

DSC 40B

***Lecture 25 : Shortest
Paths in Weighted
Graphs. Bellman-Ford***

Bellman-Ford

Bellman-Ford

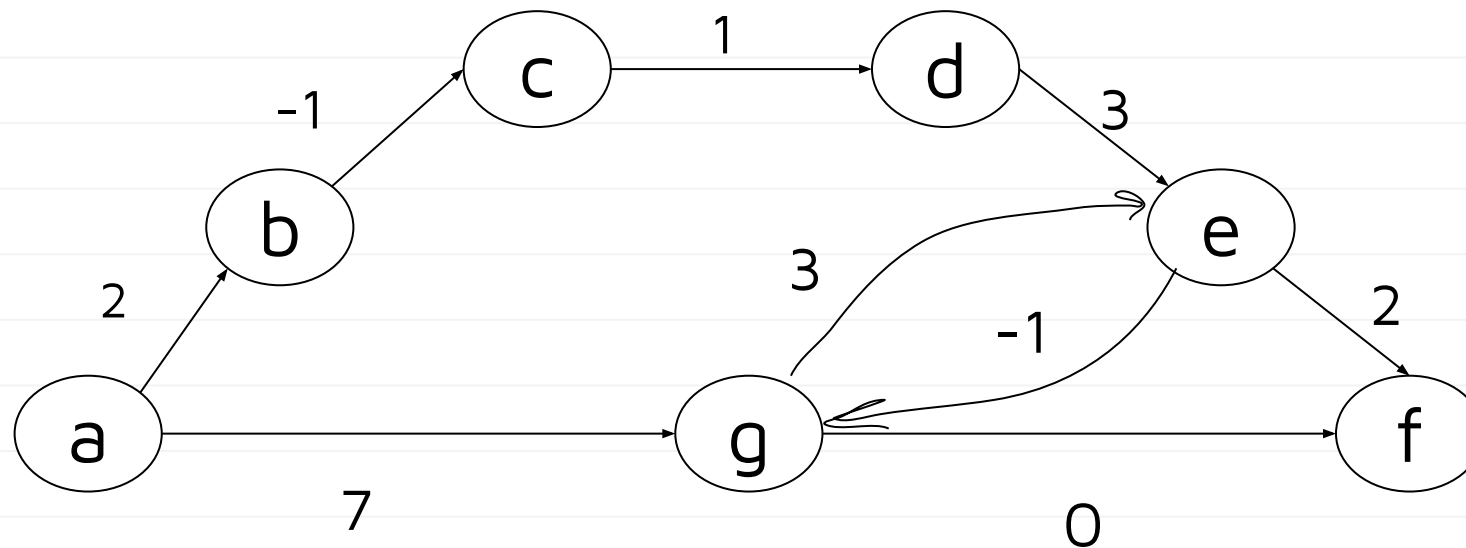
- Dijkstra's algorithm works only with **positive** weights.
- Bellman-Ford allows us to find shortest distances in the graphs with negative weights as well.

Intuition

- Shortest paths that have many edges are “harder” to discover.
 - May require many updates.
- Shortest paths that have few edges are “easier” to discover.
- Once we’ve discovered all of the shortest paths with few edges, it makes it easier to discover the shortest paths with more edges.

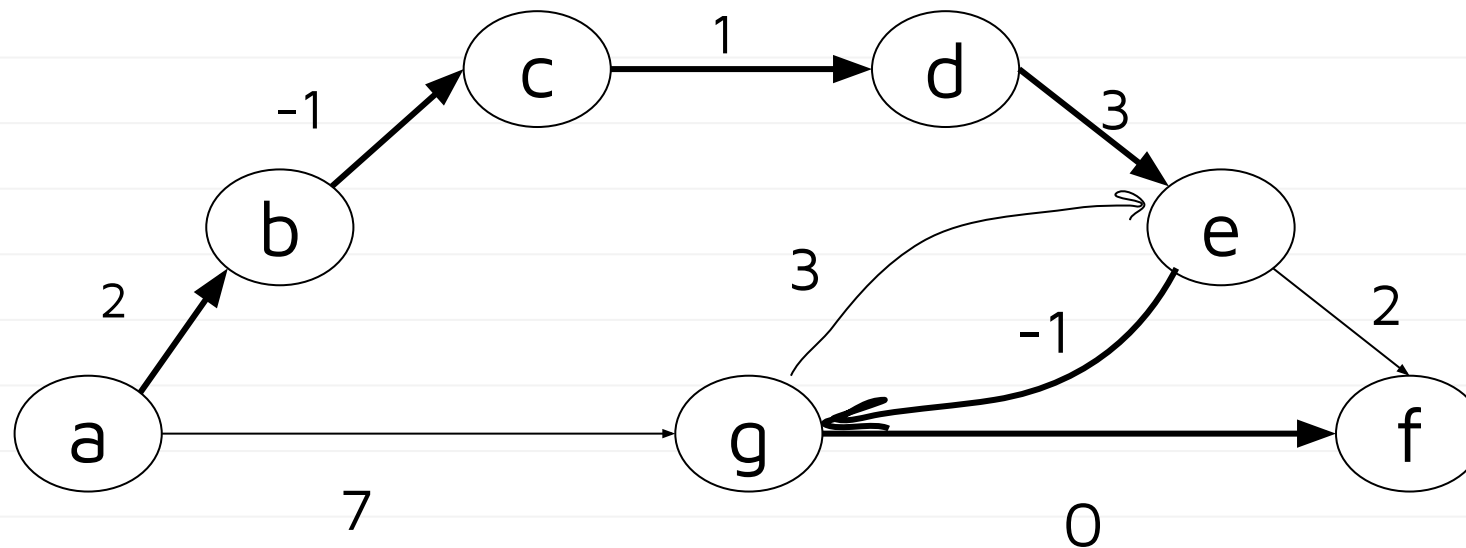
Example

What is the shortest path from **a** to **f**?



Example

What is the shortest path from **a** to **f**?

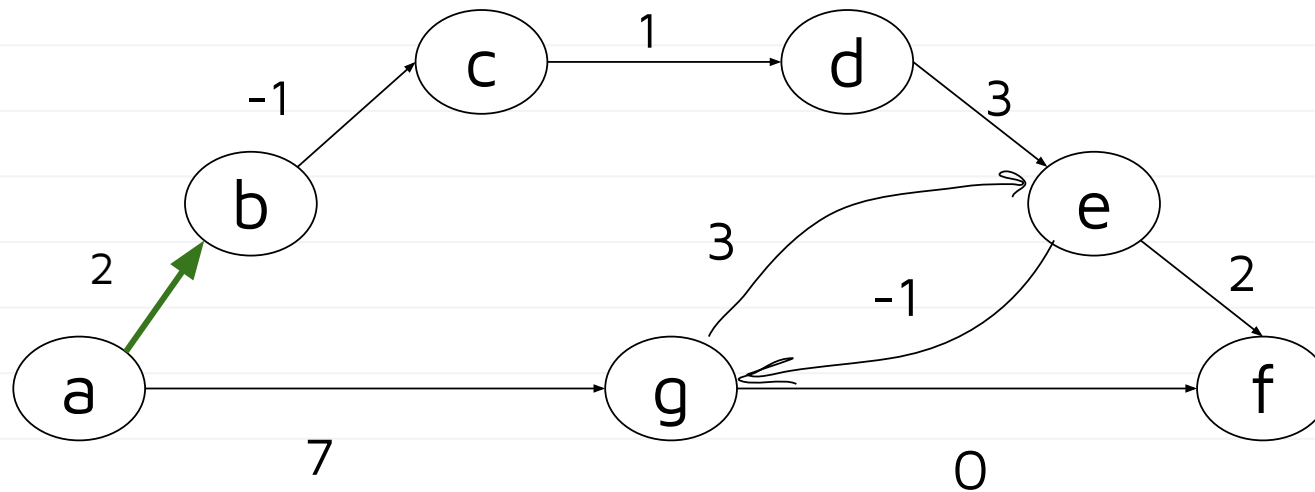


Path: a, b, c, d, e, g, f

Length: $2 - 1 + 1 + 3 - 1 + 0 = 4$

Updating All Edges

- Suppose we update all of the edges, one by one.
- Then all nodes whose shortest path from s has only **one** edge are **guaranteed** to be estimated correctly.



Loop Invariant

- **One** iteration: update *all* edges in arbitrary order.
- **Loop invariant:** After α iterations, all nodes whose shortest path from s has $\leq \alpha$ edges are **guaranteed** to be estimated correctly.

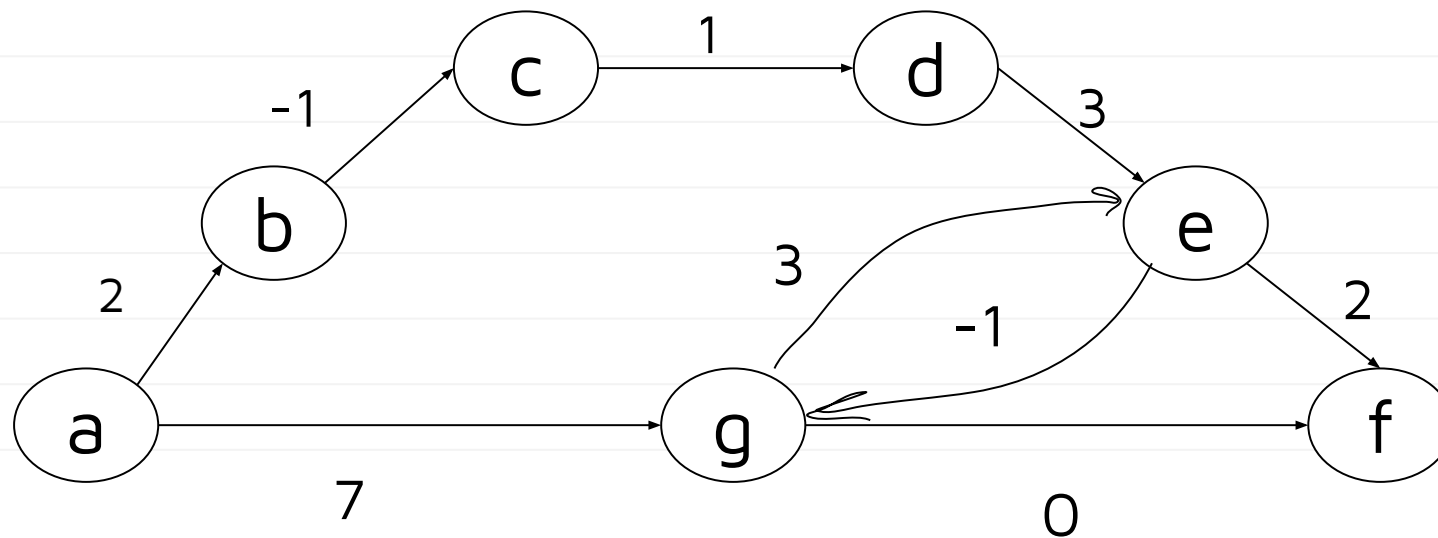
The Bellman-Ford Algorithm

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(?):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),

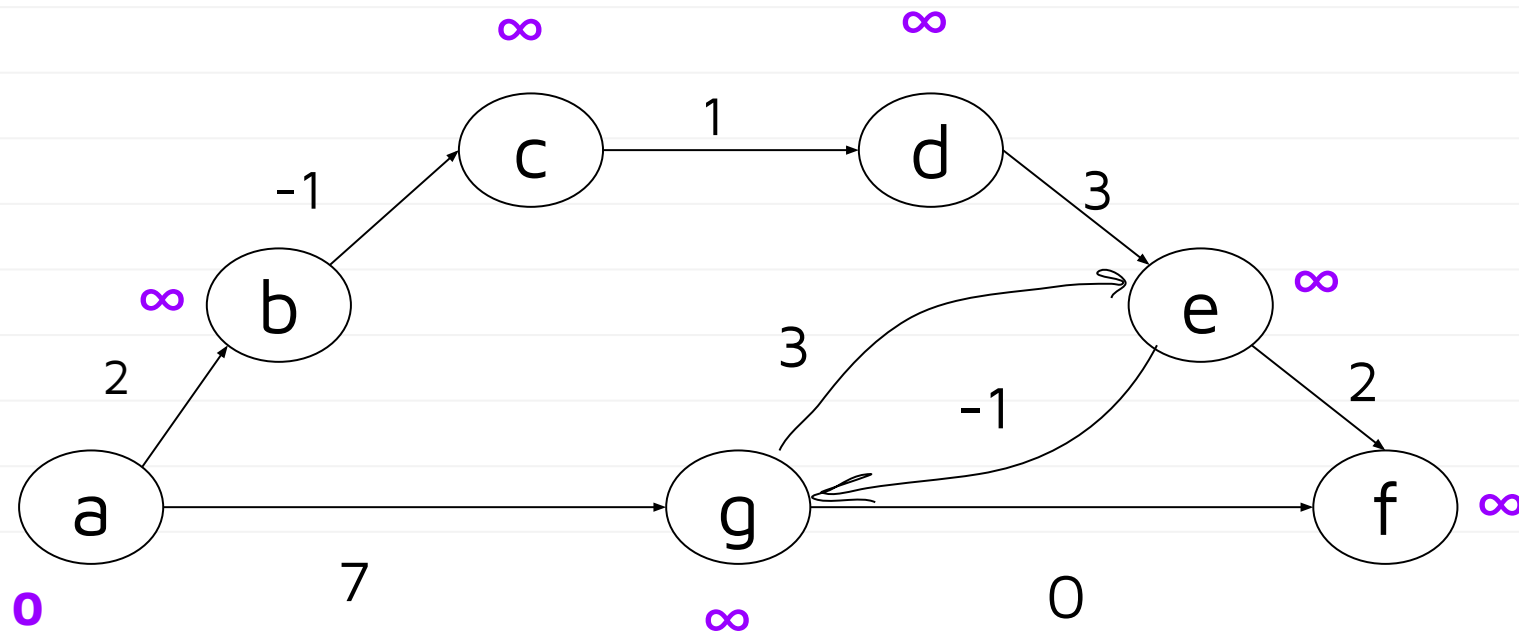
(g, e), (d, e), (e, f), (a, g)



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),

(g, e), (d, e), (e, f), (a, g)

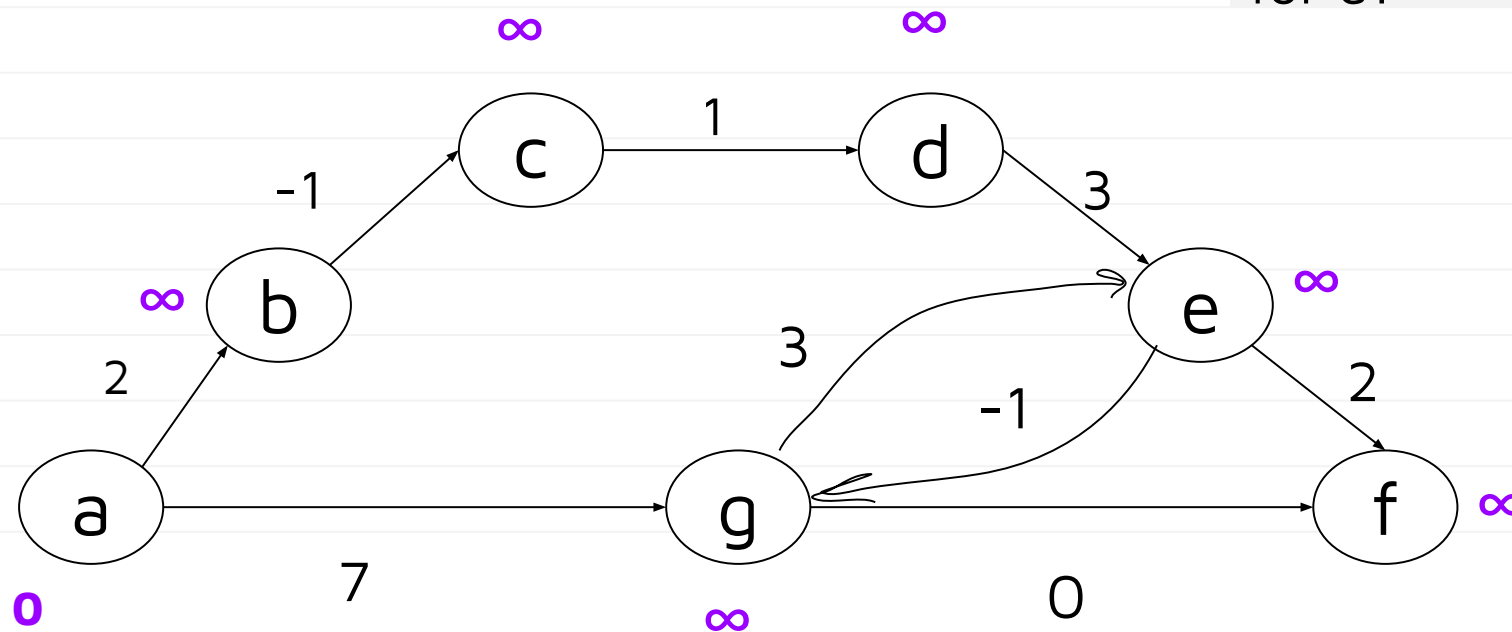


Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),

(g, e), (d, e), (e, f), (a, g)

Do we update distance
for d?

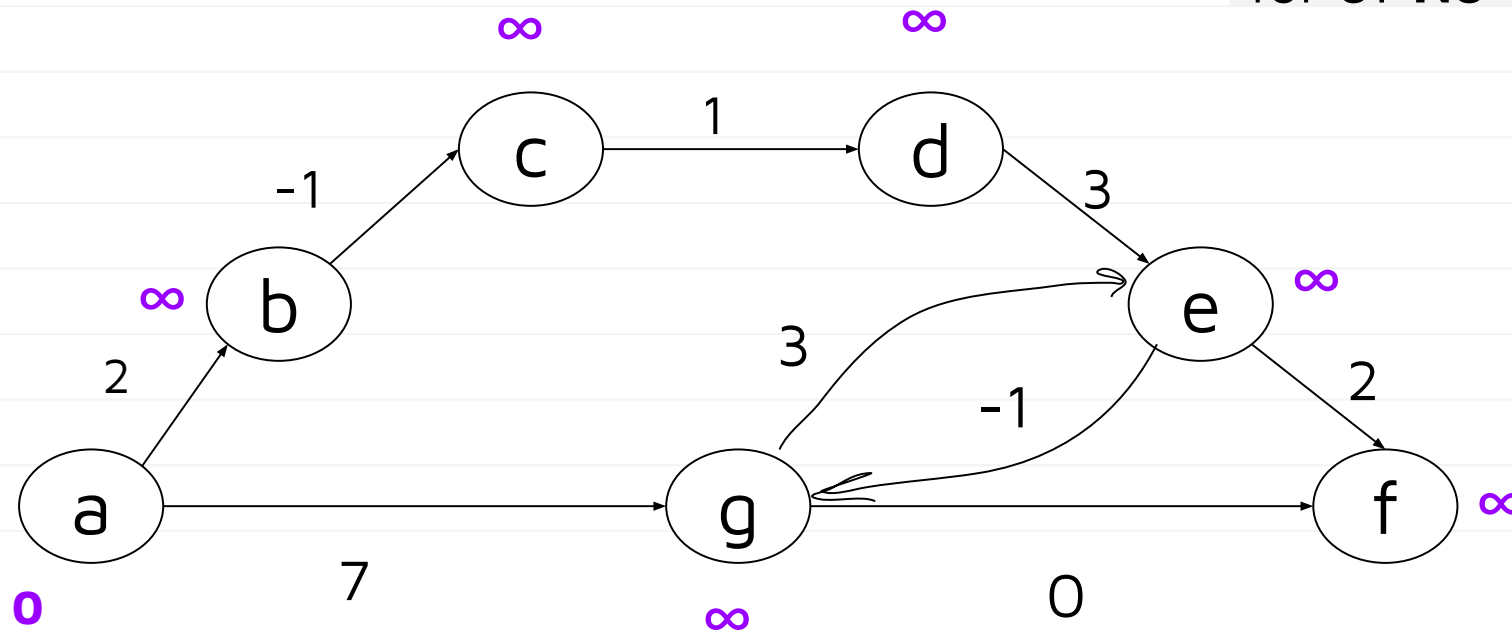


Suppose graph.edges returns edges in following order:

~~(c, d)~~, (a, b), (b, c), (g, f), (e, g),

(g, e), (d, e), (e, f), (a, g)

Do we update distance for d? **No**

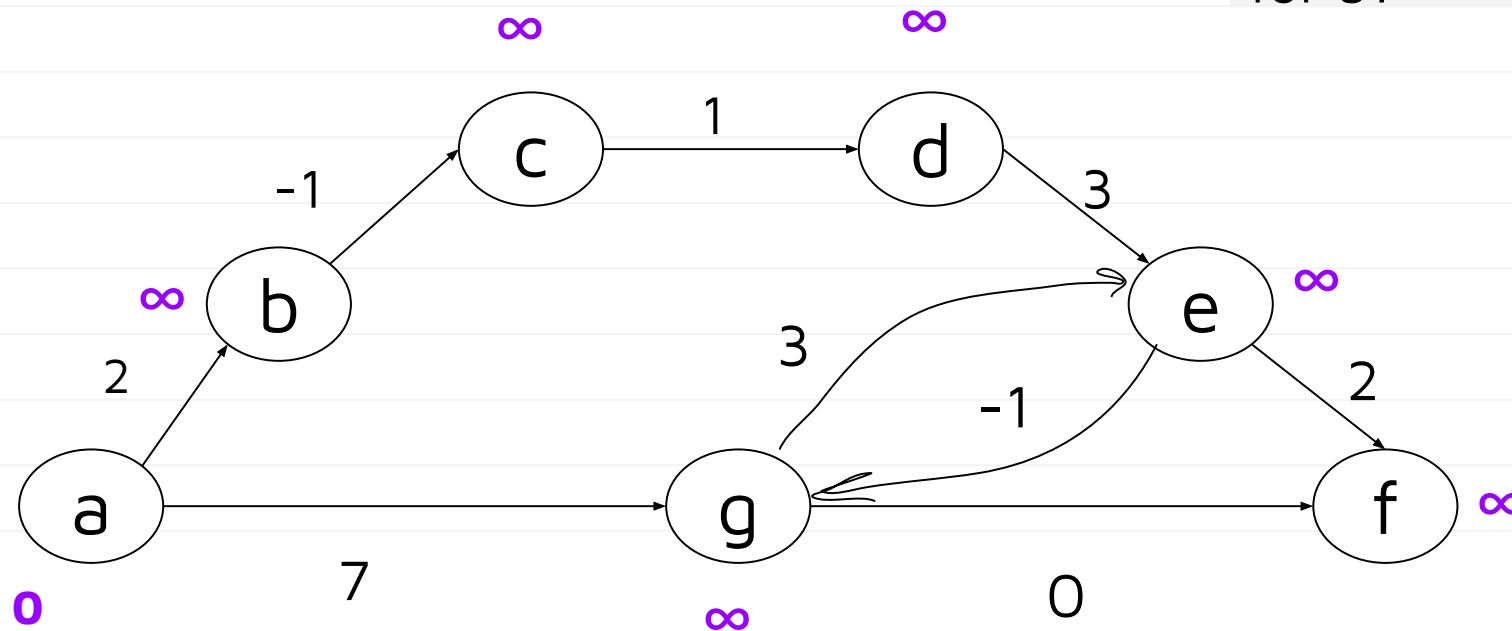


Suppose graph.edges returns edges in following order:

~~(e, d)~~, (a, b), (b, c), (g, f), (e, g),

(g, e), (d, e), (e, f), (a, g)

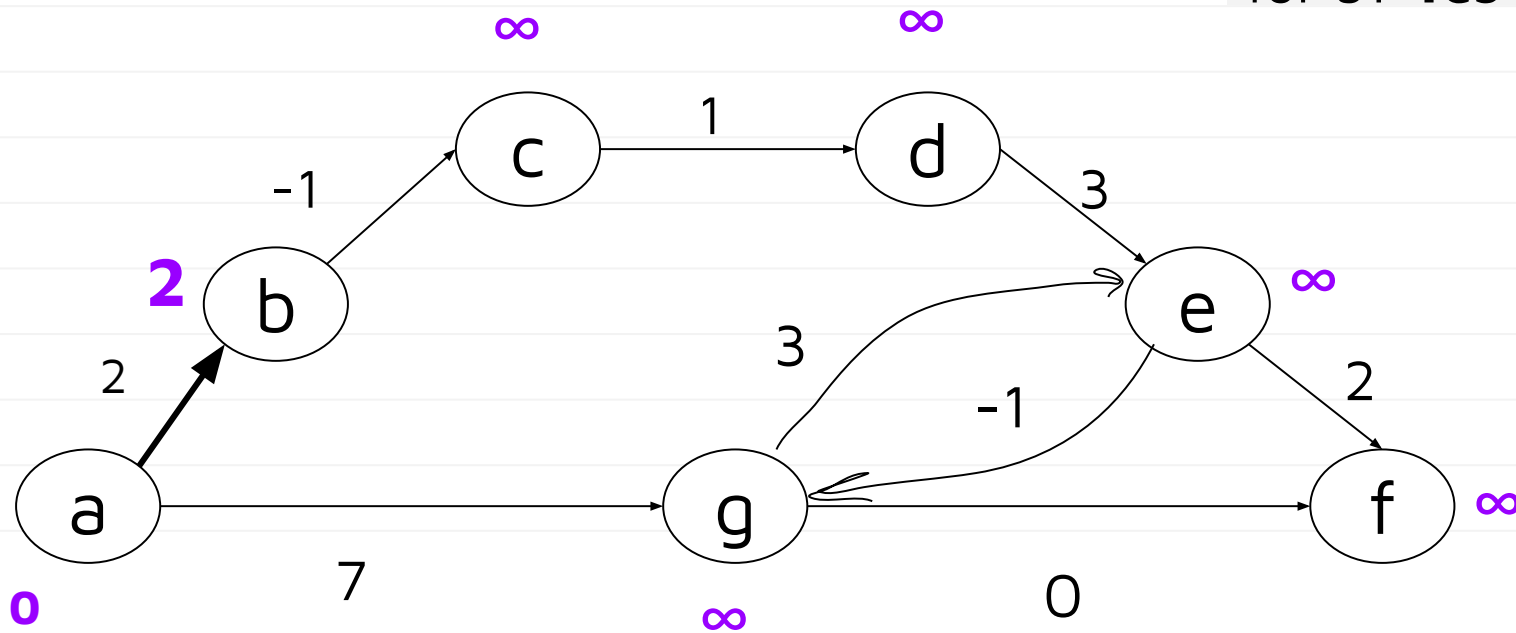
Do we update distance
for b?



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

Do we update distance
for b? **Yes**

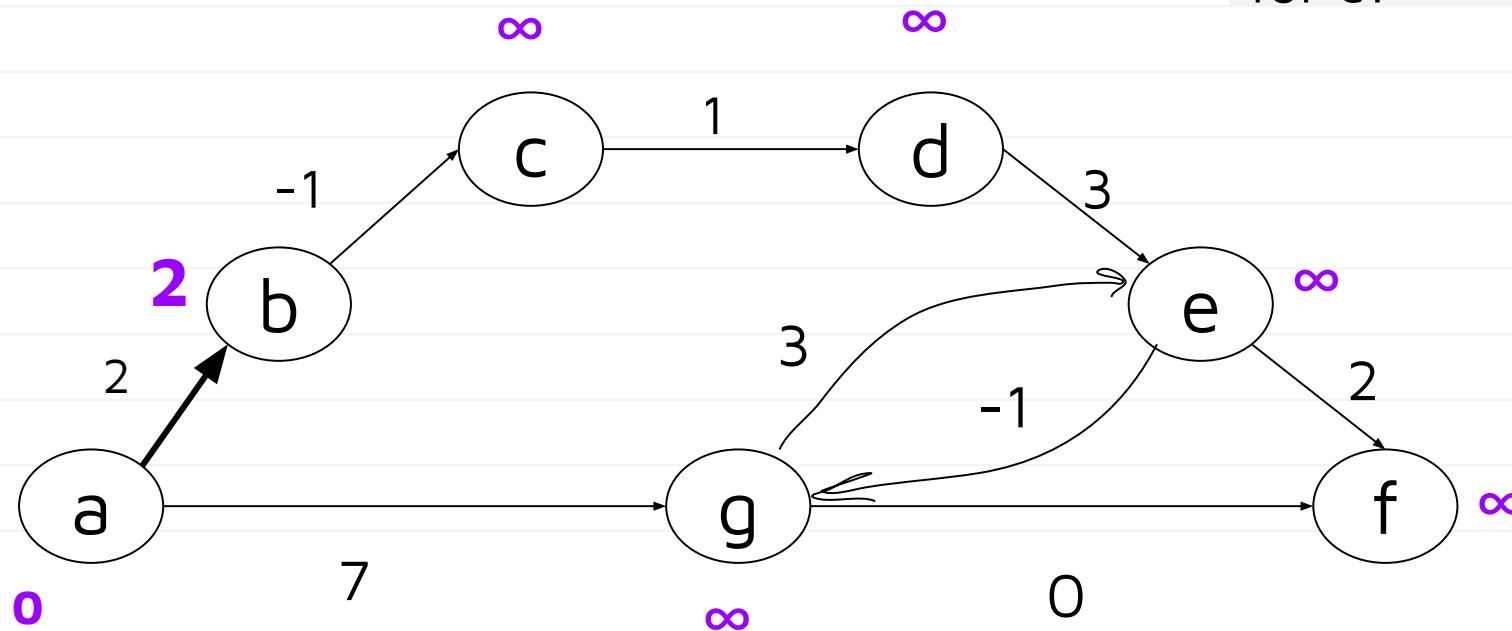


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, **(b, c)**, (g, f), (e, g),

(g, e), (d, e), (e, f), (a, g)

Do we update distance
for c?

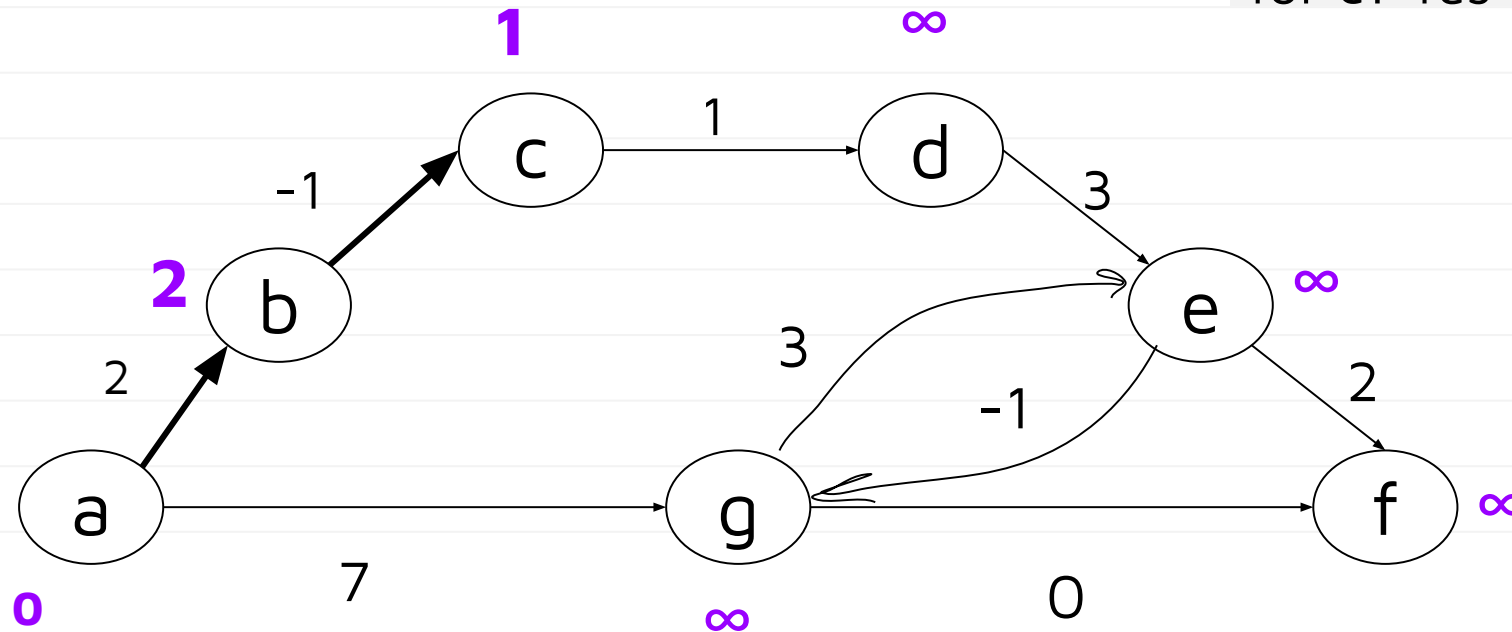


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, (g, f), (e, g),

(g, e), (d, e), (e, f), (a, g)

Do we update distance for c? Yes

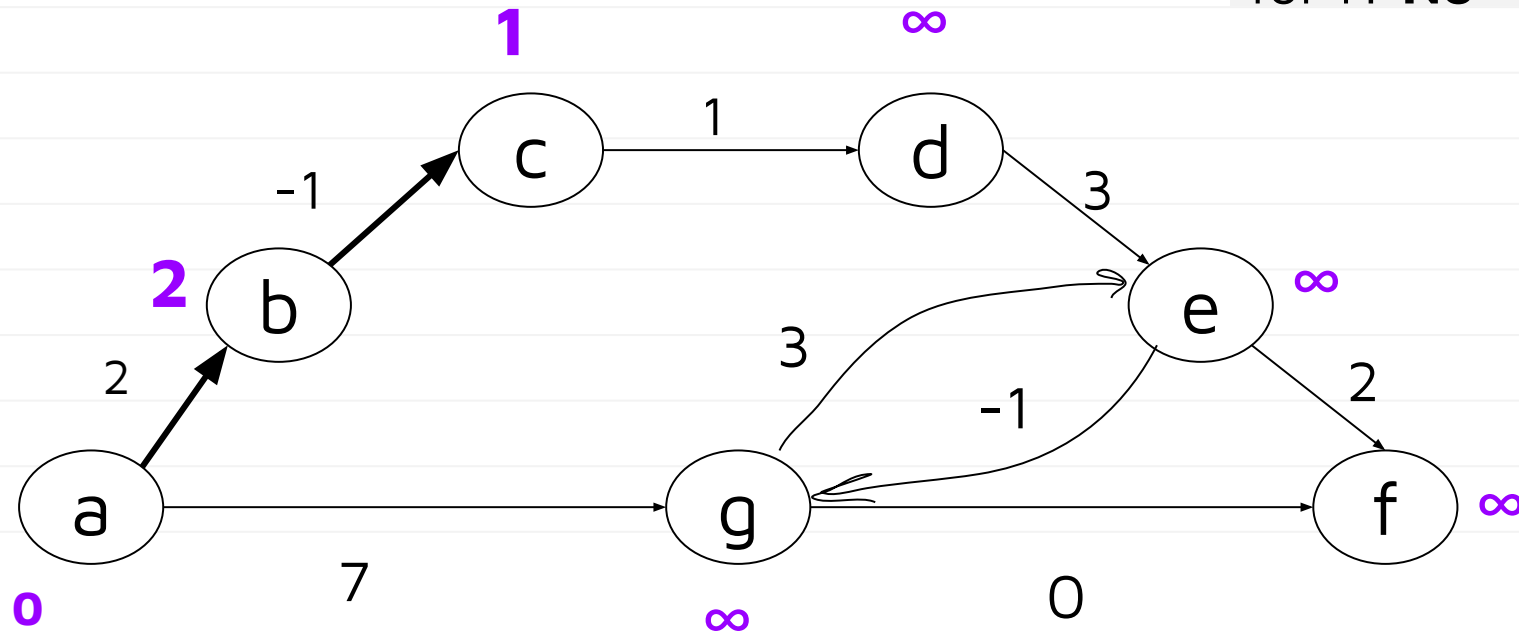


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, **(g, f)**, (e, g),

(g, e), (d, e), (e, f), (a, g)

Do we update distance for f? **No**

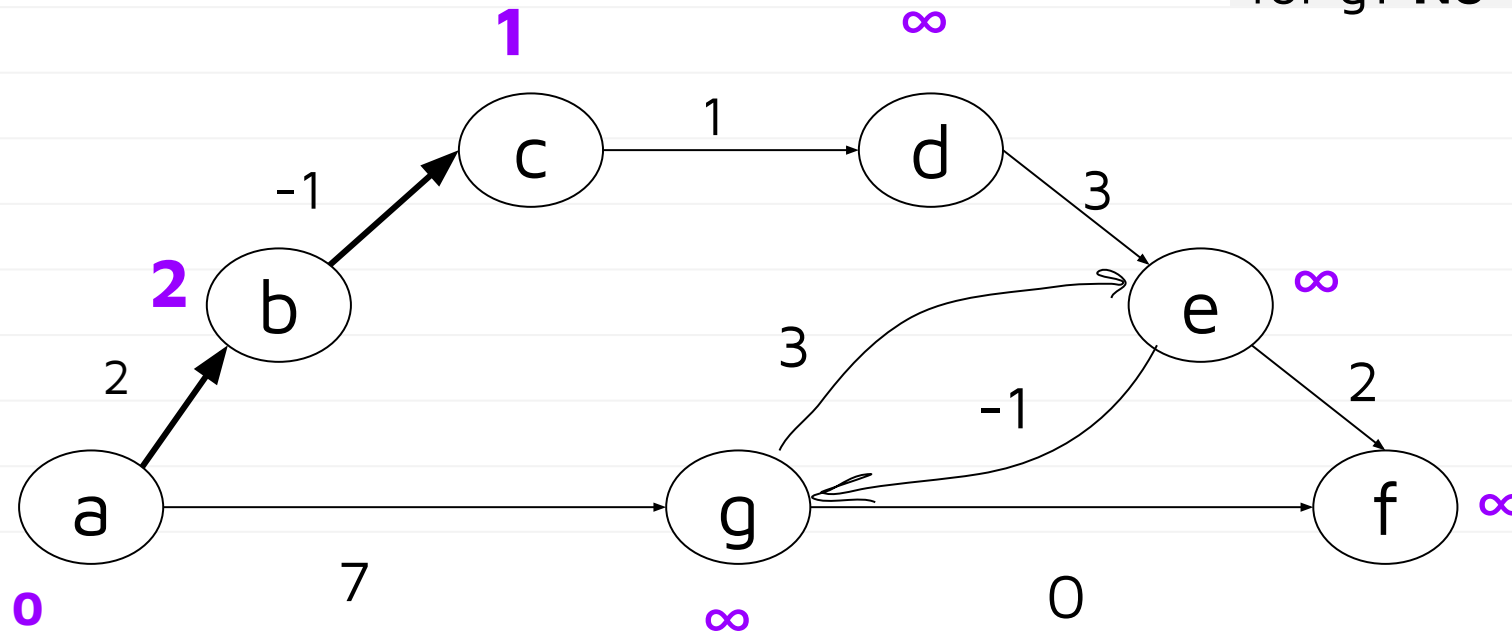


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, (e, g),

(g, e), (d, e), (e, f), (a, g)

Do we update distance for g? **No**

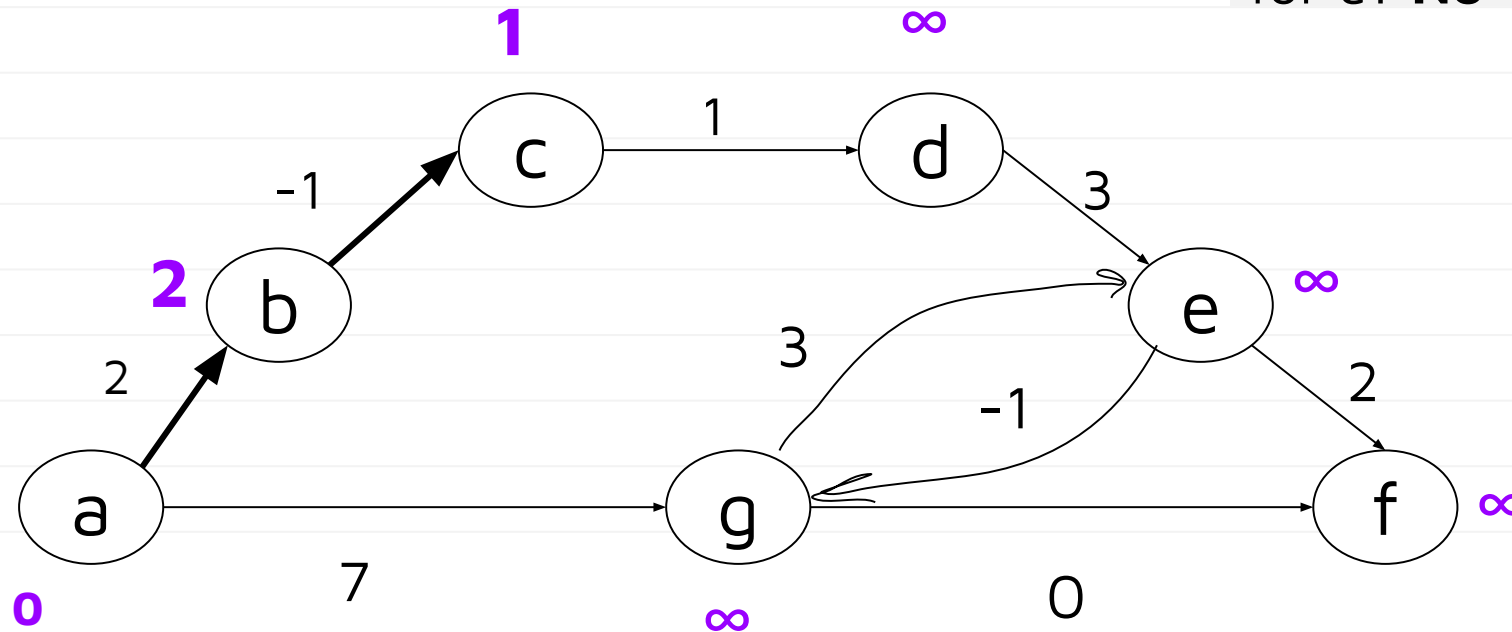


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

(g, e), (d, e), (e, f), (a, g)

Do we update distance for e? **No**

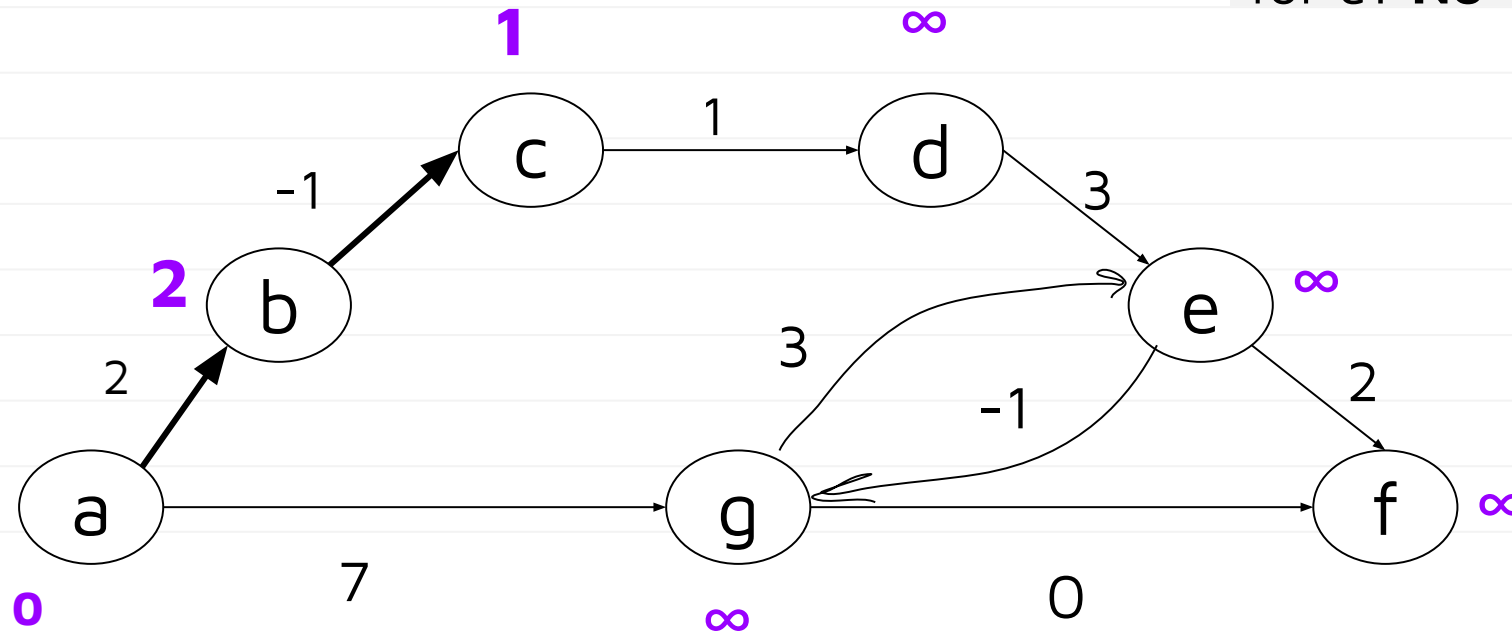


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

~~(g, e)~~, (d, e), (e, f), (a, g)

Do we update distance for e? **No**

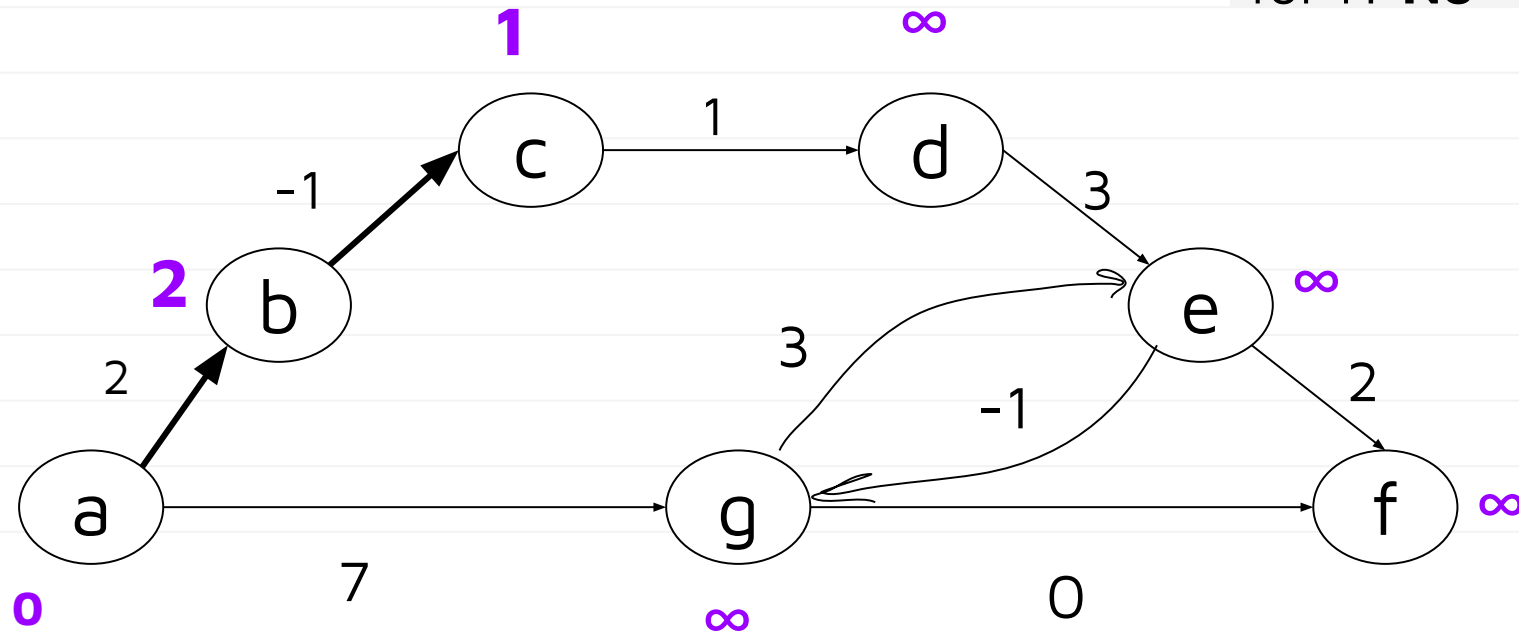


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, (a, g)

Do we update distance for f? **No**

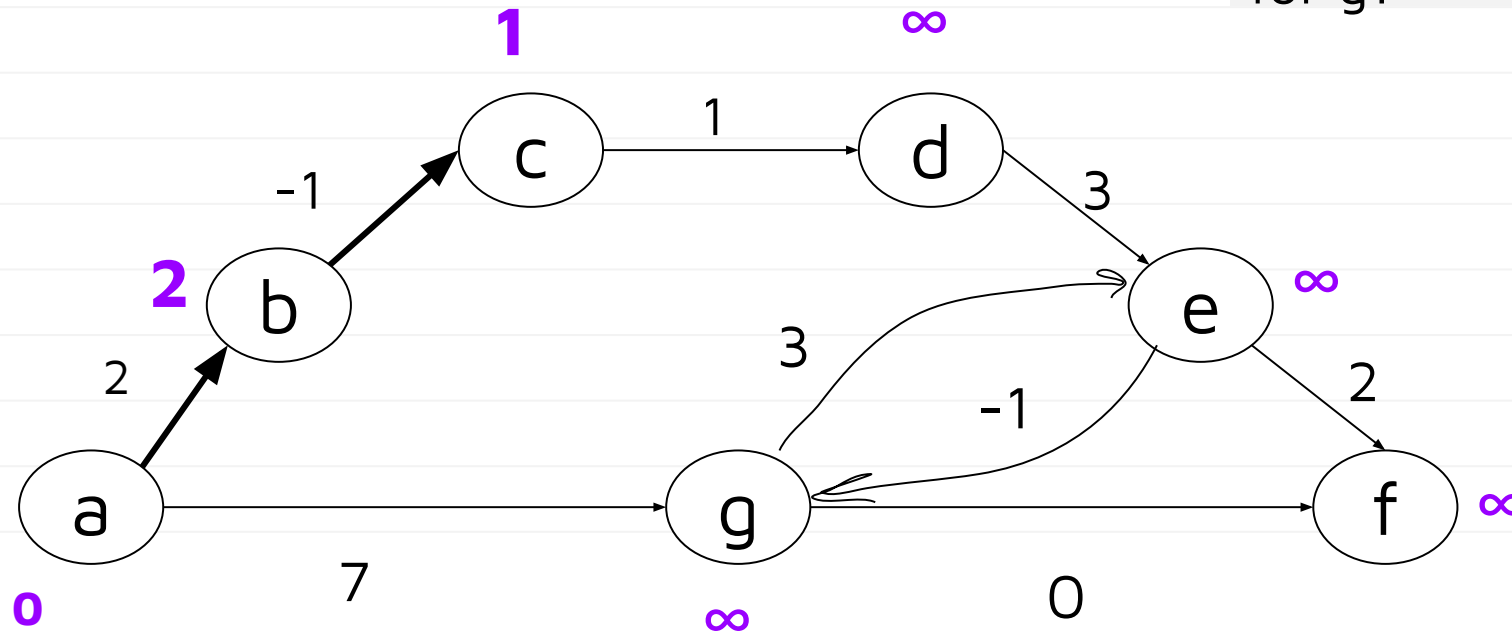


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, (a, g)

Do we update distance
for g?

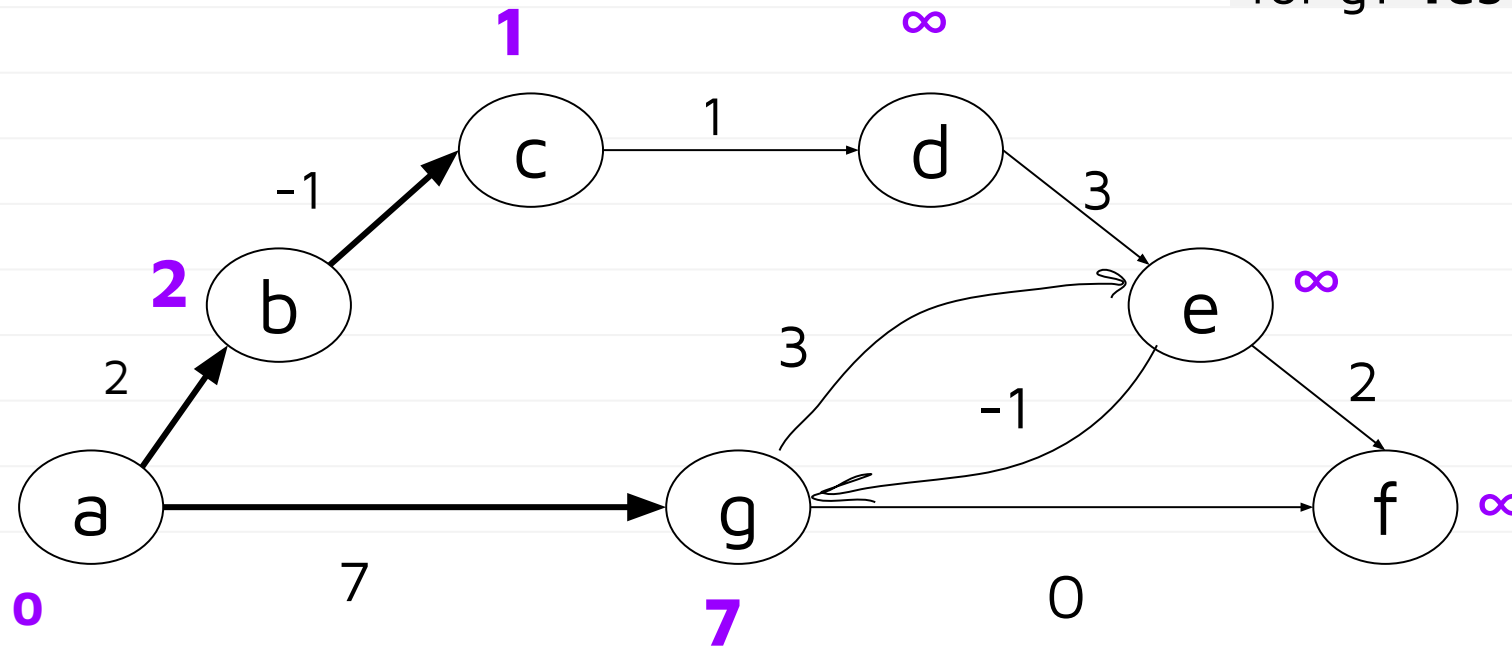


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, ~~(a, g)~~

Do we update distance for g? **Yes**

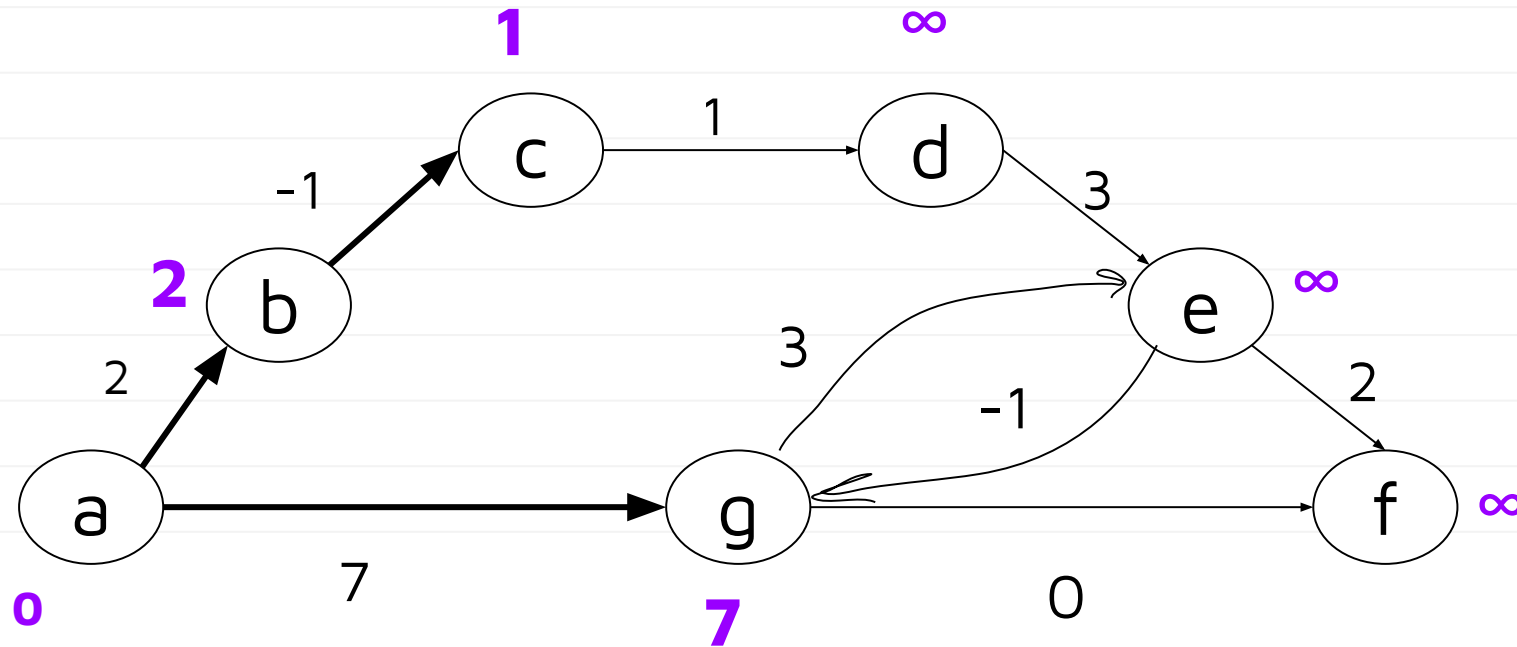


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, ~~(a, g)~~

It was just a **single** iteration of BF

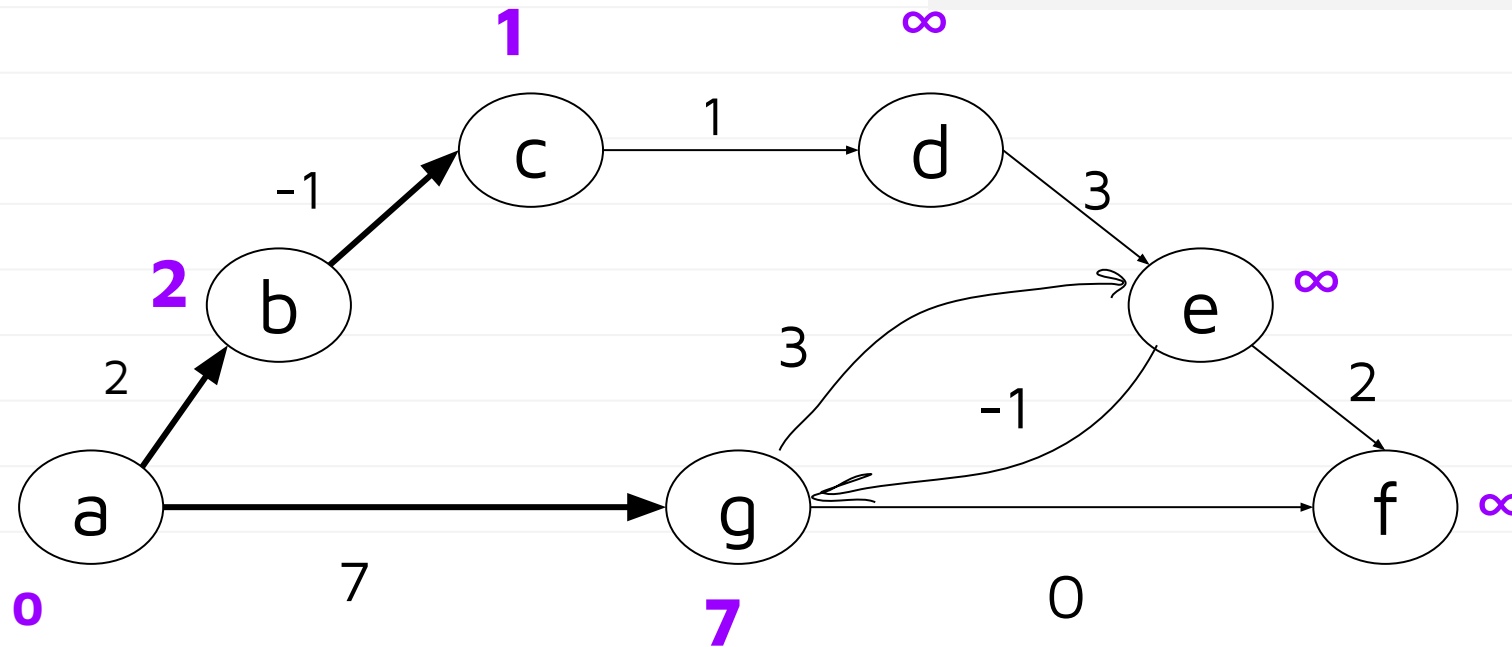


Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,

~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, ~~(a, g)~~

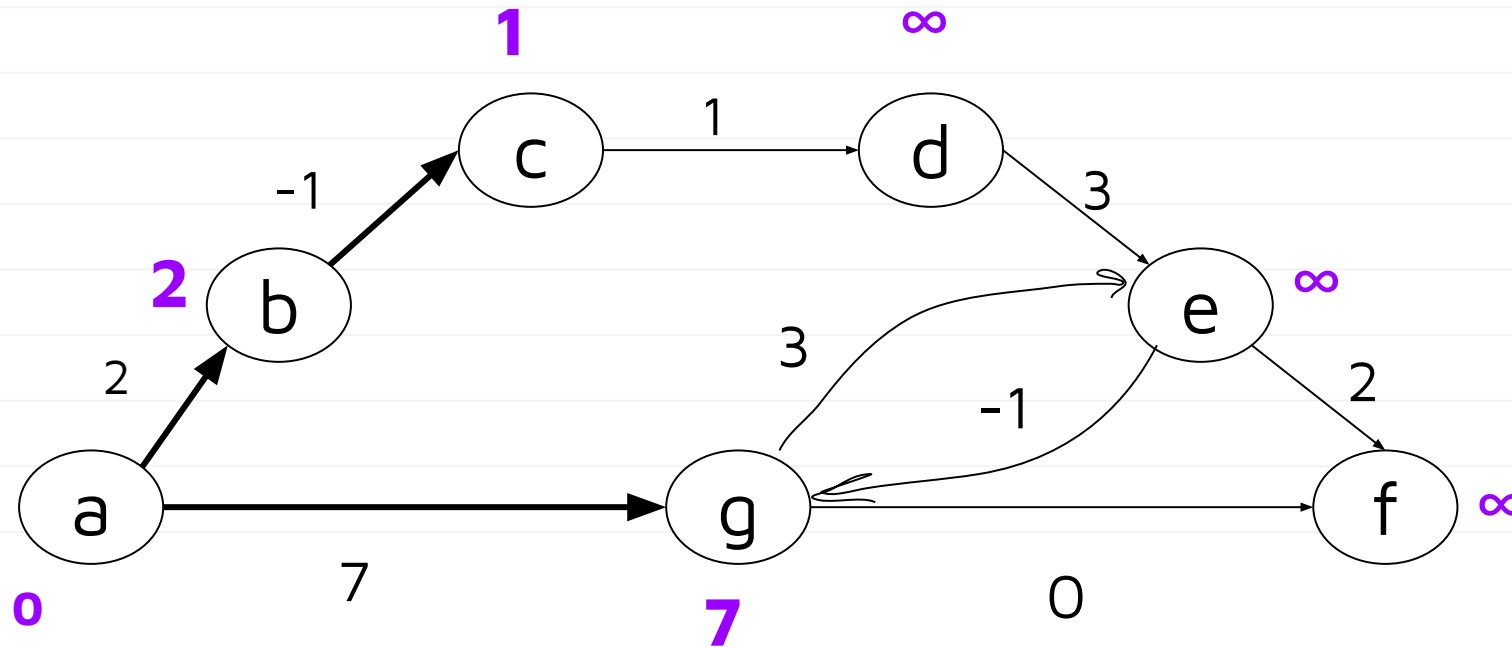
We only can guarantee one correct edge after this iteration: (a, b)



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

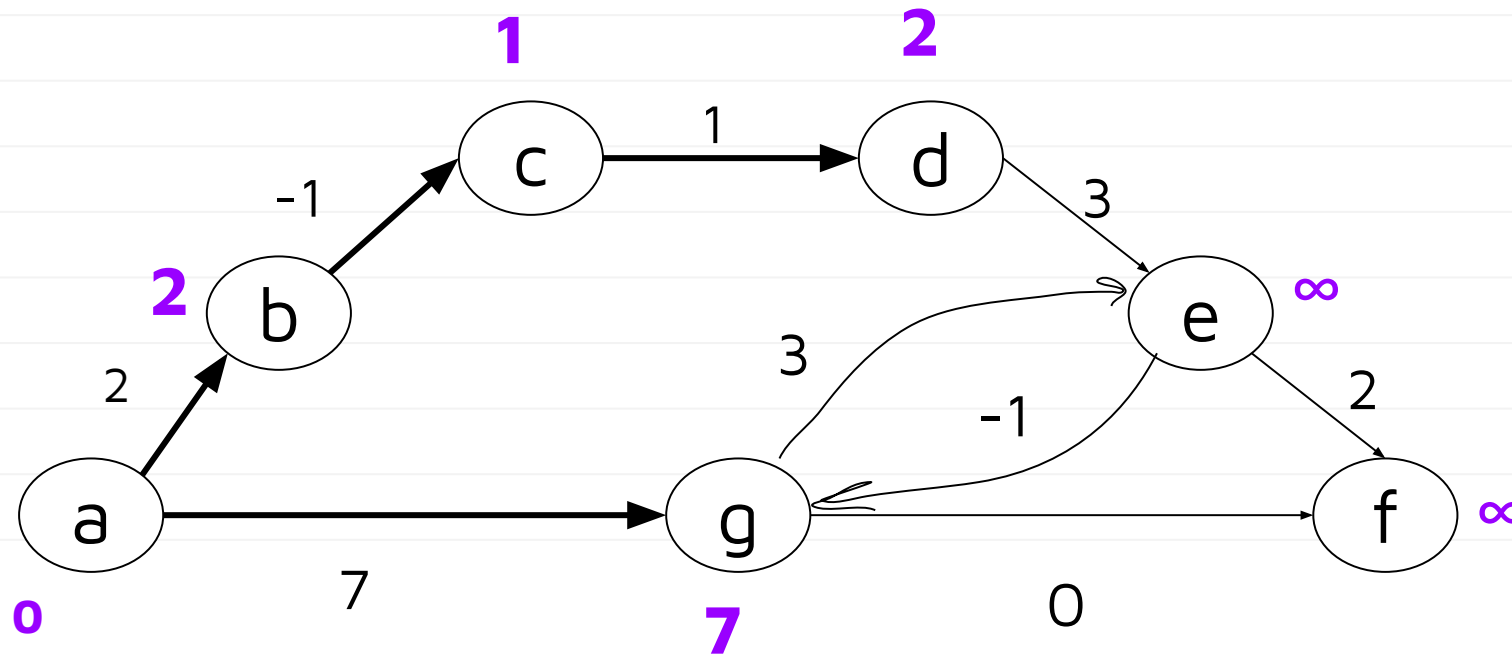
Do we update distance for d? **Yes**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

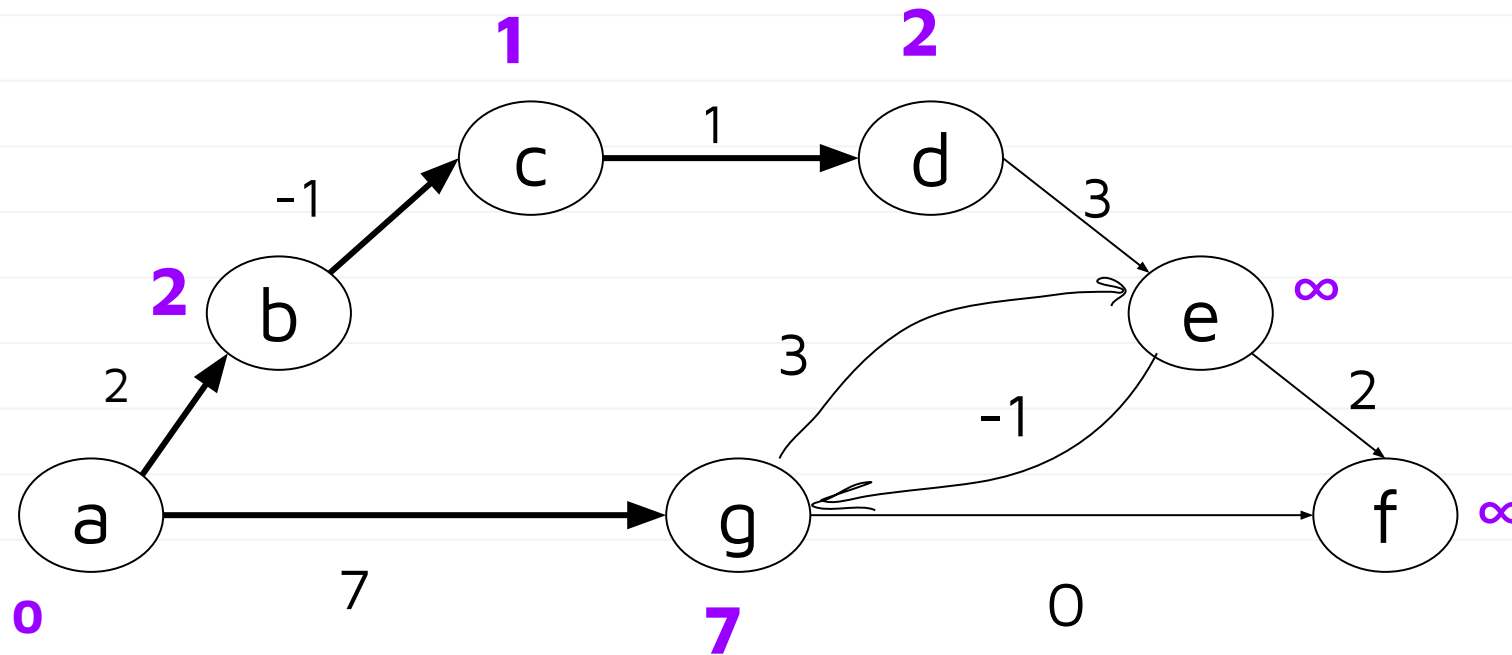
Do we update distance for b? **No**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, **(b, c)**, (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

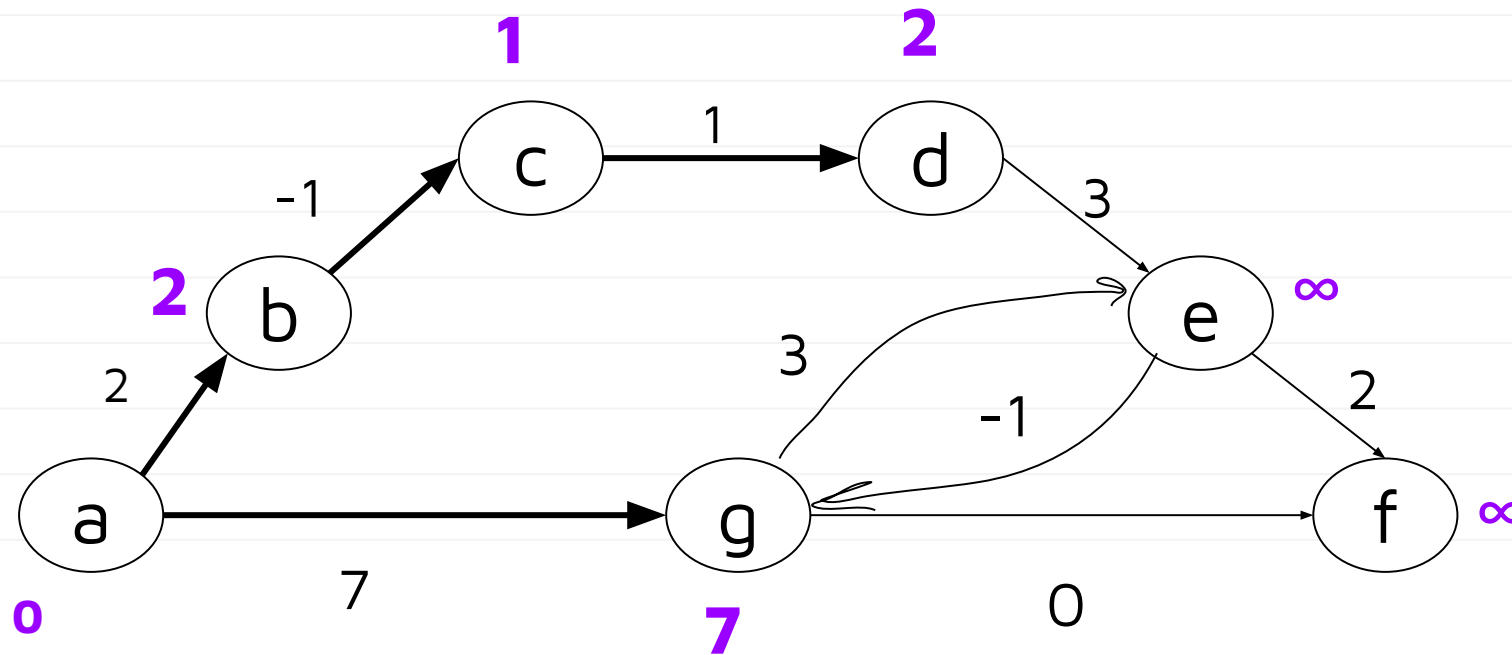
Do we update distance for c? **No**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, **(g, f)**, (e, g),
(g, e), (d, e), (e, f), (a, g)

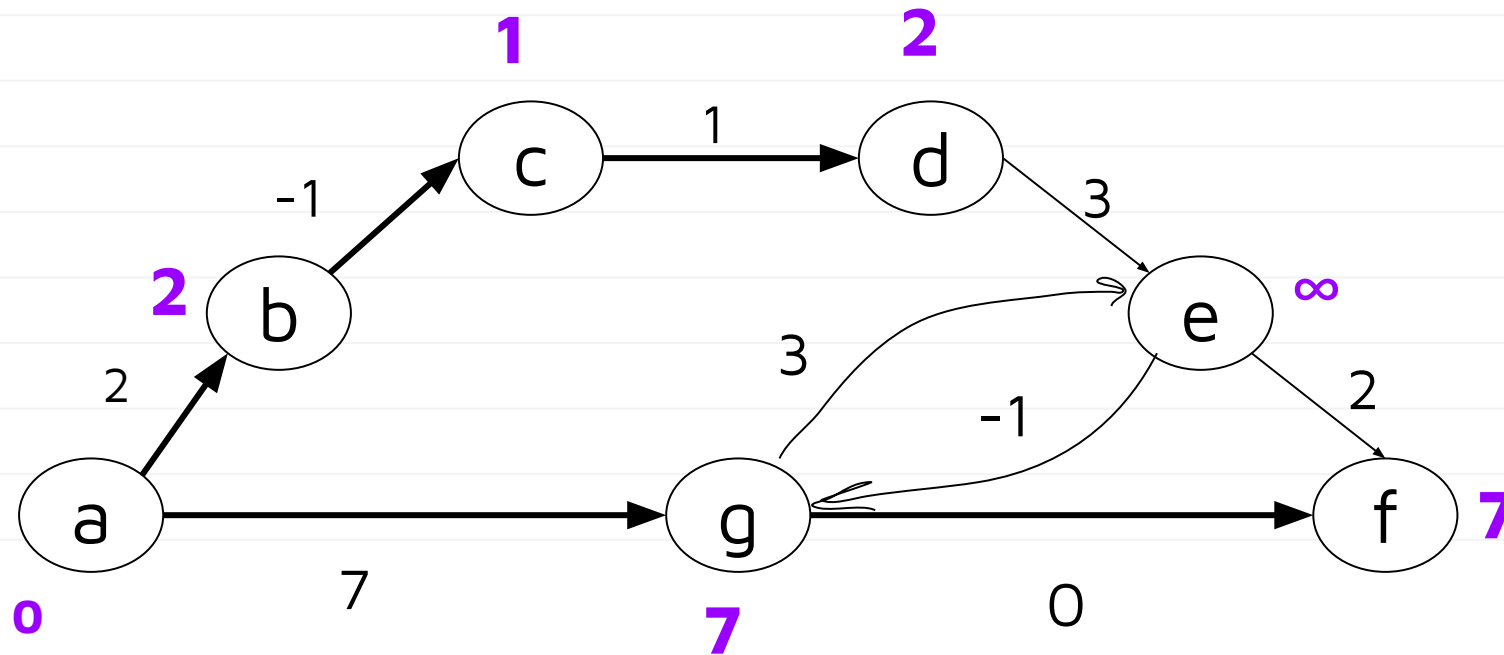
Do we update distance for f? **Yes**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, **(g, f)**, (e, g),
(g, e), (d, e), (e, f), (a, g)

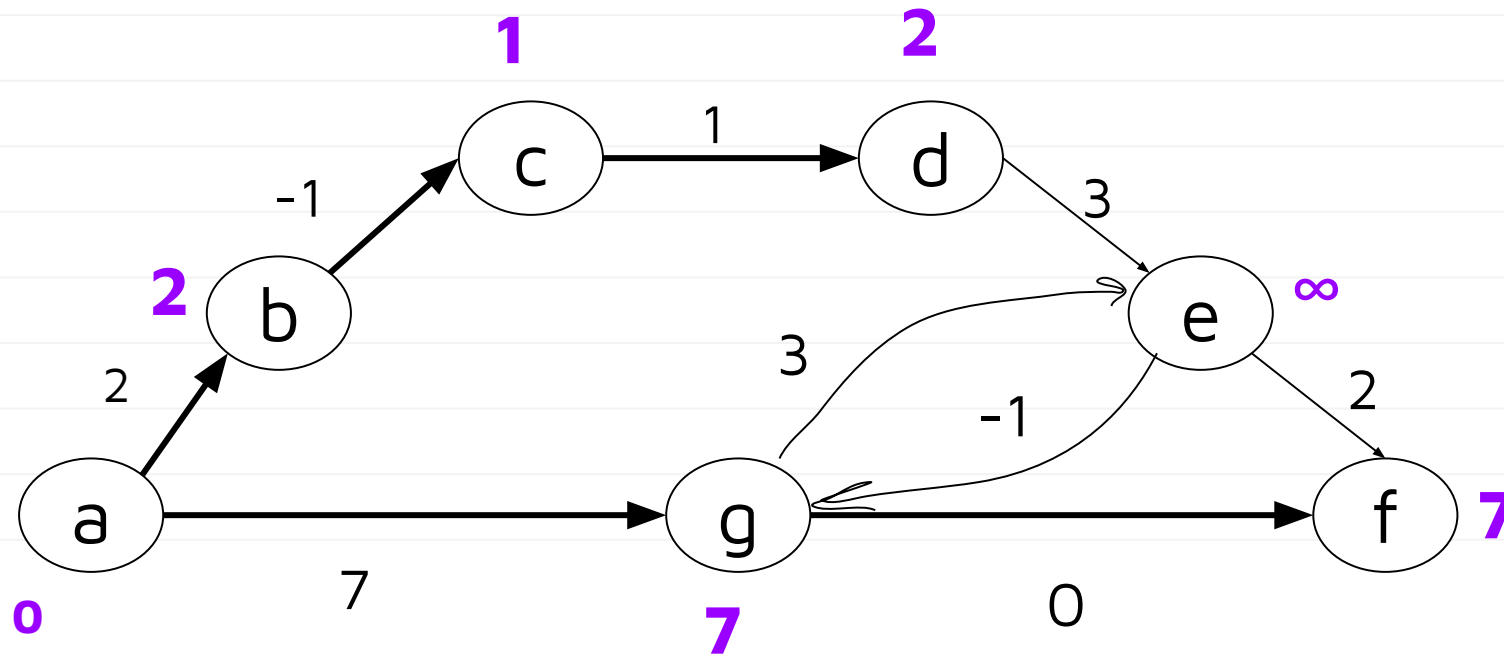
Do we update distance for f? **Yes**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, (e, g),
(g, e), (d, e), (e, f), (a, g)

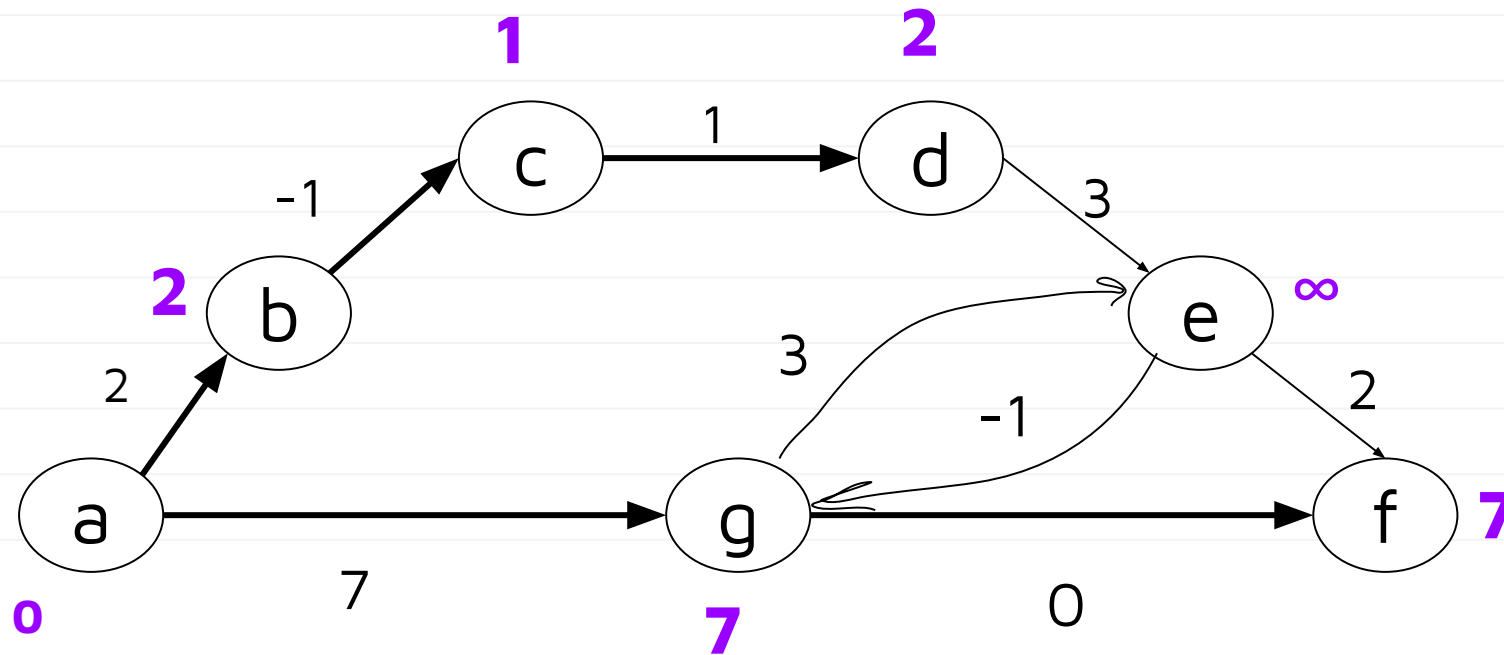
Do we update distance for g? **No**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,
(g, e), (d, e), (e, f), (a, g)

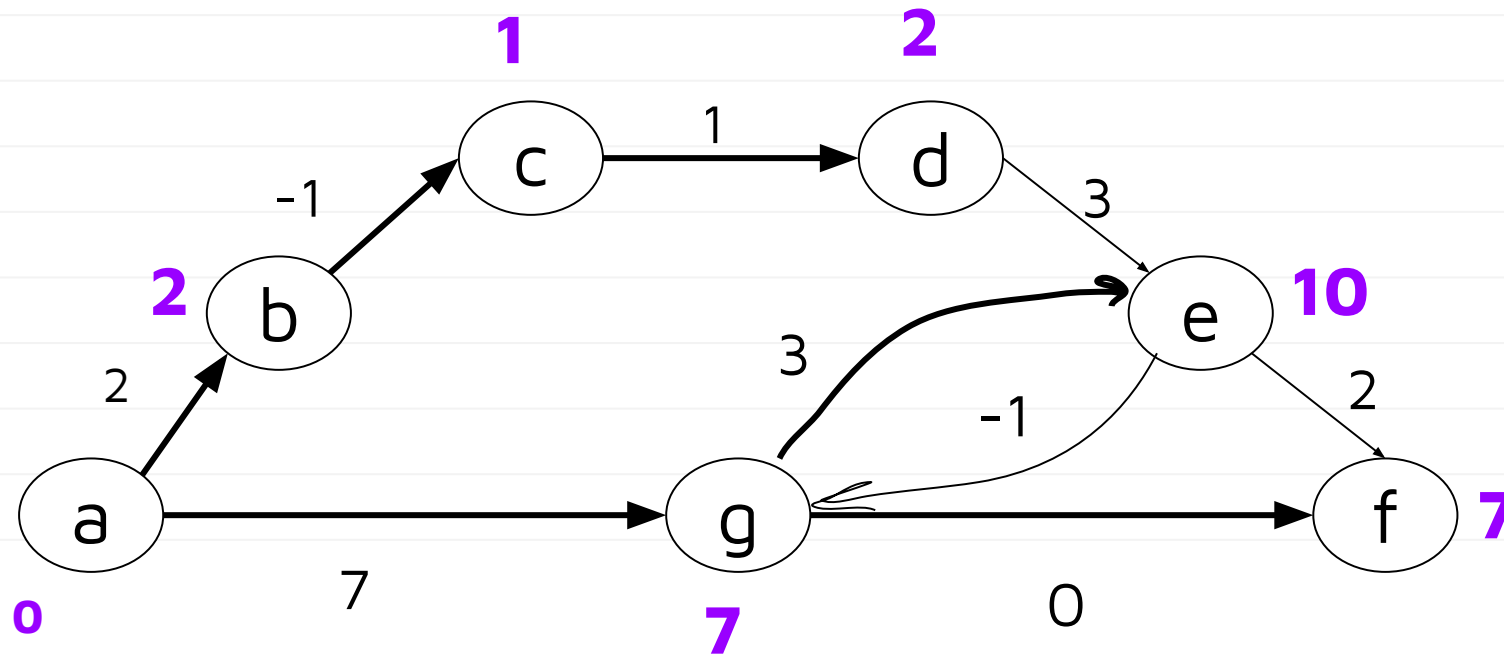
Do we update distance for e? **Yes**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,
~~(g, e)~~, (d, e), (e, f), (a, g)

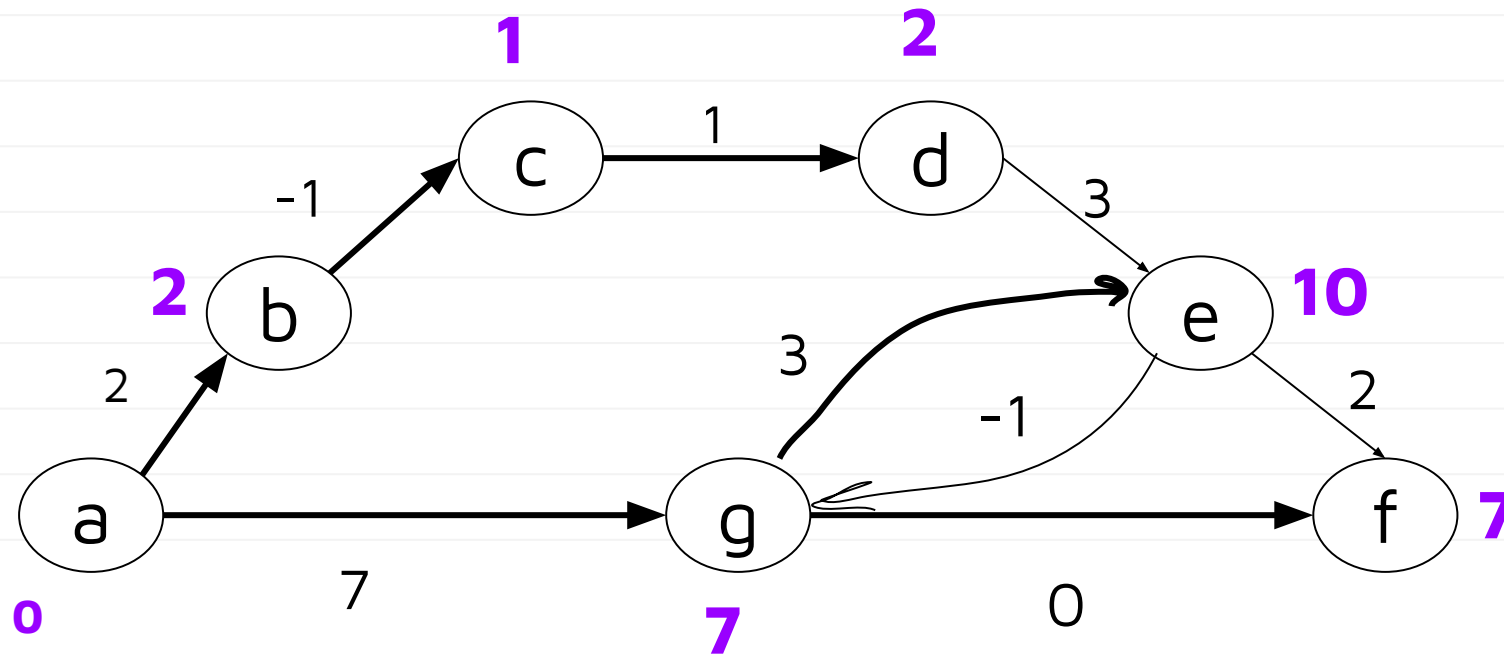
Do we update distance for e? **Yes**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,
~~(g, e)~~, (d, e), (e, f), (a, g)

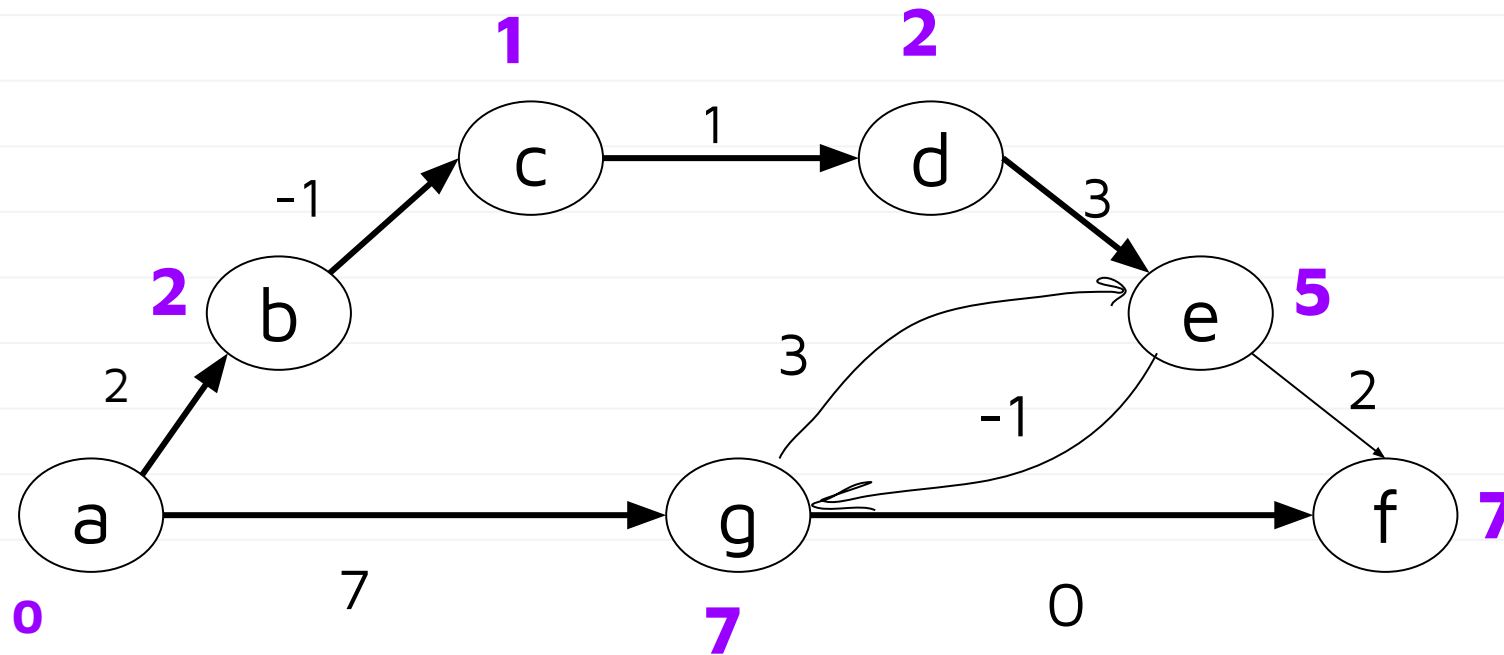
Do we update distance for e? **Yes**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,
~~(g, e)~~, ~~(d, e)~~, (e, f), (a, g)

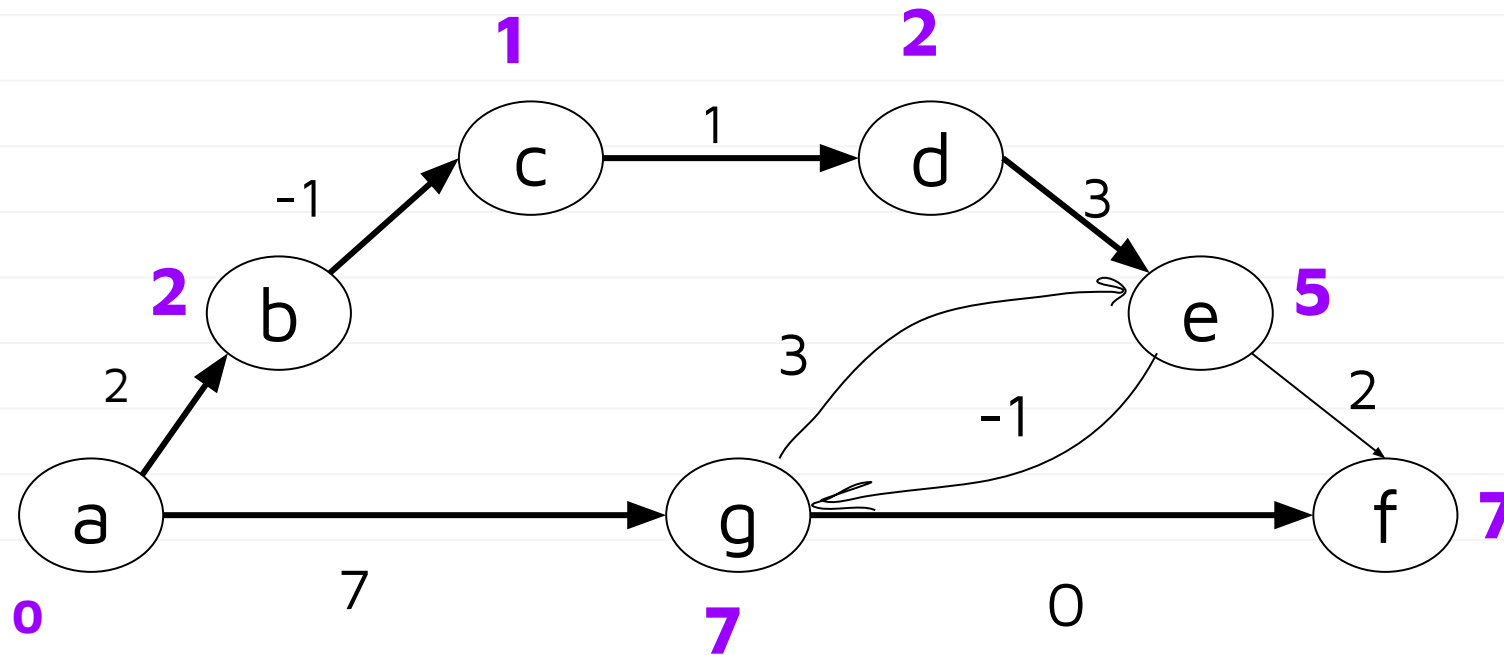
Do we update distance for f? **No**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,
~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, (a, g)

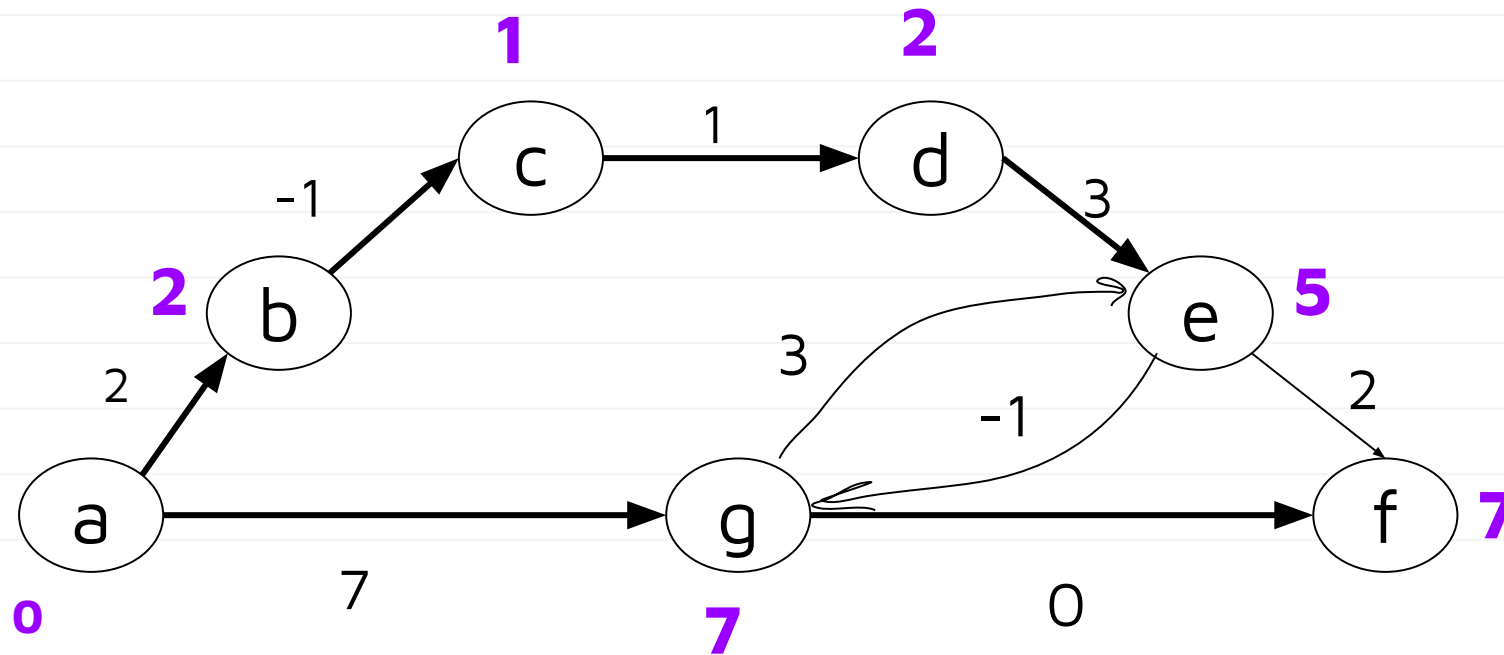
Do we update distance for g? **No**



Suppose graph.edges returns edges in following order:

~~(c, d)~~, ~~(a, b)~~, ~~(b, c)~~, ~~(g, f)~~, ~~(e, g)~~,
~~(g, e)~~, ~~(d, e)~~, ~~(e, f)~~, ~~(a, g)~~

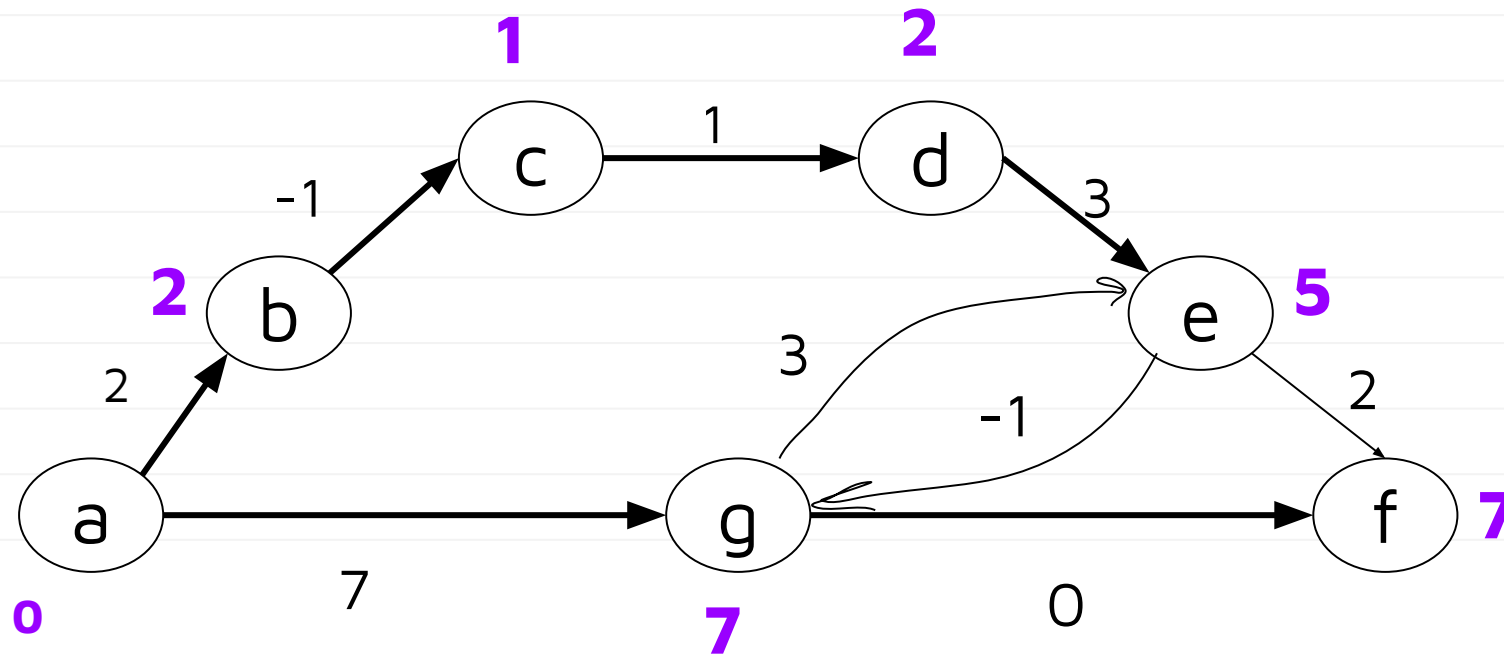
It was the end of the **second** iteration of BF



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

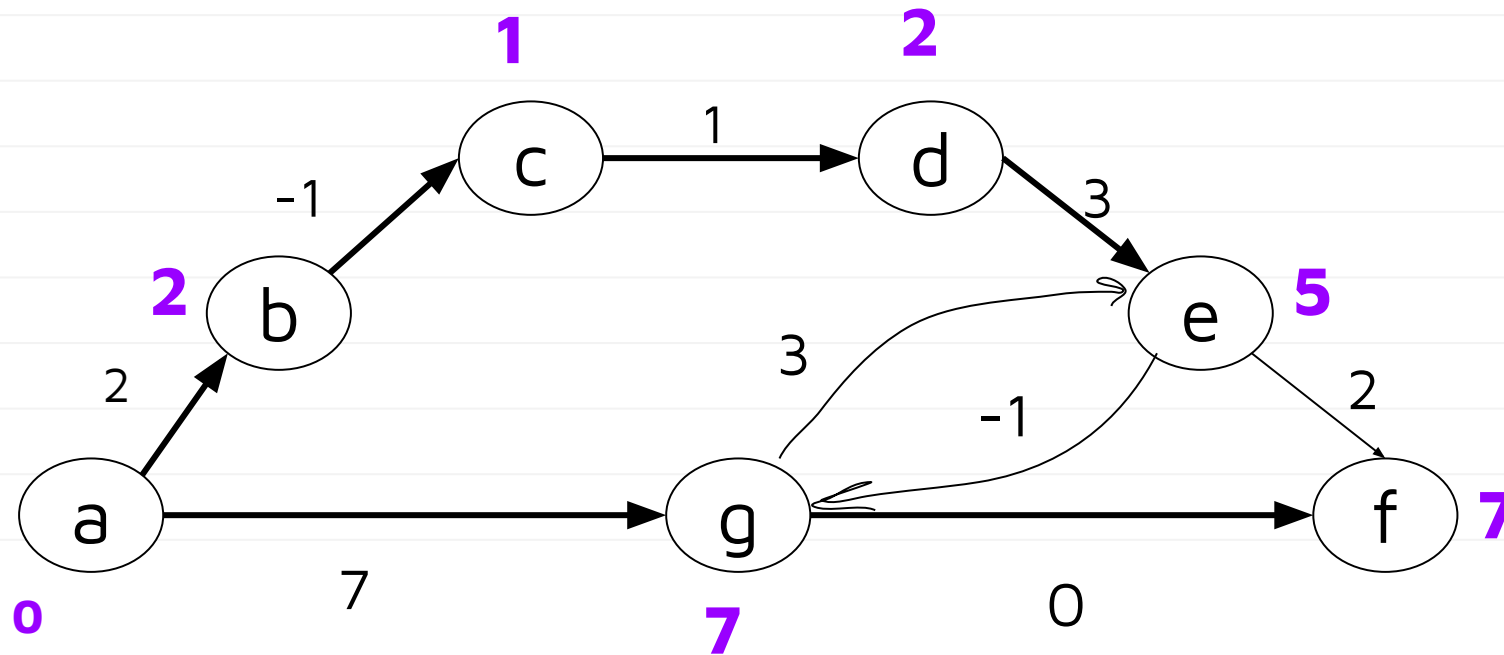
One more time



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

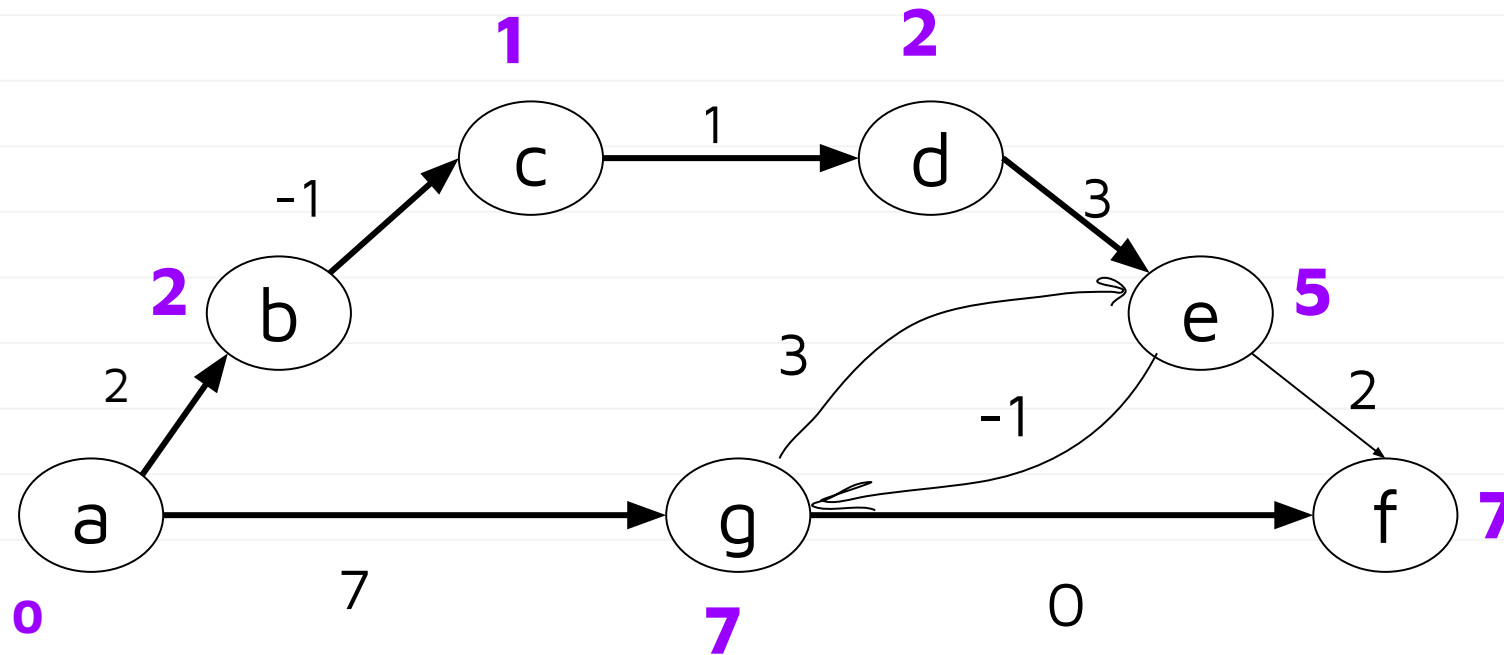
No change



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

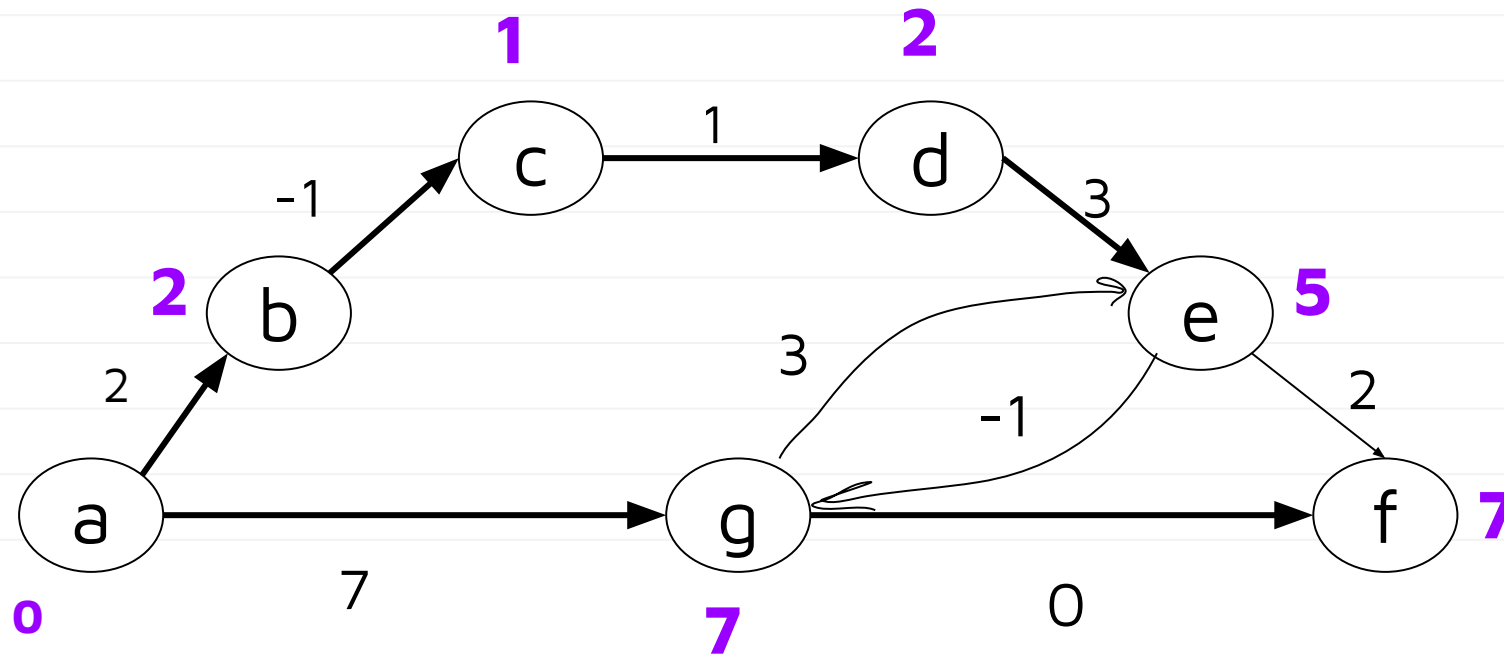
No change



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

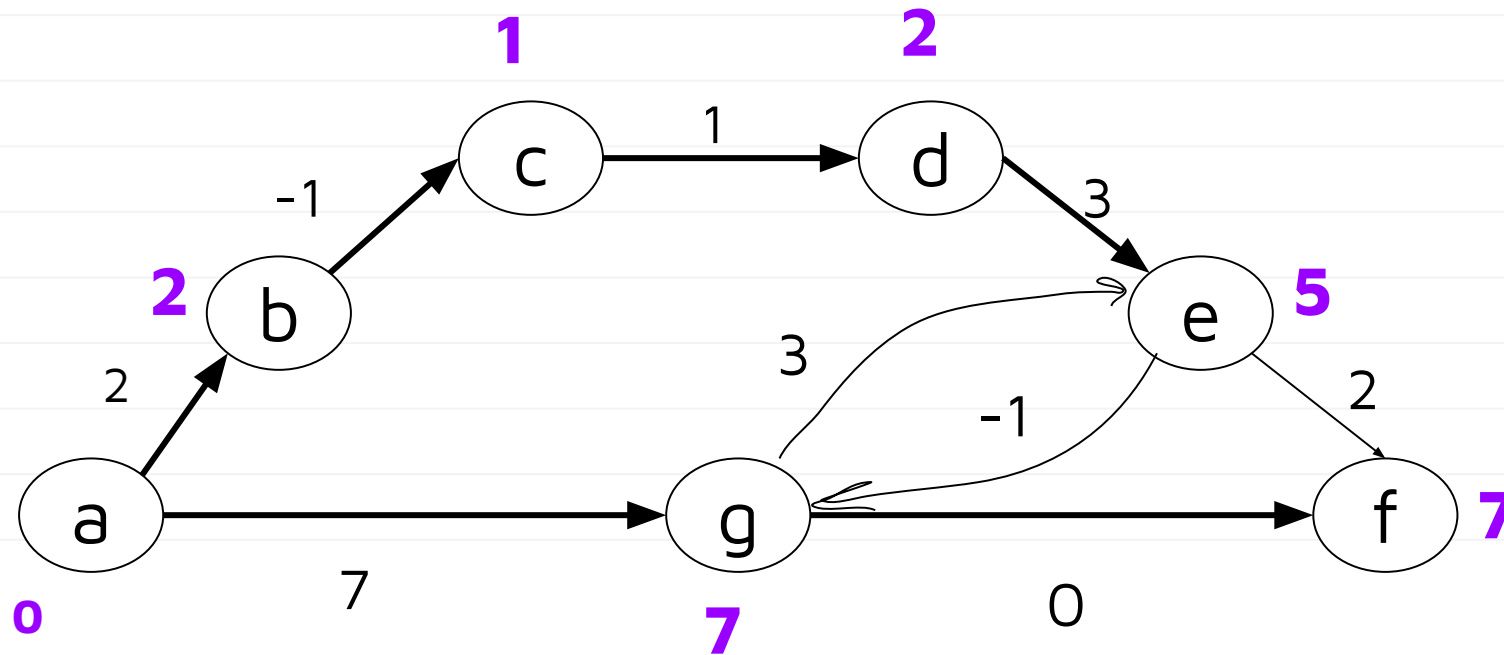
No change



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

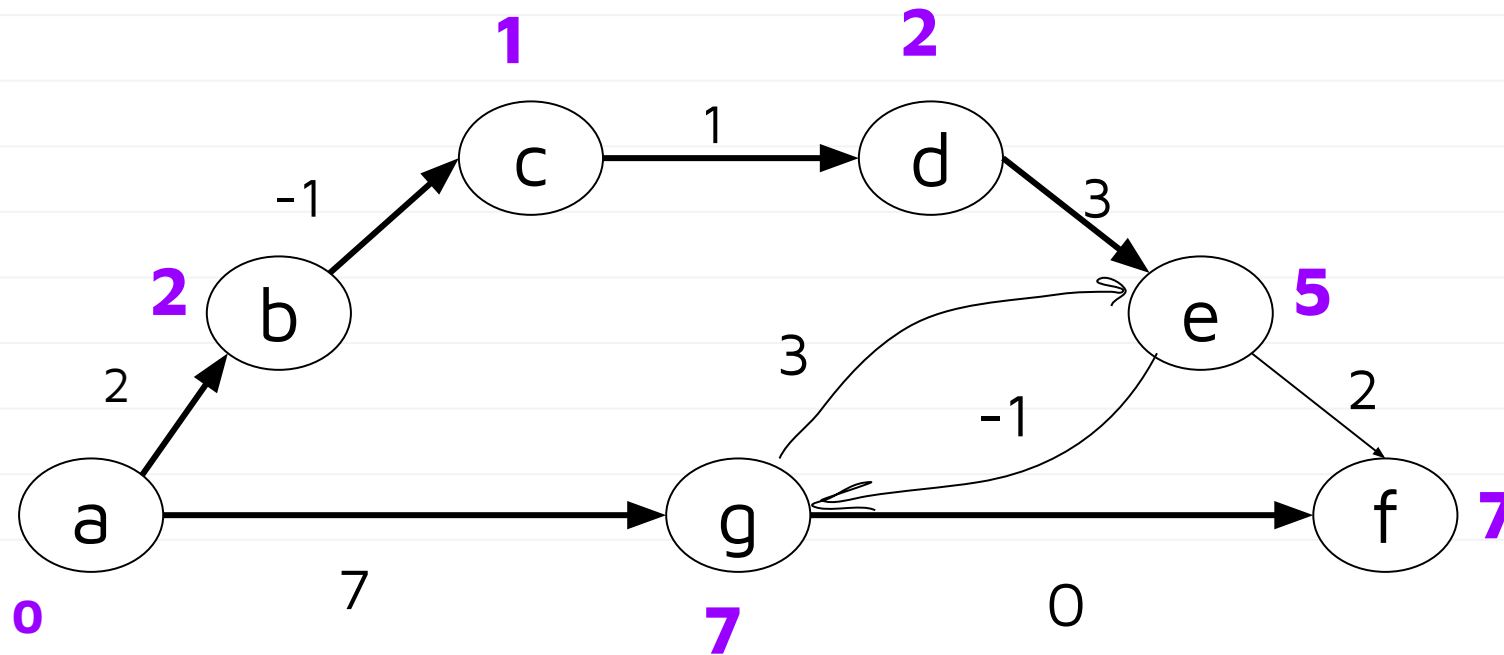
No change



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

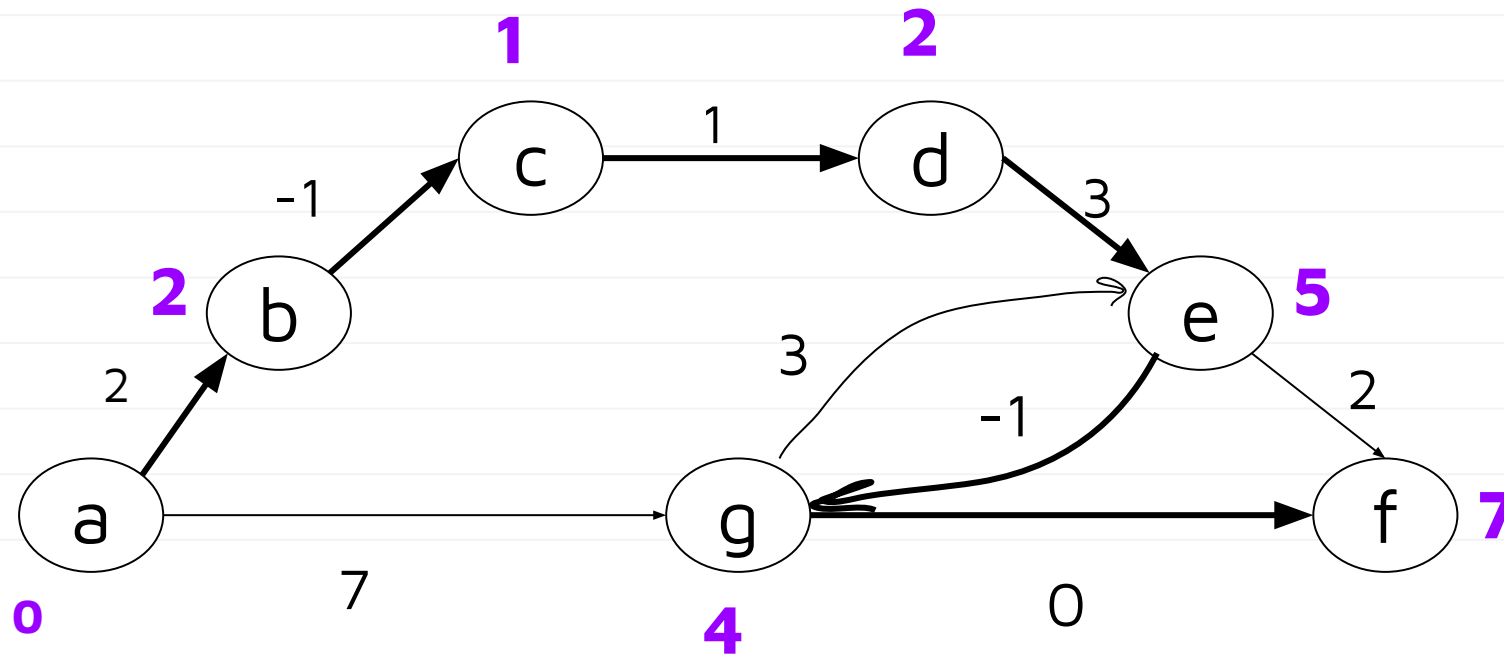
Update for node g.



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

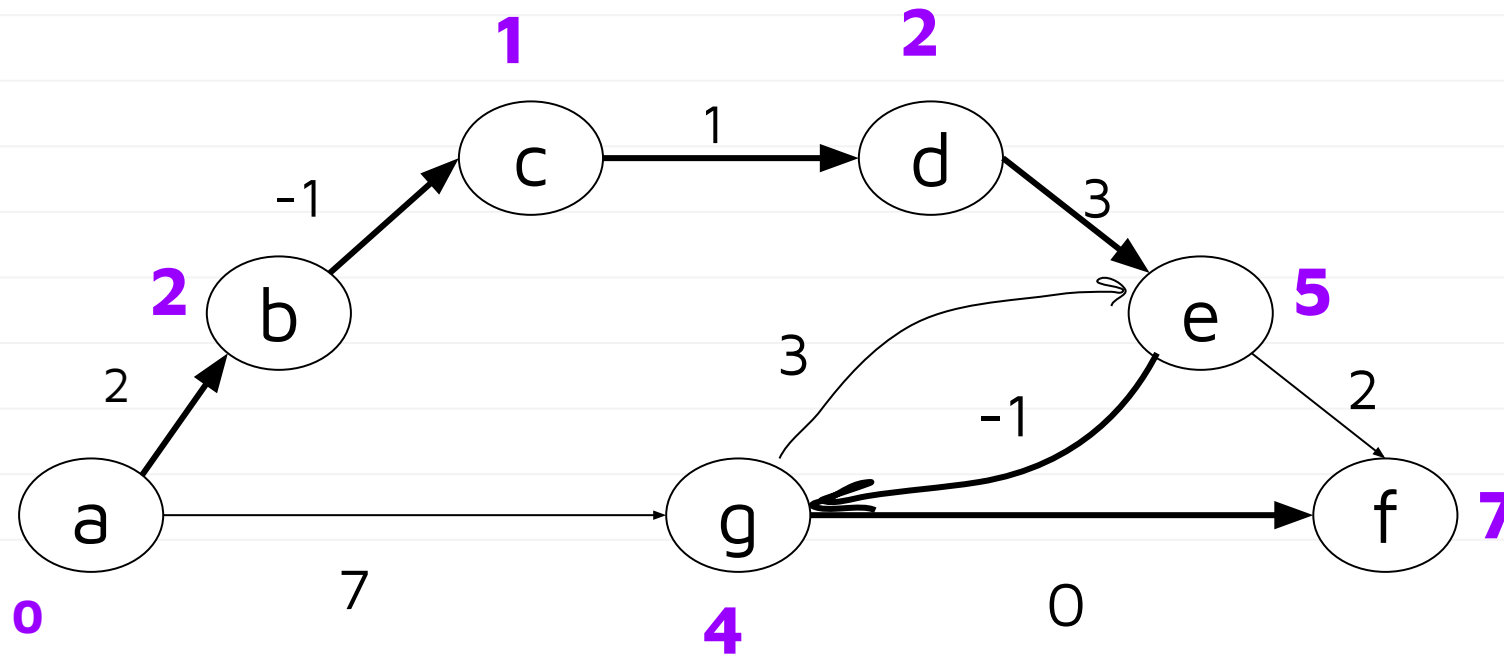
Update for node g.



Suppose graph.edges returns edges in following order:

**(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)**

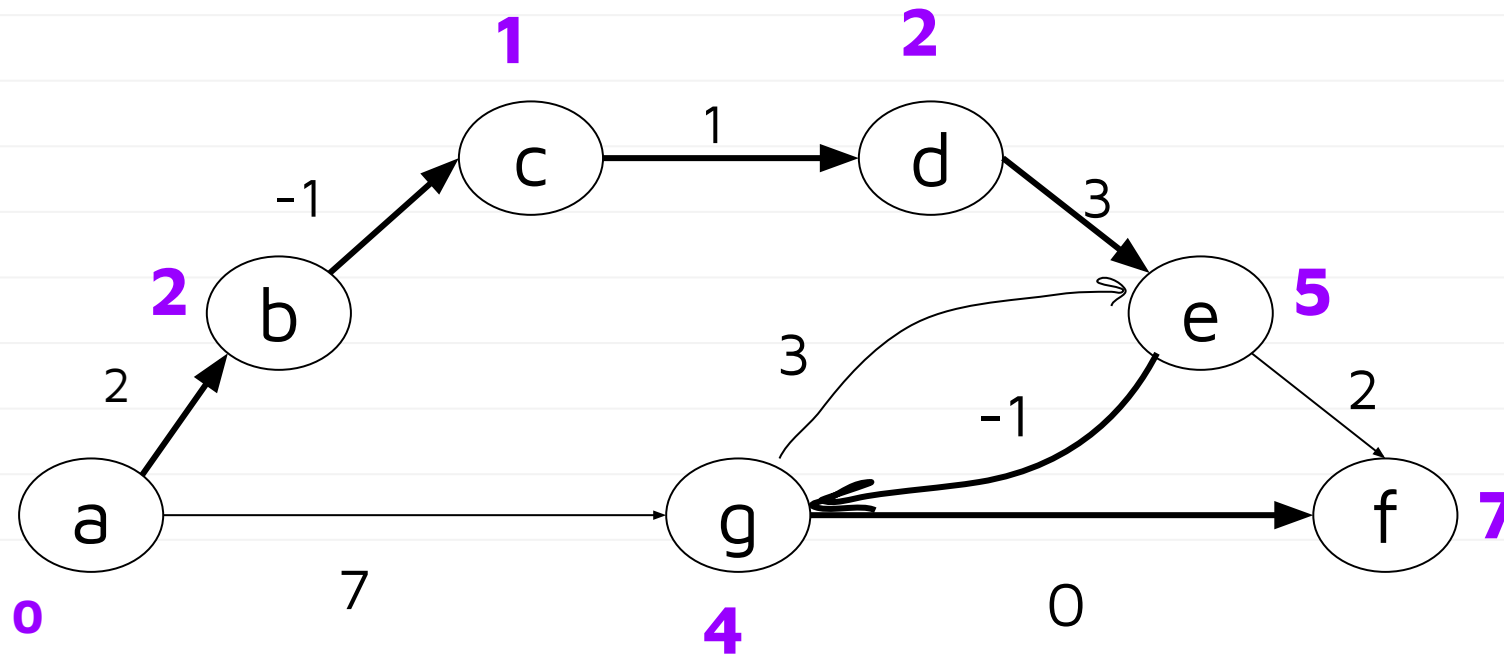
No change



Suppose graph.edges returns edges in following order:

**(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)**

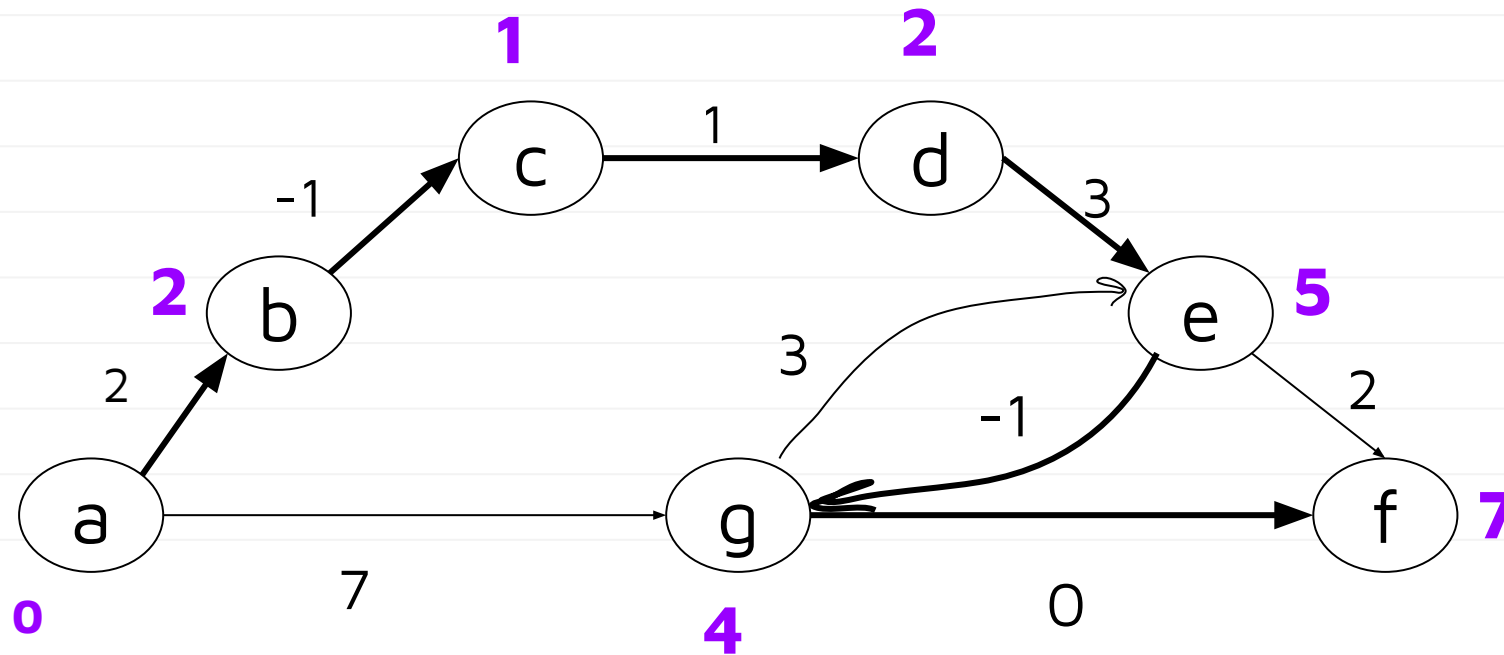
No change



Suppose graph.edges returns edges in following order:

**(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)**

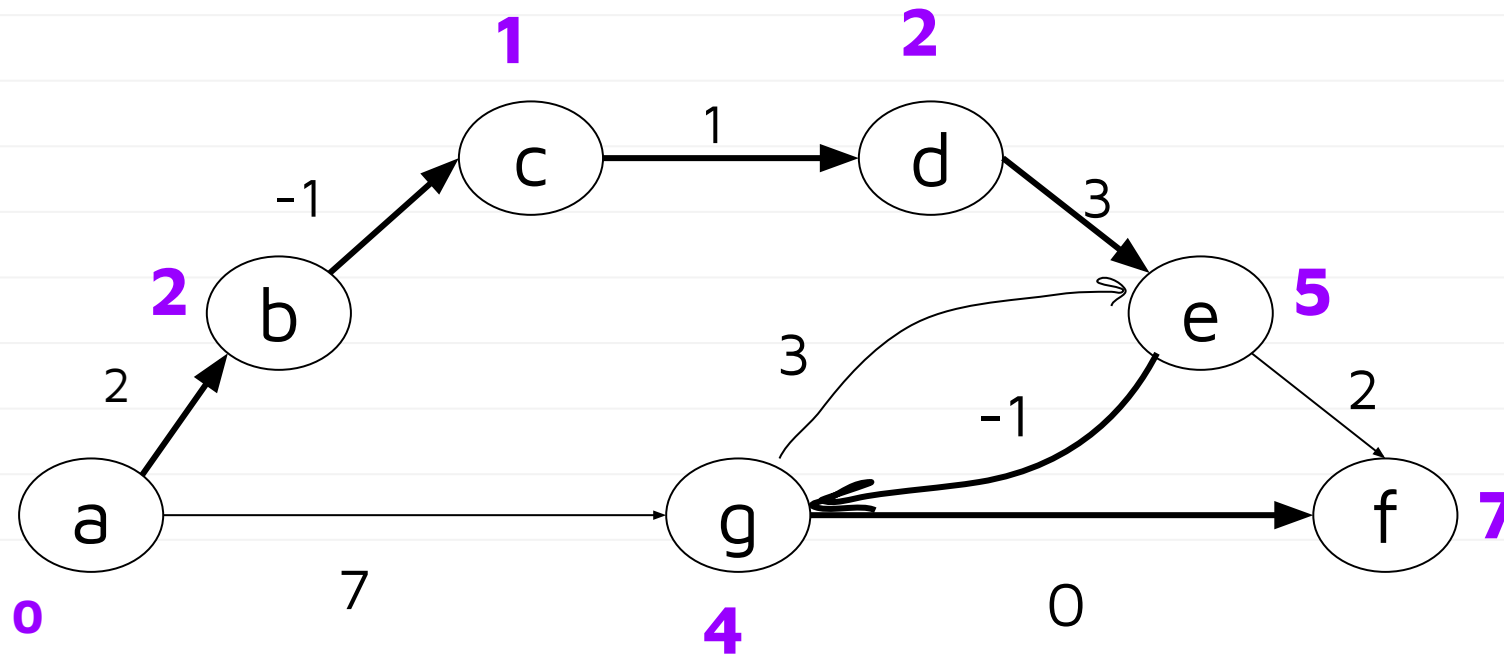
No change



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

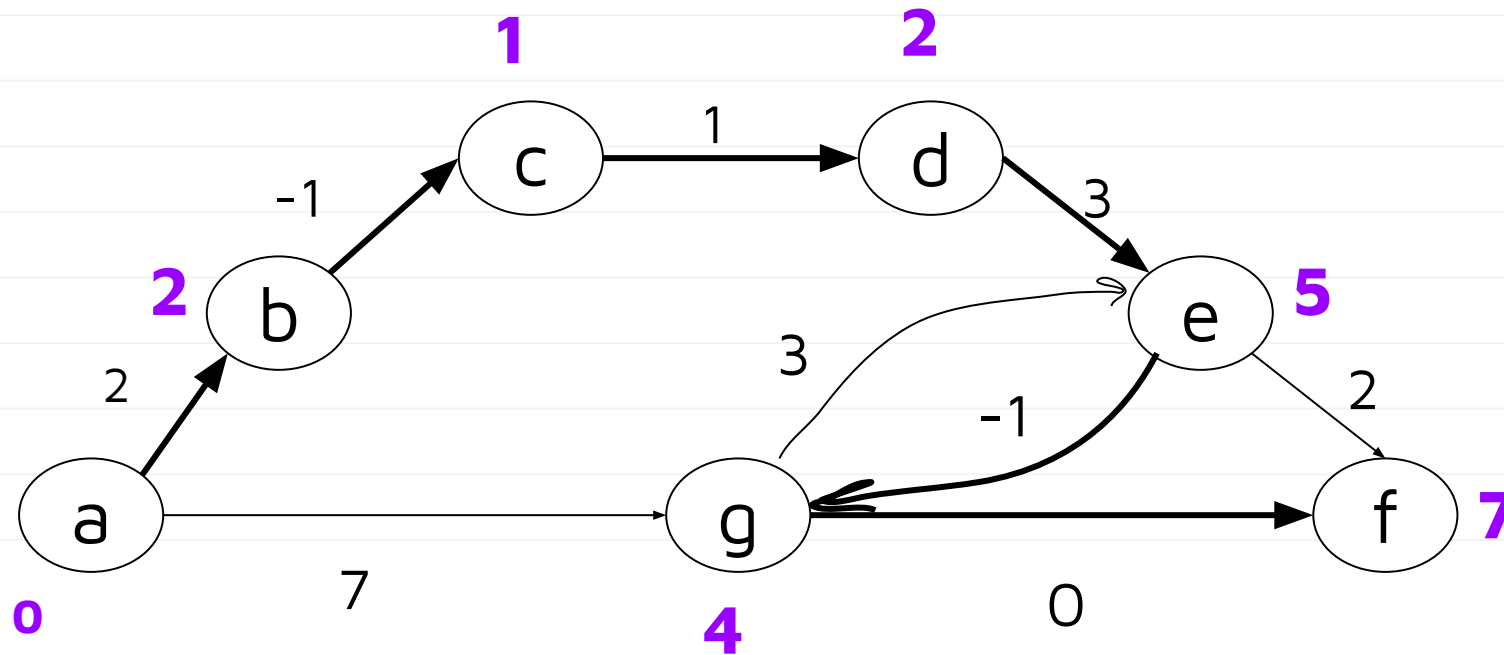
No change



Suppose graph.edges returns edges in following order:

**(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)**

It was the end of the **third** iteration of BF



Suppose graph.edges returns edges in following order:

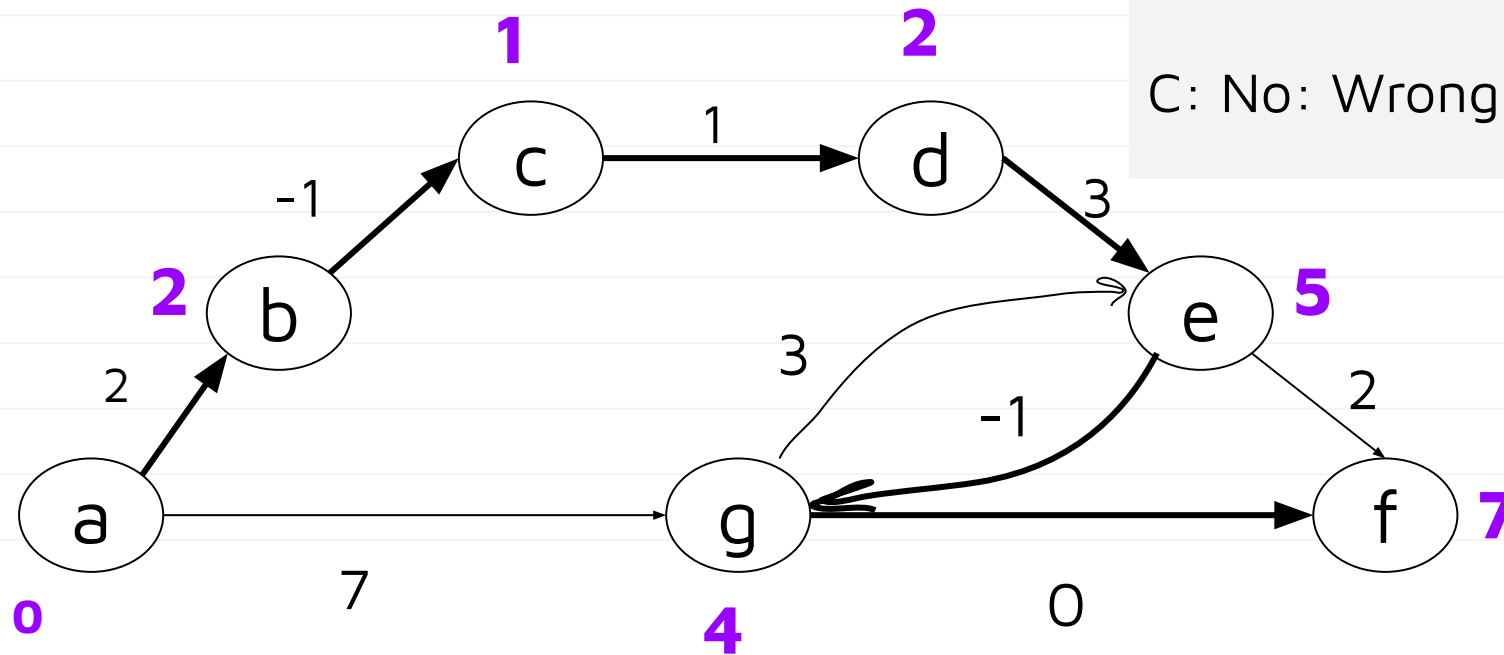
(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

Is everything look correct?

A: Yes, we are done!

B: No: Wrong number

C: No: Wrong predecessor



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

What edge to we need to update?

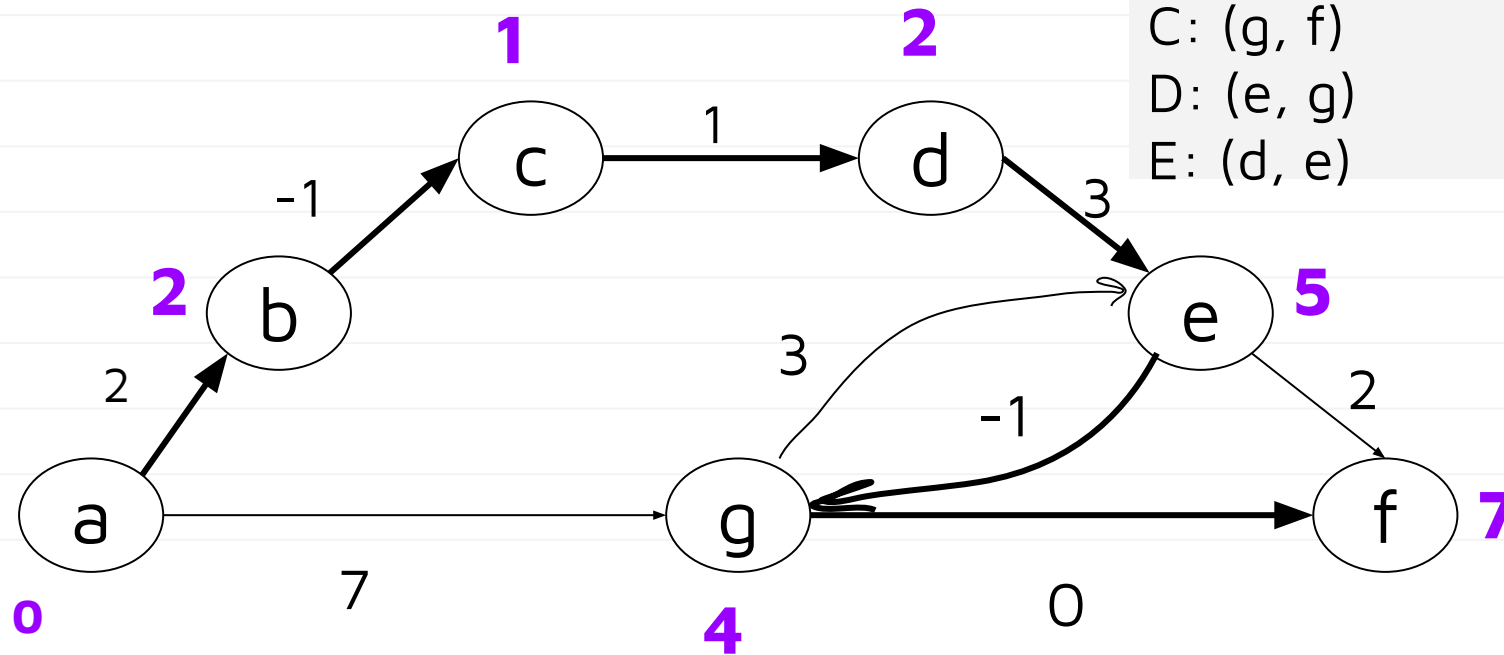
A: All of them

B: (e, f)

C: (g, f)

D: (e, g)

E: (d, e)



Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

What edge to we need to update?

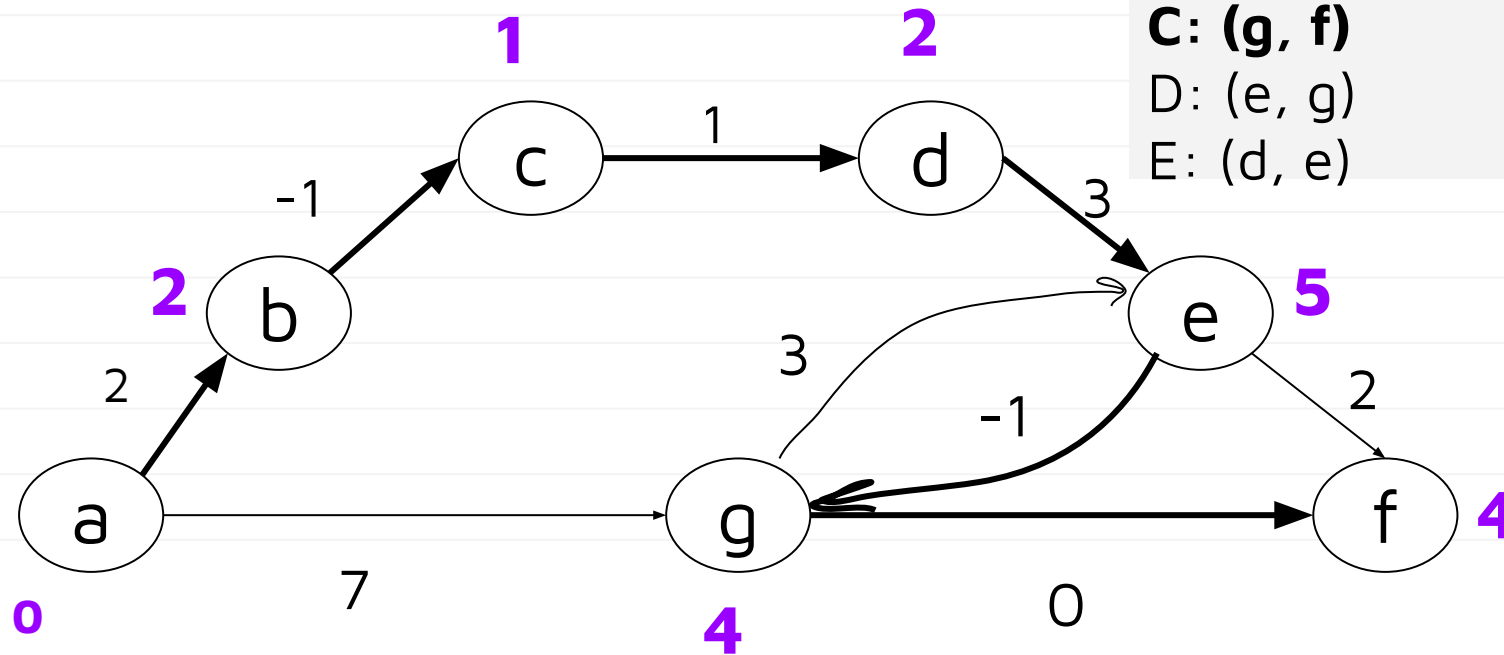
A: All of them

B: (e, f)

C: (g, f)

D: (e, g)

E: (d, e)

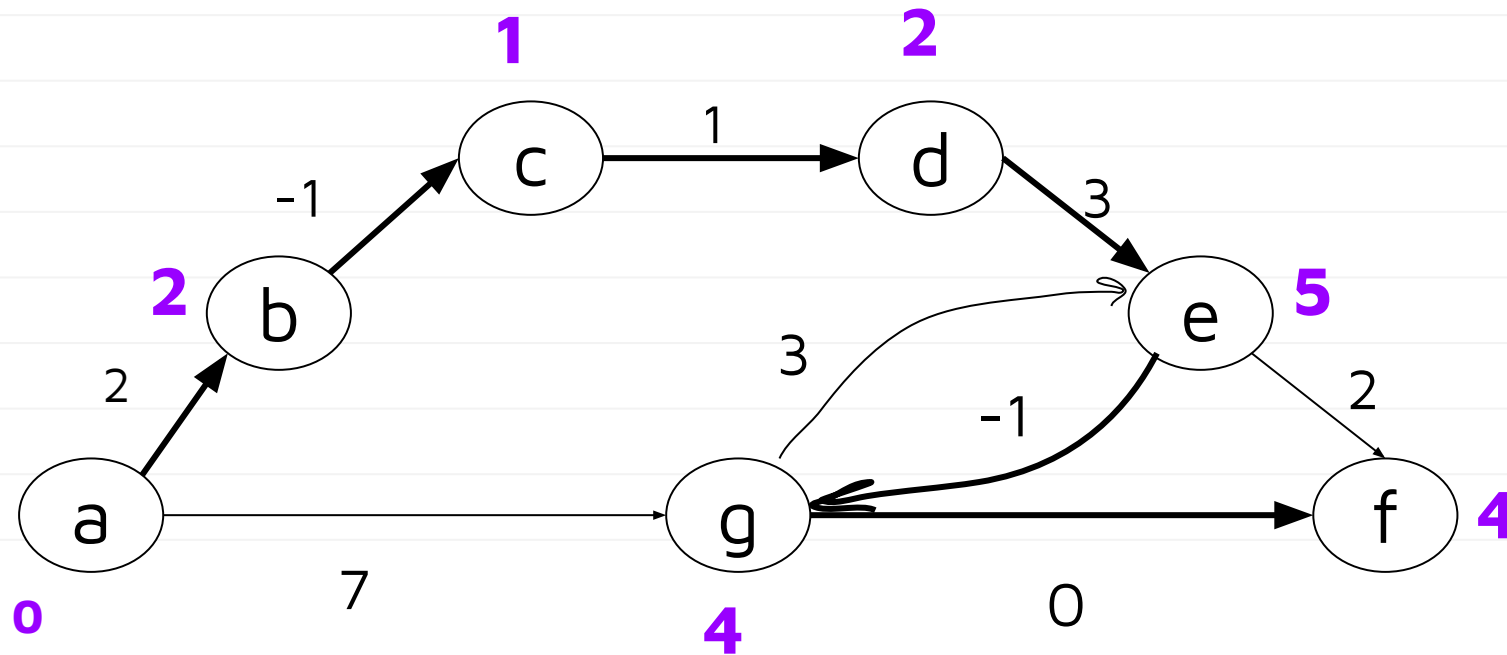


Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

**How many total iterations
did we perform?**

4

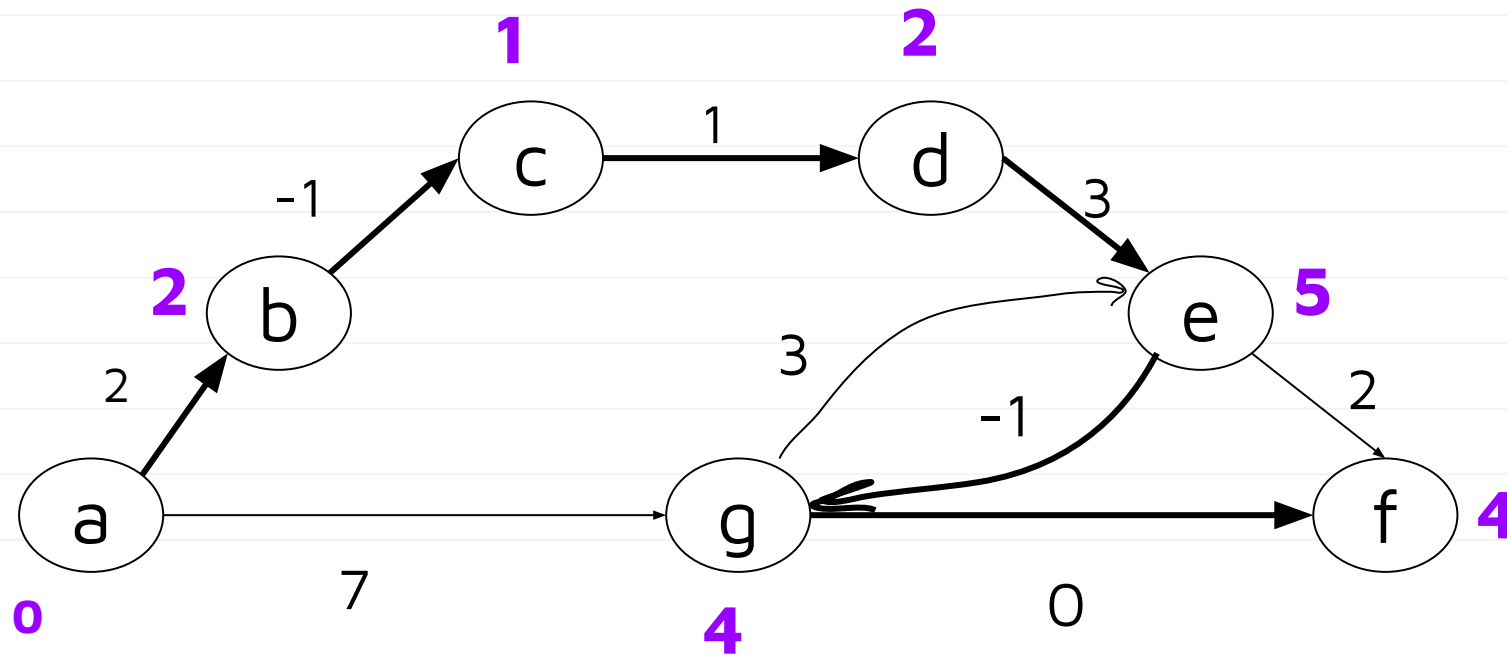


Suppose graph.edges returns edges in following order:

(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

Do we have to run one more iteration?

No



Loop Invariant

- **One** iteration: update *all* edges in arbitrary order.
- **Loop invariant:** After α iterations, all nodes whose ***actual*** shortest path from s has $\leq \alpha$ edges are guaranteed to be estimated **correctly**.

Bellman-Ford

- **Claim:** each node must have a shortest path which is simple (Edge case: cycles of weight zero)
- The most edges a simple path can have is $|V| - 1$
- **Idea of Bellman-Ford:** iteratively update all edges, repeat $|V| - 1$ times.

The Bellman-Ford Algorithm

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)-1):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

The Bellman-Ford Algorithm: Complexity

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)-1):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

Setup: _____time:

Each update takes _____time

There are exactly _____updates

Total time complexity: _____

The Bellman-Ford Algorithm: Complexity

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)-1):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

Setup: $\Theta(V)$ time:

Each update takes _____time

There are exactly _____updates

Total time complexity: _____

The Bellman-Ford Algorithm: Complexity

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)-1):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

Setup: $\Theta(V)$ time

Each update takes $\Theta(1)$ time

There are exactly _____ updates

Total time complexity: _____

The Bellman-Ford Algorithm: Complexity

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)-1):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

Setup: $\Theta(V)$ time

Each update takes $\Theta(1)$ time

There are exactly $(|V|-1) \times |E|$ updates

Total time complexity: _____

The Bellman-Ford Algorithm: Complexity

```
def bellman_ford(graph, weights, source):  
    """Assume graph is directed."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)-1):  
        for (u, v) in graph.edges:  
            update(u, v, weights, est, predecessor)  
    return est, predecessor
```

Setup: $\Theta(V)$ time

Each update takes $\Theta(1)$ time

There are exactly $(|V|-1) \times |E|$ updates

Total time complexity: $\Theta(V + VE)$



Early Stopping and Negative Cycles

Early Stopping

- B-F may not need to run for $|V| - 1$ iterations.
- If no predecessors change, we can break:

Suppose graph.edges returns edges in following order:

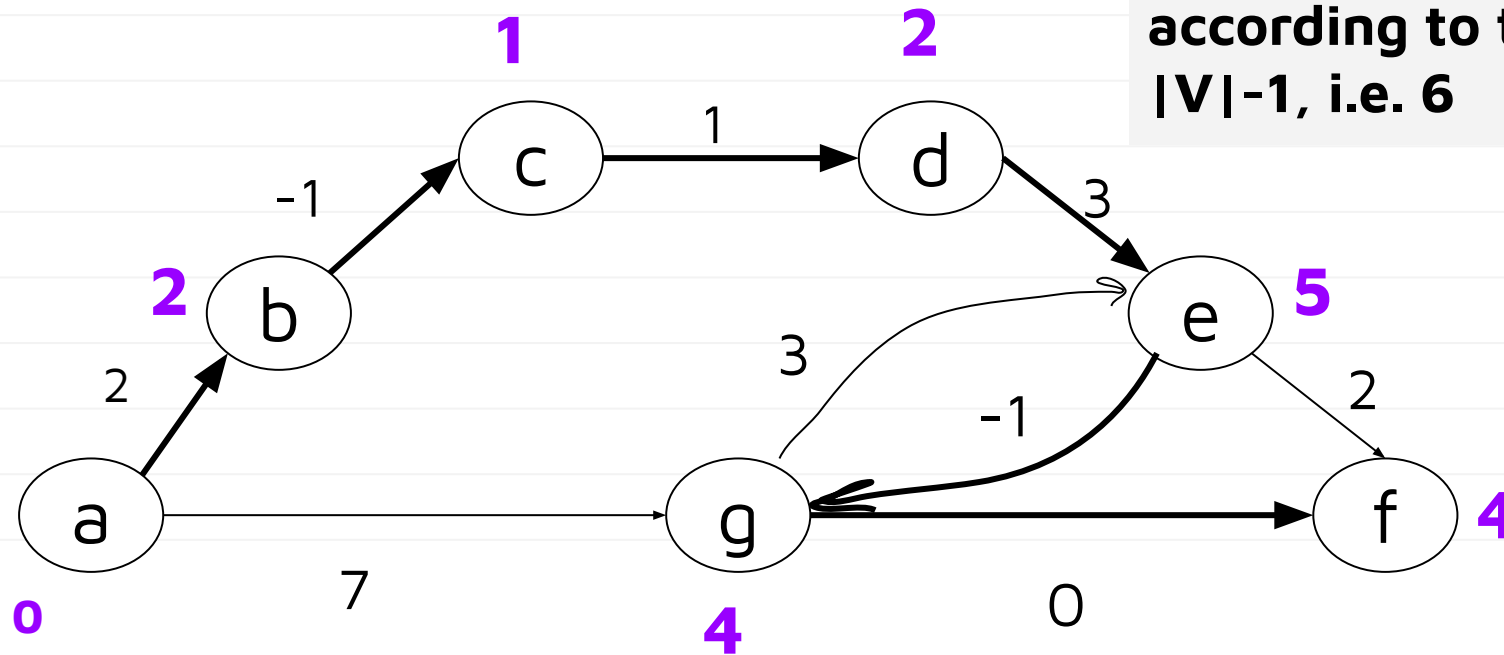
(c, d), (a, b), (b, c), (g, f), (e, g),
(g, e), (d, e), (e, f), (a, g)

**How many total iterations
did we perform?**

4

**How many iterations
according to the code?**

$|V|-1$, i.e. 6



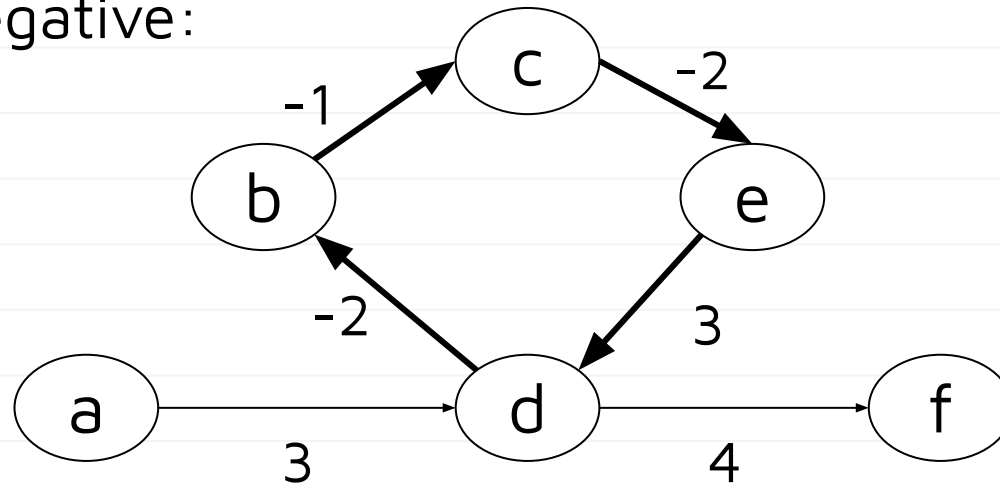
Early Stopping

- B-F may not need to run for $|V| - 1$ iterations.
- If no predecessors change, we can break:

```
def bellman_ford(graph, weights, source):  
    """Early stopping version."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes) - 1):  
        any_changes = False  
        for (u, v) in graph.edges:  
            changed = update(u, v, weights, est, predecessor)  
            any_changes = changed or any_changes  
        if not any_changes:  
            break  
    return est, predecessor
```

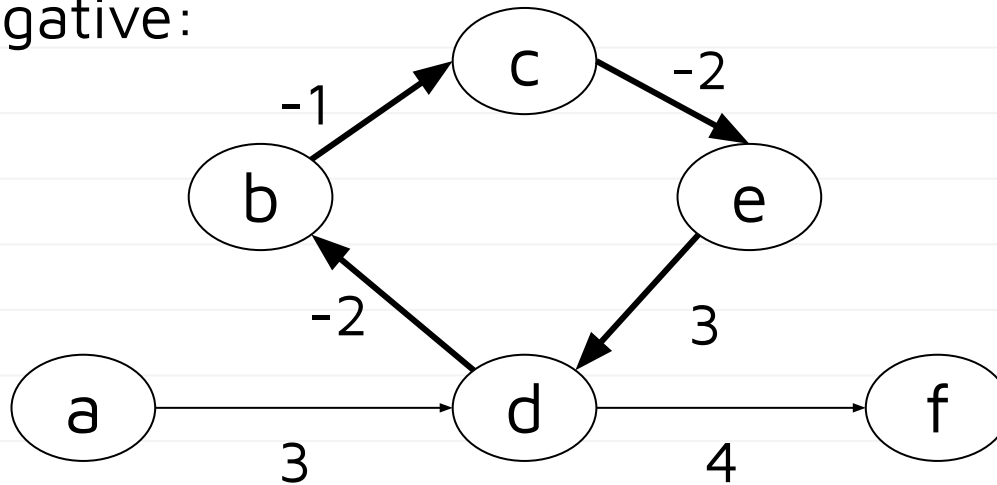
Negative Cycles

- A **negative cycle** is a cycle whose total edge weight is negative:



Negative Cycles

- A **negative cycle** is a cycle whose total edge weight is negative:



- If a graph has a negative cycle, (some) shortest paths are **not well defined**.

Detecting Negative Cycles

- If graph **does not have** negative cycles, estimated distances eventually stop changing (after at most $|V| - 1$ iterations).
- If graph **has** negative cycles, estimated distances *always* decrease.
- To detect them: run a $|V|$ th iteration; if distances change, a negative cycle exists.

Detecting Negative Cycles

```
def bellman_ford(graph, weights, source):  
    """Early stopping version, detects negative cycles."""  
    est = {node: float('inf') for node in graph.nodes}  
    est[source] = 0  
    predecessor = {node: None for node in graph.nodes}  
  
    for i in range(len(graph.nodes)):  
        any_changes = False  
        for (u, v) in graph.edges:  
            changed = update(u, v, weights, est, predecessor)  
            any_changes = changed or any_changes  
        if not any_changes:  
            break  
    # this will be True if negative cycles exist  
    contains_negative_cycles = any_changes  
    return est, predecessor, contains_negative_cycles
```



Thank you!

Do you have any questions?

CampusWire!