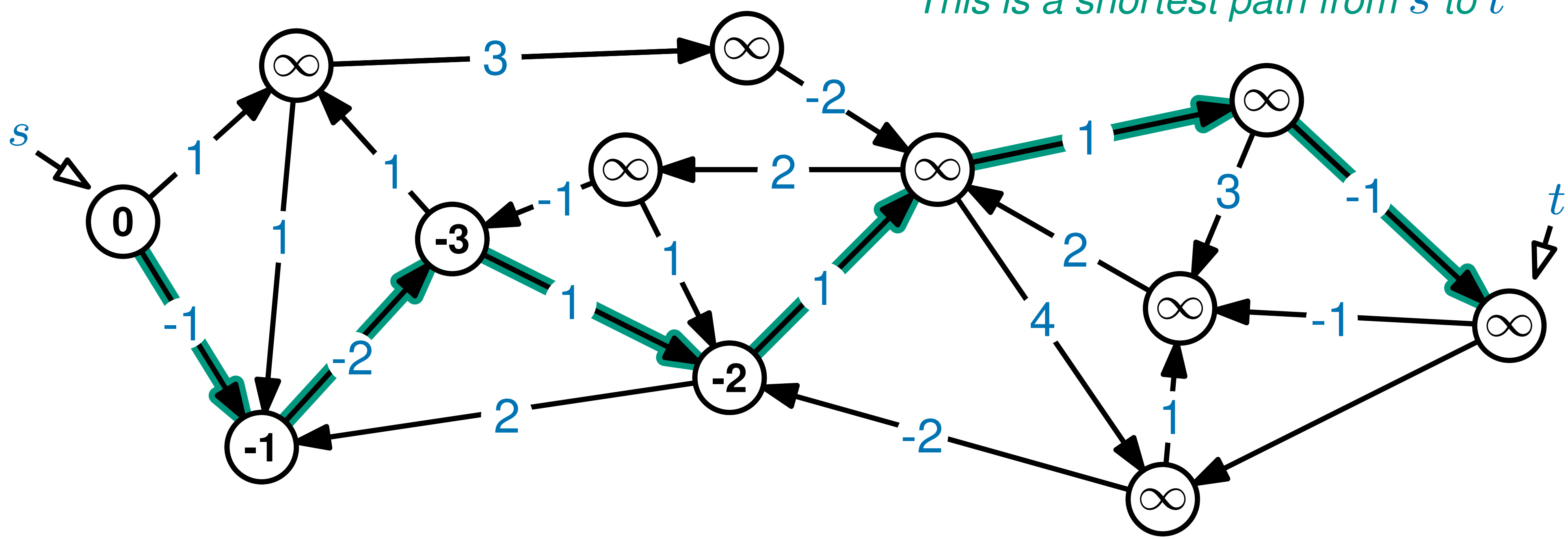
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 4:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

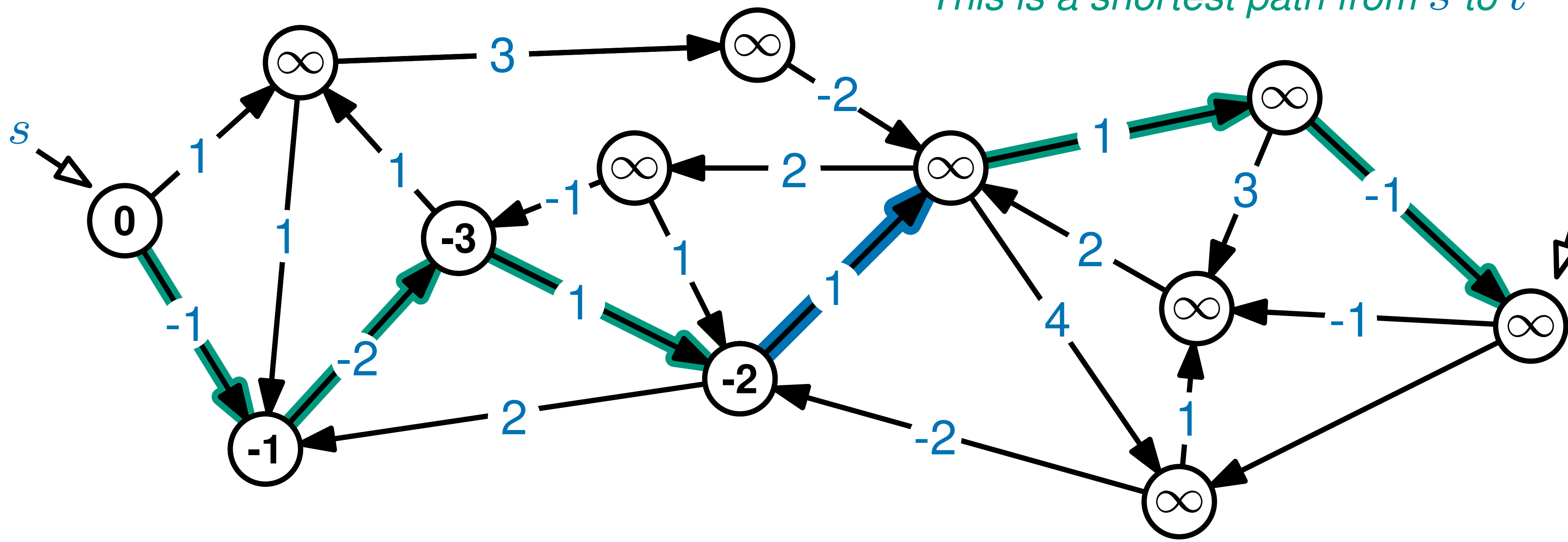
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 4:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

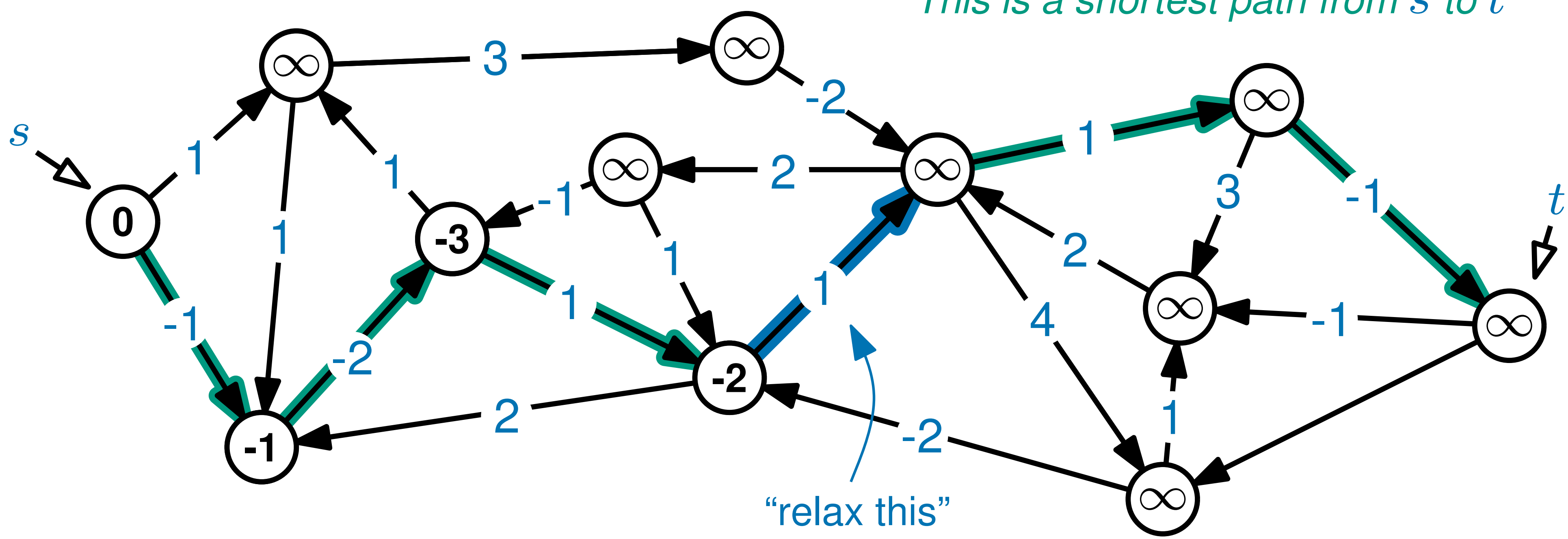
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 4:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

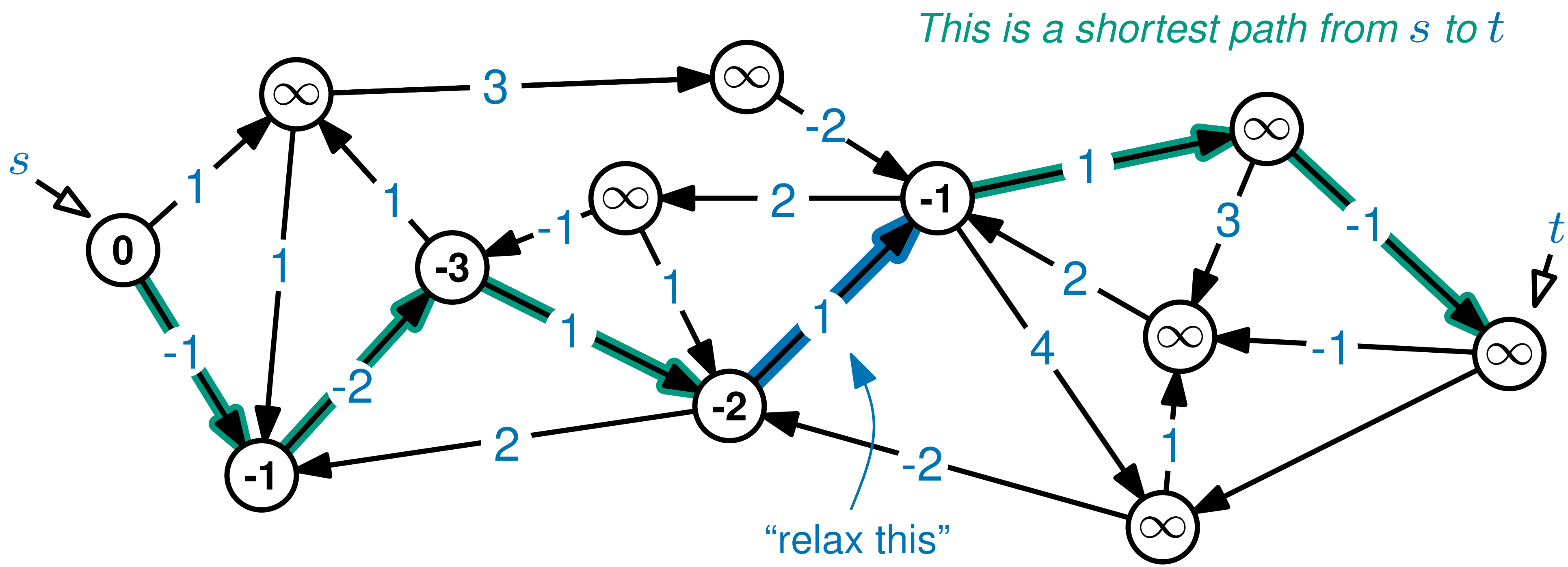
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 4:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

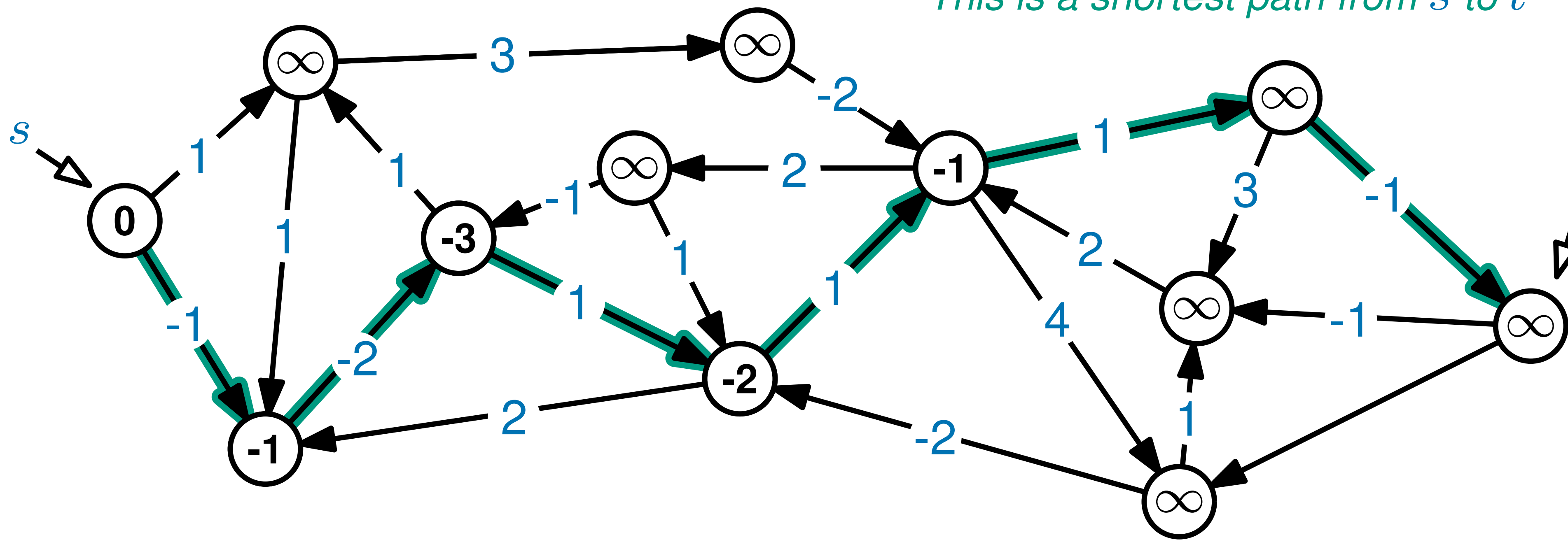
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 4:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

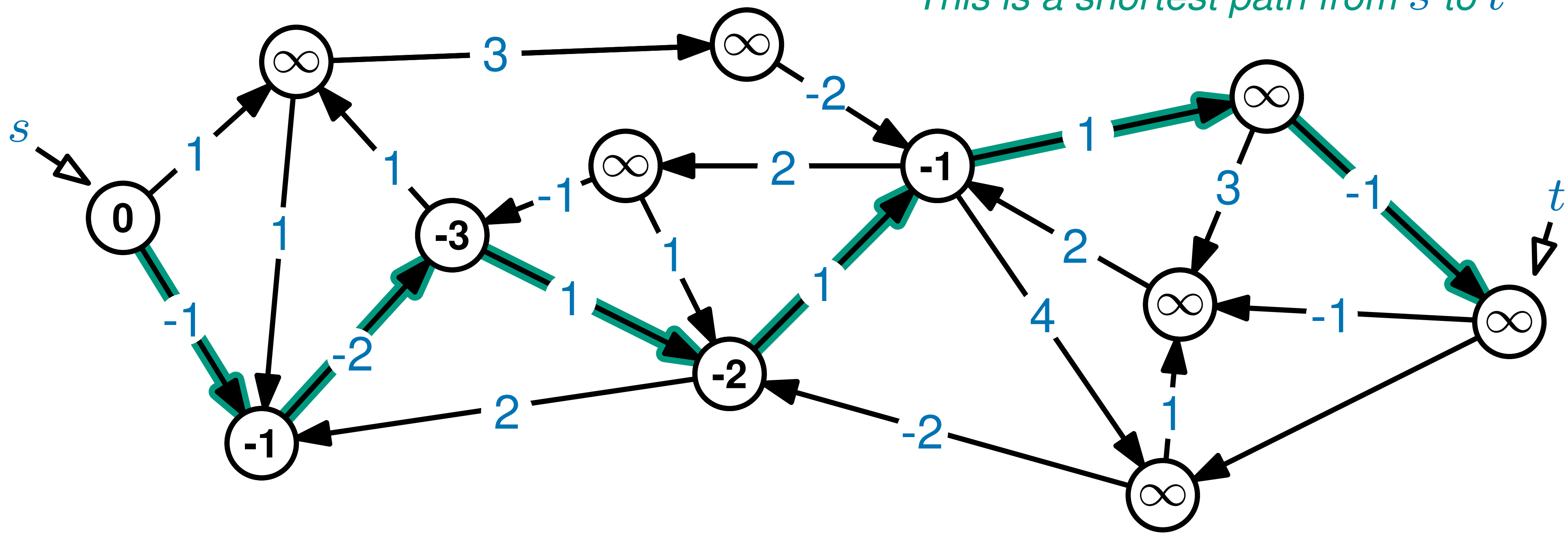
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 5:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

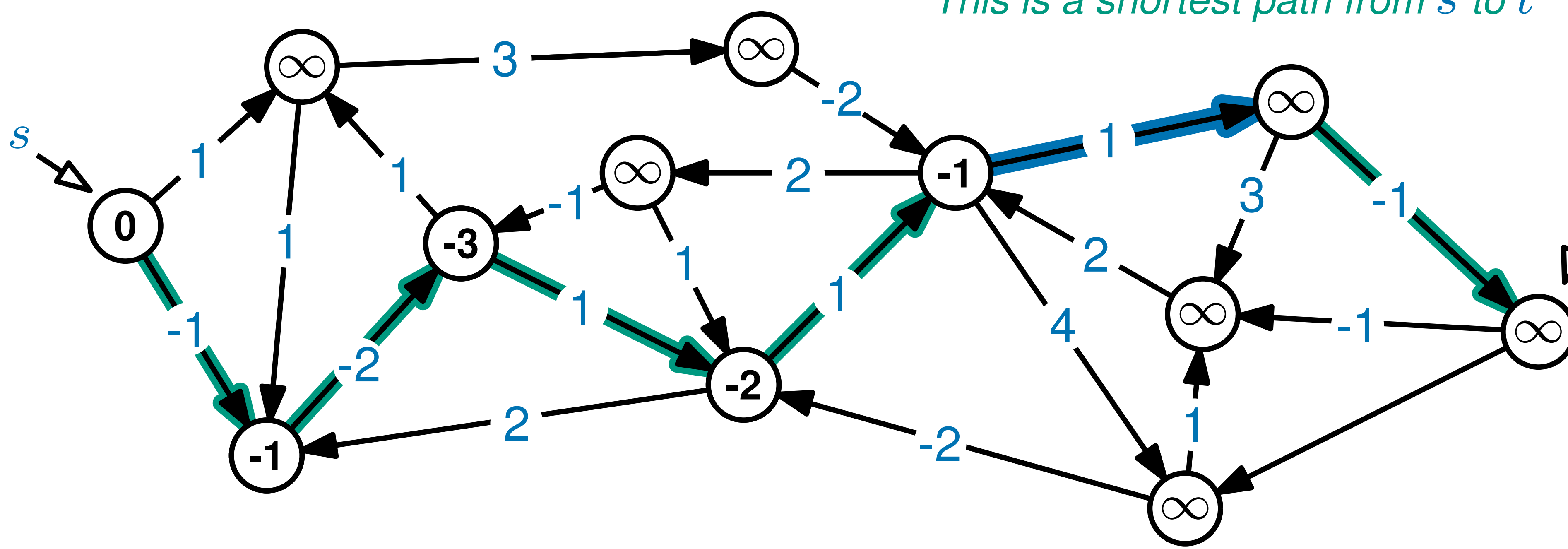
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 5:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

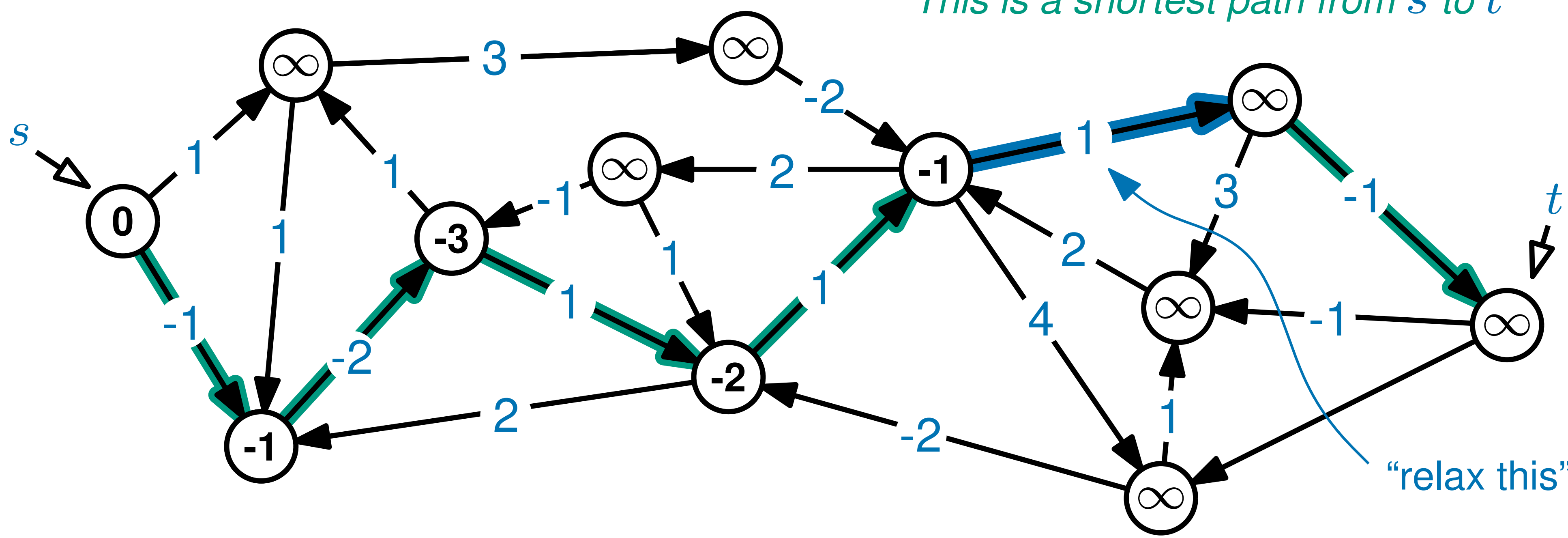
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 5:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

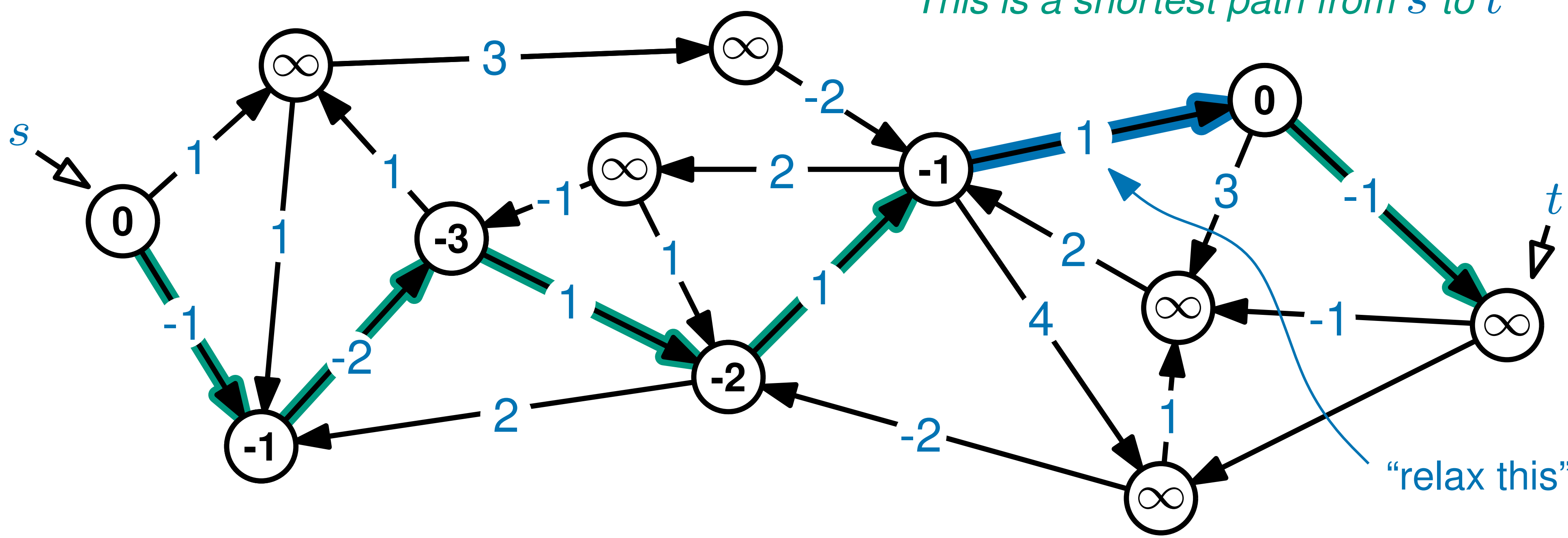
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 5:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

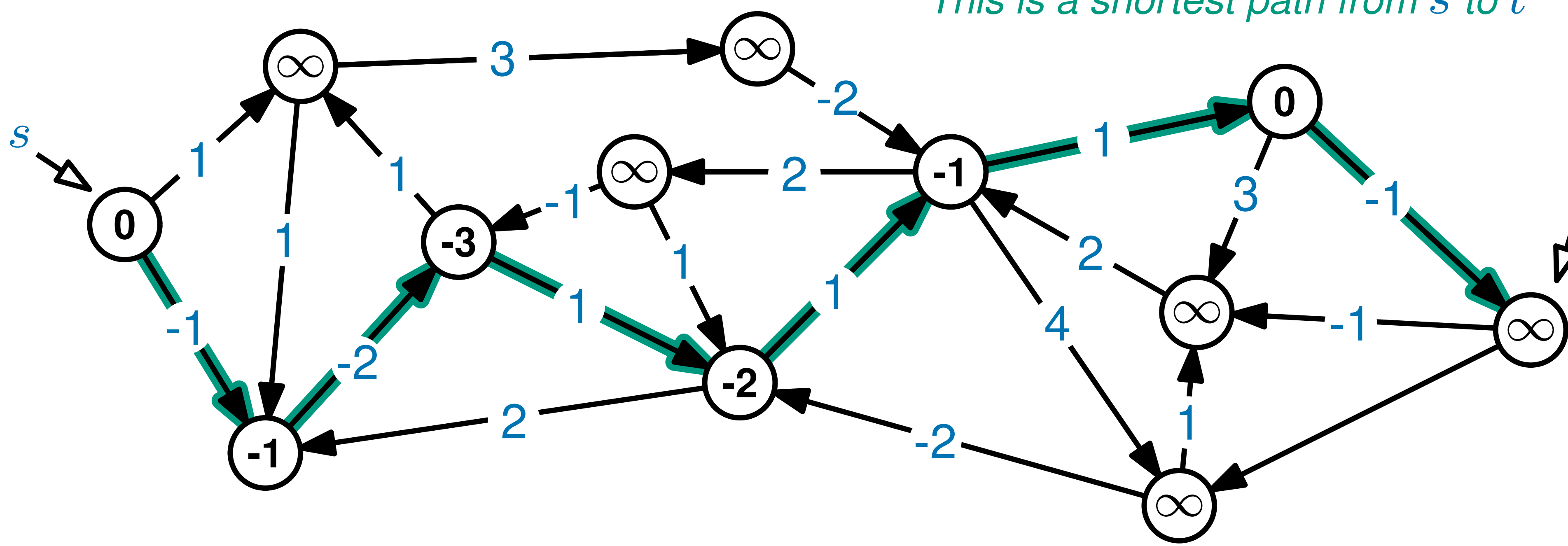
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 5:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

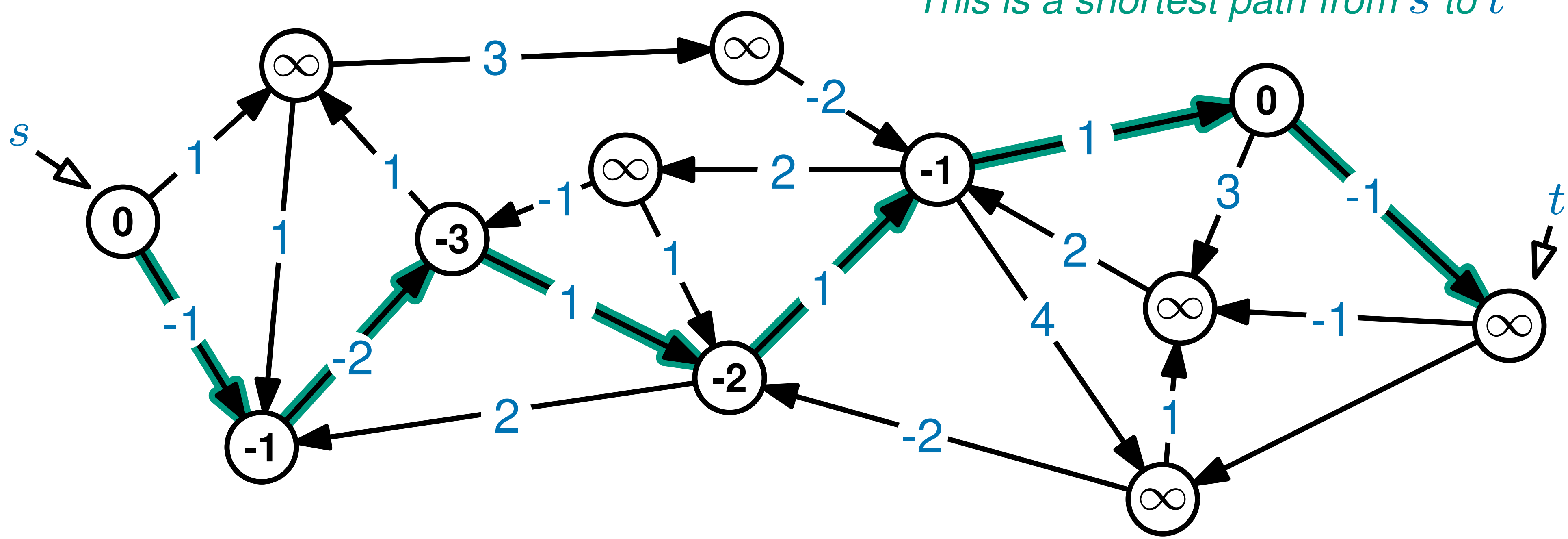
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 6:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

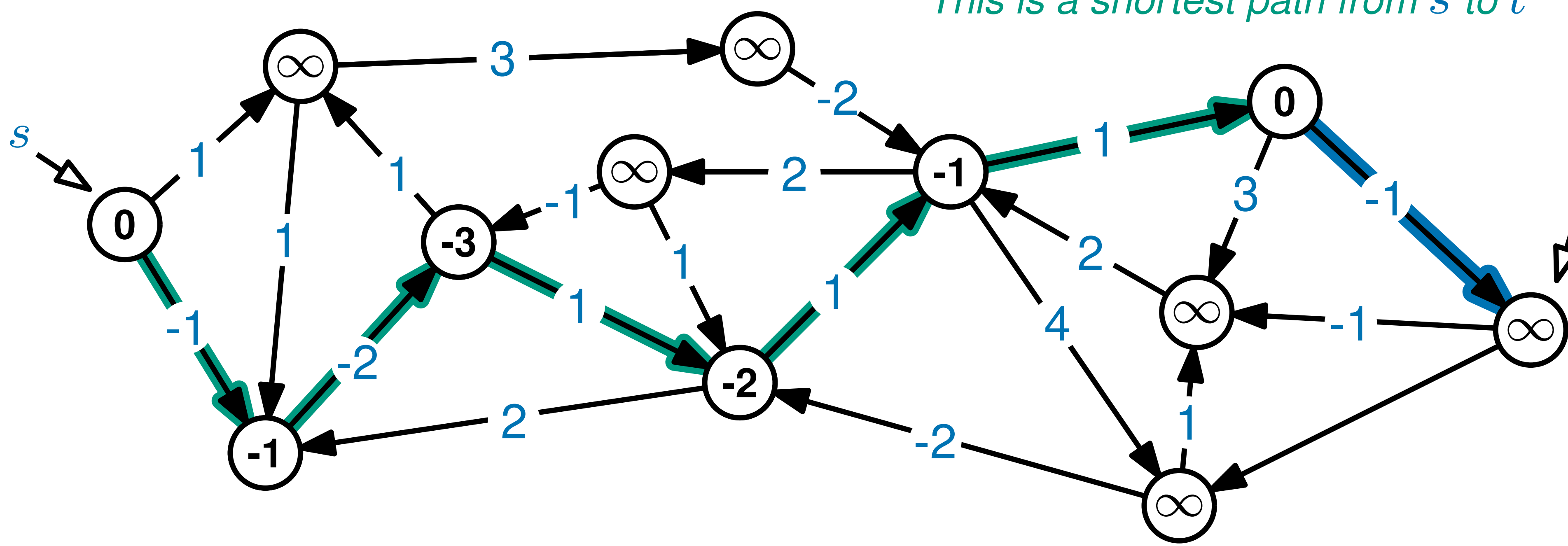
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 6:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

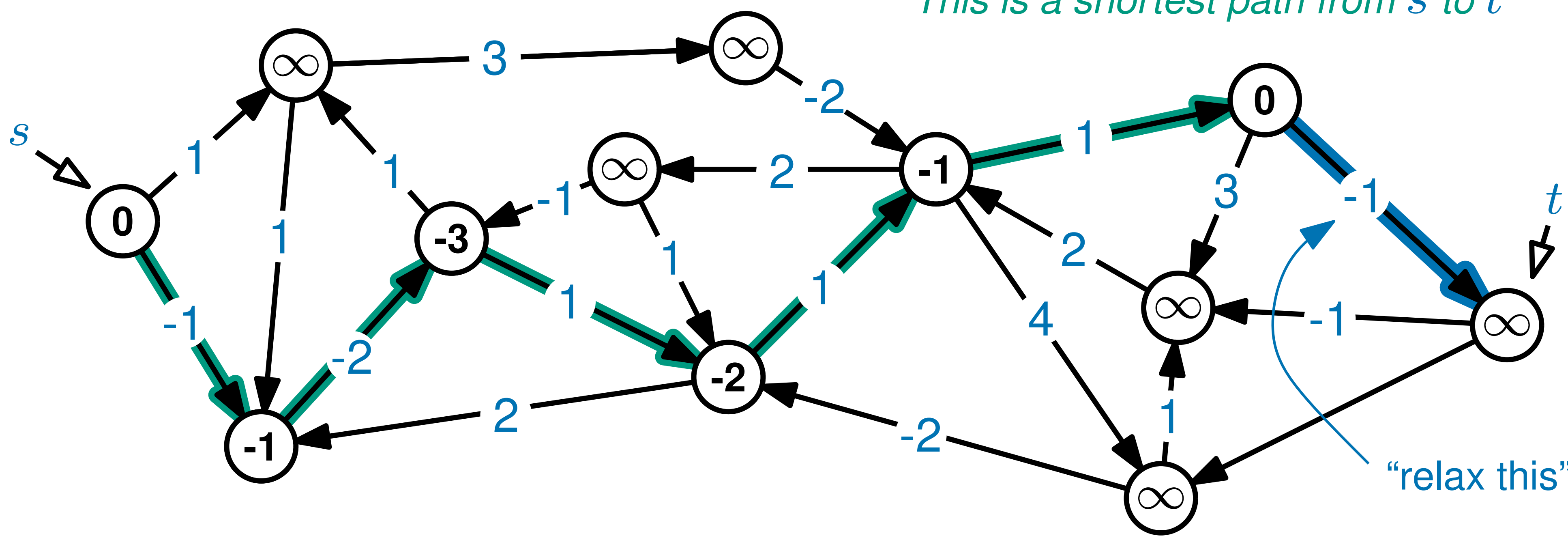
The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 6:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

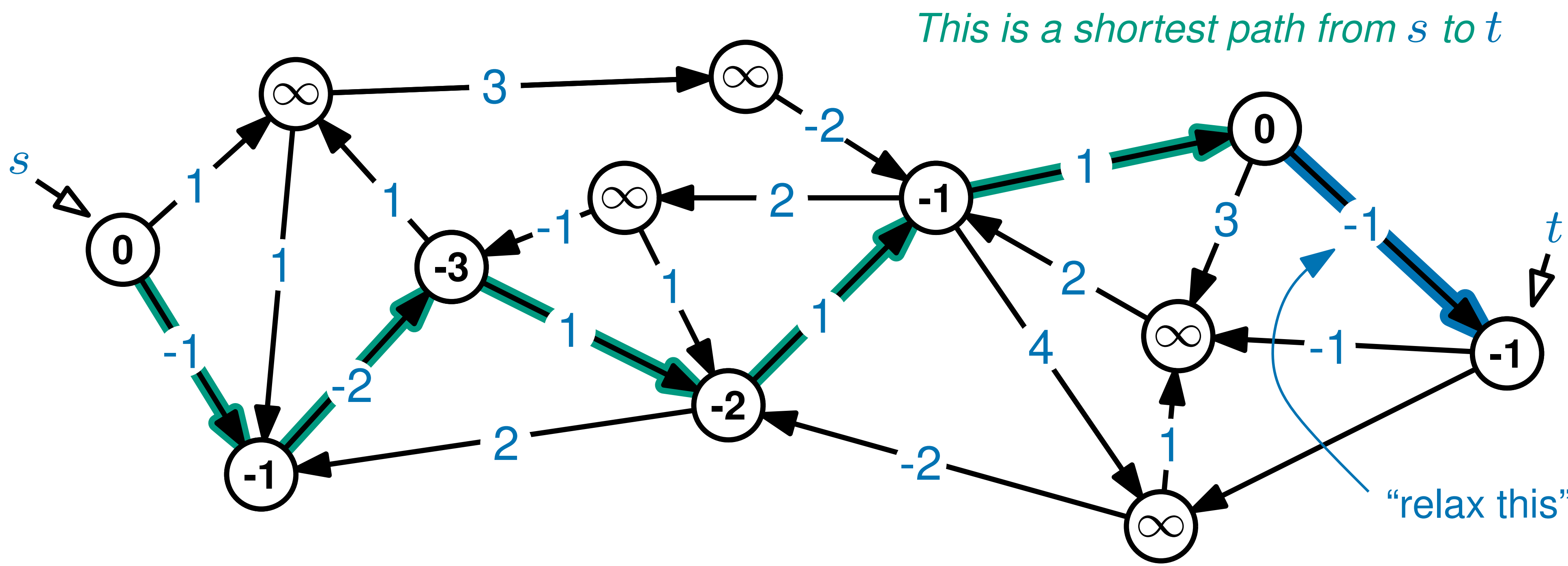
is the i-th edge in a shortest path from s to some vertex t

The proof idea

Imagine a different algorithm where in each iteration. . .

you only relax one edge *(rather than all edges)*

Iteration 6:



Further, imagine that (magically or otherwise), the edge that you relax in iteration i is the i-th edge in a shortest path from s to some vertex t

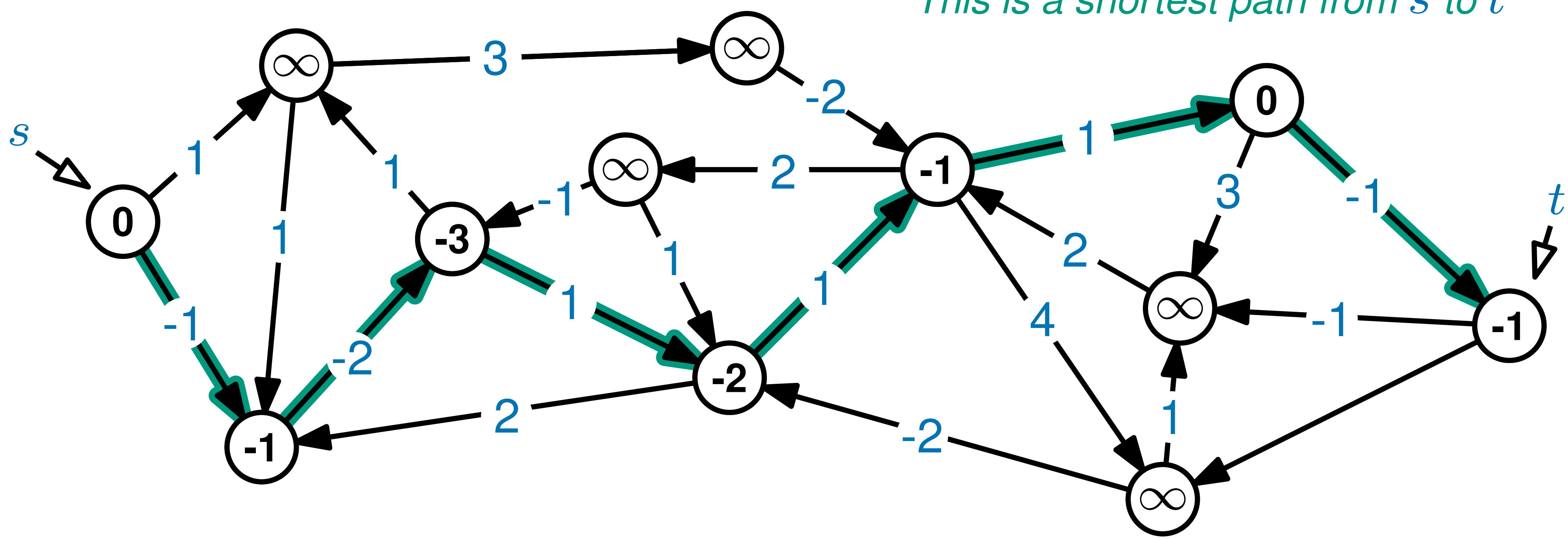
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 6:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

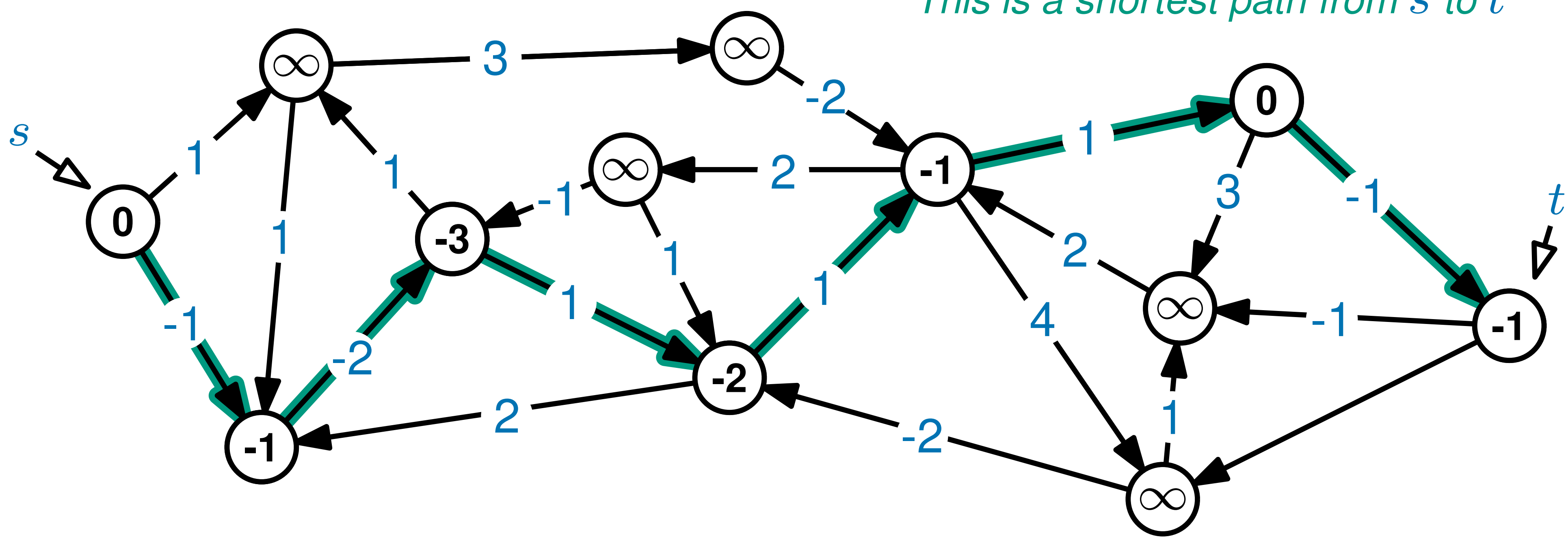
The proof idea

Imagine a different algorithm where in each iteration...

you only relax one edge *(rather than all edges)*

Iteration 6:

This is a shortest path from s to t



Further, imagine that (magically or otherwise), the edge that you relax in iteration i

is the i-th edge in a shortest path from s to some vertex t

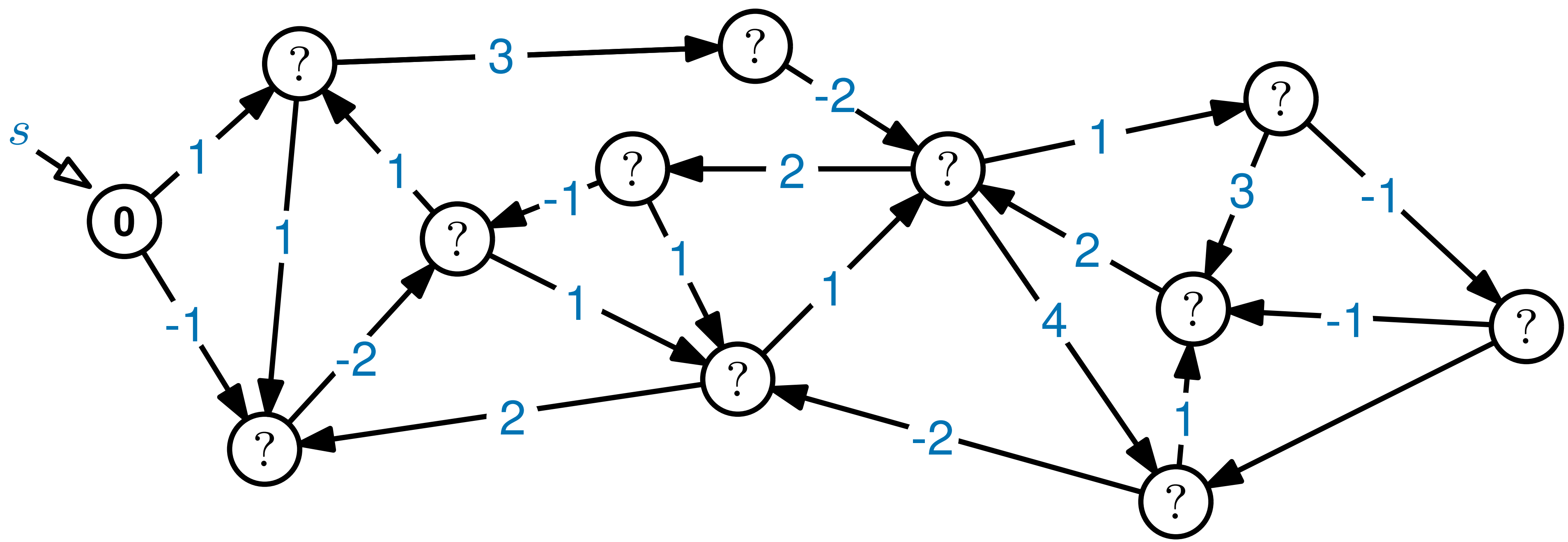
When this algorithm terminates...

$\texttt{dist}(t)$ is the length of the shortest path from s to t

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

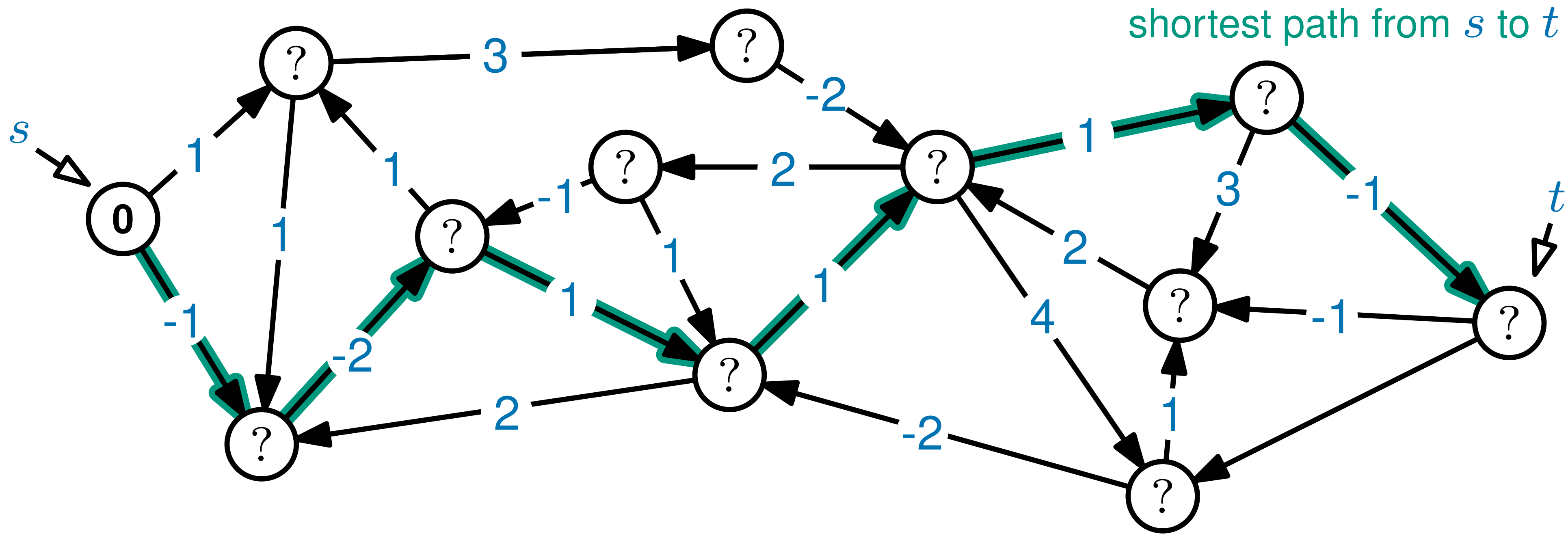


The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Consider the same shortest path from s to t

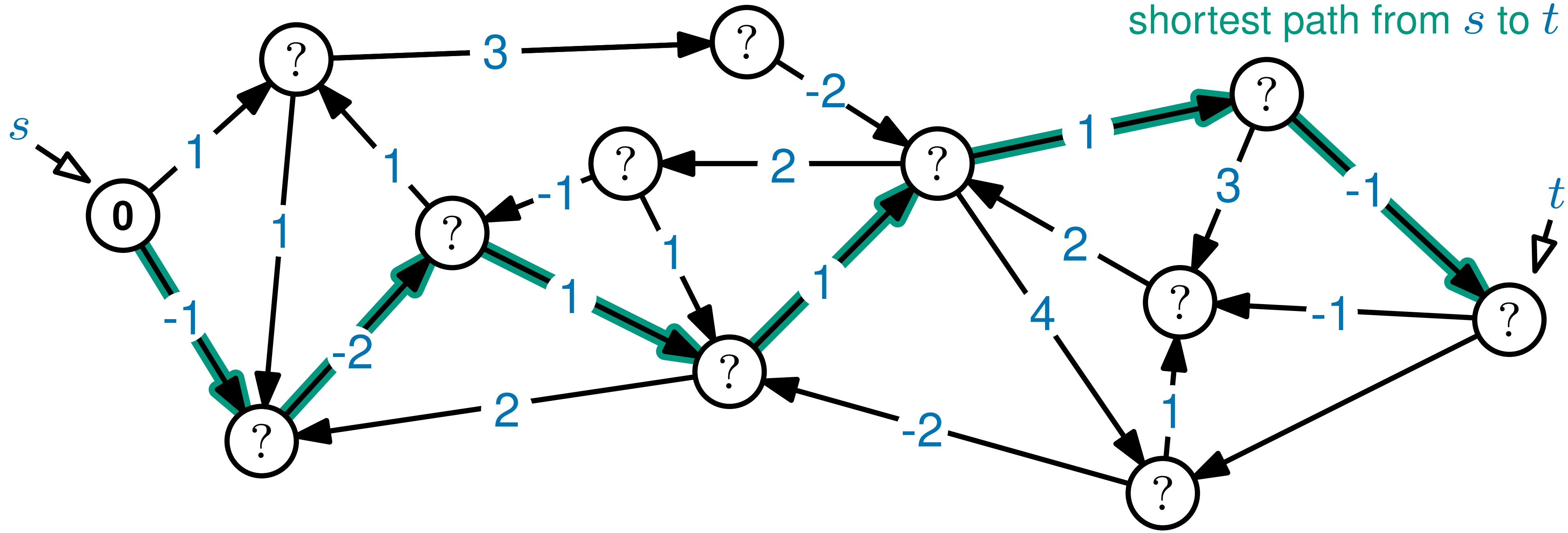


The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

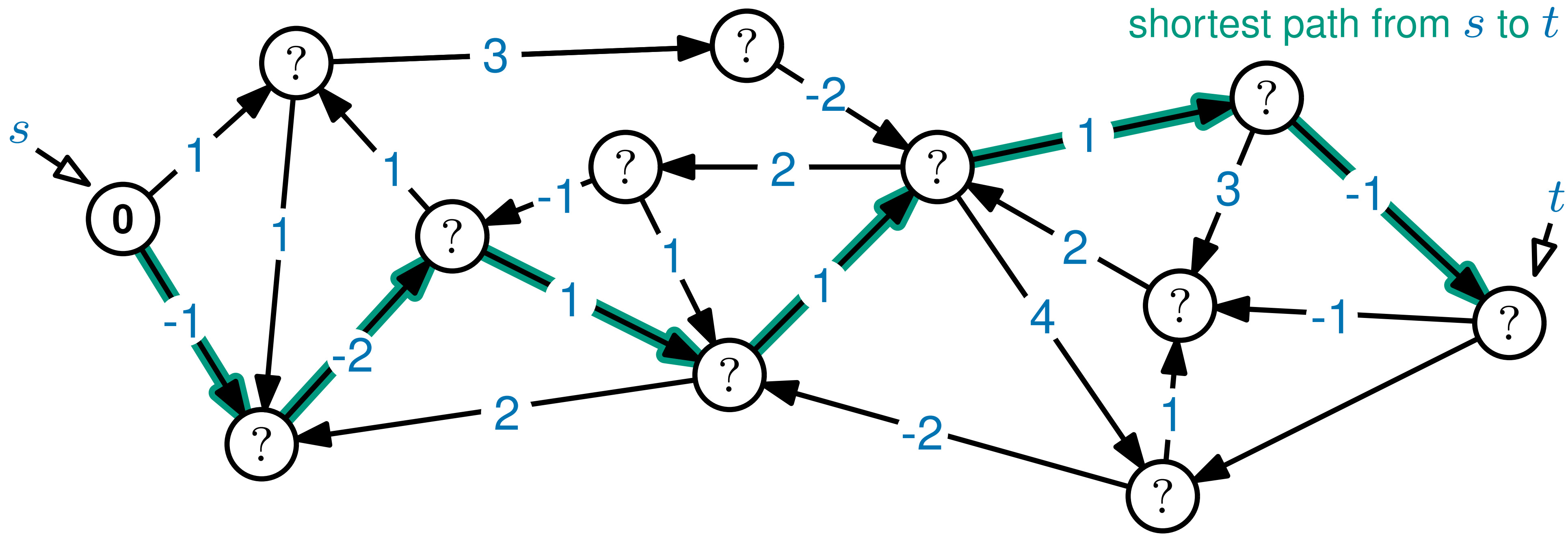
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

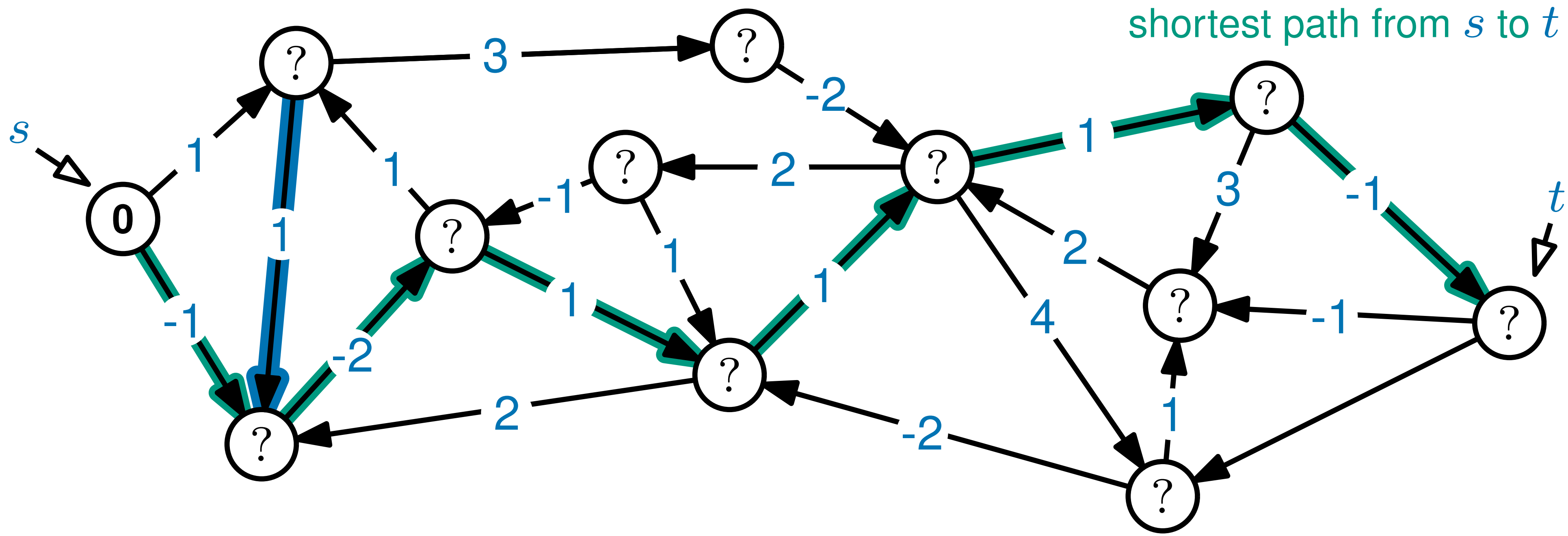
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 1:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

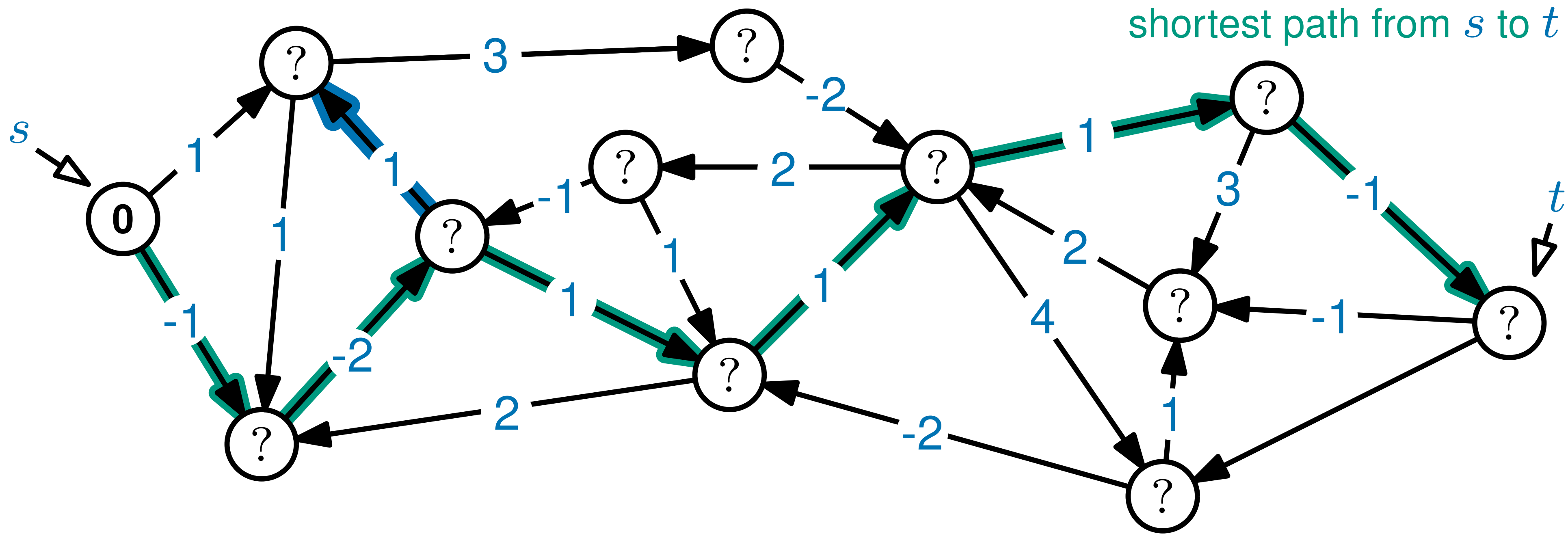
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration…

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t…

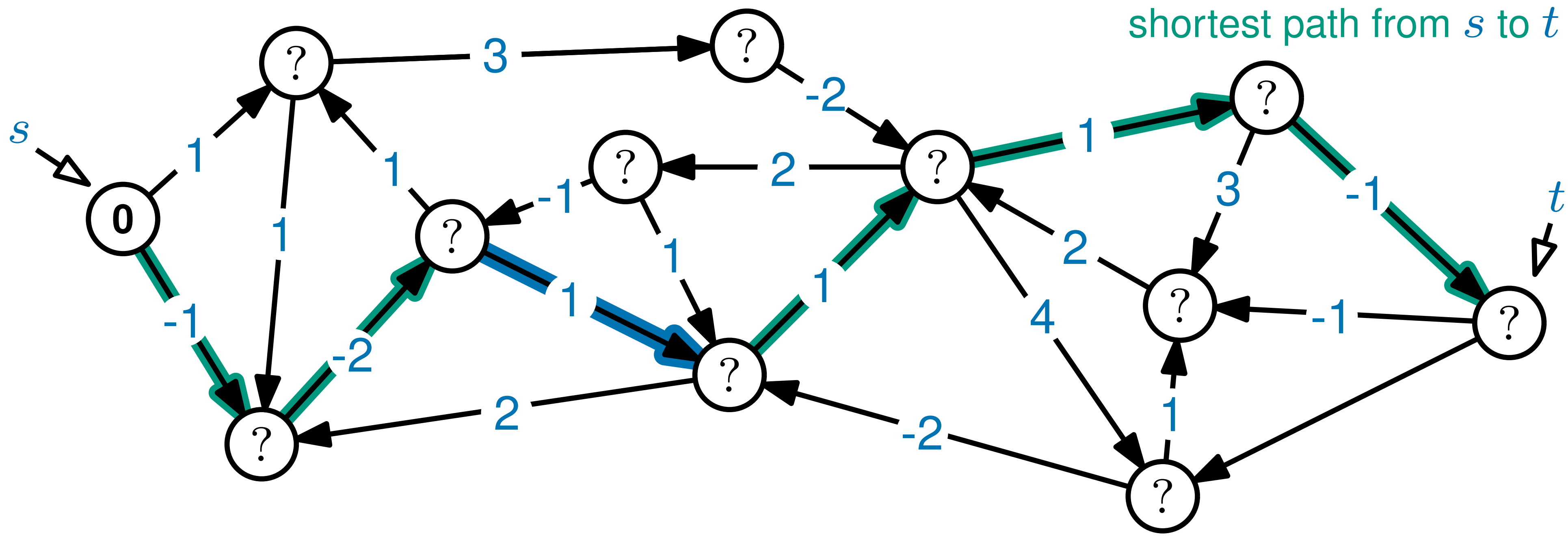
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

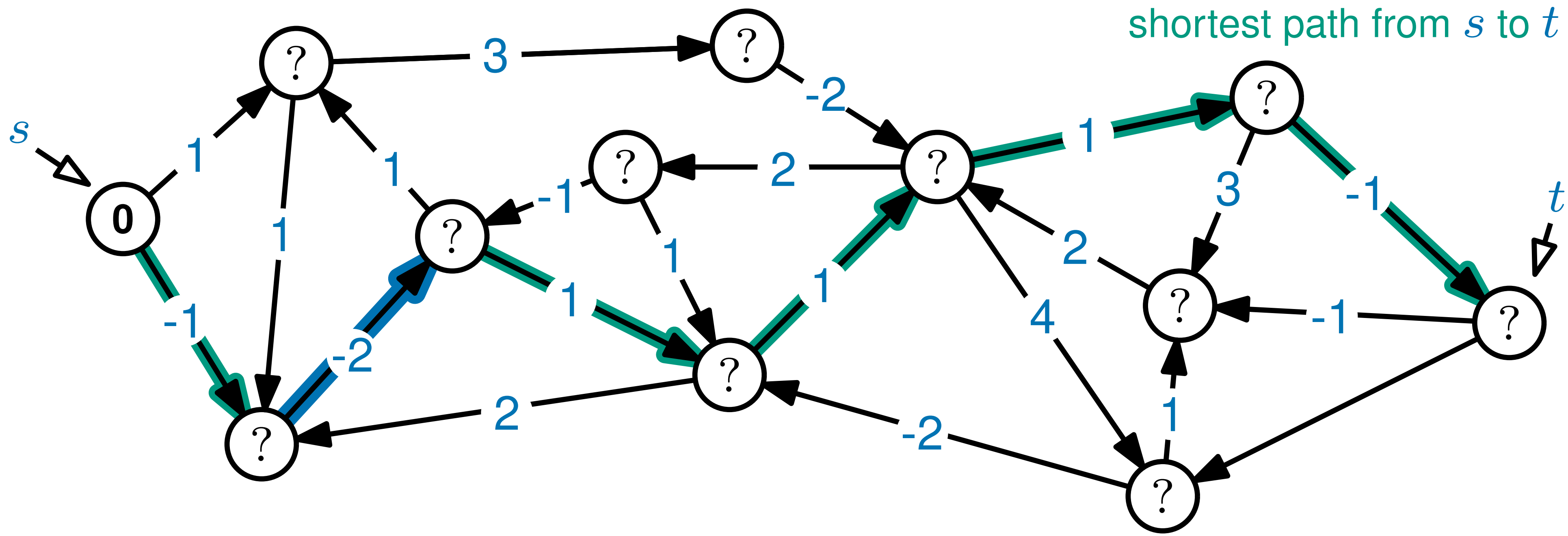
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration…

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t…

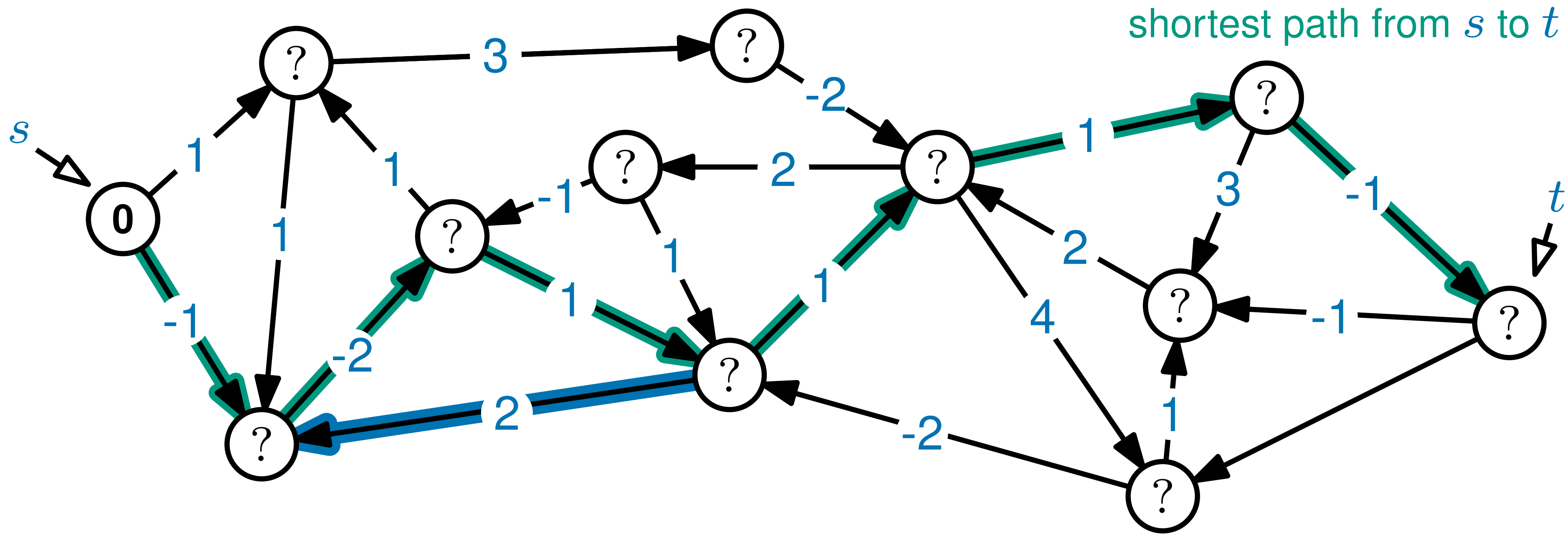
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

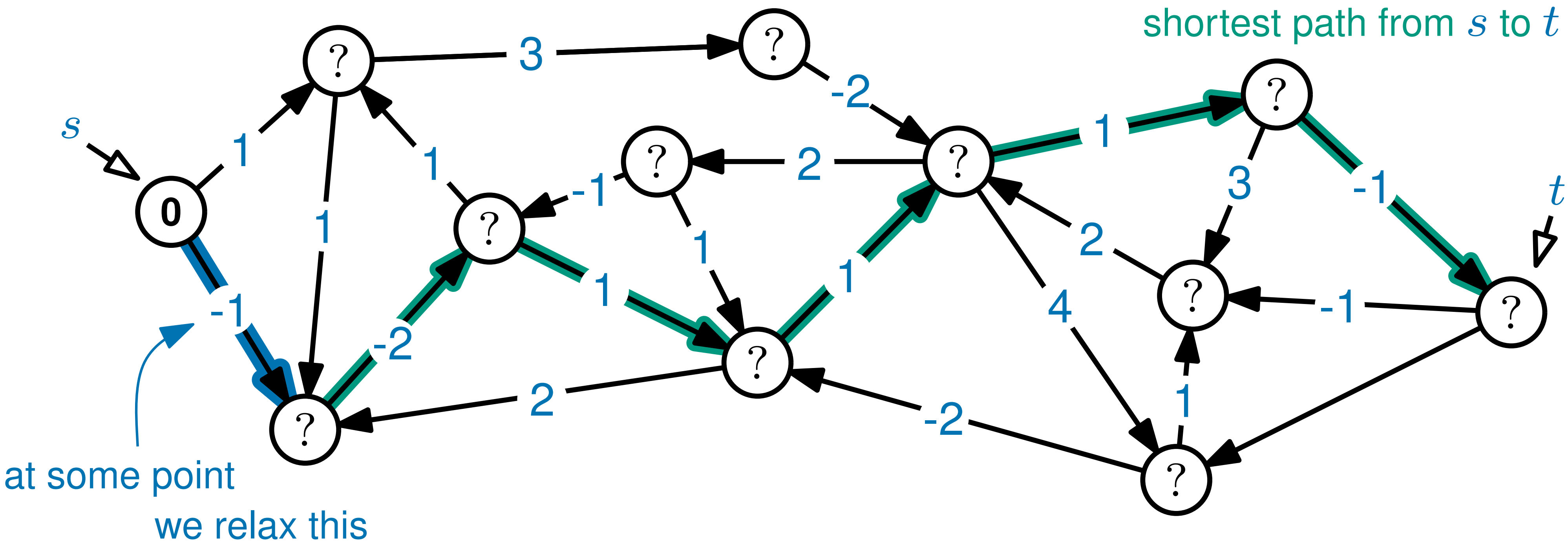
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:



At some point in iteration i

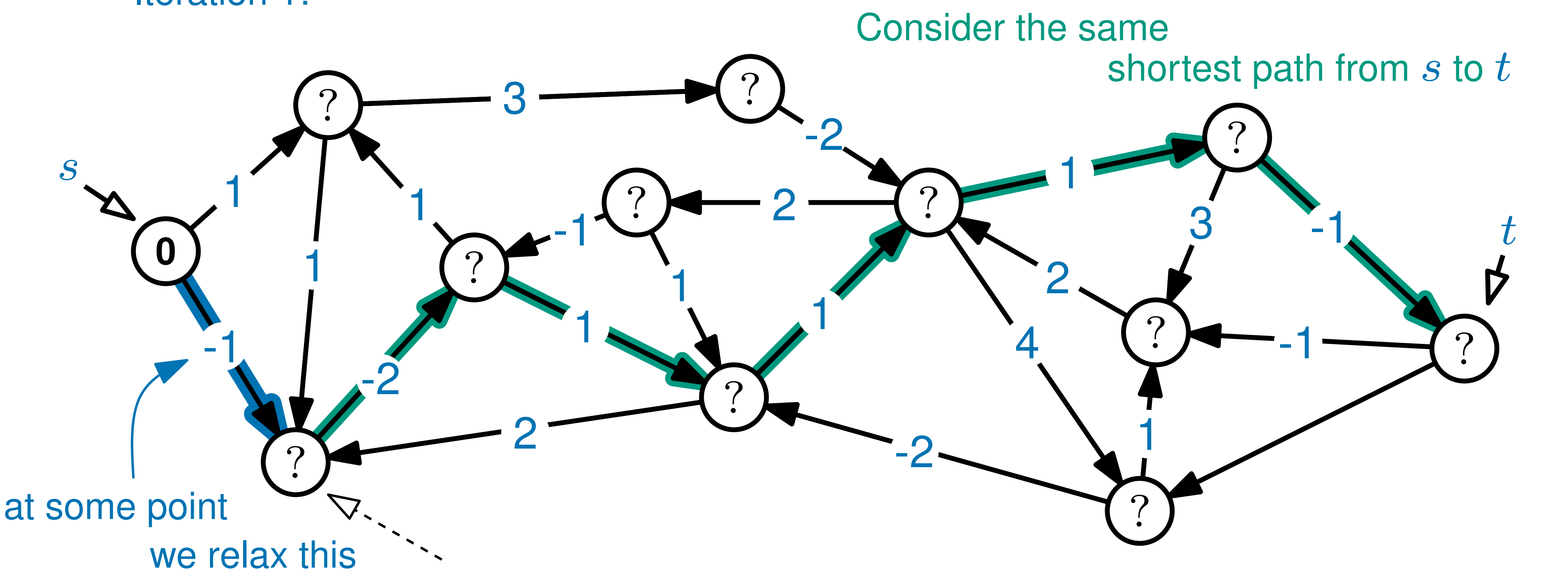
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:



Don't worry about what ? was before. Relax picks the smaller of

? and $\mathbf{0} + (-1)$ so...

At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

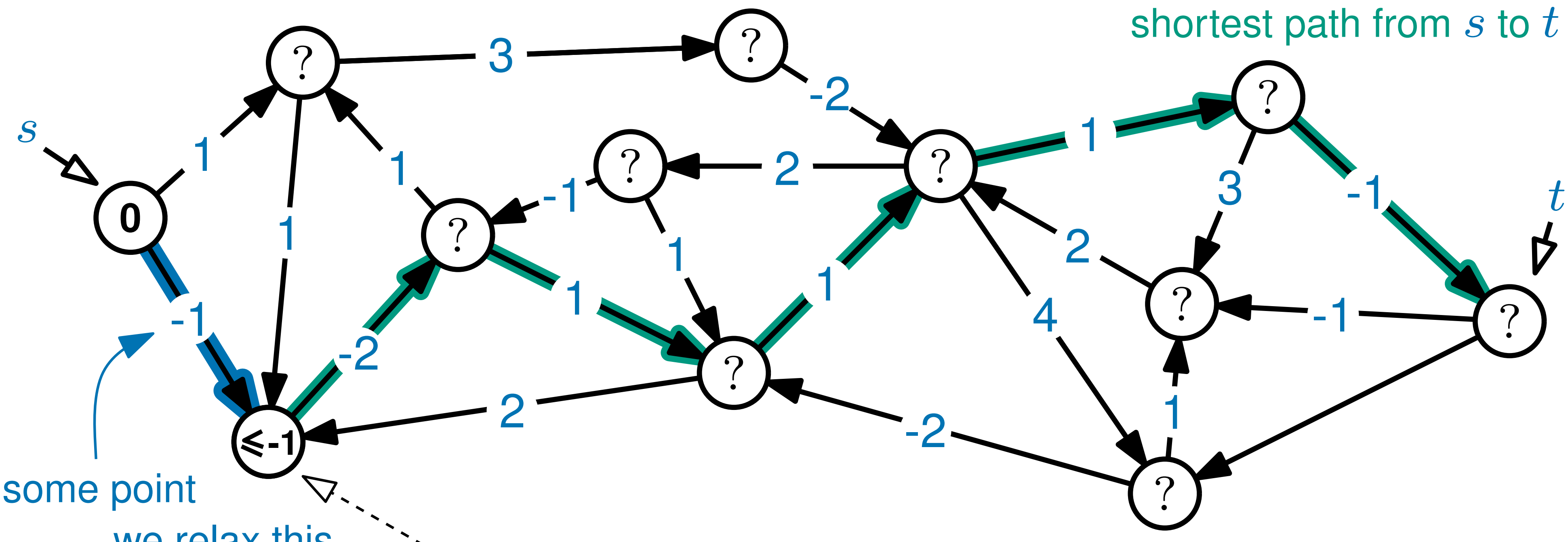
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



at some point we relax this

Don't worry about what ? was before. RELAX picks the smaller of ? and $\mathbf{0} + (-1)$ so...

At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

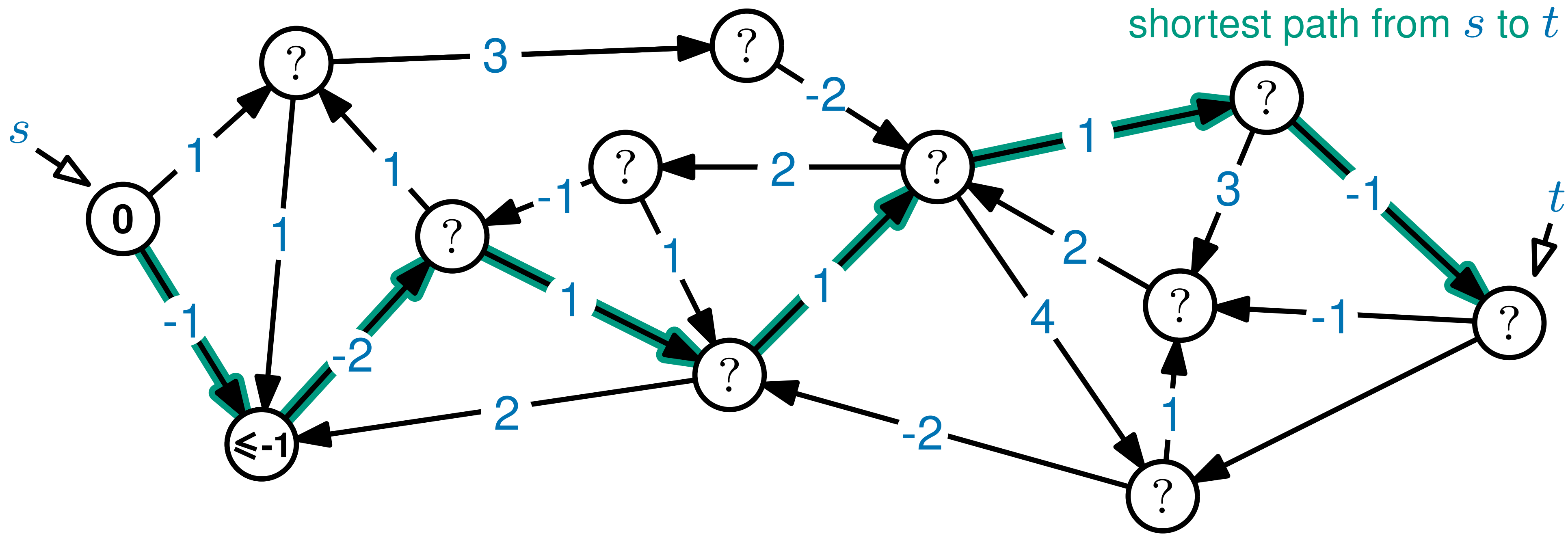
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

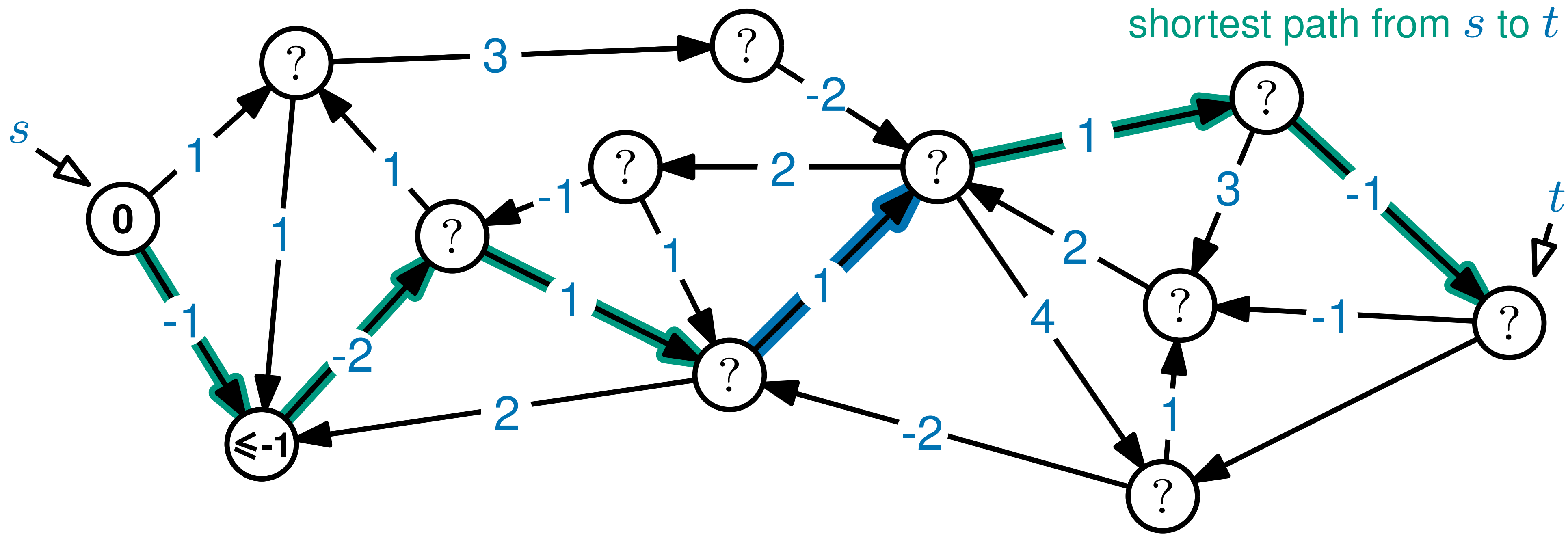
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

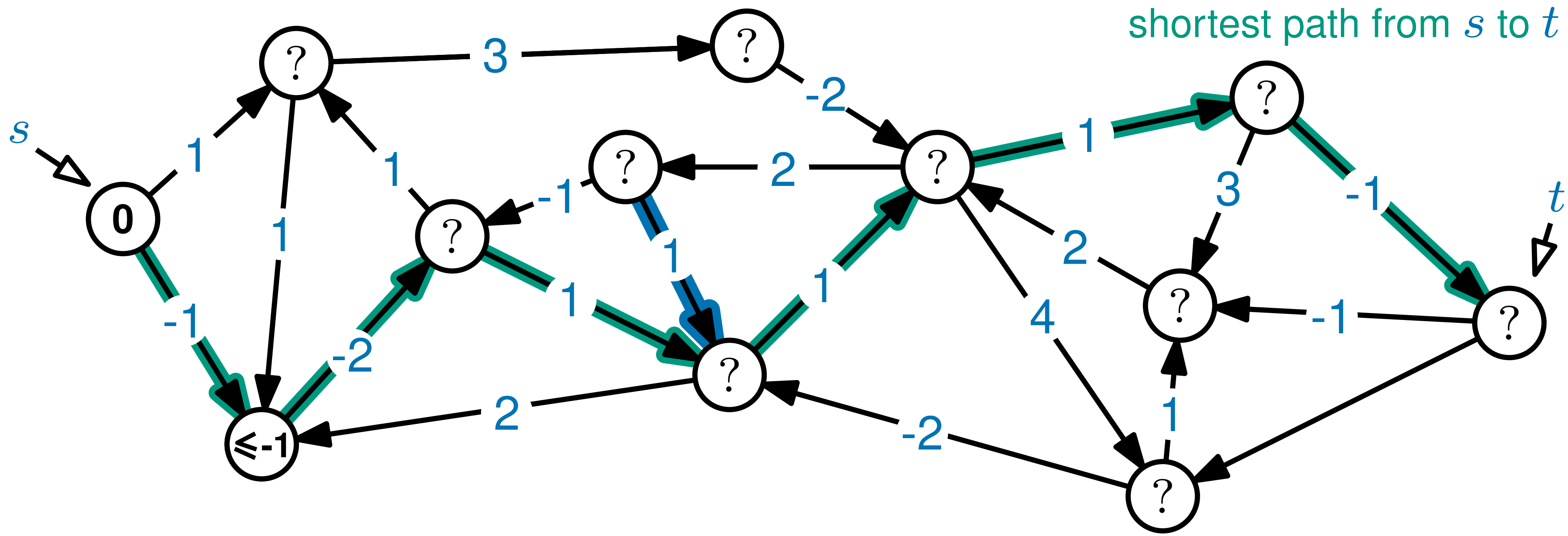
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

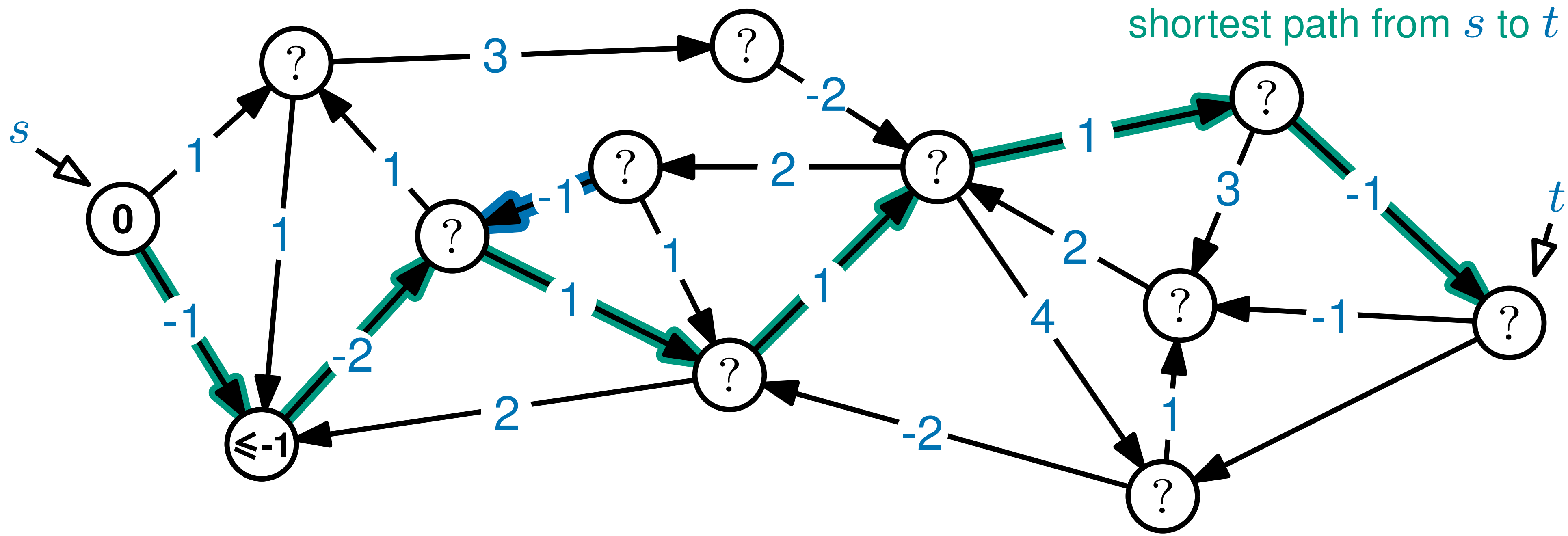
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

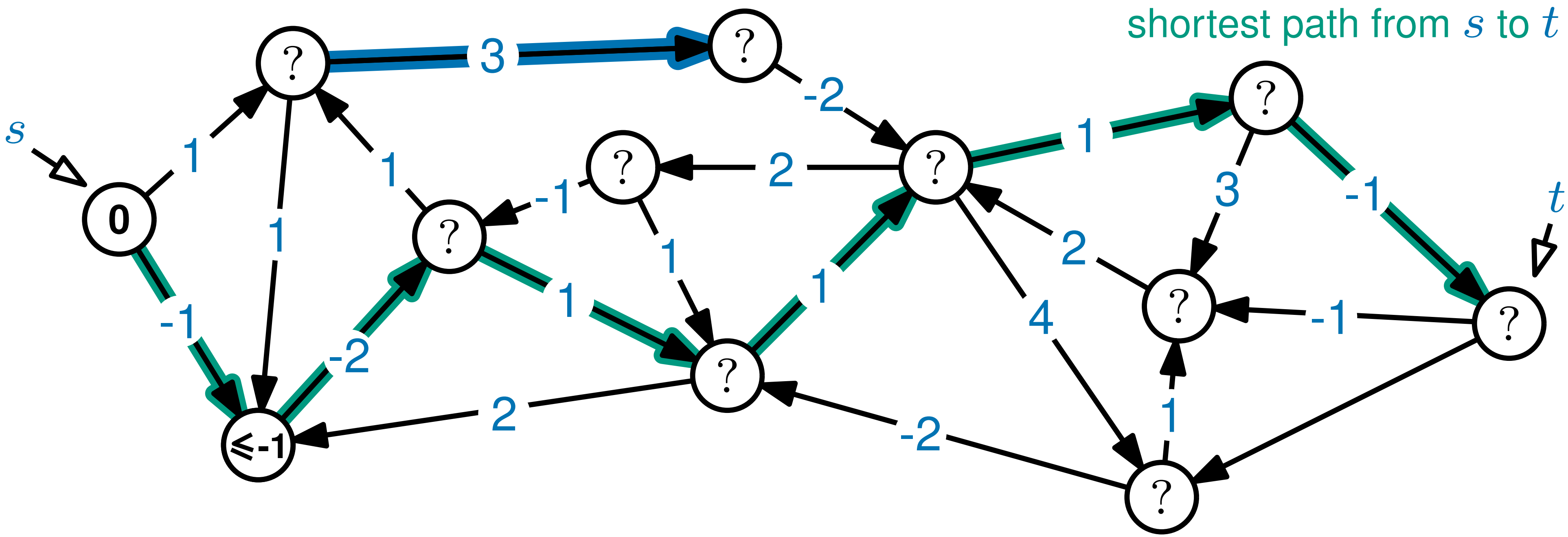
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...
you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same
shortest path from s to t



At some point in iteration i
you relax the i-th edge in the shortest path from s to t...

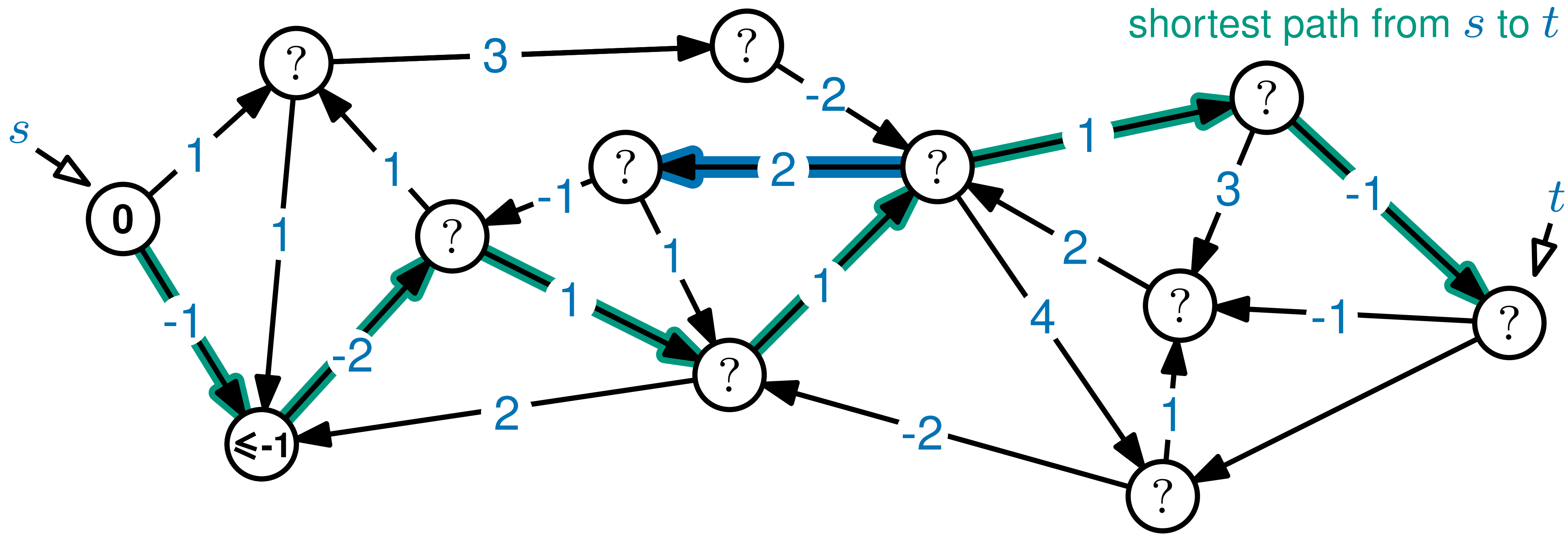
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

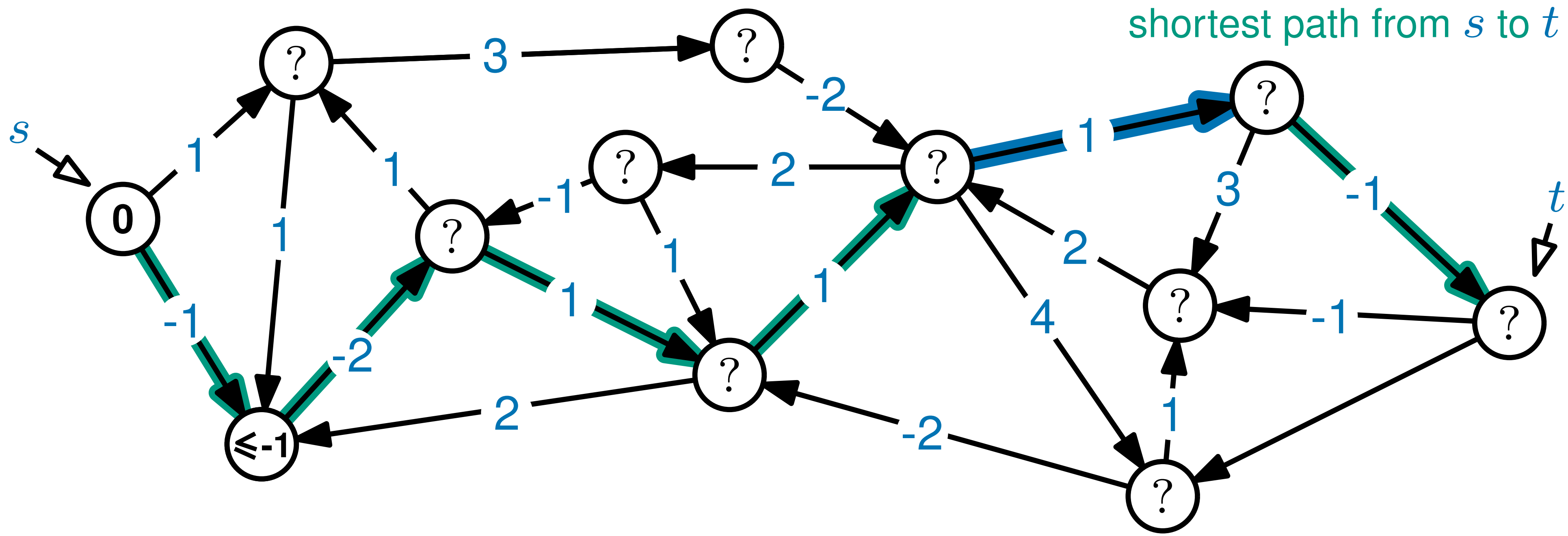
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration…

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t…

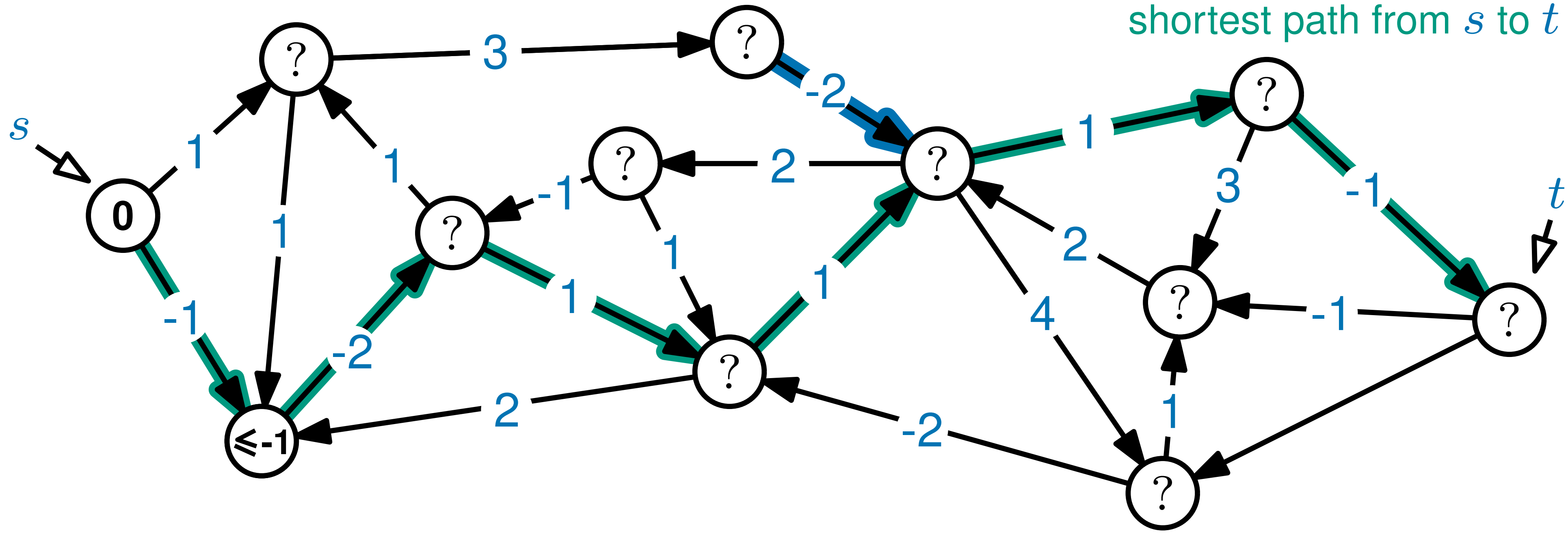
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

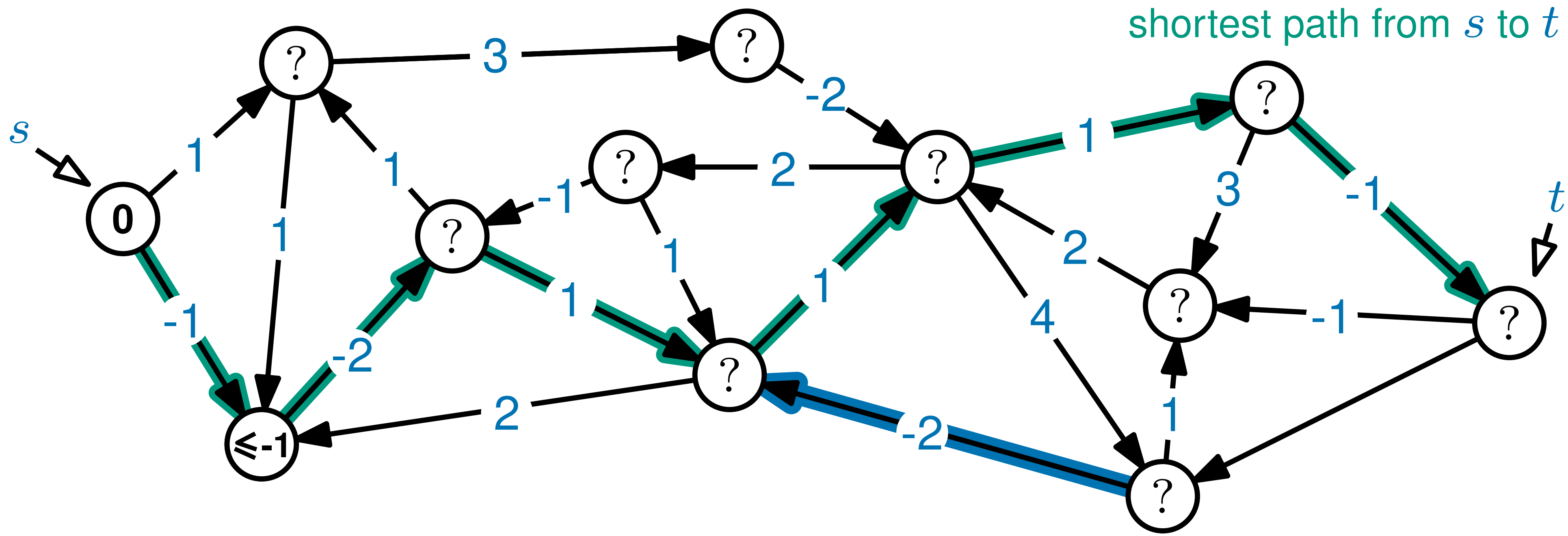
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

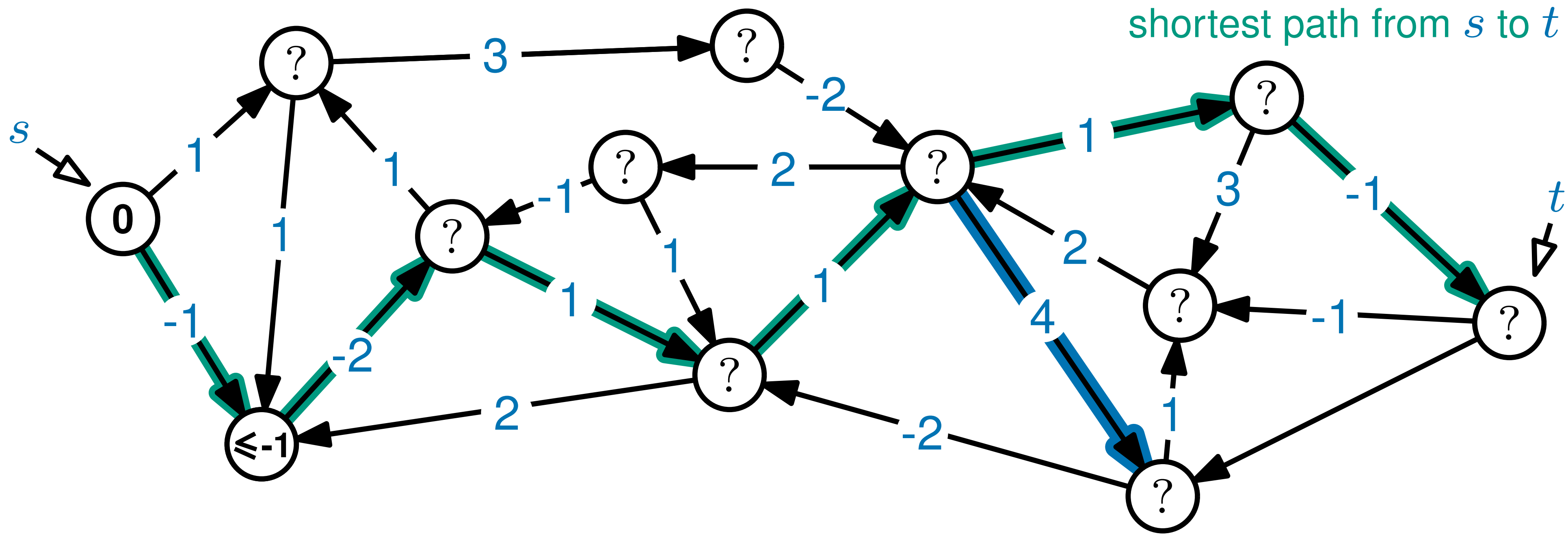
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

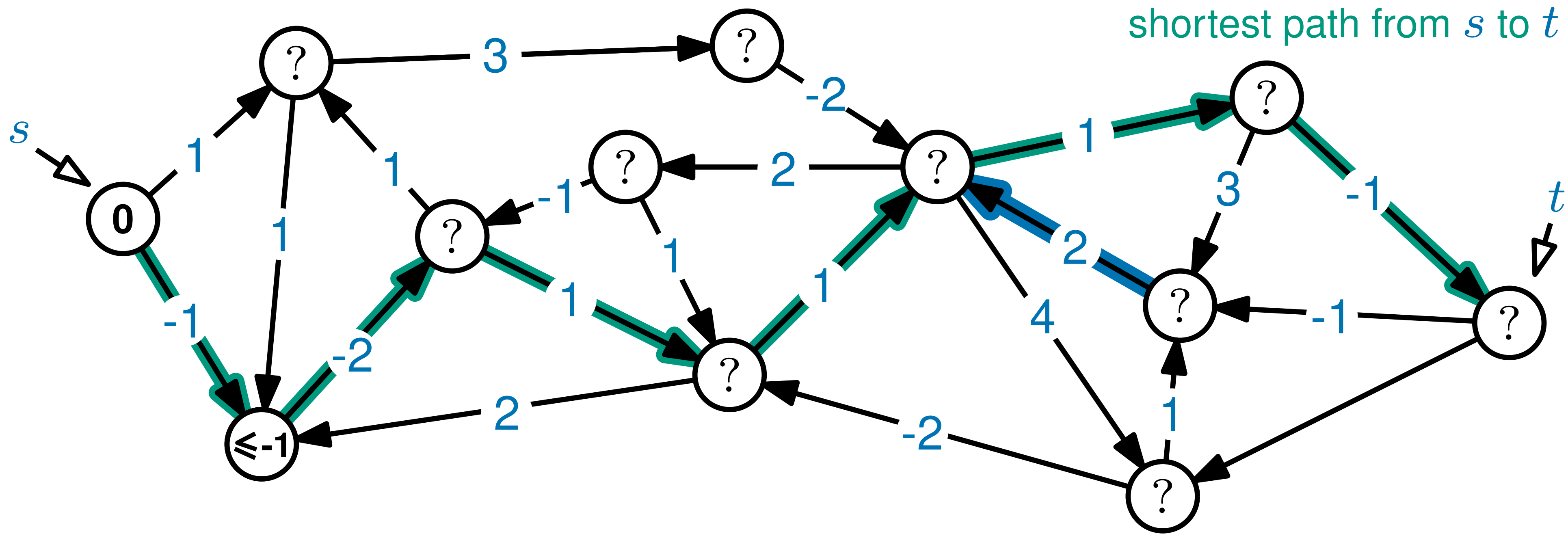
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

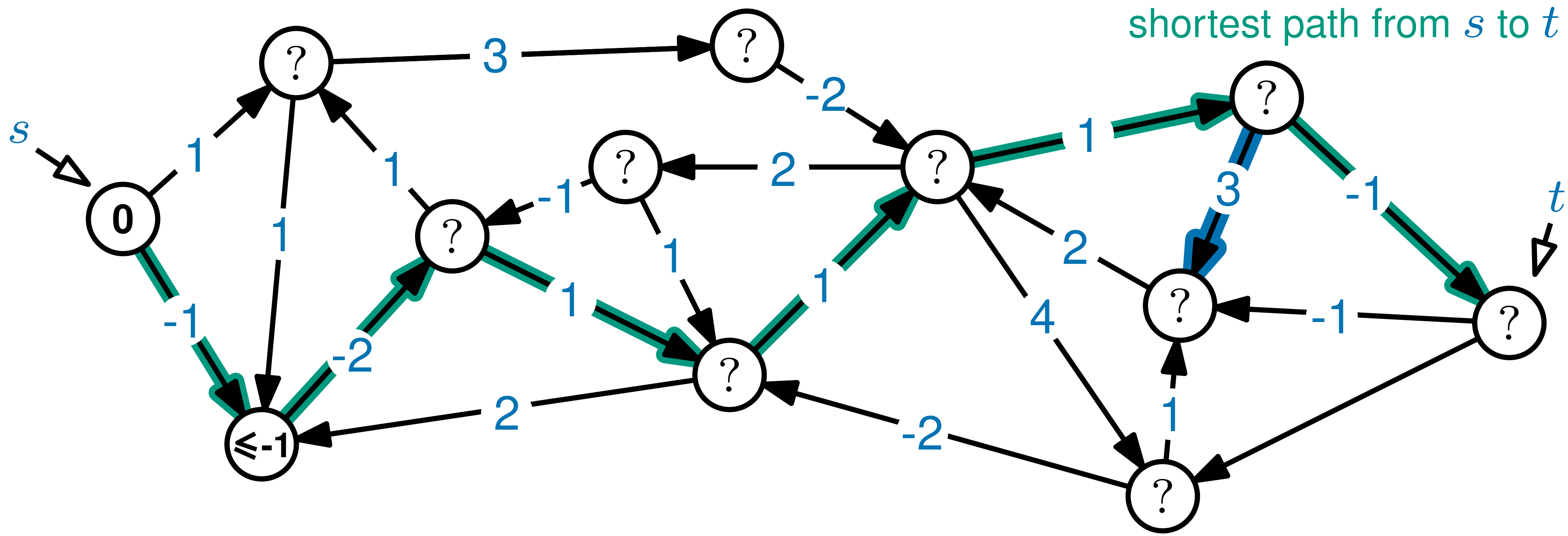
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:



At some point in iteration i

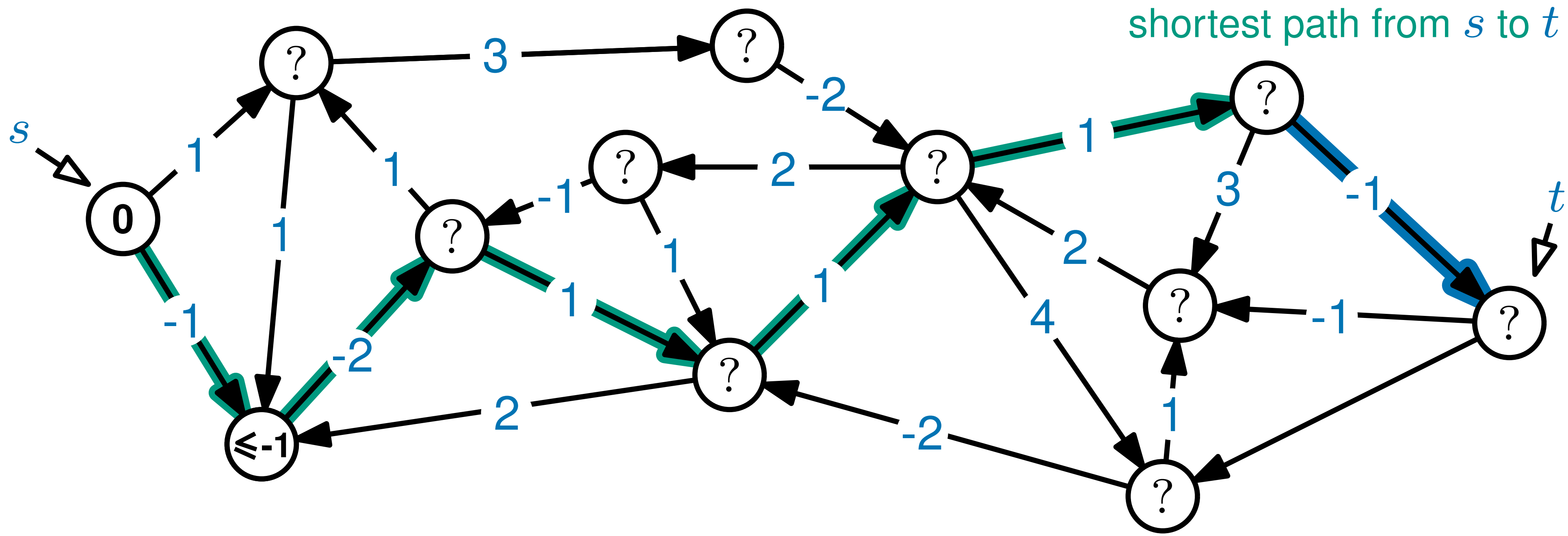
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

University of BRISTOL

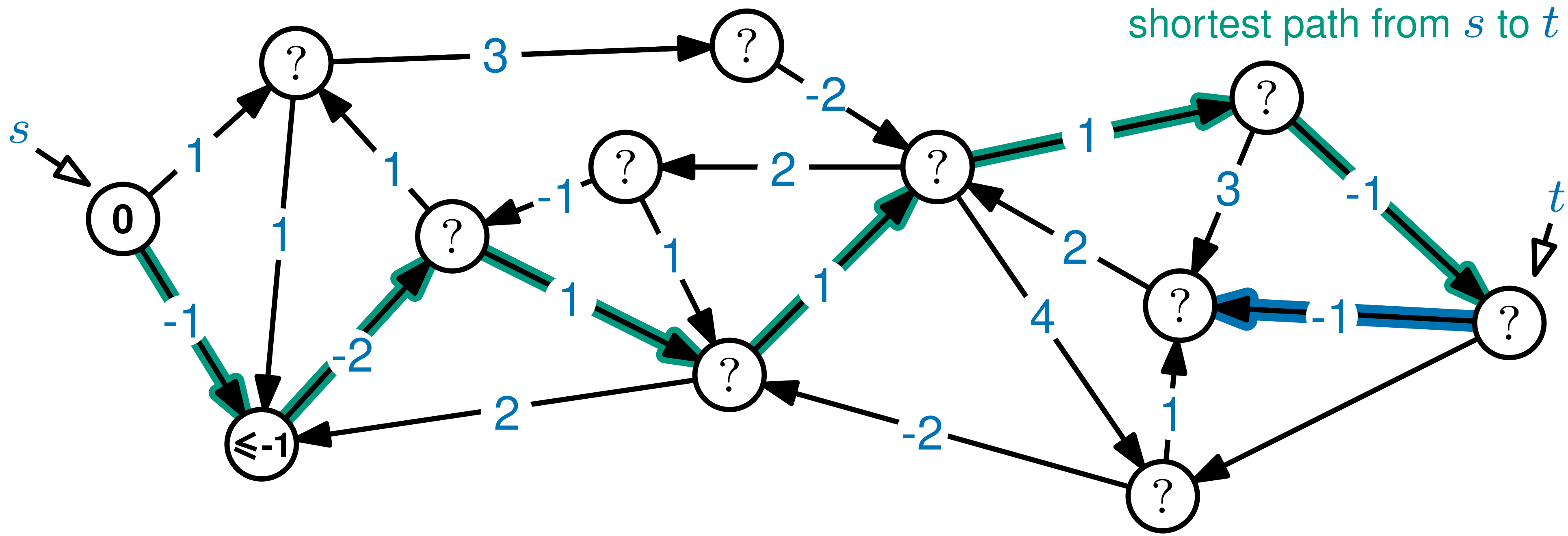
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

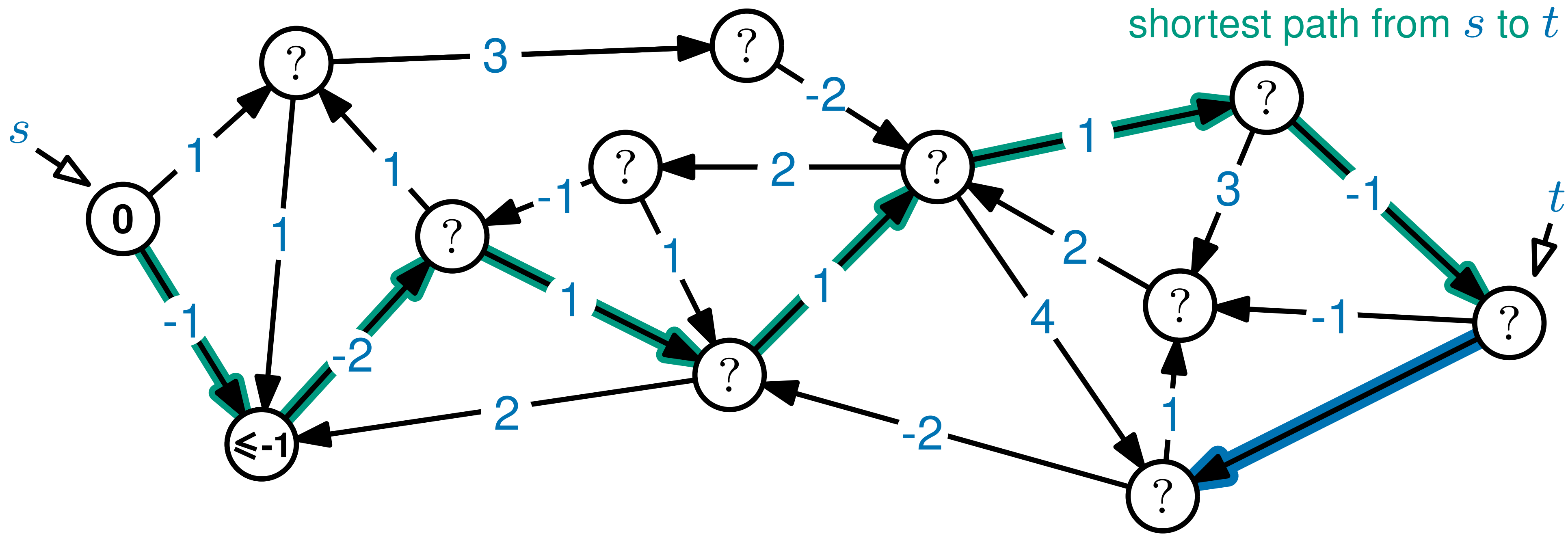
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

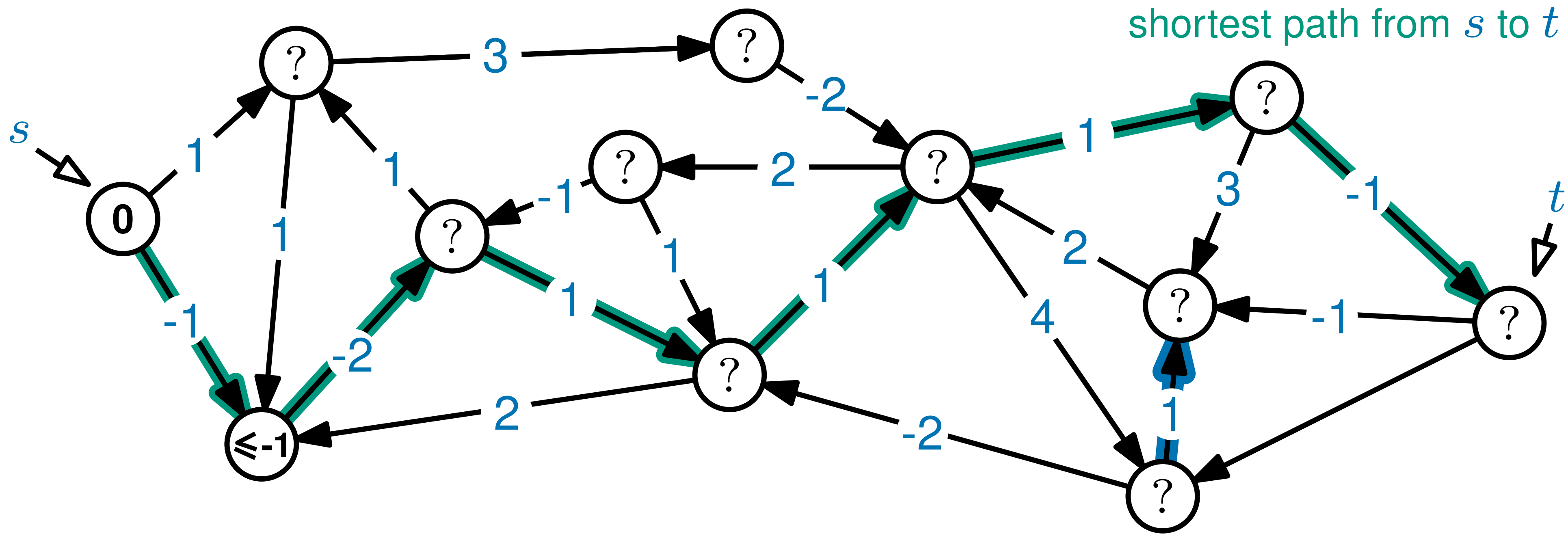
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 1:

Consider the same shortest path from s to t



At some point in iteration i

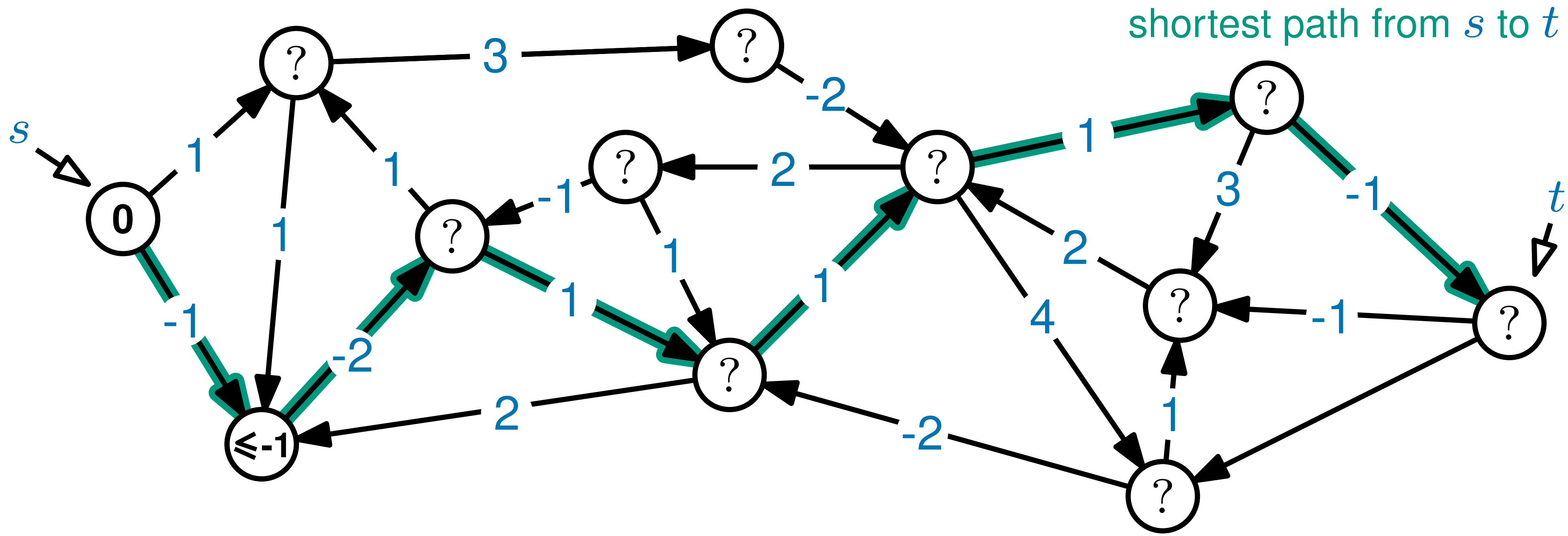
you relax the i-th edge in the shortest path from s to t. . .

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 1:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

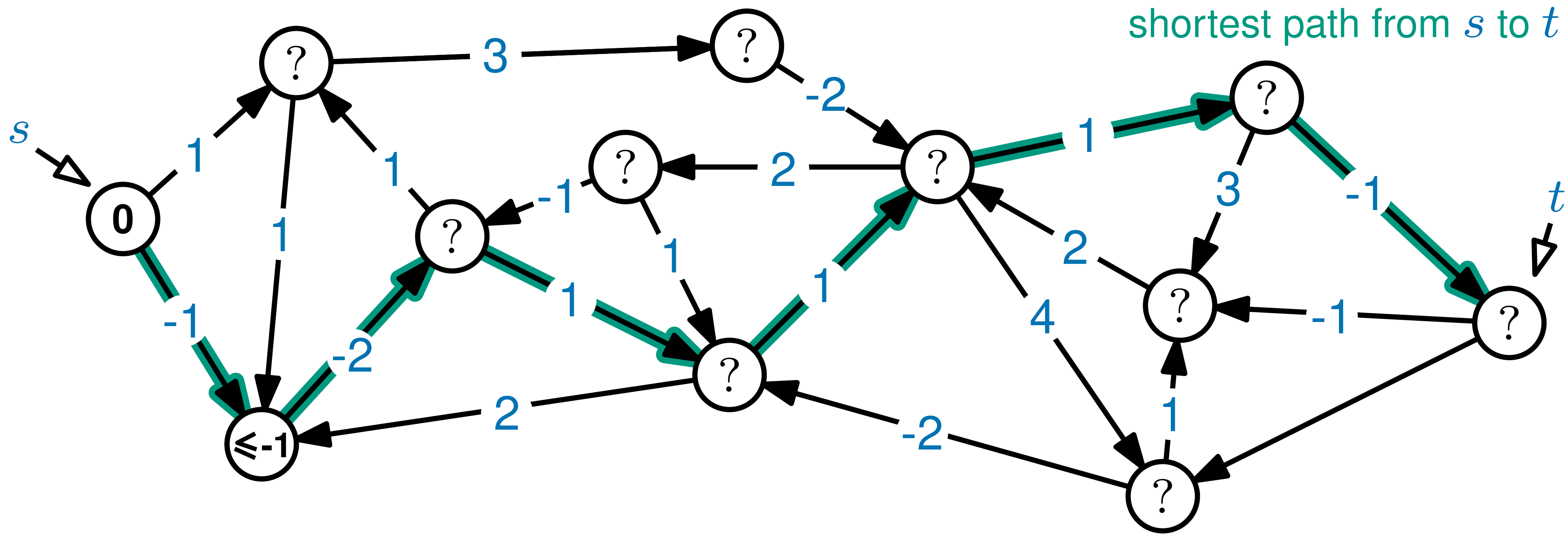
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

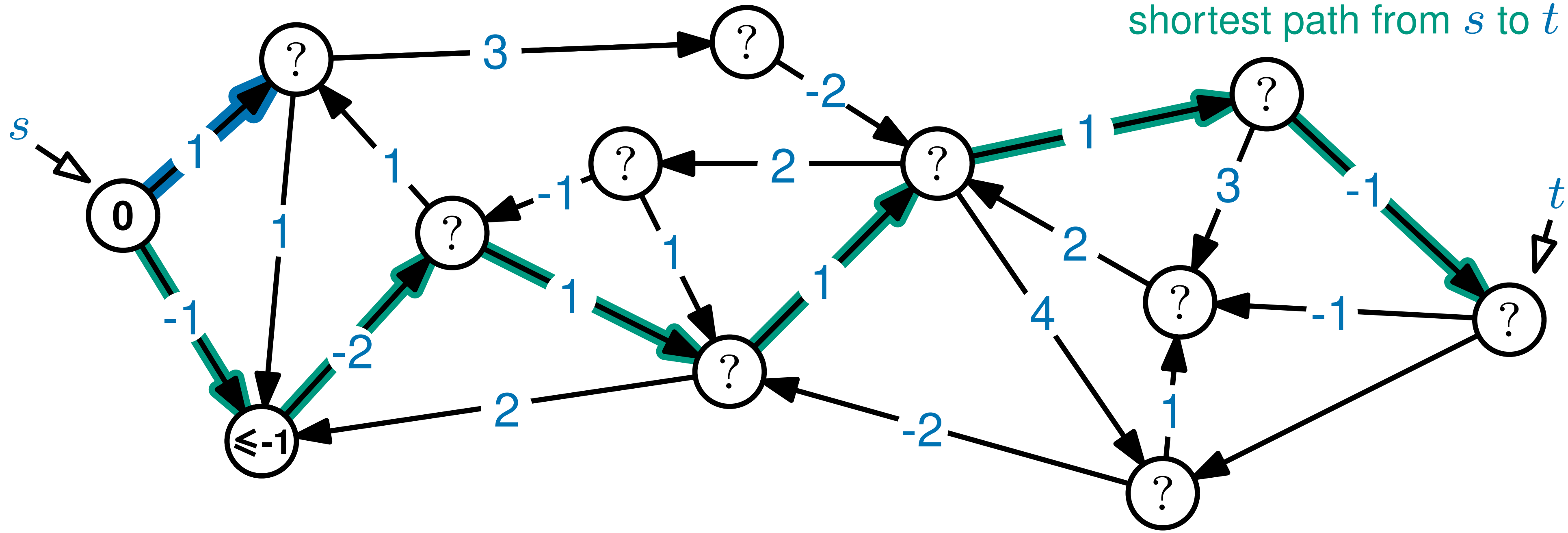
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

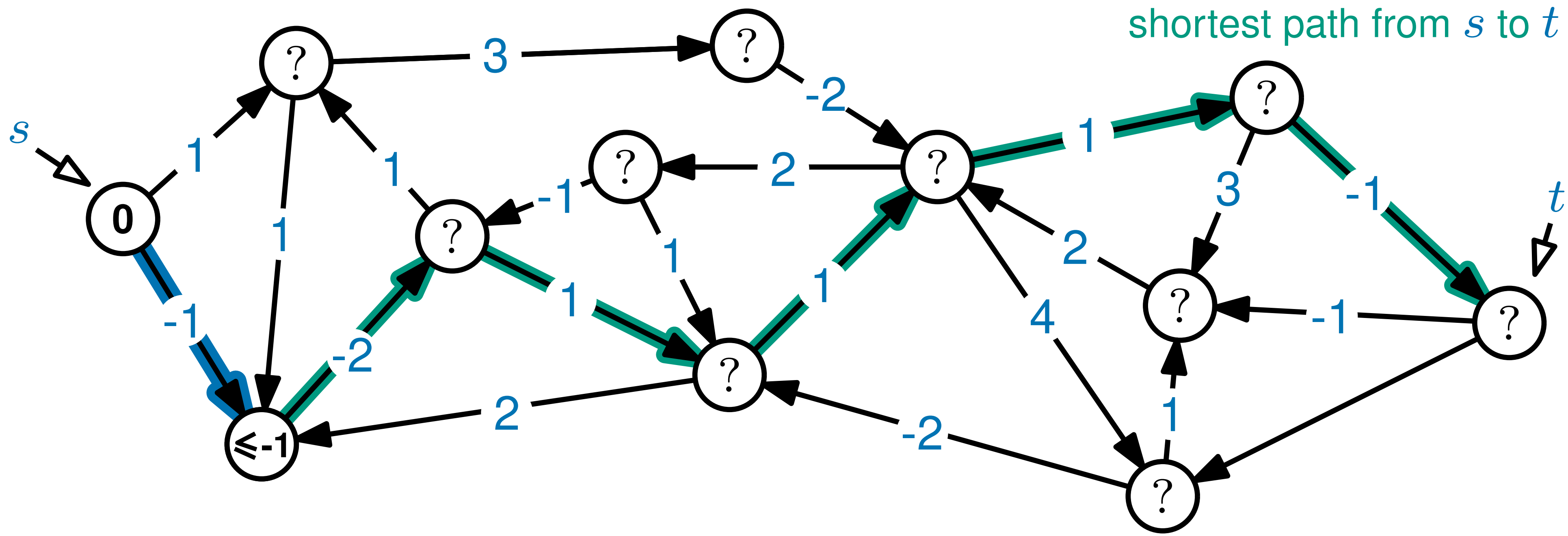
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

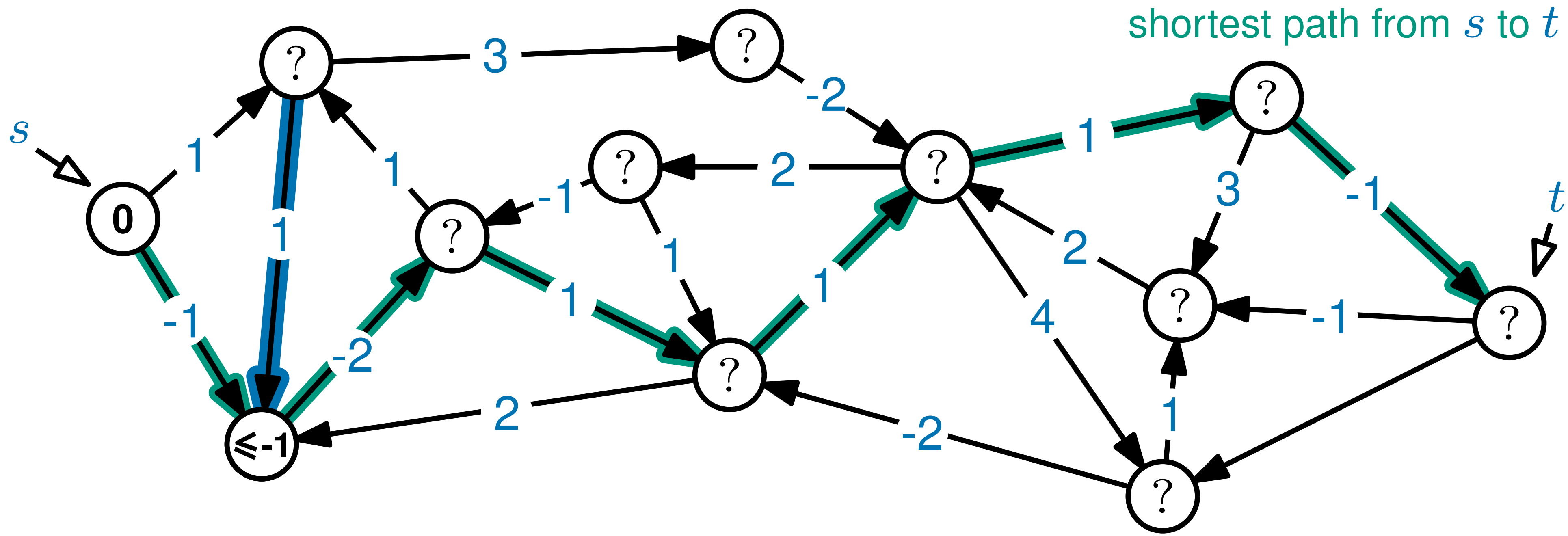
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

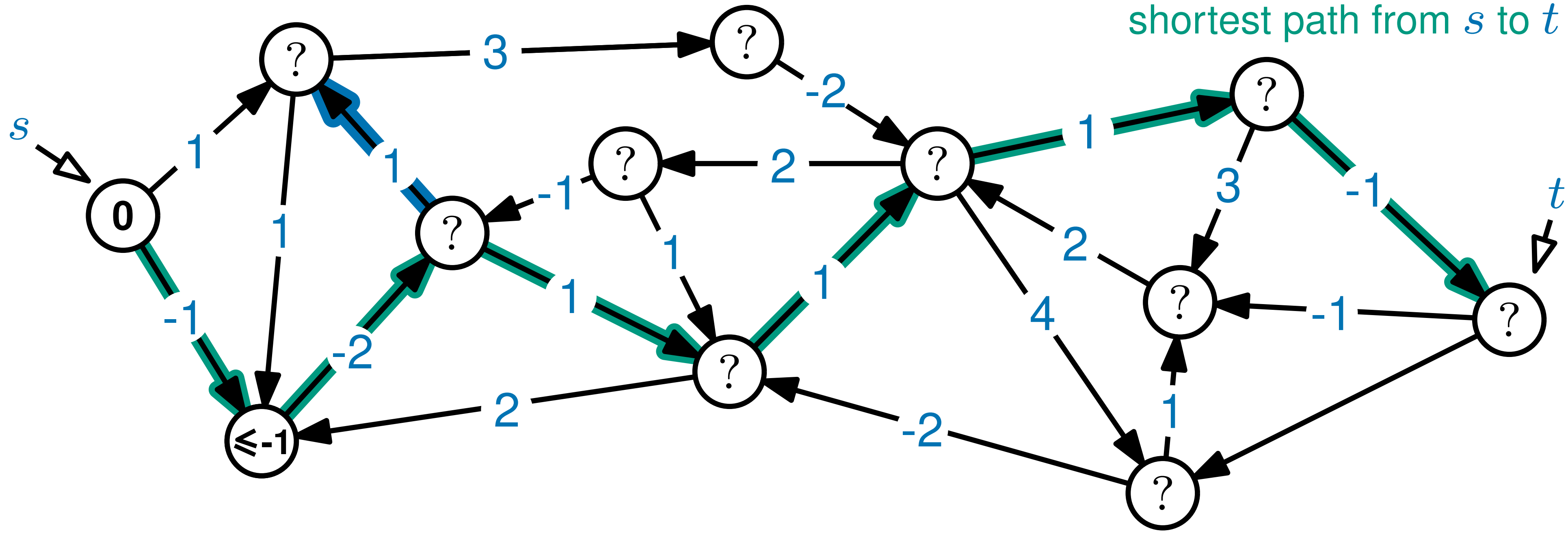
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

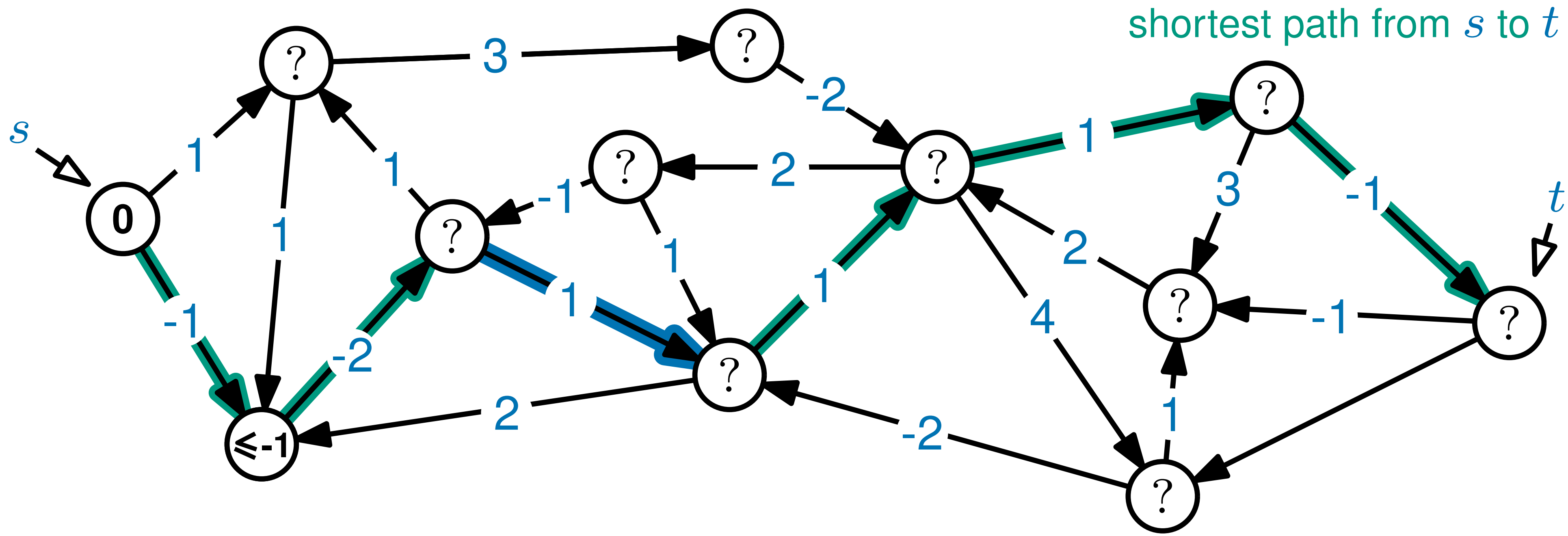
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...
you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i
you relax the i-th edge in the shortest path from s to t...

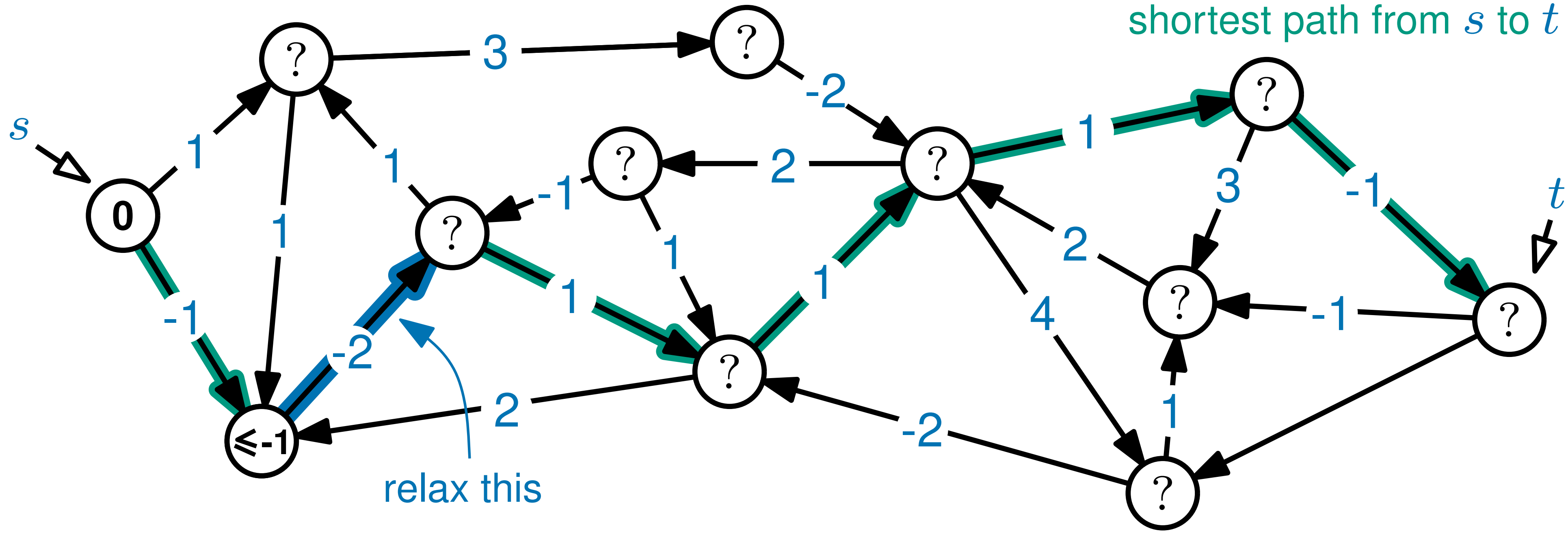
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

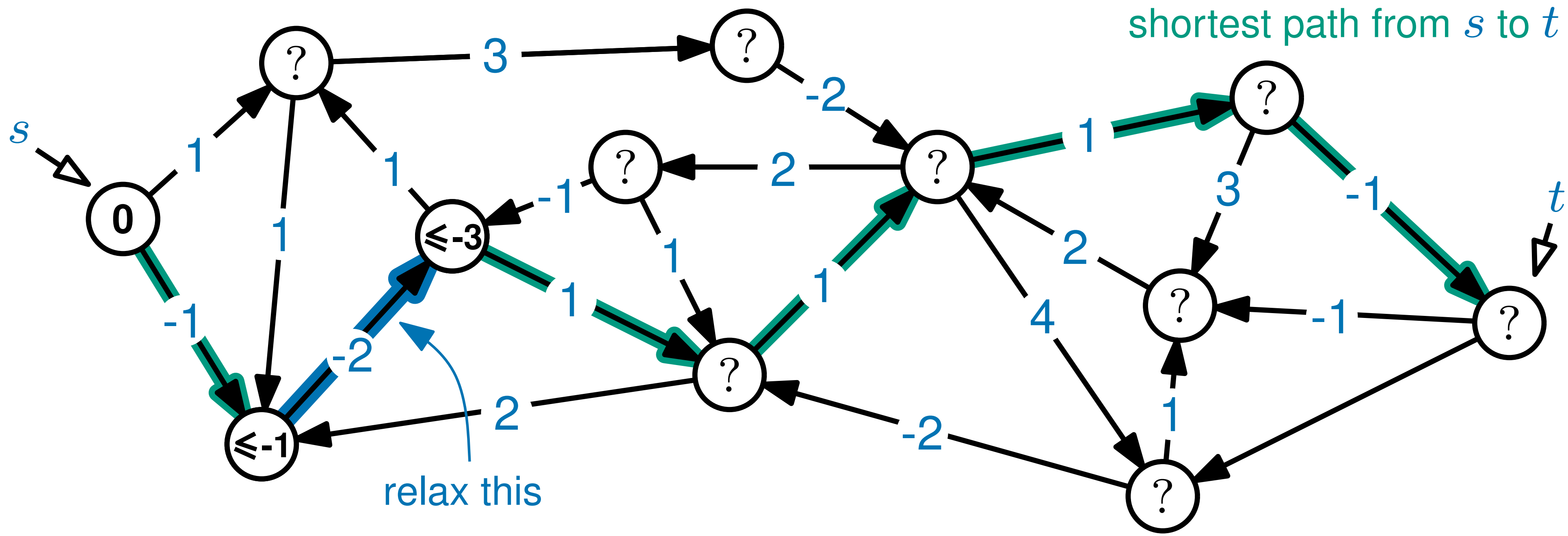
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration…

you relax every edge *(rather than one edge)*

Iteration 2:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t…

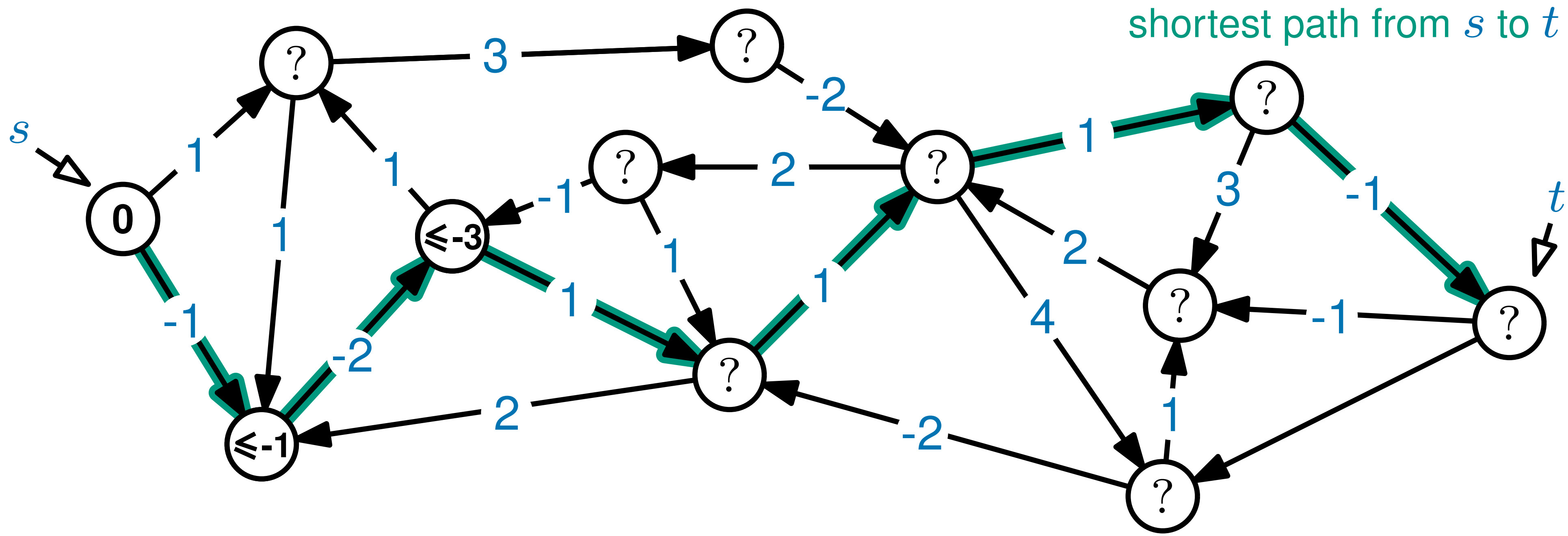
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

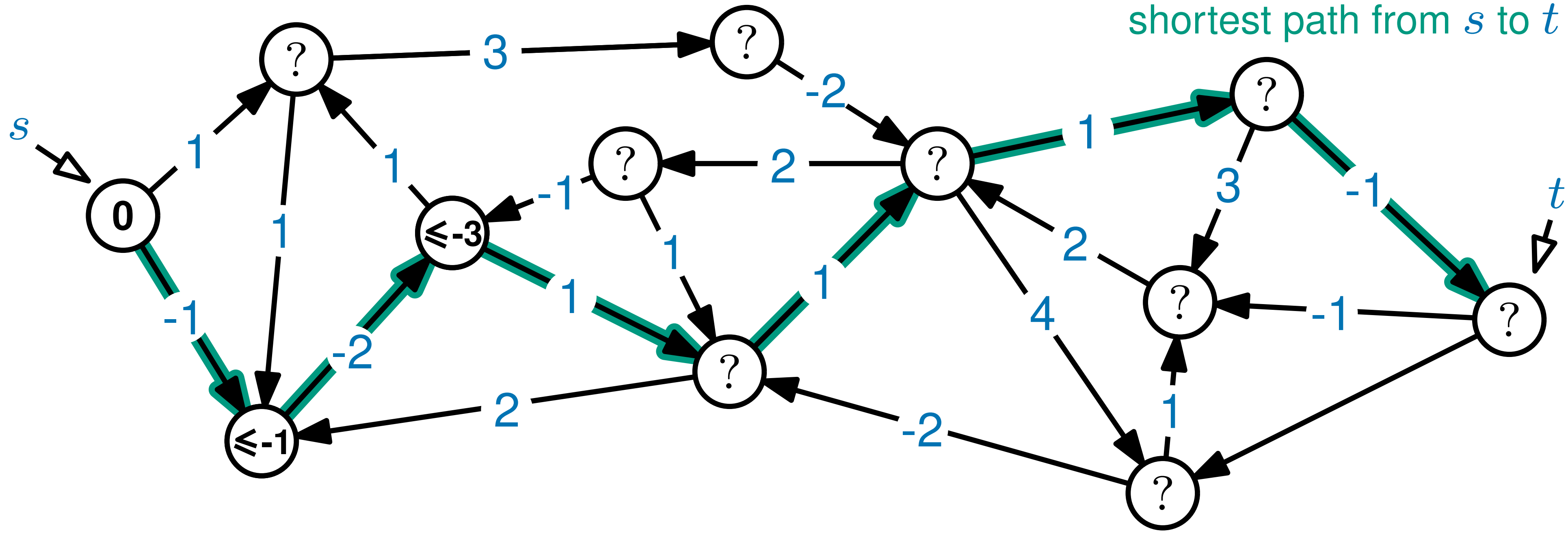
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

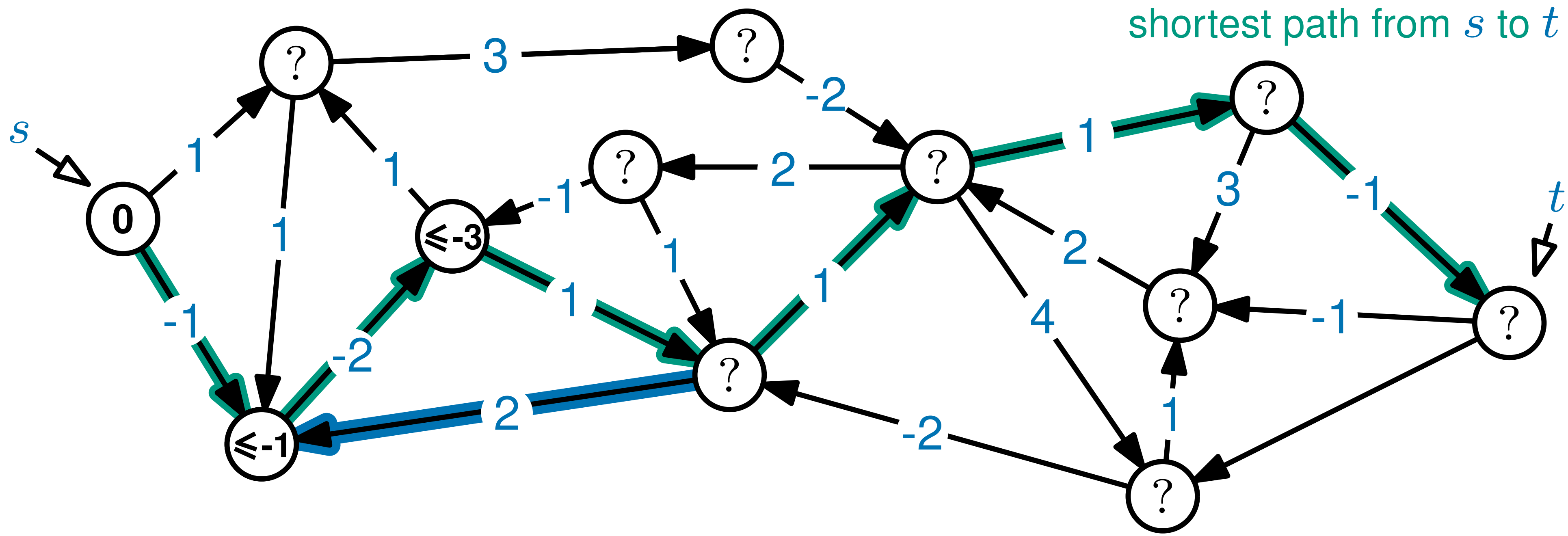
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

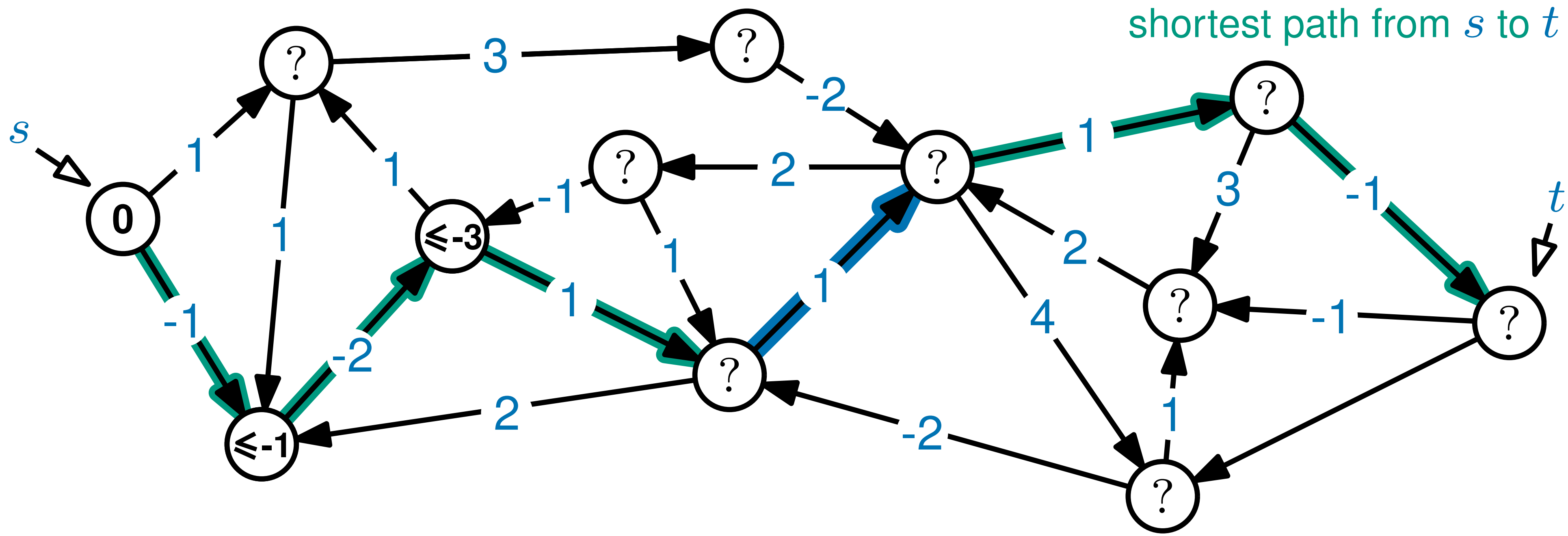
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

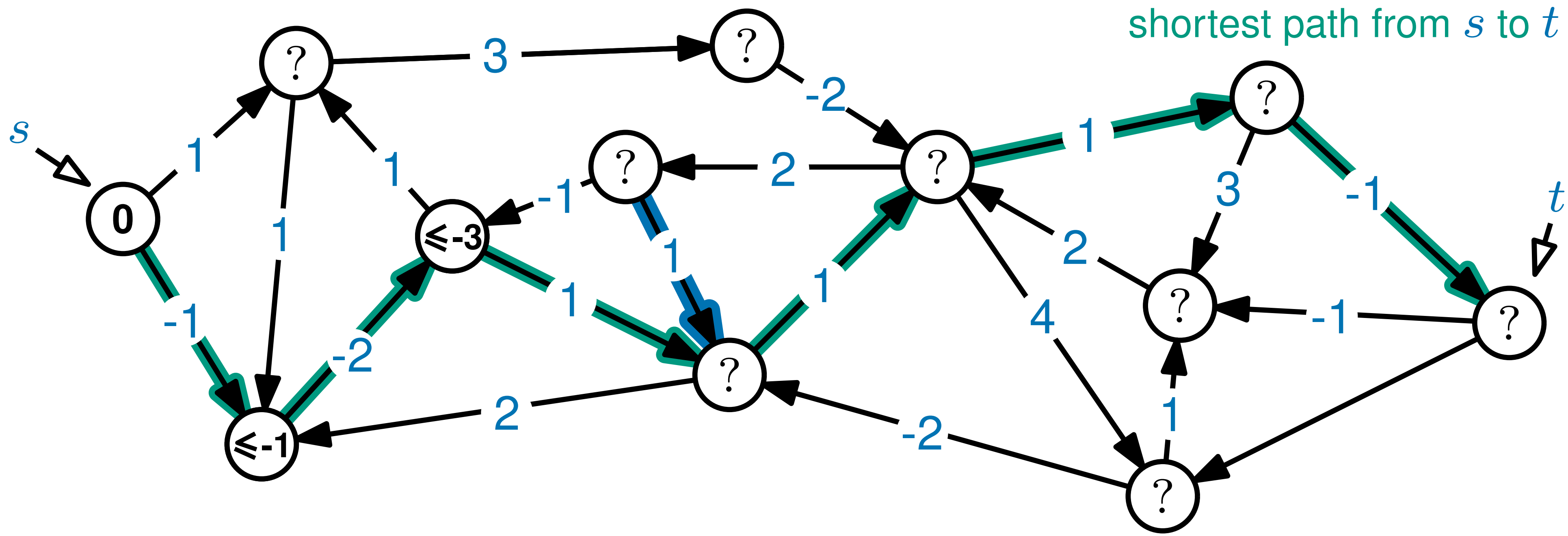
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

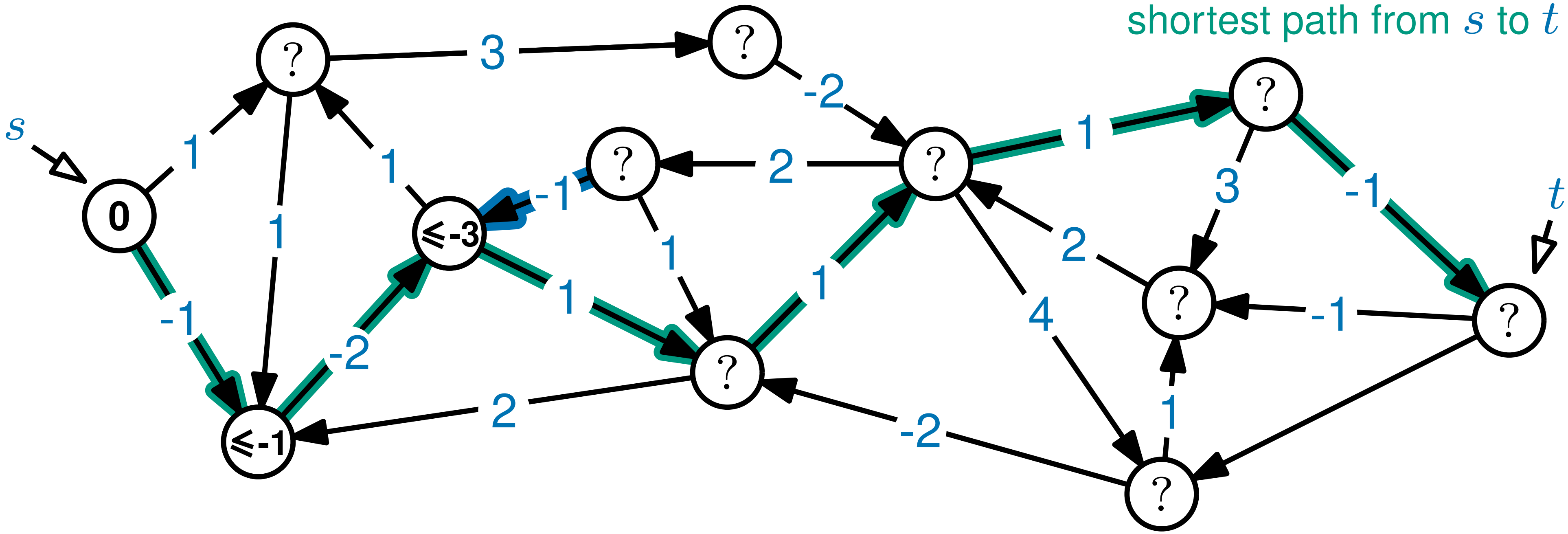
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

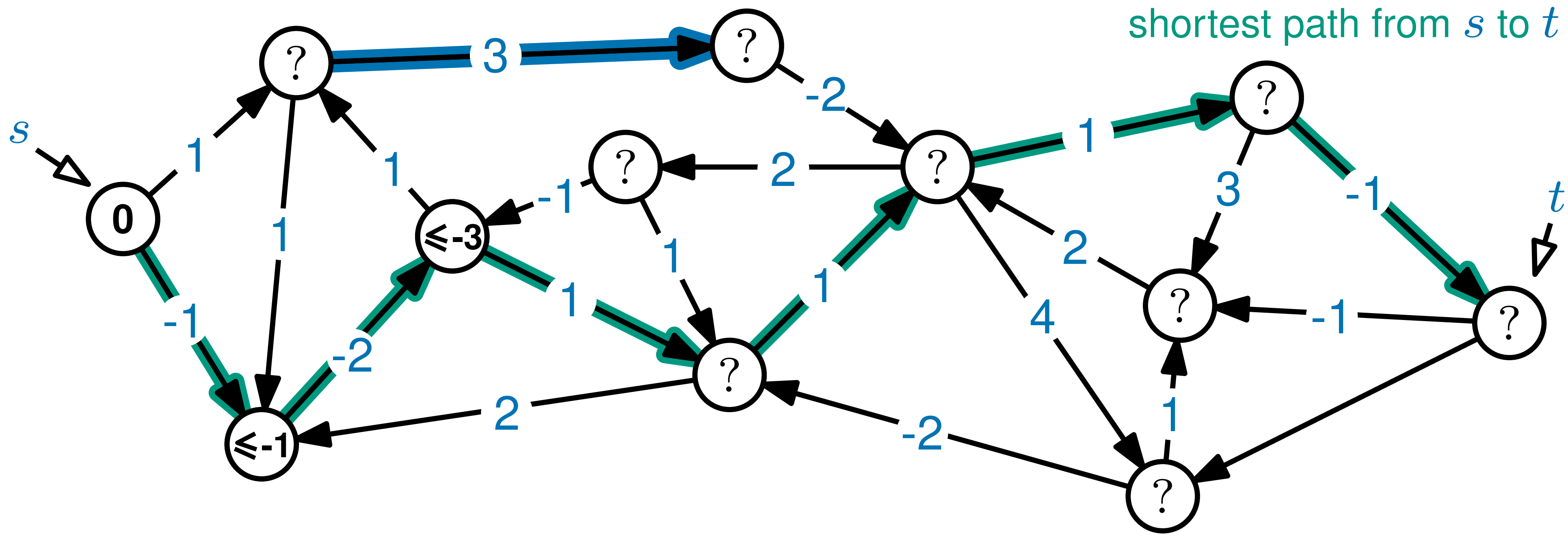
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration…

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t…

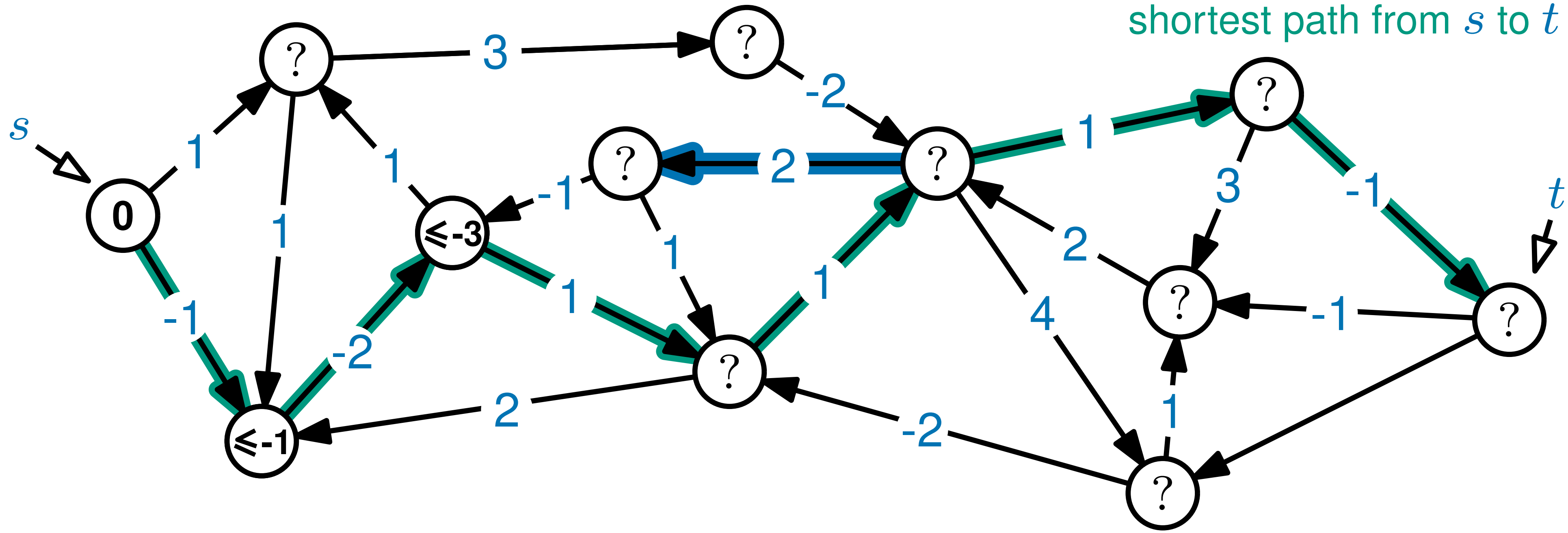
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

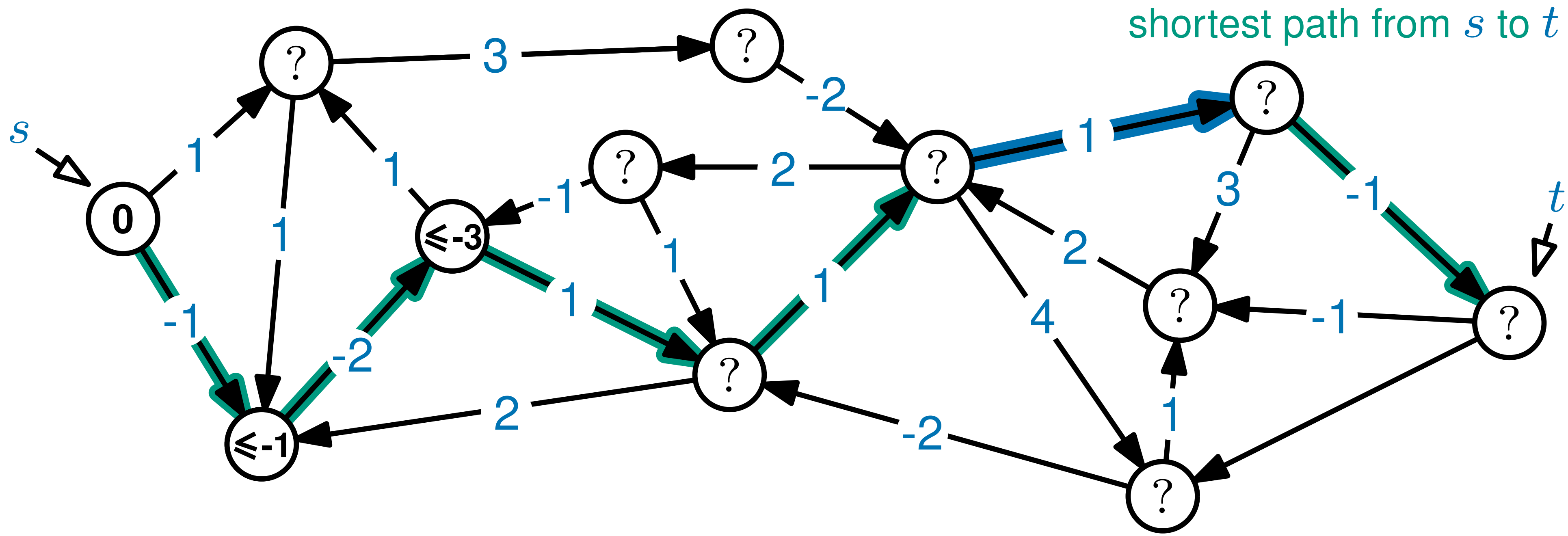
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

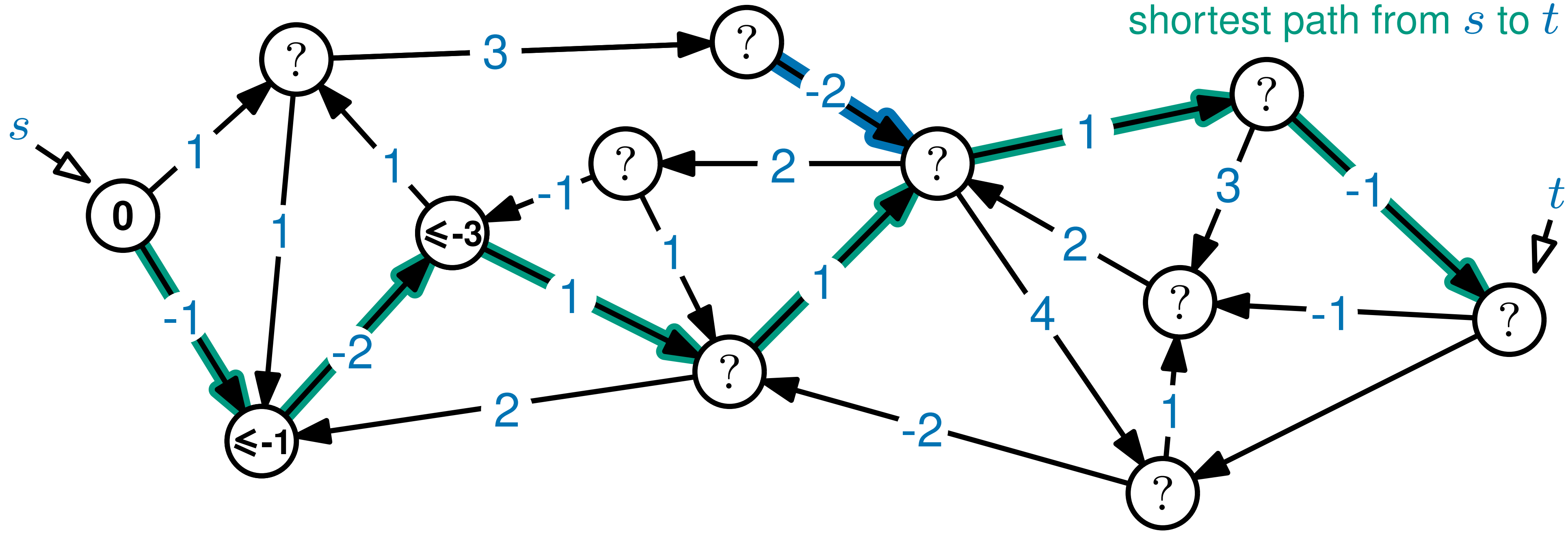
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

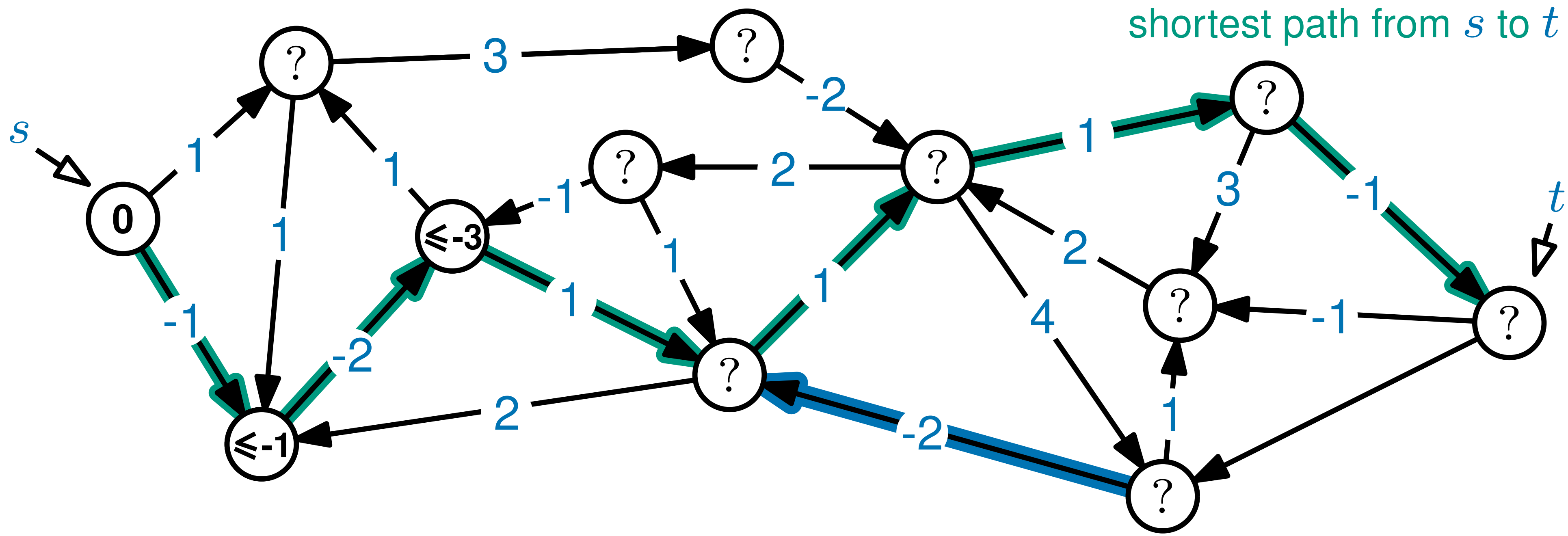
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

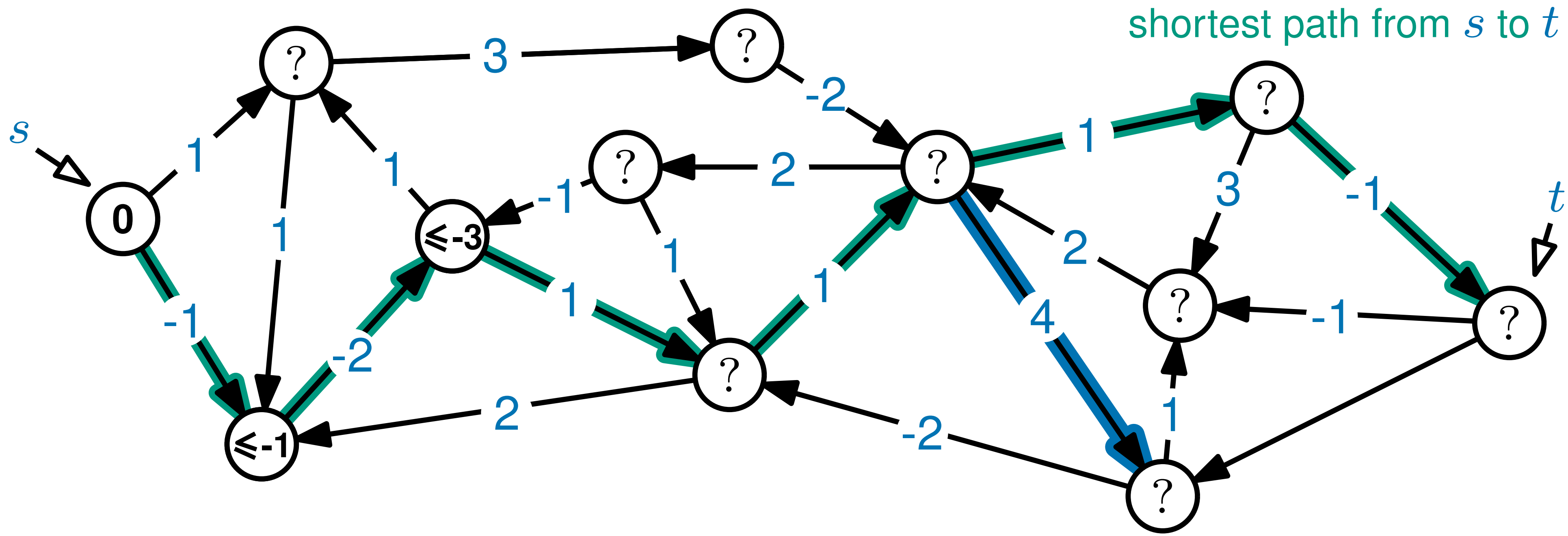
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

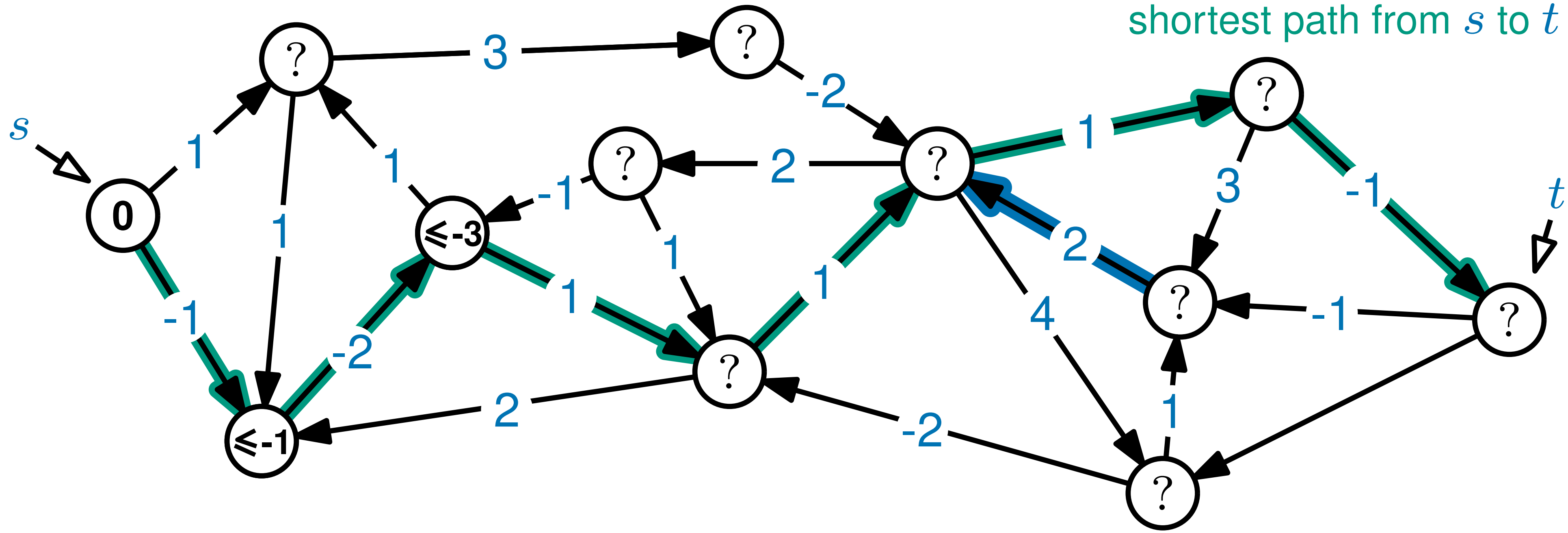
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

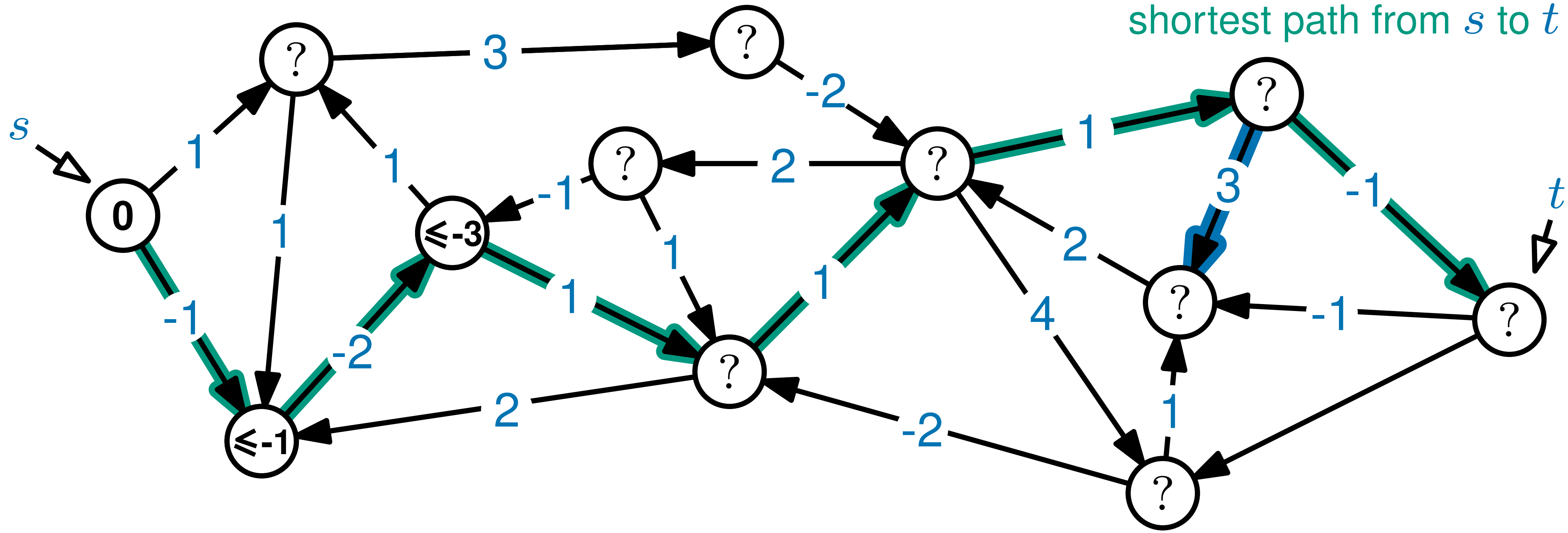
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

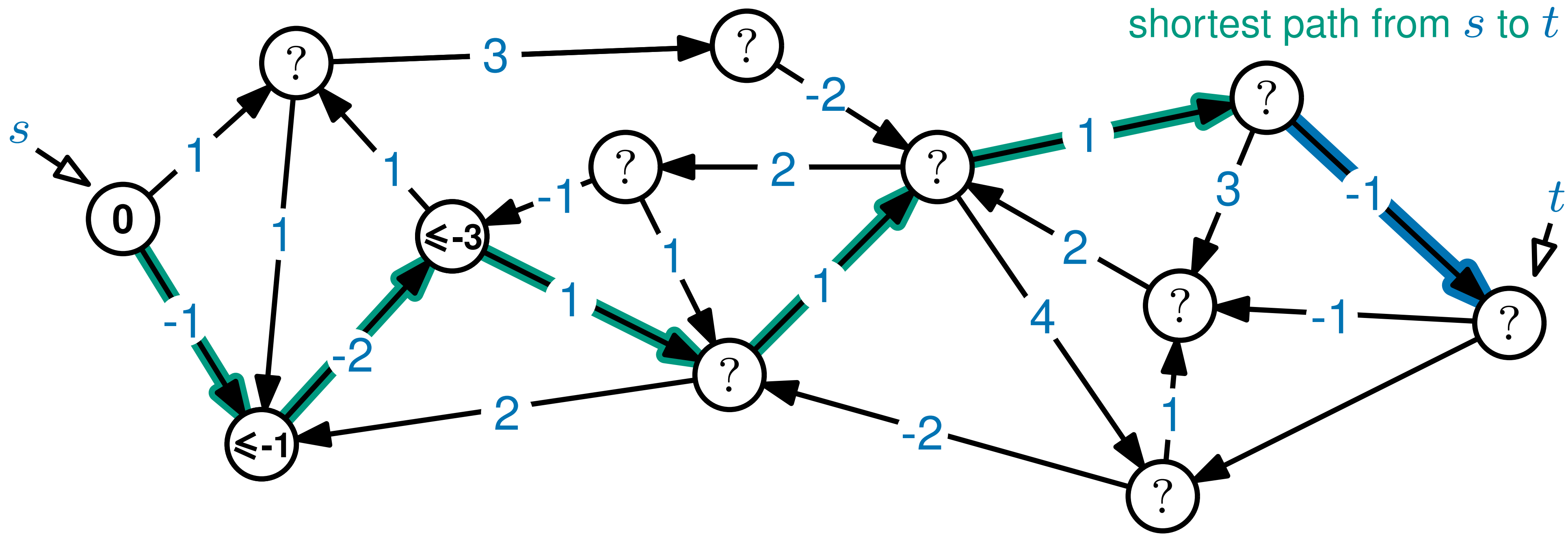
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

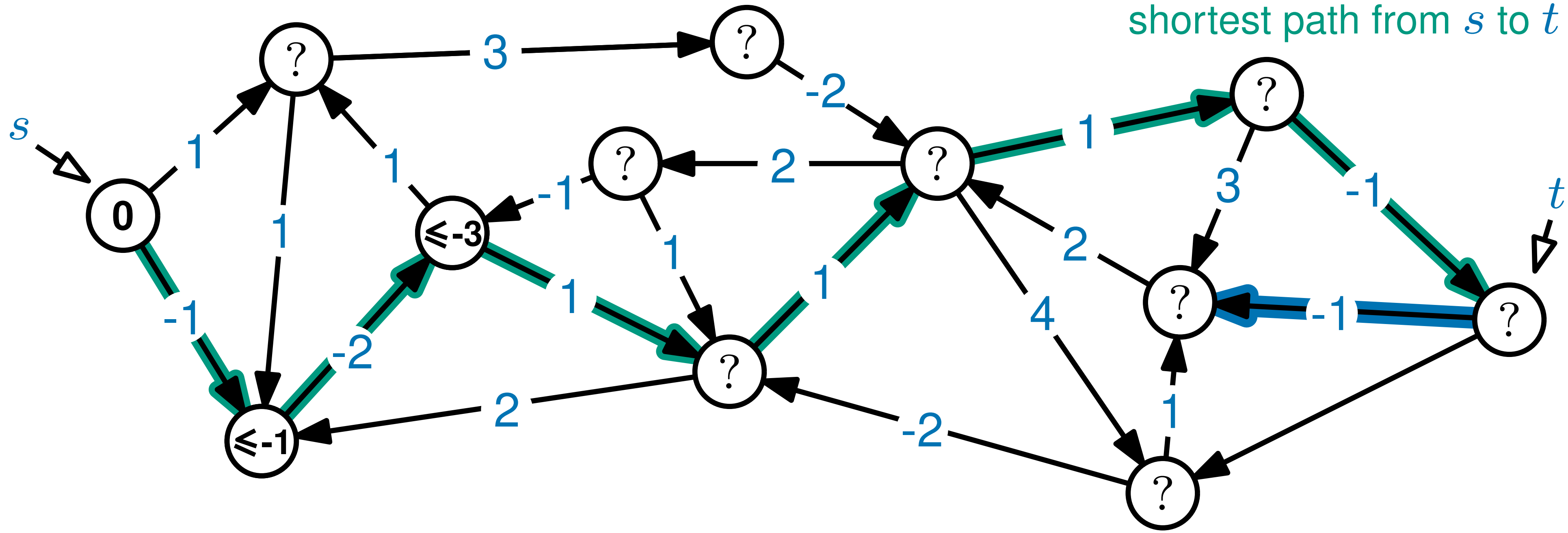
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

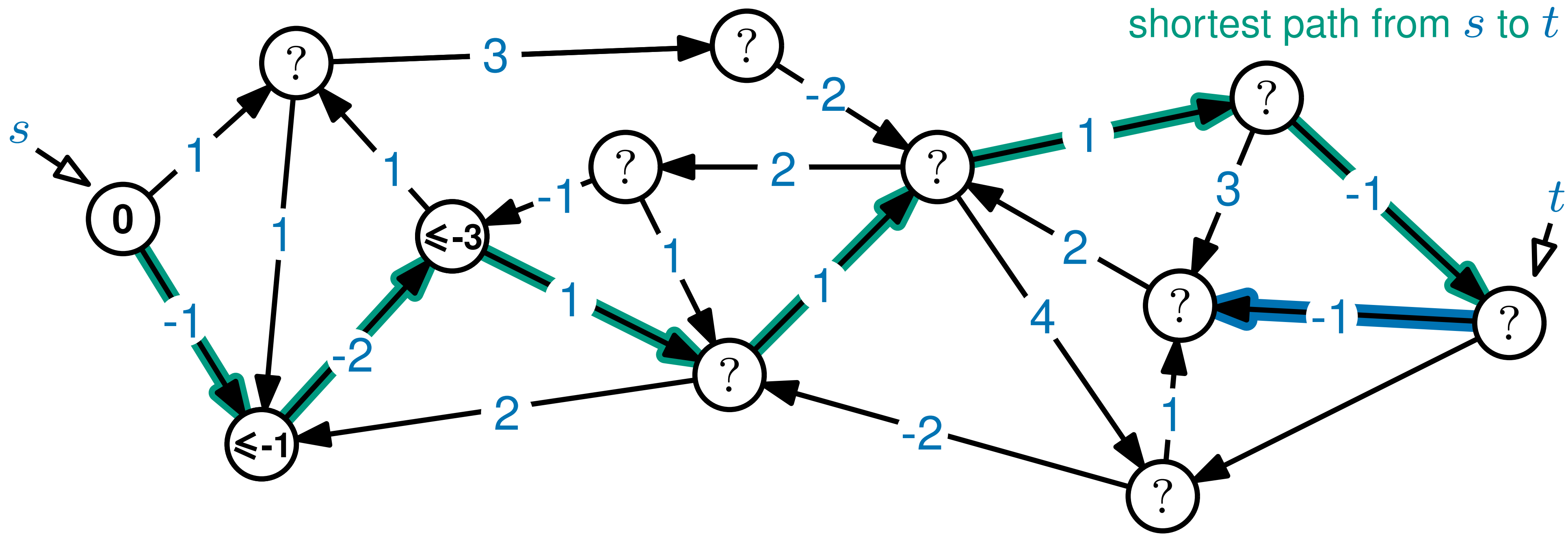
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

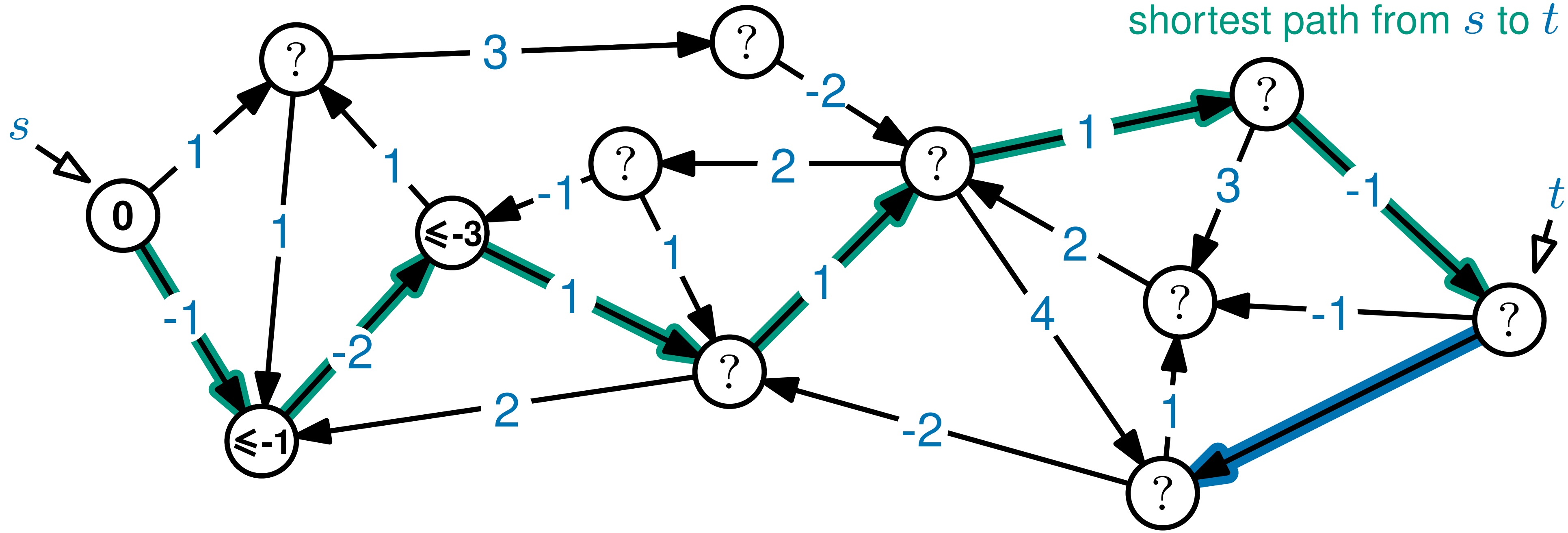
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

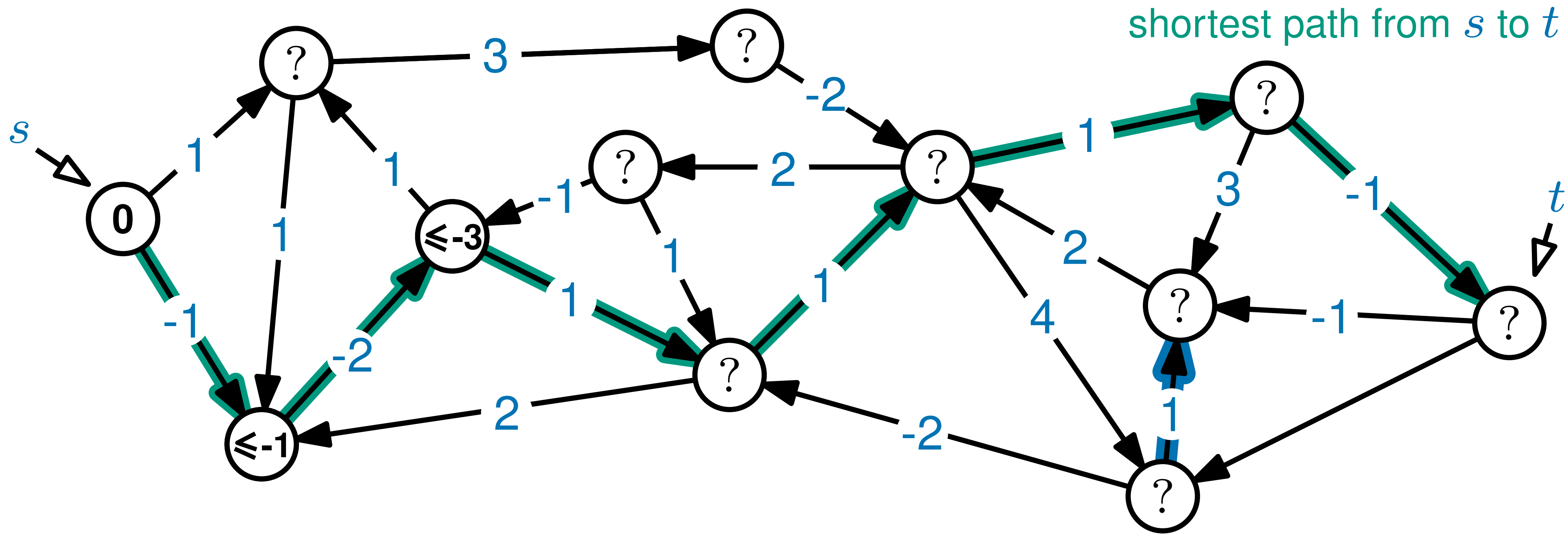
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

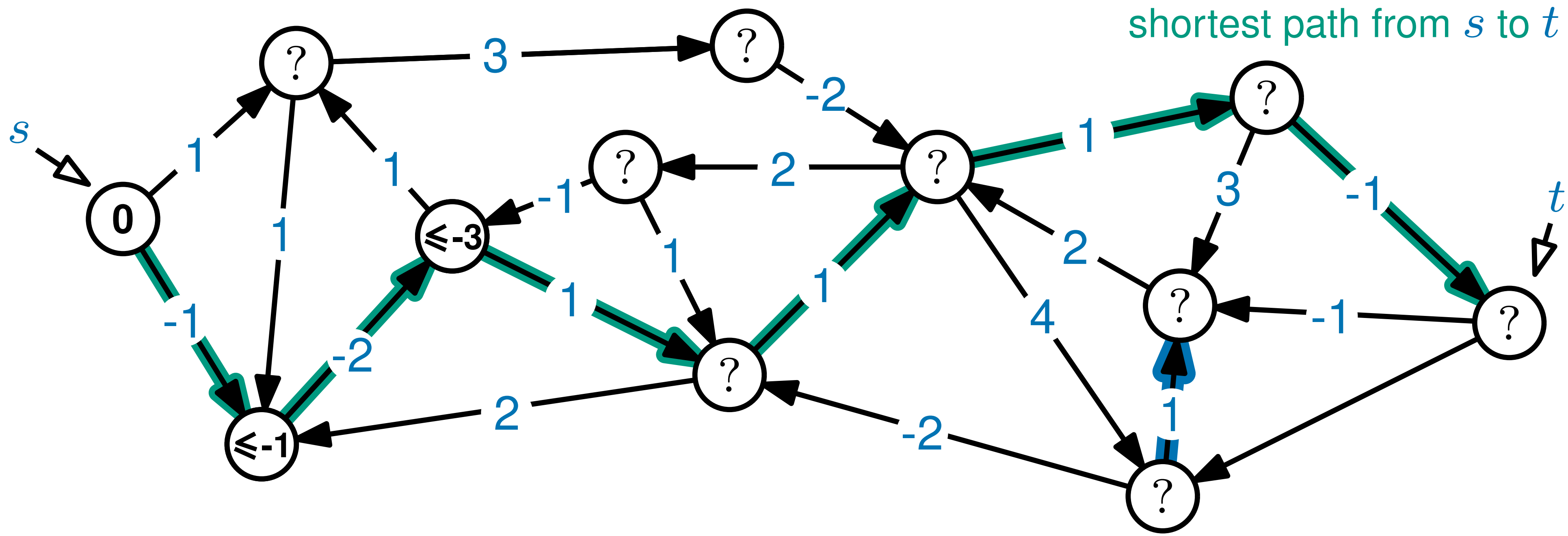
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 2:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

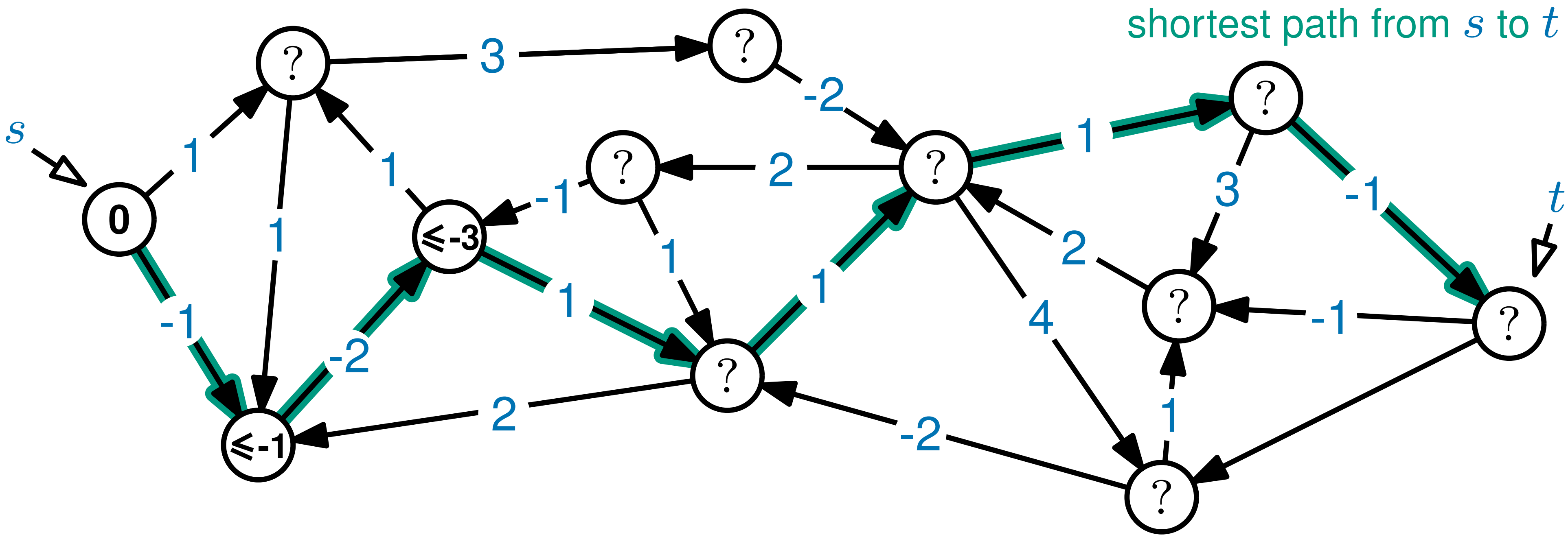
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

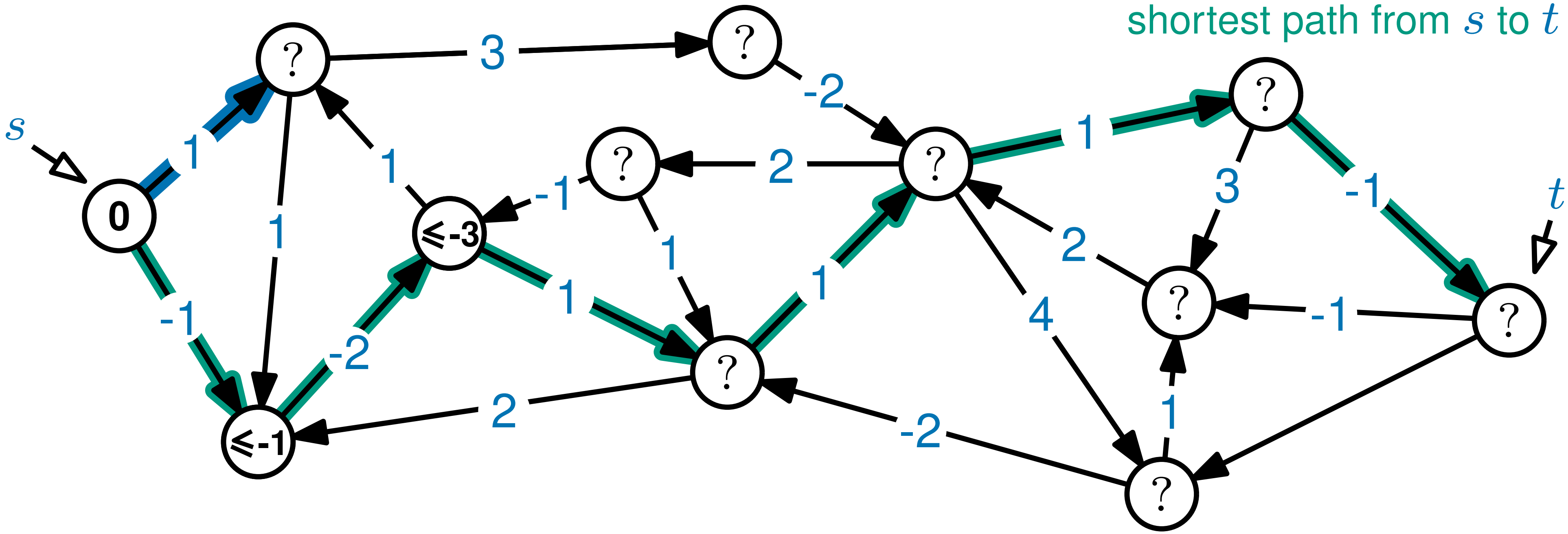
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

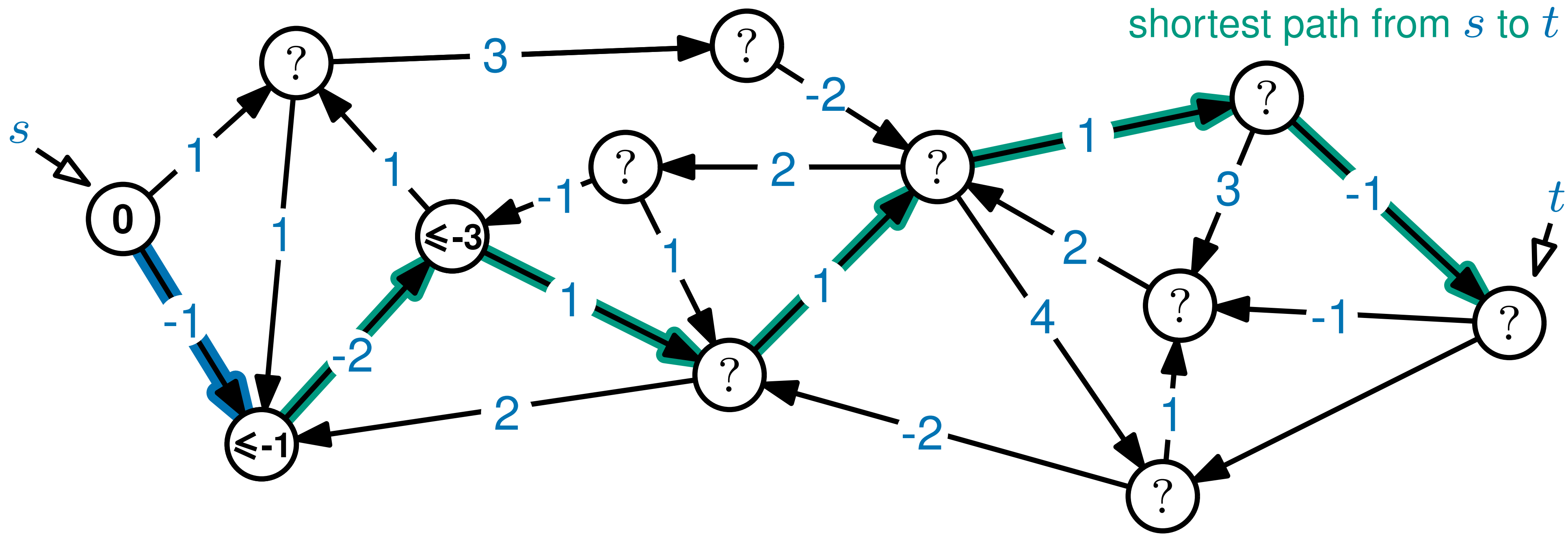
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

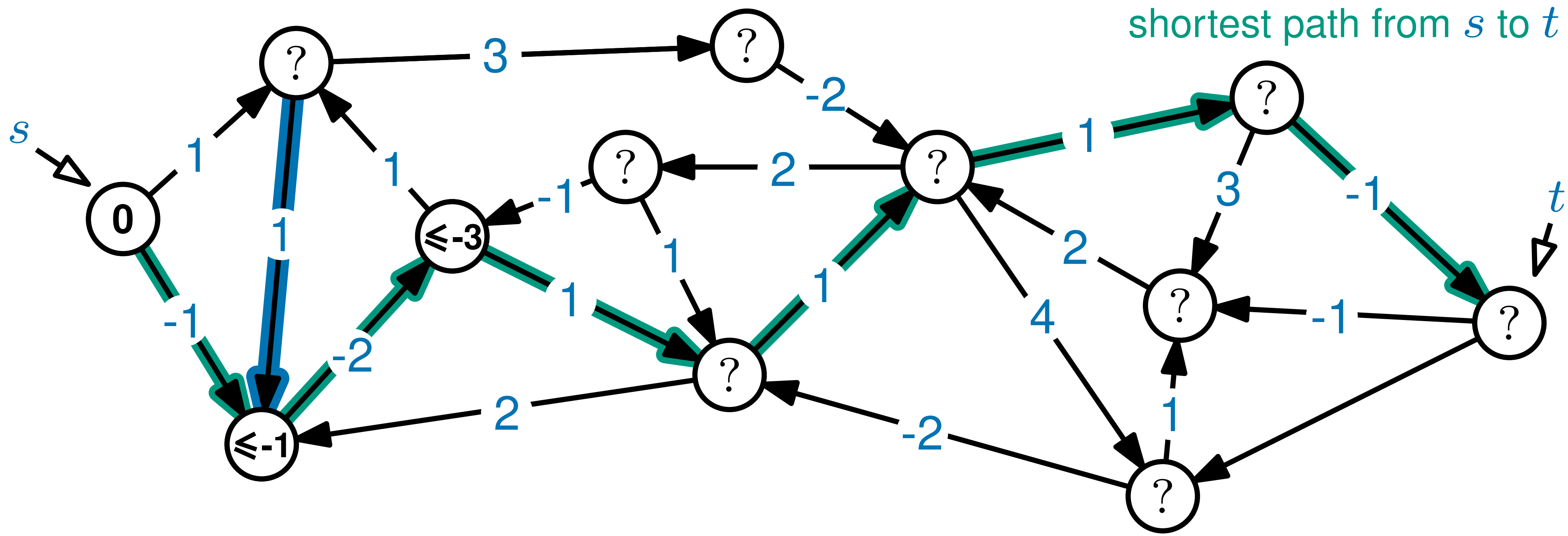
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

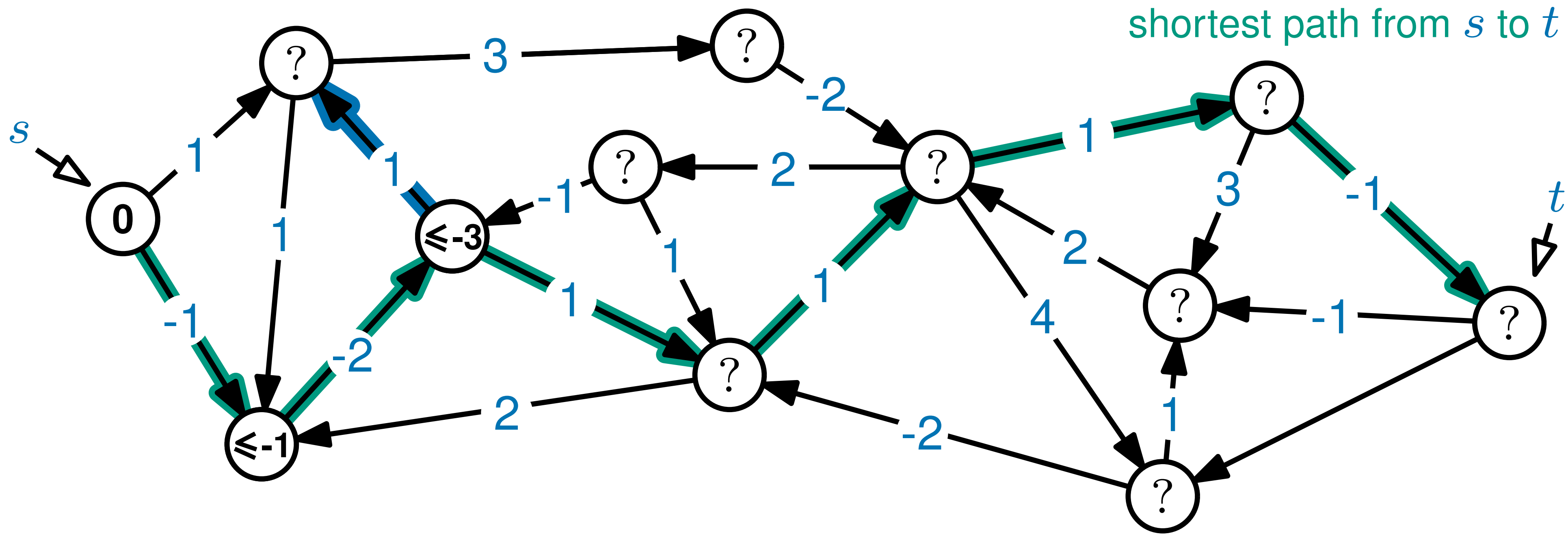
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

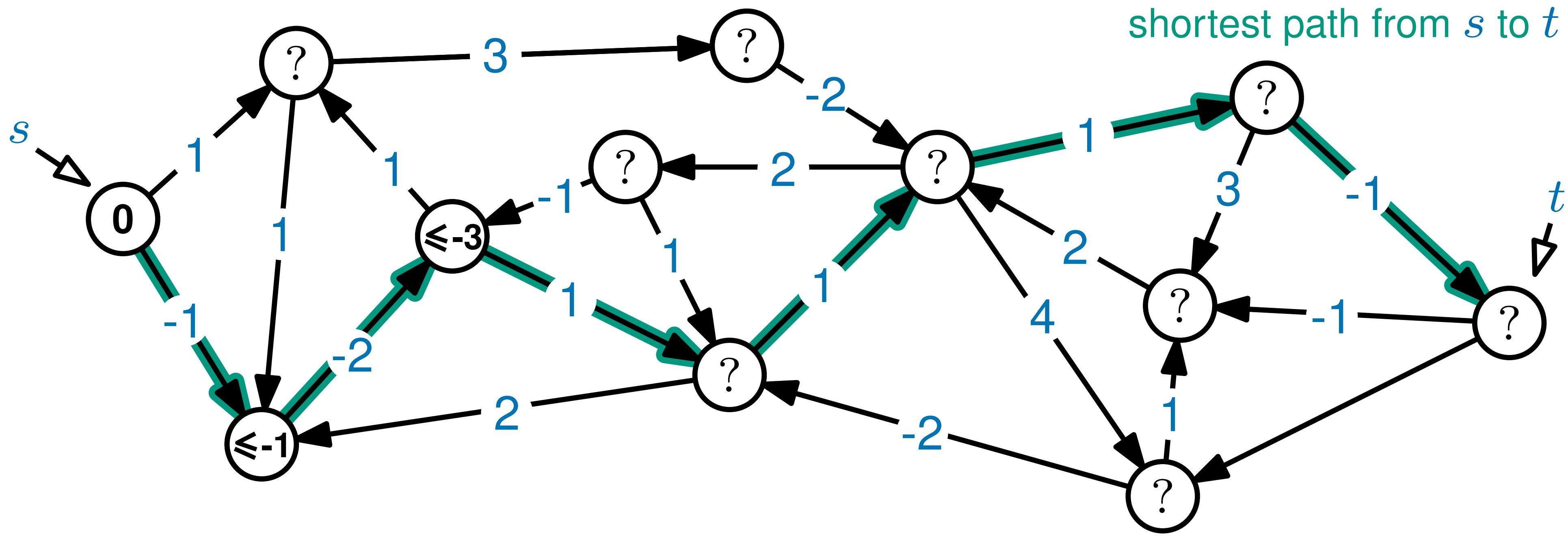
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



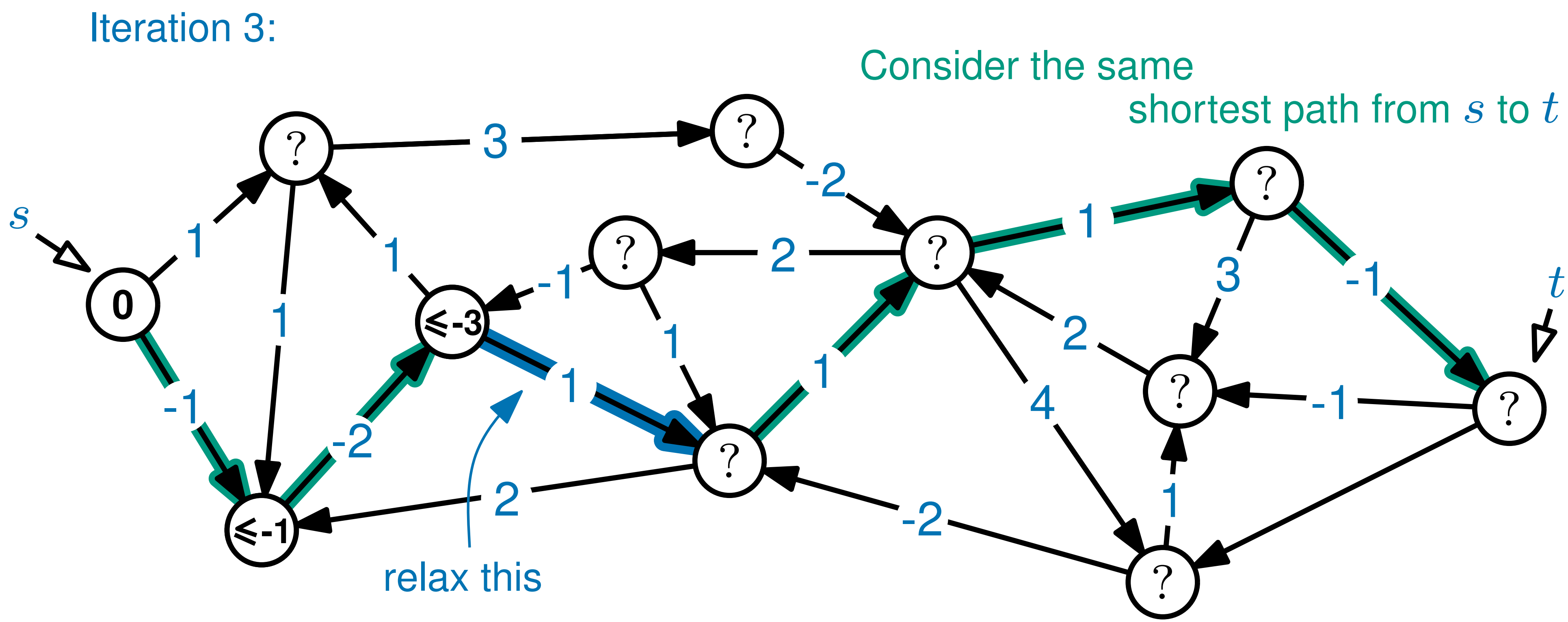
At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*



At some point in iteration i

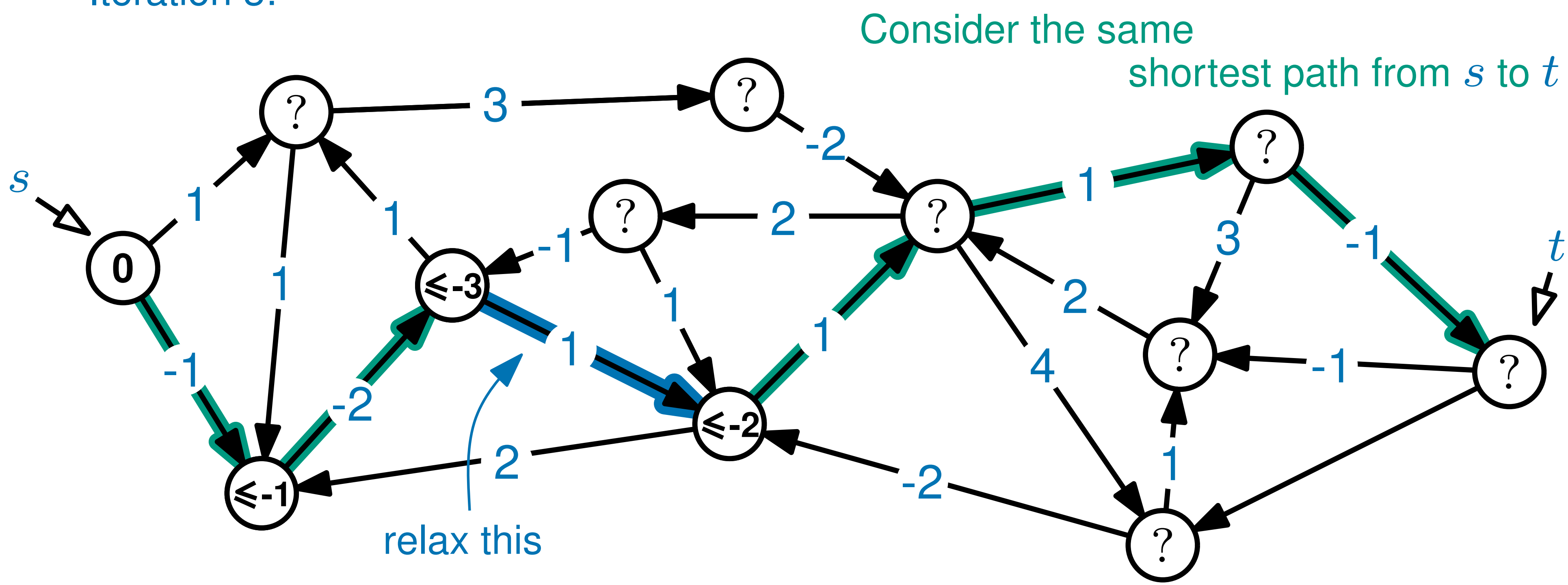
you relax the i-th edge in the shortest path from s to t. . .

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:



At some point in iteration i

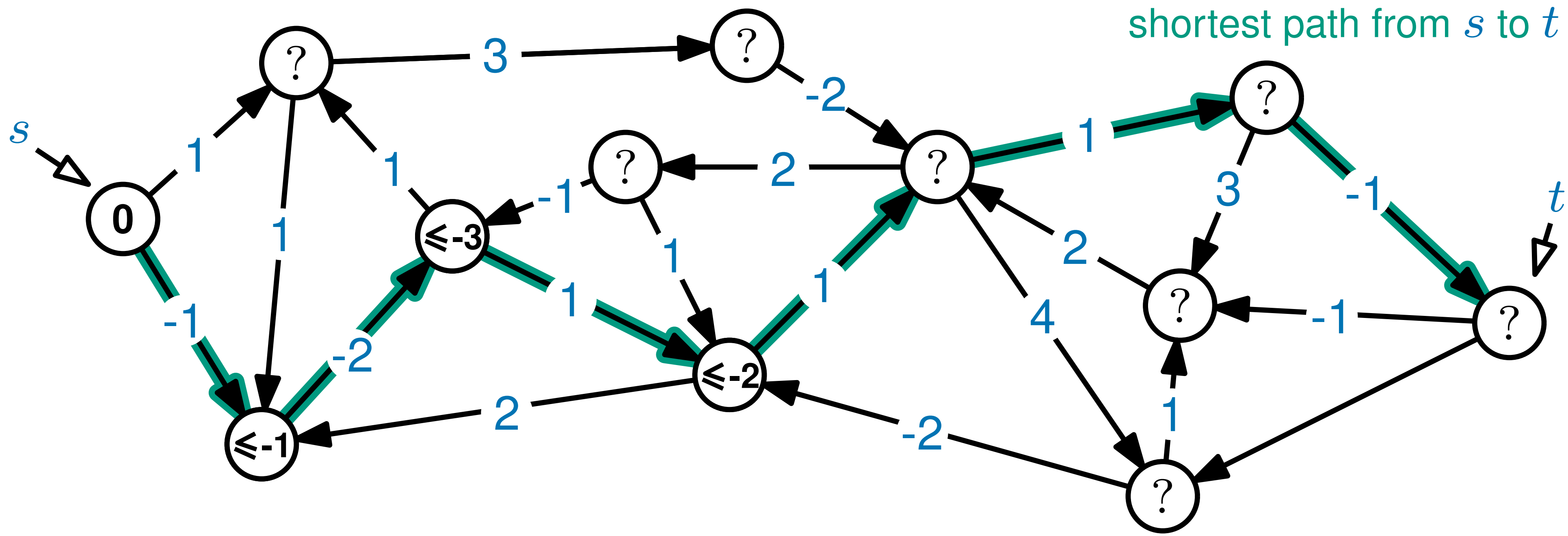
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

The proof idea

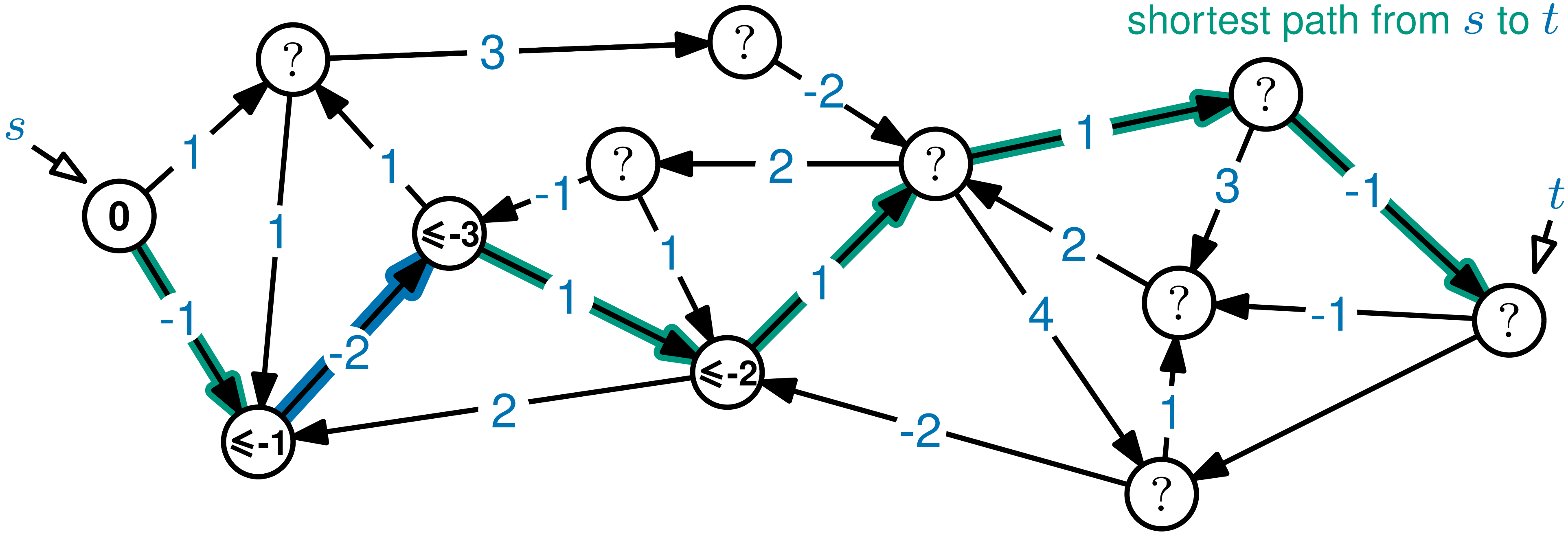
Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same

shortest path from s to t



At some point in iteration i

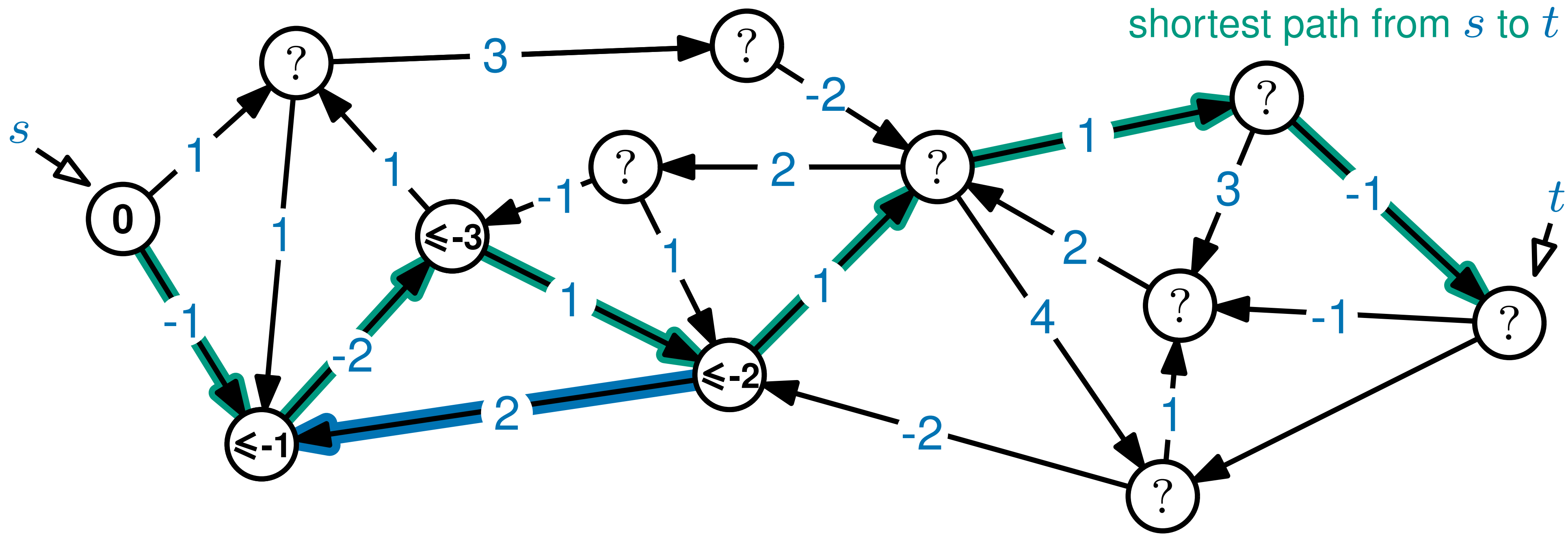
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

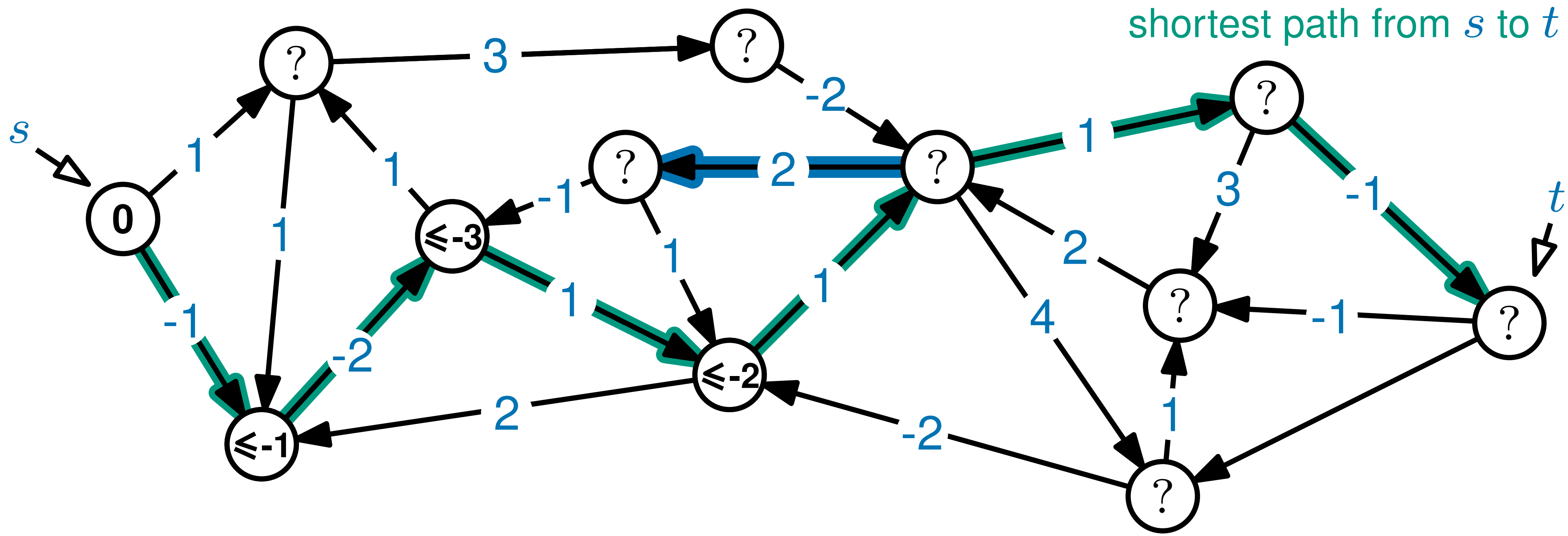
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

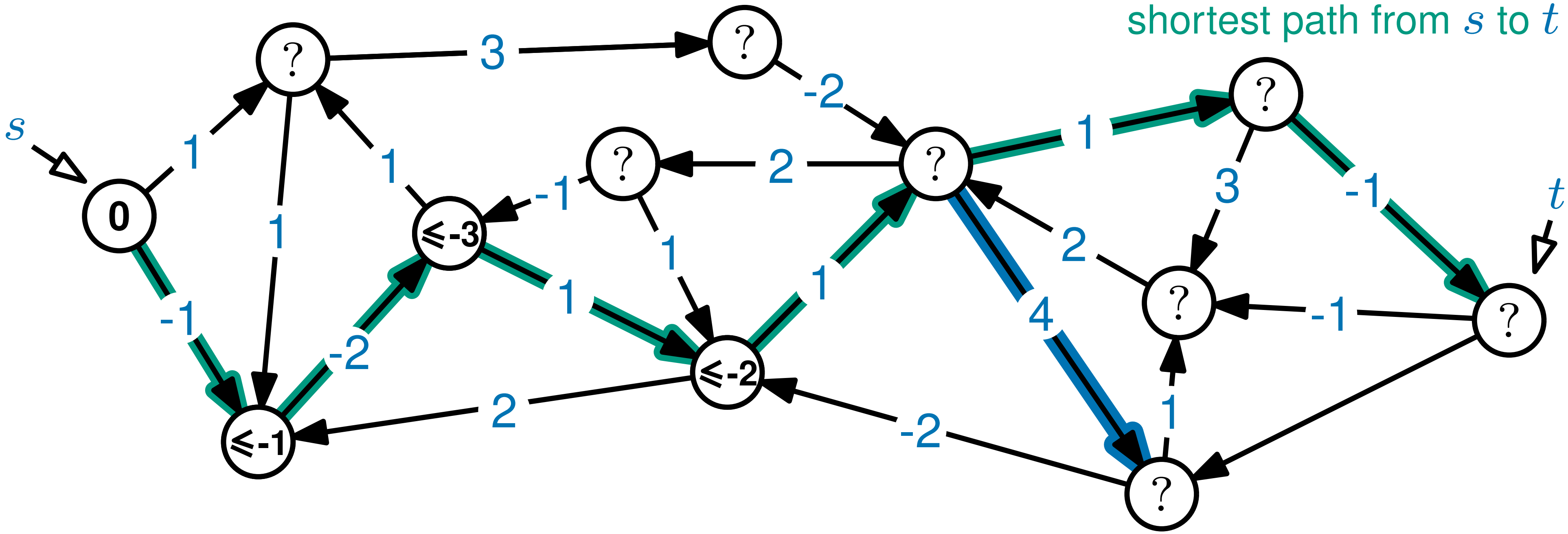
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

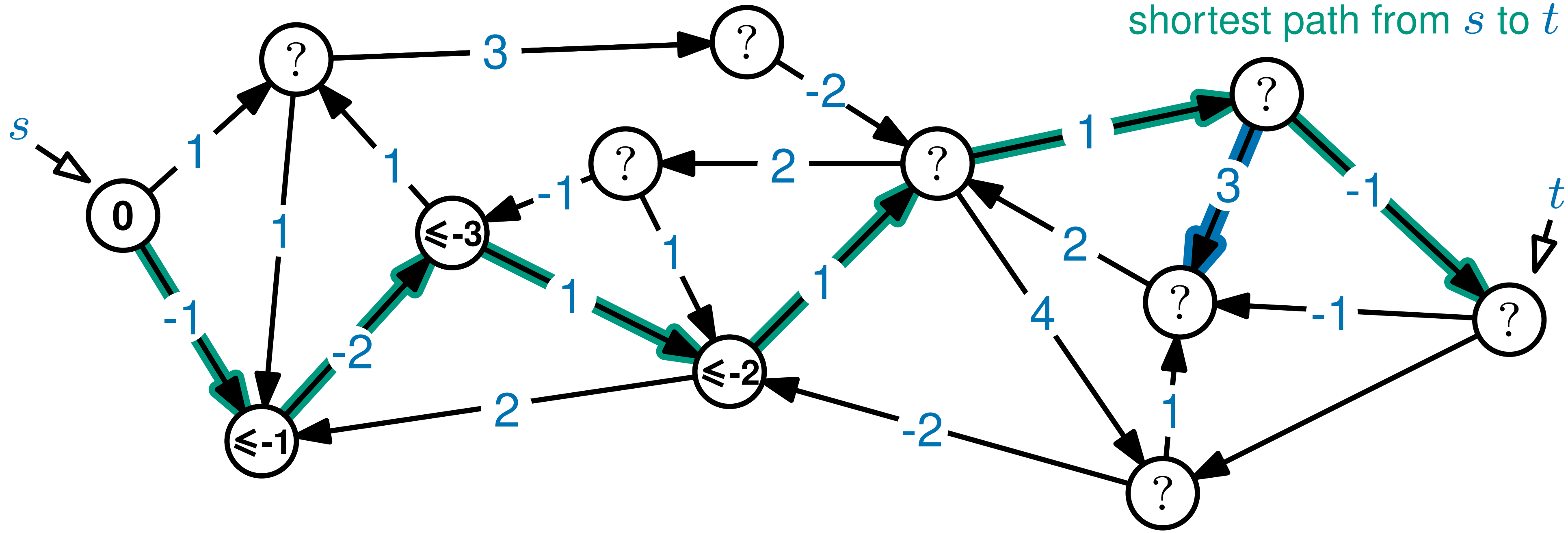
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 3:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

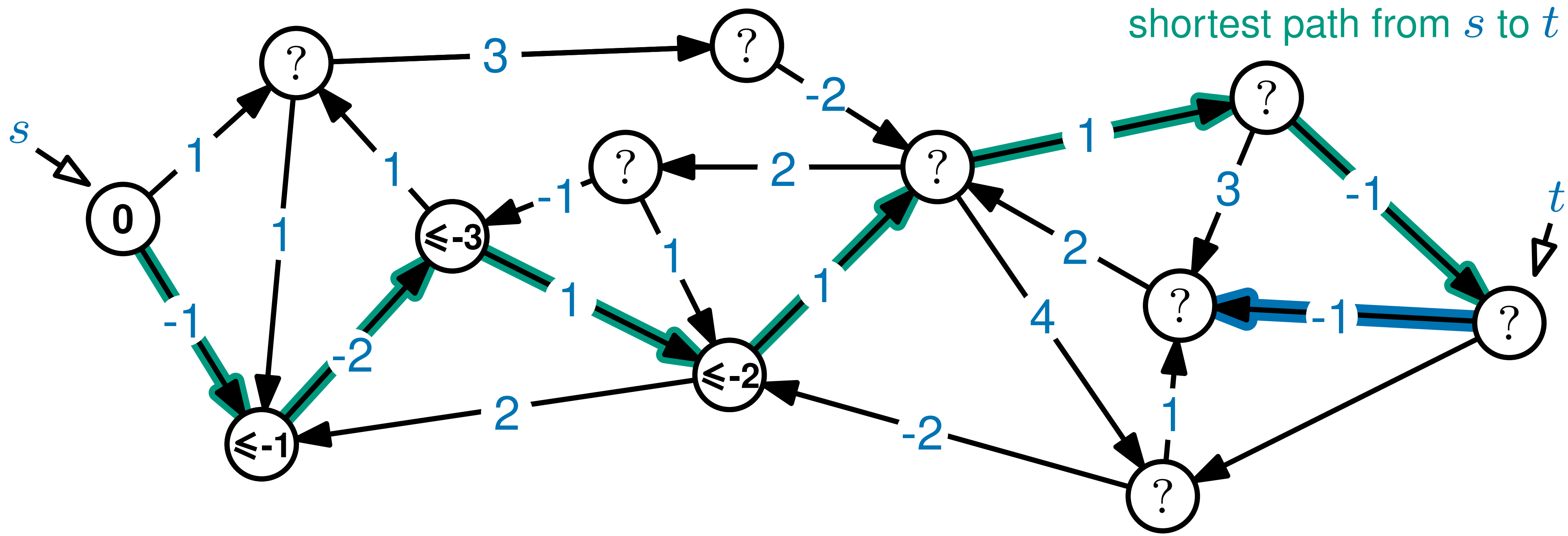
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

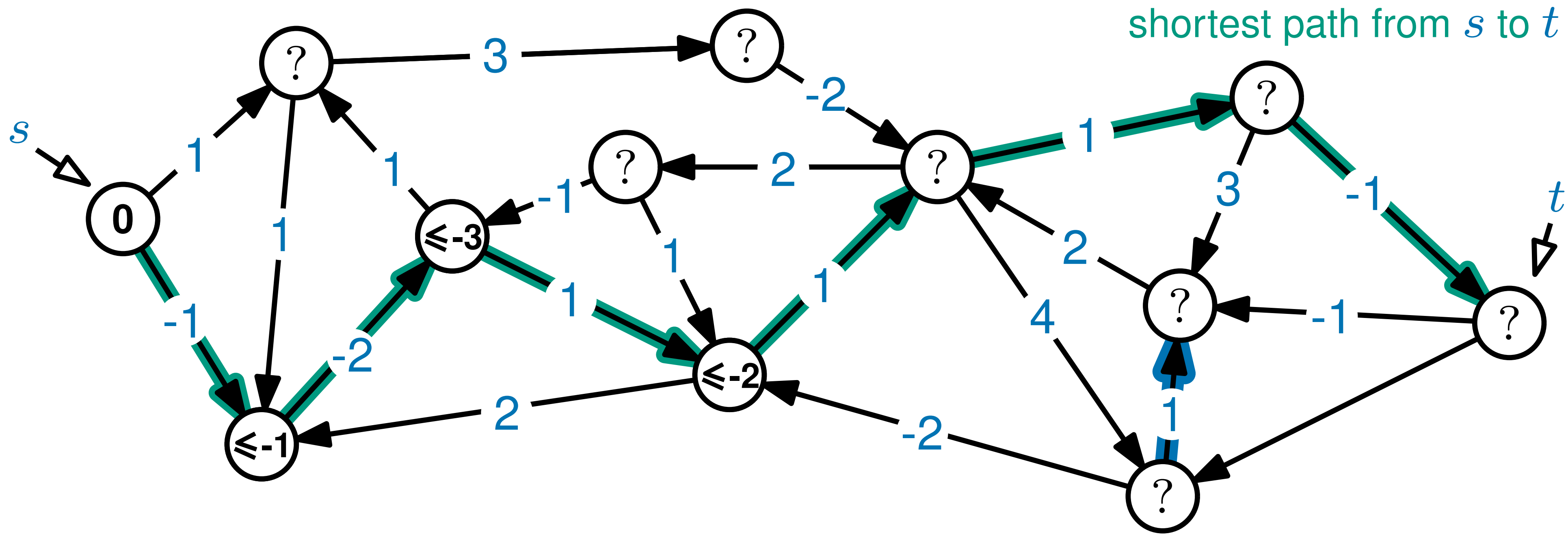
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:

Consider the same shortest path from s to t



At some point in iteration i

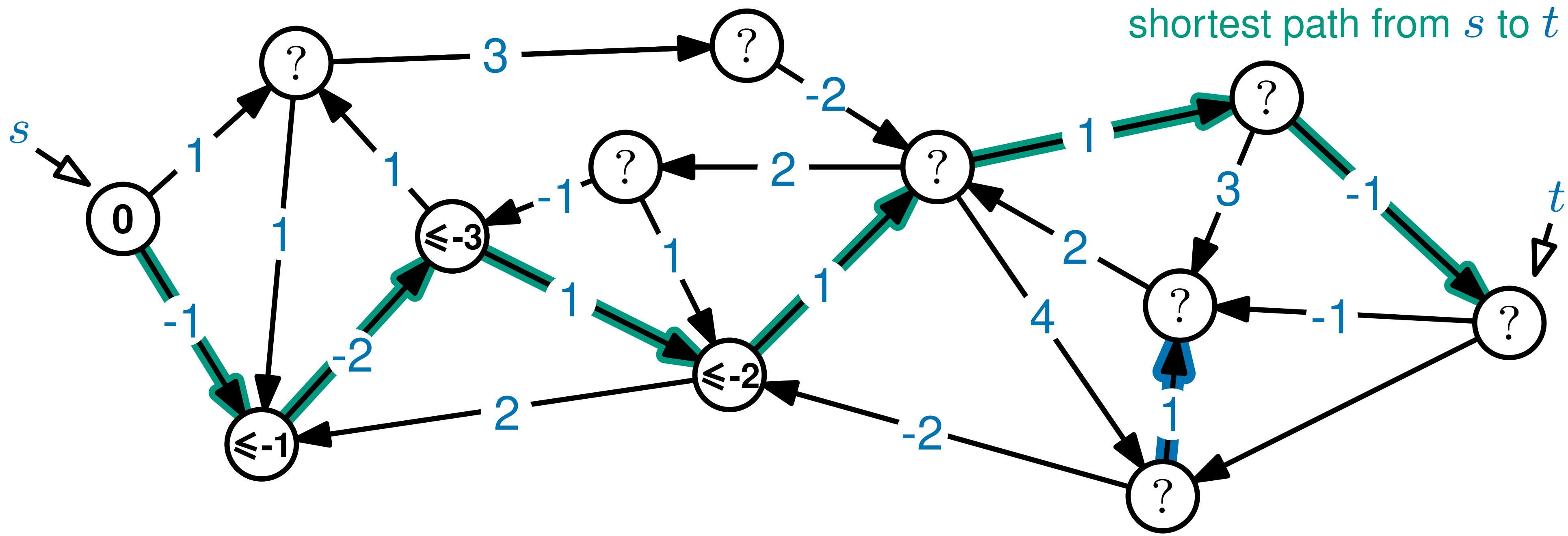
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 3:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

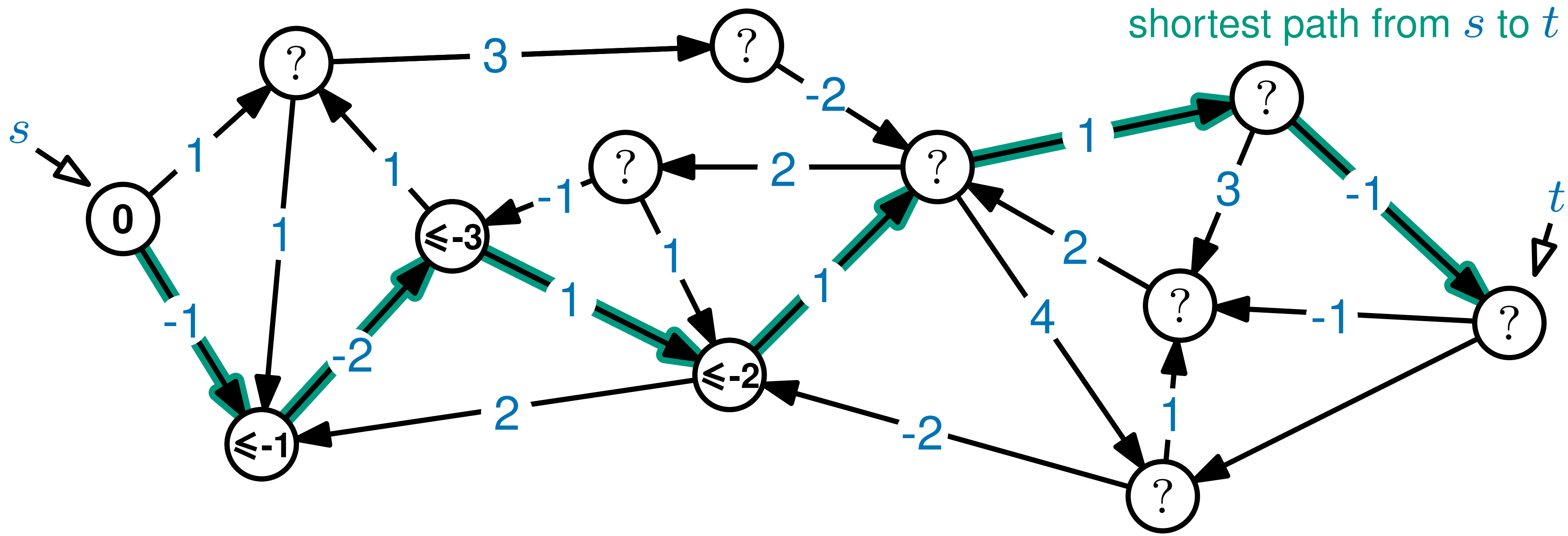
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

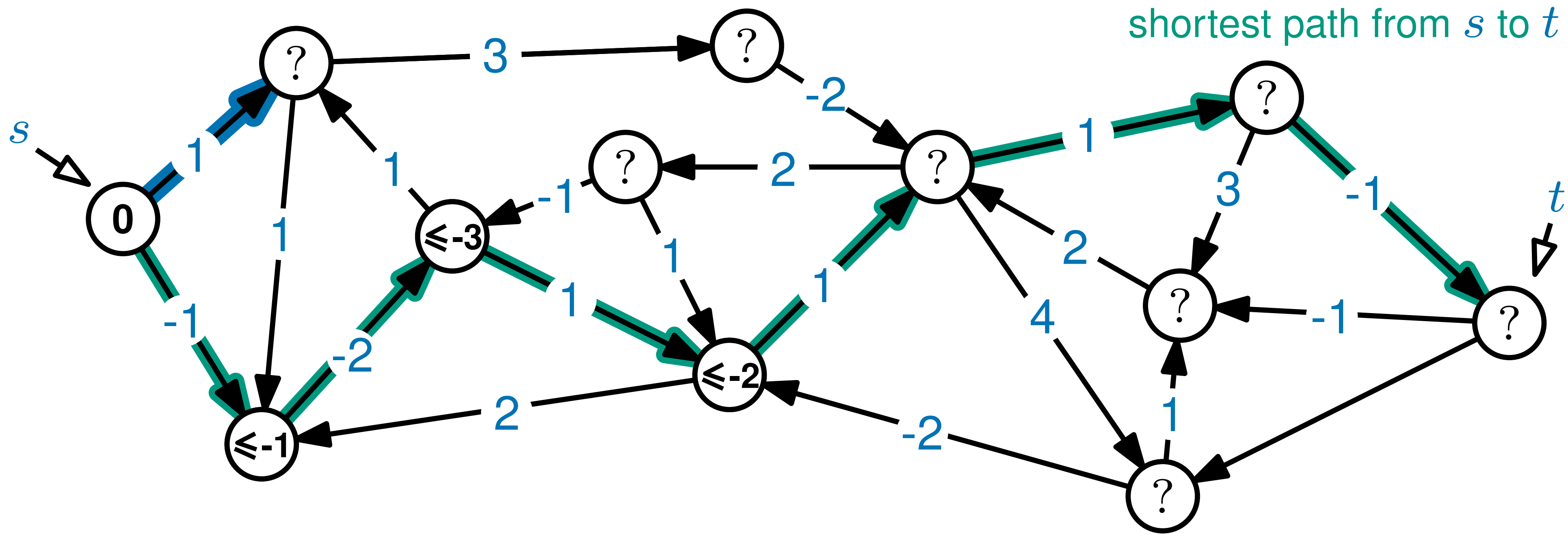
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

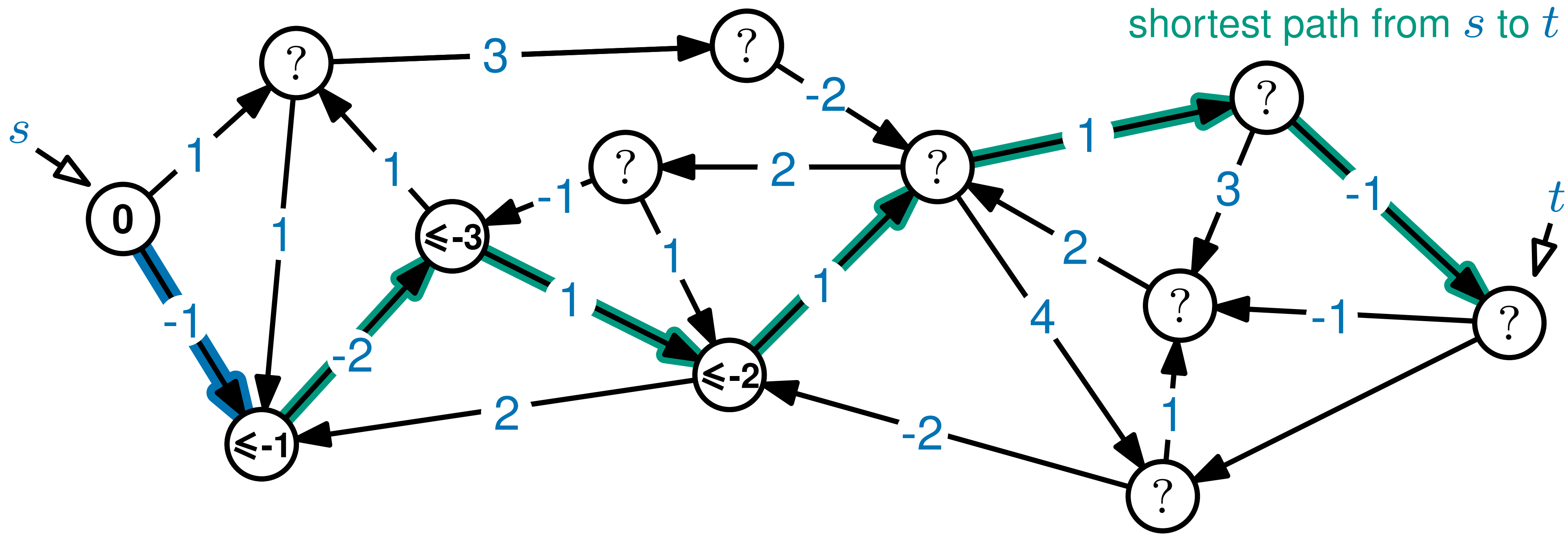
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

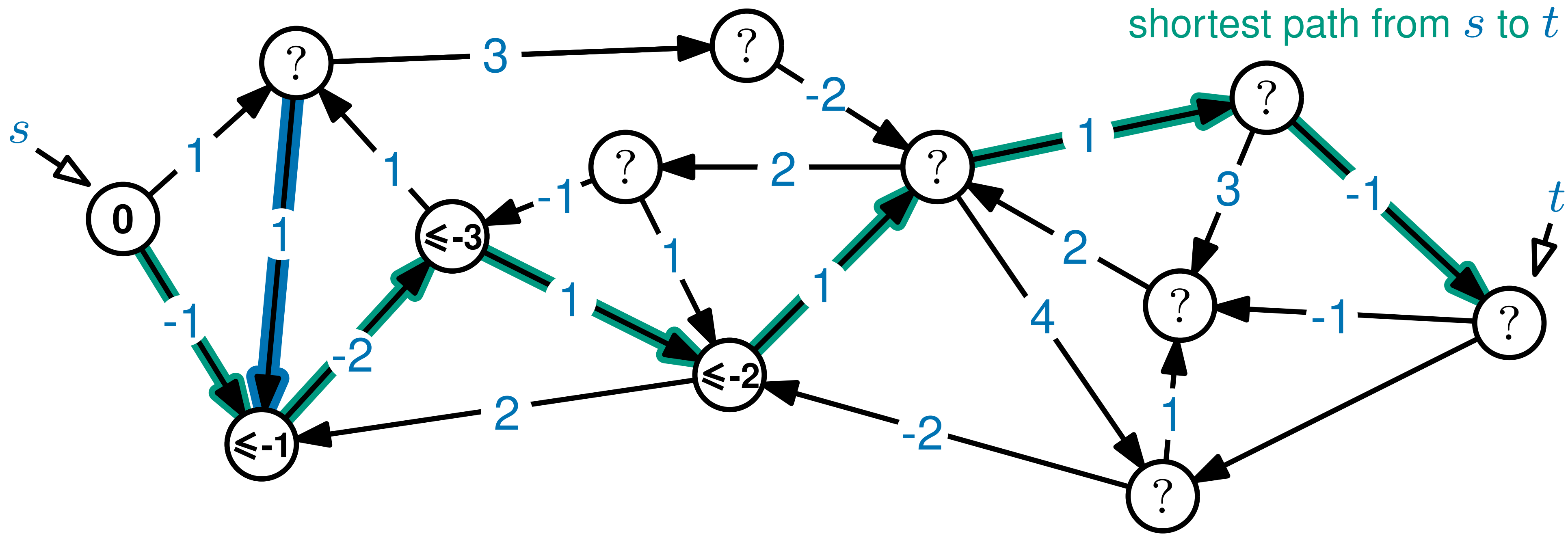
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

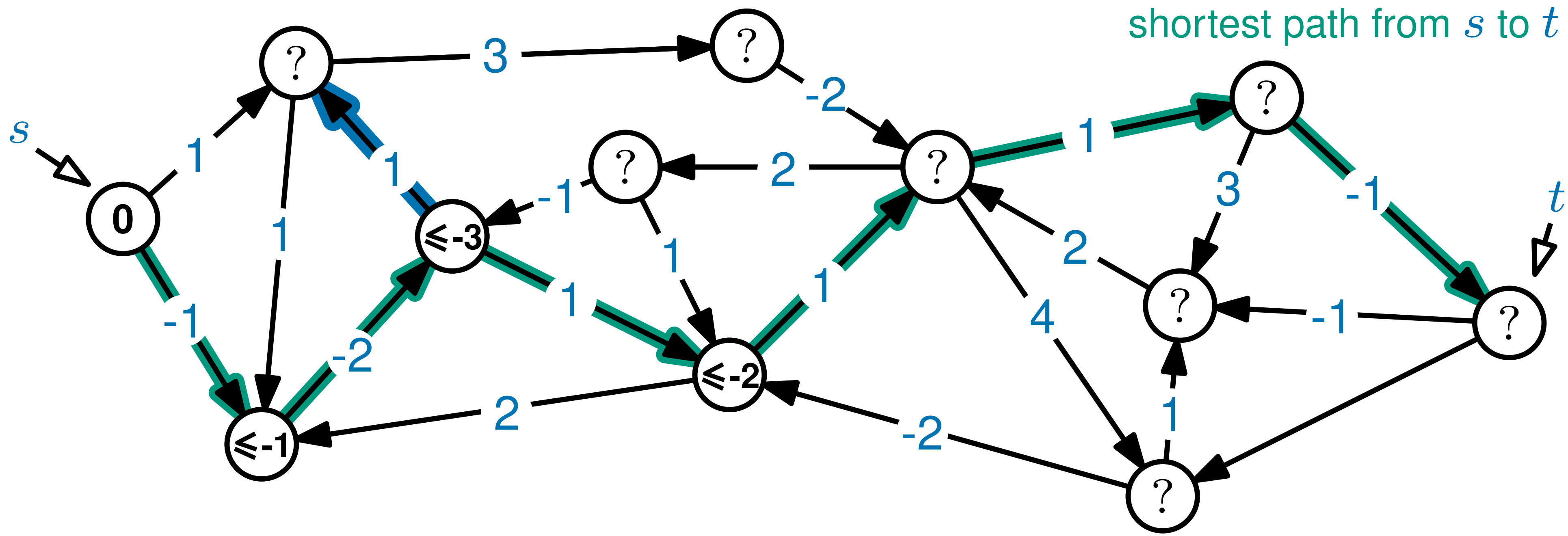
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

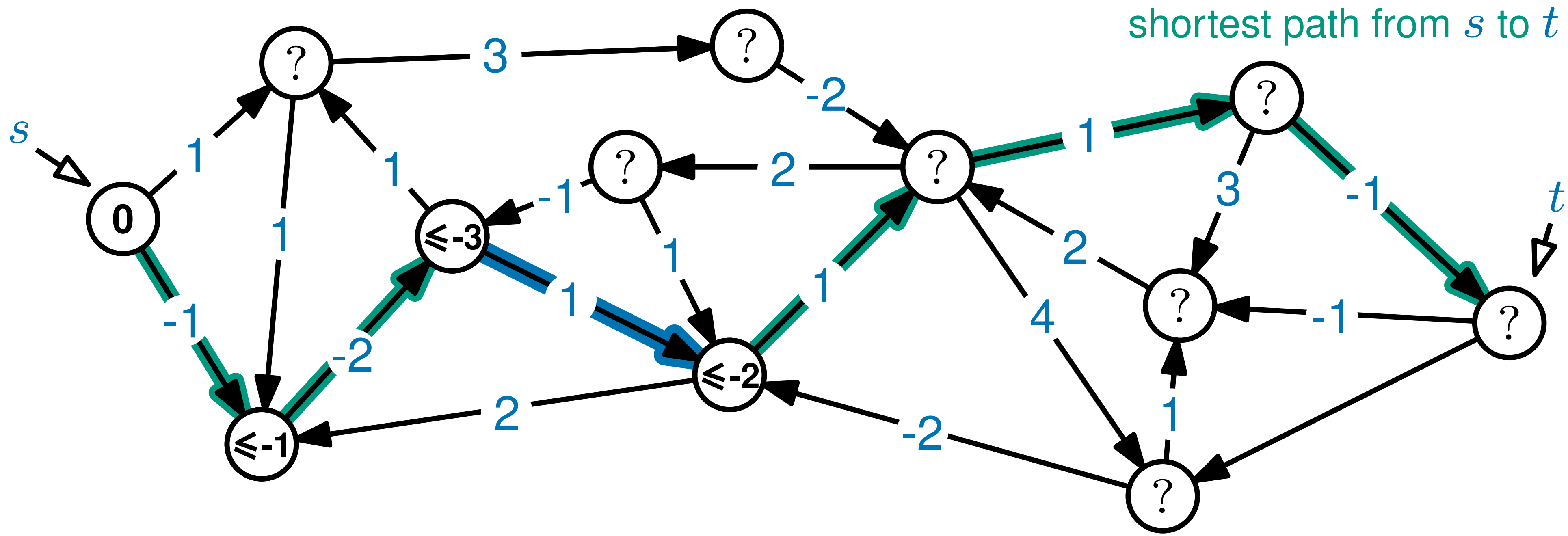
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

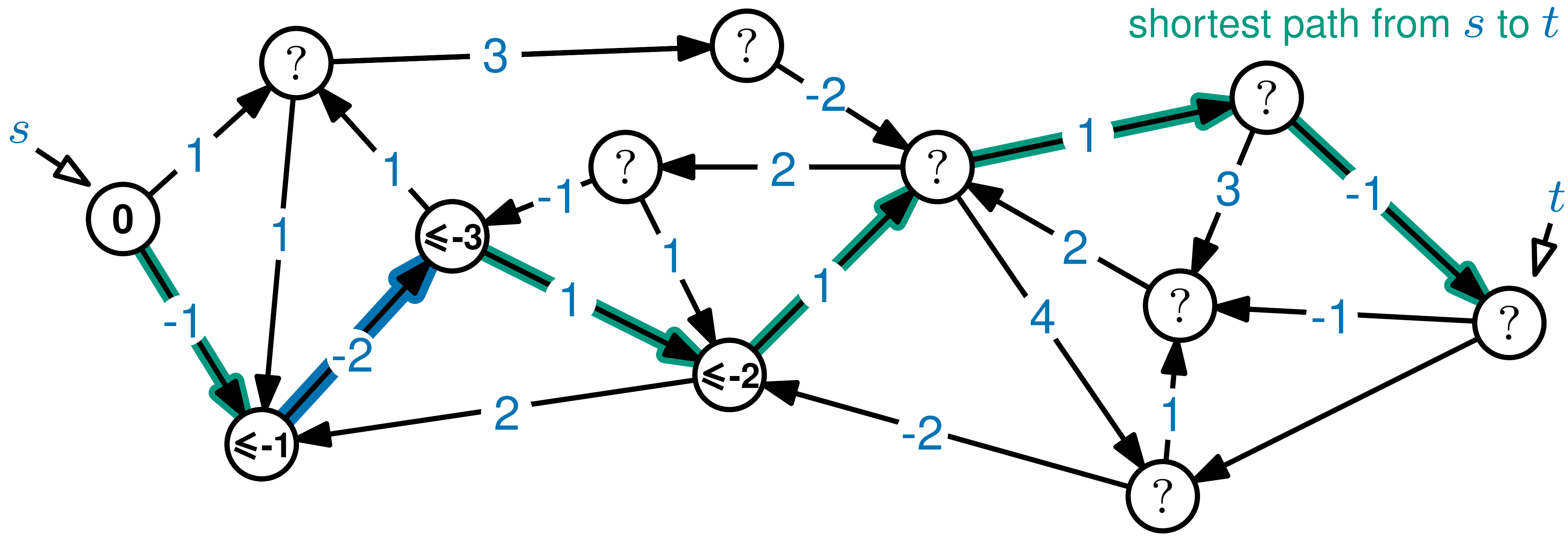
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

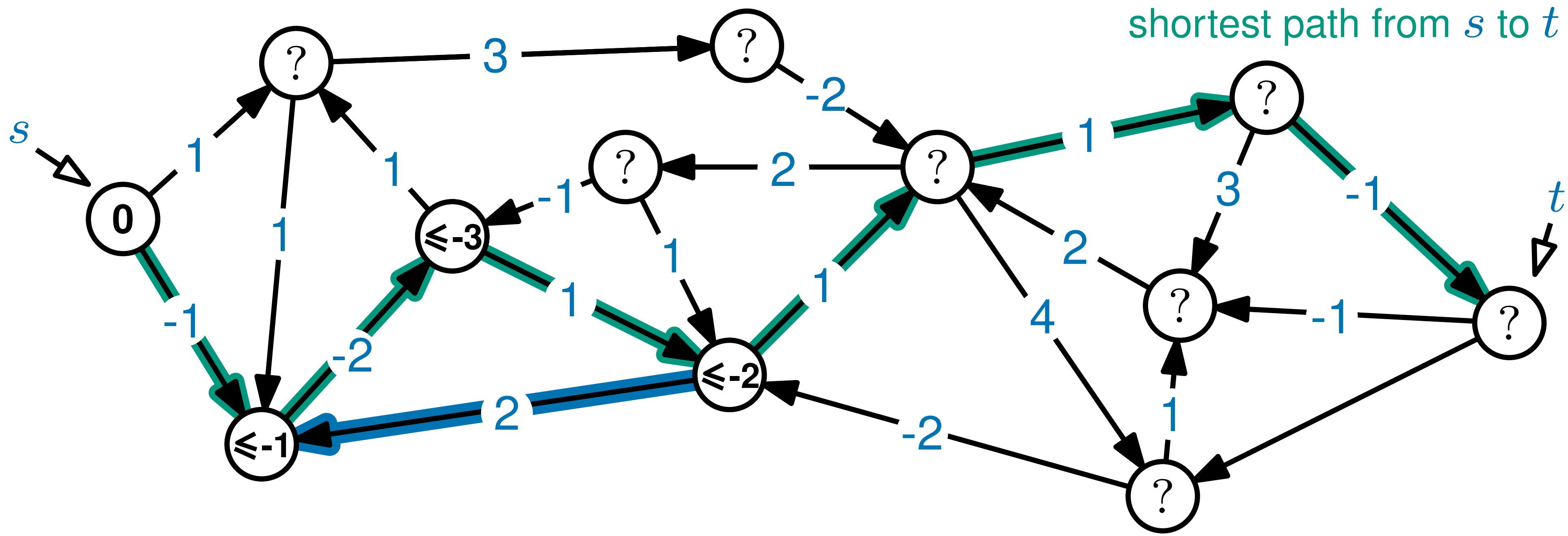
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

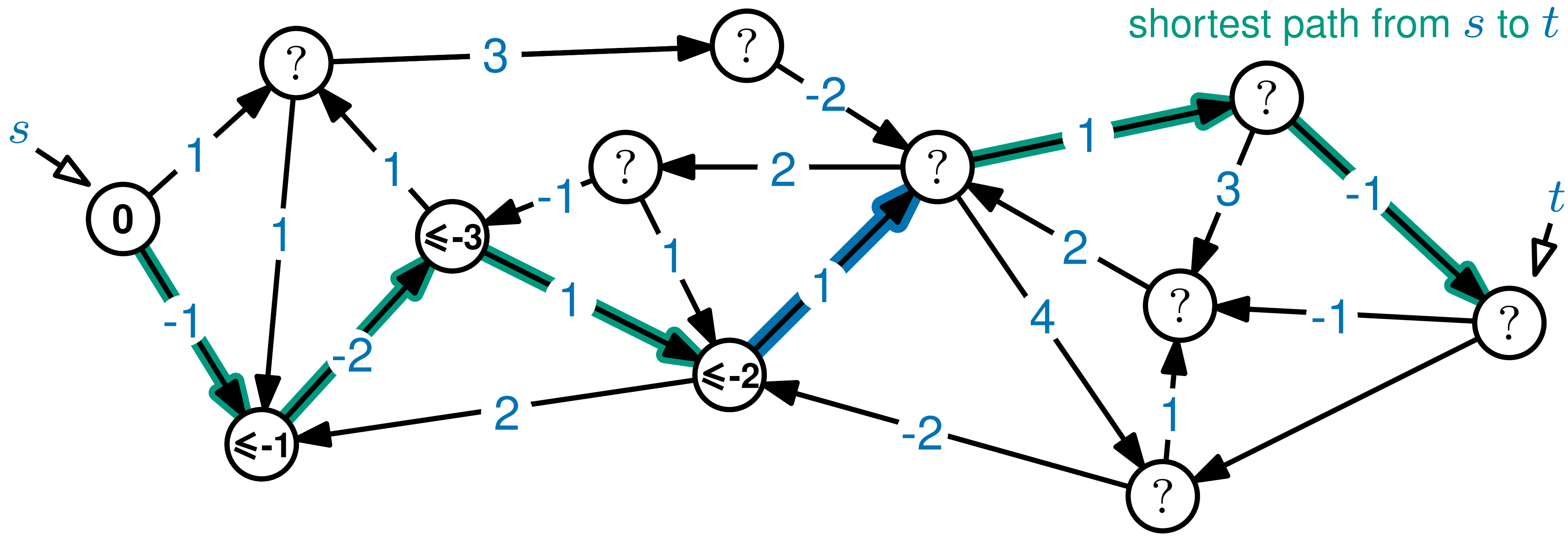
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:



At some point in iteration i

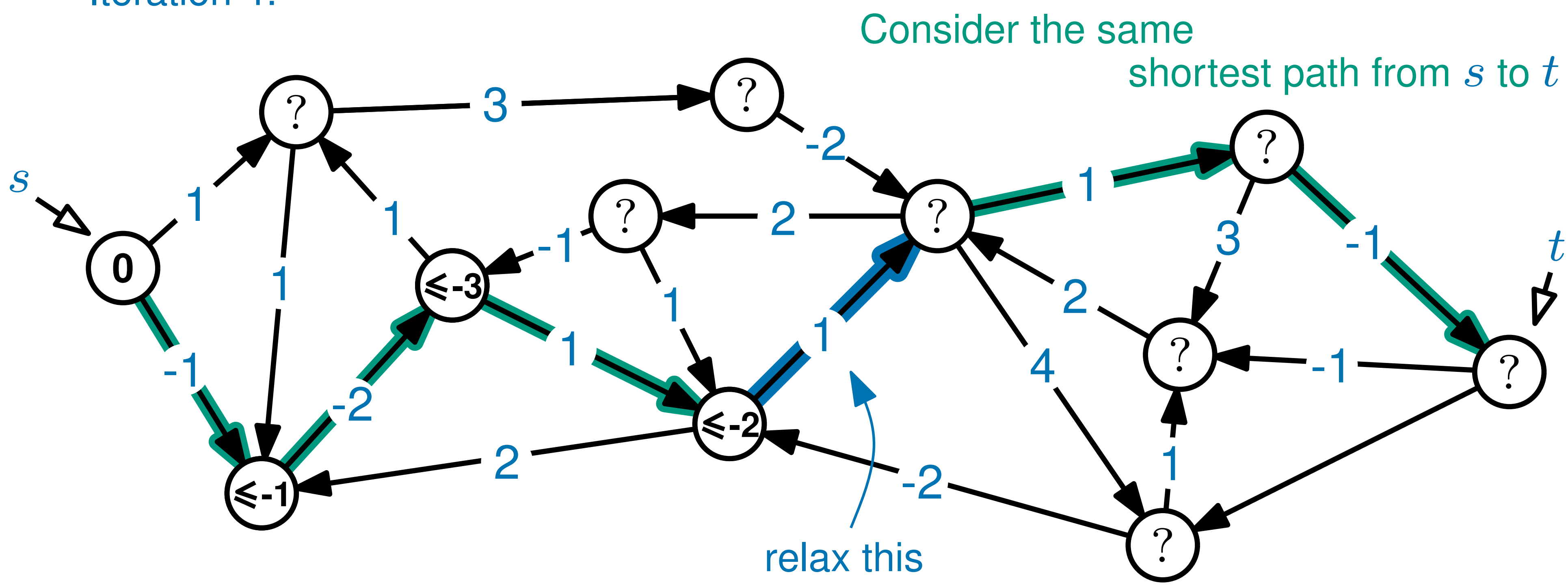
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:



At some point in iteration i

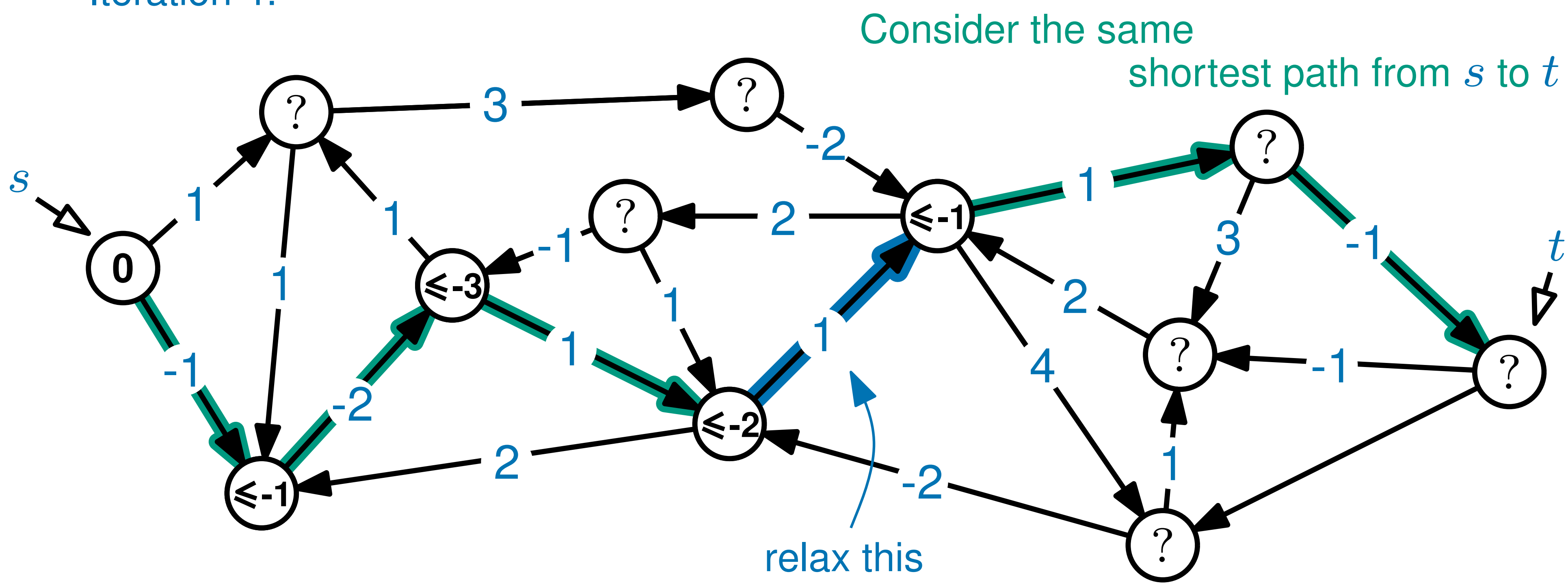
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

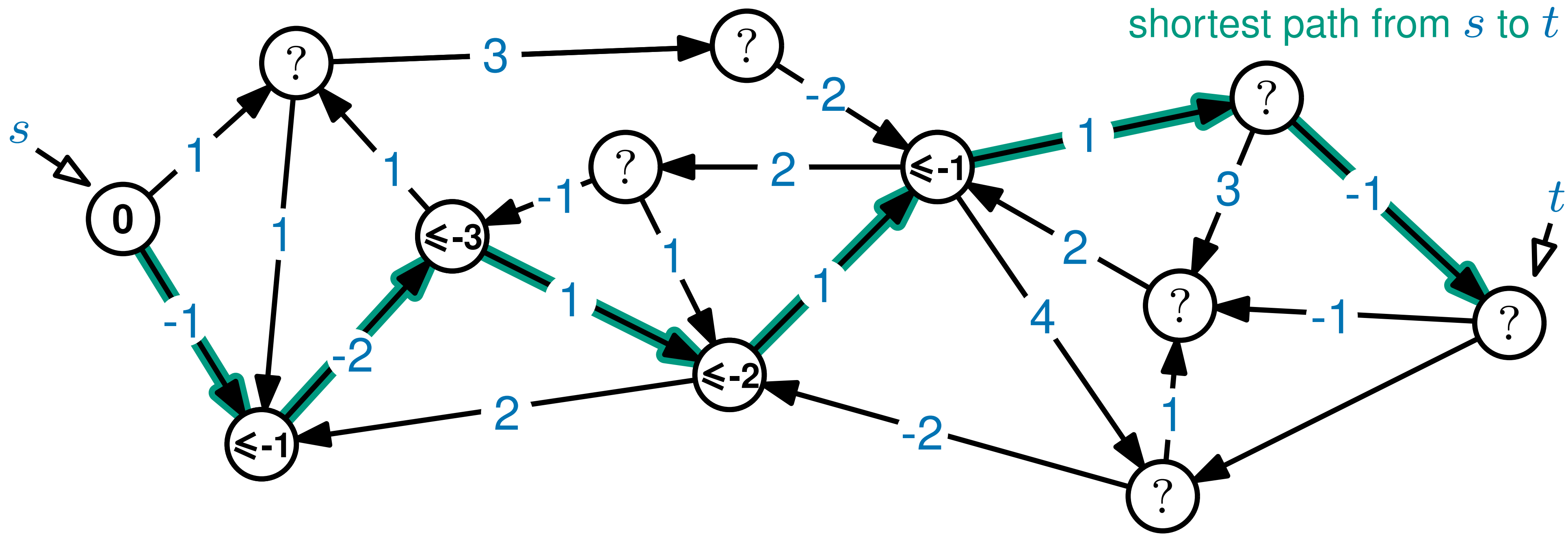
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

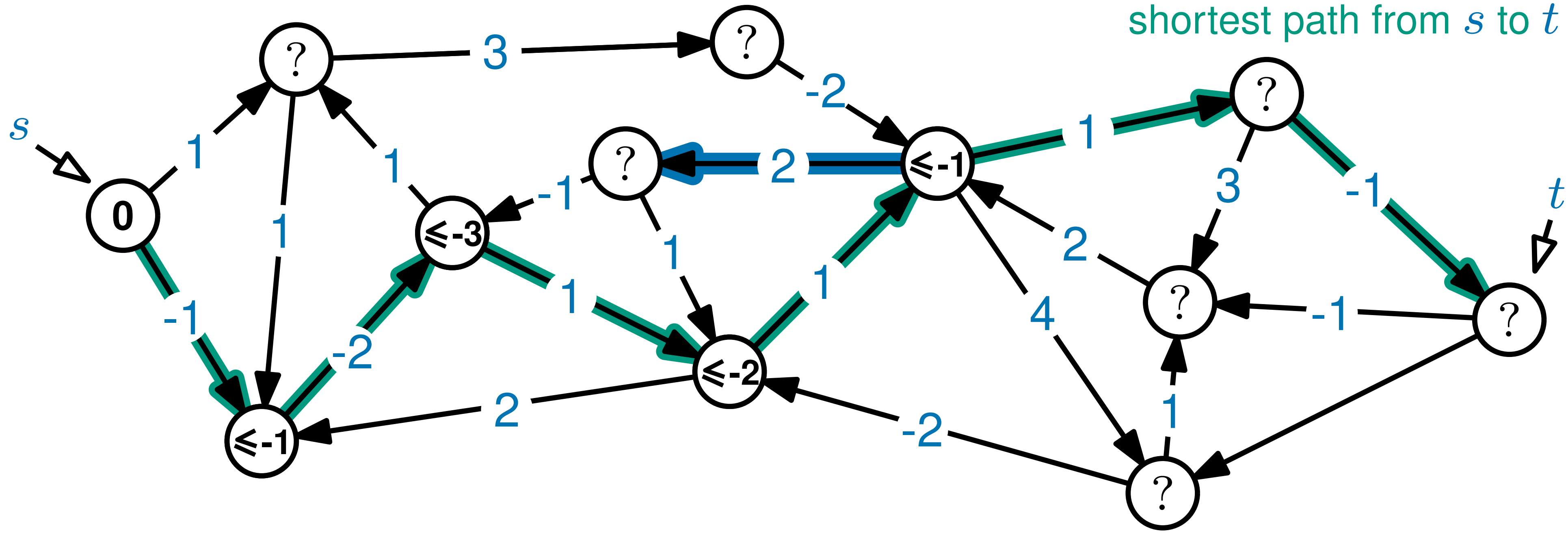
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

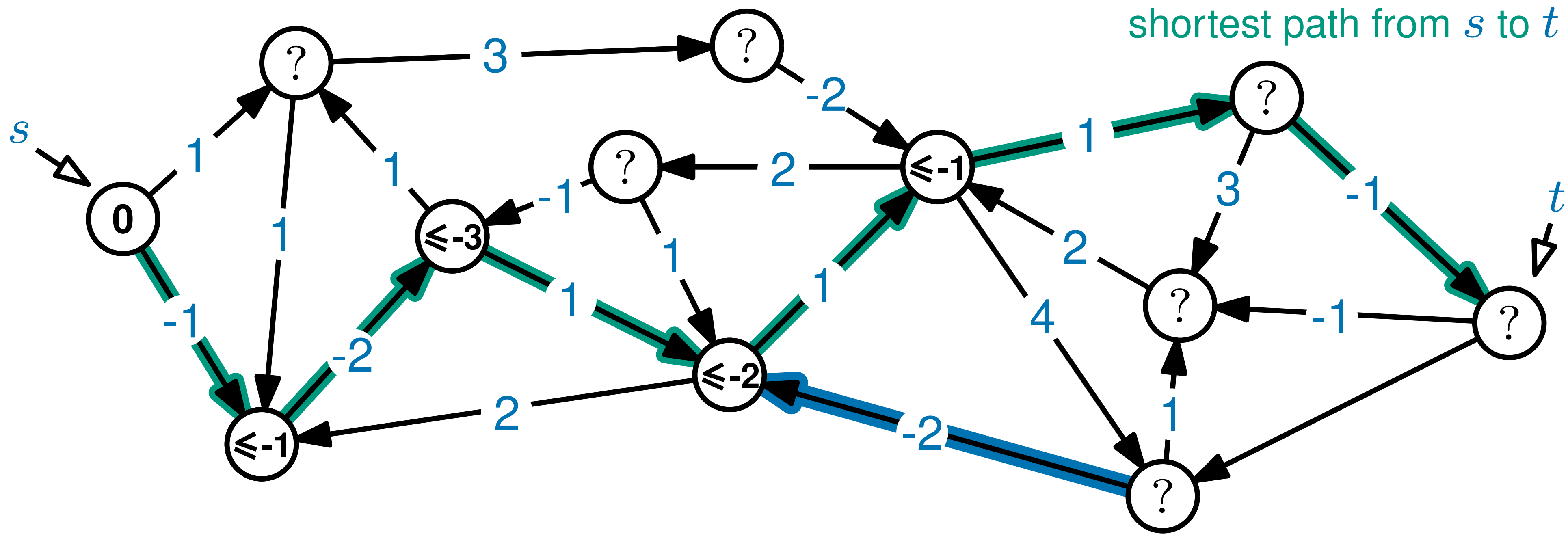
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

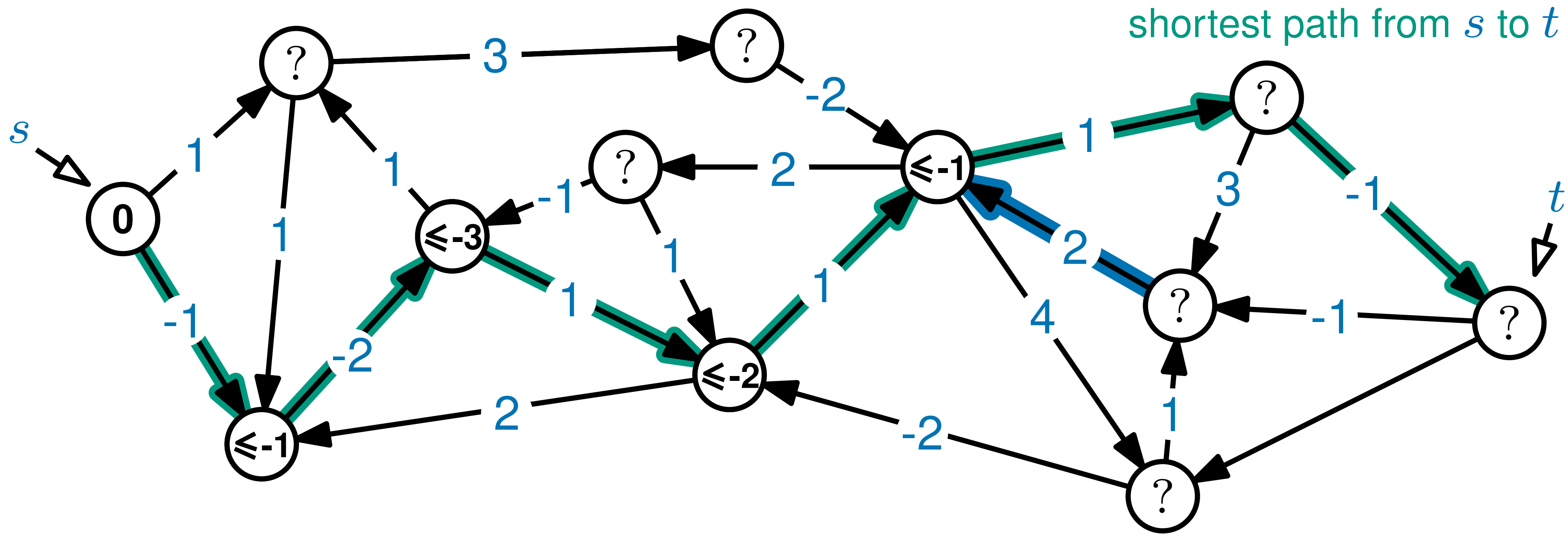
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

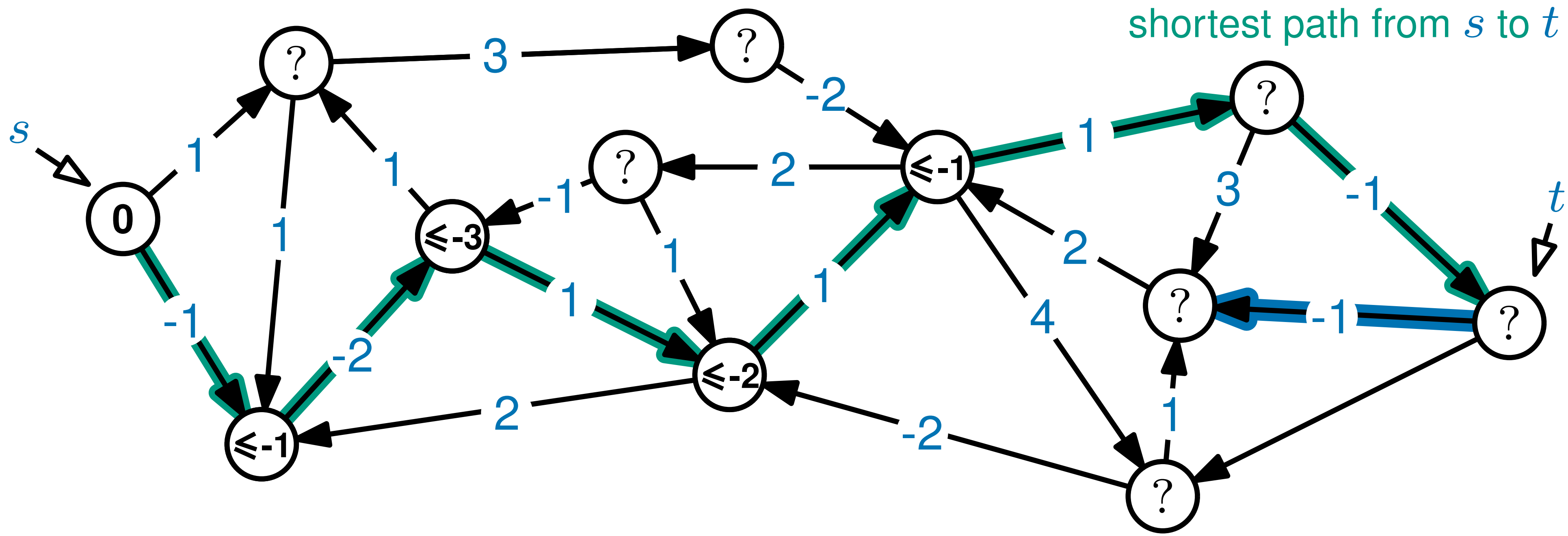
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

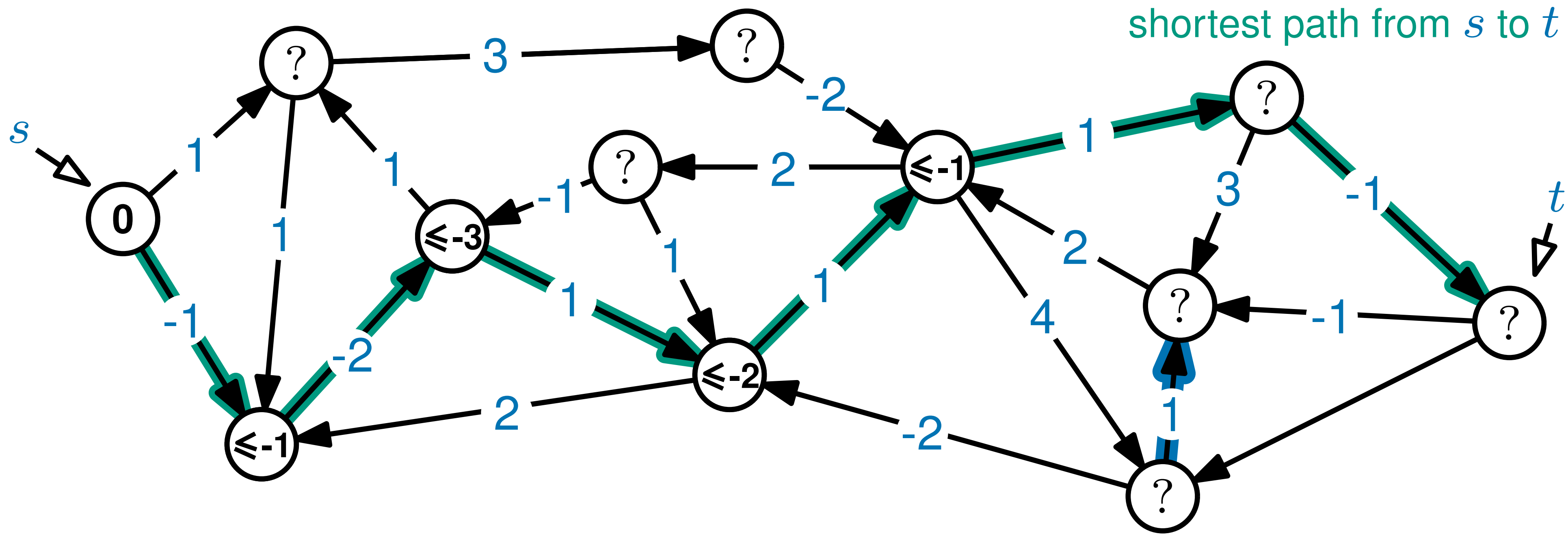
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

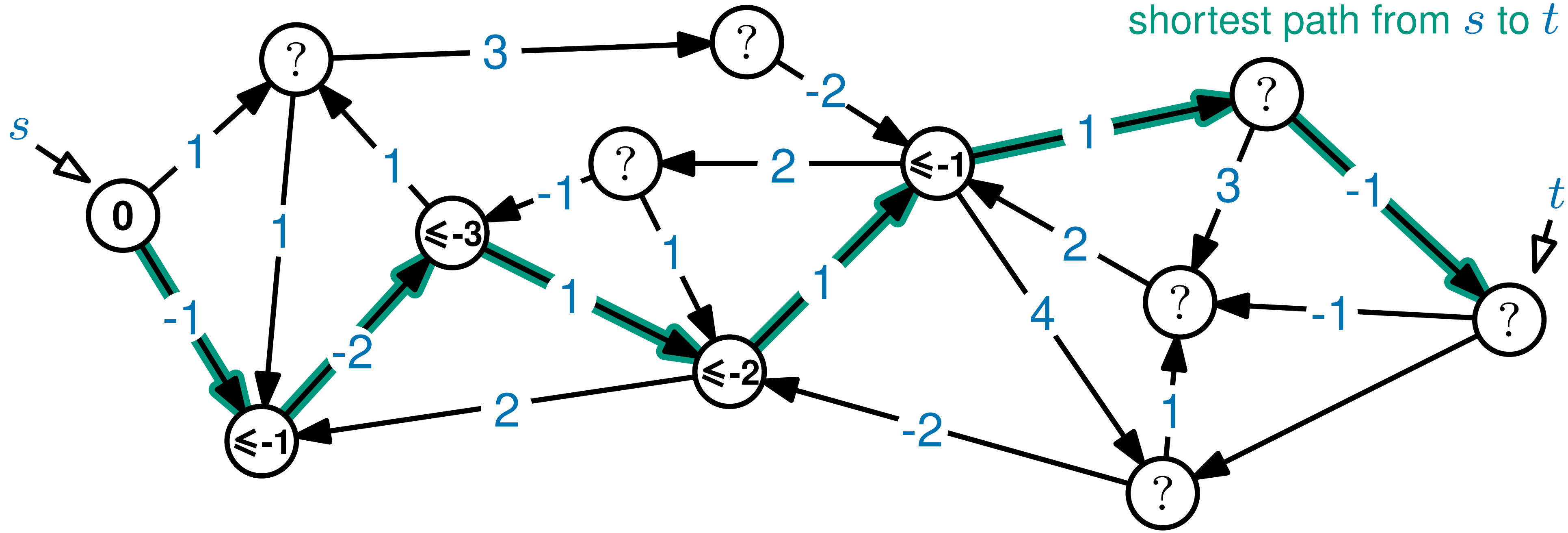
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 4:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

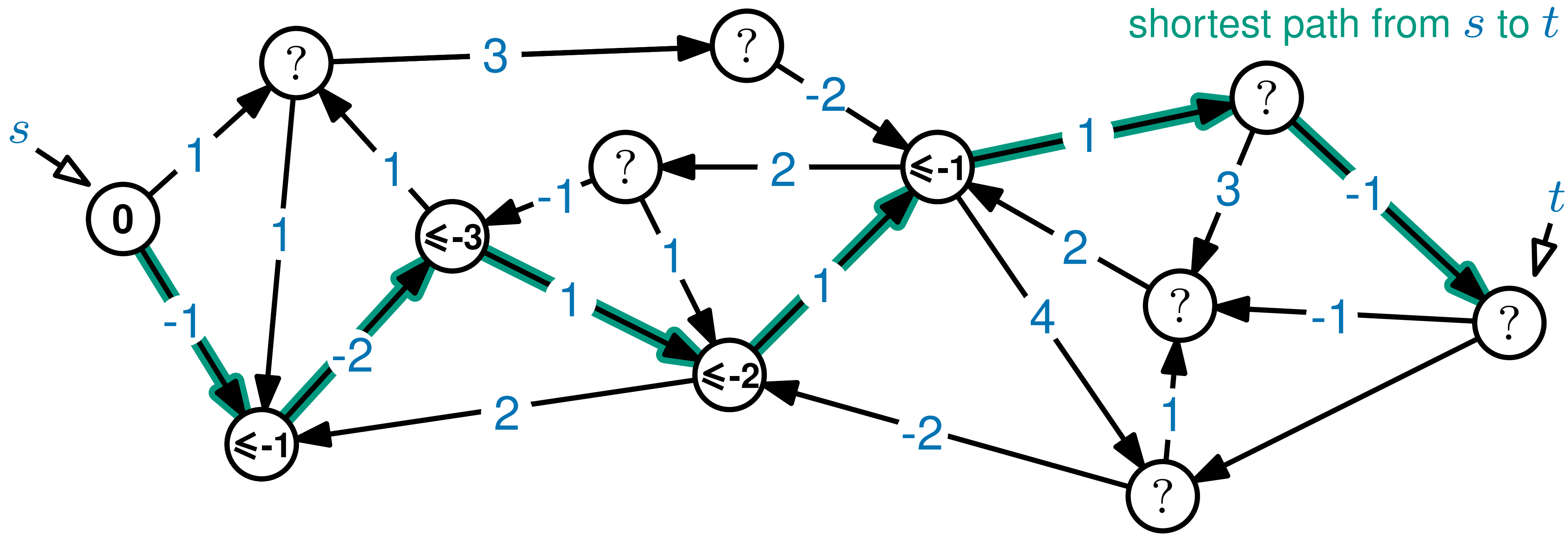
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

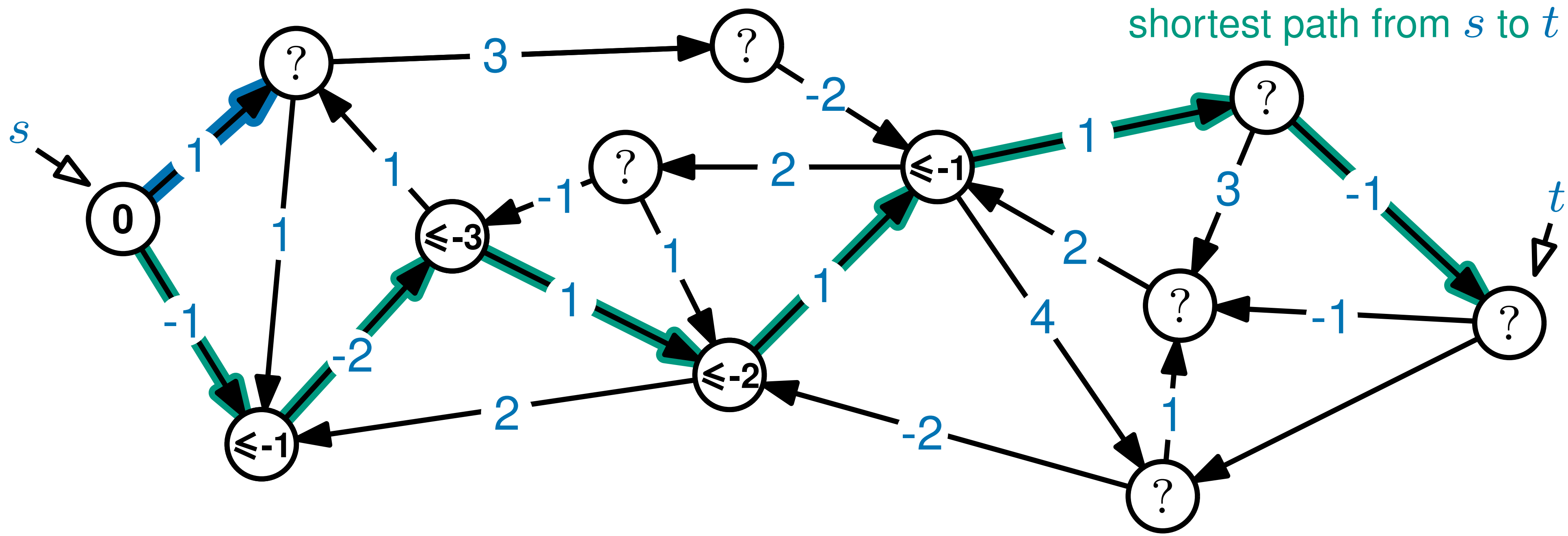
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

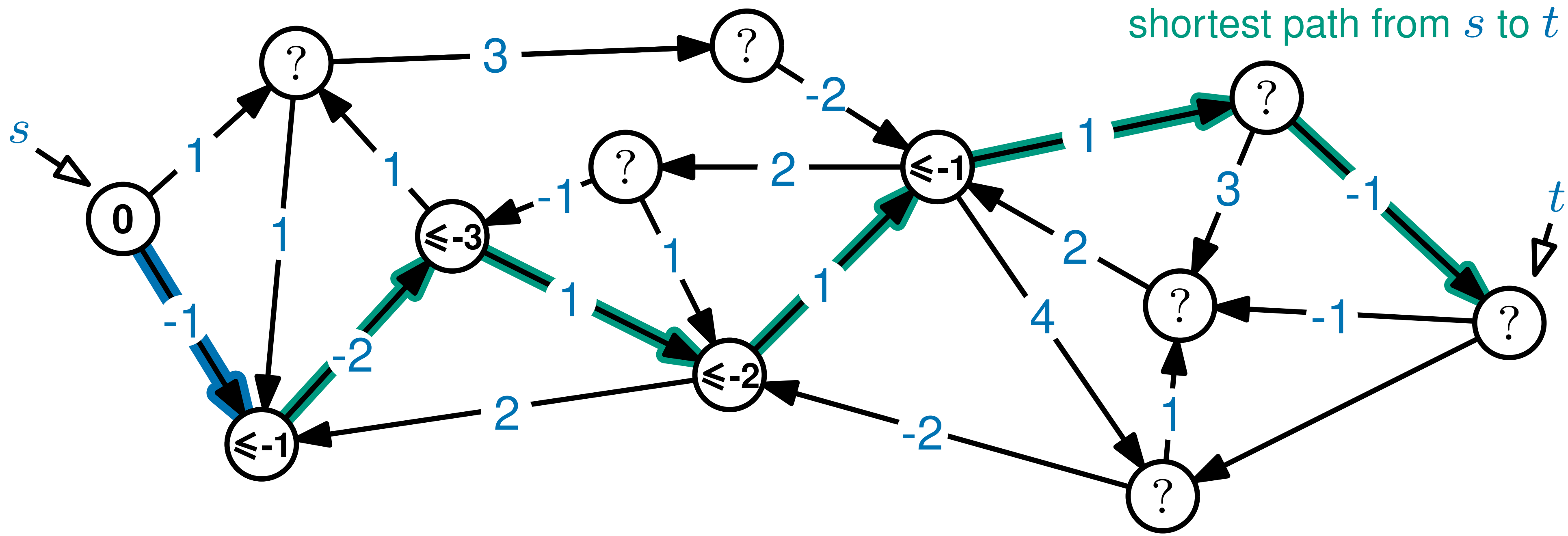
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

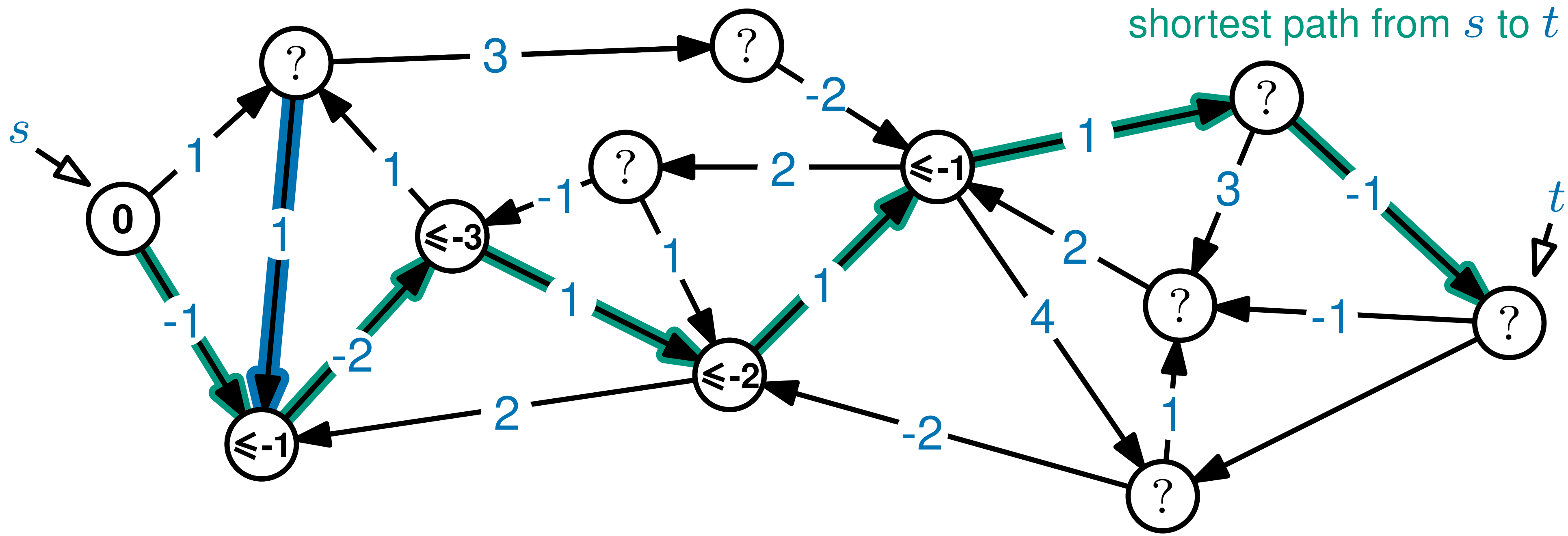
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

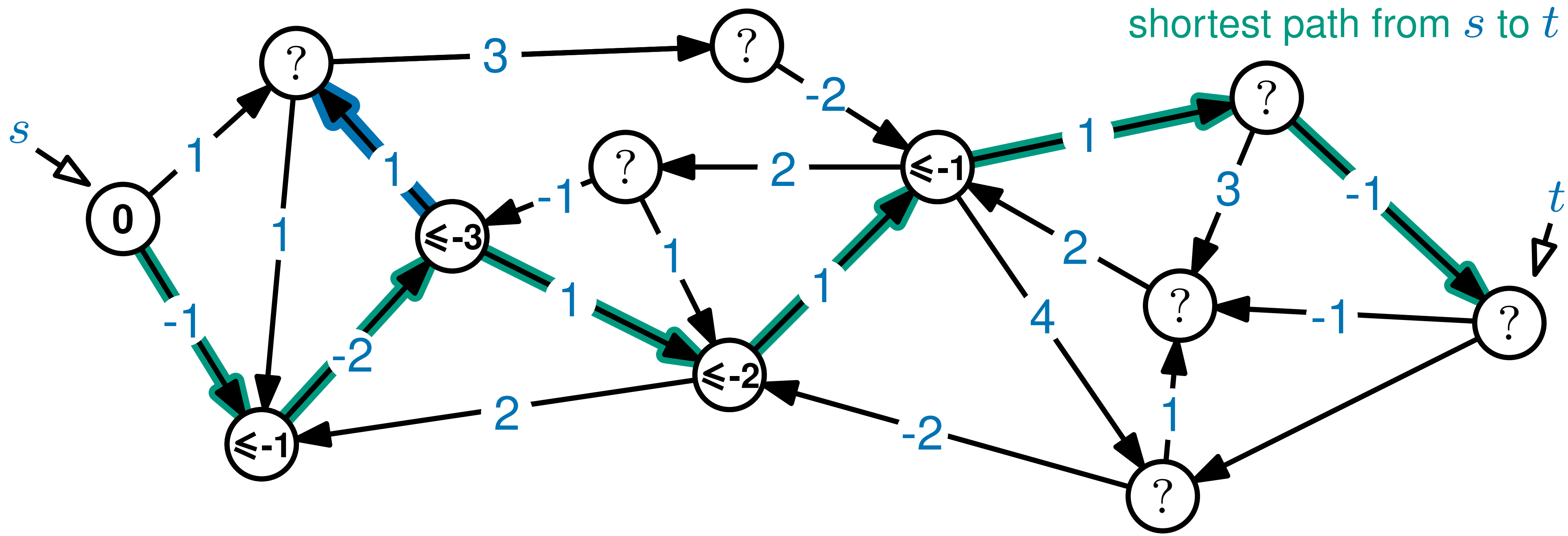
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

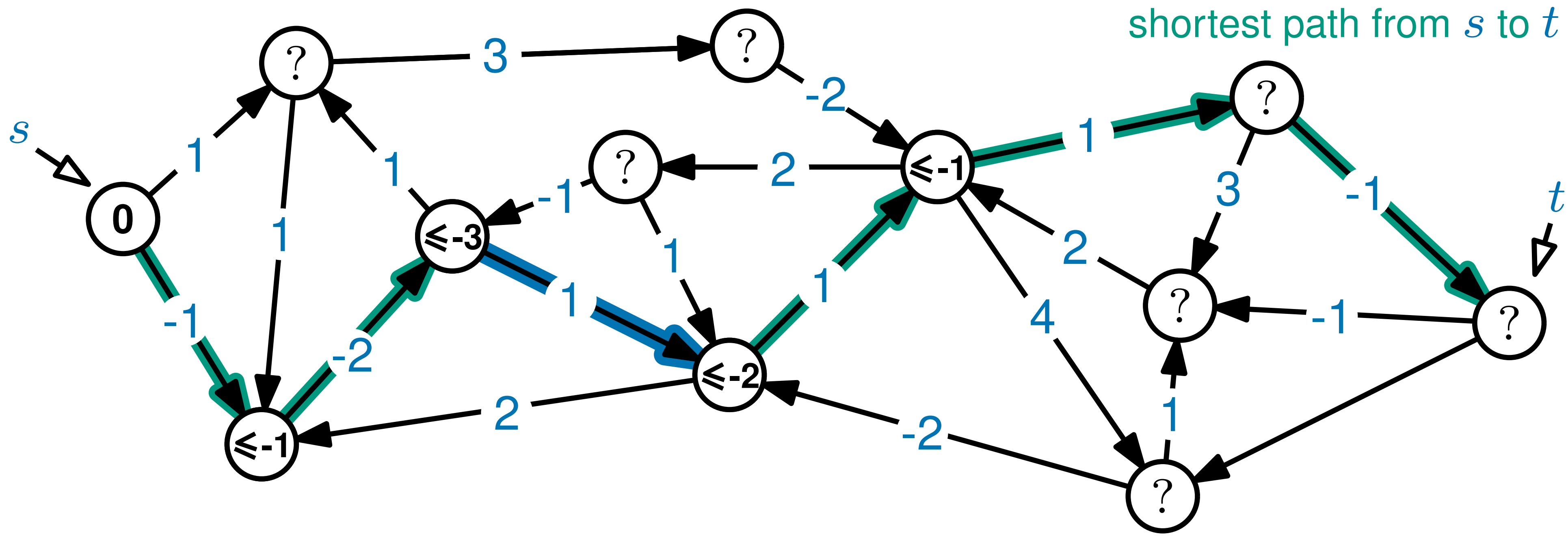
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

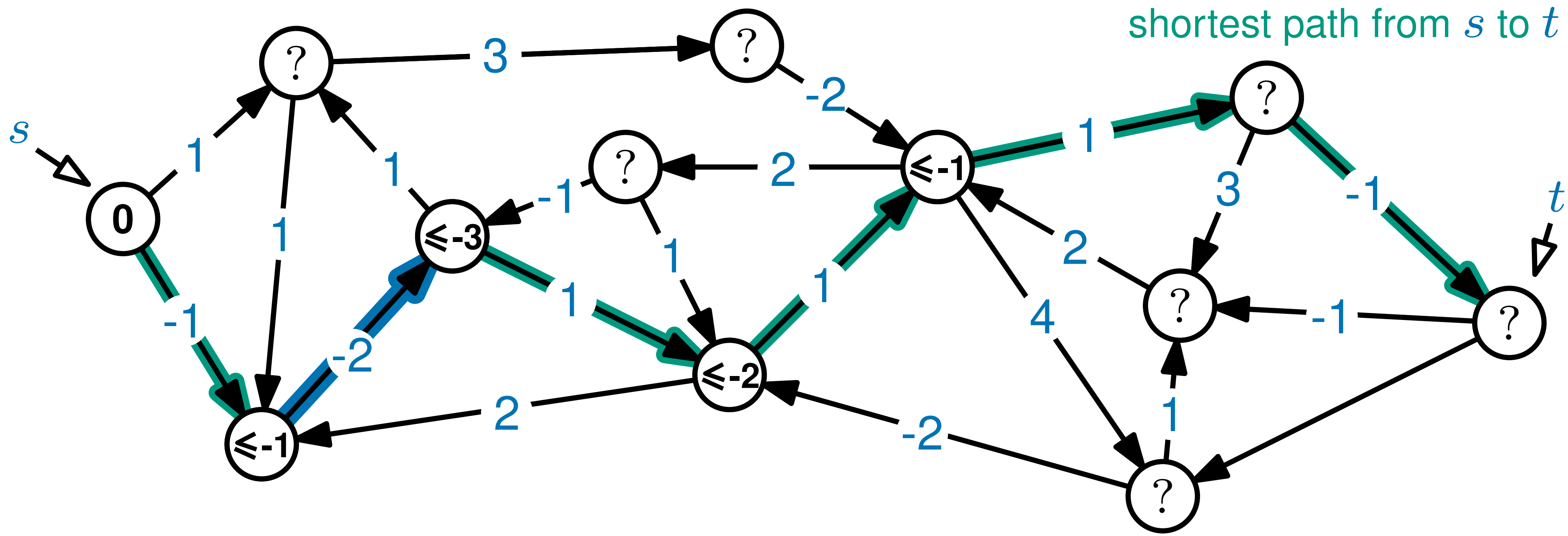
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

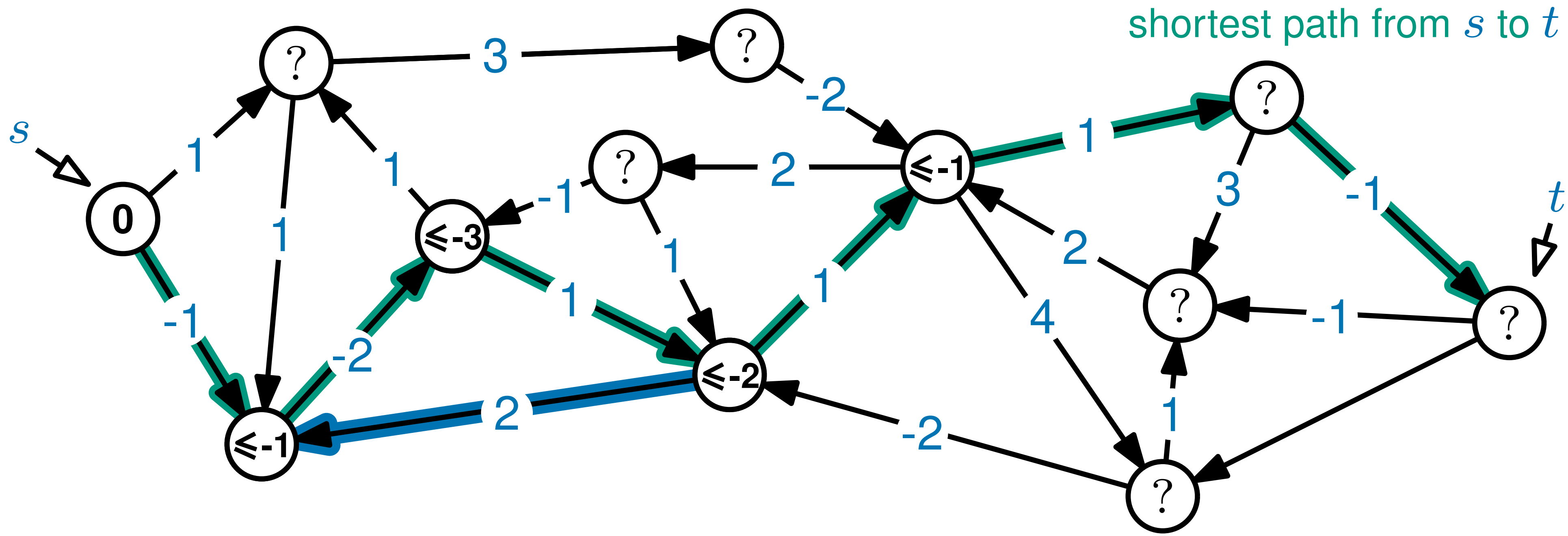
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

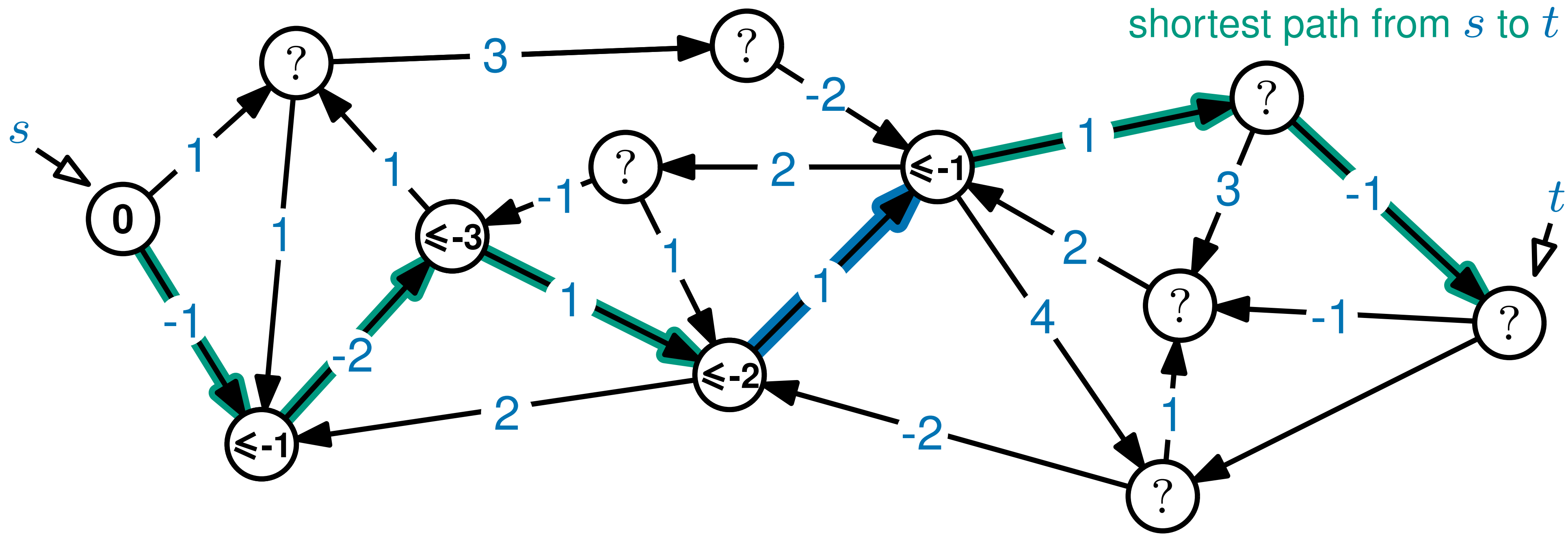
you relax the i-th edge in the shortest path from s to t. . .

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

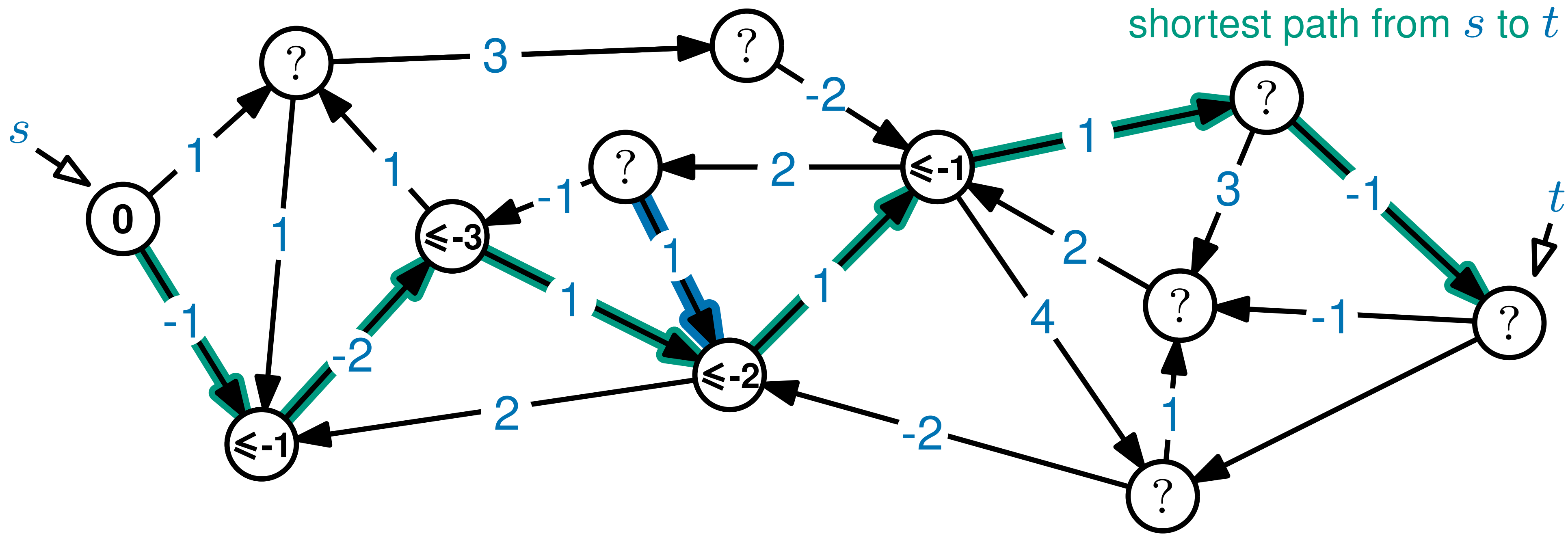
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

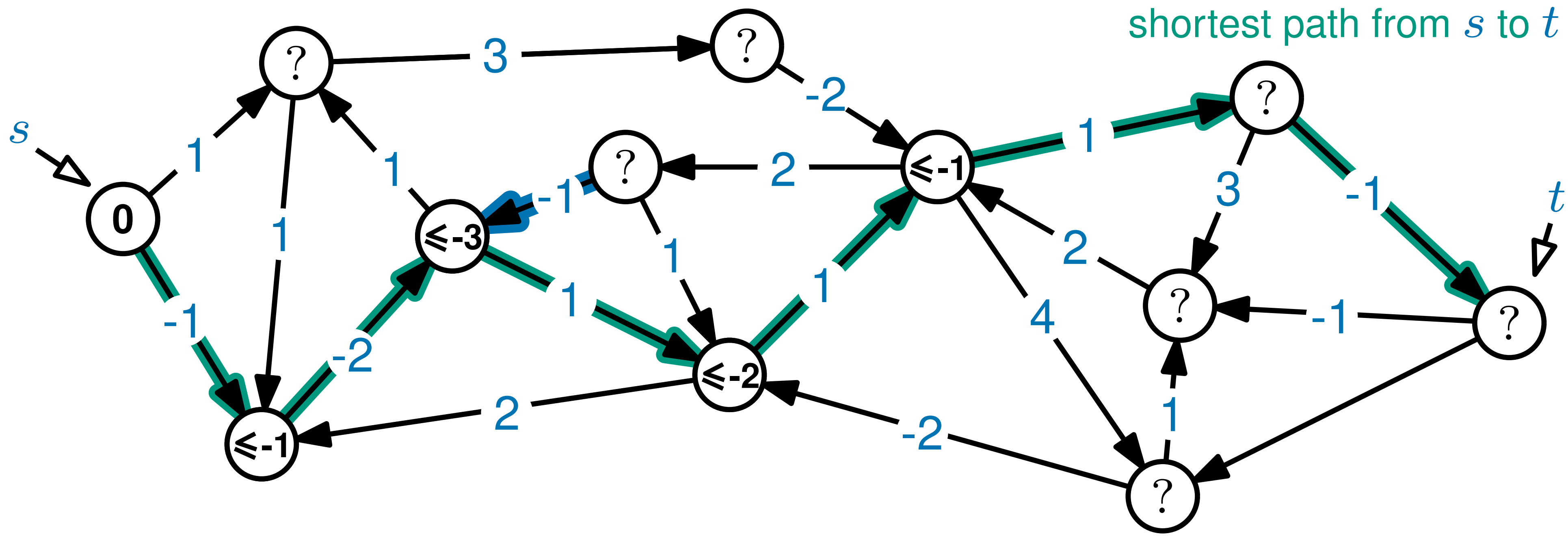
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

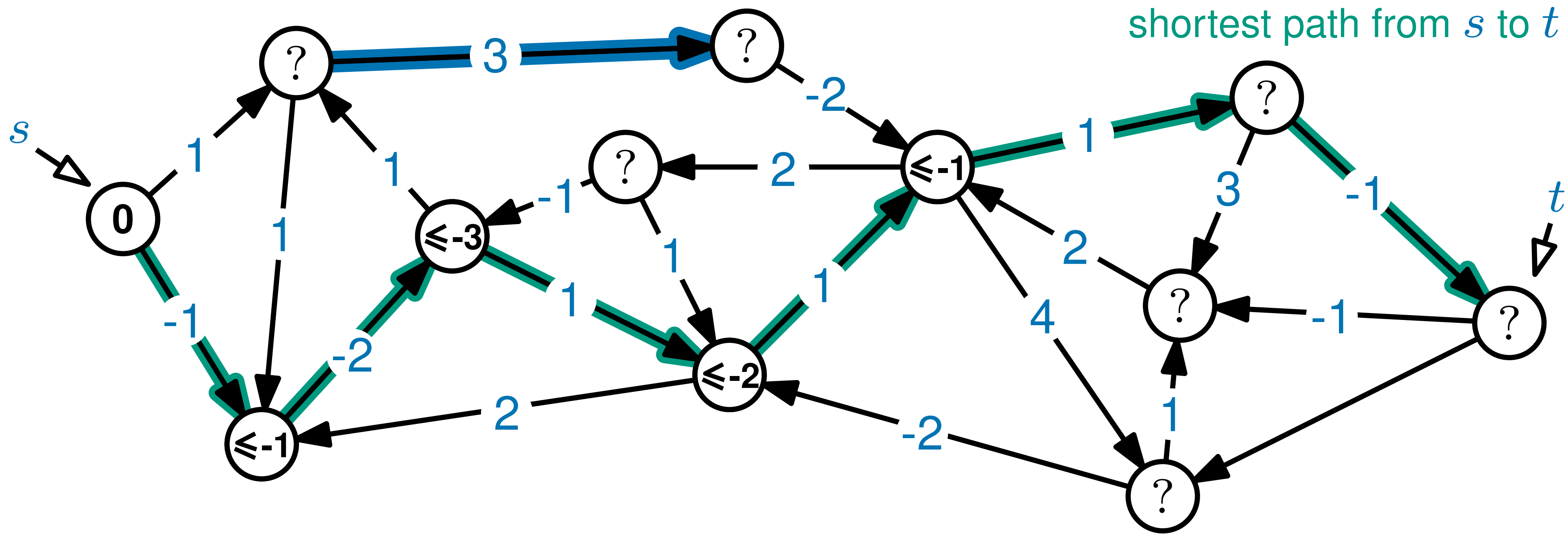
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



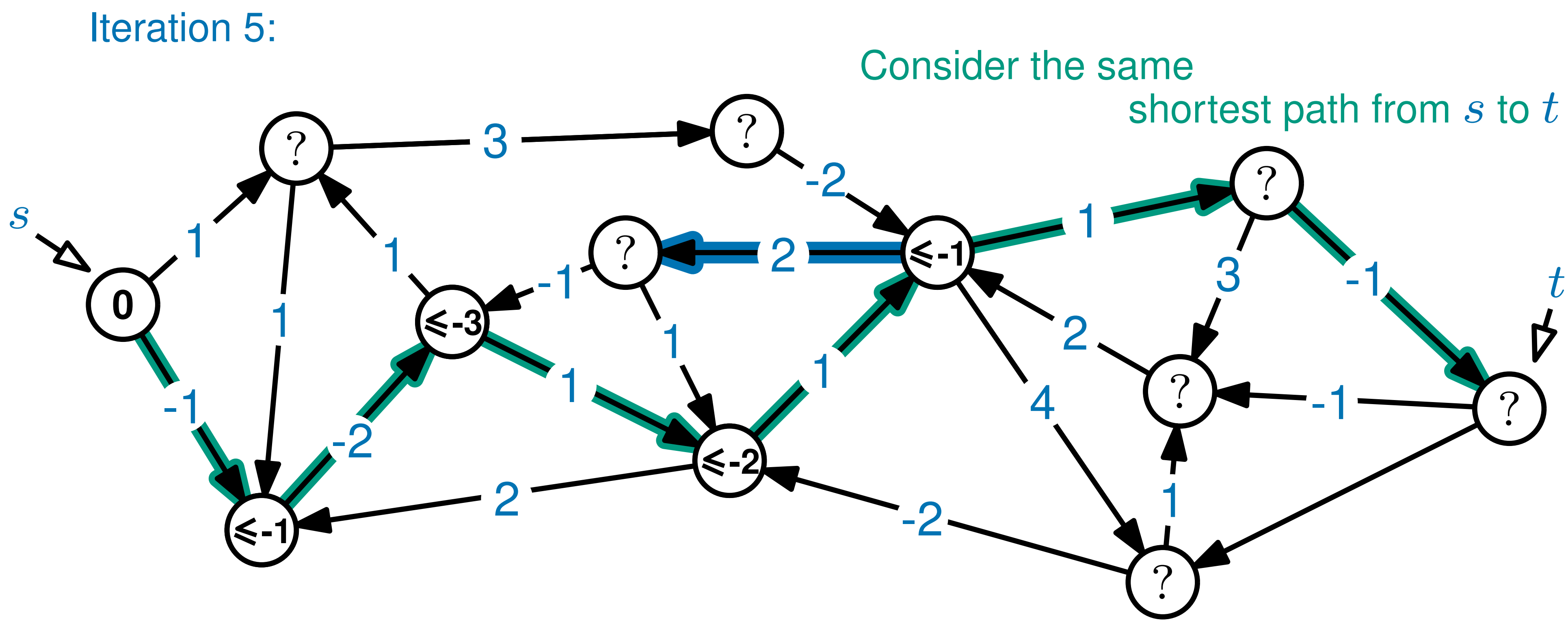
At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

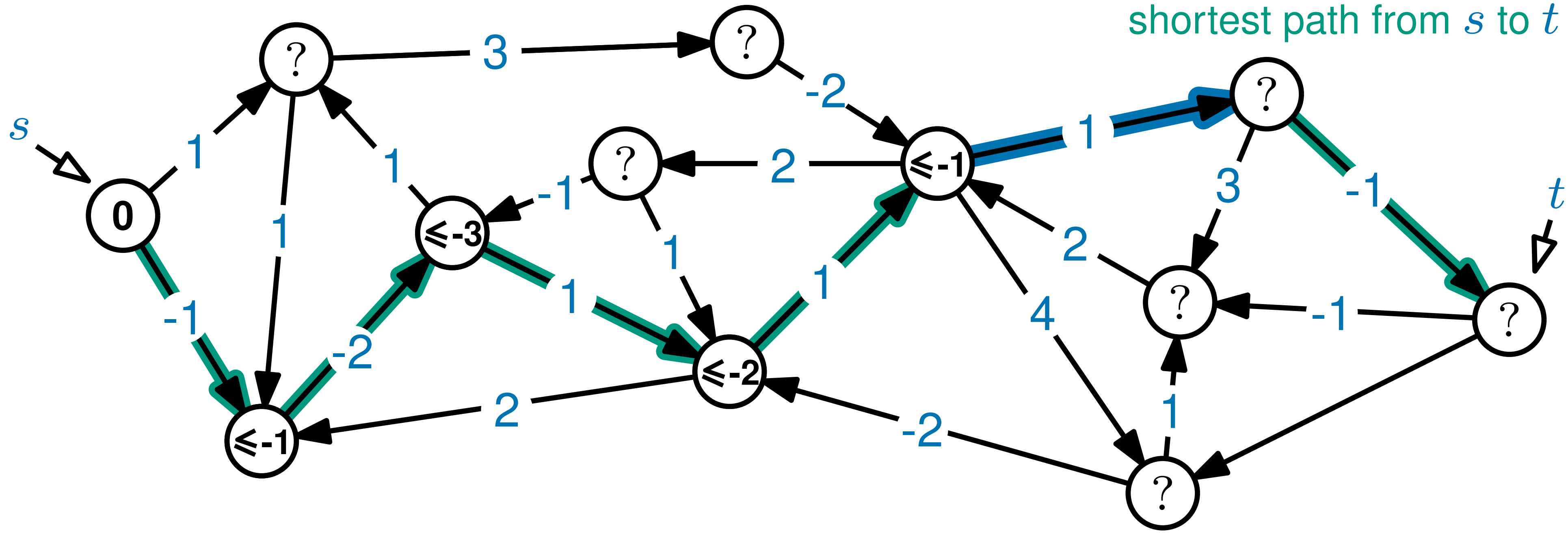
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

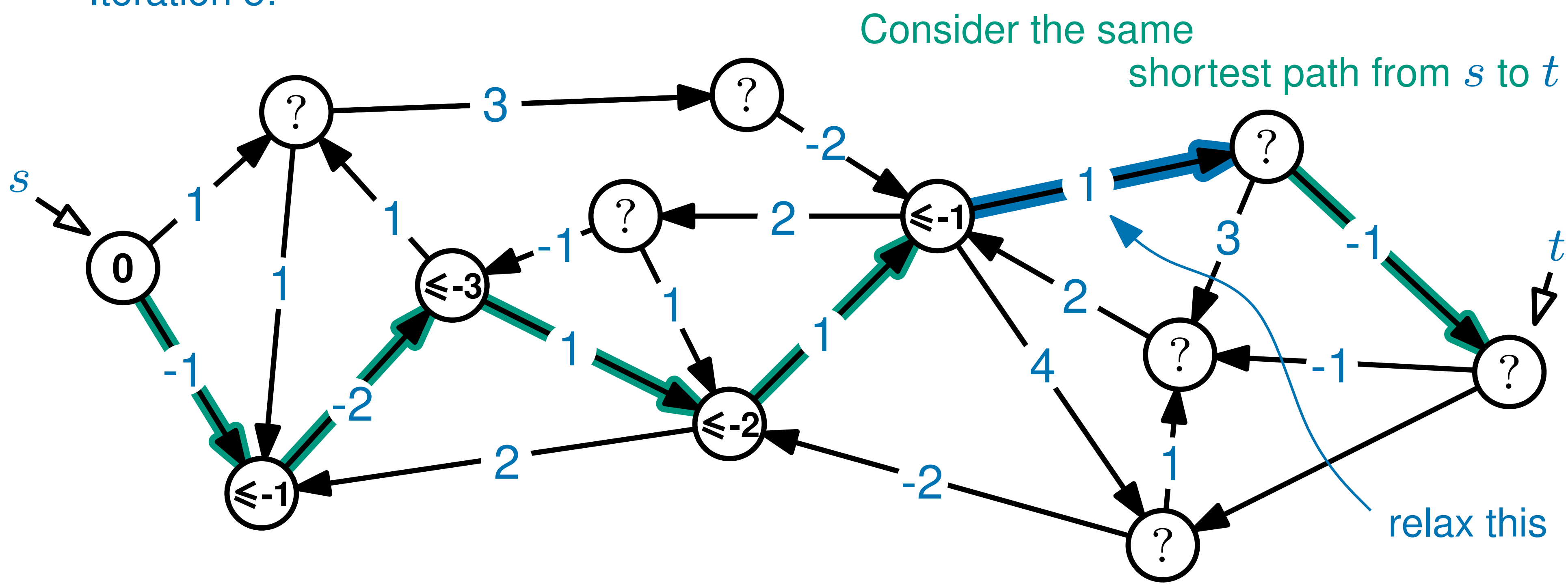
you relax the i-th edge in the shortest path from s to t. . .

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

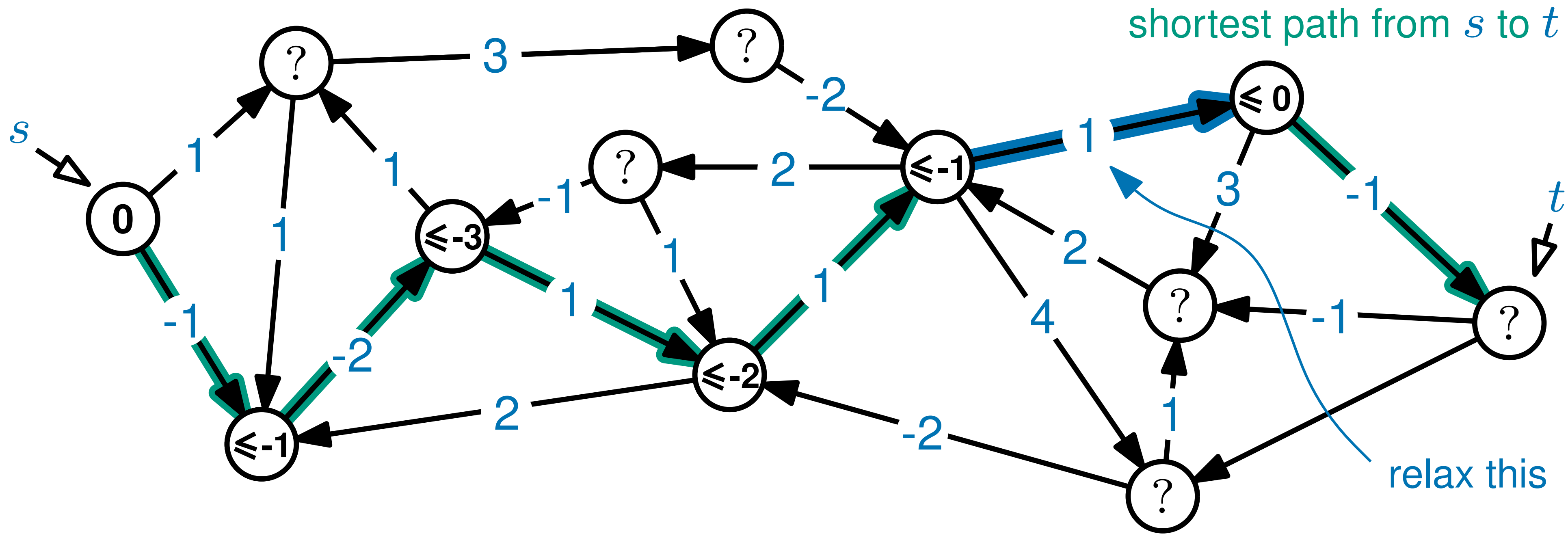
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

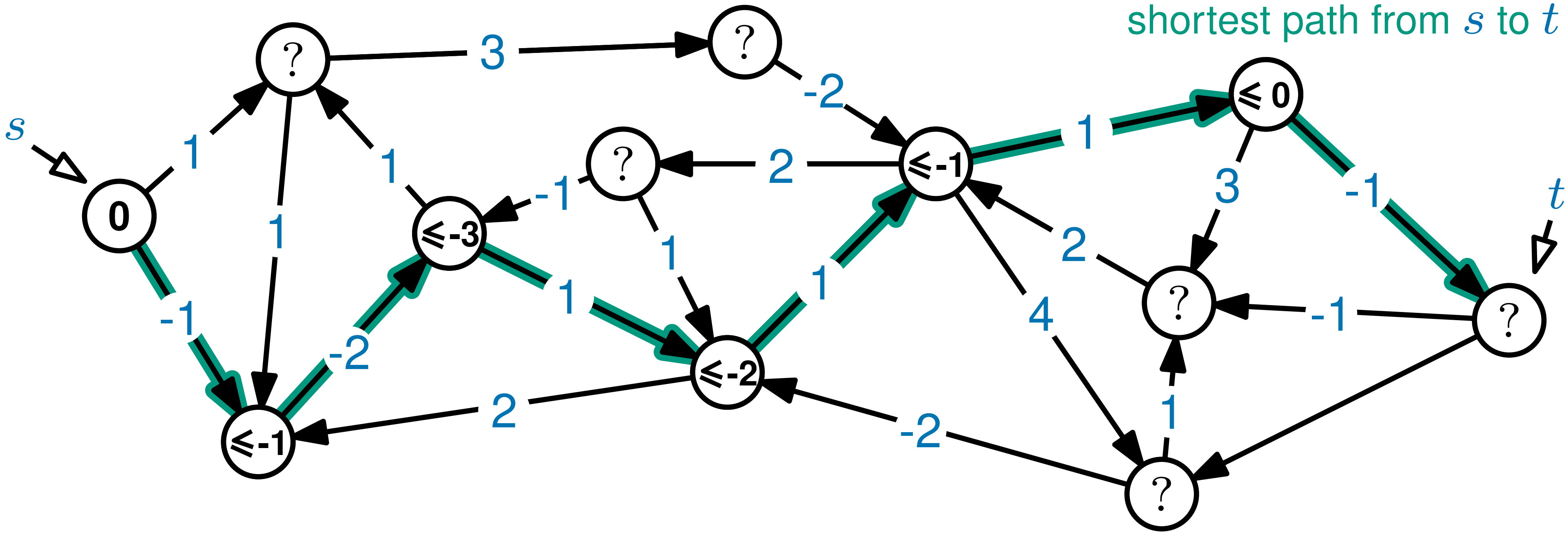
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

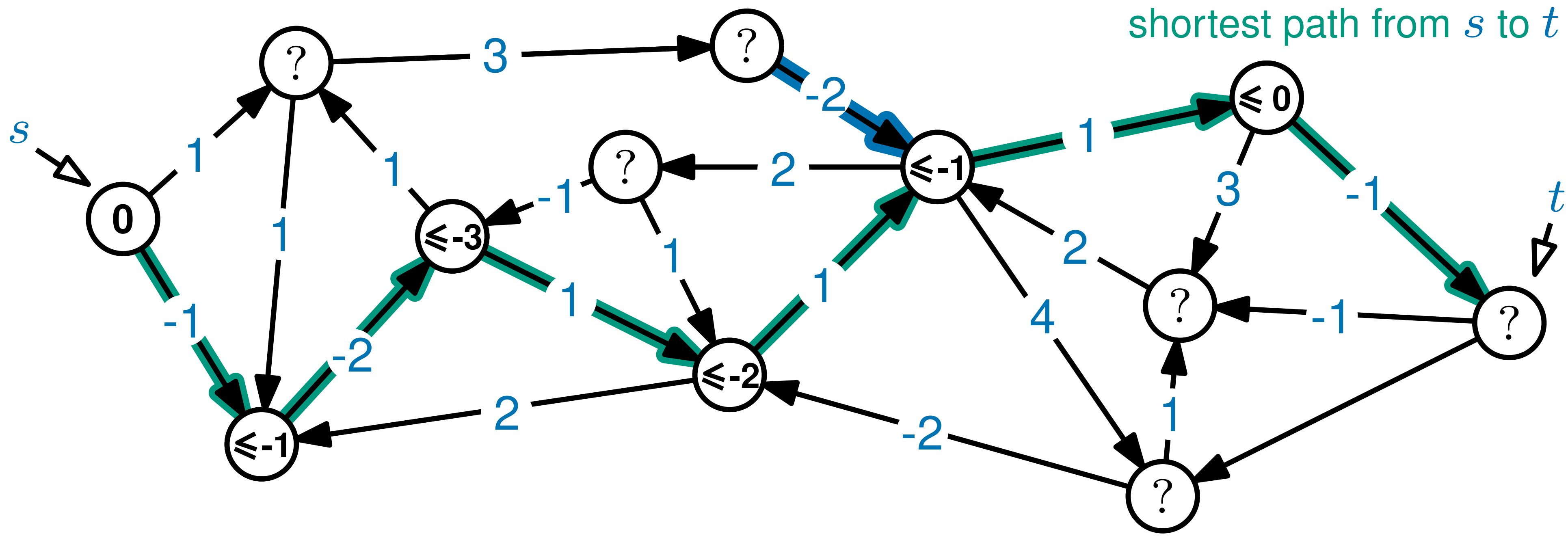
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

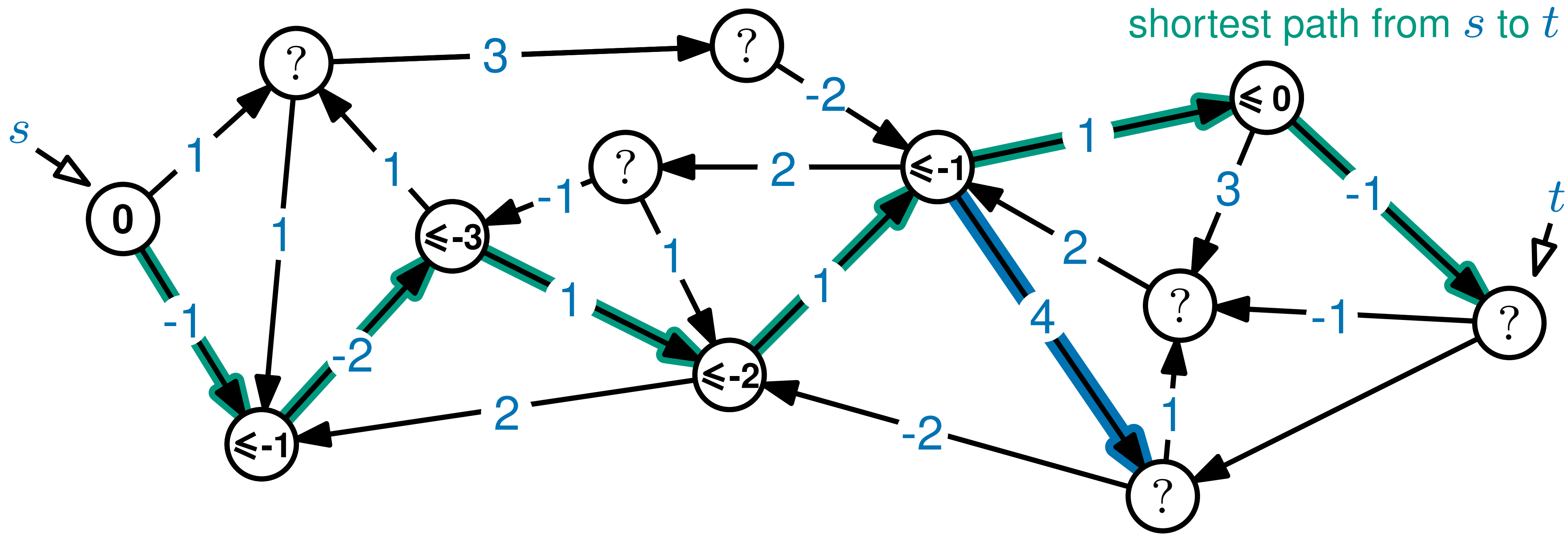
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

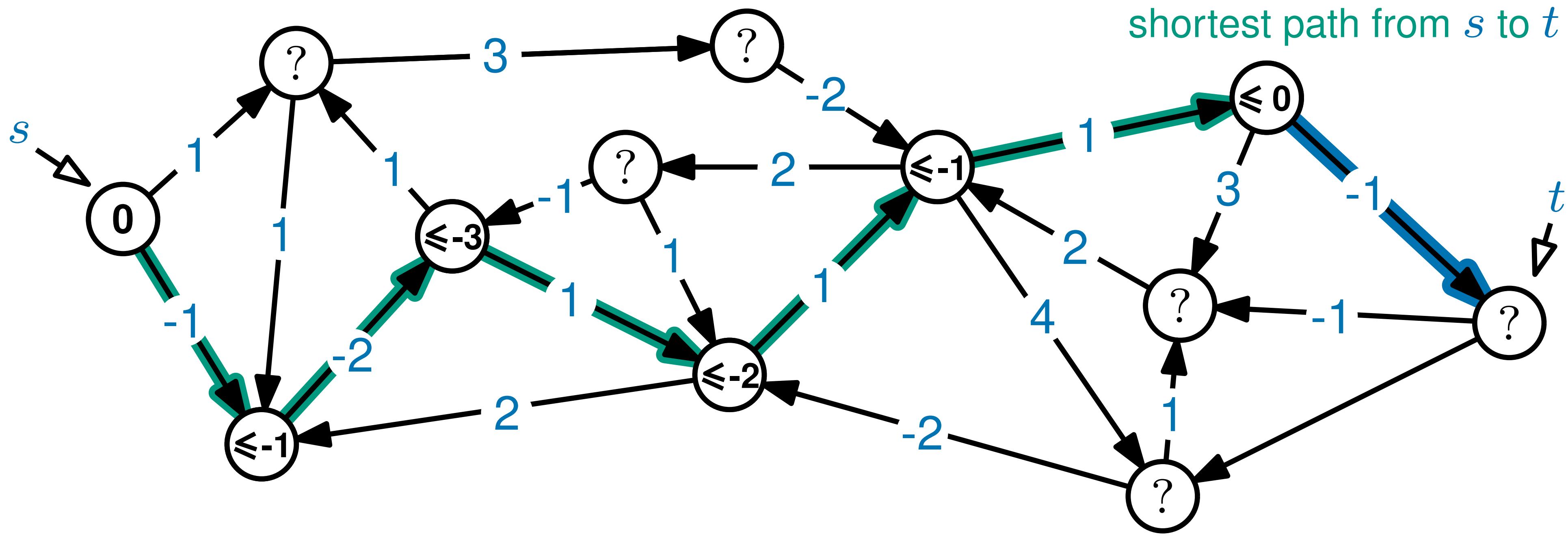
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

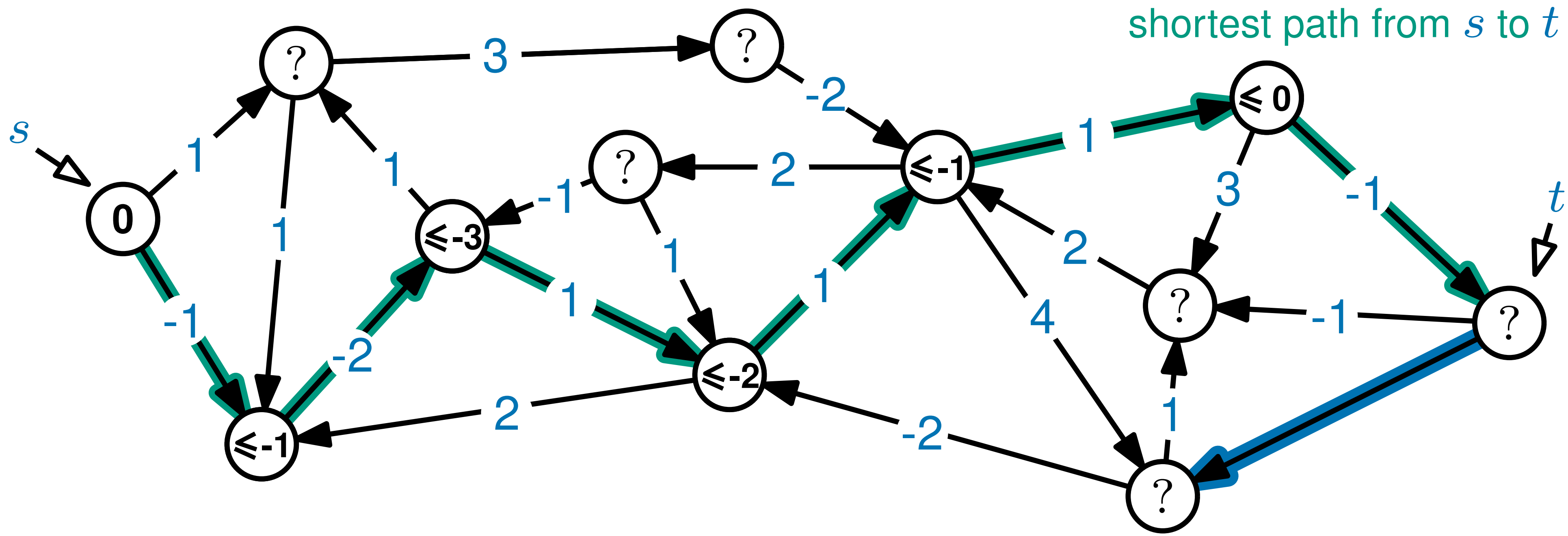
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

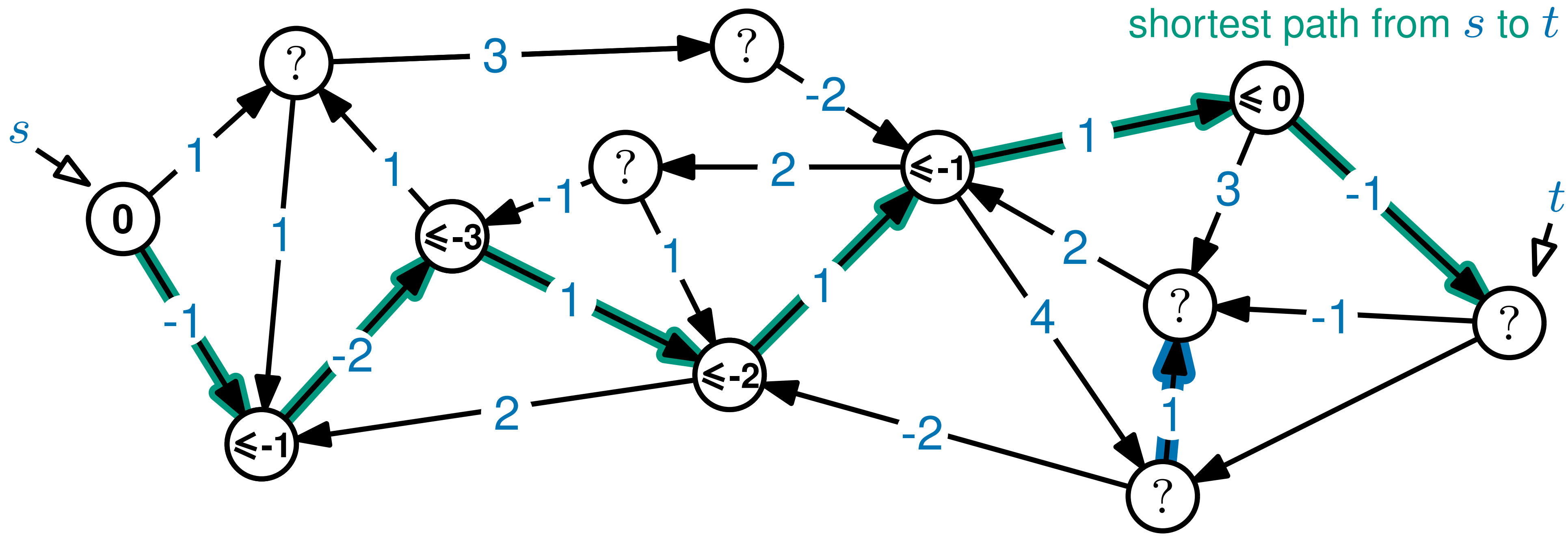
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 5:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

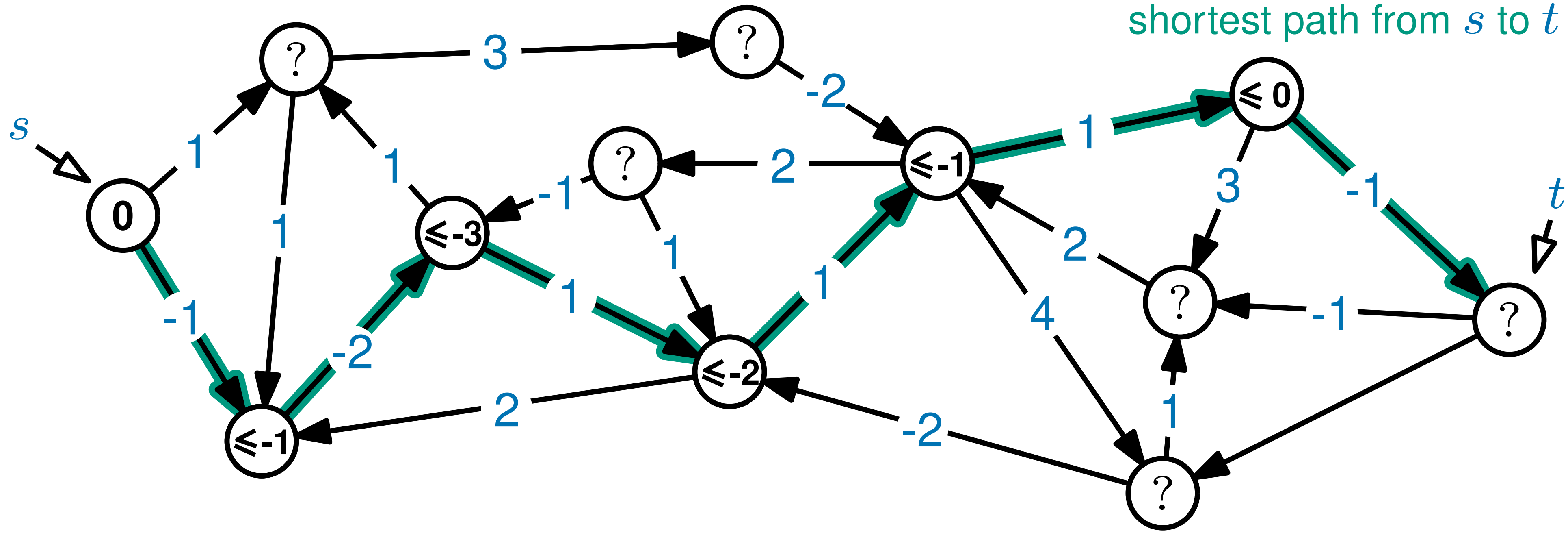
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

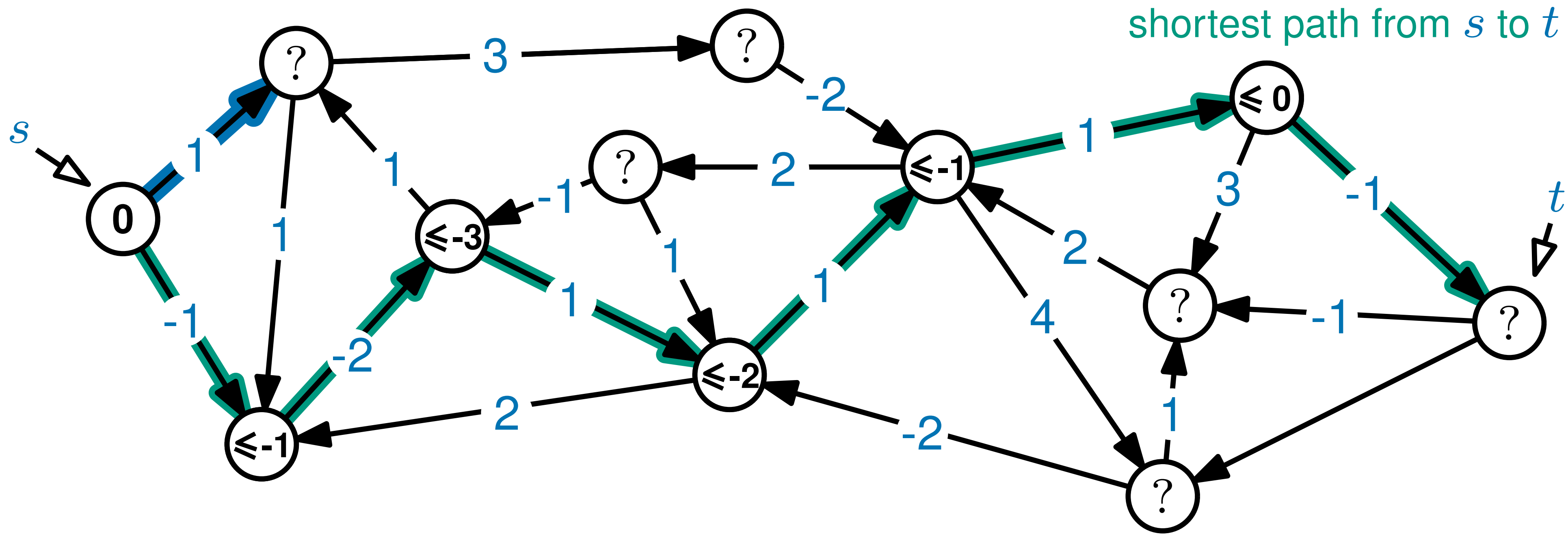
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

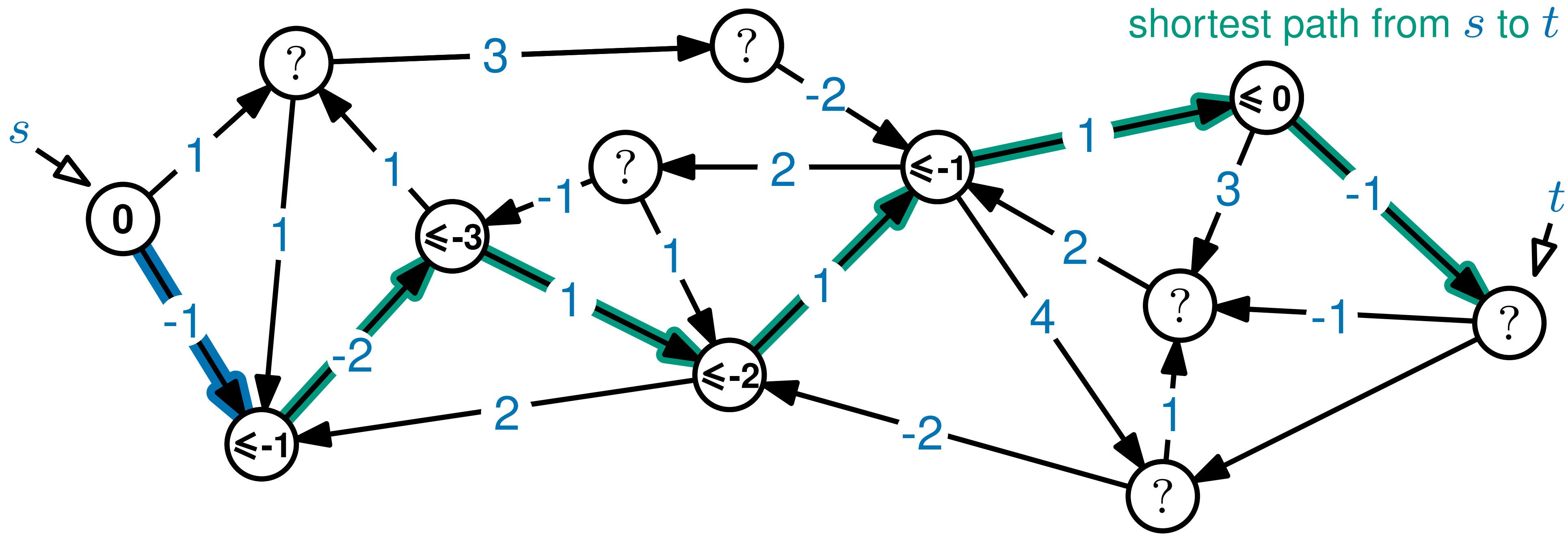
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...
you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



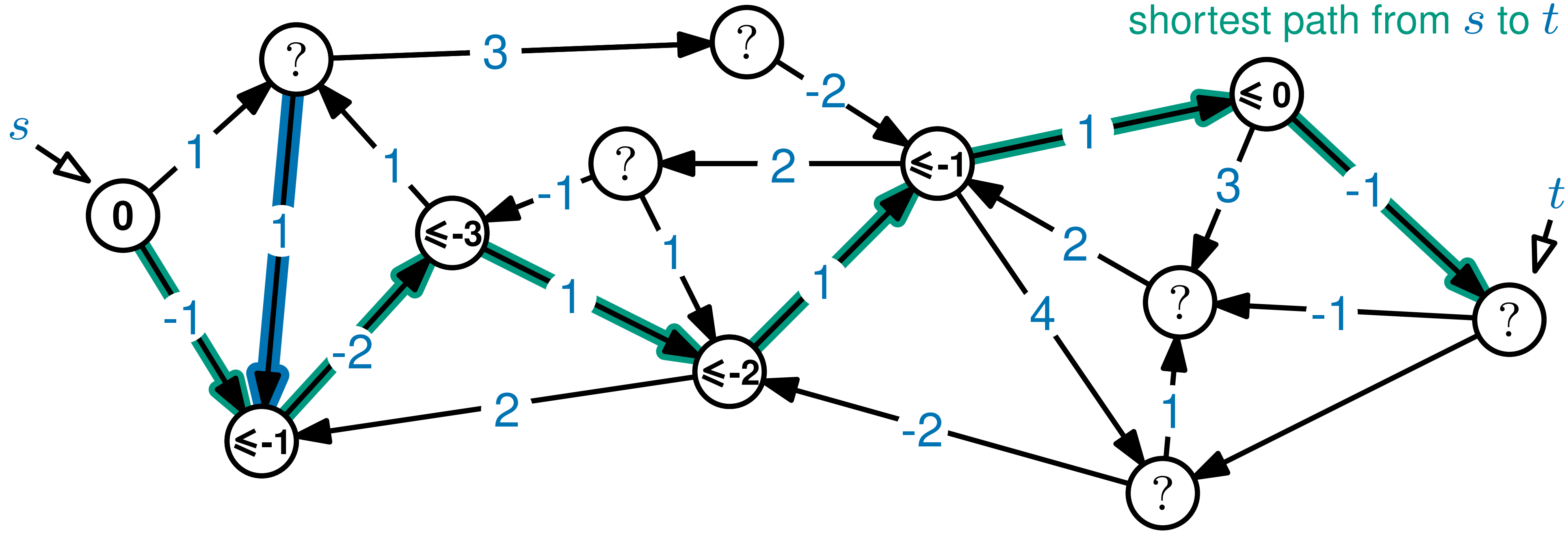
At some point in iteration i
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

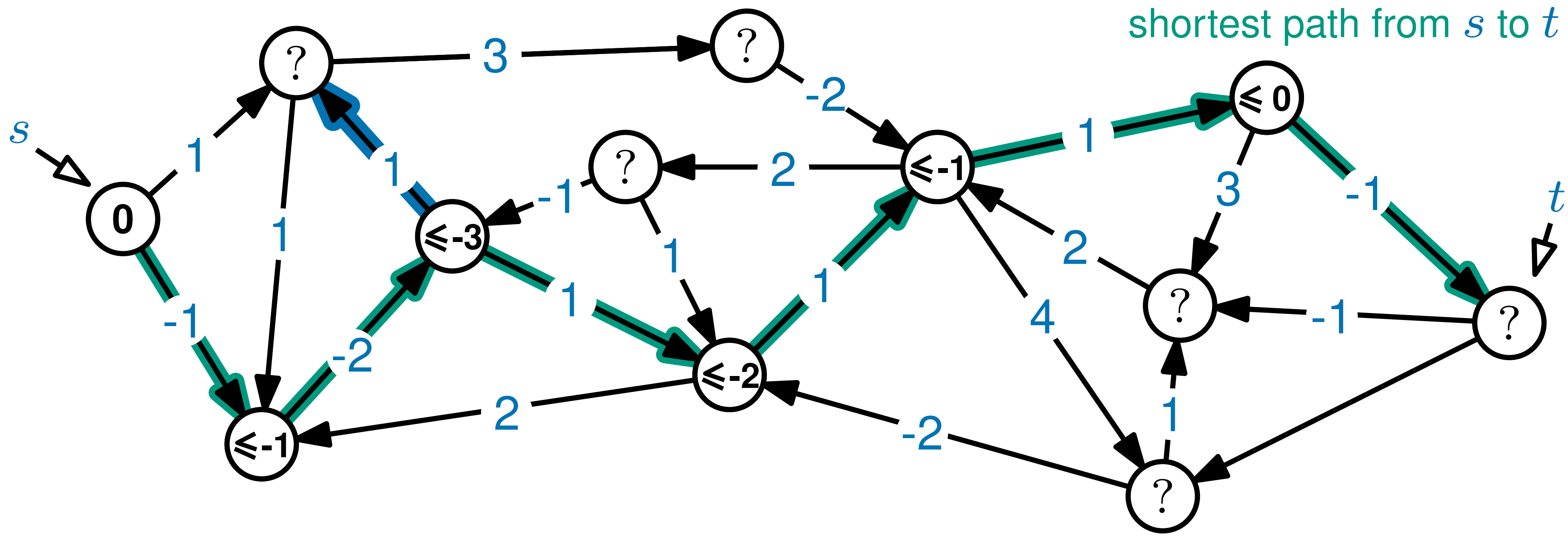
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

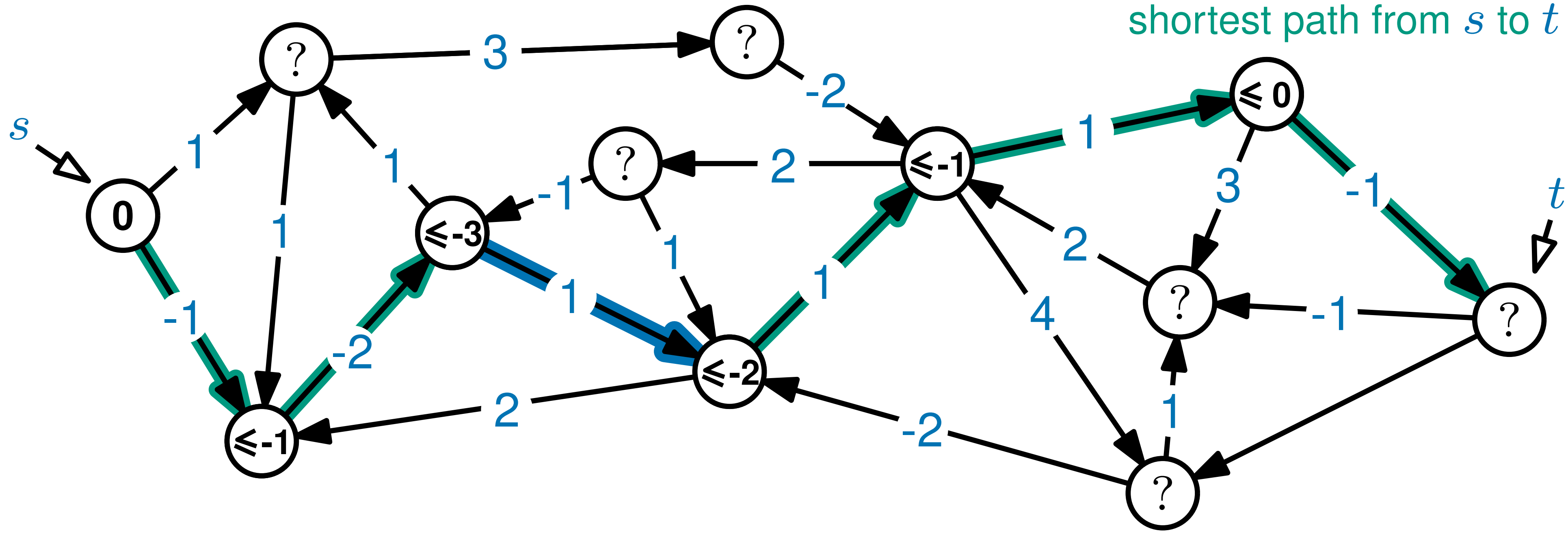
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration…

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t…

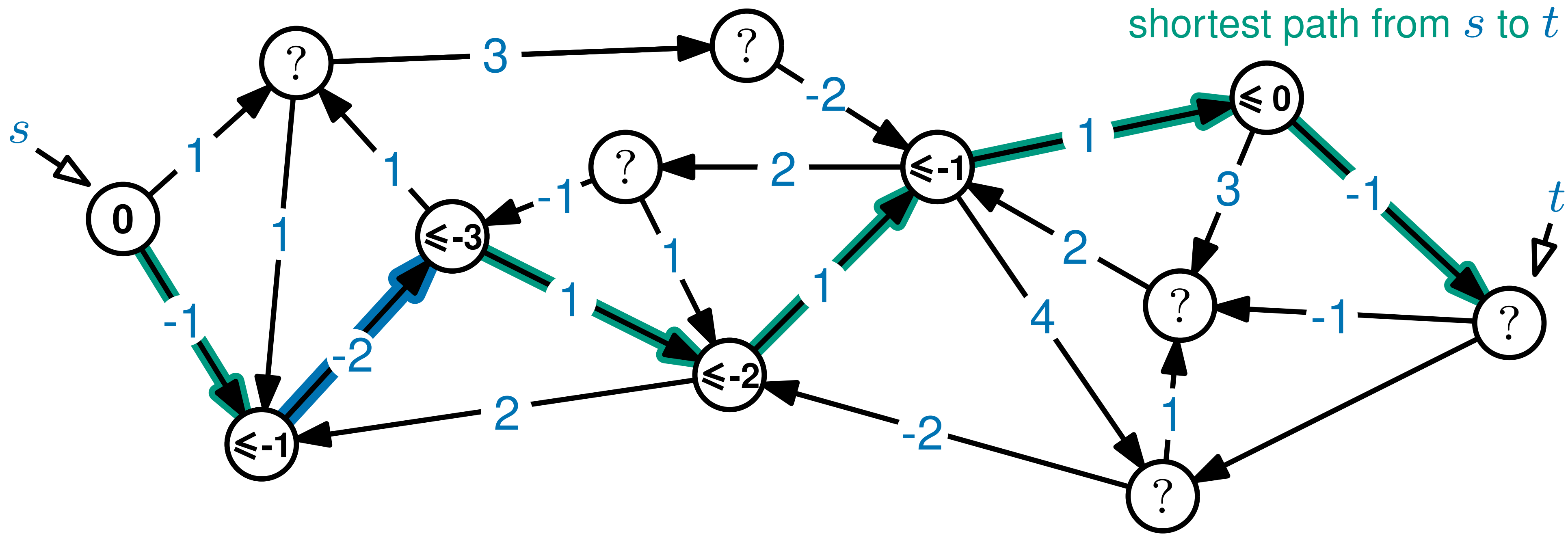
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

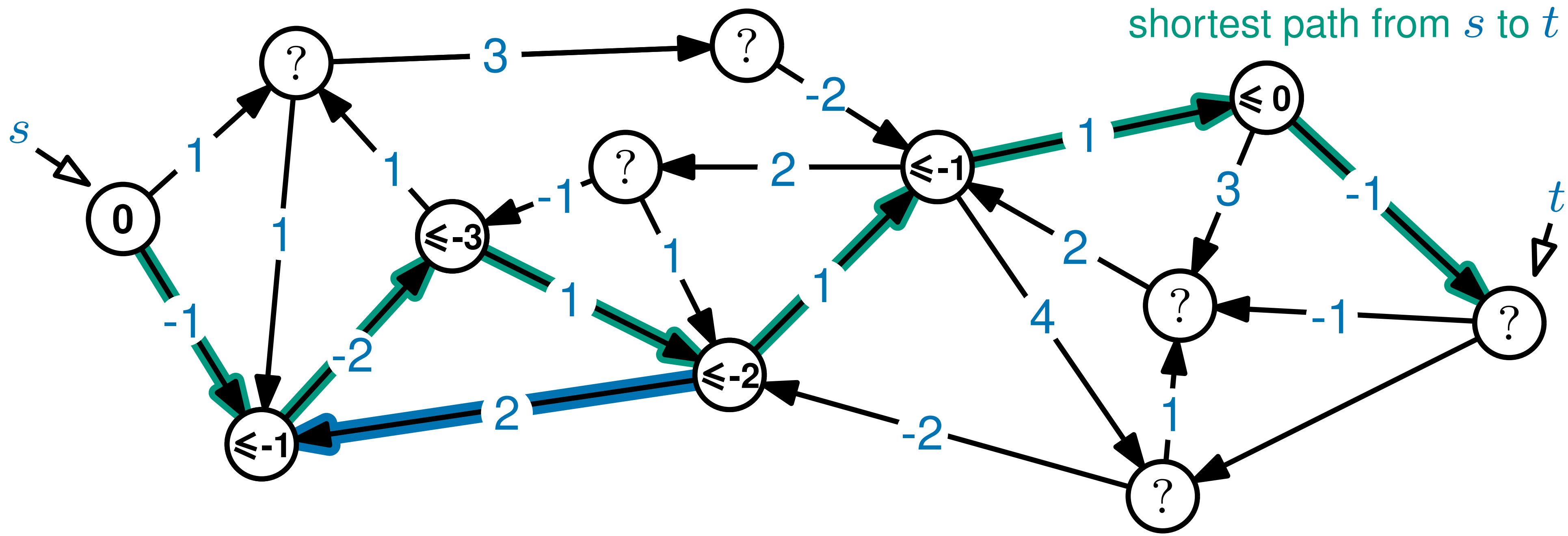
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

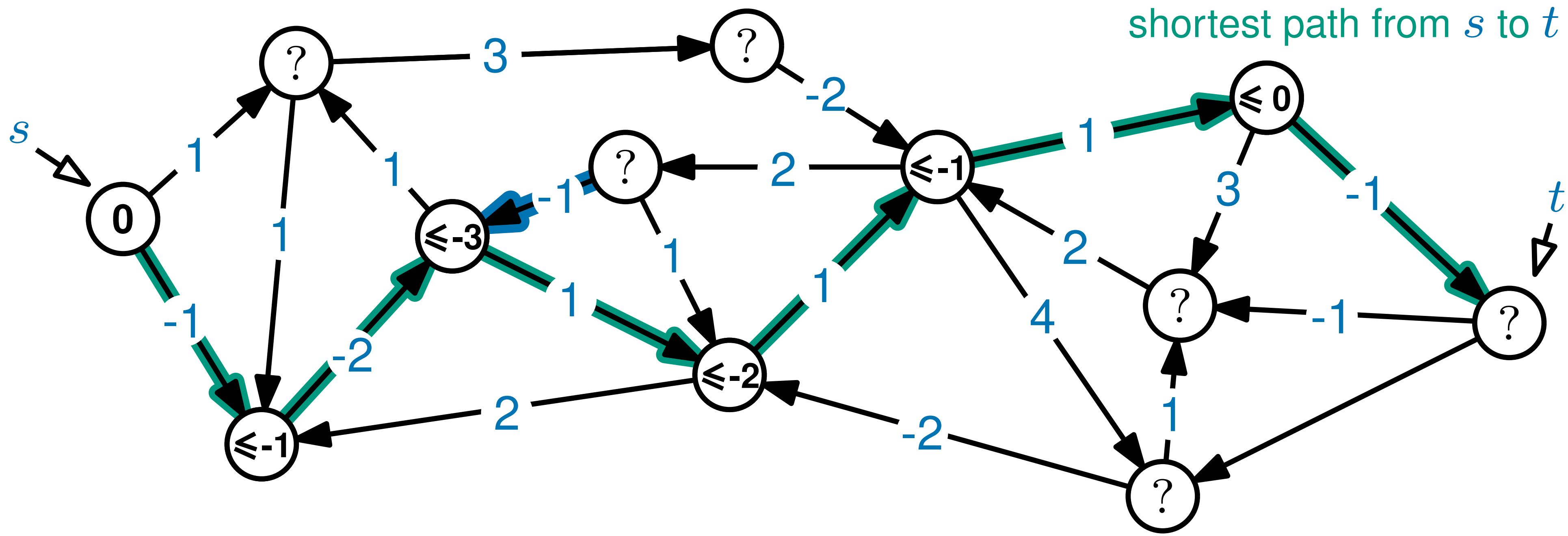
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

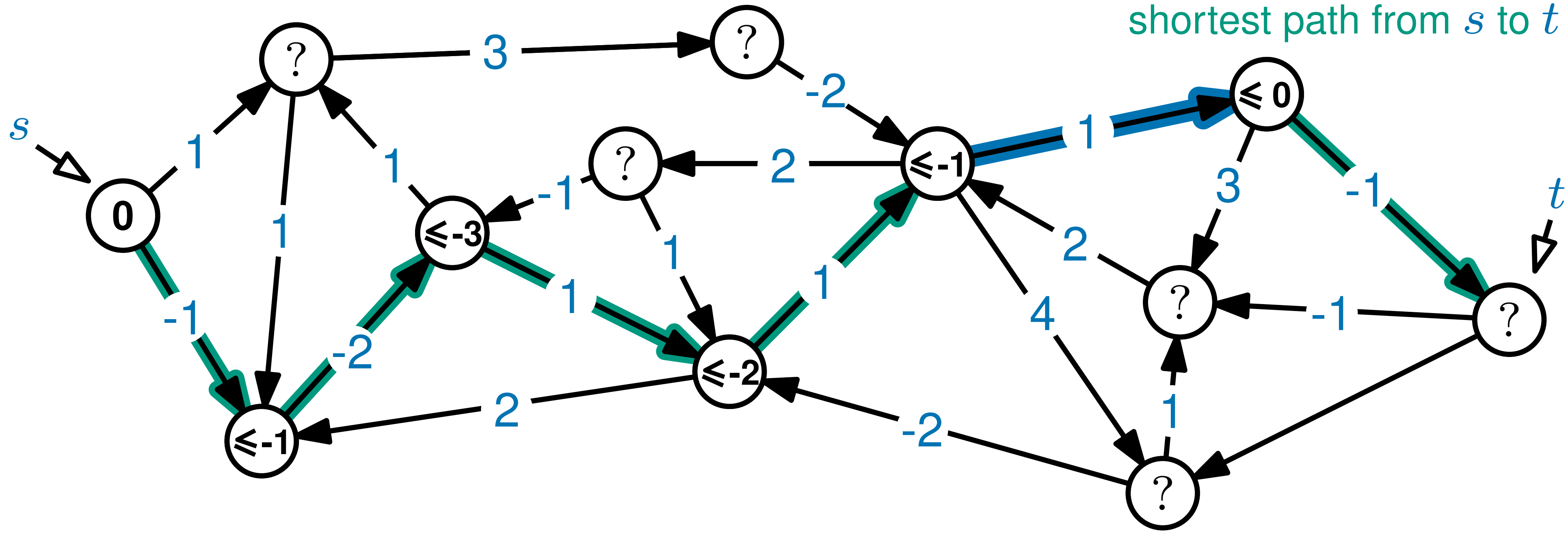
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

University of BRISTOL

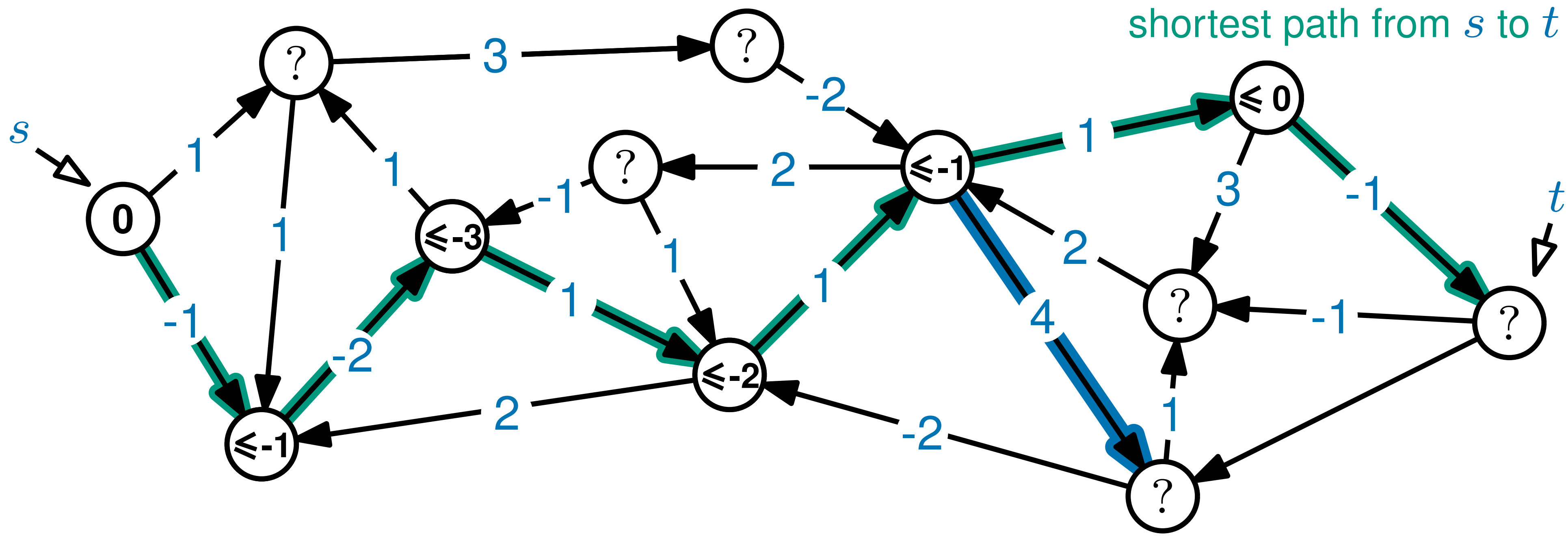
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

The proof idea

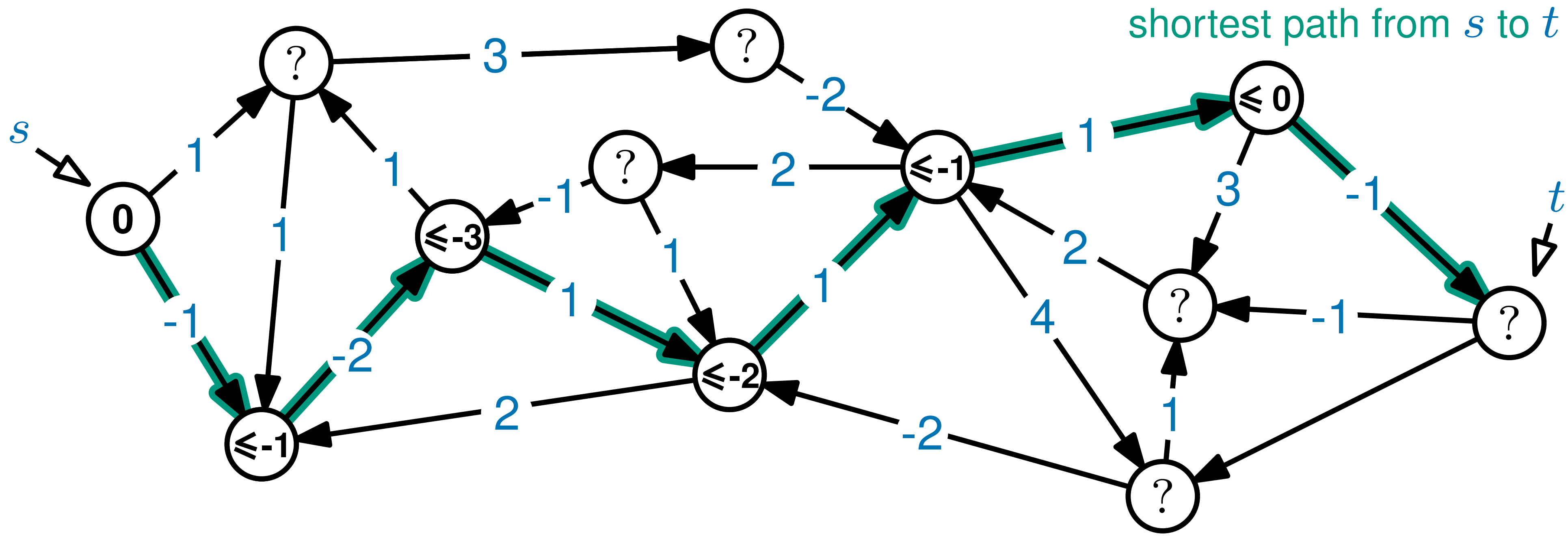
Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same

shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

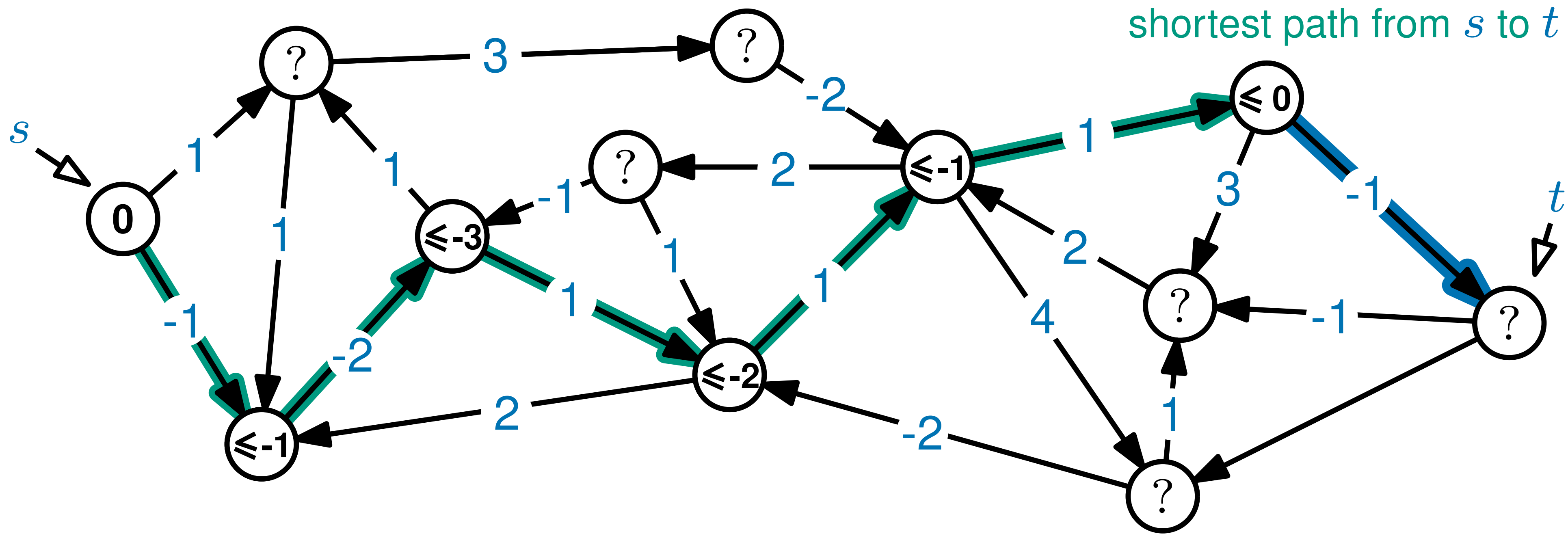
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

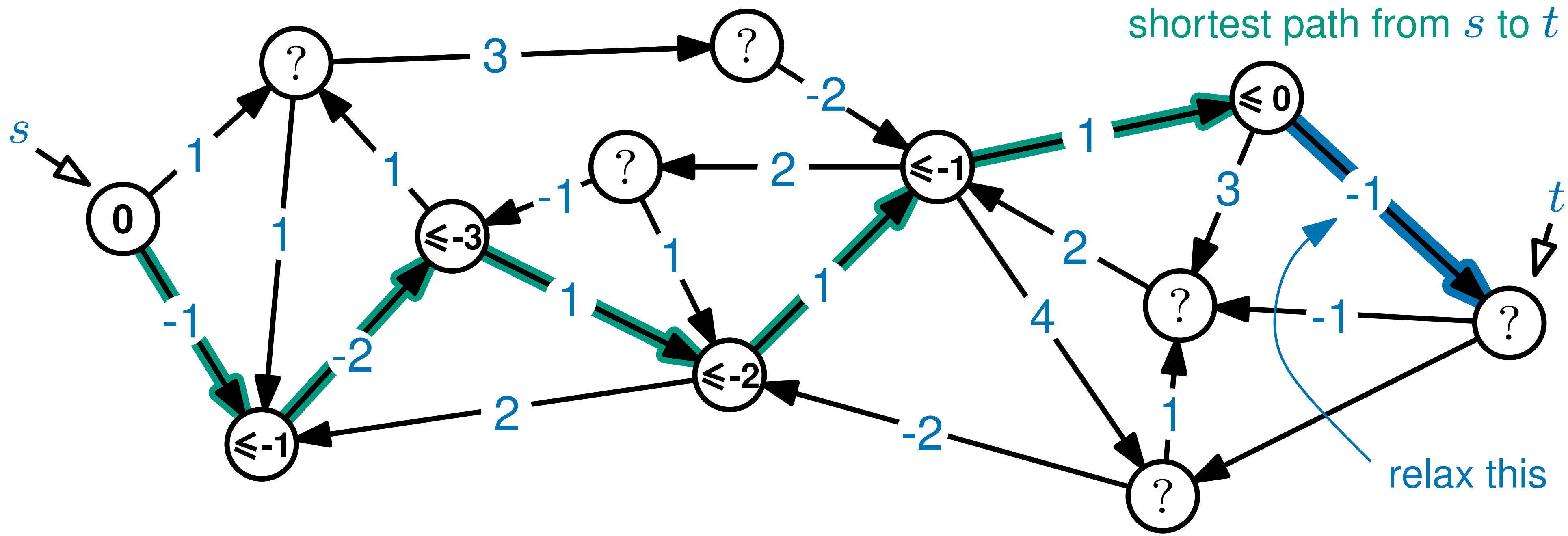
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



At some point in iteration i

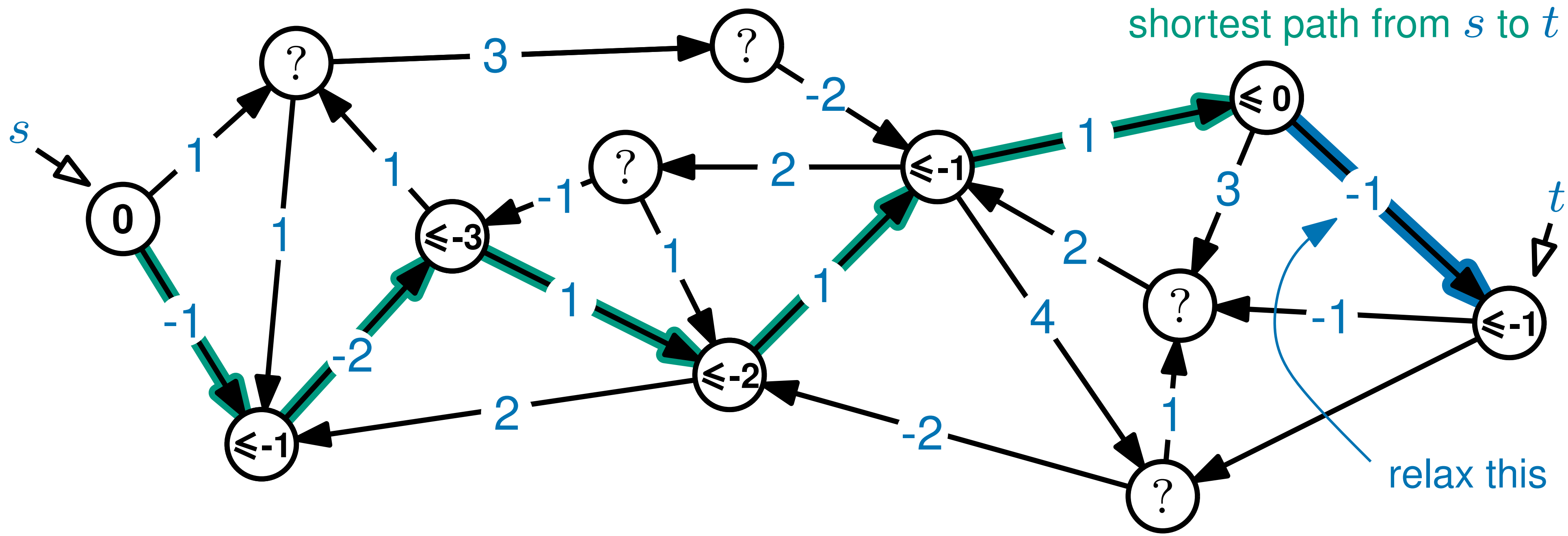
you relax the i-th edge in the shortest path from s to t...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:



At some point in iteration i

you relax the i-th edge in the shortest path from s to t...

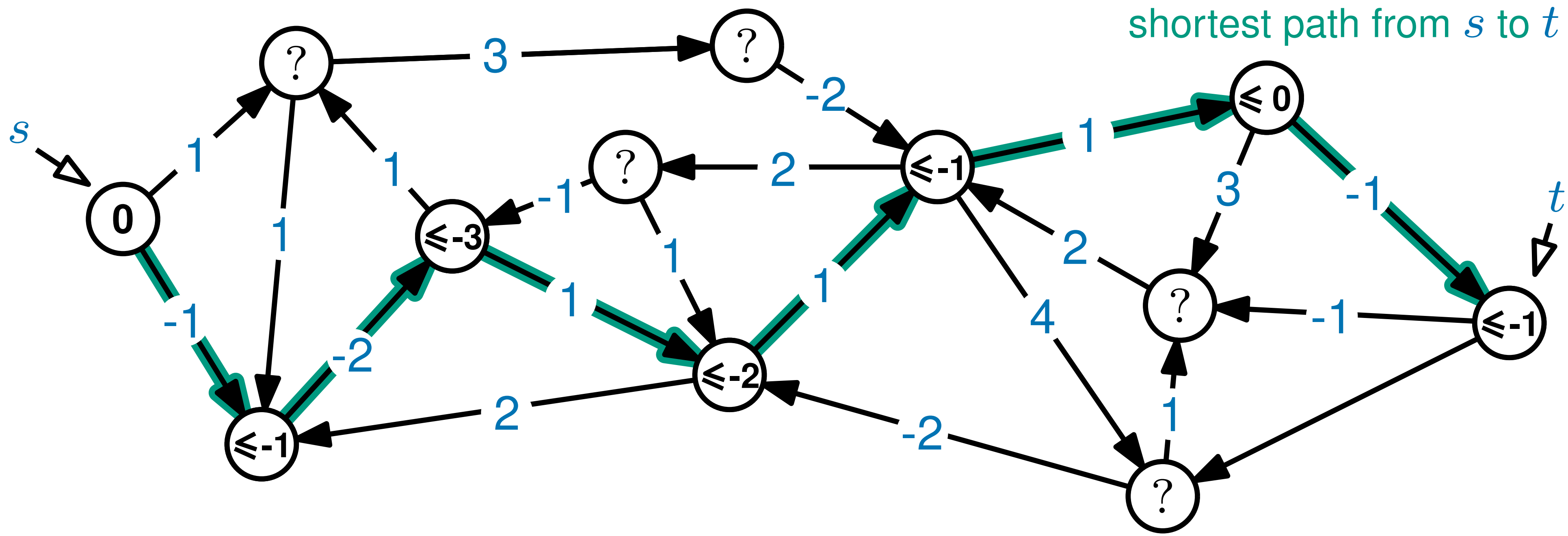
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



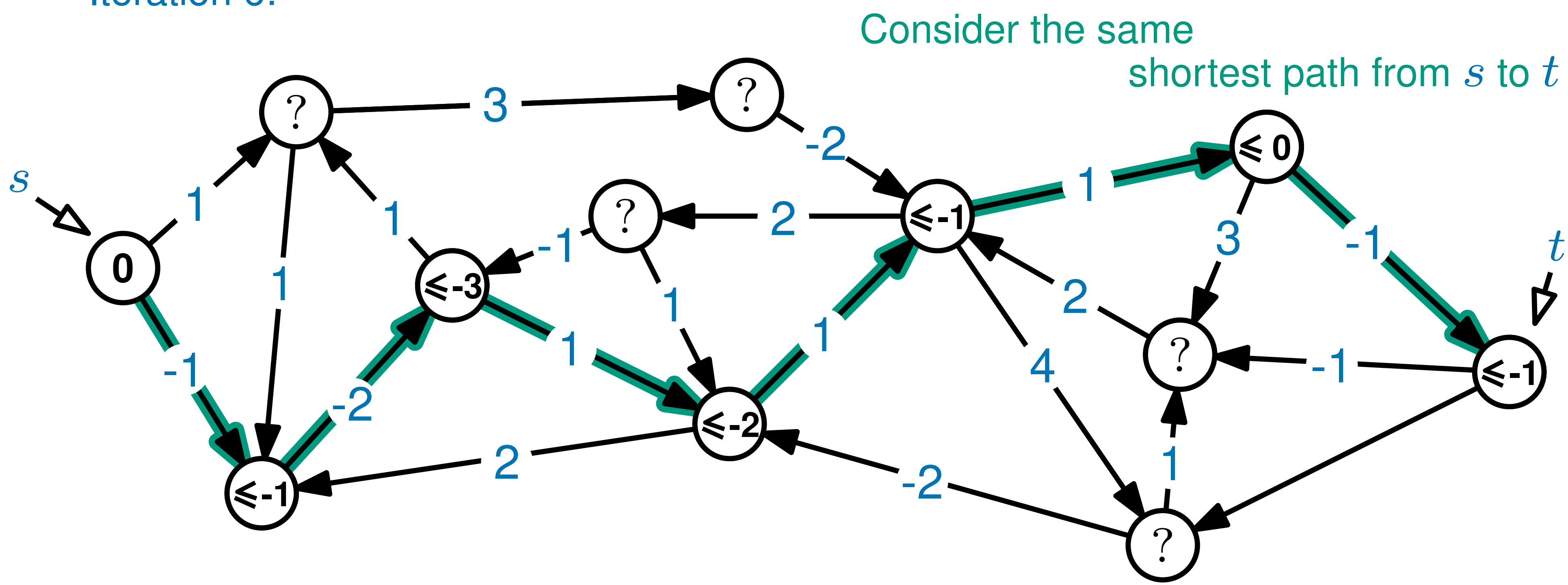
At some point in iteration i

you relax the i-th edge in the shortest path from s to t. . .

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .
you relax every edge *(rather than one edge)*

Iteration 6:

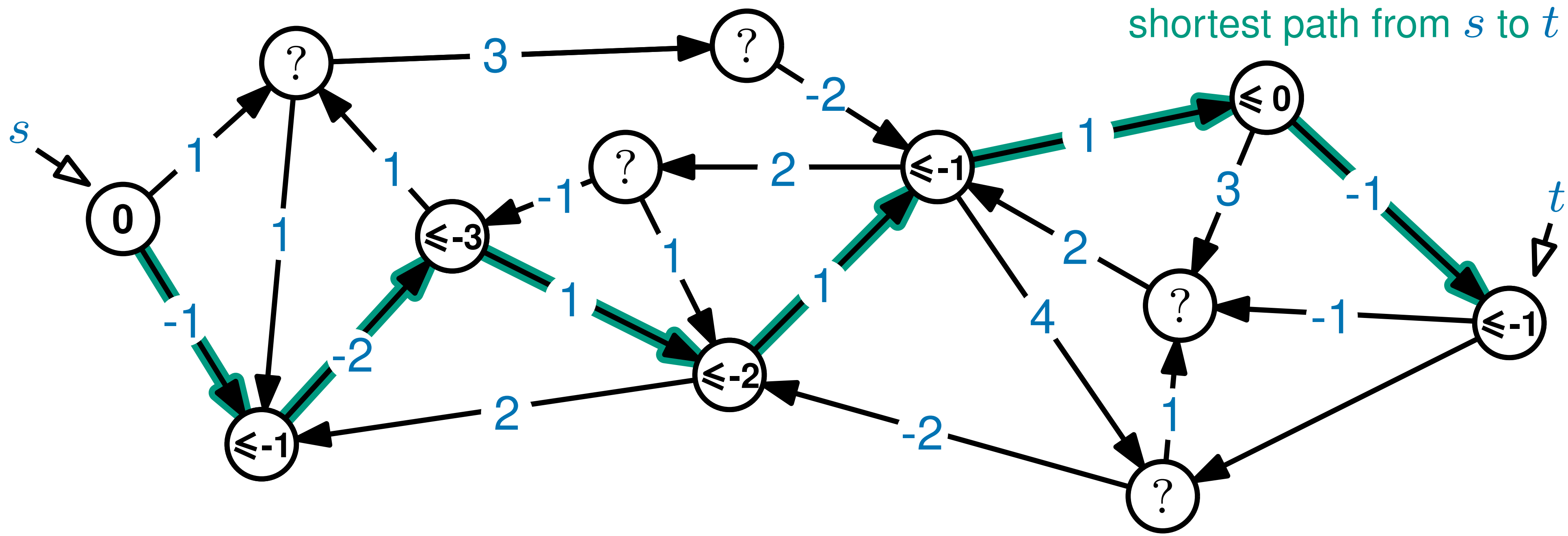


The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:



So after *enough* iterations...

$\texttt{dist}(t)$ is the length of a path from s to t

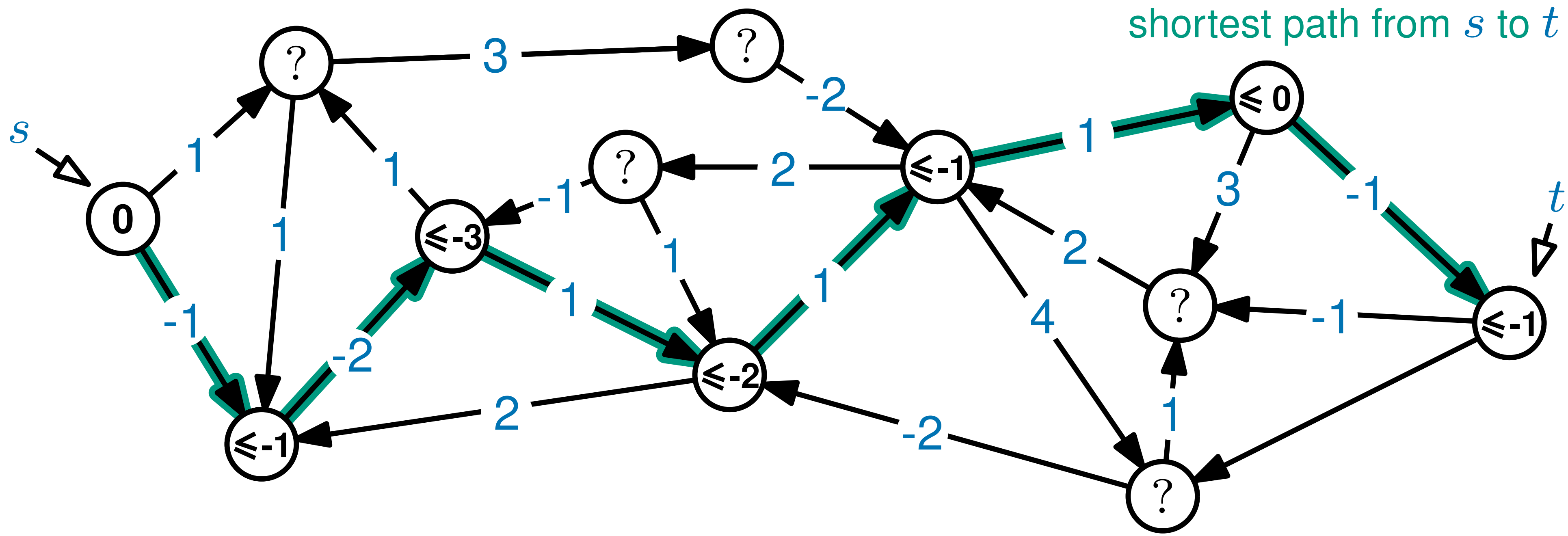
which is at least as short as the shortest path...

The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:



So after *enough* iterations...

$\texttt{dist}(t)$ is the length of a path from s to t

which is at least as short as the shortest path...

In other words $\texttt{dist}(t)$ is the length of a shortest path from s to t

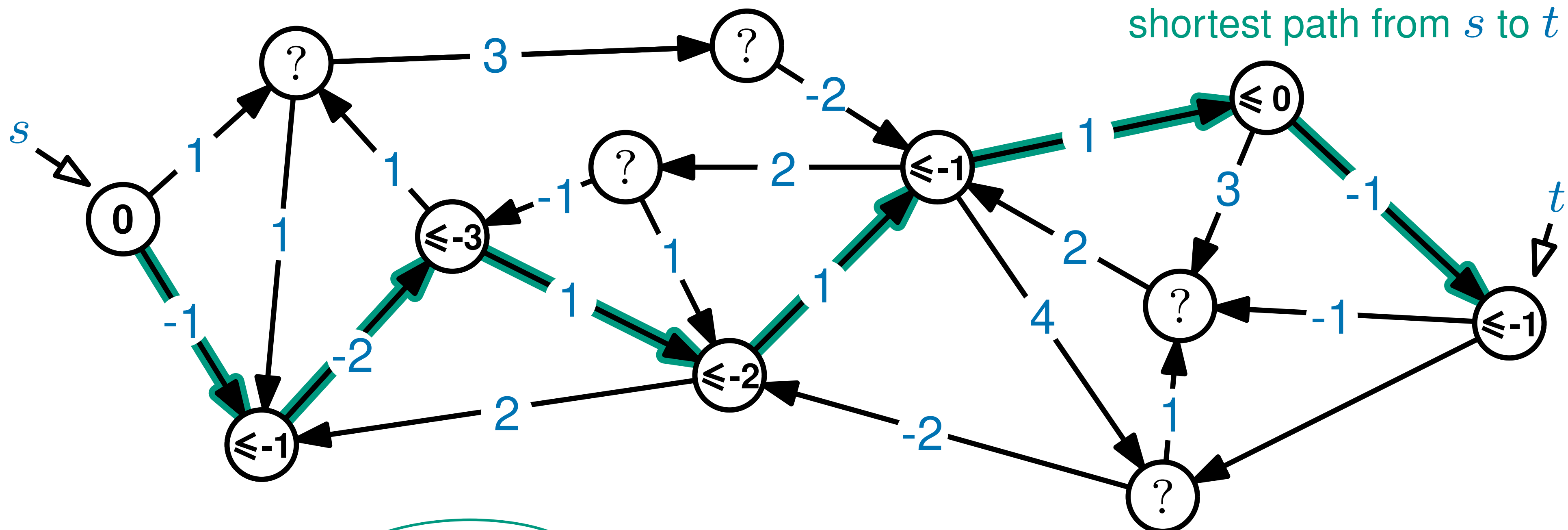
The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



how many iterations are needed?

So after *enough* iterations...

$\texttt{dist}(t)$ is the length of a path from s to t

which is at least as short as the shortest path...

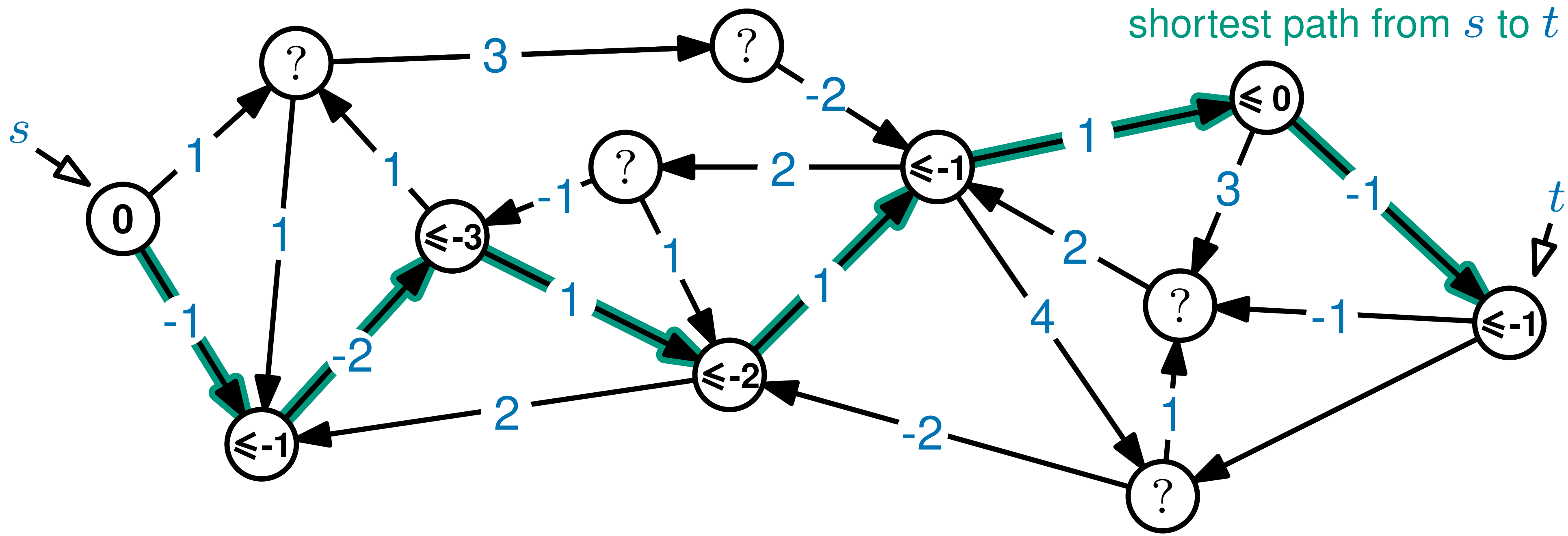
In other words $\texttt{dist}(t)$ is the length of a shortest path from s to t

The proof idea

Now consider the MostOfBellman-Ford algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:



For this argument to hold we need to do one iteration

for every edge in the shortest path from s to t

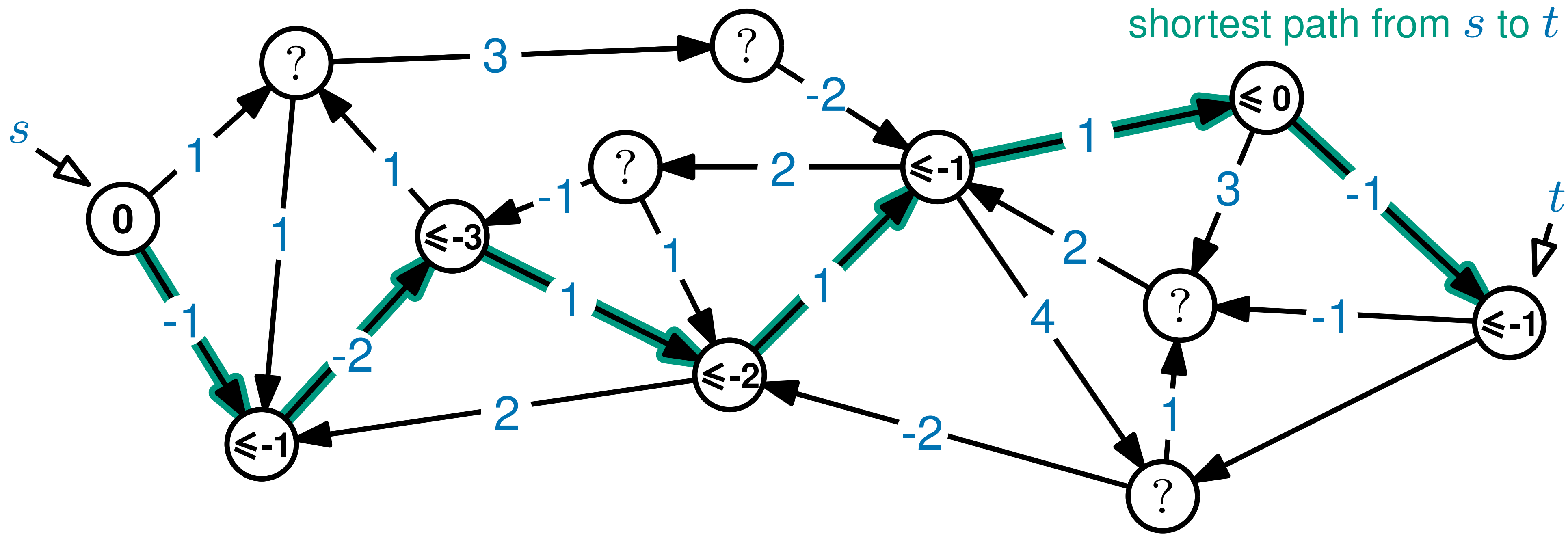
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration. . .

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



For this argument to hold we need to do one iteration for every edge in the shortest path from s to t

We will now prove that *if there are no negative cycles* in the graph,

the shortest path between s and t contains at most $|V|$ edges

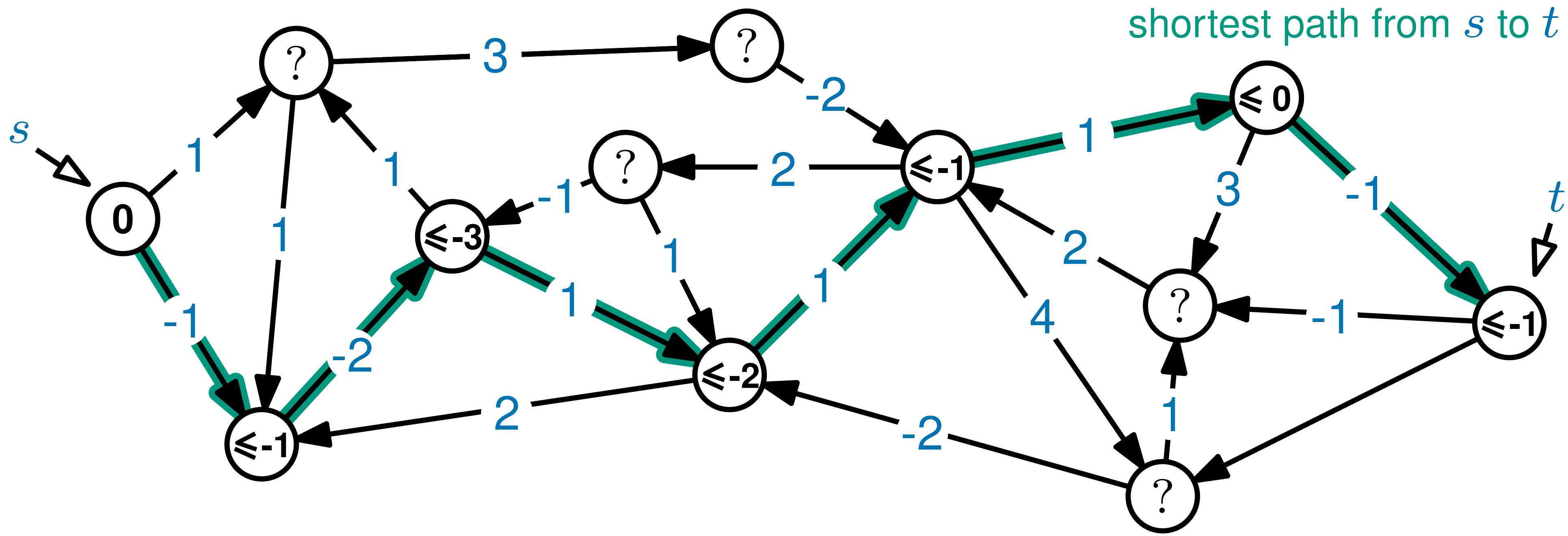
The proof idea

Now consider the MOSTOFBELLMAN-FORD algorithm where in each iteration...

you relax every edge *(rather than one edge)*

Iteration 6:

Consider the same shortest path from s to t



For this argument to hold we need to do one iteration for every edge in the shortest path from s to t

We will now prove that *if there are no negative cycles* in the graph,

the shortest path between s and t contains at most $|V|$ edges

and therefore $|V|$ iterations will be enough

Claim *if there are no negative weight cycles* in the graph,

there is a shortest path between s and t containing at most $|V|$ edges

(or there is no path from s to t)

Claim *if there are no negative weight cycles* in the graph,
there is a shortest path between s and t containing at most $|V|$ edges
(or there is no path from s to t)

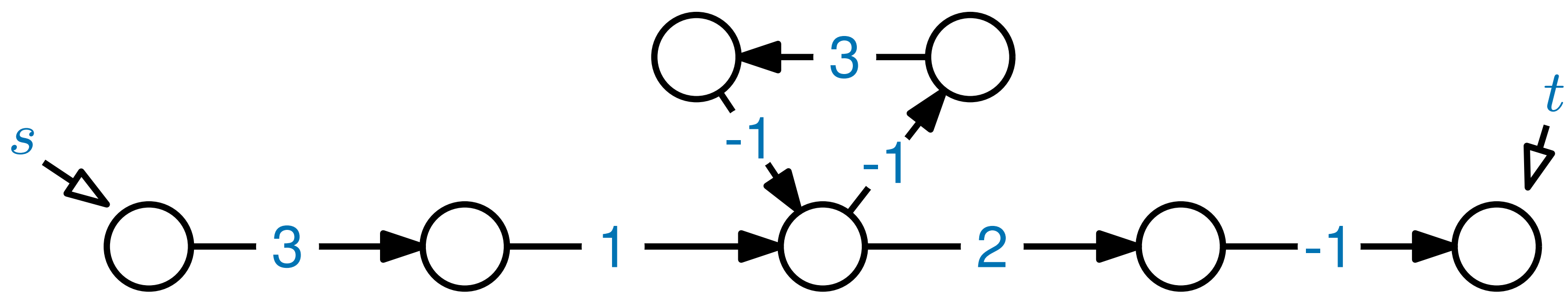
Consider a path between s and t with a cycle. . .

Claim *if there are no negative weight cycles* in the graph,

there is a shortest path between s and t containing at most $|V|$ edges

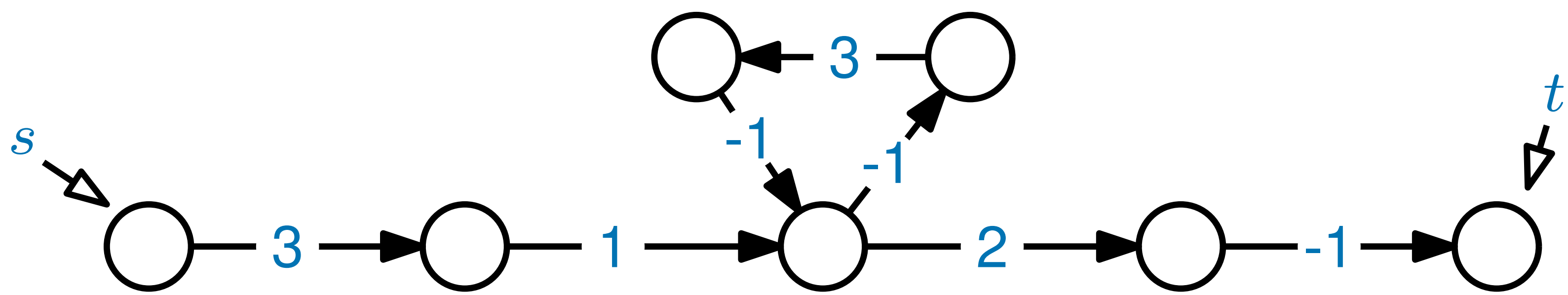
(or there is no path from s to t)

Consider a path between s and t with a cycle...



Claim *if there are no negative weight cycles* in the graph,
there is a shortest path between s and t containing at most $|V|$ edges
(or there is no path from s to t)

Consider a path between s and t with a cycle...

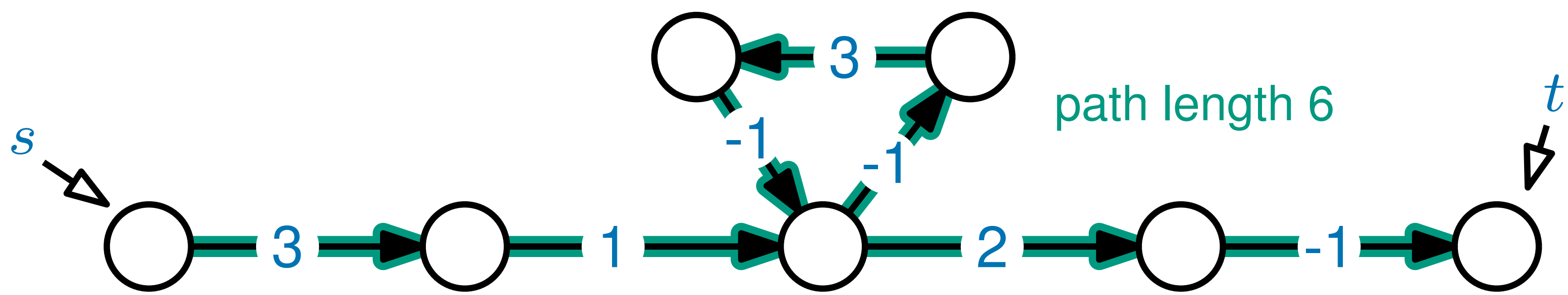


As there are no negative weight cycles

deleting this cycle from the path cannot increase it's length

Claim *if there are no negative weight cycles* in the graph,
there is a shortest path between s and t containing at most $|V|$ edges
(or there is no path from s to t)

Consider a path between s and t with a cycle. . .



As there are no negative weight cycles

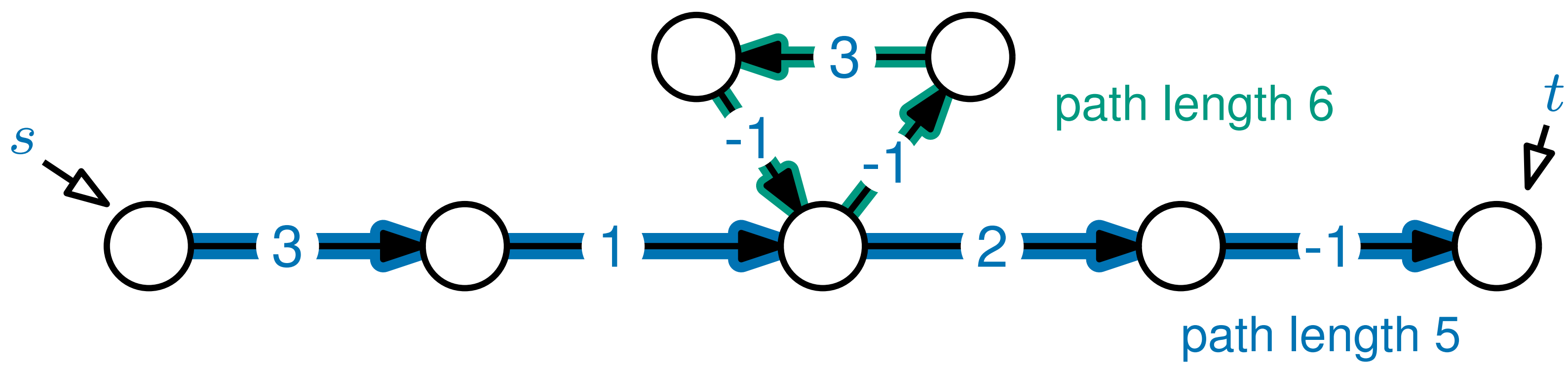
deleting this cycle from the path cannot increase it's length

Claim *if there are no negative weight cycles* in the graph,

there is a shortest path between s and t containing at most $|V|$ edges

(or there is no path from s to t)

Consider a path between s and t with a cycle. . .



As there are no negative weight cycles

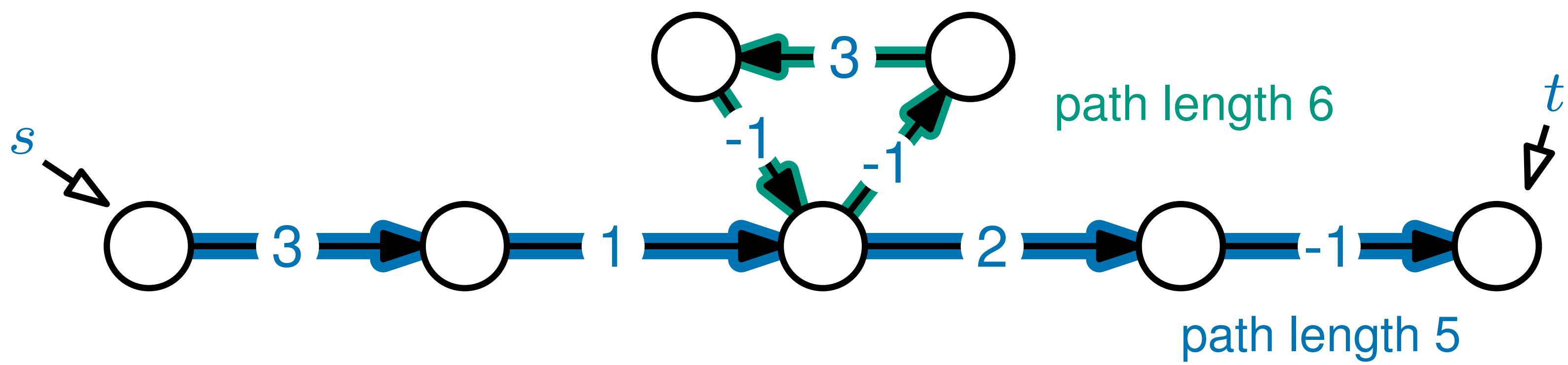
deleting this cycle from the path cannot increase it's length

Claim *if there are no negative weight cycles* in the graph,

there is a shortest path between s and t containing at most $|V|$ edges

(or there is no path from s to t)

Consider a path between s and t with a cycle. . .



As there are no negative weight cycles

deleting this cycle from the path cannot increase it's length

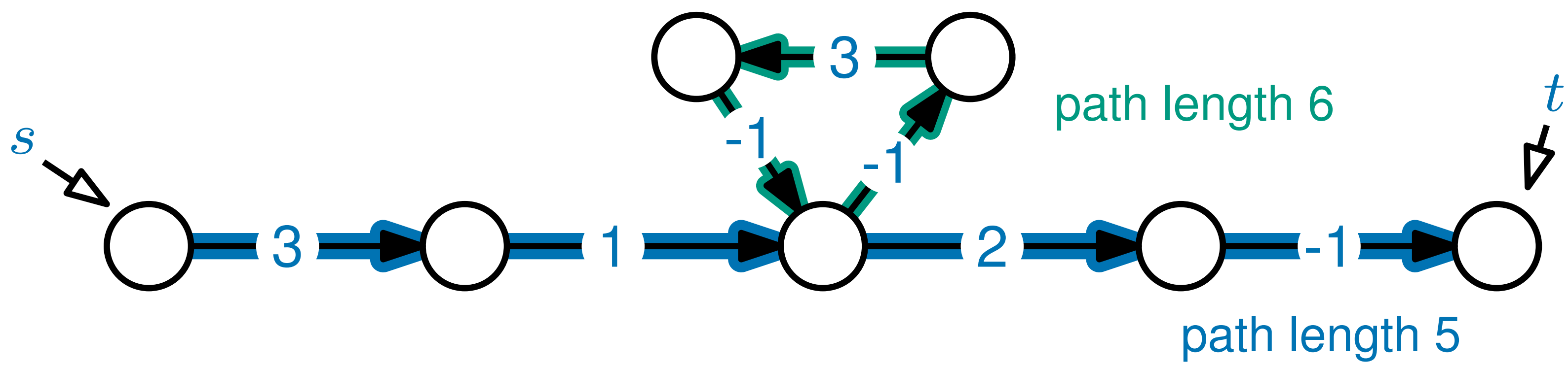
Therefore, there is a shortest path between s and t containing no cycles

Claim *if there are no negative weight cycles* in the graph,

there is a shortest path between s and t containing at most $|V|$ edges

(or there is no path from s to t)

Consider a path between s and t with a cycle. . .



As there are no negative weight cycles

deleting this cycle from the path cannot increase it's length

Therefore, there is a shortest path between s and t containing no cycles

A path with no cycles enters each vertex at most once

so contains at most $|V|$ edges

MostOfBellman-Ford Summary

Claim When the MostOfBellman-Ford algorithm terminates,

for each vertex v, $\mathtt{dist}(v)$ is the distance between s and v

(assuming there are no-negative weight cycles)

MostOfBellman-Ford Summary

Claim When the MostOfBellman-Ford algorithm terminates,

for each vertex v, $\texttt{dist}(v)$ is the distance between s and v

(assuming there are no-negative weight cycles)

If there is no path between s and v then

at termination, $\texttt{dist}(v) = \infty$

by the algorithm description, MostOfBellman-Ford
doesn't find paths that aren't there

MostOfBellman-Ford Summary

Claim When the MostOfBellman-Ford algorithm terminates,

for each vertex v, $\texttt{dist}(v)$ is the distance between s and v

(assuming there are no-negative weight cycles)

If there is no path between s and v then

at termination, $\texttt{dist}(v) = \infty$

by the algorithm description, MostOfBellman-Ford *doesn't find paths that aren't there*

If there is a shortest path between s and v then

it contains at most $|V|$ edges

(assuming there are no-negative weight cycles)

MostOfBellman-Ford Summary

Claim When the MostOfBellman-Ford algorithm terminates,

for each vertex v, $\mathtt{dist}(v)$ is the distance between s and v

(assuming there are no-negative weight cycles)

If there is no path between s and v then

at termination, $\mathtt{dist}(v) = \infty$

by the algorithm description, MostOfBellman-Ford

doesn't find paths that aren't there

If there is a shortest path between s and v then

it contains at most $|V|$ edges

(assuming there are no-negative weight cycles)

After each iteration, the algorithm

has taken (at least) one more 'step' along this path

MostOfBellman-Ford Summary

Claim When the MostOfBellman-Ford algorithm terminates,

for each vertex v, $\texttt{dist}(v)$ is the distance between s and v

(assuming there are no-negative weight cycles)

If there is no path between s and v then

at termination, $\texttt{dist}(v) = \infty$

by the algorithm description, MostOfBellman-Ford *doesn't find paths that aren't there*

If there is a shortest path between s and v then
it contains at most $|V|$ edges
(assuming there are no-negative weight cycles)

After each iteration, the algorithm
has taken (at least) one more 'step' along this path

After $|V|$ iterations of the algorithm
$\texttt{dist}(v)$ is the length of the shortest path between s and v

So what is the rest of the algorithm?

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

So what is the rest of the algorithm?

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

We've added a final check which determines whether another iteration would find an even shorter path from s to some v

So what is the rest of the algorithm?

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output ‘Negative weight cycle found’
```

We’ve added a final check which determines whether another iteration would find an even shorter path from s to some v

We will prove that this is happens if and only if there is a negative weight cycle

Bellman-Ford without negative weight cycles

We first argue that the algorithm still works when there is no negative weight cycle

Bellman-Ford(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

BELLMAN-FORD without negative weight cycles

We first argue that the algorithm still works when there is no negative weight cycle

```
BELLMAN-FORD(s)
```

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
   For every edge (u, v) ∈ E
      if dist(v) > dist(u) + weight(u, v)
         dist(v) = dist(u) + weight(u, v)
-------------------------------------------
For every edge (u, v) ∈ E
   if dist(v) > dist(u) + weight(u, v)
      Output 'Negative weight cycle found'
```

After the first $|V|$ iterations (which haven't changed),

for each vertex v, $\texttt{dist}(v)$ is the length of the shortest path between s and v

BELLMAN-FORD without negative weight cycles

We first argue that the algorithm still works when there is no negative weight cycle

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)
-----------------------------------------------
For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

After the first $|V|$ iterations (which haven't changed),

for each vertex v, $\texttt{dist}(v)$ is the length of the shortest path between s and v

If the final check outputs: 'Negative weight cycle found'

then there is a path from s to some v (via u) with length $\texttt{dist}(u) + \texttt{weight}(u, v) < \texttt{dist}(v)$

BELLMAN-FORD without negative weight cycles

We first argue that the algorithm still works when there is no negative weight cycle

```
BELLMAN-FORD(s)
```

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)
-----------------------------------------
For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

After the first $|V|$ iterations (which haven't changed),

for each vertex v,

$\mathtt{dist}(v)$ is the length

of the shortest path between s and v

If the final check outputs: `'Negative weight cycle found'`

then there is a path from s to some v (via u)

with length $\mathtt{dist}(u) + \mathtt{weight}(u, v) < \mathtt{dist}(v)$

i.e. a path which is shorter than the shortest path

BELLMAN-FORD without negative weight cycles

We first argue that the algorithm still works when there is no negative weight cycle

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)
-----------------------------------------
For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

After the first $|V|$ iterations (which haven't changed),

for each vertex v, $\texttt{dist}(v)$ is the length of the shortest path between s and v

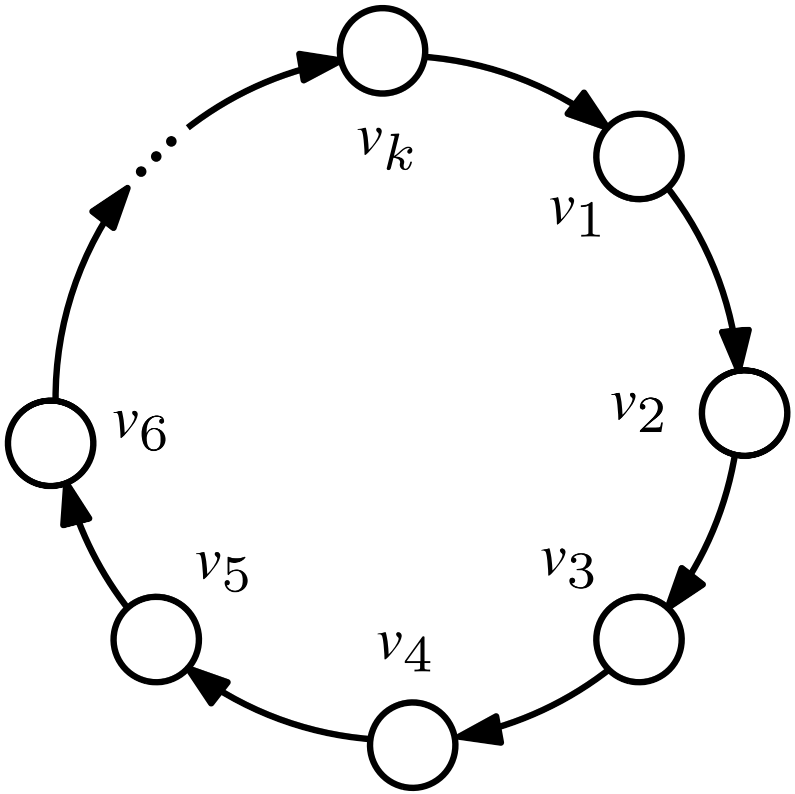
If the final check outputs: 'Negative weight cycle found'

then there is a path from s to some v (via u) with length $\texttt{dist}(u) + \texttt{weight}(u, v) < \texttt{dist}(v)$

i.e. a path which is shorter than the shortest path *this is a contradiction.*

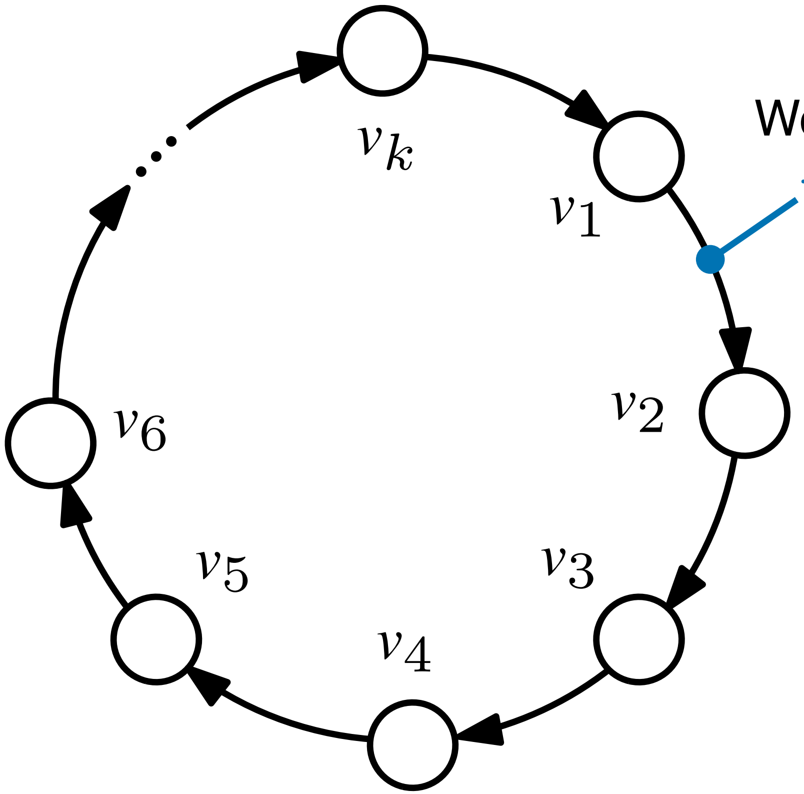
BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



Bellman-Ford *with* a negative weight cycle

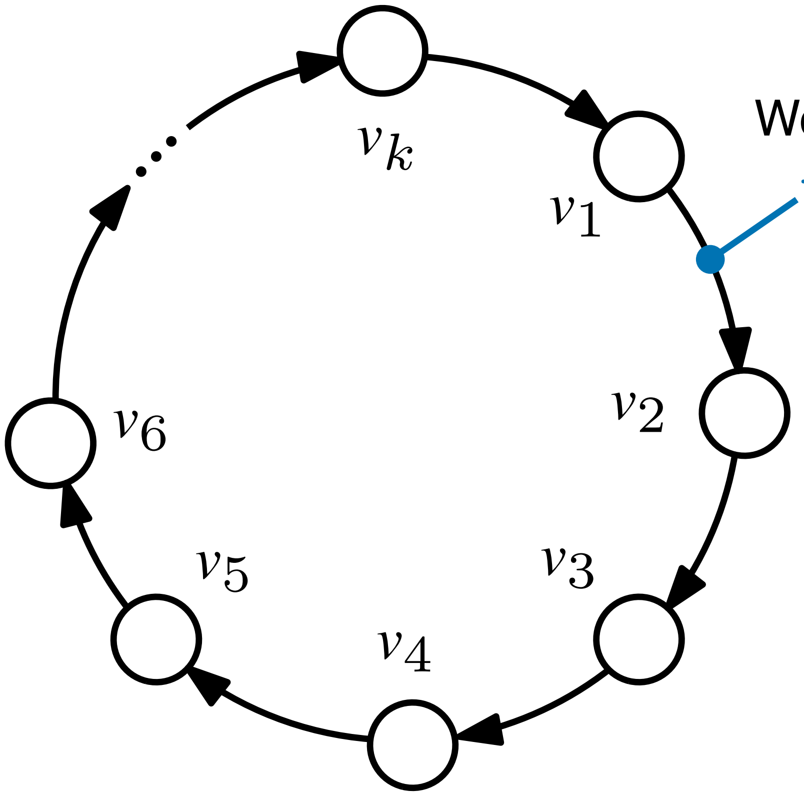
Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,
$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$

Bellman-Ford *with* a negative weight cycle

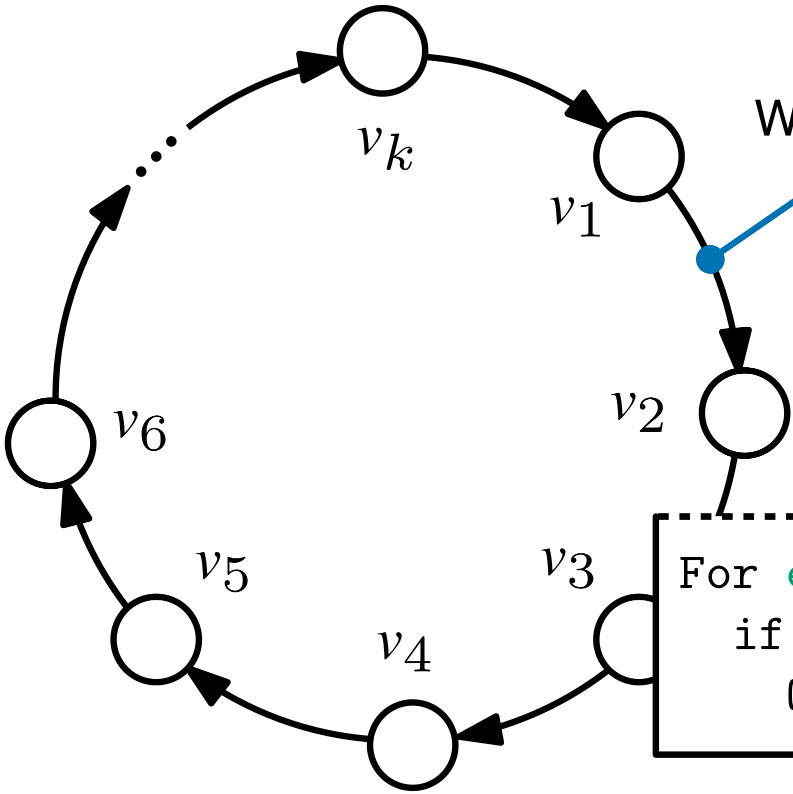
Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,
$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

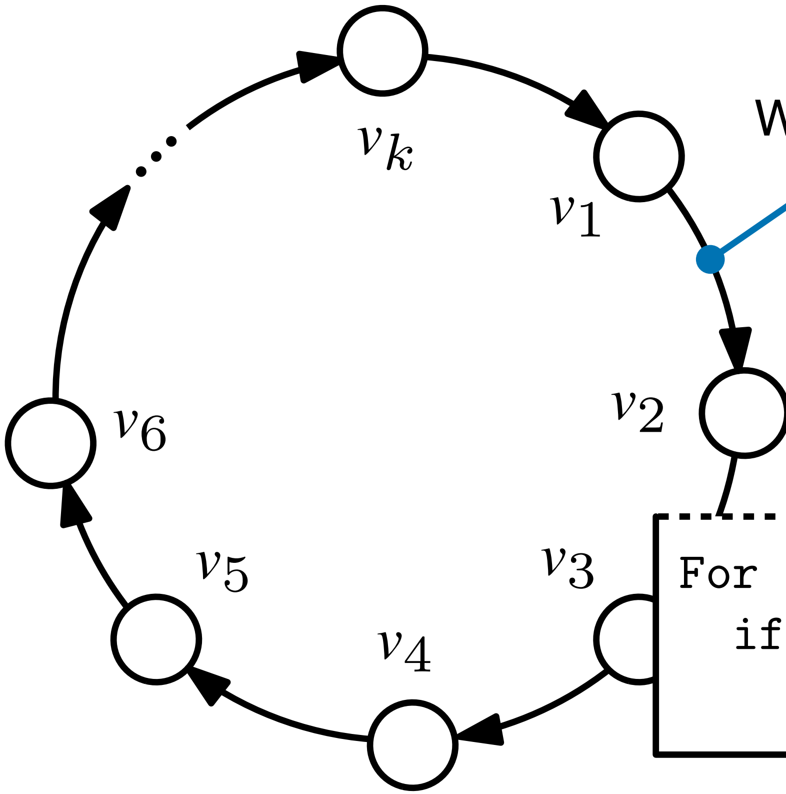
$$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$$

because we ran the following checks
but found no negative weight cycle:

For every edge $(u, v) \in E$
  if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
    Output 'Negative weight cycle found'

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$$

because we ran the following checks
but found no negative weight cycle:

```
For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

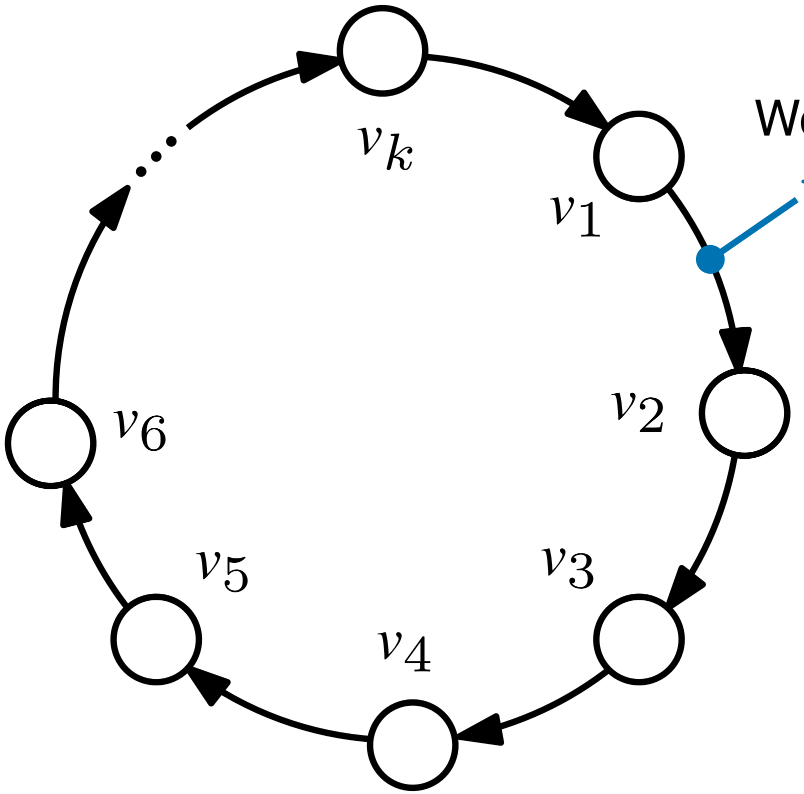
So,

$$\texttt{dist}(v_2) \leqslant \texttt{dist}(v_1) + \texttt{weight}(v_1, v_2)$$

(then rearrange)

Bellman-Ford *with* a negative weight cycle

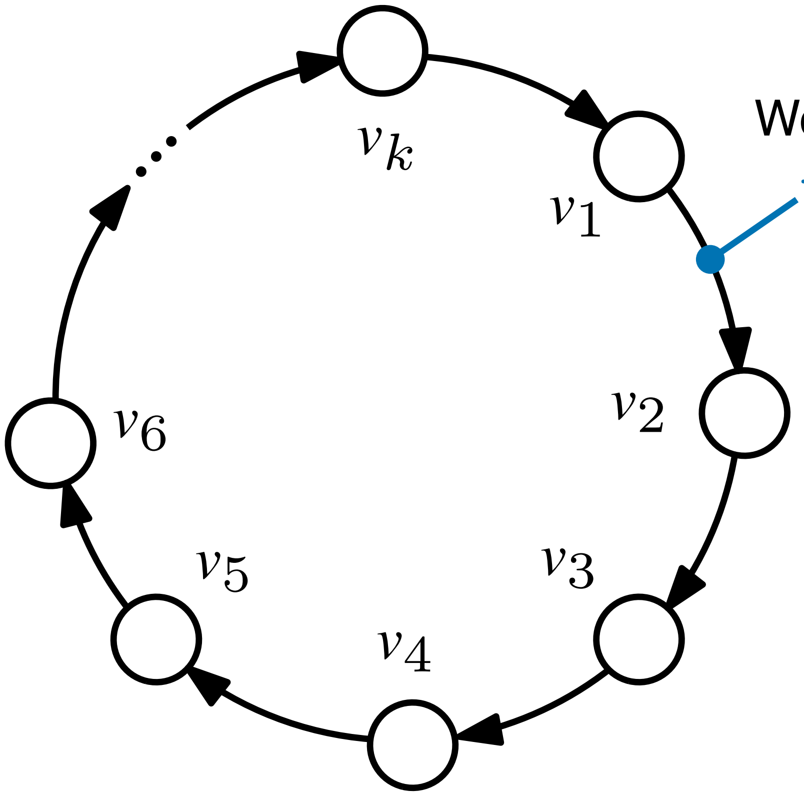
Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,
$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



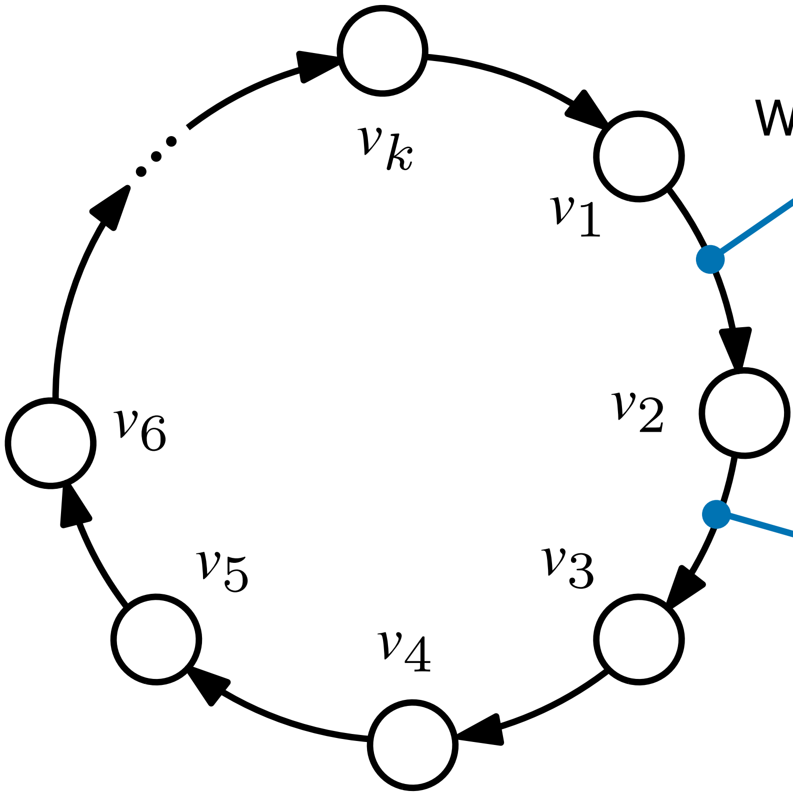
We have that,
$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$

and by the same argument...

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

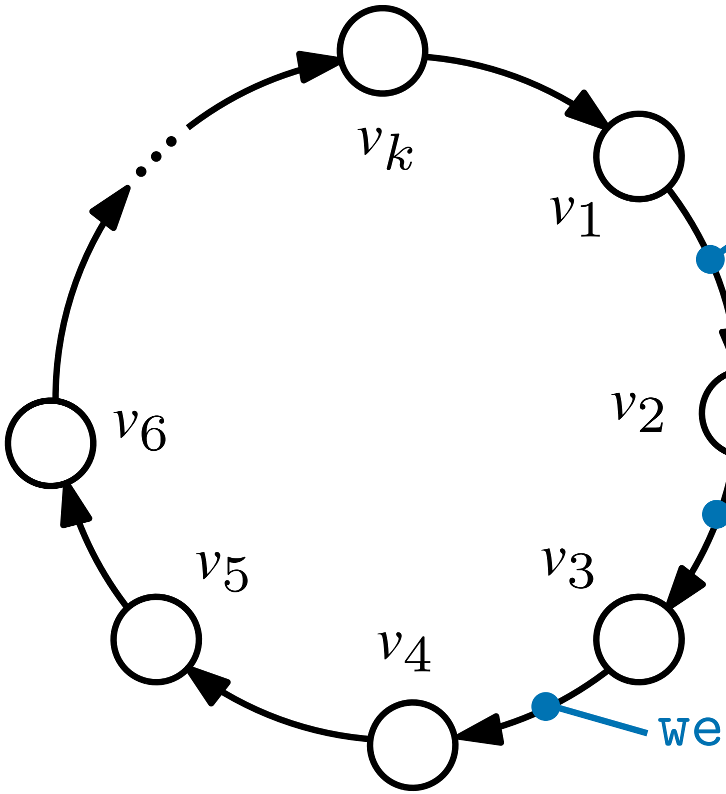
$\mathtt{weight}(v_1, v_2) \geqslant \mathtt{dist}(v_2) - \mathtt{dist}(v_1)$

and by the same argument. . .

$\mathtt{weight}(v_2, v_3) \geqslant \mathtt{dist}(v_3) - \mathtt{dist}(v_2)$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \dots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$$

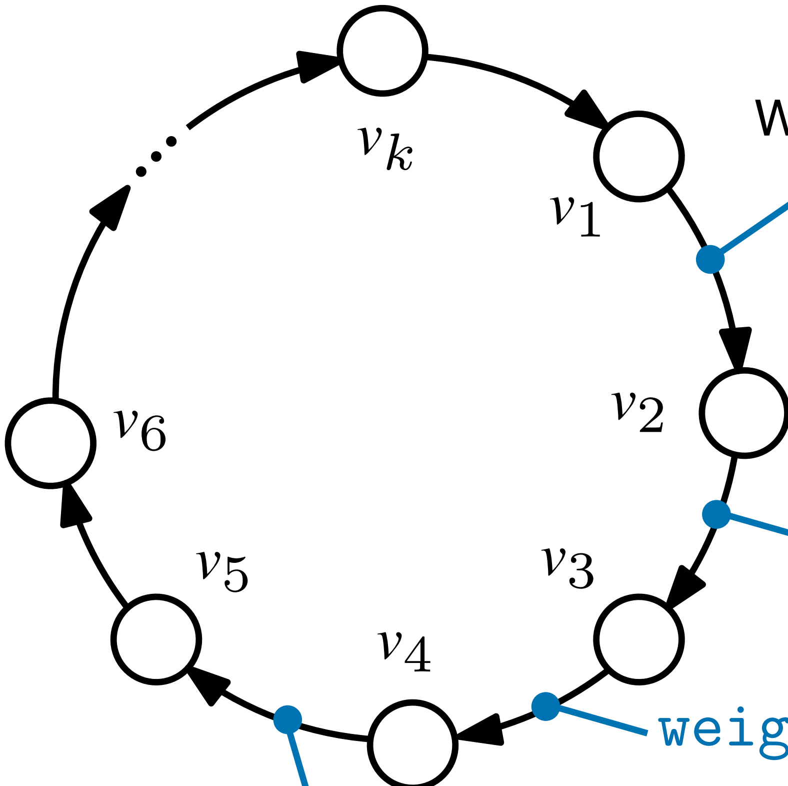
and by the same argument...

$$\texttt{weight}(v_2, v_3) \geqslant \texttt{dist}(v_3) - \texttt{dist}(v_2)$$

$$\texttt{weight}(v_3, v_4) \geqslant \texttt{dist}(v_4) - \texttt{dist}(v_3)$$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$$

and by the same argument. . .

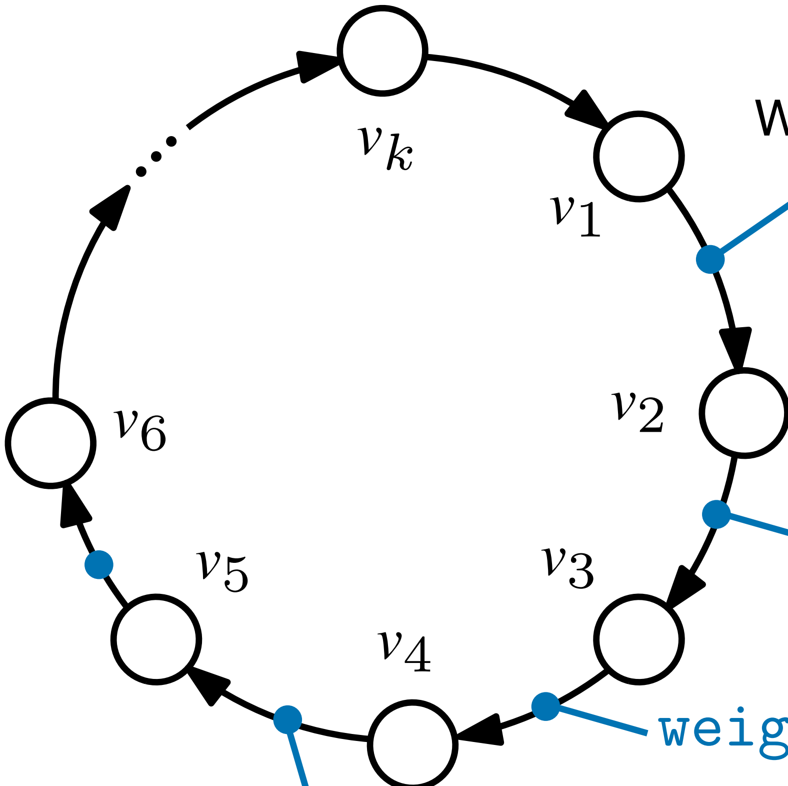
$$\texttt{weight}(v_2, v_3) \geqslant \texttt{dist}(v_3) - \texttt{dist}(v_2)$$

$$\texttt{weight}(v_3, v_4) \geqslant \texttt{dist}(v_4) - \texttt{dist}(v_3)$$

$$\texttt{weight}(v_4, v_5) \geqslant \texttt{dist}(v_5) - \texttt{dist}(v_4)$$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$$

and by the same argument...

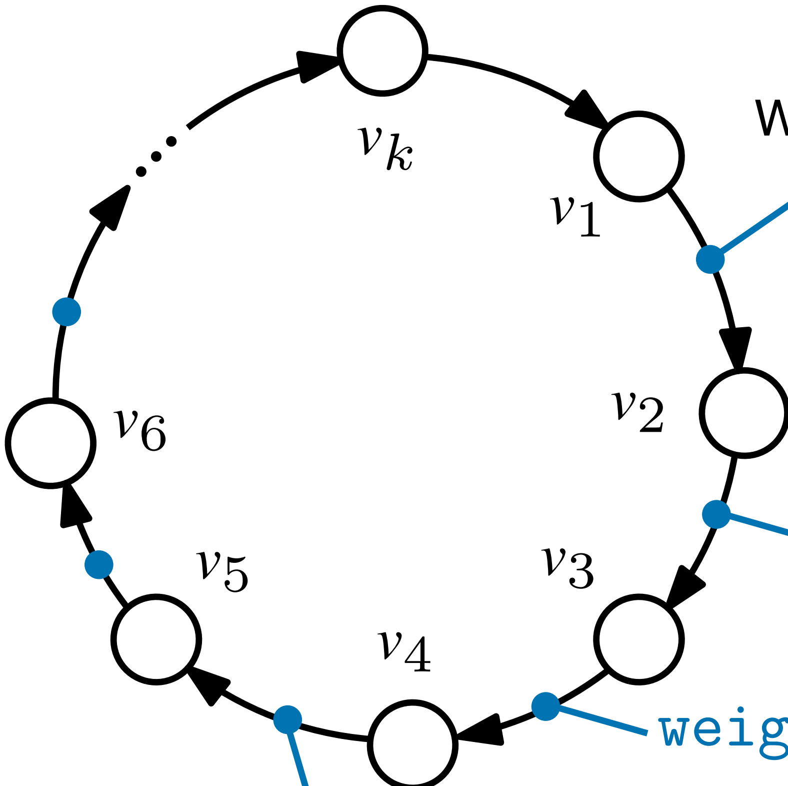
$$\texttt{weight}(v_2, v_3) \geqslant \texttt{dist}(v_3) - \texttt{dist}(v_2)$$

$$\texttt{weight}(v_3, v_4) \geqslant \texttt{dist}(v_4) - \texttt{dist}(v_3)$$

$$\texttt{weight}(v_4, v_5) \geqslant \texttt{dist}(v_5) - \texttt{dist}(v_4)$$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$

and by the same argument...

$\texttt{weight}(v_2, v_3) \geqslant \texttt{dist}(v_3) - \texttt{dist}(v_2)$

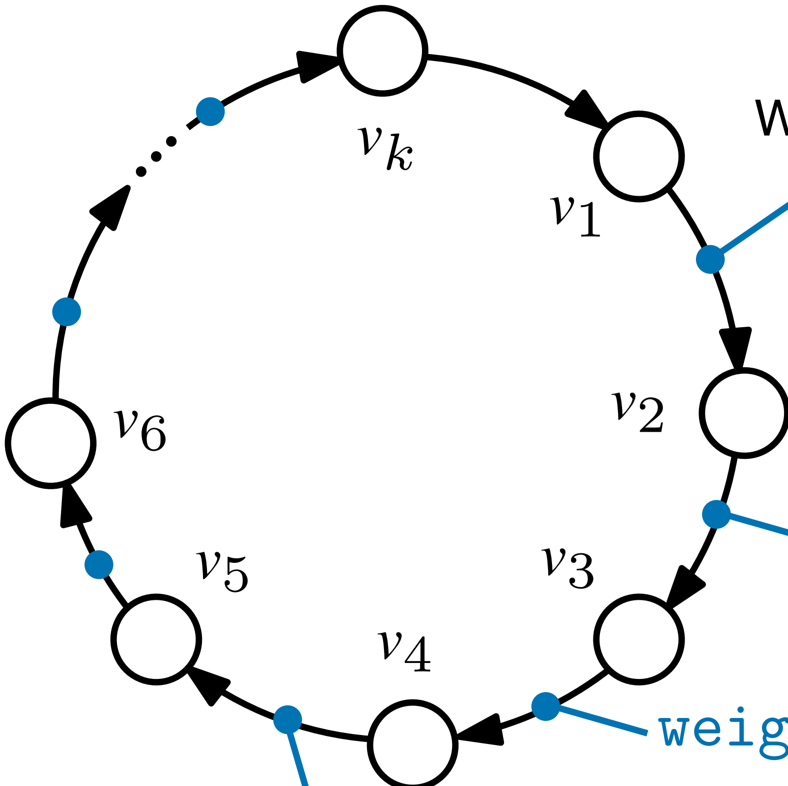
$\texttt{weight}(v_3, v_4) \geqslant \texttt{dist}(v_4) - \texttt{dist}(v_3)$

$\texttt{weight}(v_4, v_5) \geqslant \texttt{dist}(v_5) - \texttt{dist}(v_4)$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$\mathtt{weight}(v_1, v_2) \geqslant \mathtt{dist}(v_2) - \mathtt{dist}(v_1)$

and by the same argument...

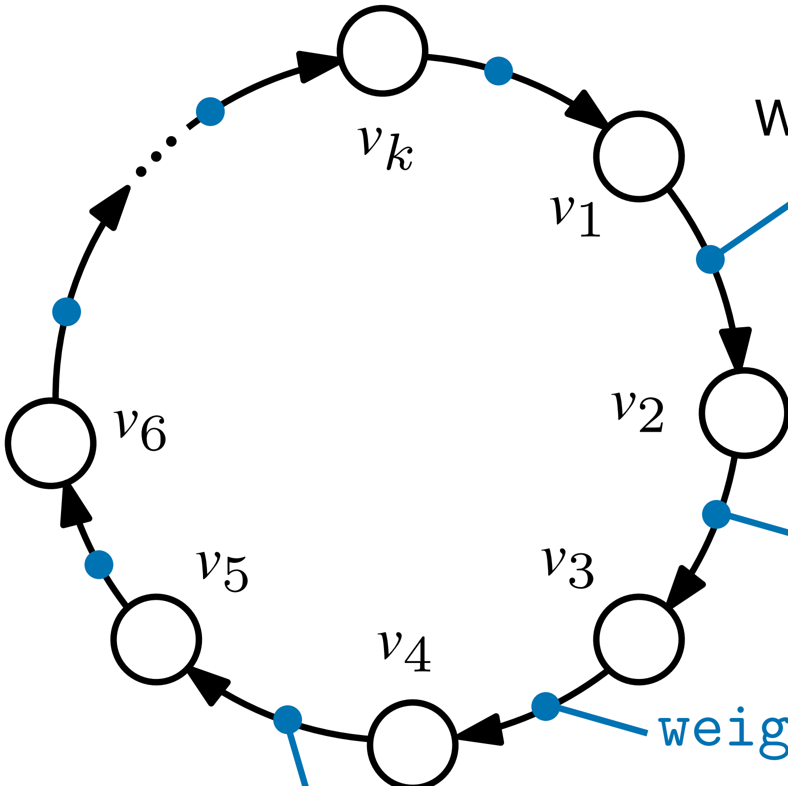
$\mathtt{weight}(v_2, v_3) \geqslant \mathtt{dist}(v_3) - \mathtt{dist}(v_2)$

$\mathtt{weight}(v_3, v_4) \geqslant \mathtt{dist}(v_4) - \mathtt{dist}(v_3)$

$\mathtt{weight}(v_4, v_5) \geqslant \mathtt{dist}(v_5) - \mathtt{dist}(v_4)$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



We have that,

$\texttt{weight}(v_1, v_2) \geqslant \texttt{dist}(v_2) - \texttt{dist}(v_1)$

and by the same argument. . .

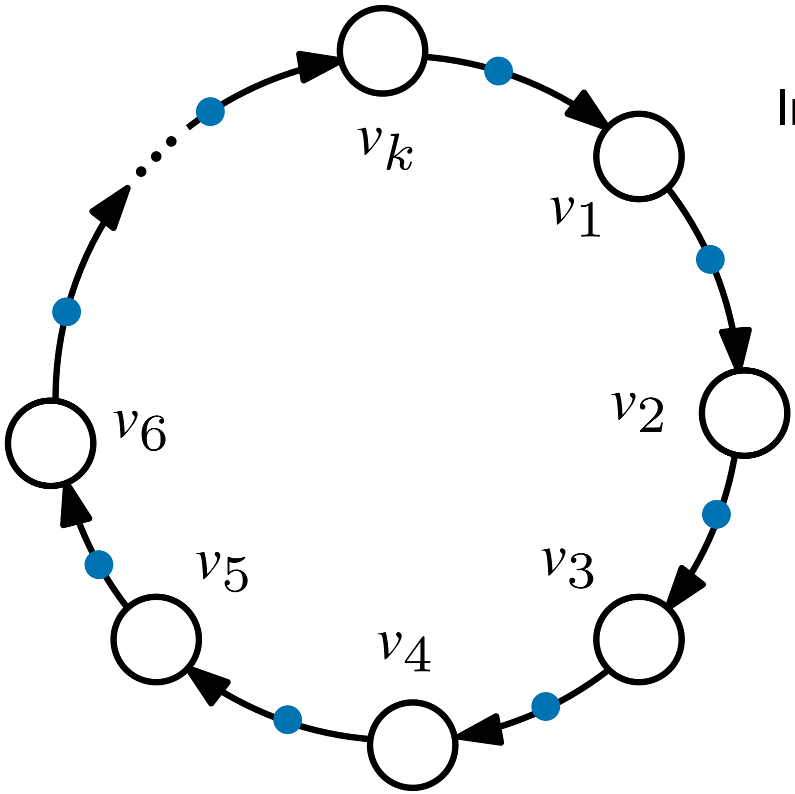
$\texttt{weight}(v_2, v_3) \geqslant \texttt{dist}(v_3) - \texttt{dist}(v_2)$

$\texttt{weight}(v_3, v_4) \geqslant \texttt{dist}(v_4) - \texttt{dist}(v_3)$

$\texttt{weight}(v_4, v_5) \geqslant \texttt{dist}(v_5) - \texttt{dist}(v_4)$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



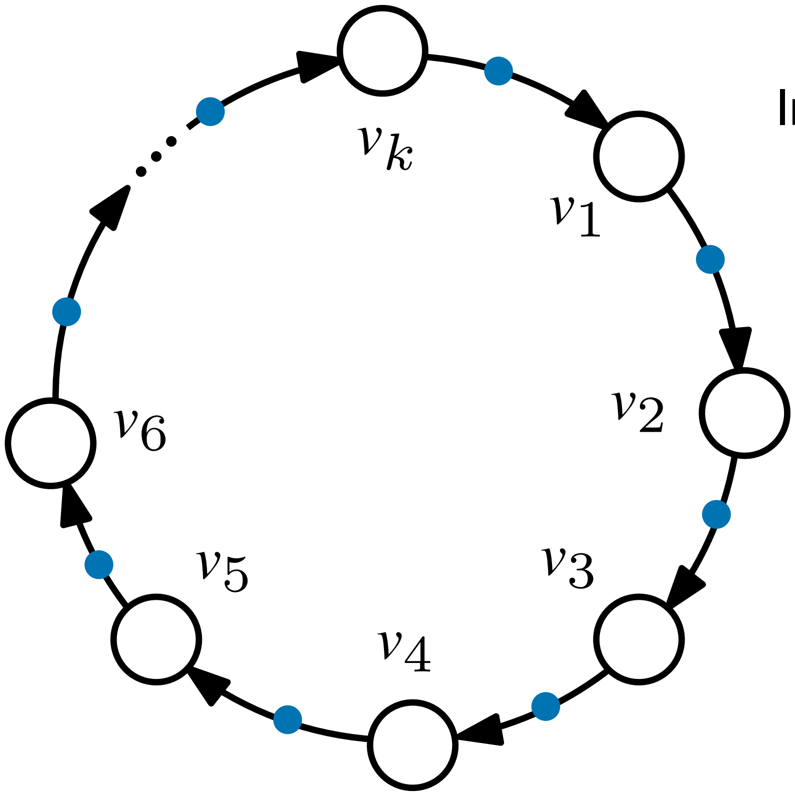
In general for every edge (v_j, v_{j+1}), we have

$$\texttt{weight}(v_j, v_{j+1}) \geqslant \texttt{dist}(v_{j+1}) - \texttt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



In general for every edge (v_j, v_{j+1}), we have

$$\texttt{weight}(v_j, v_{j+1}) \geqslant \texttt{dist}(v_{j+1}) - \texttt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

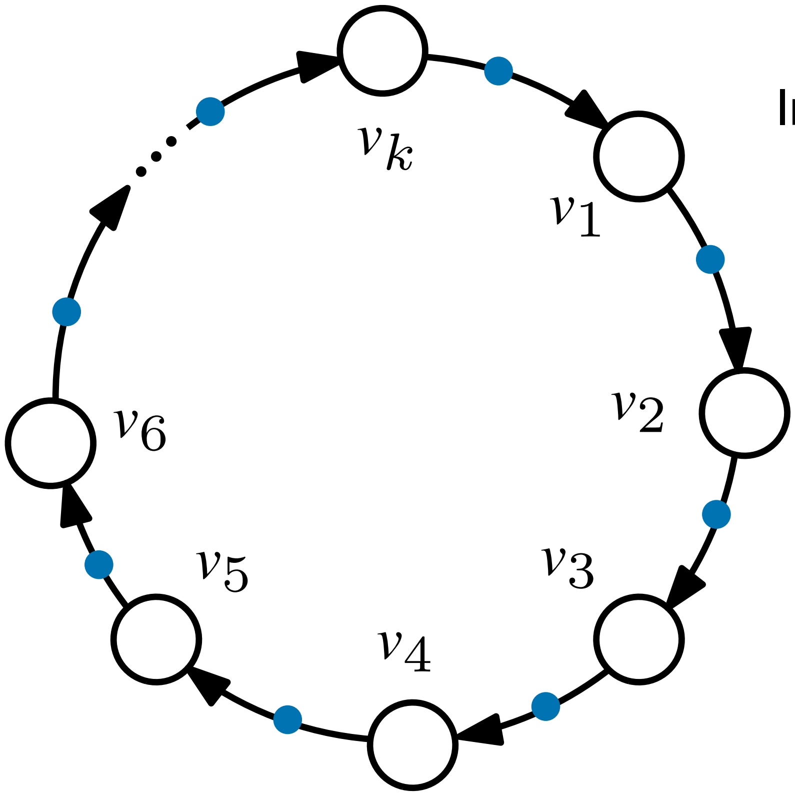
But the cycle has *negative weight* so,

$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) < 0$$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

and assume for a contradiction, that it wasn't reported by the algorithm



In general for every edge (v_j, v_{j+1}), we have

$$\texttt{weight}(v_j, v_{j+1}) \geqslant \texttt{dist}(v_{j+1}) - \texttt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

But the cycle has *negative weight* so,

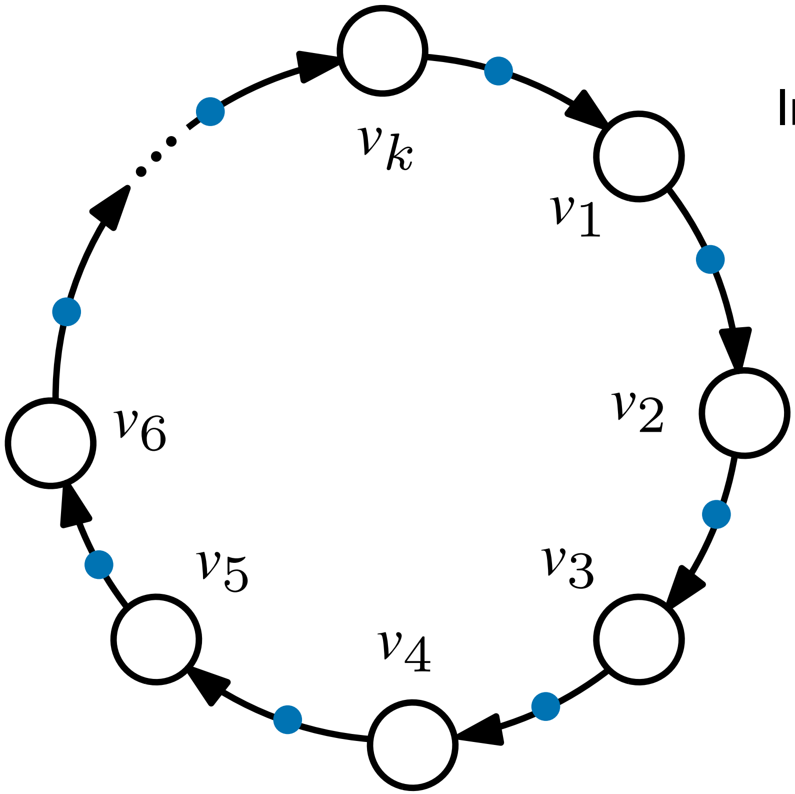
$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) < 0$$

However, if we sum all the edge weights using the top expressions we get,

$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1})$$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



In general for every edge (v_j, v_{j+1}), we have

$$\texttt{weight}(v_j, v_{j+1}) \geqslant \texttt{dist}(v_{j+1}) - \texttt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

But the cycle has *negative weight* so,

$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) < 0$$

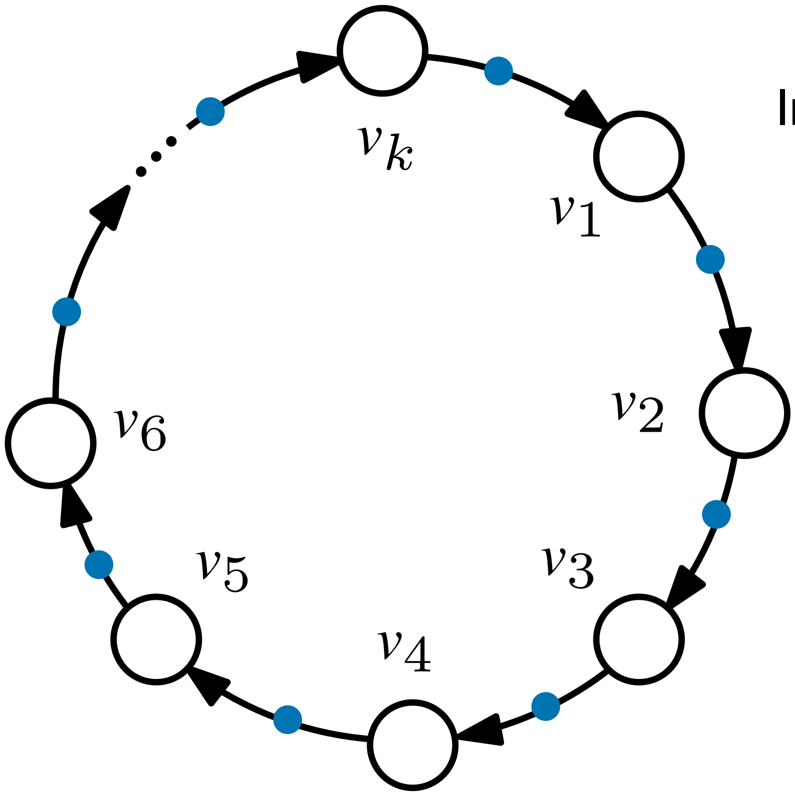
However, if we sum all the edge weights using the top expressions we get,

$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) \geqslant \sum_{j=1}^{k} (\texttt{dist}(v_{j+1}) - \texttt{dist}(v_j))$$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

and assume for a contradiction, that it wasn't reported by the algorithm



In general for every edge (v_j, v_{j+1}), we have

$$\texttt{weight}(v_j, v_{j+1}) \geqslant \texttt{dist}(v_{j+1}) - \texttt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

But the cycle has *negative weight* so,

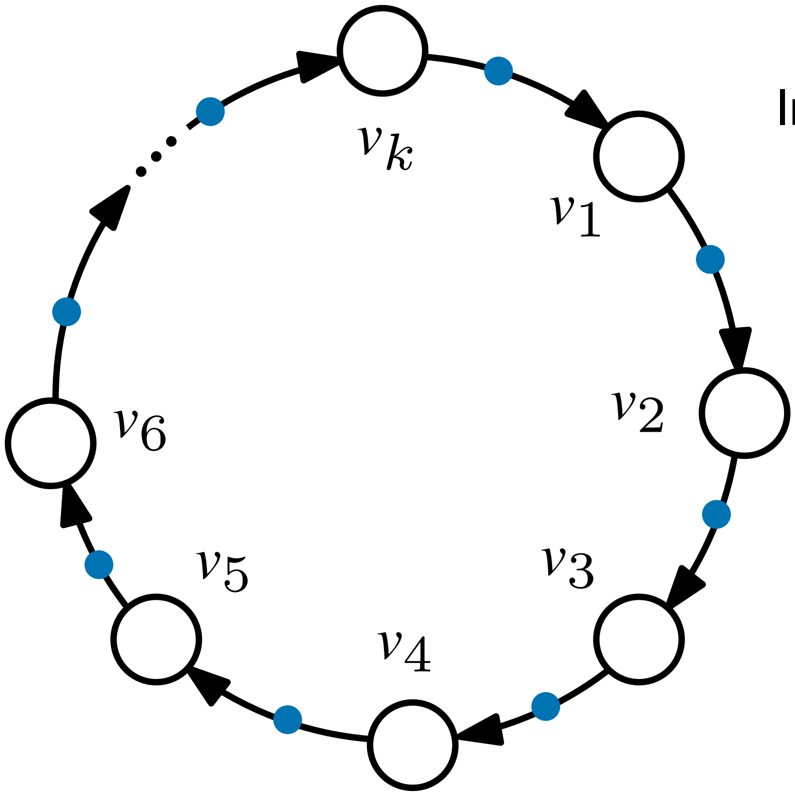
$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) < 0$$

However, if we sum all the edge weights using the top expressions we get,

$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) \geqslant \sum_{j=1}^{k} \texttt{dist}(v_{j+1}) - \sum_{j=1}^{k} \texttt{dist}(v_j)$$

Bellman-Ford *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



In general for every edge (v_j, v_{j+1}), we have

$$\texttt{weight}(v_j, v_{j+1}) \geqslant \texttt{dist}(v_{j+1}) - \texttt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

But the cycle has *negative weight* so,

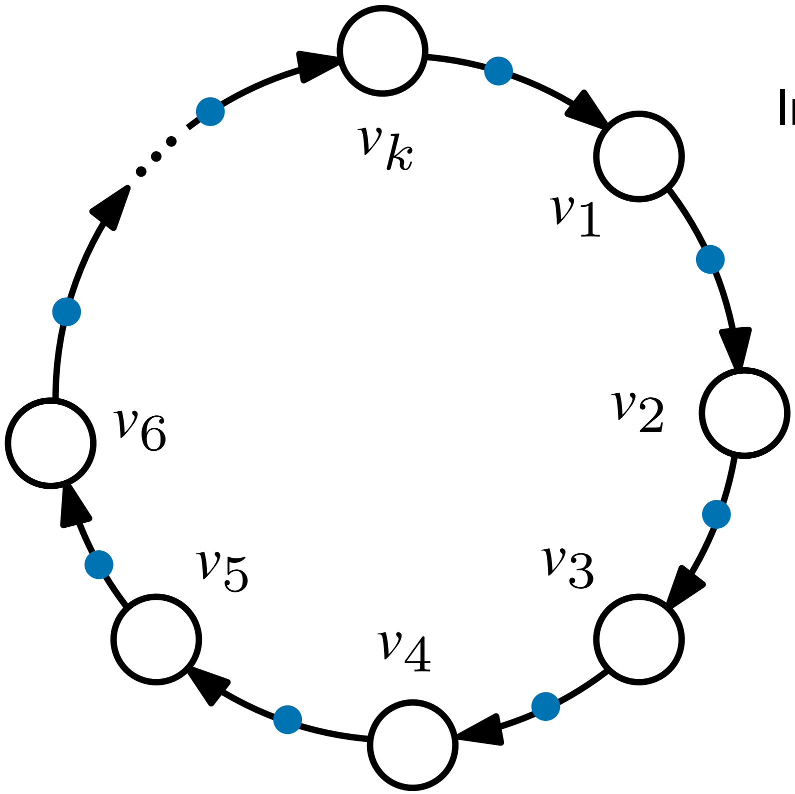
$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) < 0$$

However, if we sum all the edge weights using the top expressions we get,

$$\sum_{j=1}^{k} \texttt{weight}(v_j, v_{j+1}) \geqslant \sum_{j=1}^{k} \texttt{dist}(v_{j+1}) - \sum_{j=1}^{k} \texttt{dist}(v_j) = 0$$

BELLMAN-FORD *with* a negative weight cycle

Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle
and assume for a contradiction, that it wasn't reported by the algorithm



In general for every edge (v_j, v_{j+1}), we have

$$\mathtt{weight}(v_j, v_{j+1}) \geqslant \mathtt{dist}(v_{j+1}) - \mathtt{dist}(v_j)$$

(where $v_{k+1} = v_1$)

But the cycle has *negative weight* so,

$$\sum_{j=1}^{k} \mathtt{weight}(v_j, v_{j+1}) < 0$$

However, if we sum all the edge weights using the top expressions we get,

$$\sum_{j=1}^{k} \mathtt{weight}(v_j, v_{j+1}) \geqslant \sum_{j=1}^{k} \mathtt{dist}(v_{j+1}) - \sum_{j=1}^{k} \mathtt{dist}(v_j) = 0$$

Contradiction!

BELLMAN-FORD Summary

When the BELLMAN-FORD algorithm terminates,

for each vertex v, $\texttt{dist}(v)$ is the distance between s and v

(or it reports that there is a negative weight cycle)

BELLMAN-FORD(s)

For all v, set $\texttt{dist}(v) = \infty$
set $\texttt{dist}(s) = 0$
For $i = 1, 2, \ldots, |V|$,
  For every edge $(u, v) \in E$
    if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
      $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

For every edge $(u, v) \in E$
  if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
    Output ‘Negative weight cycle found’

BELLMAN-FORD Summary

When the BELLMAN-FORD algorithm terminates,

for each vertex v, $\texttt{dist}(v)$ is the distance between s and v

(or it reports that there is a negative weight cycle)

BELLMAN-FORD(s)

For all v, set $\texttt{dist}(v) = \infty$
set $\texttt{dist}(s) = 0$
For $i = 1, 2, \ldots, |V|$,
  For every edge $(u, v) \in E$
    if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
      $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

For every edge $(u, v) \in E$
  if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
    Output ‘Negative weight cycle found’

What is the time complexity of the BELLMAN-FORD algorithm?

Time Complexity

BELLMAN-FORD(s)

For all v, set $\texttt{dist}(v) = \infty$
set $\texttt{dist}(s) = 0$
For $i = 1, 2, \ldots, |V|$,
  For every edge $(u, v) \in E$
    if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
      $\texttt{dist}(v) = \texttt{dist}(u) + \texttt{weight}(u, v)$

For every edge $(u, v) \in E$
  if $\texttt{dist}(v) > \texttt{dist}(u) + \texttt{weight}(u, v)$
    Output 'Negative weight cycle found'

Time Complexity

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

$O(|V|)$ time

Time Complexity

```
BELLMAN-FORD(s)
```

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

$O(|V|)$ time

(*we store* dist *in an array of length* $|V|$)

Time Complexity

```
BELLMAN-FORD(s)

For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
   For every edge (u, v) ∈ E
      if dist(v) > dist(u) + weight(u, v)
         dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
   if dist(v) > dist(u) + weight(u, v)
      Output 'Negative weight cycle found'
```

$O(|V|)$ time

(*we store* dist *in an array of length* $|V|$)

$O(1)$ time

Time Complexity

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
   For every edge (u,v) ∈ E
      if dist(v) > dist(u) + weight(u,v)
         dist(v) = dist(u) + weight(u,v)

For every edge (u,v) ∈ E
   if dist(v) > dist(u) + weight(u,v)
      Output 'Negative weight cycle found'
```

$O(|V|)$ time

(*we store* $\texttt{dist}$ *in an array of length* $|V|$)

$O(1)$ time

$O(|E|)$ time for each iteration

Time Complexity

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output 'Negative weight cycle found'
```

$O(|V|)$ time

(*we store* dist *in an array of length* $|V|$)

$O(1)$ time

$O(|E|)$ time for each iteration

…and there are $|V|$ iterations

Time Complexity

BELLMAN-FORD(s)

For all v, set dist$(v) = \infty$
set dist$(s) = 0$
For $i = 1, 2, \ldots, |V|$,
  For every edge $(u, v) \in E$
    if dist$(v) >$ dist$(u) +$ weight(u, v)
      dist$(v) =$ dist$(u) +$ weight(u, v)

For every edge $(u, v) \in E$
  if dist$(v) >$ dist$(u) +$ weight(u, v)
    Output 'Negative weight cycle found'

$O(|V|)$ time

(*we store* dist *in an array of length* $|V|$)

$O(1)$ time

$O(|V||E|)$ time overall

Time Complexity

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u, v) ∈ E
    if dist(v) > dist(u) + weight(u, v)
      dist(v) = dist(u) + weight(u, v)

For every edge (u, v) ∈ E
  if dist(v) > dist(u) + weight(u, v)
    Output ‘Negative weight cycle found
```

$O(|V|)$ time

(*we store* dist *in an array of length* $|V|$)

$O(1)$ time

$O(|V||E|)$ time overall

$O(|E|)$ time

Time Complexity

BELLMAN-FORD(s)

```
For all v, set dist(v) = ∞
set dist(s) = 0
For i = 1, 2, ..., |V|,
  For every edge (u,v) ∈ E
    if dist(v) > dist(u) + weight(u,v)
      dist(v) = dist(u) + weight(u,v)

For every edge (u,v) ∈ E
  if dist(v) > dist(u) + weight(u,v)
    Output 'Negative weight cycle found
```

$O(|V|)$ time

(*we store* $\texttt{dist}$ *in an array of length* $|V|$)

$O(1)$ time

$O(|V||E|)$ time overall

$O(|E|)$ time

so overall the BELLMAN-FORD algorithm takes $O(|V||E|)$ time

Summary

The Bellman-Ford algorithm solves the **single source shortest paths** problem in a directed, weighted graph with **positive and negative edge weights**

If there is a negative weight cycle, the algorithm reports this and aborts *(in this case, the problem is not well-defined)*

The Bellman-Ford algorithm runs in $O(|V||E|)$ time

and uses no non-elementary data structures

This is not as good as Dijkstra's algorithm which runs in $O((|V| + |E|) \log |V|)$ time

(when implemented using a binary heap)

However, Dijkstra's algorithm only works for graphs with **non-negative edge weights**

University of BRISTOL

End of part one

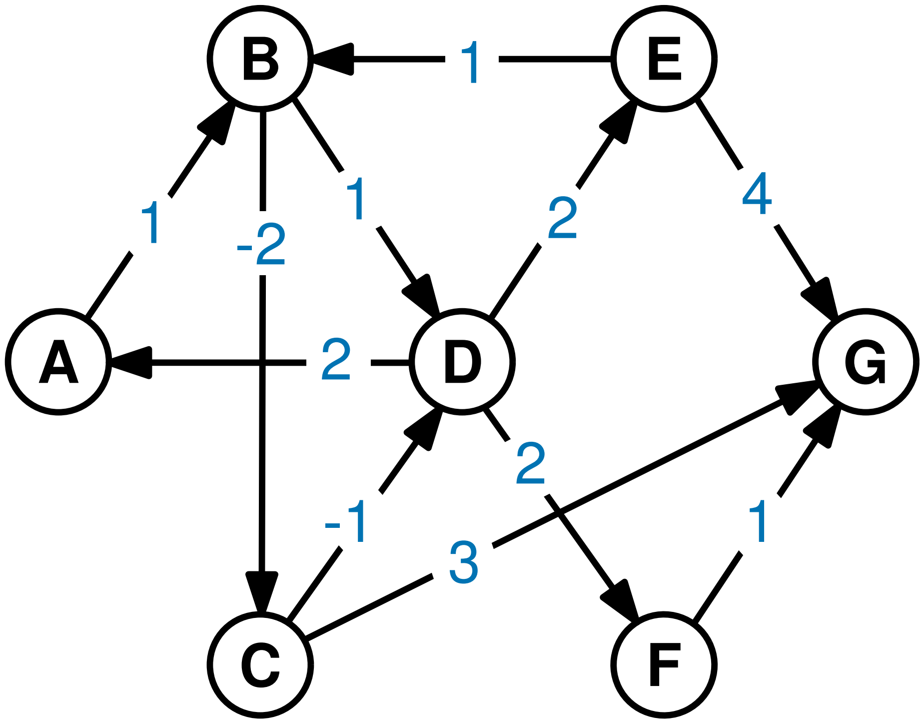
Part two

All-Pairs Shortest Paths

All-Pairs Shortest Paths

In previous lectures, we have focused on the **single source** **shortest paths** problem

in a **weighted**, directed graph. . .

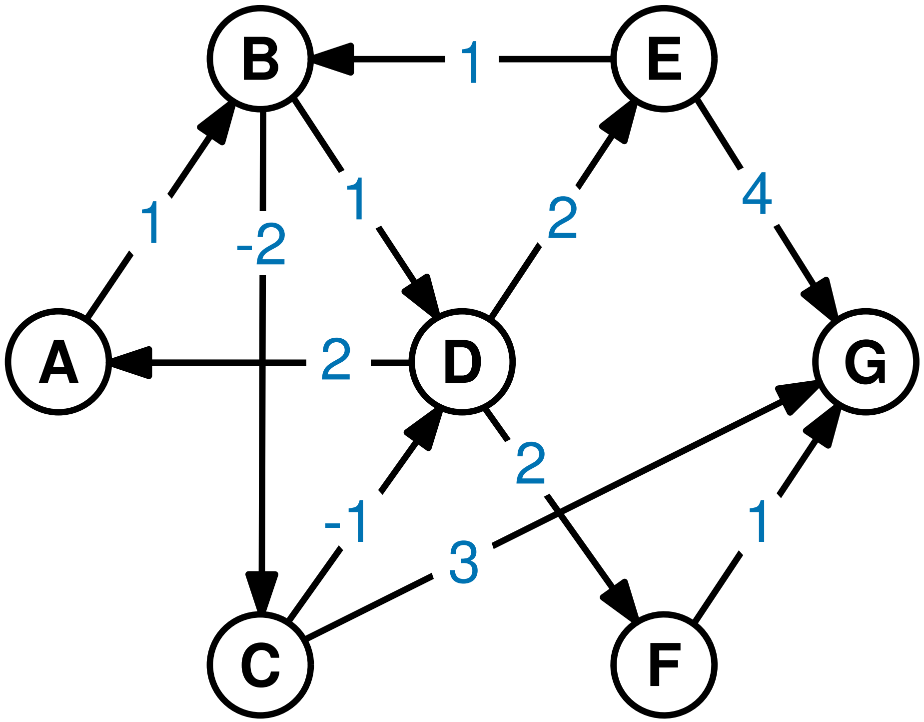


All-Pairs Shortest Paths

In previous lectures, we have focused on the **single source** **shortest paths** problem

in a **weighted**, directed graph...

i.e. in finding the shortest paths from a single given source vertex
to every other vertex

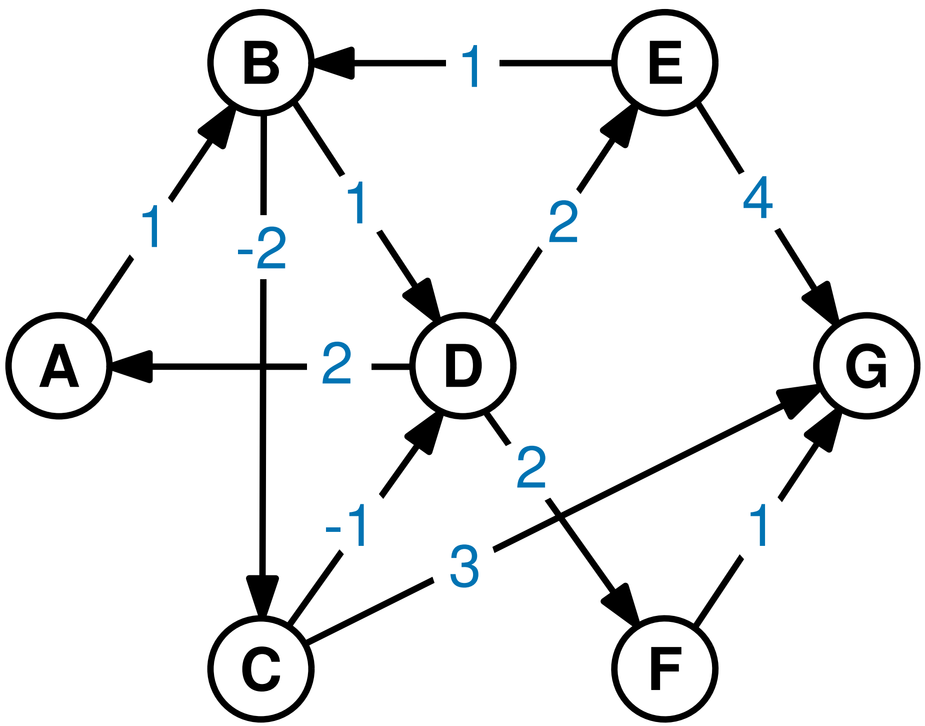


All-Pairs Shortest Paths

In previous lectures, we have focused on the **single source shortest paths** problem

in a **weighted**, directed graph. . .

i.e. in finding the shortest paths from a single given source vertex
to every other vertex



What should we do if we want to find the shortest paths from every vertex
to every other vertex?

Use something we saw before

What should we do if we want to find the shortest paths from every vertex to every other vertex?

Use something we saw before

What should we do if we want to find the shortest paths from every vertex
to every other vertex?

We already have two options:

Use something we saw before

What should we do if we want to find the shortest paths from every vertex to every other vertex?

We already have two options:

If the graph has **non-negative edge weights**,

we could run DIJKSTRA's algorithm $|V|$ times, once with each vertex as the source

this takes $O(|V||E| \log |V|)$ time

(if the priority queue is a binary heap)

Use something we saw before

What should we do if we want to find the shortest paths from every vertex to every other vertex?

We already have two options:

If the graph has **non-negative edge weights**,

we could run DIJKSTRA's algorithm $|V|$ times, once with each vertex as the source

this takes $O(|V||E| \log |V|)$ time

(if the priority queue is a binary heap)

If the graph has **positive and negative edge weights**,

we could run BELLMAN-FORD's algorithm $|V|$ times, once for each vertex

this takes $O(|V|^2|E|)$ time

University of BRISTOL

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good?

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?*

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does* $|V|$ *compare to* $|E|$*?*

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does* $|V|$ *compare to* $|E|$*?*

The output contains the length of the shortest path between every pair of vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does* $|V|$ *compare to* $|E|$*?*

The output contains the length of the shortest path between every pair of vertices

There are $|V| \cdot (|V| - 1)$ pairs of vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does* $|V|$ *compare to* $|E|$*?*

The output contains the length of the shortest path between every pair of vertices

There are $|V| \cdot (|V| - 1)$ pairs of vertices

so we can't expect to do better than $O(|V|^2)$ time

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Imagine that $|E| \approx \frac{|V|^2}{4}$ (the graph is very dense)

e.g. each vertex has an edge to about half of the other vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

this becomes $O(|V|^3 \log |V|)$

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Imagine that $|E| \approx \frac{|V|^2}{4}$ (the graph is very dense)

e.g. each vertex has an edge to about half of the other vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

this becomes $O(|V|^3 \log |V|)$

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

this becomes $O(|V|^4)$

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Imagine that $|E| \approx \dfrac{|V|^2}{4}$ (the graph is very dense)

e.g. each vertex has an edge to about half of the other vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does* $|V|$ *compare to* $|E|$*?*

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Instead, imagine that $|E| \approx 5|V|$ (the graph is very sparse)

e.g. each vertex has an edge to about 5 other vertices

What should we do if we want to find the shortest paths from every vertex
to every other vertex?

this becomes $O(|V|^2 \log |V|)$

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Instead, imagine that $|E| \approx 5|V|$ (the graph is very sparse)

e.g. each vertex has an edge to about 5 other vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

this becomes $O(|V|^2 \log |V|)$

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

this becomes $O(|V|^3)$

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Instead, imagine that $|E| \approx 5|V|$ (the graph is very sparse)

e.g. each vertex has an edge to about 5 other vertices

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

this becomes $O(|V|^2 \log |V|)$

repeatedly running DIJKSTRA's algorithm takes $O(|V||E| \log |V|)$ time

If the graph has **positive and negative edge weights**,

this becomes $O(|V|^3)$

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

Are these any good? *Can we do better?* *How does $|V|$ compare to $|E|$?*

Instead, imagine that $|E| \approx 5|V|$ (the graph is very sparse)

e.g. each vertex has an edge to about 5 other vertices

$O(|V|^2 \log |V|)$ is *a lot better* than $O(|V|^3)$

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

In this lecture, we will discuss JOHNSON's algorithm for all-pairs shortest paths in graphs with **positive and negative edge weights** which takes $O(|V||E|\log|V|)$ time

What should we do if we want to find the shortest paths from every vertex to every other vertex?

If the graph has **non-negative edge weights**,

repeatedly running DIJKSTRA's algorithm takes $O(|V||E|\log|V|)$ time

If the graph has **positive and negative edge weights**,

repeatedly running BELLMAN-FORD's algorithm takes $O(|V|^2|E|)$ time

In this lecture, we will discuss JOHNSON's algorithm for all-pairs shortest paths in graphs with **positive and negative edge weights** which takes $O(|V||E|\log|V|)$ time

It will use *both* DIJKSTRA's algorithm (implemented with a binary heap) and BELLMAN-FORD to achieve this

JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths
which takes $O(|V||E|\log|V|)$ time

JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E| \log |V|)$ time

If the graph had **non-negative edge weights**,

we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E| \log |V|)$ time

If the graph had **non-negative edge weights**,

we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

However, we are interested in graphs with both **positive and negative edge weights**,

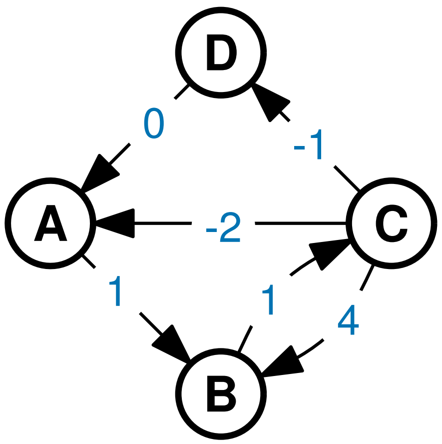
JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E|\log|V|)$ time

If the graph had **non-negative edge weights**,

we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

However, we are interested in graphs with both **positive and negative edge weights**,



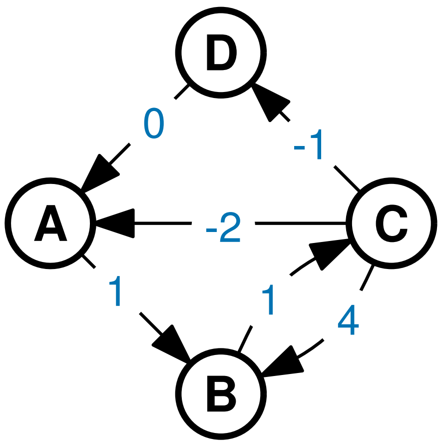
JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E|\log|V|)$ time

If the graph had **non-negative edge weights**,

we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

However, we are interested in graphs with both **positive and negative edge weights**,



The approach employed by JOHNSON's algorithm is to **reweight** the edges

so that the resulting graph has **non-negative edge weights**,

and then (repeatedly) run DIJKSTRA*'s algorithm*

JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E|\log|V|)$ time

If the graph had **non-negative edge weights**,

we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

However, we are interested in graphs with both **positive and negative edge weights**,

The approach employed by JOHNSON's algorithm is to **reweight** the edges

so that the resulting graph has **non-negative edge weights**,

and then (repeatedly) run DIJKSTRA*'s algorithm*

JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E|\log|V|)$ time

If the graph had **non-negative edge weights**,

we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

However, we are interested in graphs with both **positive and negative edge weights**,

The approach employed by JOHNSON's algorithm is to **reweight** the edges

so that the resulting graph has **non-negative edge weights**,

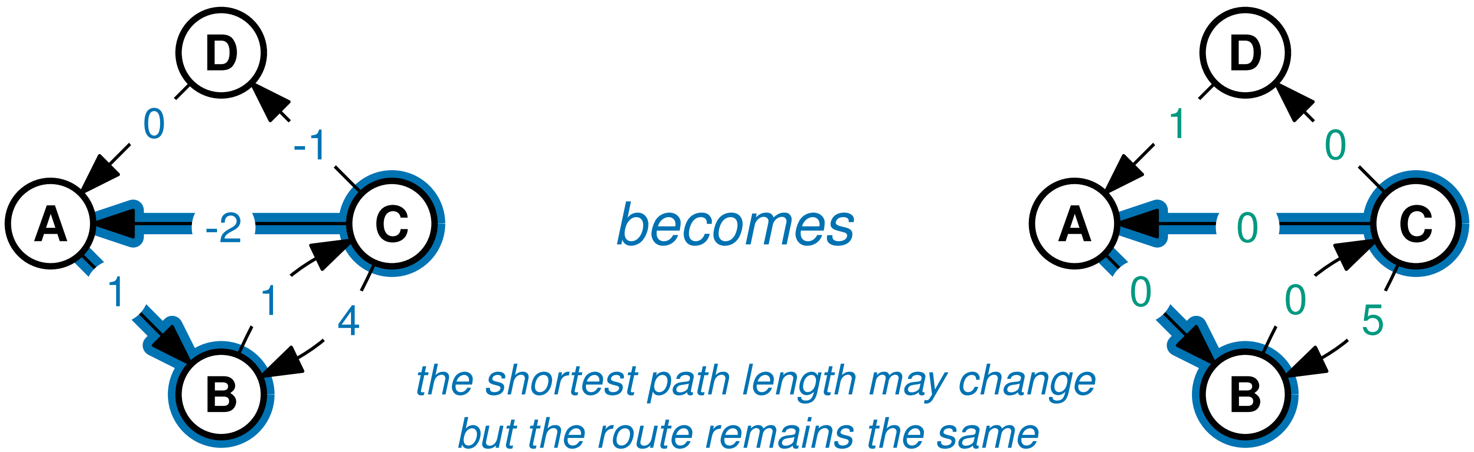
and then (repeatedly) run DIJKSTRA*'s algorithm*

JOHNSON's algorithm - the overall approach

We have already seen one algorithm for all-pairs shortest paths which takes $O(|V||E|\log|V|)$ time

If the graph had **non-negative edge weights**, we could have achieved this complexity by repeatedly running DIJKSTRA's algorithm

However, we are interested in graphs with both **positive and negative edge weights**,



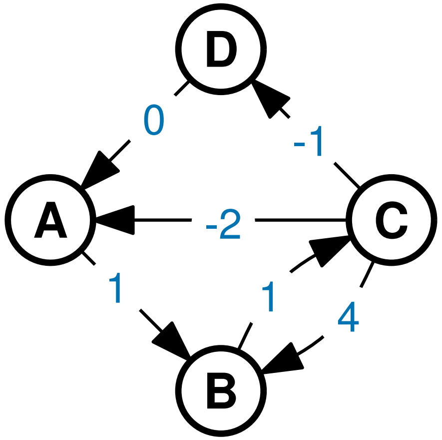
The approach employed by JOHNSON's algorithm is to **reweight** the edges so that the resulting graph has **non-negative edge weights**, *and then (repeatedly) run* DIJKSTRA*'s algorithm*

A first attempt

One natural attempt at reweighting is to increase every edge weight
by the same amount. . .

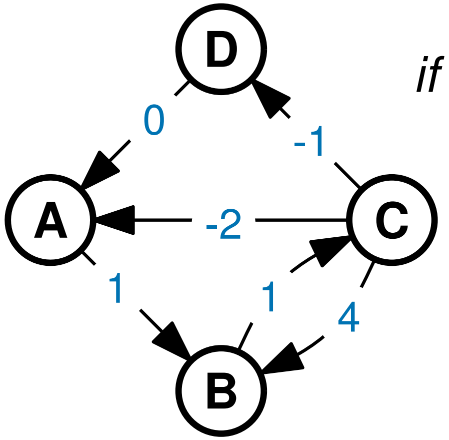
A first attempt

One natural attempt at reweighting is to increase every edge weight
by the same amount. . .



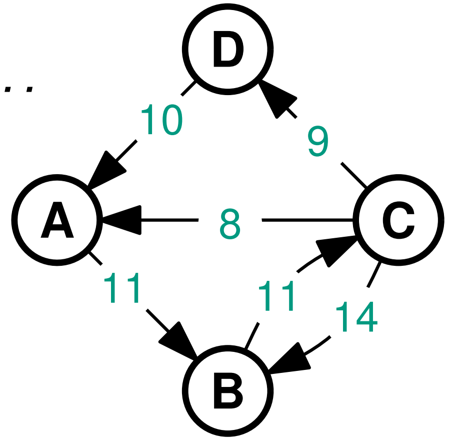
A first attempt

One natural attempt at reweighting is to increase every edge weight by the same amount. . .



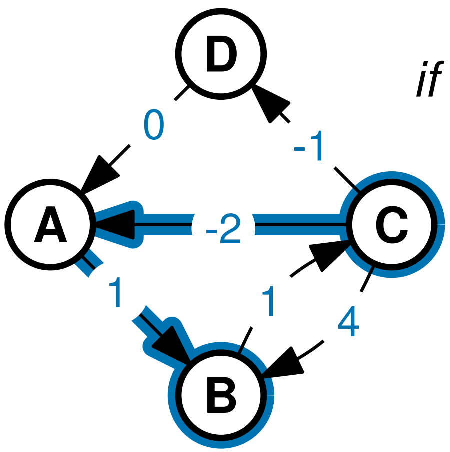
if we add 10 *to every edge. . .*

becomes



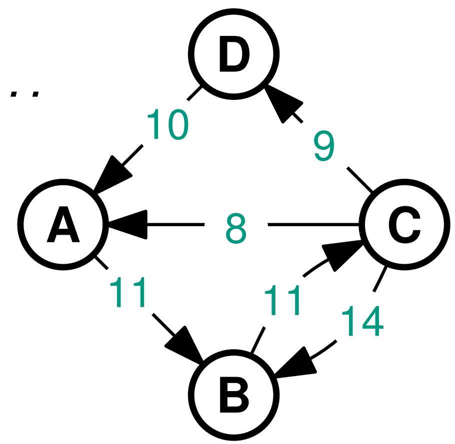
A first attempt

One natural attempt at reweighting is to increase every edge weight by the same amount...



if we add 10 *to every edge...*

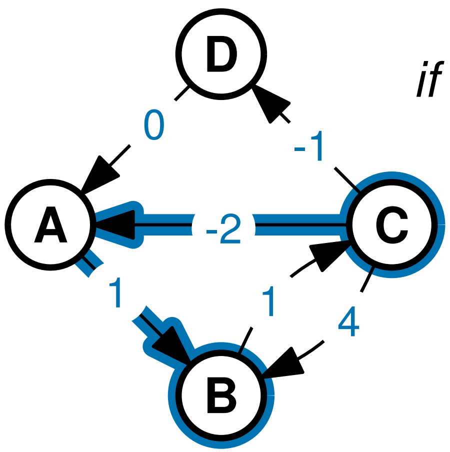
becomes



The shortest path from **C** to **B** goes this way in the original graph

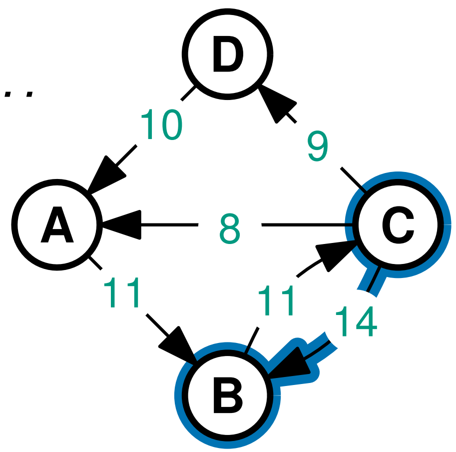
A first attempt

One natural attempt at reweighting is to increase every edge weight by the same amount...



if we add 10 *to every edge...*

becomes

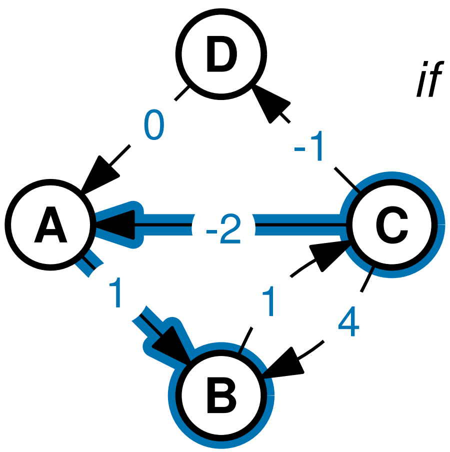


The shortest path from **C** to **B** goes this way in the original graph

The shortest path from **C** to **B** goes this way in the reweighted graph

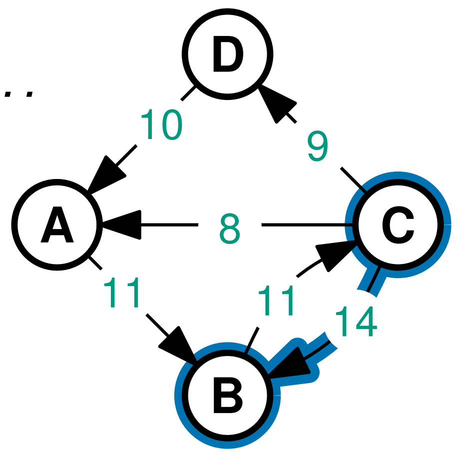
A first attempt

One natural attempt at reweighting is to increase every edge weight by the same amount...



if we add 10 *to every edge...*

becomes



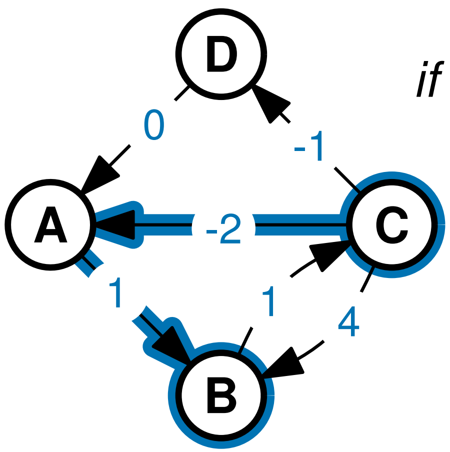
The shortest path from **C** to **B** goes this way in the original graph

The shortest path from **C** to **B** goes this way in the reweighted graph

Unfortunately, if we reweight the graph like this, both the shortest paths lengths and the routes might change

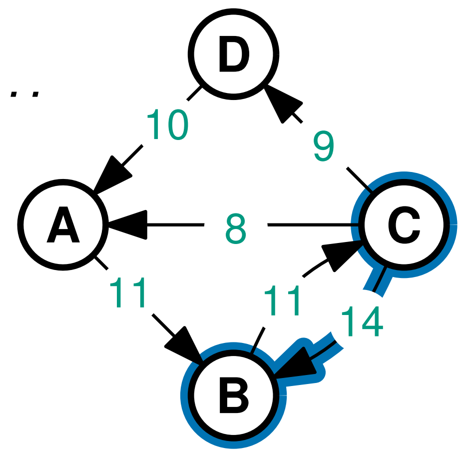
A first attempt

One natural attempt at reweighting is to increase every edge weight by the same amount…



if we add 10 *to every edge…*

becomes



The shortest path from **C** to **B** goes this way in the original graph

The shortest path from **C** to **B** goes this way in the reweighted graph

Unfortunately, if we reweight the graph like this,
both the shortest paths lengths and the routes might change

this doesn't look good

Reweighting based on vertex potential

To overcome this, we are going to reweight each edge differently

the new weight of each edge will depend on which vertices it connects

We are going to associate a value $\mathsf{h}(v)$ with each vertex $v \in V$

- we will call this the potential of v

Reweighting based on vertex potential

To overcome this, we are going to reweight each edge differently

the new weight of each edge will depend on which vertices it connects

We are going to associate a value $\mathsf{h}(v)$ with each vertex $v \in V$

- we will call this the potential of v

Reweighting based on vertex potential

To overcome this, we are going to reweight each edge differently

the new weight of each edge will depend on which vertices it connects

We are going to associate a value $\mathtt{h}(v)$ with each vertex $v \in V$

- we will call this the potential of v

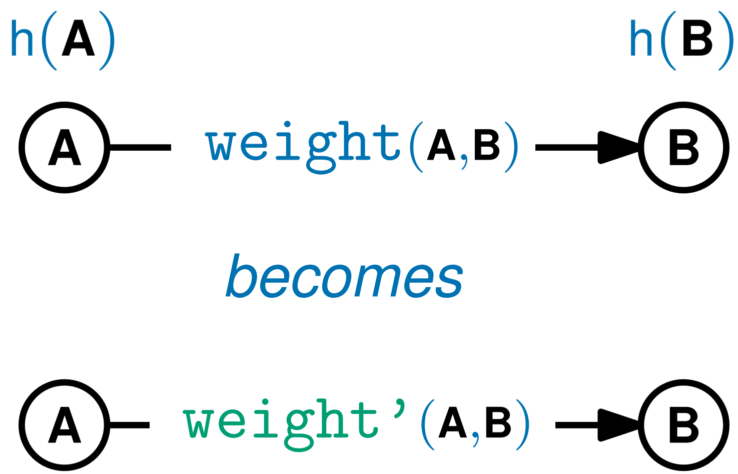
Reweighting based on vertex potential

To overcome this, we are going to reweight each edge differently

the new weight of each edge will depend on which vertices it connects

We are going to associate a value $\mathsf{h}(v)$ with each vertex $v \in V$

- we will call this the potential of v



where $\texttt{weight'}(\mathbf{A},\mathbf{B}) = \texttt{weight}(\mathbf{A},\mathbf{B}) + \mathsf{h}(\mathbf{A}) - \mathsf{h}(\mathbf{B})$

Reweighting based on vertex potential

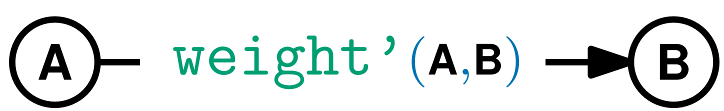
To overcome this, we are going to reweight each edge differently

the new weight of each edge will depend on which vertices it connects

We are going to associate a value $\mathtt{h}(v)$ with each vertex $v \in V$

- we will call this the potential of v

becomes



where $\mathtt{weight'}(\mathbf{A},\mathbf{B}) = \mathtt{weight}(\mathbf{A},\mathbf{B}) + \mathtt{h}(\mathbf{A}) - \mathtt{h}(\mathbf{B})$

Why is this better than the first attempt?

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example...

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

Pick some potential values

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

Pick some potential values

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example...

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$$\mathsf{h}(v_1) = 5 \quad \mathsf{h}(v_2) = 2 \quad \mathsf{h}(v_3) = 0 \quad \mathsf{h}(v_4) = 1 \quad \mathsf{h}(v_5) = 2 \quad \mathsf{h}(v_6) = 4$$

Pick some potential values

(for now you can think of the values as arbitrary)

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example...

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

Reweight the whole graph using `weight'` *(we don't just reweight this path)*...

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$$\mathsf{h}(v_1) = 5 \quad \mathsf{h}(v_2) = 2 \quad \mathsf{h}(v_3) = 0 \quad \mathsf{h}(v_4) = 1 \quad \mathsf{h}(v_5) = 2 \quad \mathsf{h}(v_6) = 4$$

Reweight the whole graph using `weight'` *(we don't just reweight this path)*. . .

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example...

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$$\mathsf{h}(v_1) = 5 \quad \mathsf{h}(v_2) = 2 \quad \mathsf{h}(v_3) = 0 \quad \mathsf{h}(v_4) = 1 \quad \mathsf{h}(v_5) = 2 \quad \mathsf{h}(v_6) = 4$$

Reweight the whole graph using $\texttt{weight'}$ *(we don't just reweight this path)*...

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

$$\texttt{weight'}(v_1, v_2) = \texttt{weight}(v_1, v_2) + \mathsf{h}(v_1) - \mathsf{h}(v_2)$$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$$\mathsf{h}(v_1) = 5 \quad \mathsf{h}(v_2) = 2 \quad \mathsf{h}(v_3) = 0 \quad \mathsf{h}(v_4) = 1 \quad \mathsf{h}(v_5) = 2 \quad \mathsf{h}(v_6) = 4$$

Reweight the whole graph using $\texttt{weight'}$ *(we don't just reweight this path)*. . .

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

$$\texttt{weight'}(v_1, v_2) = \texttt{weight}(v_1, v_2) + \mathsf{h}(v_1) - \mathsf{h}(v_2)$$
$$= (-2) + (5) - (2) = 1$$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example...

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

Reweight the whole graph using `weight'` *(we don't just reweight this path)*...

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

- *this path has length* 13

Reweight the whole graph using $\texttt{weight'}$ *(we don't just reweight this path)*. . .

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example...

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

- *this path has length* 13

Reweight the whole graph using $\texttt{weight'}$ *(we don't just reweight this path)*...

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

- *this path has length* 14

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathsf{h}(v_1) = 5$ $\mathsf{h}(v_2) = 2$ $\mathsf{h}(v_3) = 0$ $\mathsf{h}(v_4) = 1$ $\mathsf{h}(v_5) = 2$ $\mathsf{h}(v_6) = 4$

- *this path has length* 13

Reweight the whole graph using $\texttt{weight'}$ *(we don't just reweight this path)*. . .

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

- *this path has length* $14 = 13 + 5 - 4 = 13 + h(v_1) - h(v_6)$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathsf{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$$\mathsf{h}(v_1) = 5 \quad \mathsf{h}(v_2) = 2 \quad \mathsf{h}(v_3) = 0 \quad \mathsf{h}(v_4) = 1 \quad \mathsf{h}(v_5) = 2 \quad \mathsf{h}(v_6) = 4$$

- *this path has length* 13

Reweight the whole graph using $\texttt{weight'}$ *(we don't just reweight this path)*. . .

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

- *this path has length* $14 = 13 + 5 - 4 = 13 + h(v_1) - h(v_6)$

where $\texttt{weight'}(v_i, v_{i+1}) = \texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})$

Reweighted paths

Each each vertex $v \in V$ has a value $\mathtt{h}(v)$ - called the potential of v

we will pick these values carefully later

Consider the following path as an example. . .

$$v_1 \xrightarrow{-2} v_2 \xrightarrow{3} v_3 \xrightarrow{1} v_4 \xrightarrow{7} v_5 \xrightarrow{4} v_6$$

$\mathtt{h}(v_1) = 5$ $\mathtt{h}(v_2) = 2$ $\mathtt{h}(v_3) = 0$ $\mathtt{h}(v_4) = 1$ $\mathtt{h}(v_5) = 2$ $\mathtt{h}(v_6) = 4$

- *this path has length* 13

Reweight the whole graph using $\mathtt{weight'}$ *(we don't just reweight this path)*. . .

$$v_1 \xrightarrow{1} v_2 \xrightarrow{5} v_3 \xrightarrow{0} v_4 \xrightarrow{6} v_5 \xrightarrow{2} v_6$$

- *this path has length* $14 = 13 + 5 - 4 = 13 + h(v_1) - h(v_6)$

is this a coincidence?

where $\mathtt{weight'}(v_i, v_{i+1}) = \mathtt{weight}(v_i, v_{i+1}) + \mathtt{h}(v_i) - \mathtt{h}(v_{i+1})$

Reweighted paths

Consider an arbitrary path...

$$v_1 \xrightarrow{?} v_2 \xrightarrow{?} v_3 \xrightarrow{?} v_4 \xrightarrow{?} v_5 \xrightarrow{\cdots} v_k$$

$$\mathsf{h}(v_1) \qquad \mathsf{h}(v_2) \qquad \mathsf{h}(v_3) \qquad \mathsf{h}(v_4) \qquad \mathsf{h}(v_5) \qquad \mathsf{h}(v_k)$$

Reweighted paths

Consider an arbitrary path...

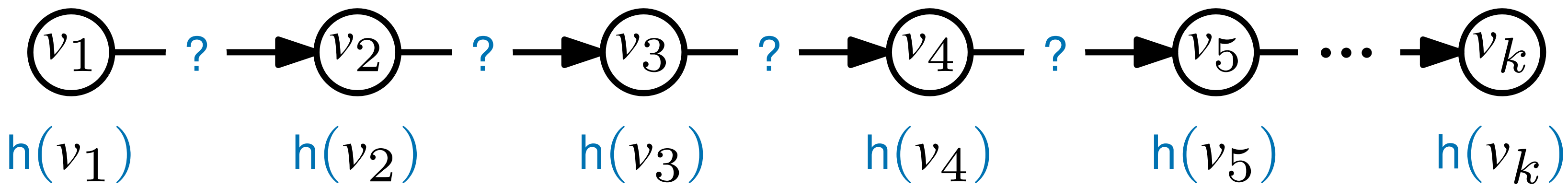
$$v_1 \xrightarrow{?} v_2 \xrightarrow{?} v_3 \xrightarrow{?} v_4 \xrightarrow{?} v_5 \xrightarrow{} \cdots \xrightarrow{} v_k$$

$$\mathtt{h}(v_1) \qquad \mathtt{h}(v_2) \qquad \mathtt{h}(v_3) \qquad \mathtt{h}(v_4) \qquad \mathtt{h}(v_5) \qquad \mathtt{h}(v_k)$$

In the original graph, the path length is $\displaystyle\sum_{i=1}^{k-1} \mathtt{weight}(v_i, v_{i+1})$

Reweighted paths

Consider an arbitrary path. . .



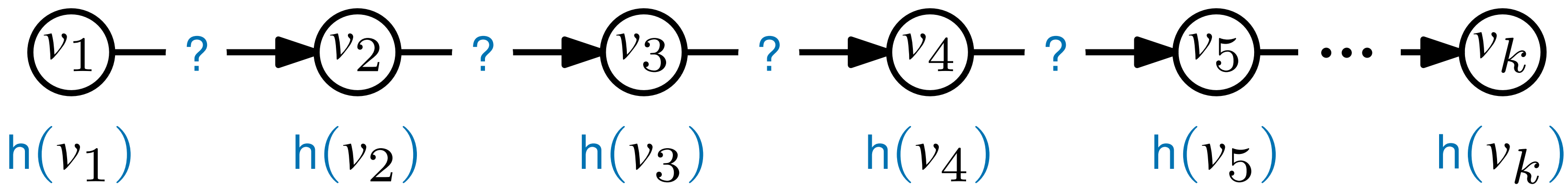
In the original graph, the path length is $\displaystyle\sum_{i=1}^{k-1} \texttt{weight}(v_i, v_{i+1})$

In the reweighted graph, the path length is

$$\sum_{i=1}^{k-1} \texttt{weight'}(v_i, v_{i+1})$$

Reweighted paths

Consider an arbitrary path...



In the original graph, the path length is $\sum_{i=1}^{k-1} \texttt{weight}(v_i, v_{i+1})$

In the reweighted graph, the path length is

$$\sum_{i=1}^{k-1} \texttt{weight'}(v_i, v_{i+1}) = \sum_{i=1}^{k-1} \big(\texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})\big)$$

Reweighted paths

Consider an arbitrary path...

$$v_1 \xrightarrow{\ ?\ } v_2 \xrightarrow{\ ?\ } v_3 \xrightarrow{\ ?\ } v_4 \xrightarrow{\ ?\ } v_5 \cdots \to v_k$$

$$\mathsf{h}(v_1) \quad \mathsf{h}(v_2) \quad \mathsf{h}(v_3) \quad \mathsf{h}(v_4) \quad \mathsf{h}(v_5) \quad \mathsf{h}(v_k)$$

In the original graph, the path length is $\displaystyle\sum_{i=1}^{k-1} \texttt{weight}(v_i, v_{i+1})$

In the reweighted graph, the path length is

$$\sum_{i=1}^{k-1} \texttt{weight'}(v_i, v_{i+1}) = \sum_{i=1}^{k-1} \Big(\texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})\Big)$$

$$= \sum_{i=1}^{k-1} \Big(\texttt{weight}(v_i, v_{i+1})\Big) + \mathsf{h}(v_1) - \mathsf{h}(v_k)$$

Reweighted paths

Consider an arbitrary path...

$$v_1 \xrightarrow{?} v_2 \xrightarrow{?} v_3 \xrightarrow{?} v_4 \xrightarrow{?} v_5 \to \cdots \to v_k$$

$$\mathsf{h}(v_1) \quad \mathsf{h}(v_2) \quad \mathsf{h}(v_3) \quad \mathsf{h}(v_4) \quad \mathsf{h}(v_5) \quad \mathsf{h}(v_k)$$

In the original graph, the path length is $\displaystyle\sum_{i=1}^{k-1} \texttt{weight}(v_i, v_{i+1})$

In the reweighted graph, the path length is

$$\sum_{i=1}^{k-1} \texttt{weight'}(v_i, v_{i+1}) = \sum_{i=1}^{k-1} \big(\texttt{weight}(v_i, v_{i+1}) + \mathsf{h}(v_i) - \mathsf{h}(v_{i+1})\big)$$

$$= \sum_{i=1}^{k-1} \big(\texttt{weight}(v_i, v_{i+1})\big) + \mathsf{h}(v_1) - \mathsf{h}(v_k)$$

So the weight of a path only changes by the potential values of the end points...

Reweighted shortest paths

Let the function $\mathtt{h}$ give a value $\mathtt{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\mathtt{weight'}(u,v) = \mathtt{weight}(u,v) + \mathtt{h}(u) - \mathtt{h}(v)$$

Reweighted shortest paths

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

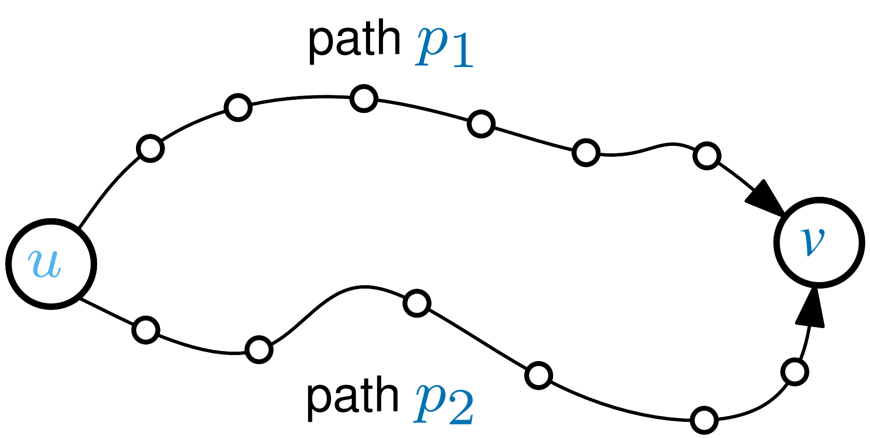
Reweighted shortest paths

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph



Reweighted shortest paths

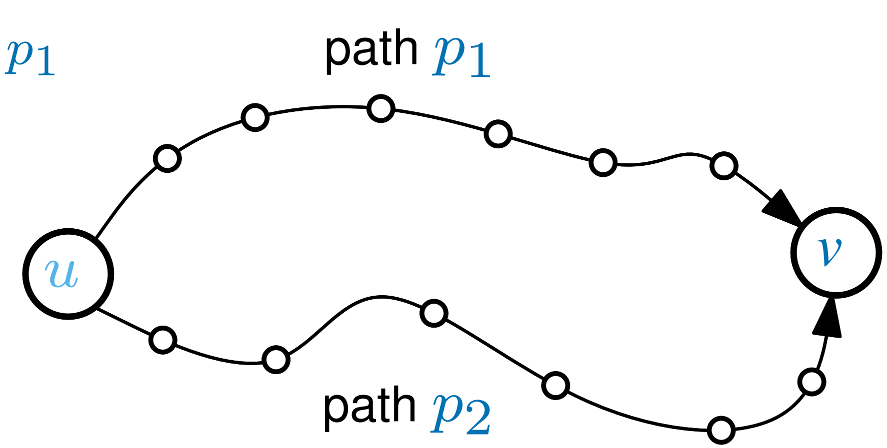
Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Let l_1 be the length of p_1
in the original graph



Reweighted shortest paths

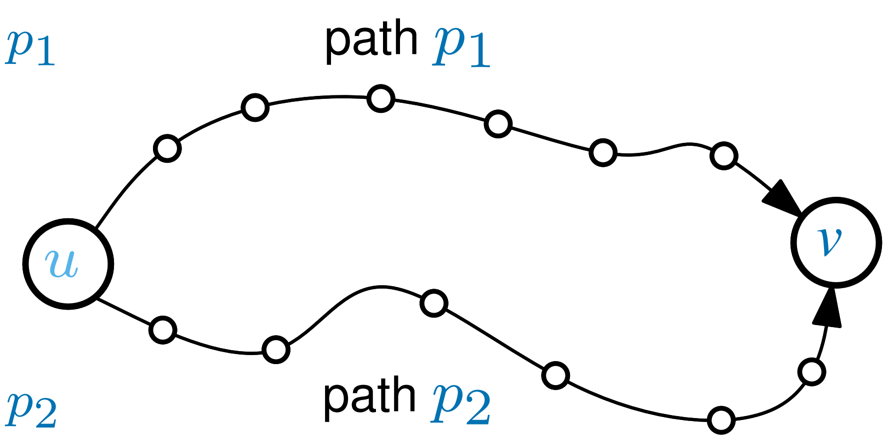
Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Let l_1 be the length of p_1
in the original graph



Let l_2 be the length of p_2
in the original graph

Reweighted shortest paths

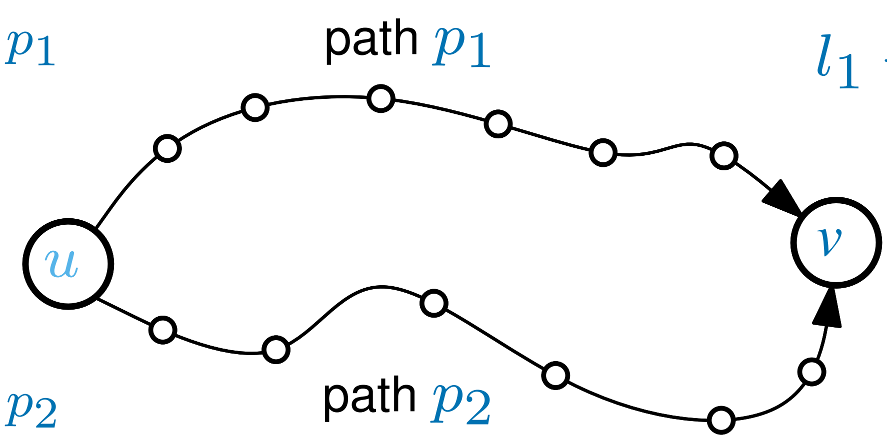
Let the function h give a value h(v) for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Let l_1 be the length of p_1
in the original graph



$l_1 + h(u) - h(v)$
is the length of p_1
in the reweighted graph

Let l_2 be the length of p_2
in the original graph

Reweighted shortest paths

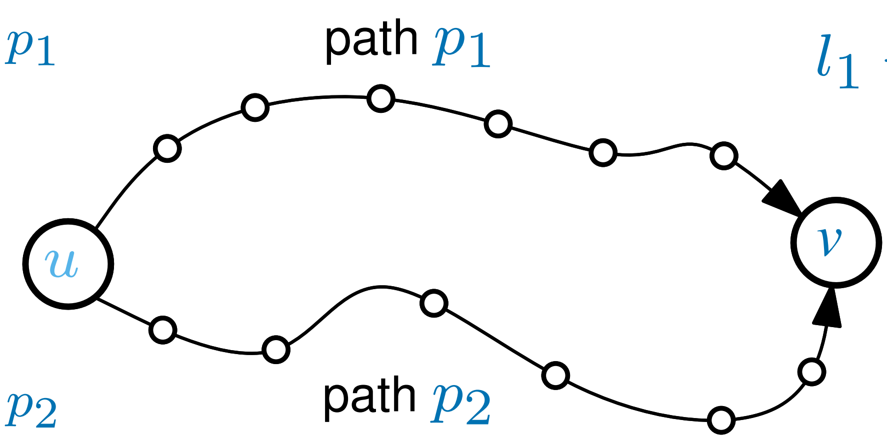
Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Let l_1 be the length of p_1
in the original graph



$l_1 + h(u) - h(v)$
is the length of p_1
in the reweighted graph

Let l_2 be the length of p_2
in the original graph

$l_2 + h(u) - h(v)$
is the length of p_2
in the reweighted graph

Reweighted shortest paths

Let the function h give a value $h(v)$ for each vertex $v \in V$

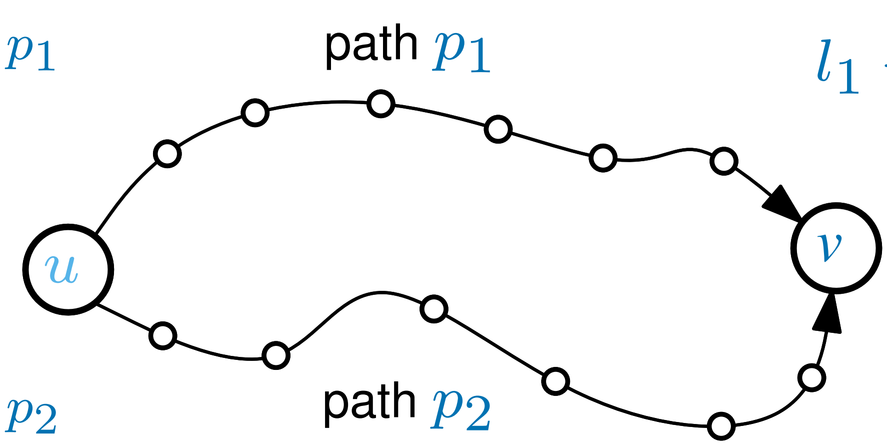
Change the weight of every edge (u,v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Let l_1 be the length of p_1
in the original graph

Let l_2 be the length of p_2
in the original graph



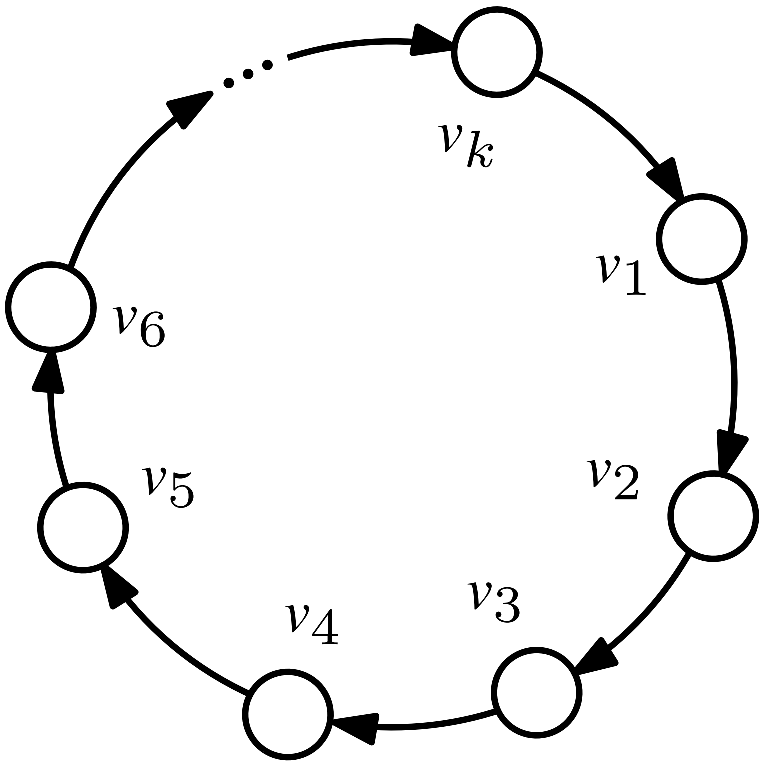
$l_1 + h(u) - h(v)$
is the length of p_1
in the reweighted graph

$l_2 + h(u) - h(v)$
is the length of p_2
in the reweighted graph

$l_1 \leqslant l_2$ if and only if

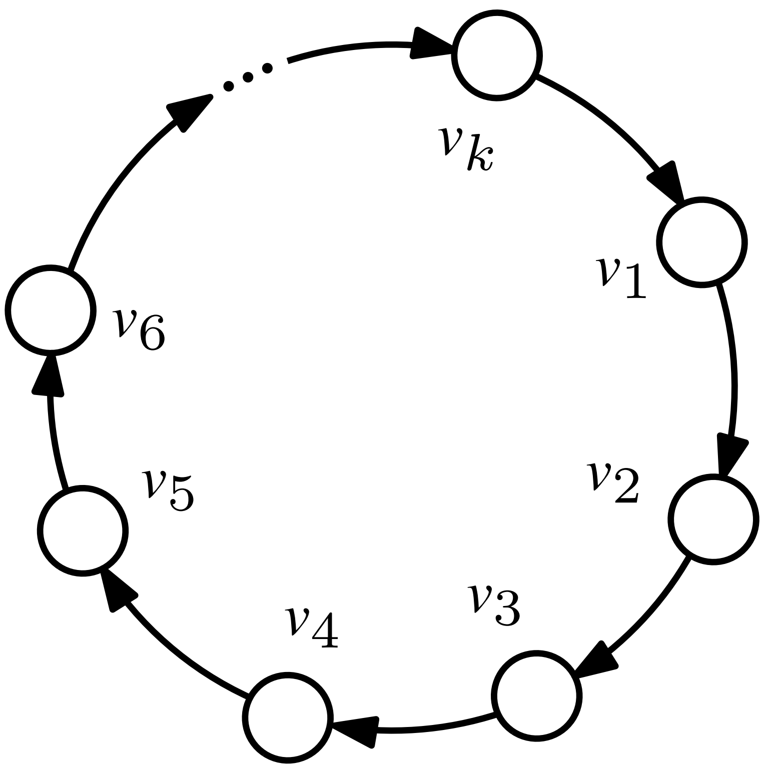
$$l_1 + h(u) - h(v) \leqslant l_2 + h(u) - h(v)$$

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

Reweighted negative cycles



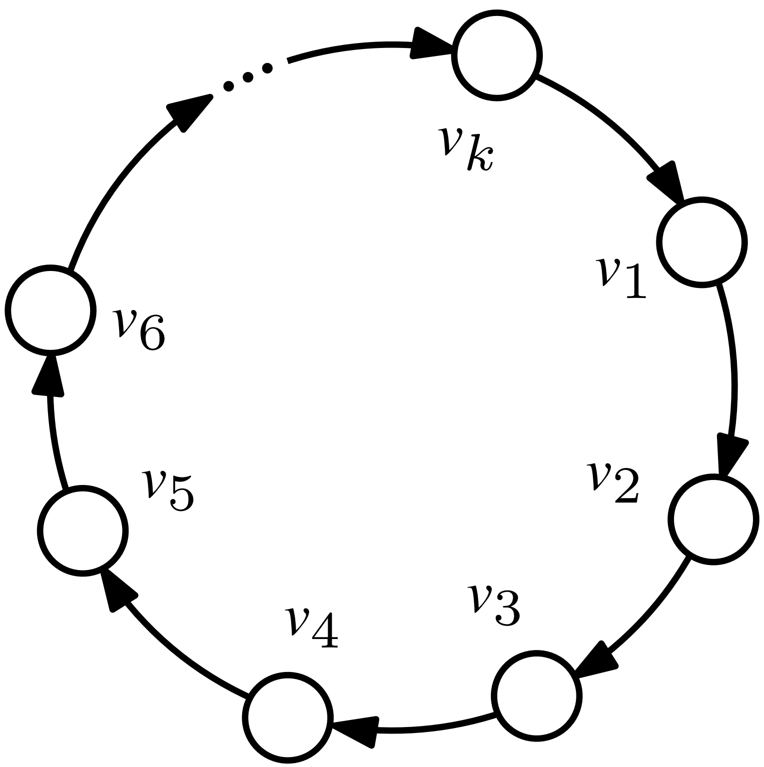
Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

The weight of this cycle in the original graph is,

$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

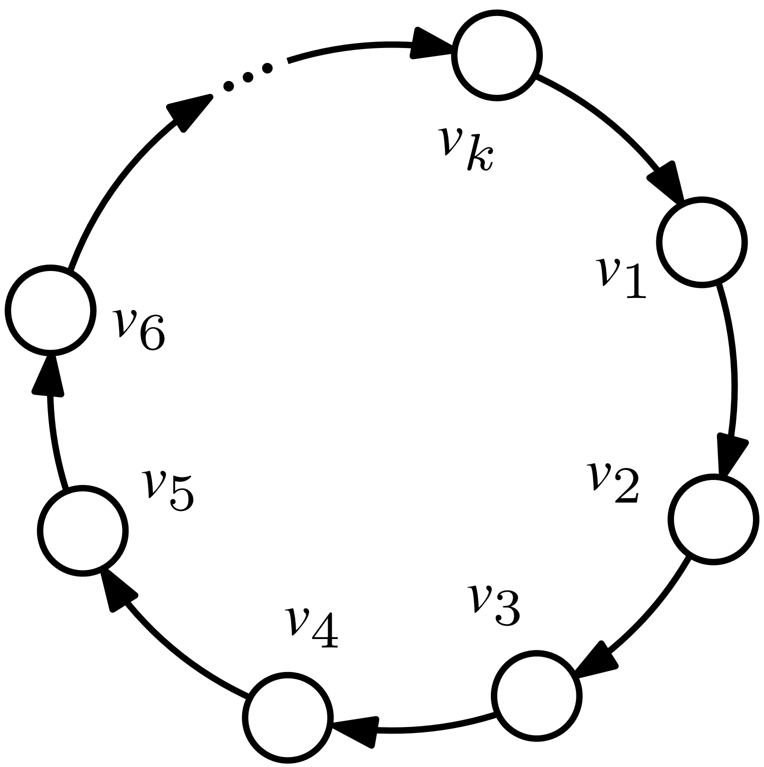
The weight of this cycle in the original graph is,

$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

The weight of this cycle in the reweighted graph is

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

The weight of this cycle in the original graph is,

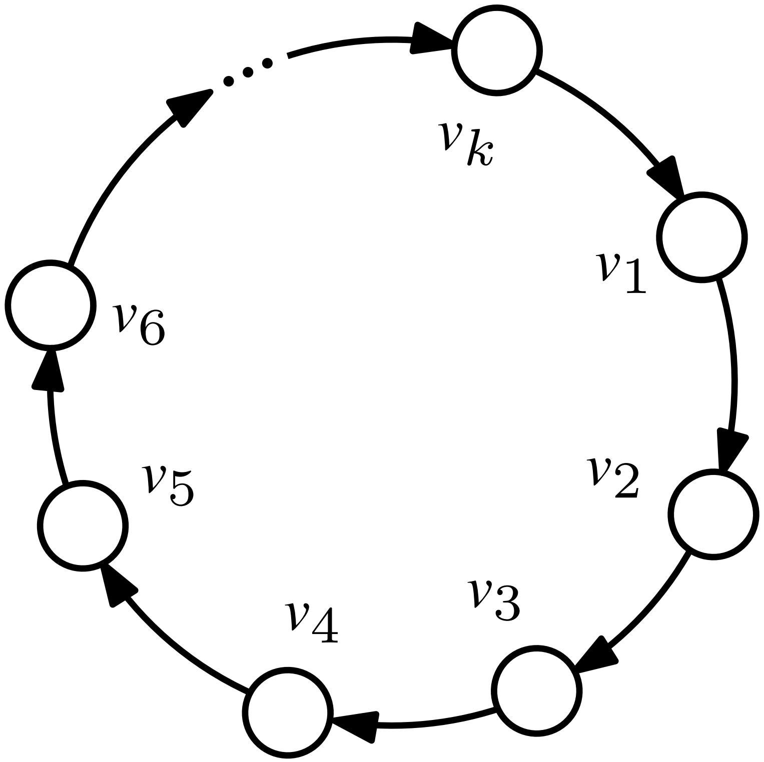
$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

The weight of this cycle in the reweighted graph is

$$\sum_{i=1}^{k} \texttt{weight'}(v_i, v_{i+1})$$

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

The weight of this cycle in the original graph is,

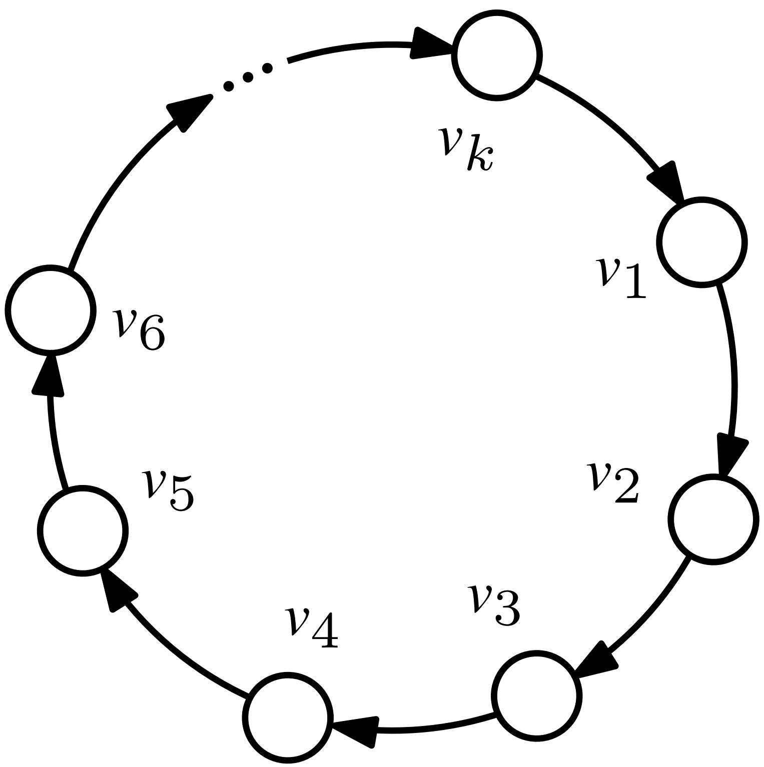
$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

The weight of this cycle in the reweighted graph is

$$\sum_{i=1}^{k} \texttt{weight'}(v_i, v_{i+1}) = \sum_{i=1}^{k} \big(\texttt{weight}(v_i, v_{i+1})\big) + \mathsf{h}(v_1) - \mathsf{h}(v_1)$$

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$

be a negative weight cycle

The weight of this cycle

in the original graph is,

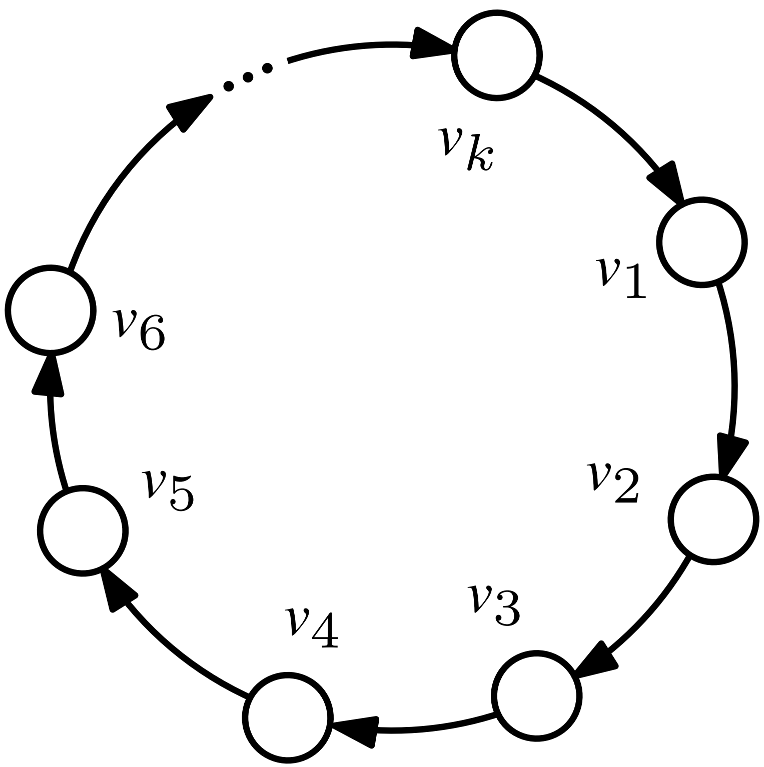
$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

The weight of this cycle in the reweighted graph is

$$\sum_{i=1}^{k} \texttt{weight'}(v_i, v_{i+1}) = \sum_{i=1}^{k} \big(\texttt{weight}(v_i, v_{i+1})\big)$$

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$

be a negative weight cycle

The weight of this cycle

in the original graph is,

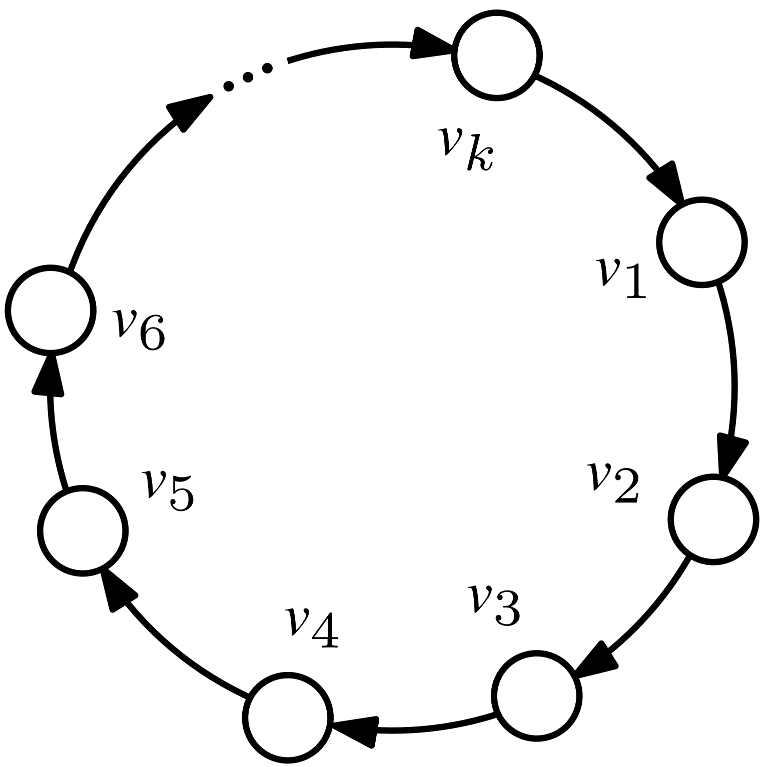
$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

The weight of this cycle in the reweighted graph is

$$\sum_{i=1}^{k} \texttt{weight'}(v_i, v_{i+1}) \quad = \sum_{i=1}^{k} \big(\texttt{weight}(v_i, v_{i+1})\big) \; < 0$$

Reweighted negative cycles



Let $v_1, v_2, \ldots v_k \in V$ be a negative weight cycle

The weight of this cycle in the original graph is,

$$\sum_{i=1}^{k} \texttt{weight}(v_i, v_{i+1}) < 0$$

(where $v_{k+1} = v_1$)

The weight of this cycle in the reweighted graph is

$$\sum_{i=1}^{k} \texttt{weight'}(v_i, v_{i+1}) = \sum_{i=1}^{k} \big(\texttt{weight}(v_i, v_{i+1})\big) < 0$$

So reweighting doesn't affect negative cycles

Reweighting summary

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Reweighting summary

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma Any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Reweighting summary

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma Any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Fact If ℓ is the length of a path from u to v in the original graph
$\ell + \mathsf{h}(u) - \mathsf{h}(v)$ is its length in the reweighted graph

Reweighting summary

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma Any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Fact If ℓ is the length of a path from u to v in the original graph
$\ell + \mathsf{h}(u) - \mathsf{h}(v)$ is its length in the reweighted graph

Fact A cycle has negative weight in the original graph
if and only if it has negative weight in the reweighted graph

Reweighting summary

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma Any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

Fact If ℓ is the length of a path from u to v in the original graph
$\ell + \mathsf{h}(u) - \mathsf{h}(v)$ is its length in the reweighted graph

Fact A cycle has negative weight in the original graph
if and only if it has negative weight in the reweighted graph

*So if we solve the all-pairs shortest paths problem on the reweighted graph. . .
we can recover the shortest path lengths for the original graph*

Reweighting summary

Let the function h give a value $\mathsf{h}(v)$ for each vertex $v \in V$

Change the weight of every edge (u, v) to be

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Lemma Any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph

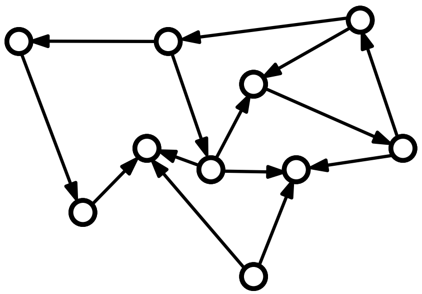
Fact If ℓ is the length of a path from u to v in the original graph
$\ell + \mathsf{h}(u) - \mathsf{h}(v)$ is its length in the reweighted graph

Fact A cycle has negative weight in the original graph
if and only if it has negative weight in the reweighted graph

So if we solve the all-pairs shortest paths problem on the reweighted graph. . .
we can recover the shortest path lengths for the original graph

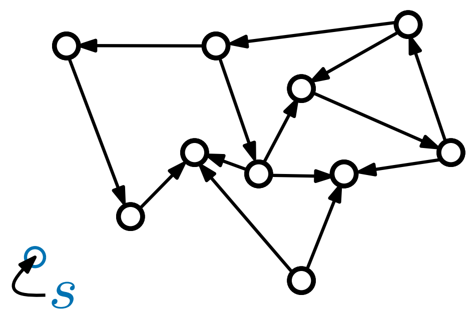
To take advantage of this, we need to make all the edge weights *non-negative*

How do we choose h?



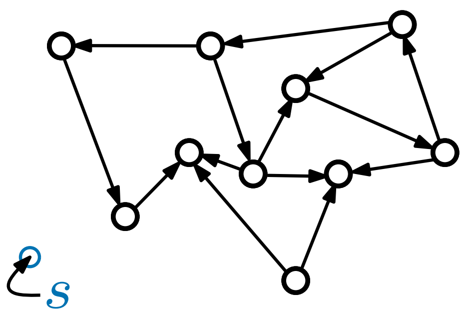
We first add one additional vertex called s to the original graph

How do we choose h?



We first add one additional vertex called s to the original graph

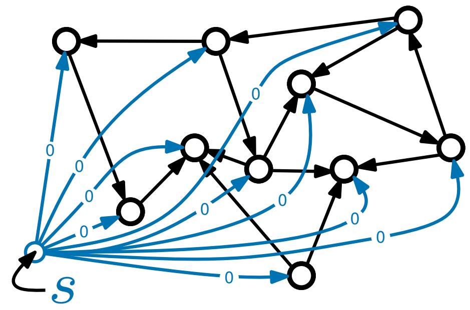
How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

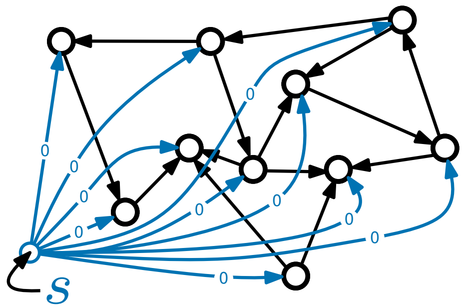
How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

How do we choose h?

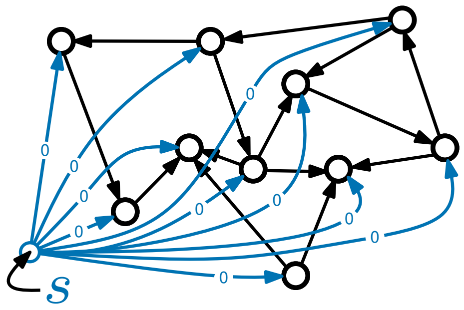


We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

How do we choose h?



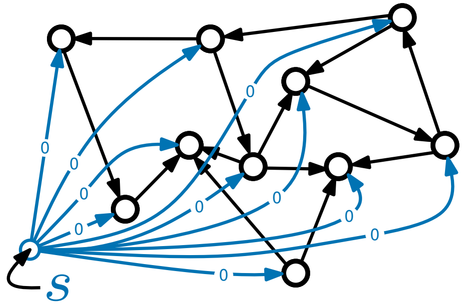
We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

How do we choose h?



We first add one additional vertex called s to the original graph

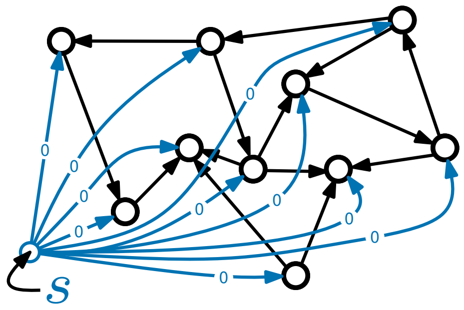
We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

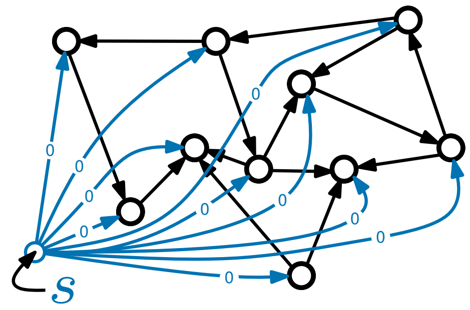
This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

Warning: If the original graph contains a negative weight cycle,

$\delta(s, v)$ may be undefined

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

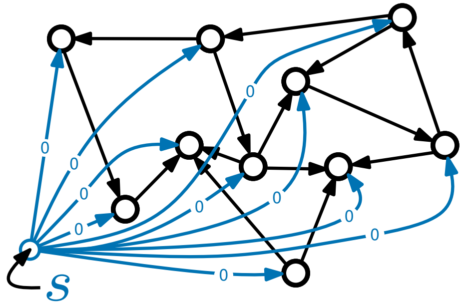
Warning: If the original graph contains a negative weight cycle,

$\delta(s, v)$ may be undefined

JOHNSON's *algorithm will detect this and abort*

Let's continue under the assumption that there is no negative weight cycle

How do we choose h?



We first add one additional vertex called s to the original graph

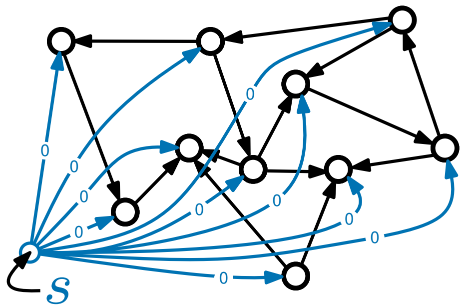
We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

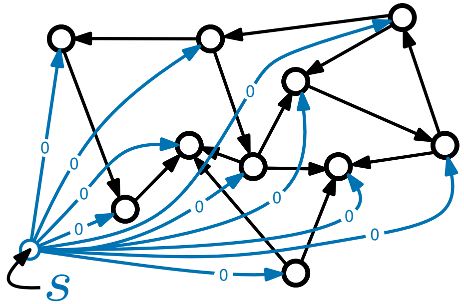
each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

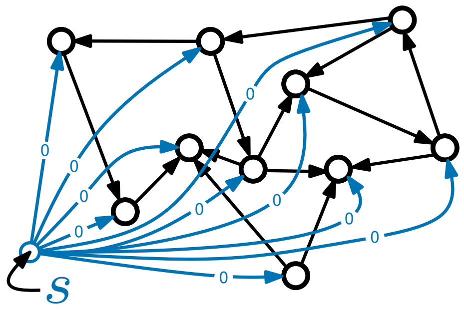
This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathrm{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

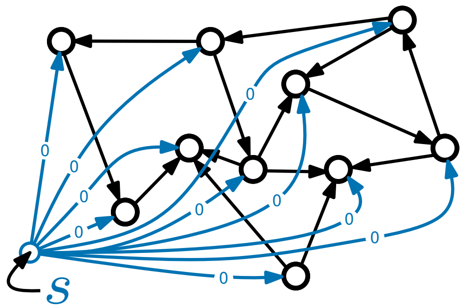
For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

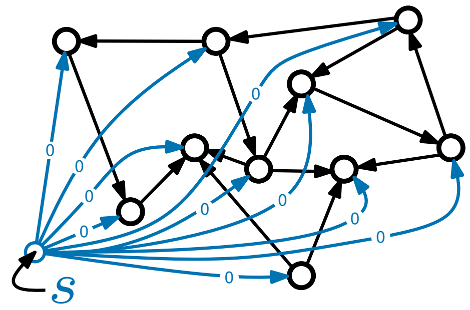
we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

This follows because there is a path from s to v via u

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

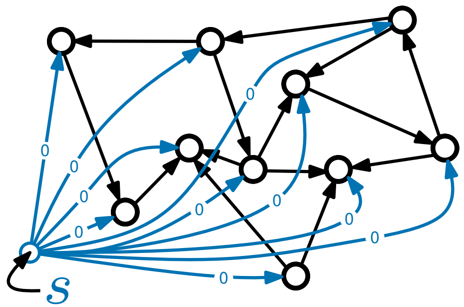
Consider any edge $(u, v) \in E$

The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

This follows because there is a path from s to v via u

with length $\delta(s, u) + \texttt{weight}(u, v)$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

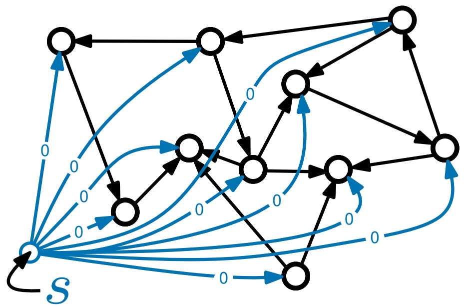
The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

This follows because there is a path from s to v via u

with length $\delta(s, u) + \texttt{weight}(u, v)$

so the shortest path can't be longer

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

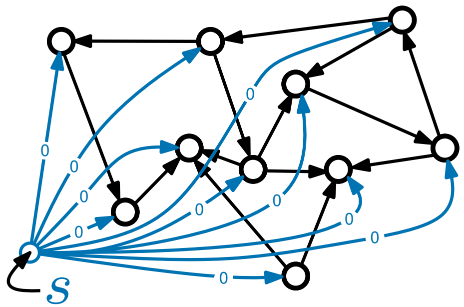
For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

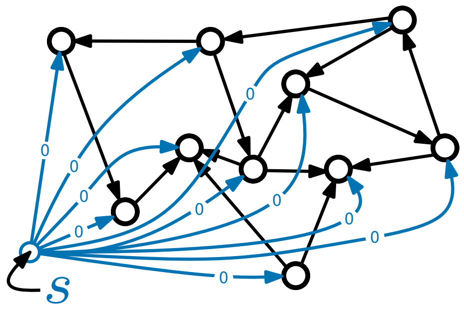
we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

Rearranging we have, $\texttt{weight}(u, v) + \delta(s, u) - \delta(s, v) \geqslant 0$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

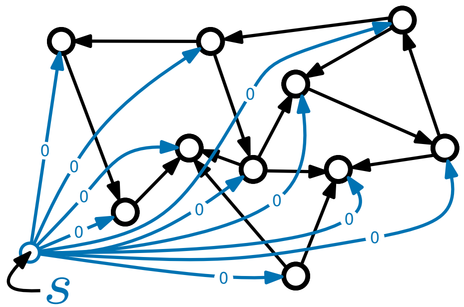
The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

Rearranging we have, $\texttt{weight}(u, v) + \delta(s, u) - \delta(s, v) \geqslant 0$

The new weight of an edge (u, v) then becomes

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

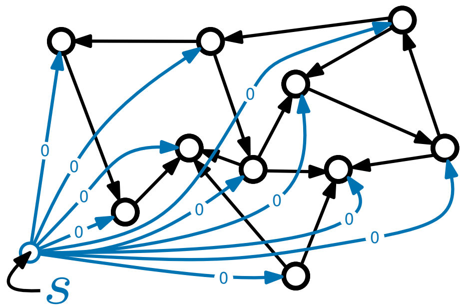
The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

Rearranging we have, $\texttt{weight}(u, v) + \delta(s, u) - \delta(s, v) \geqslant 0$

The new weight of an edge (u, v) then becomes

$$\begin{aligned}\texttt{weight'}(u,v) &= \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)\\ &= \texttt{weight}(u, v) + \delta(s, u) - \delta(s, v) \geqslant 0\end{aligned}$$

How do we choose h?



We first add one additional vertex called s to the original graph

We also add an edge (s, v) from s to each other vertex $v \in V$

each of these edges has weight 0

This does not introduce any new negative weight cycles

For each v, let $\delta(s, v)$ denote the length of the shortest path from s to v

we then define $\mathsf{h}(v)$ to equal $\delta(s, v)$

Consider any edge $(u, v) \in E$

The key observation is that $\delta(s, v) \leqslant \delta(s, u) + \texttt{weight}(u, v)$

Rearranging we have, $\texttt{weight}(u, v) + \delta(s, u) - \delta(s, v) \geqslant 0$

The new weight of an edge (u, v) then becomes

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$
$$= \texttt{weight}(u, v) + \delta(s, u) - \delta(s, v) \geqslant 0$$

So all the reweighted edge weights are non-negative

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- *this calculates the shortest path lengths* $\delta(s, v)$ *for all* v

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- this calculates the shortest path lengths $\delta(s, v)$ for all v

if there is a negative weight cycle in the original graph, it will be detected here

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- this calculates the shortest path lengths $\delta(s, v)$ *for all* v

if there is a negative weight cycle in the original graph, it will be detected here

Step 4: Reweight each edge $(u, v) \in E$ so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

where for all v, $h(v) = \delta(s, v)$

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- this calculates the shortest path lengths $\delta(s, v)$ *for all* v

if there is a negative weight cycle in the original graph, it will be detected here

Step 4: Reweight each edge $(u, v) \in E$ so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

where for all v, $h(v) = \delta(s, v)$

Step 5: For each vertex $u \in V$, run DIJKSTRA's algorithm with source $s = u$.

- this calculates the shortest path lengths $\delta'(u, v)$ *for all* u, v

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- this calculates the shortest path lengths $\delta(s, v)$ *for all* v

if there is a negative weight cycle in the original graph, it will be detected here

Step 4: Reweight each edge $(u, v) \in E$ so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

where for all v, $h(v) = \delta(s, v)$

Step 5: For each vertex $u \in V$, run DIJKSTRA's algorithm with source $s = u$.

- this calculates the shortest path lengths $\delta'(u, v)$ *for all* u, v

these are the shortest path lengths in the reweighted graph

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- *this calculates the shortest path lengths* $\delta(s, v)$ *for all* v

if there is a negative weight cycle in the original graph, it will be detected here

Step 4: Reweight each edge $(u, v) \in E$ so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

where for all v, $h(v) = \delta(s, v)$

Step 5: For each vertex $u \in V$, run DIJKSTRA's algorithm with source $s = u$.

- *this calculates the shortest path lengths* $\delta'(u, v)$ *for all* u, v

these are the shortest path lengths in the reweighted graph

Step 6: For each pair of vertices $u, v \in V$, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

JOHNSON's algorithm

We can now piece together JOHNSON's algorithm which operates as follows:

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

- *this calculates the shortest path lengths* $\delta(s, v)$ *for all* v

if there is a negative weight cycle in the original graph, it will be detected here

Step 4: Reweight each edge $(u, v) \in E$ so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

where for all v, $h(v) = \delta(s, v)$

Step 5: For each vertex $u \in V$, run DIJKSTRA's algorithm with source $s = u$.

- *this calculates the shortest path lengths* $\delta'(u, v)$ *for all* u, v

these are the shortest path lengths in the reweighted graph

Step 6: For each pair of vertices $u, v \in V$, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

these are the shortest path lengths in the original graph

Time Complexity

How long does all this take?

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

Step 4: Reweight each edge (u, v) so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

$O(1)$ time

Step 1: Add one additional vertex called s to the original graph

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

Step 4: Reweight each edge (u, v) so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

$O(1)$ time

Step 1: Add one additional vertex called s to the original graph

$O(|V|)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0

Step 3: Run the BELLMAN-FORD algorithm with source s

Step 4: Reweight each edge (u, v) so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

$O(1)$ time

Step 1: Add one additional vertex called s to the original graph

$O(|V|)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0

$O(|V||E|)$ time

Step 3: Run the BELLMAN-FORD algorithm with source s

Step 4: Reweight each edge (u, v) so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

Step 6: For each pair of vertices u, v, compute

$$\delta(u,v) = \delta'(u,v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

$O(1)$ time

Step 1: Add one additional vertex called s to the original graph

$O(|V|)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0

$O(|V||E|)$ time

Step 3: Run the BELLMAN-FORD algorithm with source s

$O(|E|)$ time

Step 4: Reweight each edge (u, v) so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

Step 1: Add one additional vertex called s to the original graph — $O(1)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0 — $O(|V|)$ time

Step 3: Run the BELLMAN-FORD algorithm with source s — $O(|V||E|)$ time

Step 4: Reweight each edge (u, v) so that, — $O(|E|)$ time

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$. — $O(|E| \log |V|)$ time per iteration *(using a binary heap)*

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

Step 1: Add one additional vertex called s to the original graph

$O(1)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0

$O(|V|)$ time

Step 3: Run the BELLMAN-FORD algorithm with source s

$O(|V||E|)$ time

Step 4: Reweight each edge (u, v) so that,

$O(|E|)$ time

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

$O(|V||E| \log |V|)$ time overall

(using a binary heap)

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

Time Complexity

How long does all this take?

Step 1: Add one additional vertex called s to the original graph

$O(1)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0

$O(|V|)$ time

Step 3: Run the BELLMAN-FORD algorithm with source s

$O(|V||E|)$ time

Step 4: Reweight each edge (u, v) so that,

$O(|E|)$ time

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

$O(|V||E| \log |V|)$ time overall

(using a binary heap)

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

$O(|V|^2)$ time

(The previous version of this slide said $O(|E|)$ *for* **Step 6** *which was a typo)*

Time Complexity

How long does all this take?

$O(1)$ time

Step 1: Add one additional vertex called s to the original graph

$O(|V|)$ time

Step 2: For each vertex, add an edge (s, v) with weight 0

$O(|V||E|)$ time

Step 3: Run the BELLMAN-FORD algorithm with source s

$O(|E|)$ time

Step 4: Reweight each edge (u, v) so that,

$$\texttt{weight'}(u,v) = \texttt{weight}(u,v) + \mathsf{h}(u) - \mathsf{h}(v)$$

Step 5: For each vertex u, run DIJKSTRA's algorithm with source $s = u$.

$O(|V||E|\log|V|)$ time overall

(using a binary heap)

Step 6: For each pair of vertices u, v, compute

$$\delta(u, v) = \delta'(u, v) + \mathsf{h}(v) - \mathsf{h}(u)$$

$O(|V|^2)$ time

(The previous version of this slide said $O(|E|)$ for **Step 6** *which was a typo)*

So the overall time complexity is $O(|V||E|\log|V|)$

see next slide

This is the complexity for (strongly) connected graphs (actually any graph without isolated vertices)...

where every pair of vertices is connected

In such graphs we have that $|E| > |V|/2$ so $O(|V|^2) = O(|V||E|)$

For unconnected graphs there is a (non-examinable and fiddly) fix...

Johnson's algorithm on graphs with $|E| < |V|/2$

This slide contains non-examinable material and was added post-lecture.

It covers the corner case that Johnson's algorithm is run on a graph with $|E| < |V|/2$.

The simple answer: In this case, the algorithm (as discussed) takes $O(|V|^2 + |V||E|\log|V|)$ time

We can't "hide" the $O(|V|^2)$ term because $|V|$ could be much much larger than $|E|$

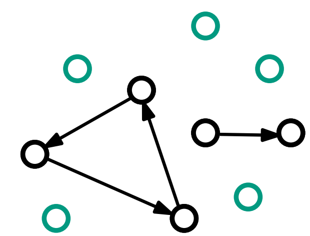
A more convoluted answer: With a simple preprocessing step, we can improve the time complexity to

$O(|V|^2 + |V||E|\log|V|)$ for this case

Sketch Proof

We define a vertex to be *isolated* if it contains no incoming and no outgoing edges.

(in the diagram the ○ are isolated and the ○ are not)



Add a new step to the start of Johnson's algorithm,

Step 0: Delete every isolated vertex ← $O(|V| + |E|)$ time

In the new graph $|E'| = |E|$, $|V'| \leqslant |V|$ and $|E'| \geqslant |V'|/2$

each remaining vertex is at one end of at least one edge

Therefore the modified algorithm takes

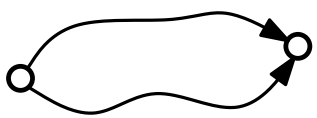
$$O(|V| + |E| + |V'||E'|\log|V'|) = O(|V||E|\log|V|) \text{ time}$$

Notice that the isolated vertices have distince ∞ to everywhere so deleting them doesn't change anything

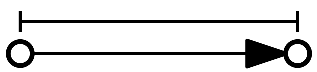
JOHNSON's algorithm summary

The overall approach taken by JOHNSON's algorithm
is to *reweight* the edges in the graph
(using new weights picked using BELLMAN-FORD*)*

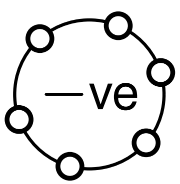
The key properties of the reweighted graph are:



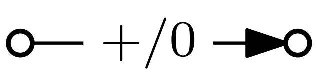
Lemma Any path is a shortest path in the original graph
if and only if it is a shortest path in the reweighted graph



Fact The length of a shortest path in the original graph
can be calculated from its length in the reweighted graph



Fact A cycle has negative weight in the original graph
if and only if it has negative weight in the reweighted graph



Fact If there are no negative weight cycles in the original graph,
all of the edges in the reweighted graph have non-negative weights

After reweighting, all-pairs shortest paths are calculated
using DIJSKTRAS algorithm, once with each vertex v as the source

Overall this takes $O(|V||E|\log|V|)$ time

Shortest paths algorithms: the summary

To compute **single source** shortest paths in a directed graph which is/has:

unweighted: Use Breadth First Search in $O(|V| + |E|)$ time

non-negative edge weights: Use DIJKSTRA's algorithm
which takes $O((|V| + |E|) \log |V|)$ time *(when implemented using a binary heap)*

positive and negative edge weights: Use BELLMAN-FORD which takes $O(|V||E|)$ time

To compute **all-pairs** shortest paths in a directed graph which is/has:

unweighted: Use Breadth First Search once for each vertex in $O(|V|^2 + |V||E|)$ time

non-negative edge weights: Use DIJKSTRA's algorithm once for each vertex,
which takes $O(|V||E| \log |V|)$ time *(when implemented using a binary heap)*

positive and negative edge weights: Use JOHNSON'S algorithm
which takes $O(|V||E| \log |V|)$ time
(when DIJKSTRA's algorithm is implemented using a binary heap)