

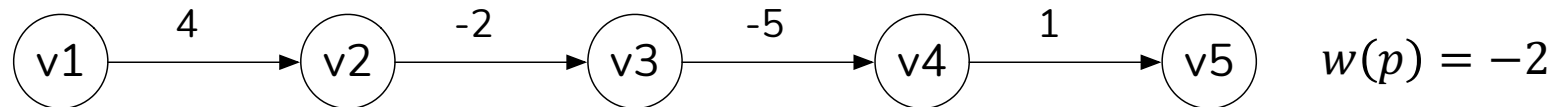
# Shortest Paths

Kristian Guillaumier

# Shortest paths

- Digraph  $G = (V, E)$
- Edge weight function:  $w: E \rightarrow \mathcal{R}$
- Weight of a path  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$  is given by:

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

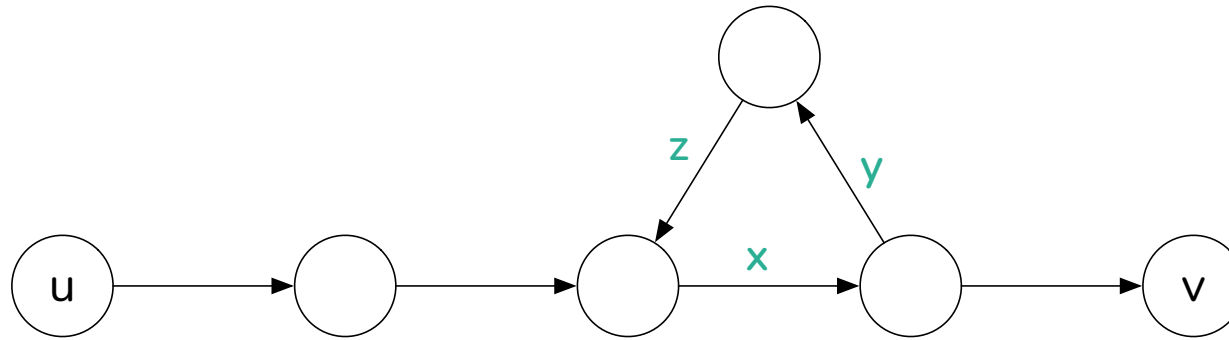


## ...continued

- A shortest path from  $u$  to  $v$  is a **path having the minimum weight** from  $u$  to  $v$ .
- The shortest path weight  $\delta$  from  $u$  to  $v$  is:
  - $\delta(u, v) = \min(w(p))$
  - Where  $p$  is a path from  $u$  to  $v$ .
  - Note that  $\delta$  is the weight of the shortest path, not the shortest path itself.
- If graph is disconnected and there is no path from  $u$  to  $v$ , then  $\delta(u, v) = \infty$ .

## ...continued

- Note: we may encounter a problem if the edges can have negative weights and there are cycles...



If  $x+y+z$  is negative, then the length of the path from  $u$  to  $v$  can be infinitely small. That is:  $w(p) = -\infty$

# Optimal substructure

*Claim: A subpath of a shortest path is a shortest path.*

- Assume I have this **shortest** path:



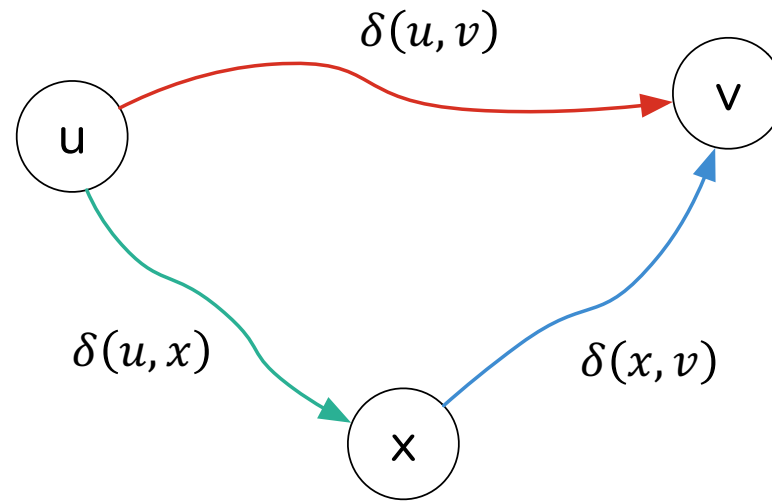
- Suppose that  $x \rightarrow y$  is not a shortest path, and there exists another one:



- This means that there is a shorter path from  $u$  to  $v$ . **Contradiction.**

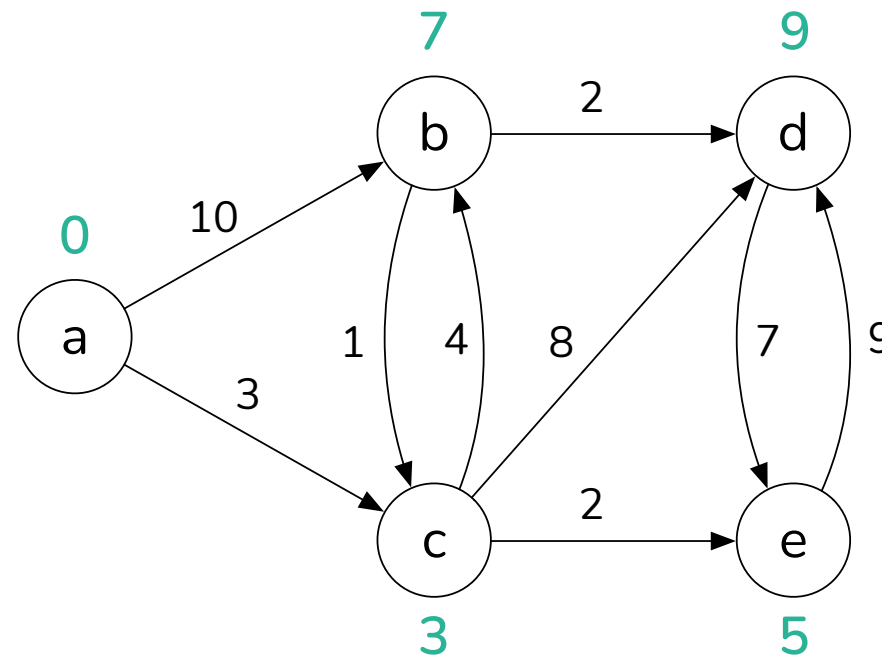
# Triangle inequality

- For all  $u, v, x \in V$
- $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$



# Single source shortest paths

- Given a source vertex  $s \in V$  find the shortest path weight  $\delta(s, v)$  to every vertex  $v \in V$ .
- Requirement: all weights are  $\geq 0$ .



Green labels show the cost of the shortest path from the starting state to a vertex.

This is what an SSSP algorithm such as Dijkstra will find.

# SSSP method

- Maintain a set  $S$  of vertices whose shortest path weight from some vertex  $s$  is known.
  - Note: the shortest path weight from  $s$  to  $s$  is known:  $\delta(s, s) = 0$ .
  - So, the vertex  $s$  is in the set  $S$ .
- For every vertex  $v \in V - S$ :
  - Add to  $S$  the vertex whose distance from  $s$  is minimal.
  - Update distance estimates for vertices adjacent to  $v$ .



# Dijkstra's algorithm

$d[s] = 0$

for each  $v$  in  $V - \{s\}$

$d[v] = \infty$

$S = \emptyset$

$Q = V = \{s\}$

while  $Q \neq \emptyset$

$u = \text{DequeueMin}(Q)$

$S = S \cup \{u\}$

for each  $v \in \text{Adj}[u]$

if  $d[v] > d[u] + w(u, v)$

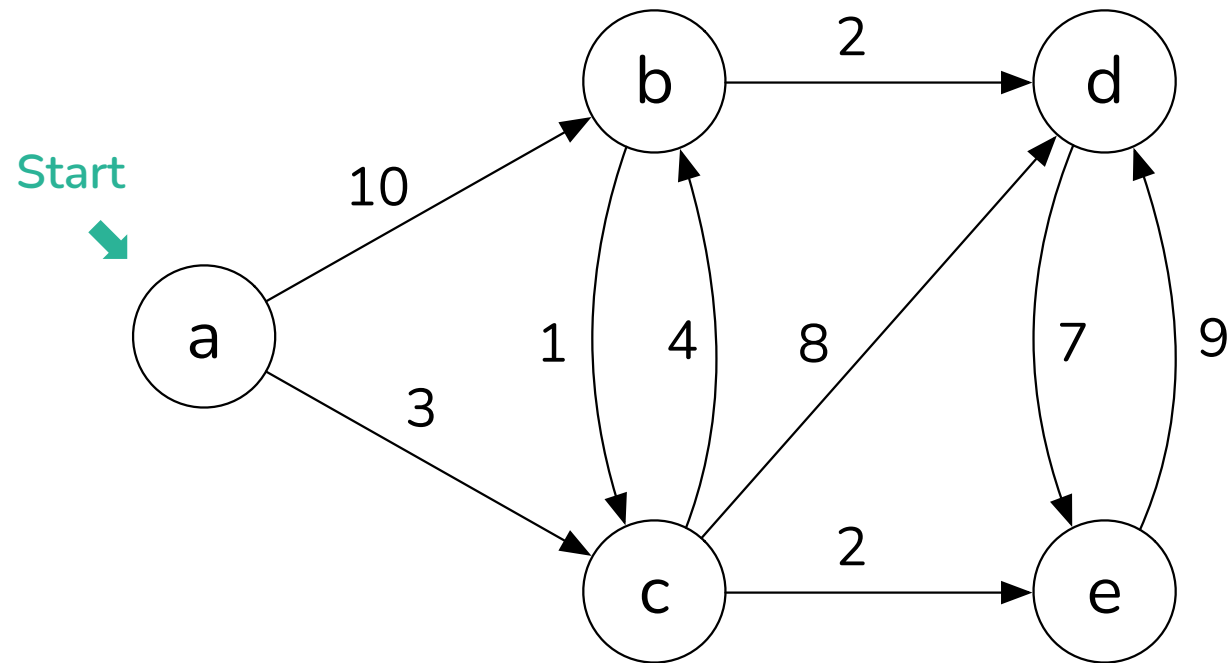
$d[v] = d[u] + w(u, v)$

$d$  is an array indexed by vertex that keeps the **estimate** distance from  $s$  to that vertex.

$Q$  is a priority queue of vertices indexed by distance estimate (i.e., distance is the key).

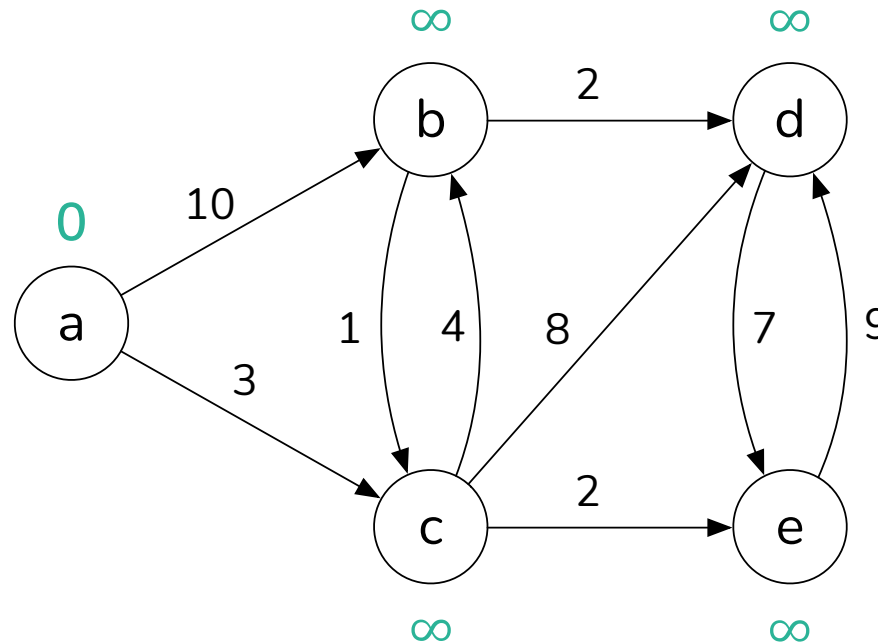
This is called the 'relaxation step'.

# Dijkstra's algorithm



# Initialise

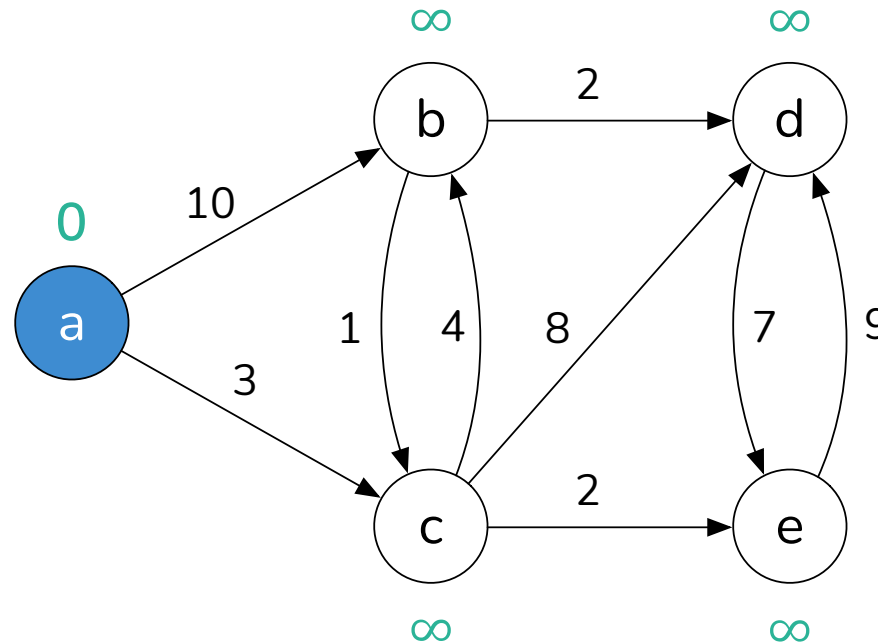
$d[s] = 0$   
for each  $v$  in  $V - \{s\}$   
     $d[v] = \infty$   
 $S = \emptyset$   
 $Q = V$



Queue				
a	b	c	d	e
0	$\infty$	$\infty$	$\infty$	$\infty$

Set
$S = \{ \}$

# Dequeue minimum



`u = DequeueMin(Q)`

`S = S ∪ {u}`

for each  $v \in \text{Adj}[u]$

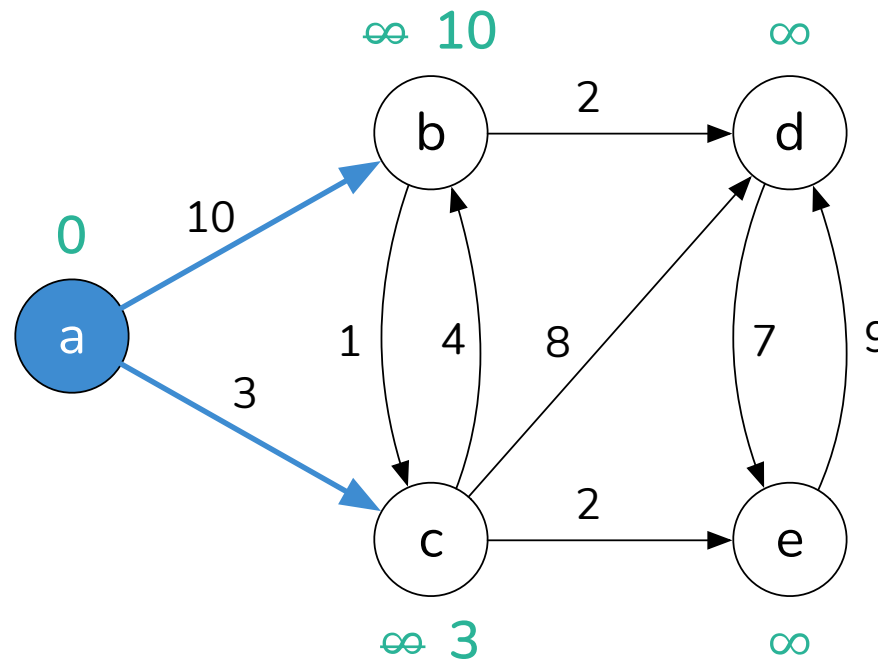
    if  $d[v] > d[u] + w(u, v)$

$d[v] = d[u] + w(u, v)$

Queue				
<del>a</del>	b	c	d	e
0	∞	∞	∞	∞

Set
$S = \{a\}$

# Fix edges adjacent to 'a'

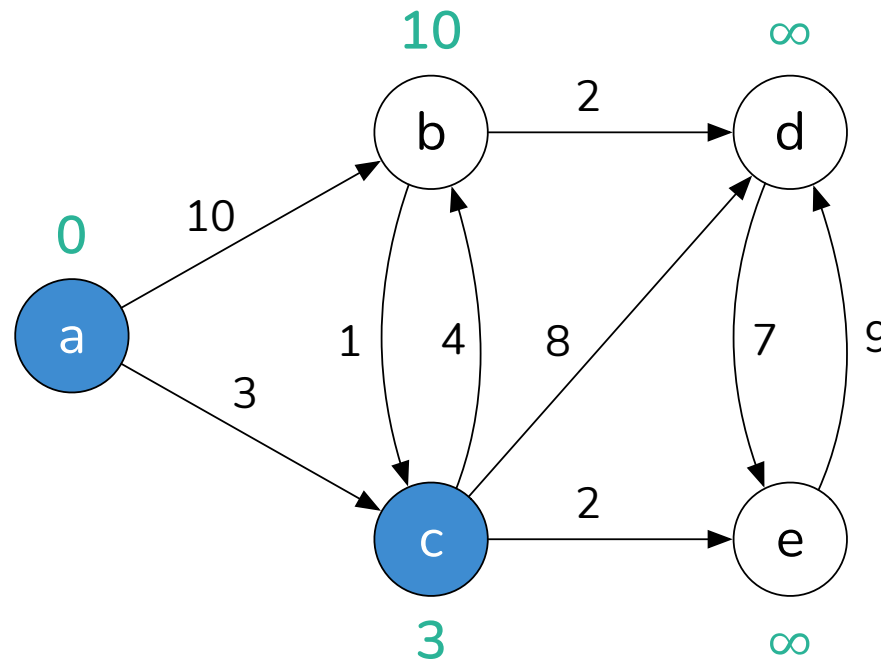


Queue				
<del>a</del>	b	c	d	e
$\emptyset$	$\infty$	$\infty$	$\infty$	$\infty$
	10	3	$\infty$	$\infty$

Set
$S = \{a\}$

```
u = DequeueMin(Q)
S = S ∪ {u}
for each v ∈ Adj[u]
    if d[v] > d[u] + w(u, v)
        d[v] = d[u] + w(u, v)
```

# Dequeue minimum



Queue				
<del>a</del>	b	<del>c</del>	d	e
0	$\infty$	$\infty$	$\infty$	$\infty$
	10	3	$\infty$	$\infty$

Set
$S = \{a, c\}$

$u = \text{DequeueMin}(Q)$

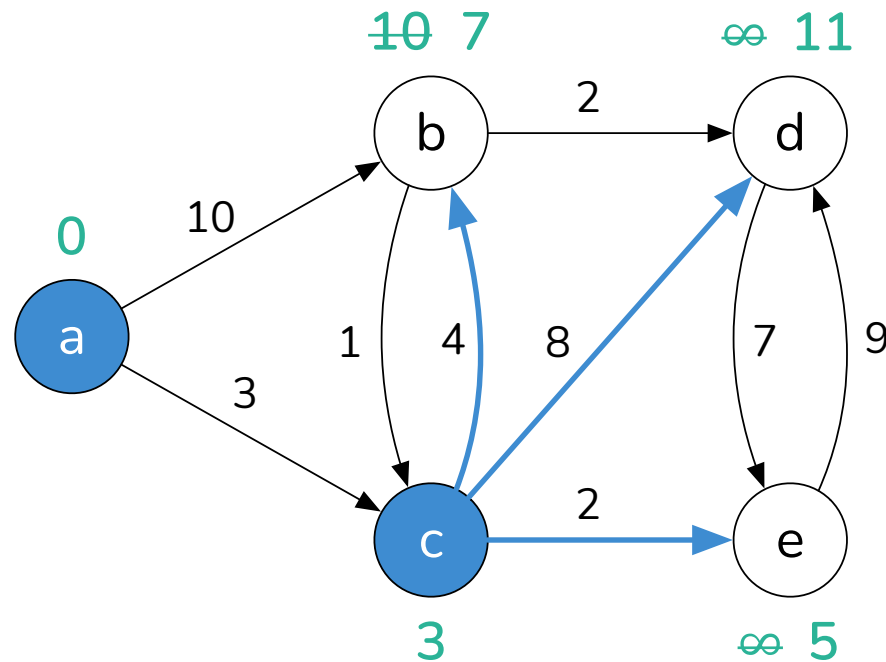
$S = S \cup \{u\}$

for each  $v \in \text{Adj}[u]$

if  $d[v] > d[u] + w(u, v)$

$d[v] = d[u] + w(u, v)$

# Fix edges adjacent to 'a'



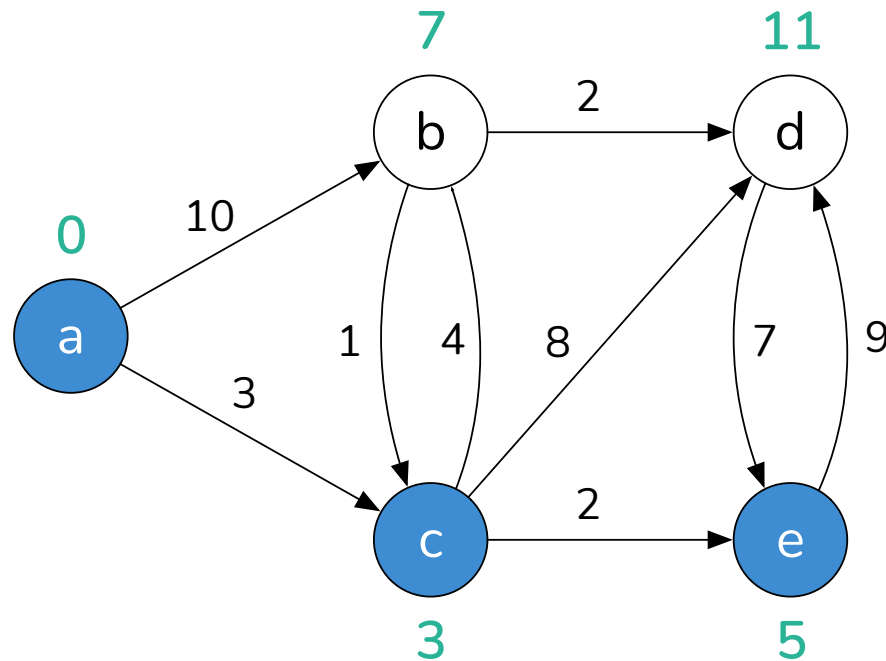
Queue				
<del>a</del>	b	<del>e</del>	d	e
<del>0</del>	$\infty$	$\infty$	$\infty$	$\infty$
	10	<del>3</del>	$\infty$	$\infty$
	7		11	5

Set
$S = \{a, c\}$

```

u = DequeueMin(Q)
S = S ∪ {u}
for each v ∈ Adj[u]
    if d[v] > d[u] + w(u, v)
        d[v] = d[u] + w(u, v)
    
```

# Dequeue minimum



$u = \text{DequeueMin}(Q)$

$S = S \cup \{u\}$

for each  $v \in \text{Adj}[u]$

if  $d[v] > d[u] + w(u, v)$

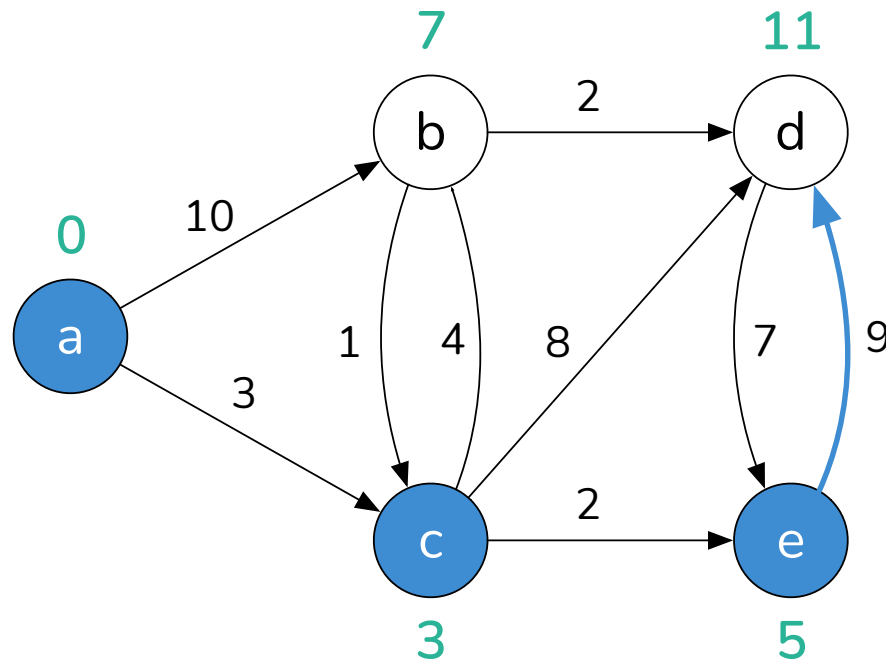
$d[v] = d[u] + w(u, v)$

Queue				
<del>a</del>	b	<del>e</del>	d	<del>e</del>
<del>0</del>	$\infty$	$\infty$	$\infty$	$\infty$
	10	<del>3</del>	$\infty$	$\infty$
	7		11	<del>5</del>

Set
$S = \{a, c, \text{e}\}$



# Fix edges adjacent to 'e'



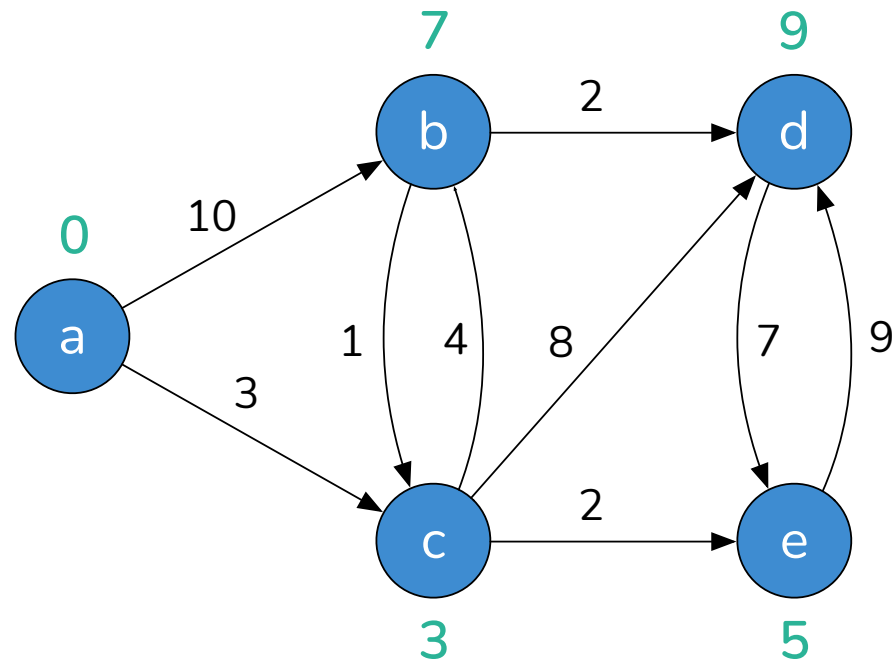
```

u = DequeueMin(Q)
S = S ∪ {u}
for each v ∈ Adj[u]
    if d[v] > d[u] + w(u, v)
        d[v] = d[u] + w(u, v)
  
```

Queue				
<del>a</del>	b	<del>e</del>	d	<del>e</del>
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5

Set
S={a,c,e}

# And so on...



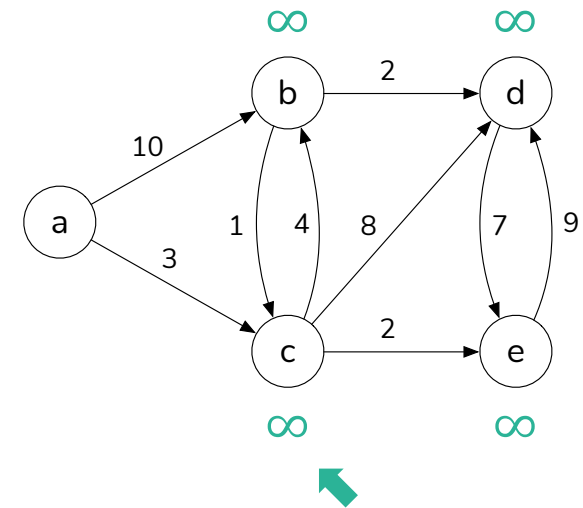
Queue				
<del>a</del>	<del>b</del>	e	<del>d</del>	<del>e</del>
<del>0</del>	$\infty$	$\infty$	$\infty$	$\infty$
	10	<del>3</del>	$\infty$	$\infty$
	7		11	<del>5</del>
	<del>7</del>		11	
			<del>11</del>	

Set
$S=\{a,c,e,b,d\}$


# Lemma 1

*Initialising  $d[s] = 0$  and  $d[v] = \infty$  for all  $v \in V - \{s\}$  means that  $d[v] \geq \delta(s, v)$  for all  $v \in V$  for any sequence of relaxation steps.*

- This is the “**overestimate**” lemma.
- **Base case:**  $d[s] = 0, d[v] = \infty$ 
  - Is  $d[s] \geq \delta(s, s)$ ? Is  $0 \geq 0$ ? Yes.
  - Is  $d[v] \geq \delta(s, v)$ ? Is  $\infty \geq$  any possible value? Yes.
  - Base case proven.



These values will always be overestimates ( $\geq$ )

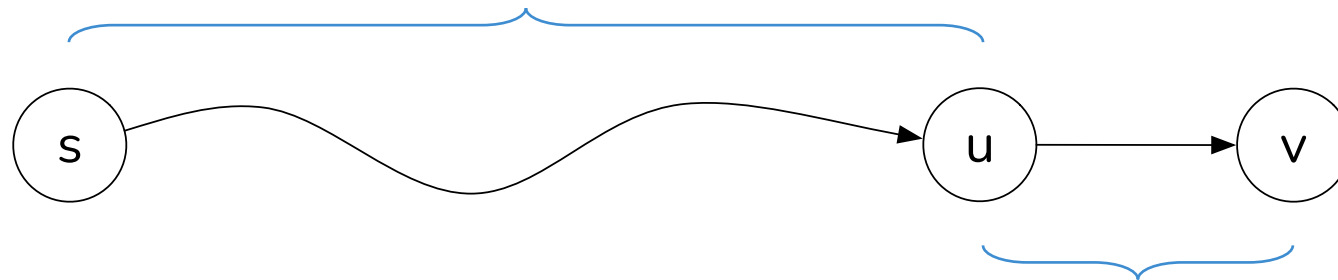
- Suppose that the lemma is not true.
- So, suppose that  $d[v] < \delta(s, v)$ . We are proving by induction so all other cases before  $d[v]$  did not fail.
- So, at some point we set  $d[v] = d[u] + w(u, v)$ .  Only part of the algorithm that can change  $d[v]$
- So, we'd get  $d[u] + w(u, v) < \delta(s, v)$ . **a**
- But we know that  $d[u] \geq \delta(s, u)$  – remember that  $d[u]$  is before  $d[v]$  so by our induction hypothesis, it must be correct.
- So,  $d[u] + w(u, v) \geq \delta(s, u) + w(u, v)$ . Note that I simply added a constant  $w(u, v)$  to both sides.
- $w(u, v)$  is the weight of the edge  $u$  to  $v$  which is certainly  $\geq$  the shortest path from  $u$  to  $v$ . The length of the shortest path from  $u$  to  $v$  is written as  $\delta(u, v)$  and must then be  $\leq w(u, v)$ .
- So,  $d[u] + w(u, v) \geq \delta(s, u) + \delta(u, v)$ . By triangle inequality we get:
- $d[u] + w(u, v) \geq \delta(s, v)$ . **b**
- Contradiction. **a b**

# Lemma 2

*Suppose I have a shortest path...*

$$s \rightarrow \dots \rightarrow u \rightarrow v$$

*if  $d[u] = \delta(s, u)$  and we relax edge  $(u, v)$  then  $d[v] = \delta(s, v)$ .*



# Proving lemma 2

- Part 1:
  - $\delta(s, v) = w(s \rightarrow \dots \rightarrow u) + w(u, v)$
  - By optimal substructure  $w(s \rightarrow \dots \rightarrow u) = \delta(s, u)$  so  $\delta(s, v) = \delta(s, u) + w(u, v)$  - remember that, by assumption,  $\delta(s, u) = d[u]$ .
- Remember that relaxation can only **decrease** the values in  $d[ ]$ . The relaxation step is:

if  $d[v] > d[u] + w(u, v)$

then  $d[v] = d[u] + w(u, v)$



## ...continued

- **Lemma 1** says that the values in  $d[ ]$  are always **overestimates**. So, for some  $v$ , the value in  $d[v]$  is either already equal to  $\delta(s, v)$  or it must be greater than  $\delta(s, v)$ .
- **Case 1** – if  $d[v]$  **is equal to**  $\delta(s, v)$  then we won't 'pass' the **if** statement, and  $d[v]$  will remain unchanged and equal to  $\delta(s, v)$ .

if  $d[v] > d[u] + w(u, v)$

then  $d[v] = d[u] + w(u, v)$

# ...continued

- **Case 2** –  $d[v]$  is greater than  $\delta(s, v)$  so we will ‘pass’ the if statement.
  - Remember that in part 1, by optimal substructure, we know that  $\delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$ .
  - If  $d[v] > \delta(s, v)$  we will pass the if statement and set  $d[v]$  to  $d[u] + w(u, v)$  which is  $\delta(s, v)$ .
- Ready.



# Proof of correctness

*When Dijkstra terminates,  $d[v] = \delta(s, v)$  for all  $v \in V$ .*

- Recall that the algorithm does not change the value of  $d[v]$  once we add it to the set  $S$ .
- So “all we have to do” is show that  $d[v] = \delta(s, v)$  when  $v$  has been added to  $S$ .

## ...continued

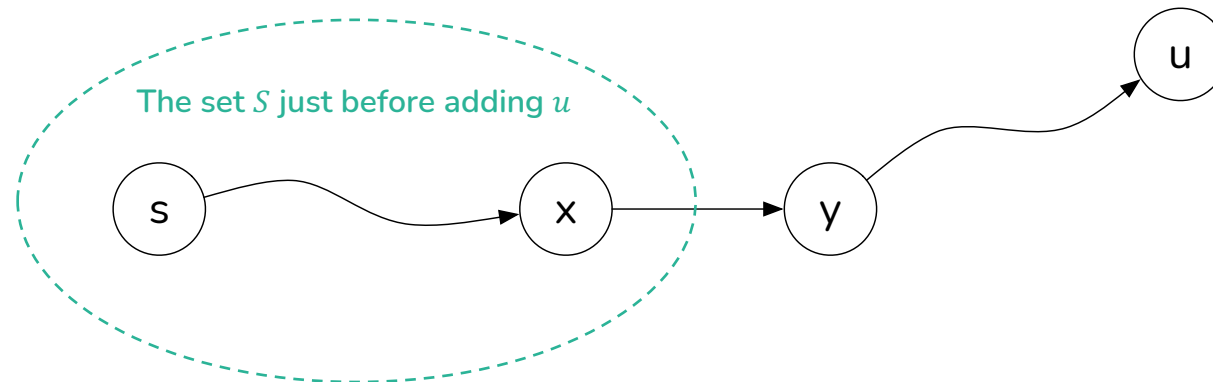
- We will prove by contradiction: suppose we are **just about** to add  $u$  to  $S$  and  $d[u] \neq \delta(s, u)$ .
- If it is not equal to, then **by lemma 1 it must be greater:**

$$d[u] > \delta(s, u) \quad \text{a}$$


- Now, let  $p$  be a shortest path from  $s$  to  $u$ .
- So,  $w(p) = \delta(s, u)$ .

# ...continued

- Remember that  $s$  is in  $S$  and  $u$  has not been added yet (we were just about to add it).
- All the values  $d[\ ]$  in  $S$  are correct because they are in  $S$ .
- Adding  $d[u]$  is going to be our first violation.
- Consider the first edge  $(x, y)$  where  $p$  exits  $S$ .



# ...continued

- $d[x] = \delta(s, x)$  is correct since it is in  $S$ .
- The algorithm says that after we put  $x$  in  $S$  we must relax all edges emanating from  $x$ . We will therefore relax the edge from  $x$  to  $y$ .
- By lemma 2, if  $d[x]$  is the shortest path weight (and it is because it's in  $S$ ), if we relax the edge  $(x, y)$  then  $d[y] = \delta(s, y)$ .
- Now note that  $\delta(s, y)$  can only be  $\leq \delta(s, u)$ .
- And my initial assumption is that  $d[u]$  is strictly  $> \delta(s, u)$ . 
- So, organising everything we know so far, we get...
- $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$ .

## ...continued

- Now, remember... which data structure are we using to storing the vertices outside of  $S$ ? We are using a min-queue.
- This means that if we chose to add the vertex  $u$  in  $S$  and not the vertex  $y$ , it was because  $d[u] \leq d[y]$ .
- Arrange again...

$$d[u] \leq d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$$



Contradiction

# Analysis

$d[s] = 0$

for each  $v$  in  $V - \{s\}$

$d[v] = \infty$

$S = \emptyset$

$Q = V$

while  $Q \neq \emptyset$

$u = \text{DequeueMin}(Q)$

$S = S \cup \{u\}$

for each  $v \in \text{Adj}[u]$

if  $d[v] > d[u] + w(u, v)$

$d[v] = d[u] + w(u, v)$

Linear time.

Degree( $u$ )  
times.

$|V|$  times.

# ...continued

- So, we have  $|V|$  DequeueMins.
- And Degree(u) UpdateKeys for  $|V|$  times giving  $O(E)$  UpdateKeys.
- Total running time then is:
  - $O(V) \times \text{TimeTo}(\text{DequeueMin}) + O(E) \times \text{TimeTo}(\text{UpdateKey})$ .
  - TimeTo() depends on the data structure I use to implement the Queue.

Remember the handshaking lemma:

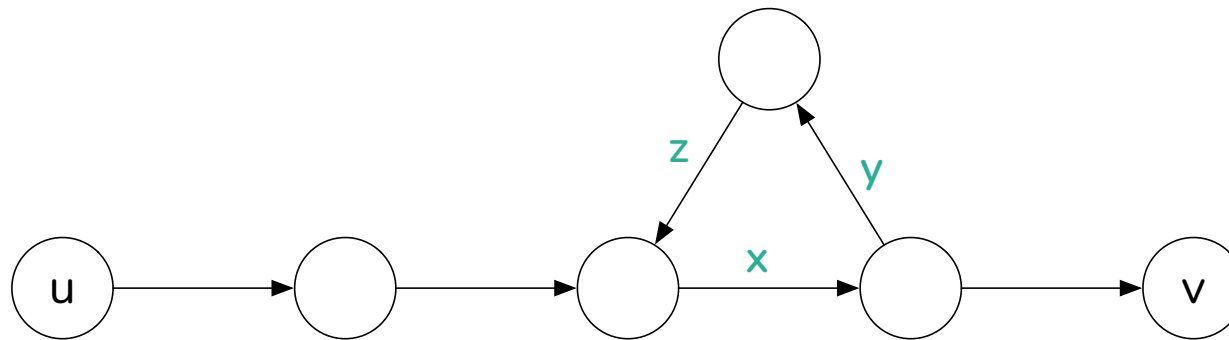
$$\sum_{v \in V} \text{Degree}(v) = 2 \times |E|$$

Q Data Structure	DequeueMin	UpdateKey	Total
Unsorted Array	$O(V)$	$O(1)$	$O(V^2) + O(E) = O(V^2)$
Min Heap	$O(\log_2 V)$	$O(\log_2 V)$	$O(V \cdot \log_2(v) + O(E \cdot \log_2(v))) = O(E \cdot \log_2(v))$

Note: if graph is dense,  $E$  approaches  $V^2$  so Unsorted Array is better. If graph is sparse, Min Heap is better.

# Bellman-Ford

- This will allow us to generate a shorter path than any other one. We cannot find a shortest path using Dijkstra.
- Bellman-Ford computes  $\delta(s,v)$  even if we have **negative weights** but reports negative weight cycles (the shortest path will be a **simple path** – no vertex repetitions are allowed).



If  $x+y+z$  is negative, then the length of the path from  $u$  to  $v$  can be infinitely small. That is:  $w(p) = -\infty$



# ...continued

$d[s] = 0$



This initial estimate of 0 may not necessarily be true since now we have negative weights.

$\forall v \in V - \{s\}$

$d[v] = \infty$

for  $(|V|-1)$  times

for each edge  $(u,v) \in E$

if  $d[v] > d[u] + w(u,v)$

$d[v] = d[u] + w(u,v)$



The relaxation step.

# Detecting negative weight cycles

$d[s] = 0$

$\forall v \in V - \{s\}, d[v] = \infty$

for  $(|V|-1)$  times

    for each edge  $(u,v) \in E$

        if  $d[v] > d[u] + w(u,v)$

$d[v] = d[u] + w(u,v)$

for  $(|V|-1)$  times

    for each edge  $(u,v) \in E$

        if  $d[v] > d[u] + w(u,v)$

            Report -ve weight cycles

← Look for problems here.

If no -ve weight cycles found then  $d[v] = \delta(s, v)$

# Running time

$d[s] = 0$

$\forall v \in V - \{s\}, d[v] = \infty$

} Linear time.

for  $(|V|-1)$  times

for each edge  $(u,v) \in E$

if  $d[v] > d[u] + w(u,v)$

$d[v] = d[u] + w(u,v)$

}  $|E|$

}  $|V|$

for  $(|V|-1)$  times

for each edge  $(u,v) \in E$

if  $d[v] > d[u] + w(u,v)$

Report -ve weight cycles

}  $|E|$

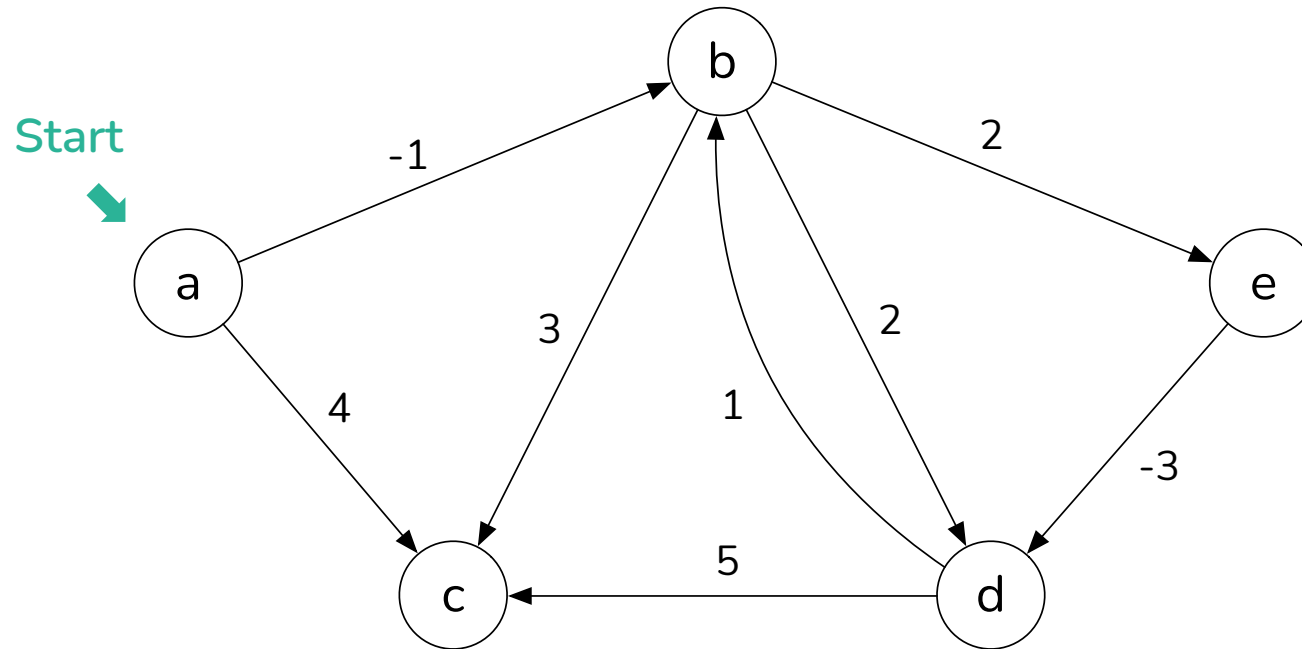
}  $|V|$

$VE + VE = O(VE)...$   
so at least quadratic in  $V$ .

(the number of edges must be  
at least the  $V-1$  since the  
graph needs to be connected)

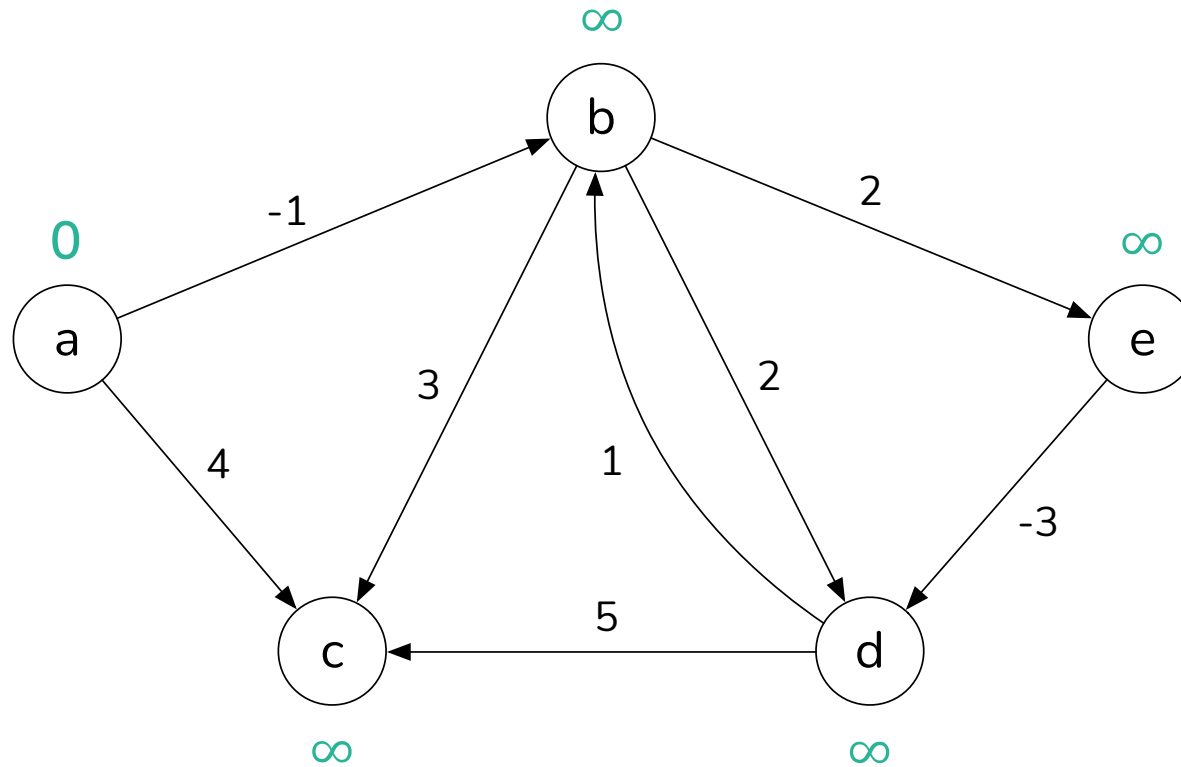
If no -ve weight cycles found then  $d[v] = \delta(s, v)$

# A simple example

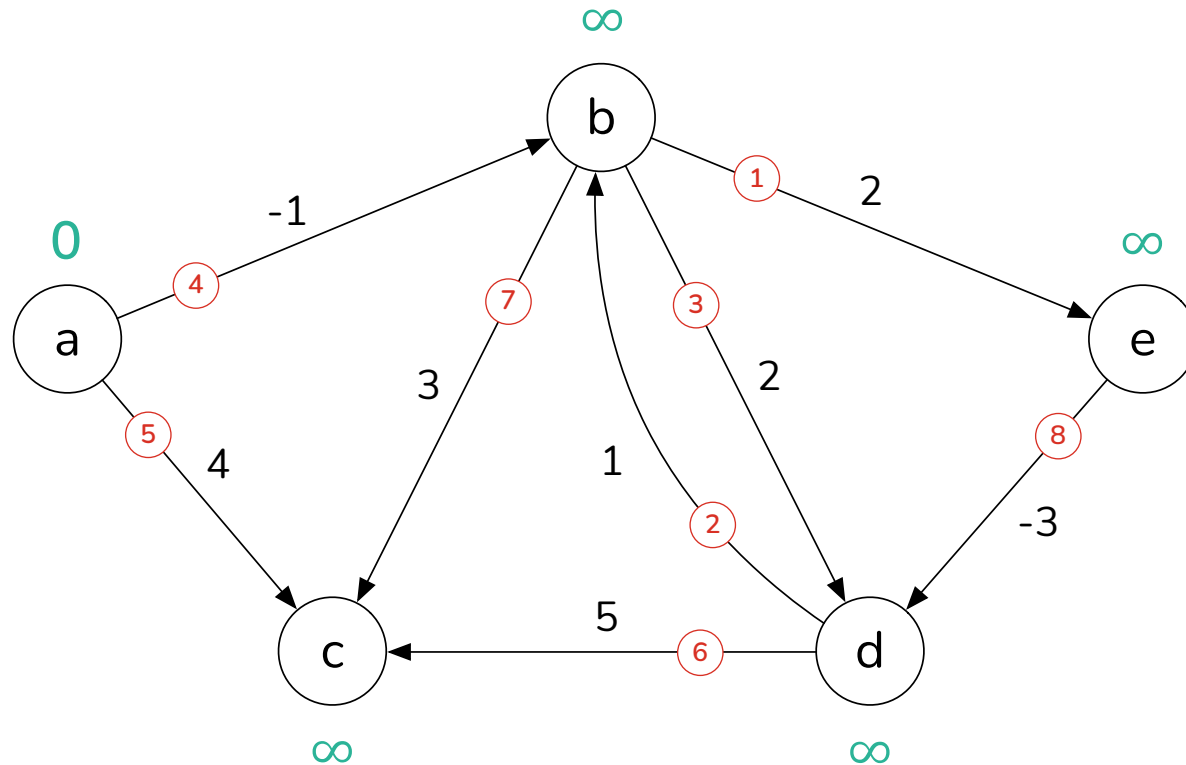


# Initialisation

$d[s] = 0$   
 $\forall v \in V - \{s\}$   
 $d[v] = \infty$



# Picking an edge order



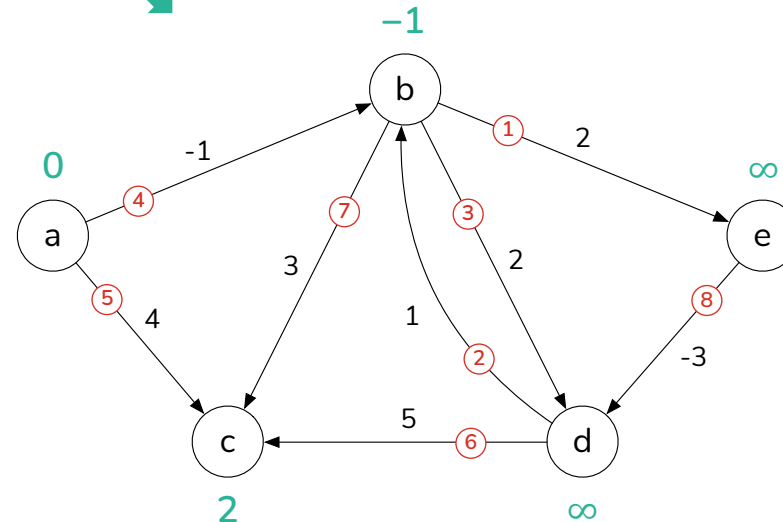
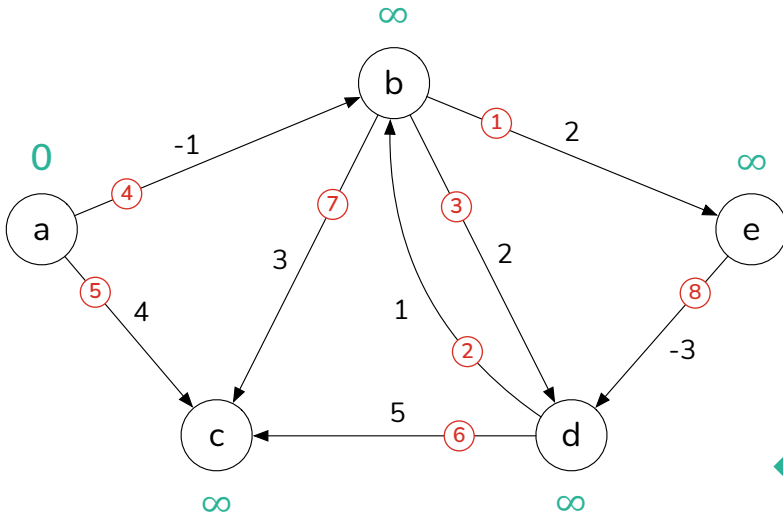
Note that the algorithm says that we must loop through all the edges in the graph (**for each edge  $(u,v) \in E$** ) but the order is not specified (and not important for the algorithm to work. As a matter of fact, the order can also change for every iteration of  $V$ ).

However, to work out the example consistently we will pick an order to work with.

Remember that the edge processing order is not important, we are doing this only for the dry run.

# Pass 1

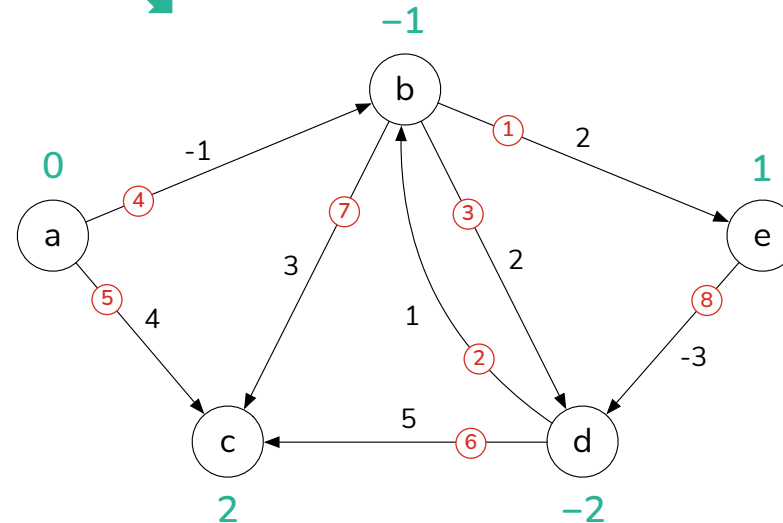
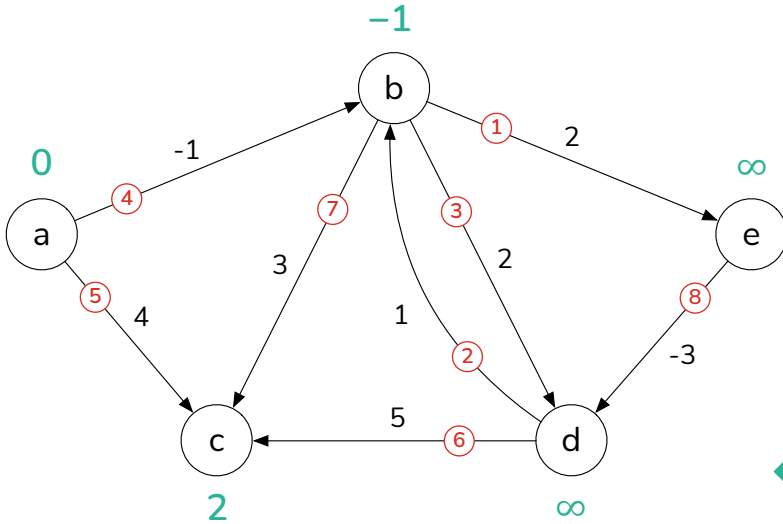
if  $d[v] > d[u] + w(u,v)$   
then  $d[v] = d[u] + w(u,v)$



- 1: Relax (b,e), so  $d[e] = \infty$
- 2: Relax (d,b), so  $d[b] = \infty$
- 3: Relax (b,d), so  $d[d] = \infty$
- 4: Relax (a,b), so  $d[b] = -1$
- 5: Relax (a,c), so  $d[c] = 4$
- 6: Relax (d,c), so  $d[c] = 4$
- 7: Relax (b,c), so  $d[c] = 2$
- 8: Relax (e,d), so  $d[d] = \infty$

# Pass 2

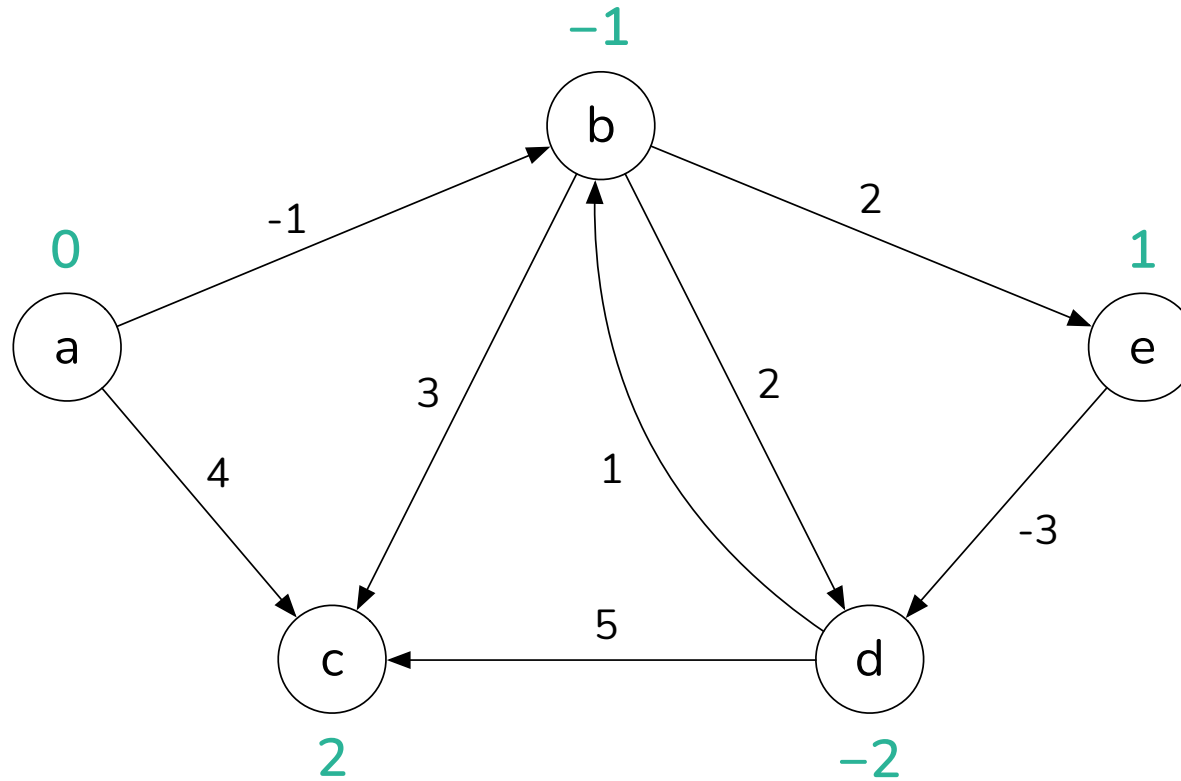
if  $d[v] > d[u] + w(u,v)$   
then  $d[v] = d[u] + w(u,v)$



- 1: Relax (b,e), so  $d[e] = 1$
- 2: Relax (d,b), so  $d[b] = -1$
- 3: Relax (b,d), so  $d[d] = 1$
- 4: Relax (a,b), so  $d[b] = -1$
- 5: Relax (a,c), so  $d[c] = 2$
- 6: Relax (d,c), so  $d[c] = 2$
- 7: Relax (b,c), so  $d[c] = 2$
- 8: Relax (e,d), so  $d[d] = -2$



# Passes 3 and 4 (remember that 4 is $|V| - 1$ )




Nothing happens in step 3, so we can just break and not bother with step 4.

# Proof of correctness

*If  $G = (V, E)$  contains no negative weight cycles, after Bellman-Ford completes  $d[v] = \delta(s, v)$  for all  $v \in V$ .*

- Consider any  $v \in V$  and let  $p$  be the **shortest path** from  $s$  to  $v$  having the **minimum** number of edges.

$$p = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k$$



- Remember the previous ‘overestimate’ lemma that showed  $d[v] \geq \delta(s, v)$ .

## ...continued

- Also remember that by enforcing that  $p$  has the minimum number of edges we are guaranteeing that  $p$  is a **simple path**.
  - Note that not all shortest paths are necessarily simple.
  - Negative weight cycles are not present because the algorithm is defined not to work for negative weight cycles in the first place.
  - This added 'minimum number of edges' constraint guarantees that we do not have any zero weight cycles.

# ...continued

- By optimal substructure (proven before for Dijkstra):

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$$

- Base case:
  - $d[v_0]$  is initialised to 0, remember  $v_0$  is  $s$ .
  - What is the shortest path from  $s$  to  $s$ ? It must be 0 because the only case in which it could be less is if we have a negative weight cycle (negative loop) which is not allowed.
  - So  $d[v_0]$  which is  $d[s] = 0 = \delta(s, v_0) = \delta(s, s)$ .
  - Base case ready.

# ...continued

- **Repeat** the following process  $i$  times:
  - During the  $i^{th}$  round, we relax all the edges, which will certainly include the edge  $(v_{i-1}, v_i)$ .
  - By the previous lemma (Dijkstra lemma 2): If  $d[v_{i-1}] = \delta(s, v_{i-1})$  and we relax the edge  $(v_{i-1}, v_i)$  then  $d[v_i] = \delta(s, v_i)$
- Note that in the above process, after every  $i^{th}$  iteration, **we optimize at least one edge** in the path.
- How many rounds must we do? How large must  $i$  be? How many vertices do we have in a simple path with  $|V|$  vertices? There are  $V - 1$ . This is exactly how much we are iterating in the outer loop.
- Done.



for  $(|V|-1)$  times  
for each edge  $(u,v) \in E$   
relax edge

# Corollary

*If  $d[v]$  fails to converge after  $|V| - 1$  cycles, then there is a negative weight cycle in  $G$  from  $s$ .*

# Further reading

- These notes should be supplemented by:
  - Introduction to Algorithms (Clifford Stein, Thomas H Cormen, Ronald L Rivest, Charles E Leiserson – MIT Press)
  - Extra material from Erik Demaine, Massachusetts Institute of Technology, MIT, [http://videolectures.net/mit6046jf05\\_demaine\\_lec17/](http://videolectures.net/mit6046jf05_demaine_lec17/))