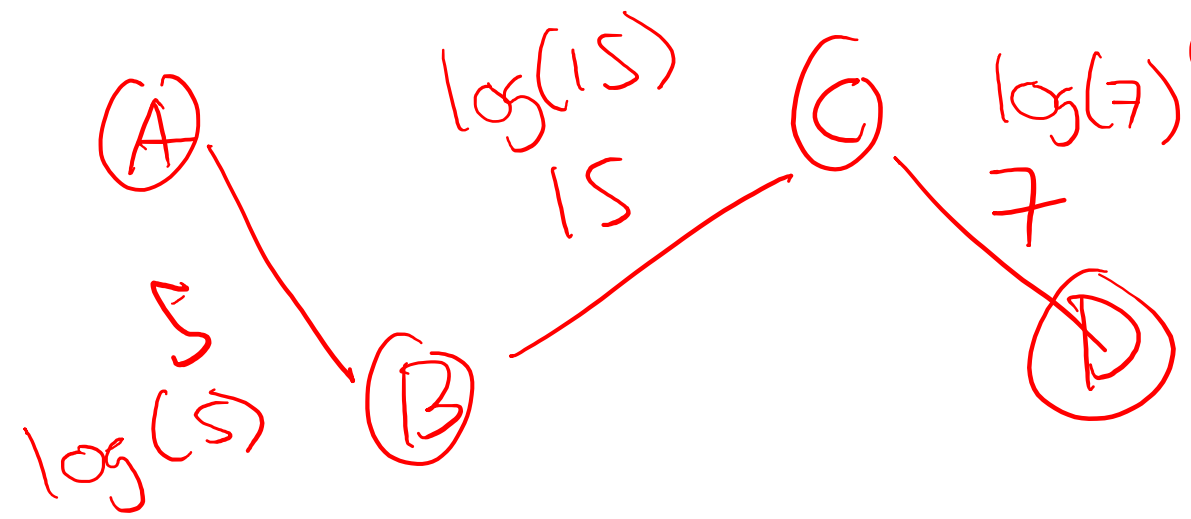


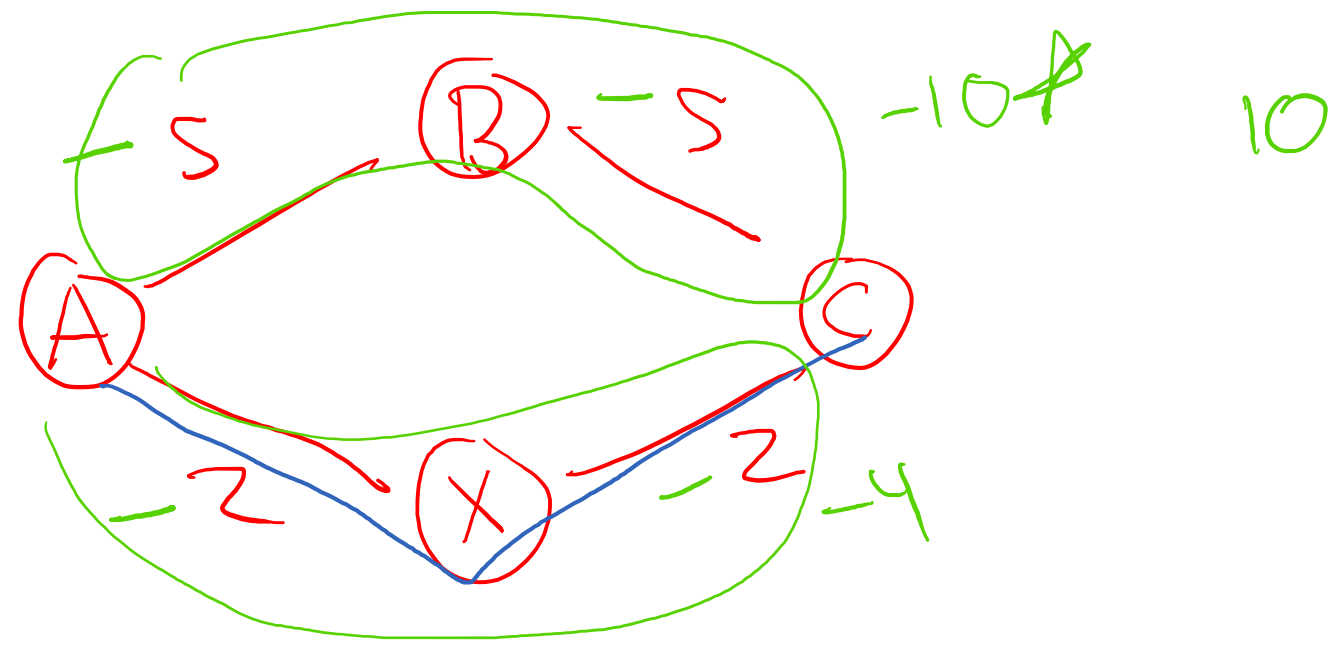
Mult

Multiplying Edges  
And  
Maximization



$$d(A, D) = 5 + 15 + 7$$
$$= \log(5) + \log(15) + \log(7)$$
$$= 2.7$$
$$10^{2.7}$$

Max



# Bellman-Ford Algorithm For Solving the Single Source Shortest Path Problem

<https://cs.pomona.edu/classes/cs140/>

SSSP  
(Dijkstra's)

# Outline

## Topics and Learning Objectives

- Discuss and analyze the Bellman-Ford Algorithm

## Exercise

- Bellman-Ford Walk-through

# Dynamic Programming

An algorithm design technique/paradigm that typically takes one of the following forms:

1. Top-Down (memoization—cache results and use recursion)
2. Bottom-Up (tabulation—store results in a table)

Used to solve problems with the following properties:

- Overlapping subproblems and
- Optimal substructure

# The Bellman-Ford Algorithm

Key Idea: leverage  
overlapping subproblems  
and optimal substructure.

A dynamic programming solution to the **Single-Source** Shortest Path problem (same problem solved by Dijkstra's)

Input:

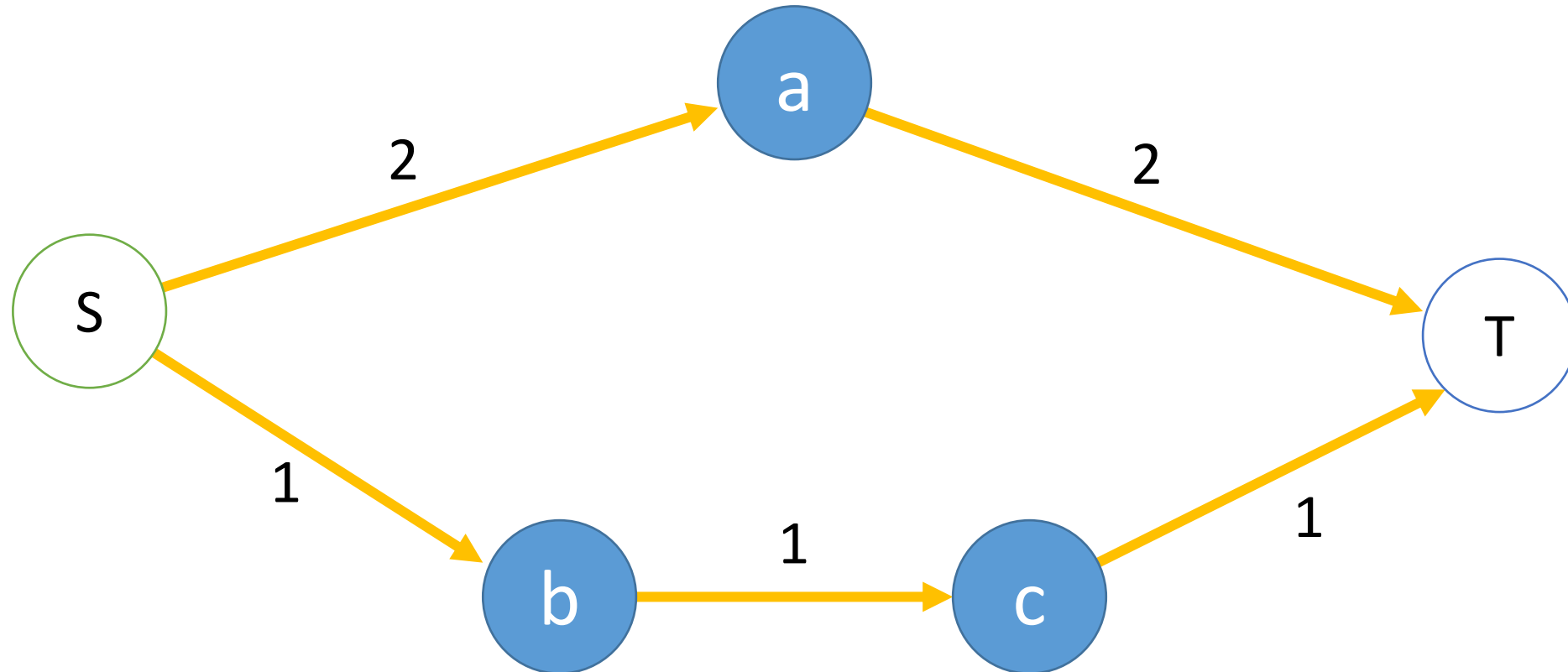
- a **weighted** graph  $G = (V, E)$  where each edge has a length  $c_e$  and
- a source vertex  $s$

Output:

- The length of the shortest path from  $s$  to all other vertices, **or**
- We output that we detected a **negative cycle** (invalid path lengths)

# Example 1

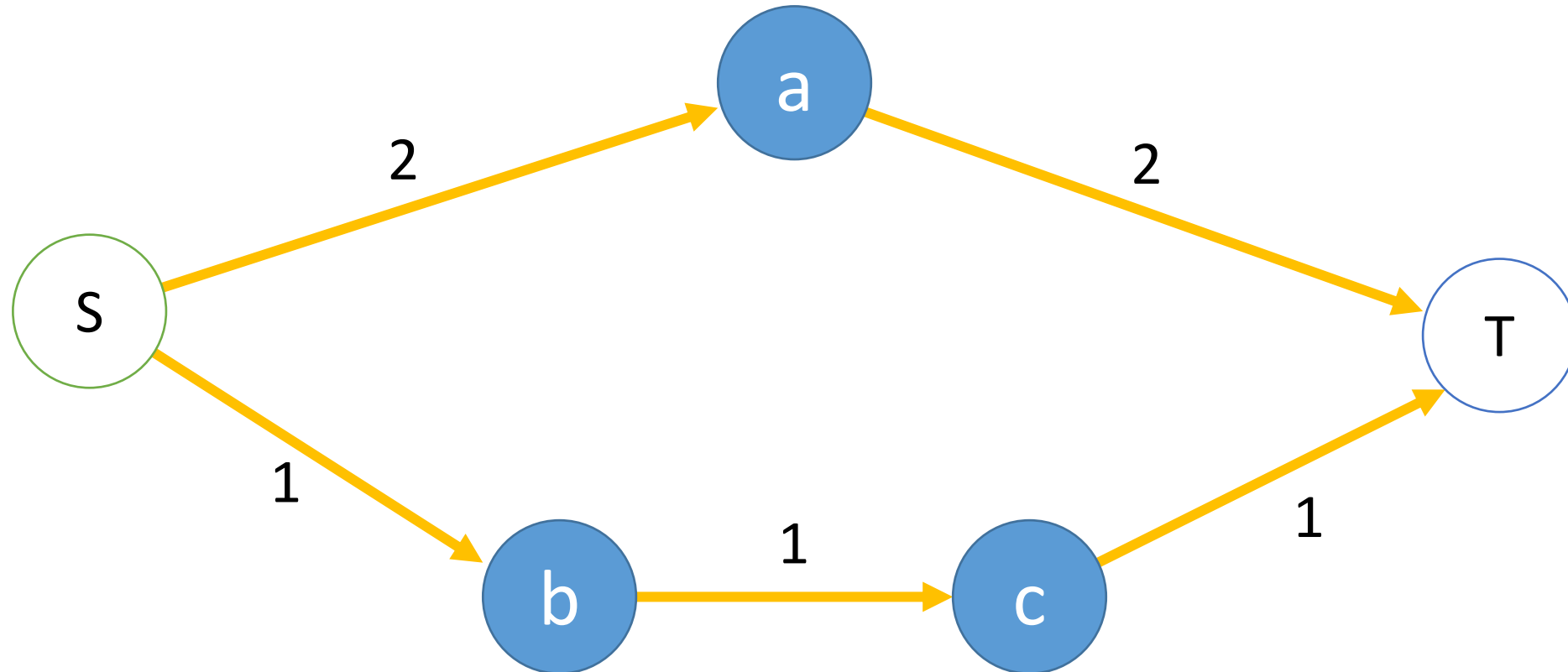
What is the shortest path from S to T using 0 edges?



Subproblem: consider only a subset of the possible paths.

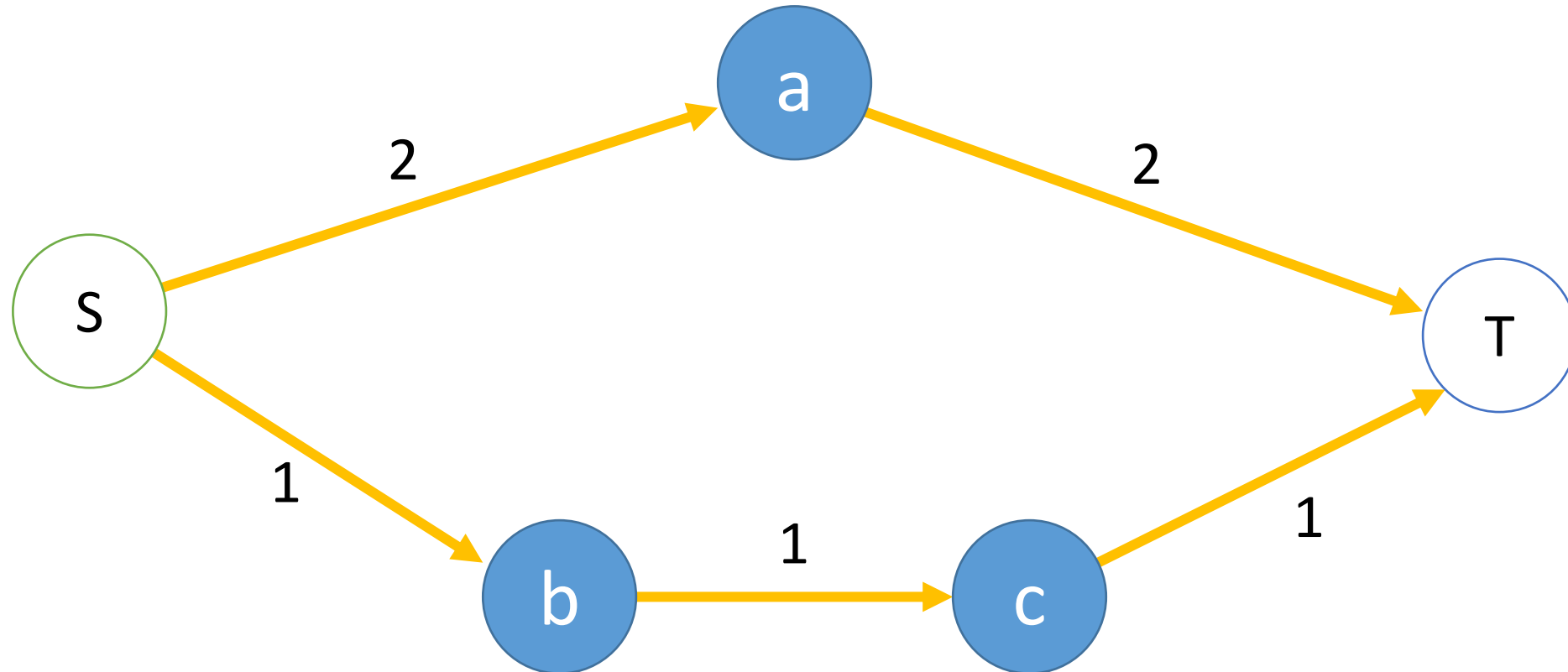
# Example 1

What is the shortest path from S to T using **1 edge**?



# Example 1

What is the shortest path using 2 edges?

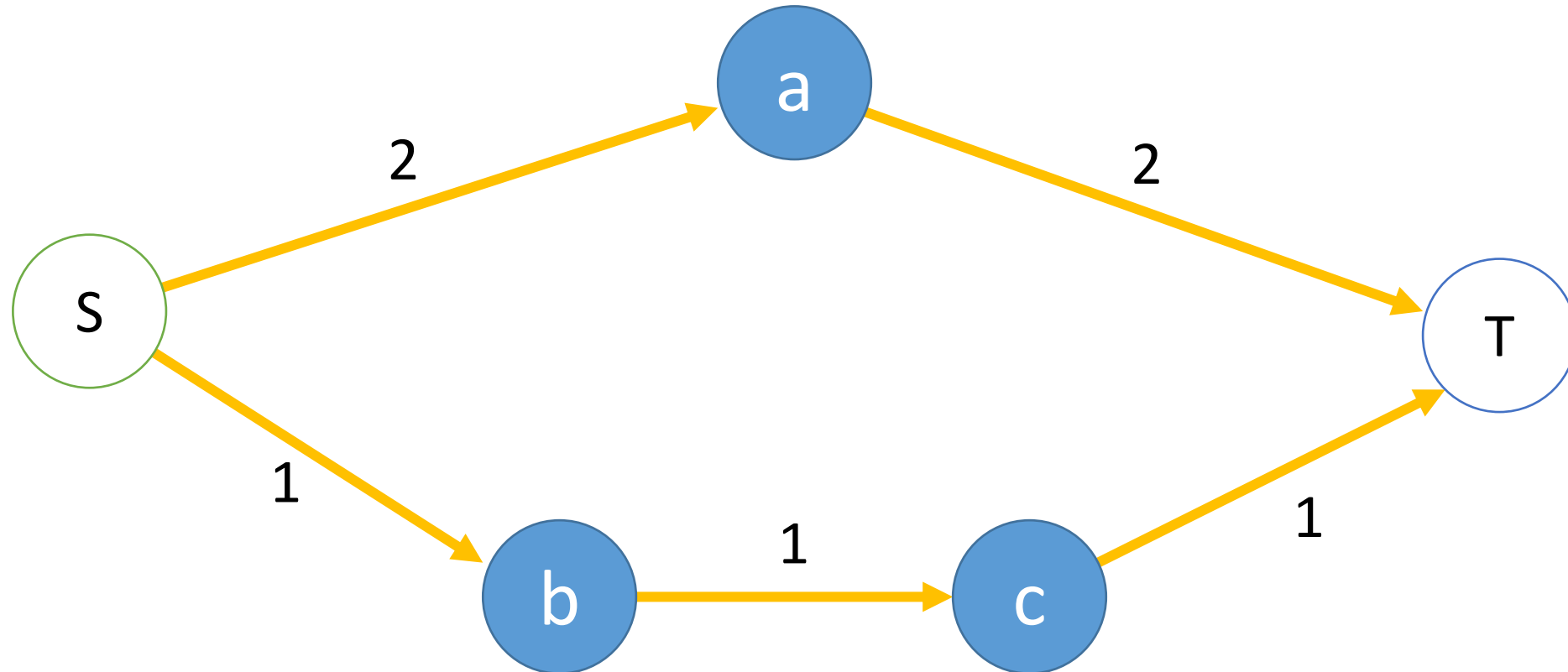




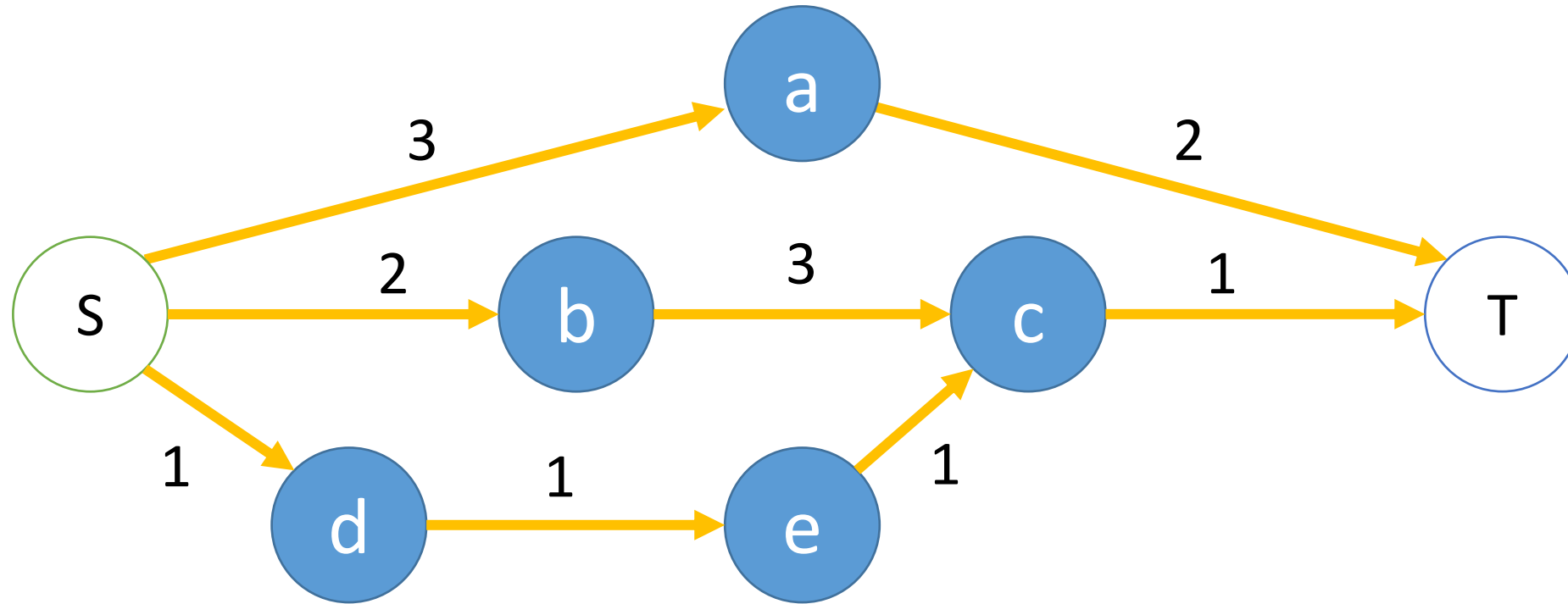
# Example 1

What is the shortest path using 3 edges?

What is the shortest path using 2 edges?

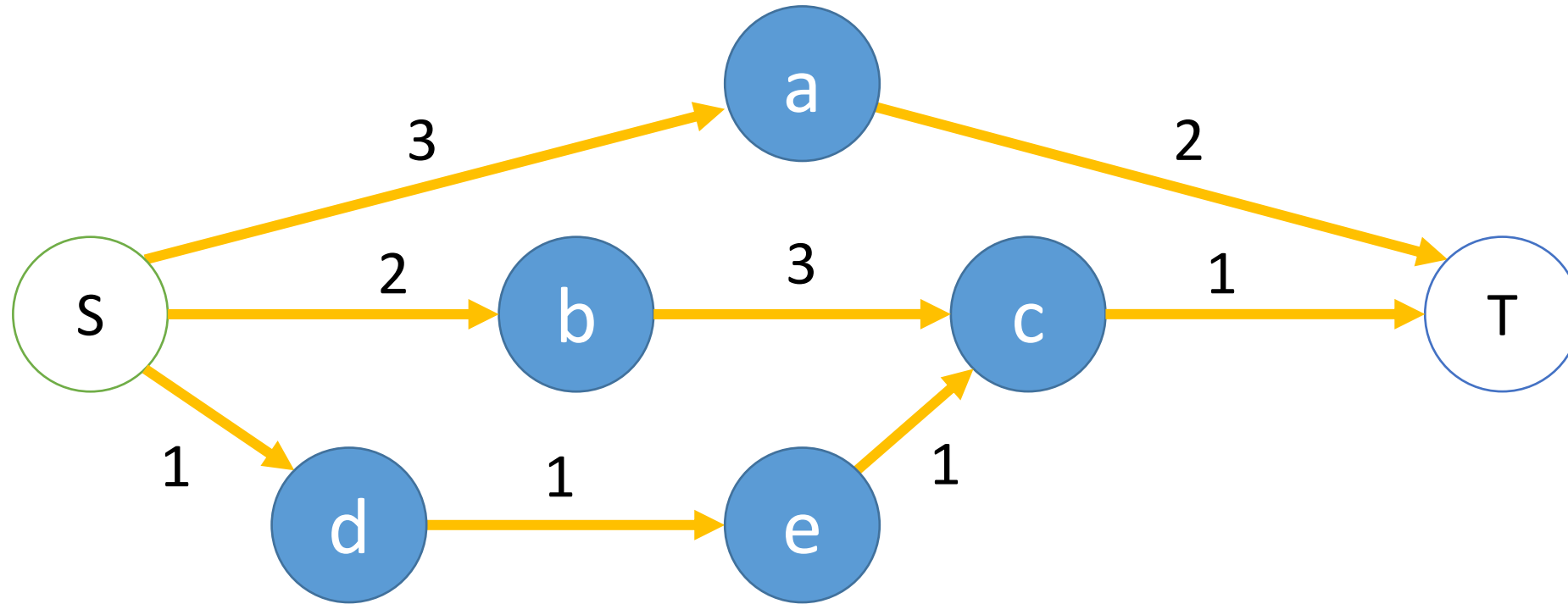


# Example 2



What is the shortest path with at most 1 edge?

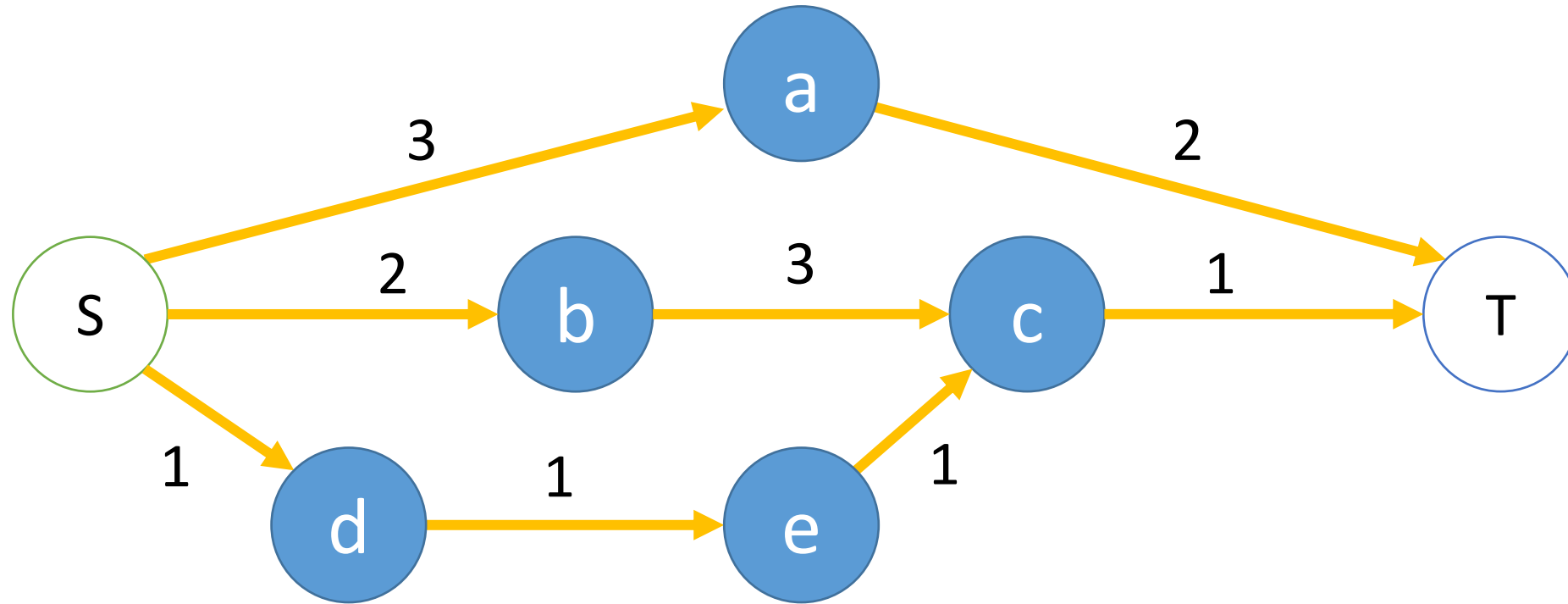
# Example 2



Shortest path with  
at most 2 edges

# Example 2

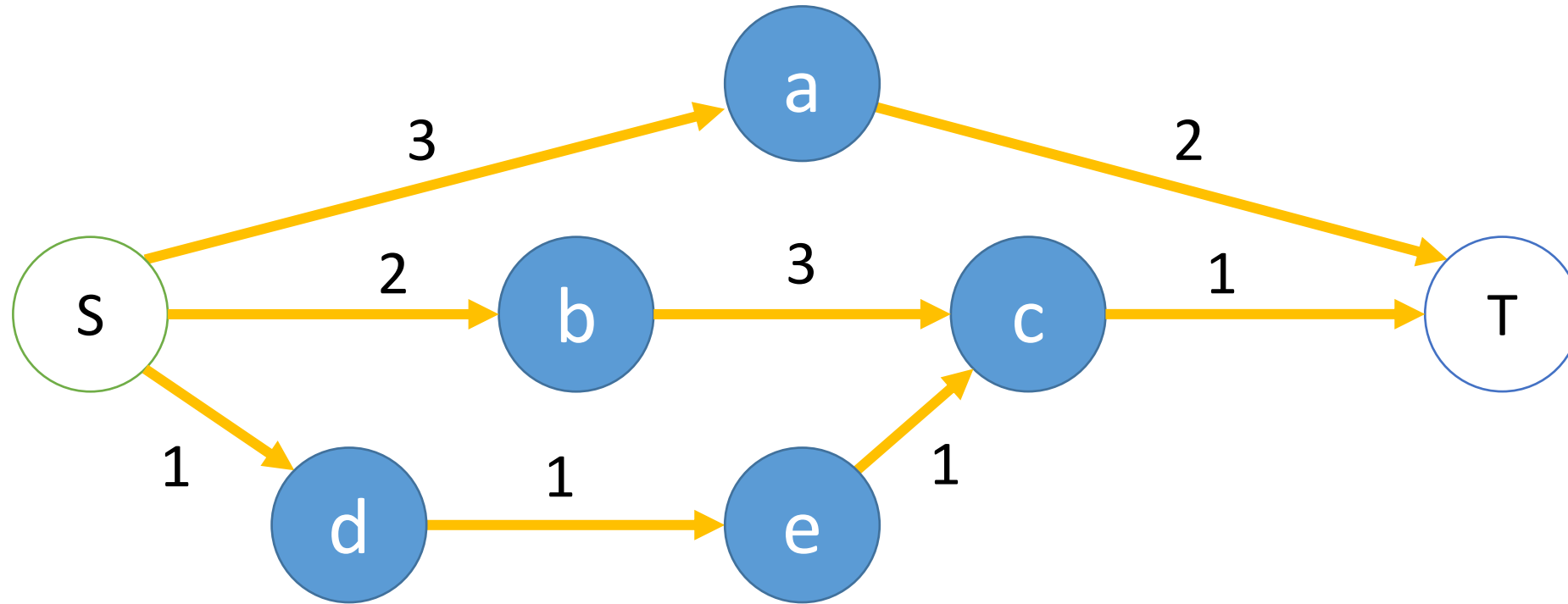
We didn't gain anything by adding the edge



Shortest path with at most 2 edges

Shortest path with at most 3 edges

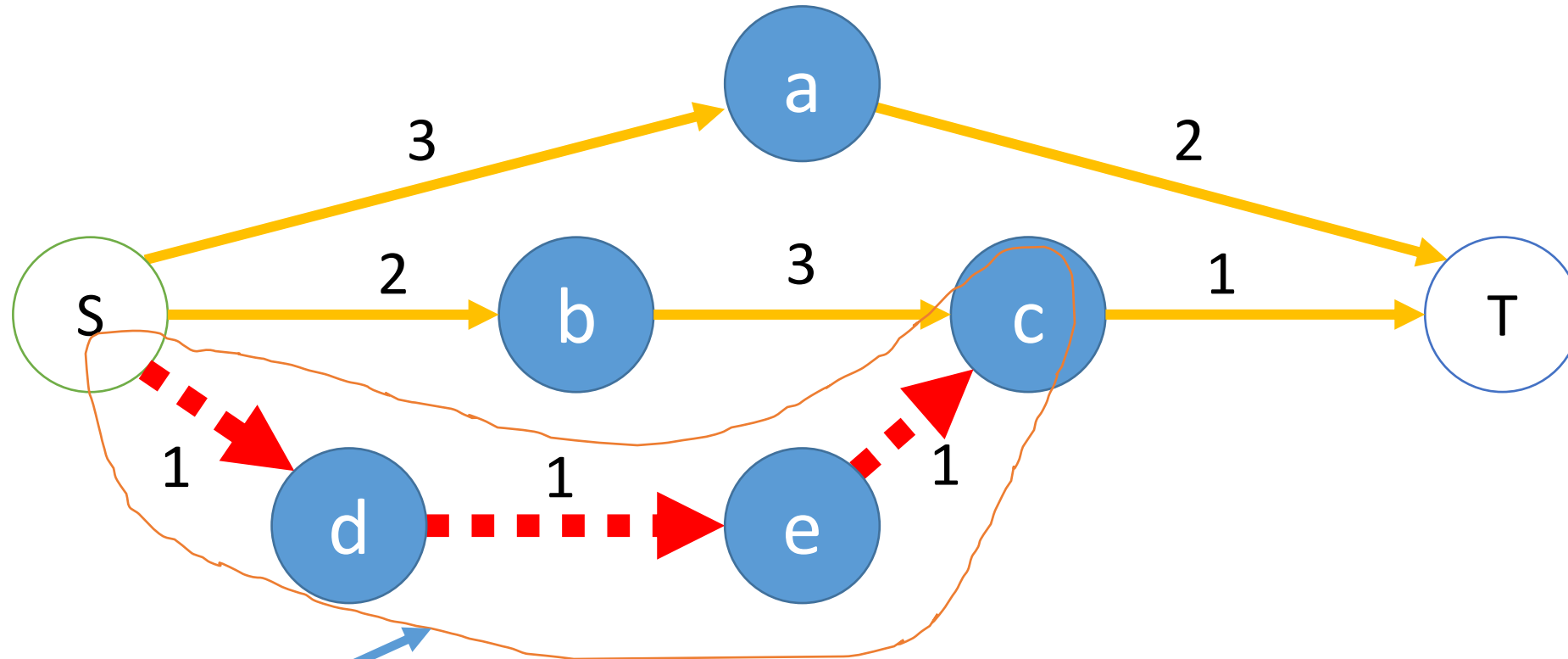
# Example 2



Shortest path with at most 4 edges

# Example 2

If rainbow is the shortest path from S to T using at most 4 edges, then the red dashed line must be the shortest path from S to C using at most 3 edges.



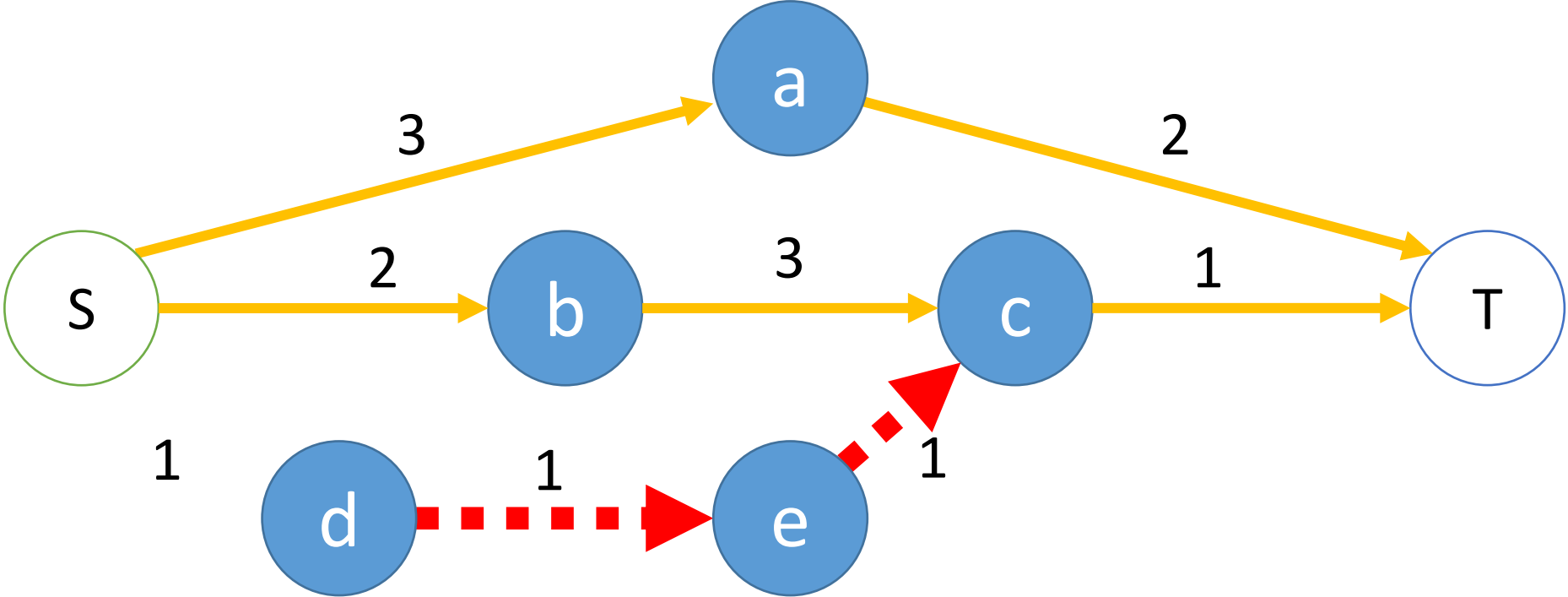
**Optimal Substructure**

This must be shortest path from S to C with at most 3 edges!

Shortest path with at most 4 edges

# Example 2

The path from D to C is used as part of the shortest path from S to T. And as part of the shortest path from S to C.



**Overlapping Subproblems**

The path from D to C is used as part of the shortest path from S to T and from D to T (and ...)

Shortest path with at most 4 edges

```
FUNCTION BellmanFord(G, start_vertex)
```

```
  n = G.vertices.length
```

```
  edges_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
```

```
  edges_lengths[0, start_vertex] = 0
```

```
  FOR num_edges IN [1 ..< n] Why won't we need more than n-1 edges?
```

```
    FOR v IN G.vertices
```

```
      min_len = INFINITY
```

```
      FOR (vFrom, v) IN G.edges Cost to get to vFrom using at most i-1 edges
```

```
        len = edges_lengths[num_edges - 1, vFrom] + G.edges[vFrom, v].cost
```

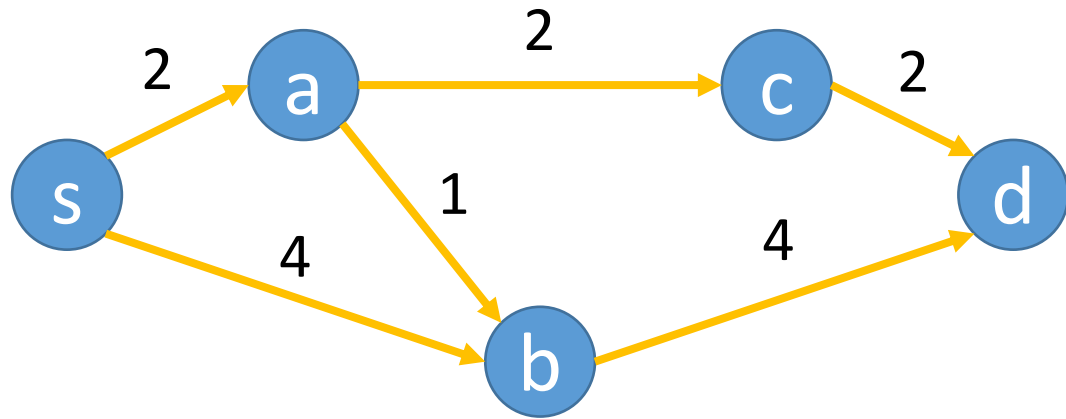
```
        IF len < min_len
```

```
          min_len = len Cost using at most i-1 edges
```

```
          edges_lengths[num_edges, v] = min(edges_lengths[num_edges - 1, v],
```

```
            min_len) Cost using at most i edges
```





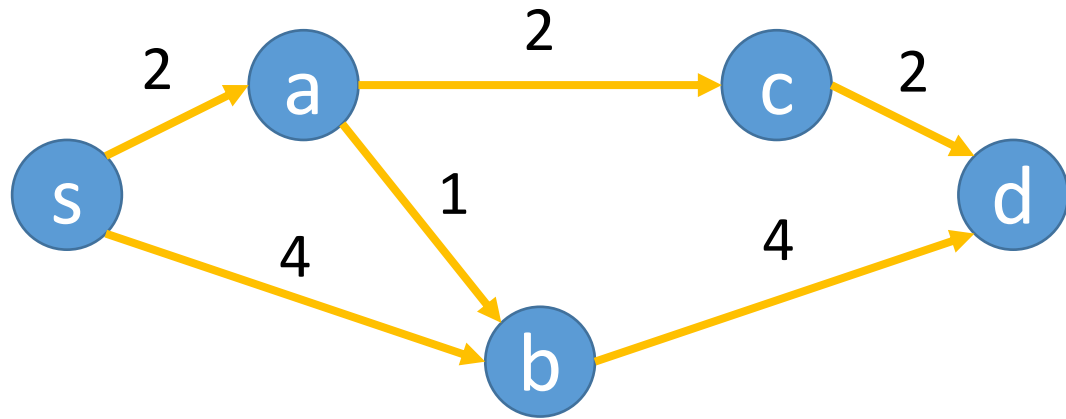
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

Max Number  
Of Edges  
On Path

i	4					
	3					
	2					
	1					
	0					
		s	a	b	c	d
		v				

End Vertex

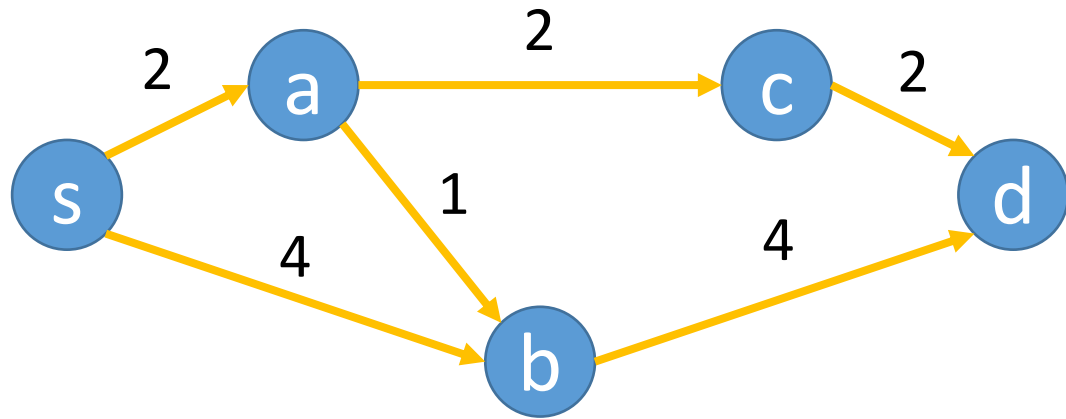


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

What does a single cell denote?

i	4					
	3					
	2					
	1					
	0					
		s	a	b	c	d
		v				



```

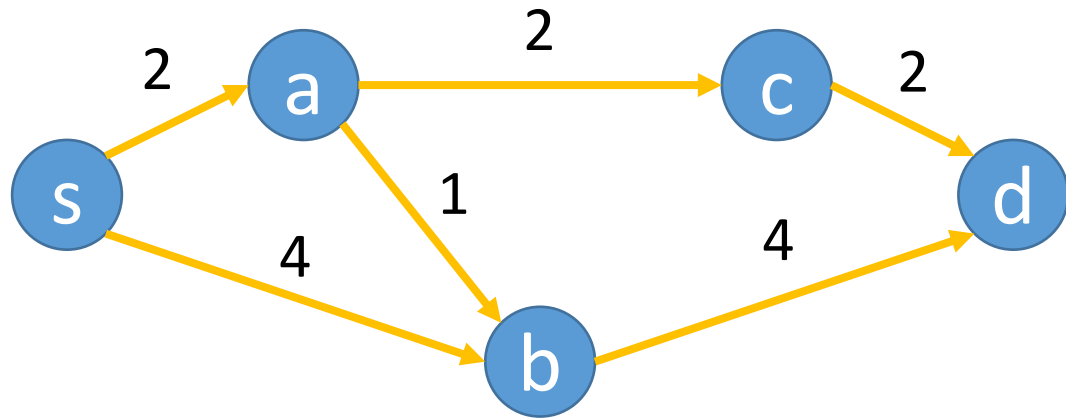
FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

```

edges_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
edges_lengths[0, start_vertex] = 0
  
```

Initialize first row  
Lengths of paths from s to  
all other vertices using zero  
edges

i	3					
	2					
	1					
	0					
		s	a	b	c	d
		v				



```

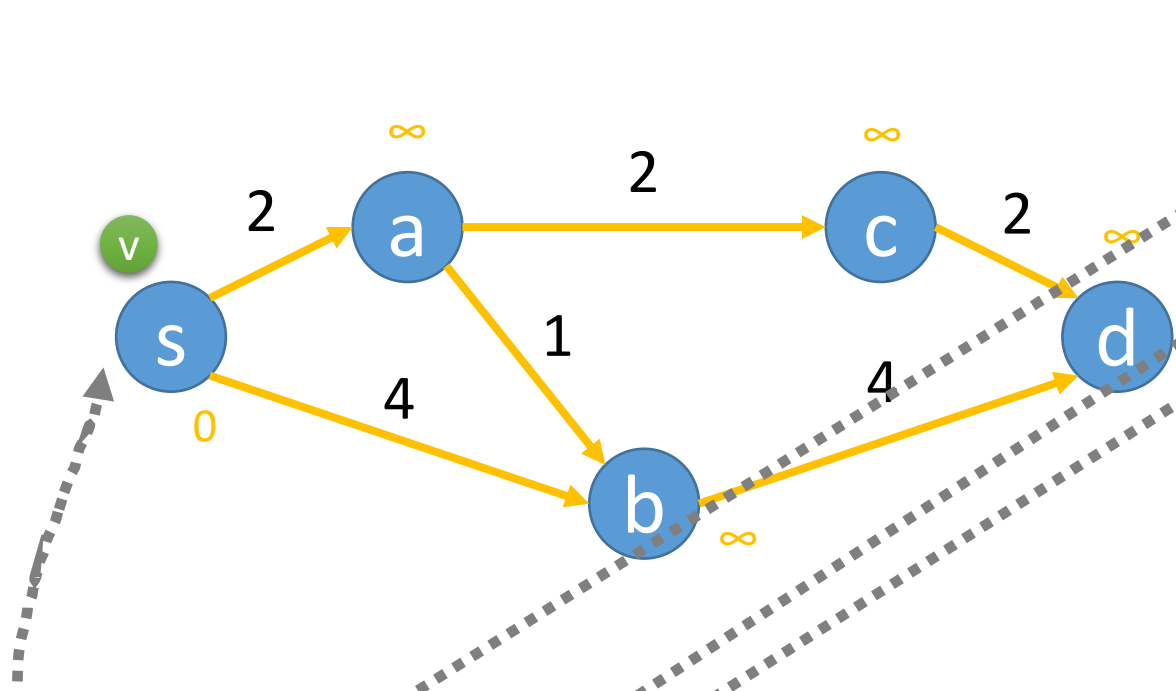
FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

```

edges_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
edges_lengths[0, start_vertex] = 0
  
```

Initialize first row  
Lengths of paths from s to  
all other vertices using zero  
edges

i	3					
	2					
	1					
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



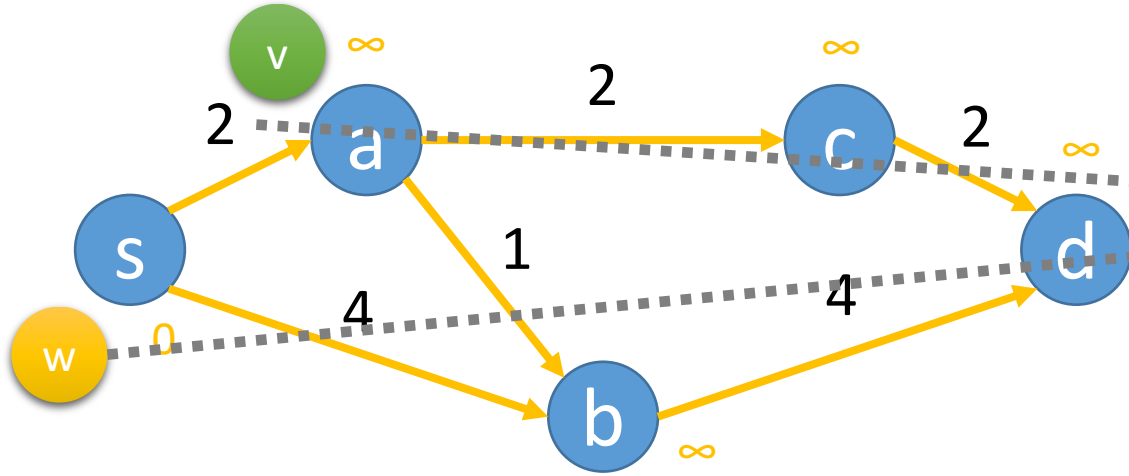
```

FOR num_edges IN [1 ..< n]
FOR v IN G.vertices
min_len = INFINITY
FOR (vFrom, v) IN G.edges
len = lens[num_edges - 1, vFrom] + c
IF len < min_len
min_len = len
lens[num_edges, v] = min(
lens[num_edges - 1, v], min_len)

```

num\_edges = 1  
v = s  
minW = inf  
Nothing to loop over

	4					
	3					
	2					
i	1					
	0	0	∞	∞	∞	
		s	a	b	c	d
		v				

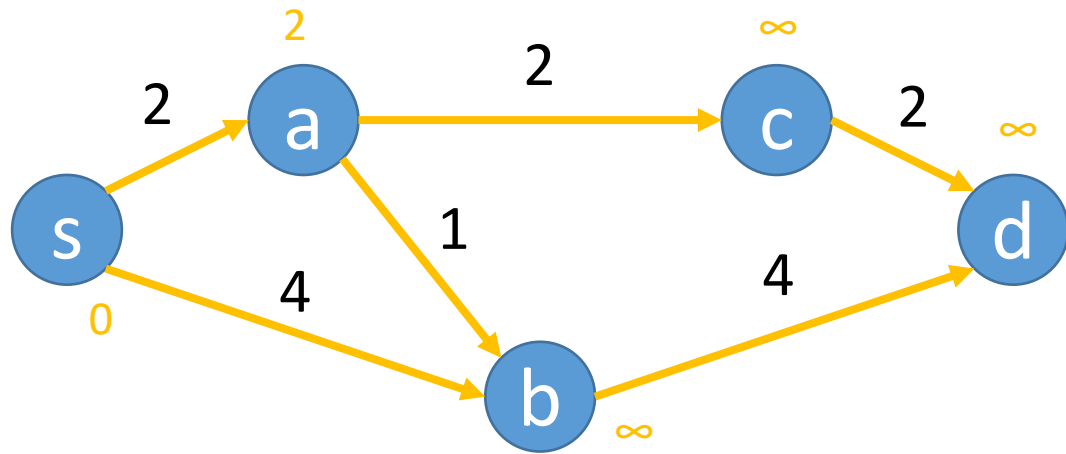


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom]
      IF len < min_len
        min_len = len
      lens[num_edges, v] = min(
        lens[num_edges - 1, v], min_len)
  
```

num\_edges = 1  
 v = a  
 minW = inf  
 minW = 2

i	4					
	3					
	2					
	1	0				
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				

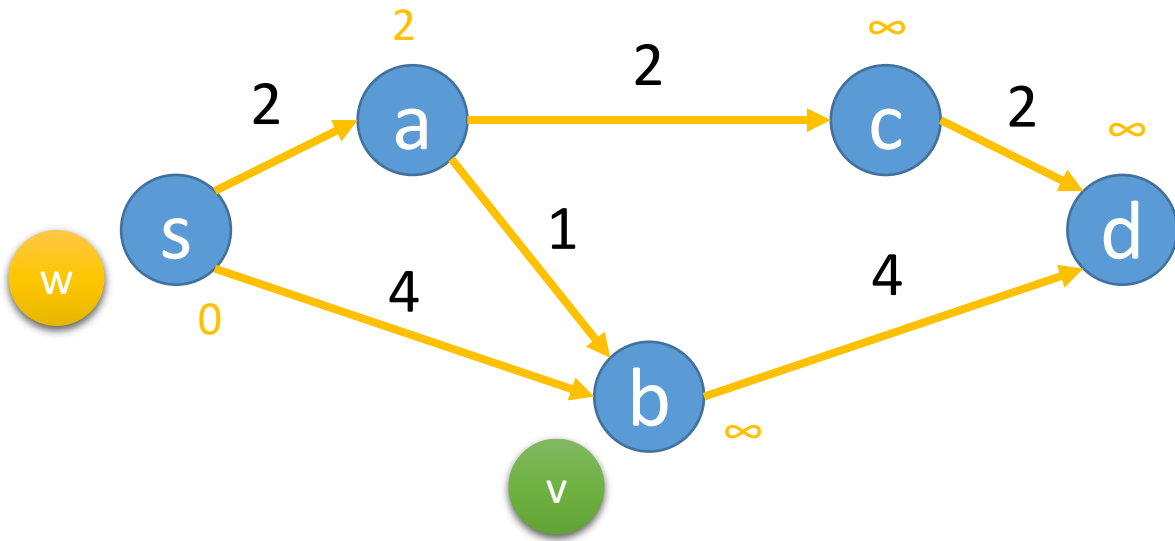


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

num\_edges = 1  
 v = a  
 minW = inf  
 minW = 2

i	4					
	3					
	2					
	1	0	2			
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



```

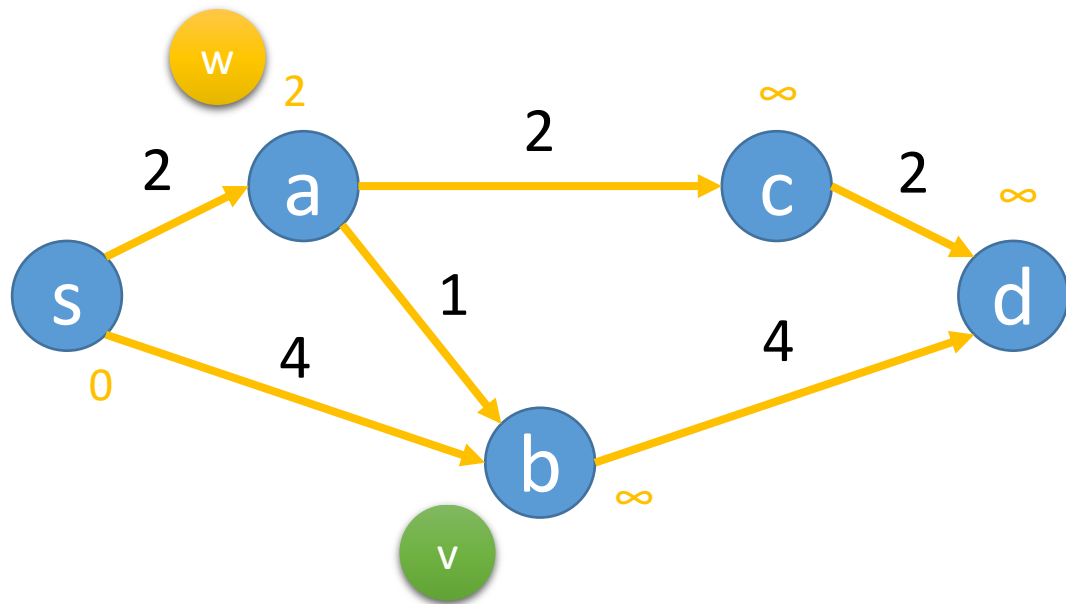
FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 1  
 v = b  
 minW = inf  
 minW = 4

i	4					
	3					
	2					
	1	0	2			
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				





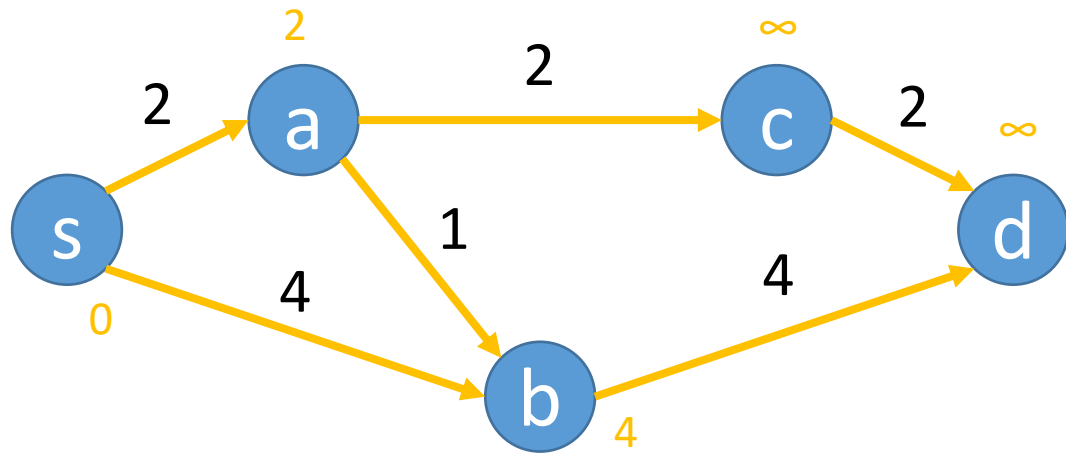
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 1  
 v = b  
 minW = inf  
 minW = 4

i	4					
	3					
	2					
	1	0	2			
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				

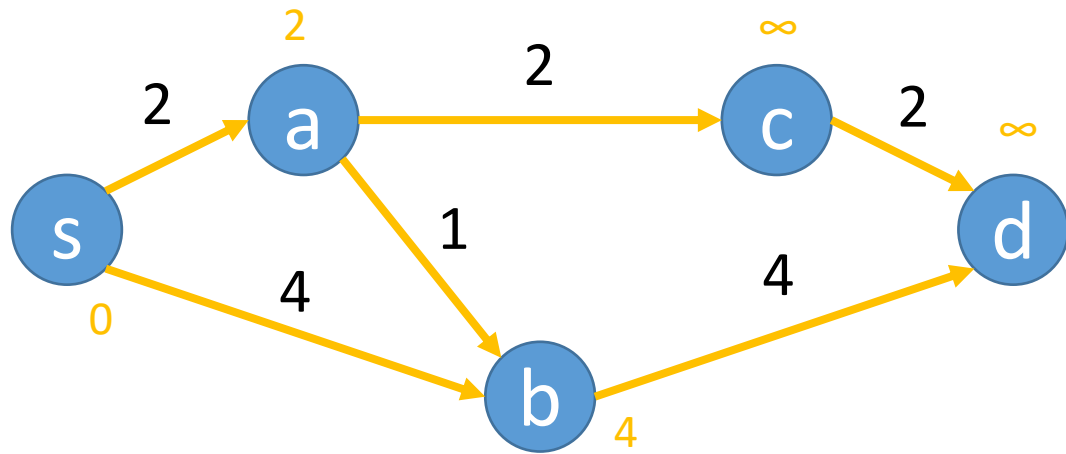


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

num\_edges = 1  
 v = b  
 minW = inf  
 minW = 4

i	4					
	3					
	2					
	1	0	2	4		
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



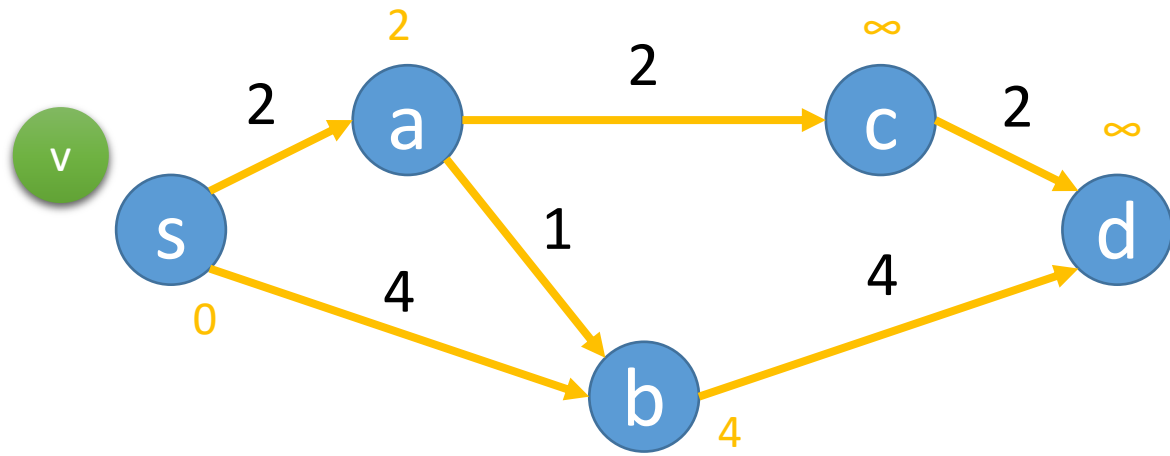
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

There are not any paths of length 1 from s to c or d

i	4					
	3					
	2					
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				

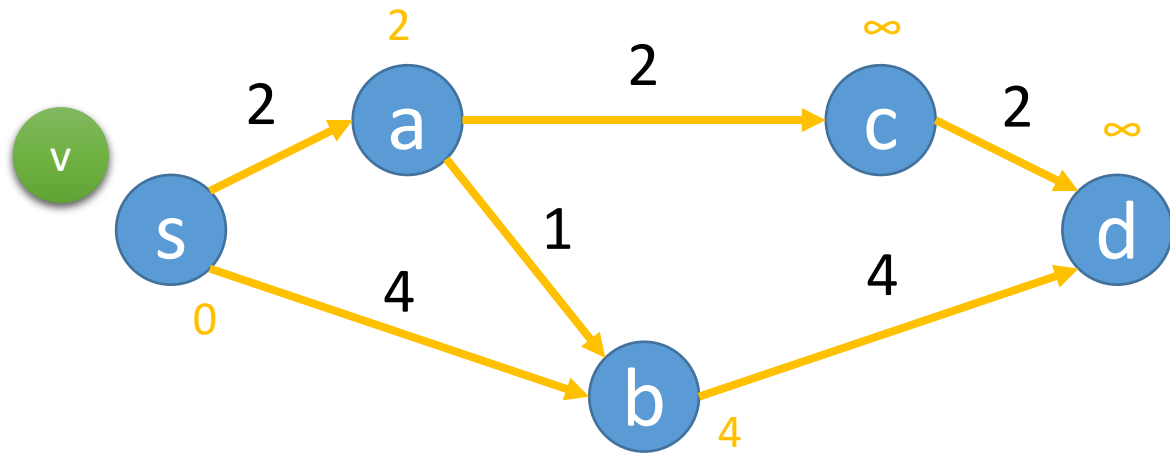


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

num\_edges = 2  
 v = s  
 minW = inf

i	4					
	3					
	2					
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



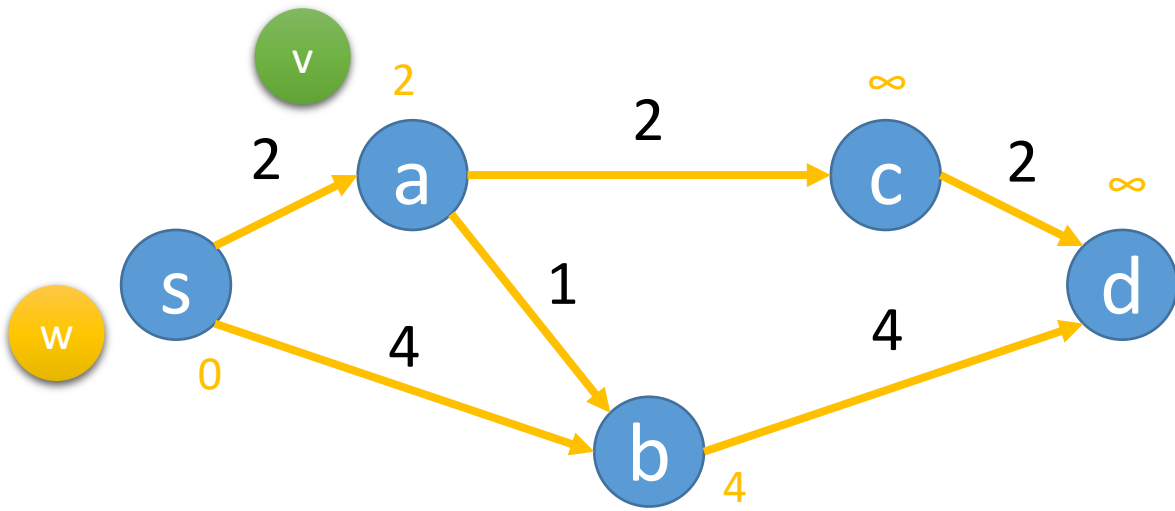
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 2  
 v = s  
 minW = inf

i	4					
	3					
	2	0				
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



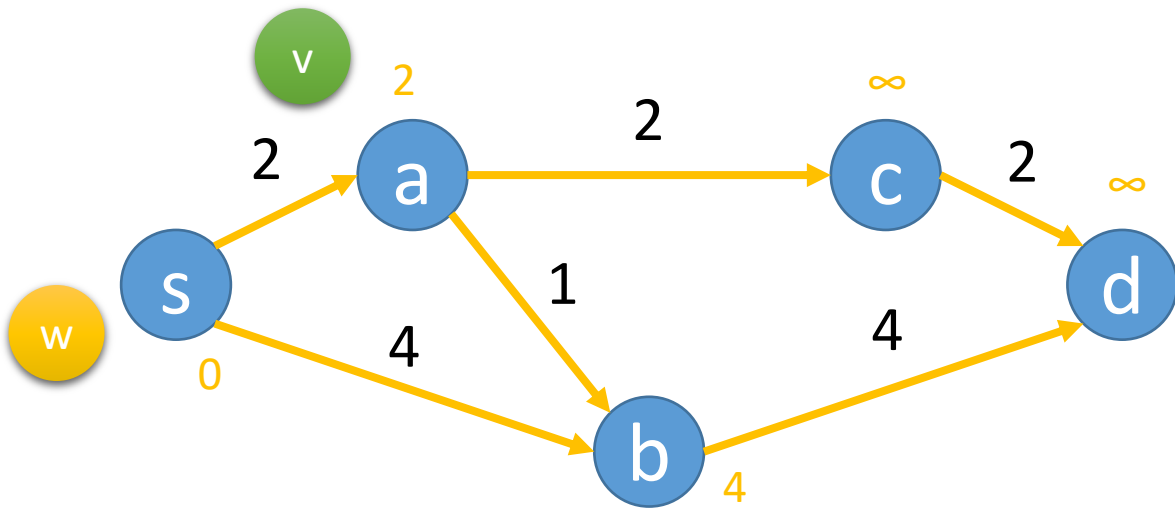
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 2  
 v = a  
 minW = inf  
 minW = 2

i	4					
	3					
	2	0				
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



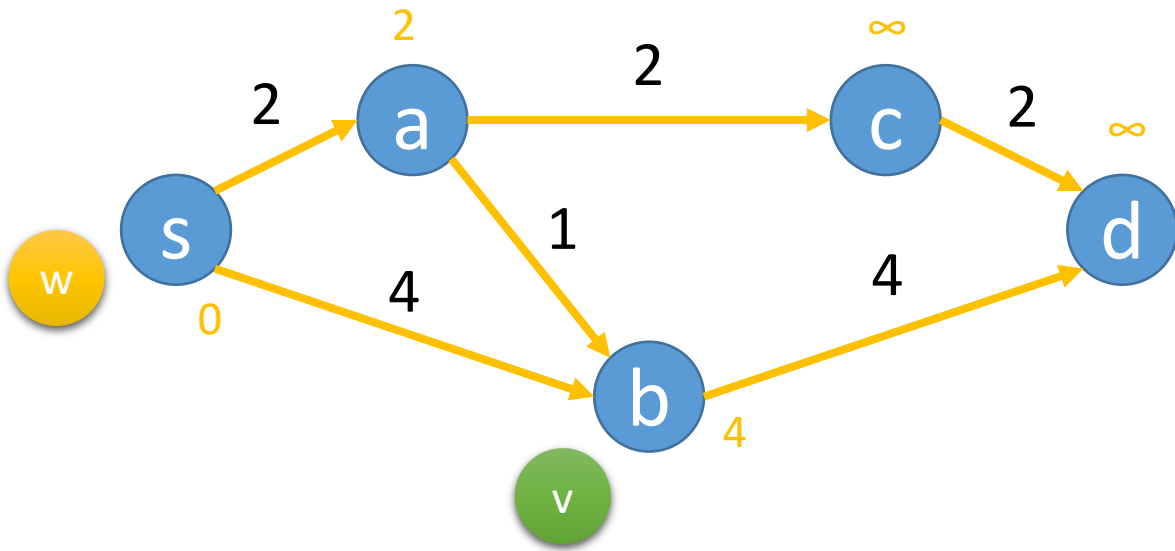
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 2  
 v = a  
 minW = inf  
 minW = 2

i	4					
	3					
	2	0	2			
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



```

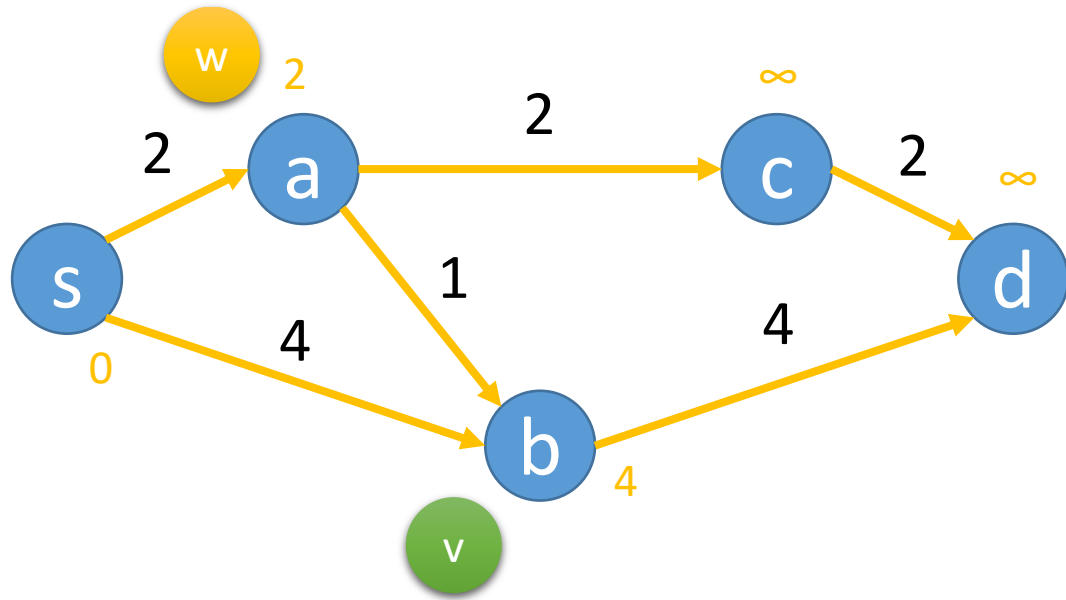
FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 2  
 v = b  
 minW = inf  
 minW = 4

i	4					
	3					
	2	0	2			
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



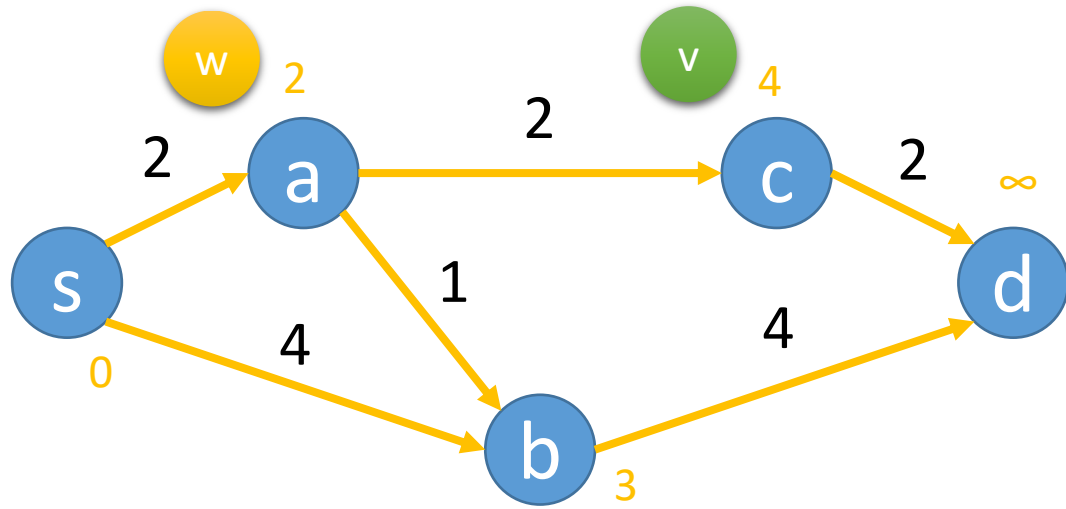


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

num\_edges = 2  
 v = b  
 minW = inf  
 minW = 4  
 minW = 3

i	4					
	3					
	2	0	2			
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



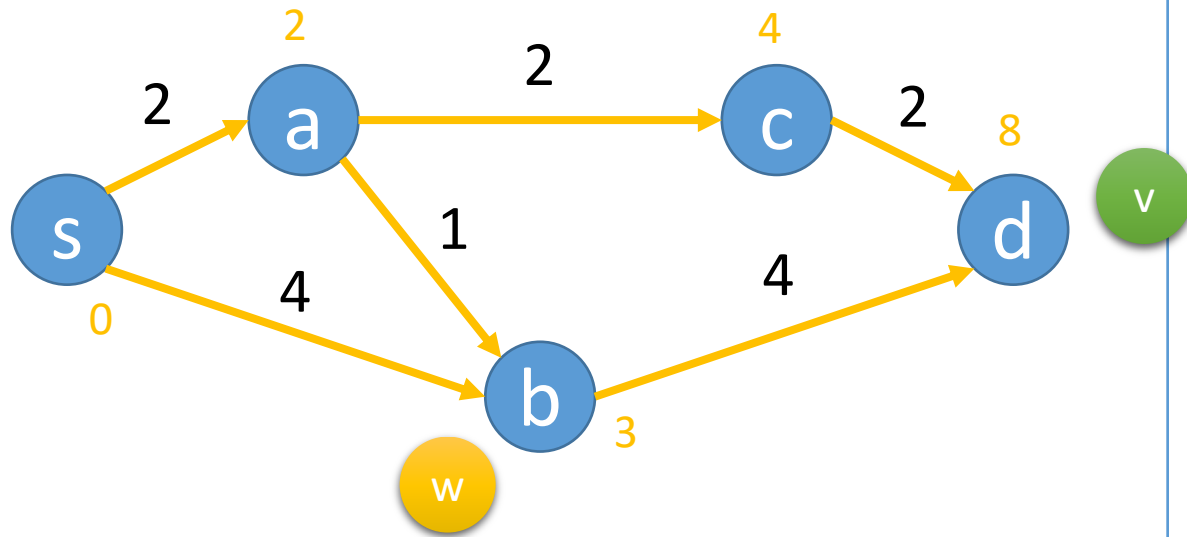
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 2  
 v = c  
 minW = inf  
 minW = 4

i	4					
	3					
	2	0	2	3	4	
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				



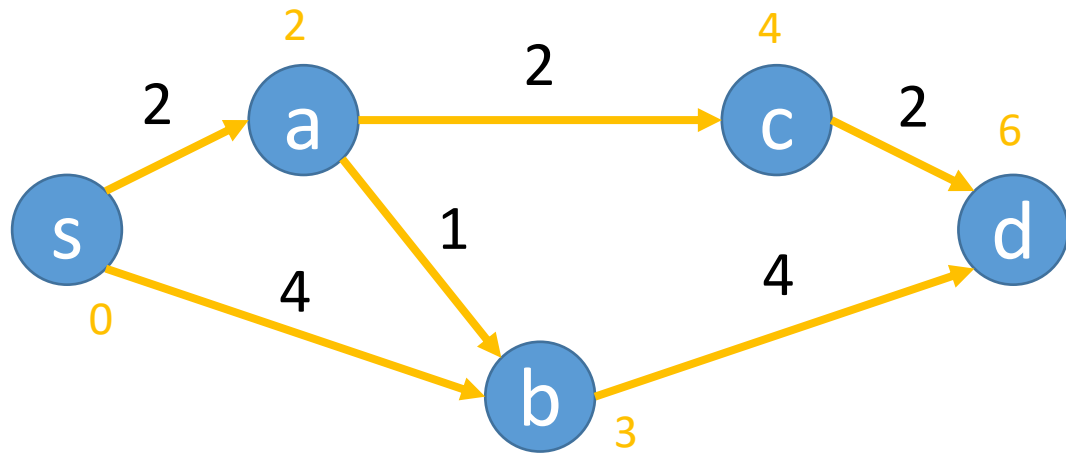
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)

```

num\_edges = 2  
 v = d  
 minW = inf  
 minW = 8

i	4					
	3					
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				

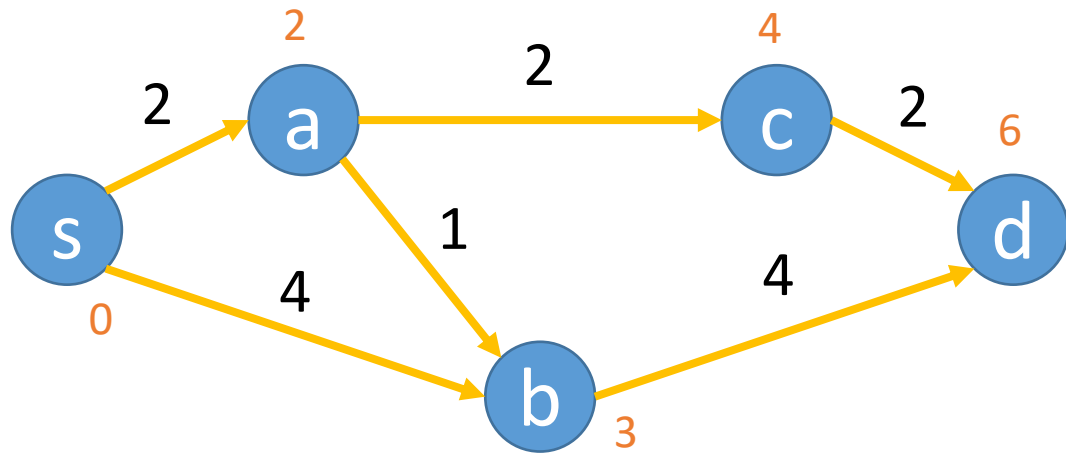


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

What is our output?

i	4	0	2	3	4	6
	3	0	2	3	4	6
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				

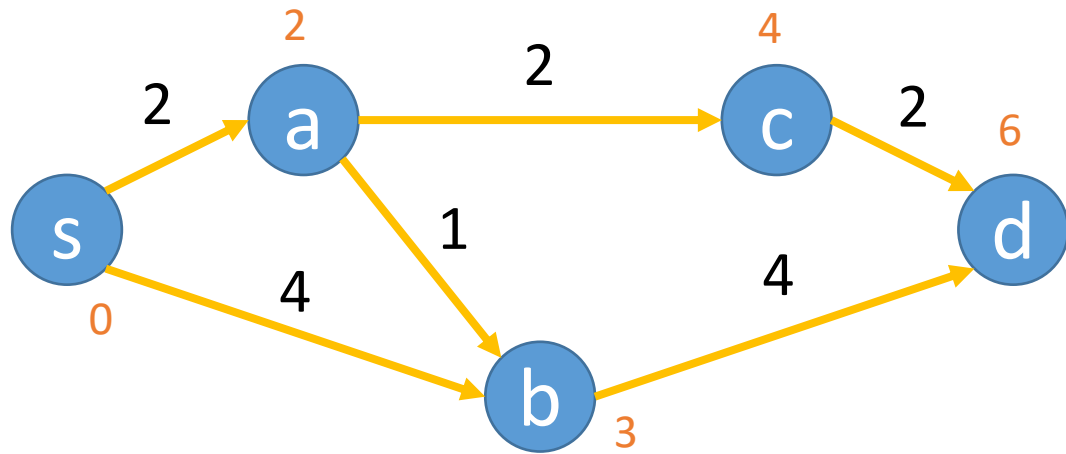


```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

What is our output?

	4	0	2	3	4	6
i	3	0	2	3	4	6
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
	v					



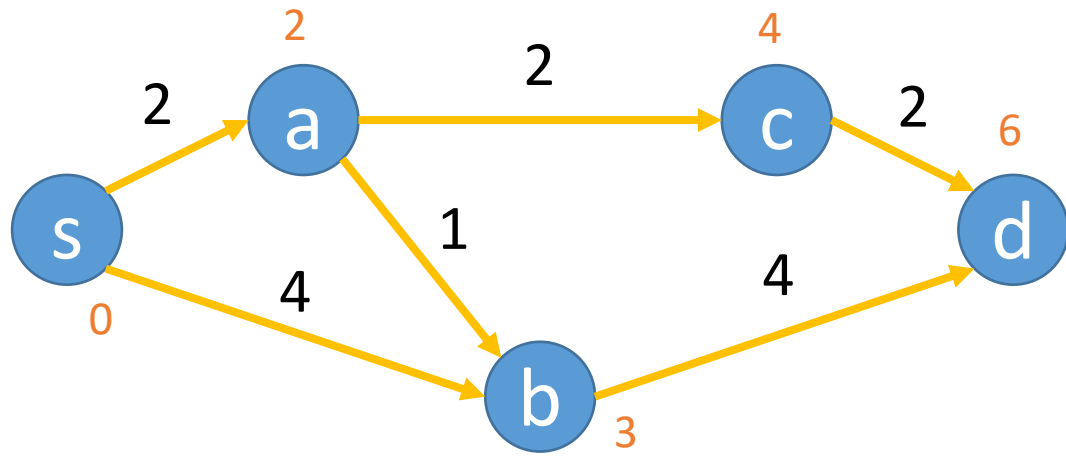
```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

What is our output?

Do we need the other rows of the table?

	4	0	2	3	4	6
i	3	0	2	3	4	6
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
	v					



```

FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

What is our output?

Do we need the other rows of the table?

i	4	0	2	3	4	6
	3	0	2	3	4	6
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				

# Running Time of Bellman-Ford Algorithm?

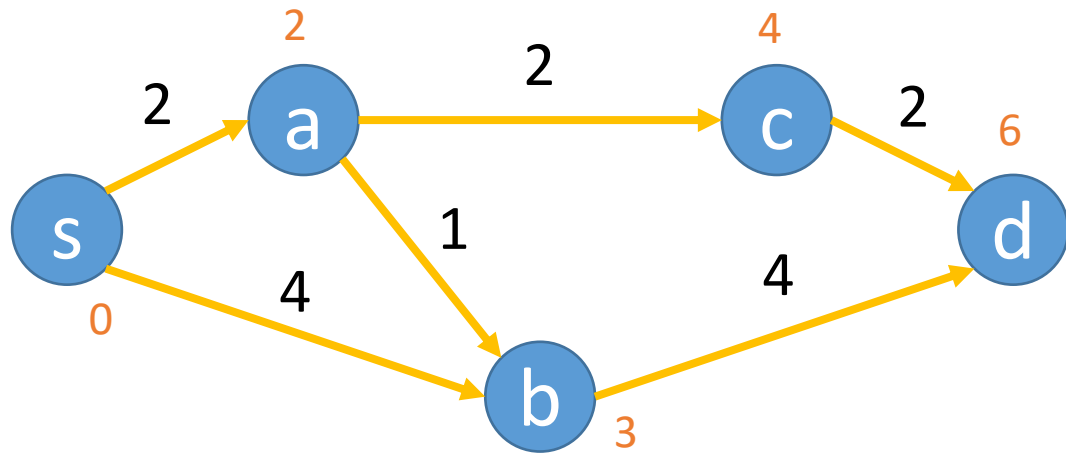
```
FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(lens[num_edges - 1, v], min_len)
```

The inner two loops go through every edge once, ordered by the vertices

---

$O(n^2)$     $O(mn)$     $O(n^3)$     $O(m^2)$



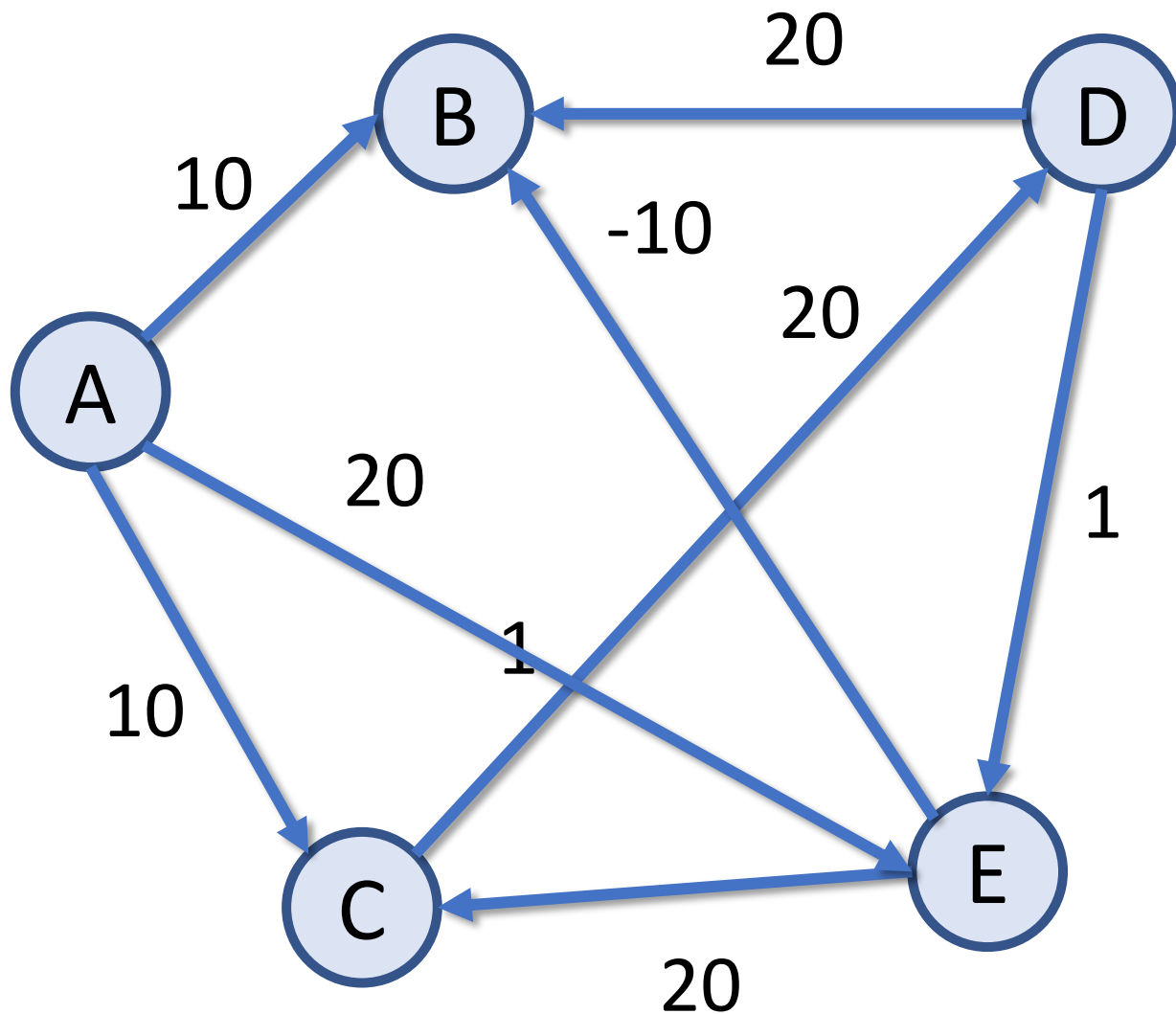


```

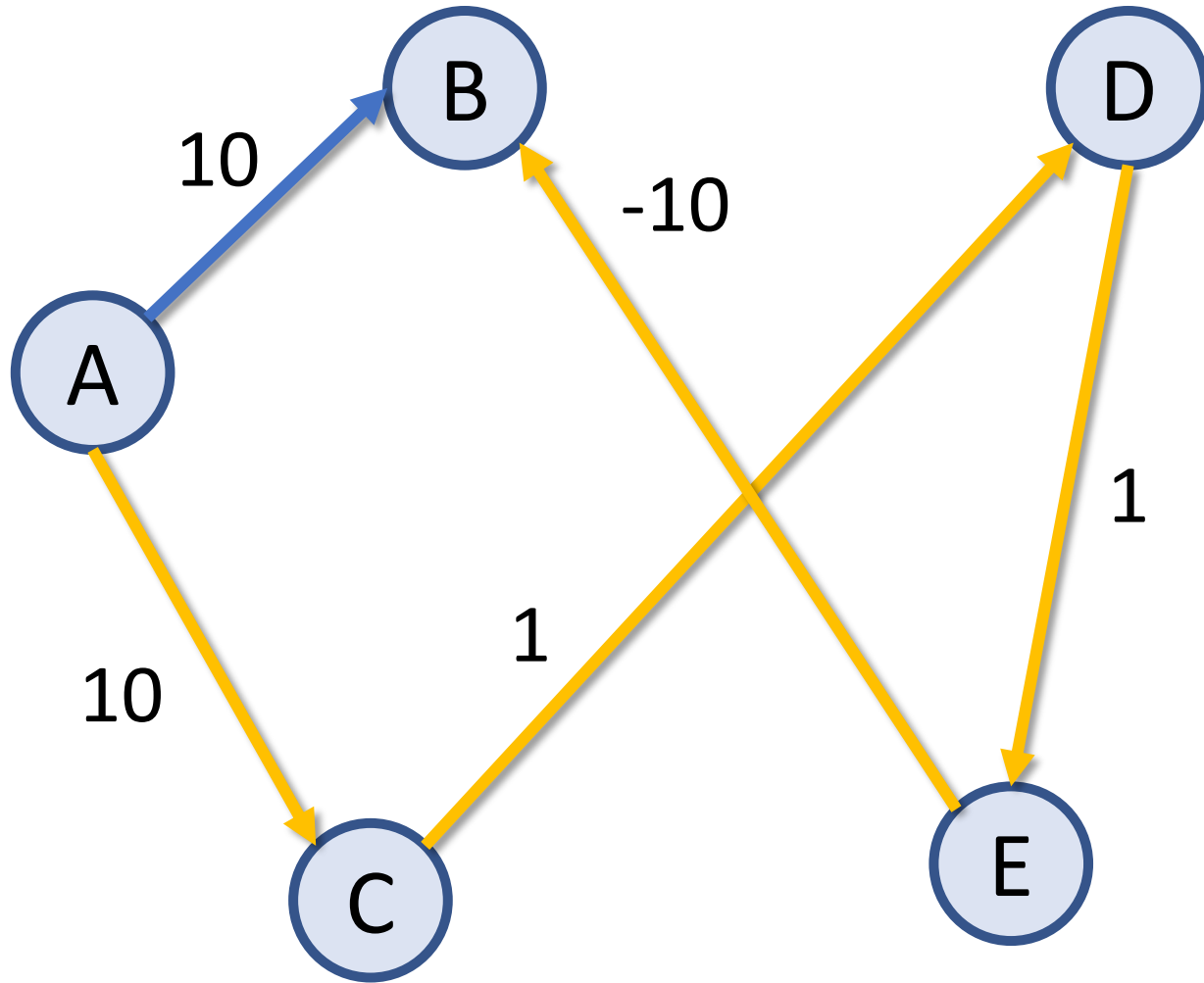
FOR num_edges IN [1 ..< n]
  FOR v IN G.vertices
    min_len = INFINITY
    FOR (vFrom, v) IN G.edges
      len = lens[num_edges - 1, vFrom] + c
      IF len < min_len
        min_len = len
    lens[num_edges, v] = min(
      lens[num_edges - 1, v], min_len)
  
```

What about negative edges?

	4	0	2	3	4	6
i	3	0	2	3	4	6
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
	v					



What is the maximum number of edges on any real (not negative infinity) **shortest** path?

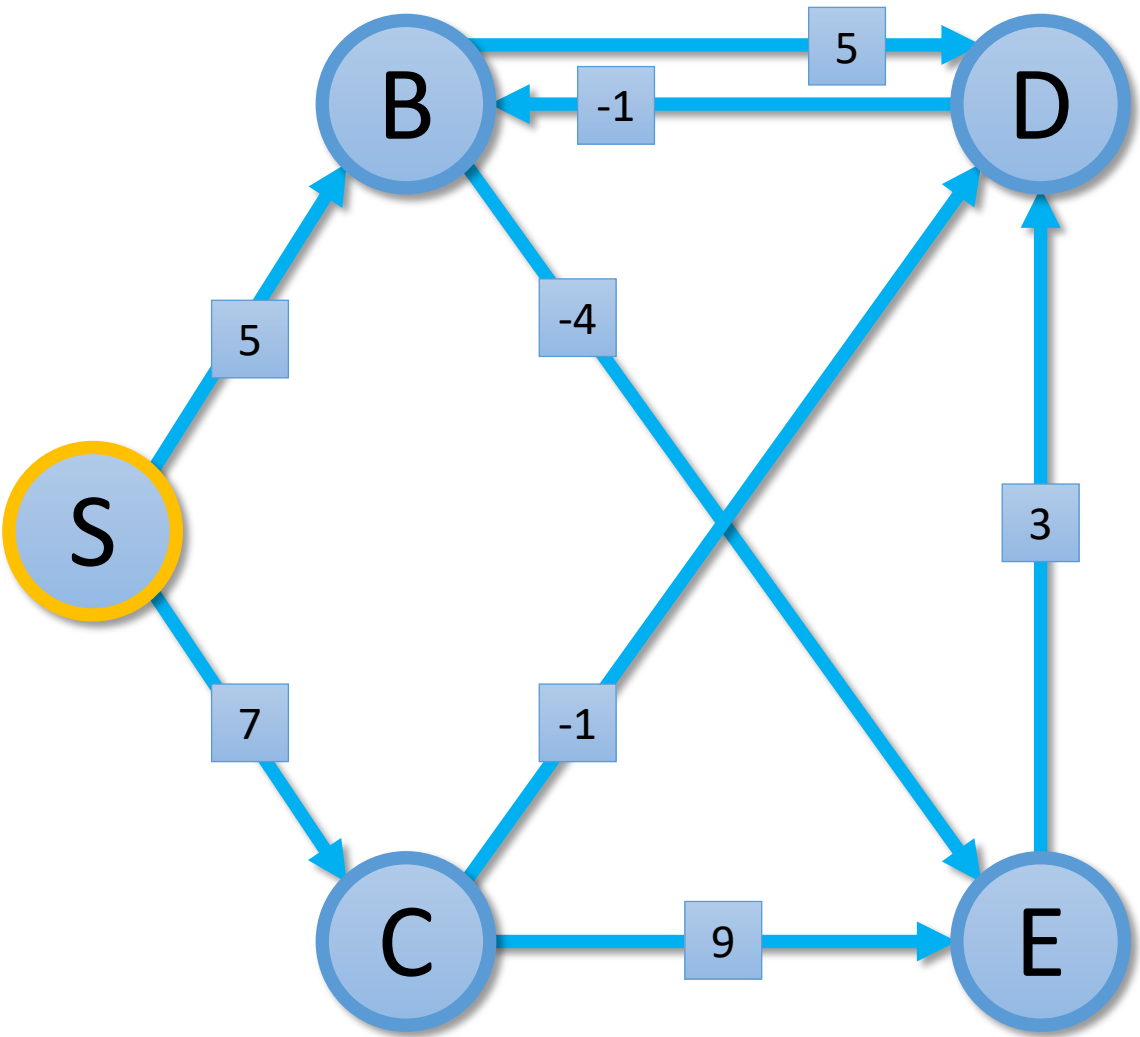


What is the maximum number of edges on any real (not negative infinity) **shortest** path?

Any additional edges will increase the path length, or otherwise must be part of a negative cycle

Exercise

What is the shortest path from S to B?

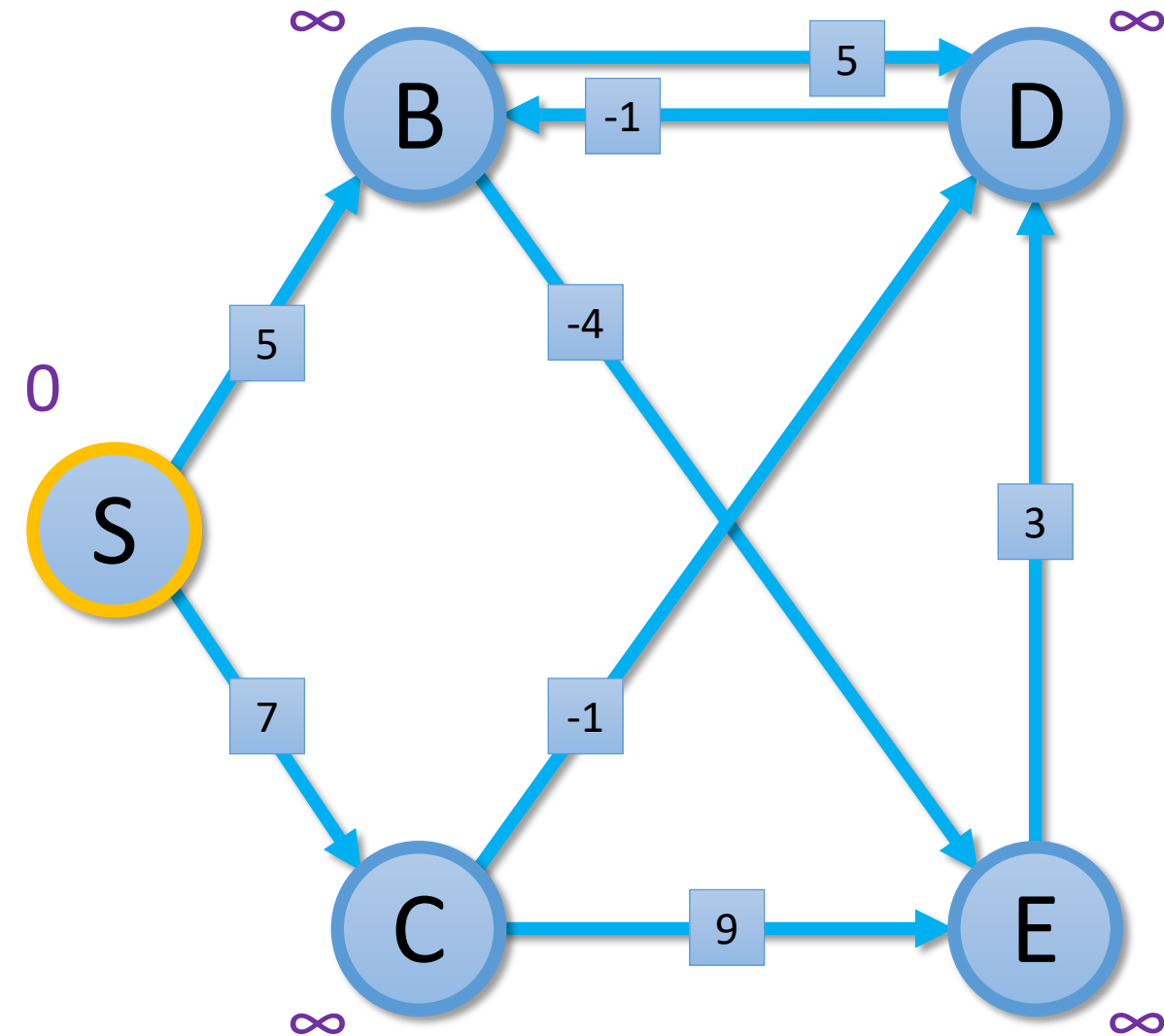


Initialization

Vertex	Predecessor	$i - 1$	$i$
S			
B			
C			
D			
E			

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

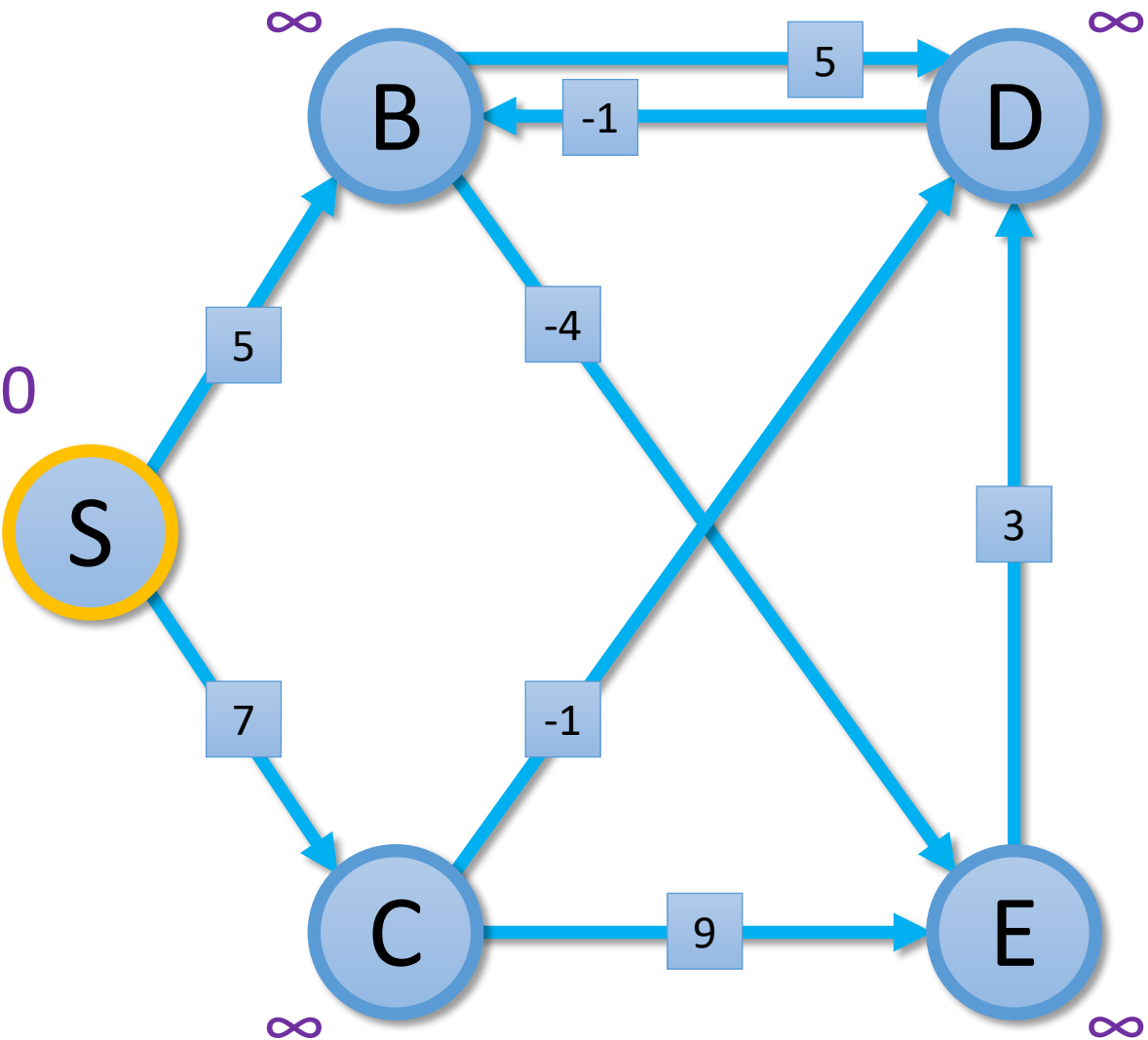


Initialization

Vertex	Predecessor	$i - 1$	$i$
S	S	0	
B	None	$\infty$	
C	None	$\infty$	
D	None	$\infty$	
E	None	$\infty$	

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

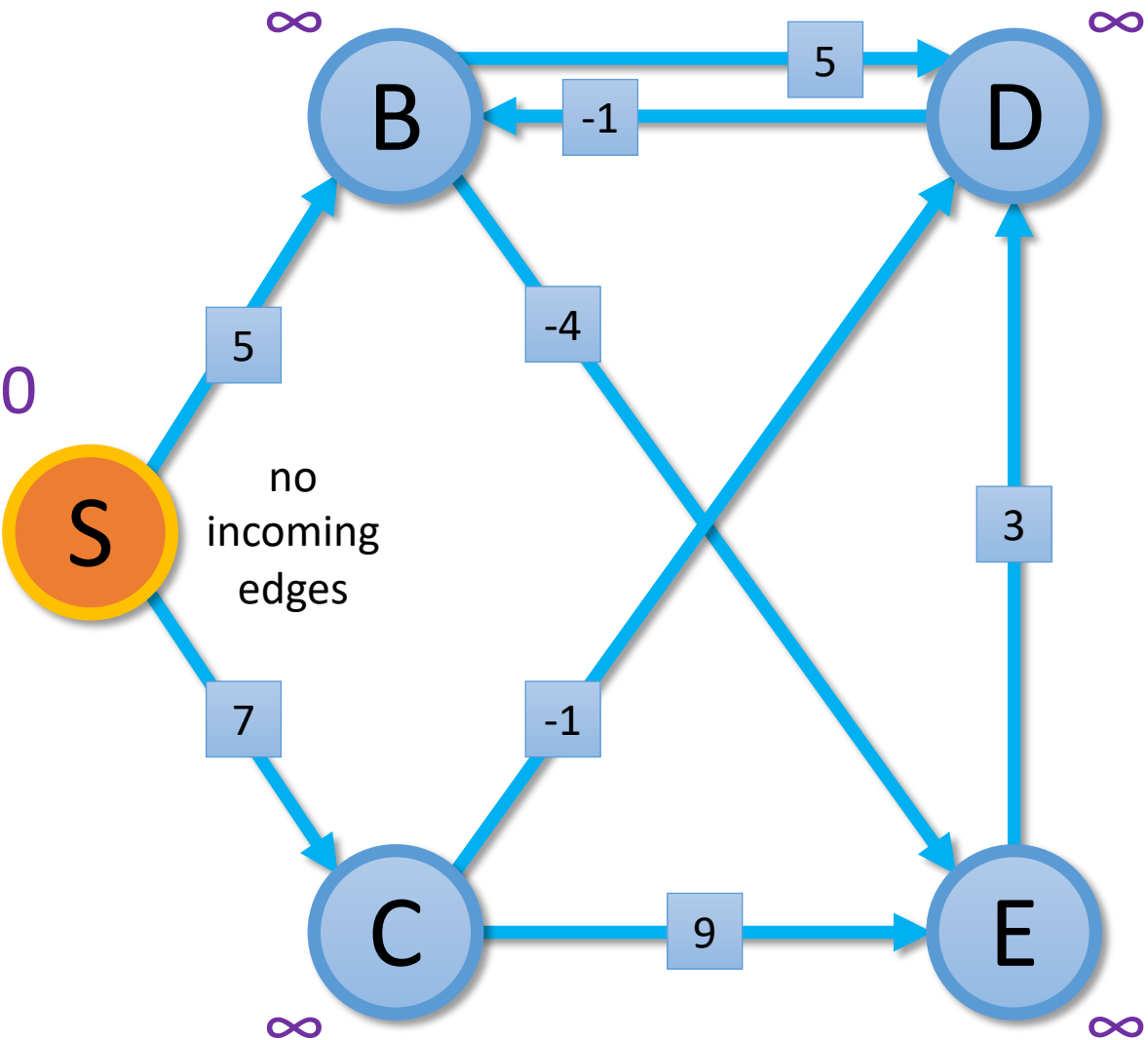


$i = 1$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	
B	None	$\infty$	
C	None	$\infty$	
D	None	$\infty$	
E	None	$\infty$	

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?



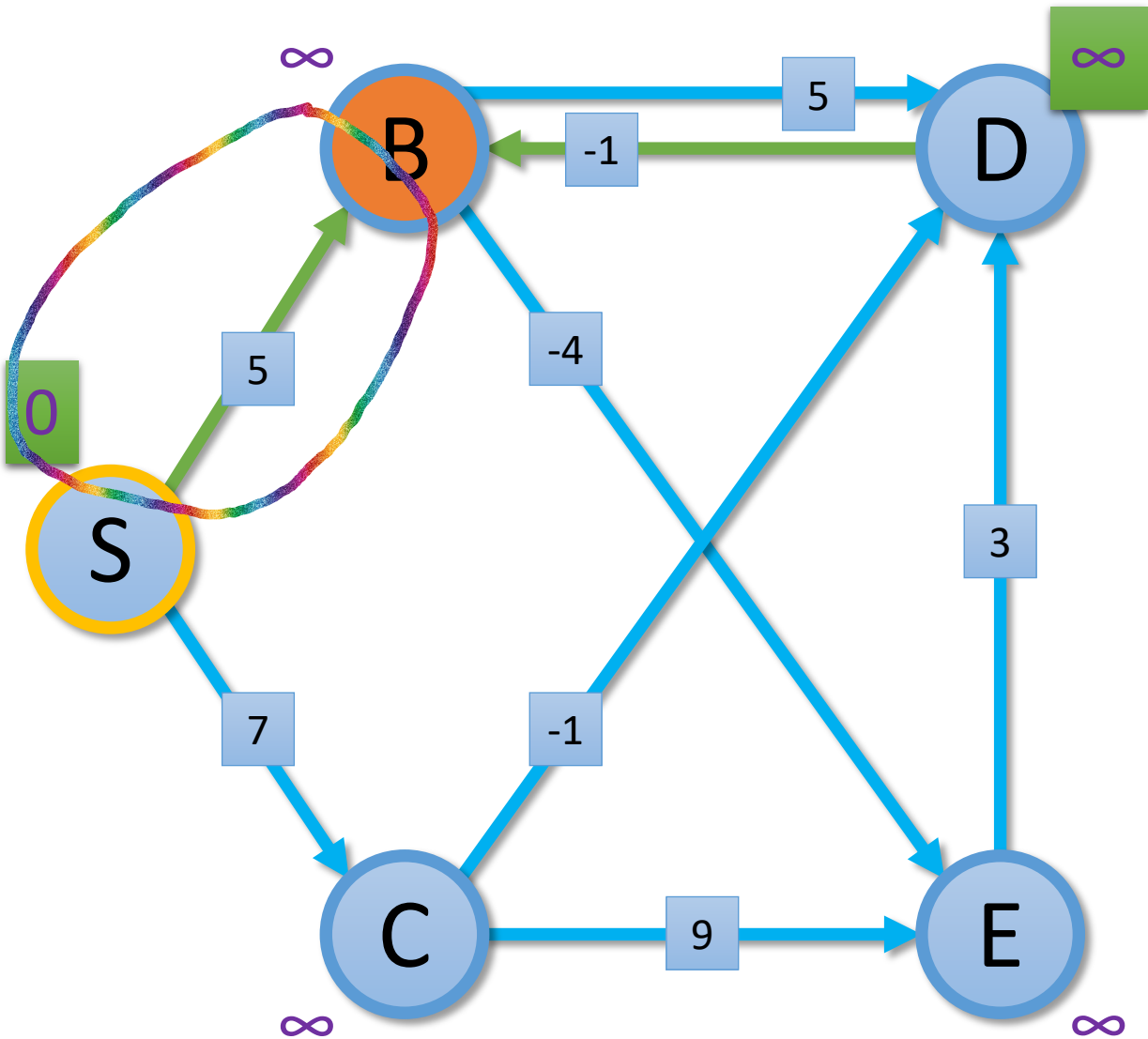
$i = 1$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	None	$\infty$	
C	None	$\infty$	
D	None	$\infty$	
E	None	$\infty$	

Table is rotated when compared to previous example (easier to fit on the slide)



What is the shortest path from S to B?

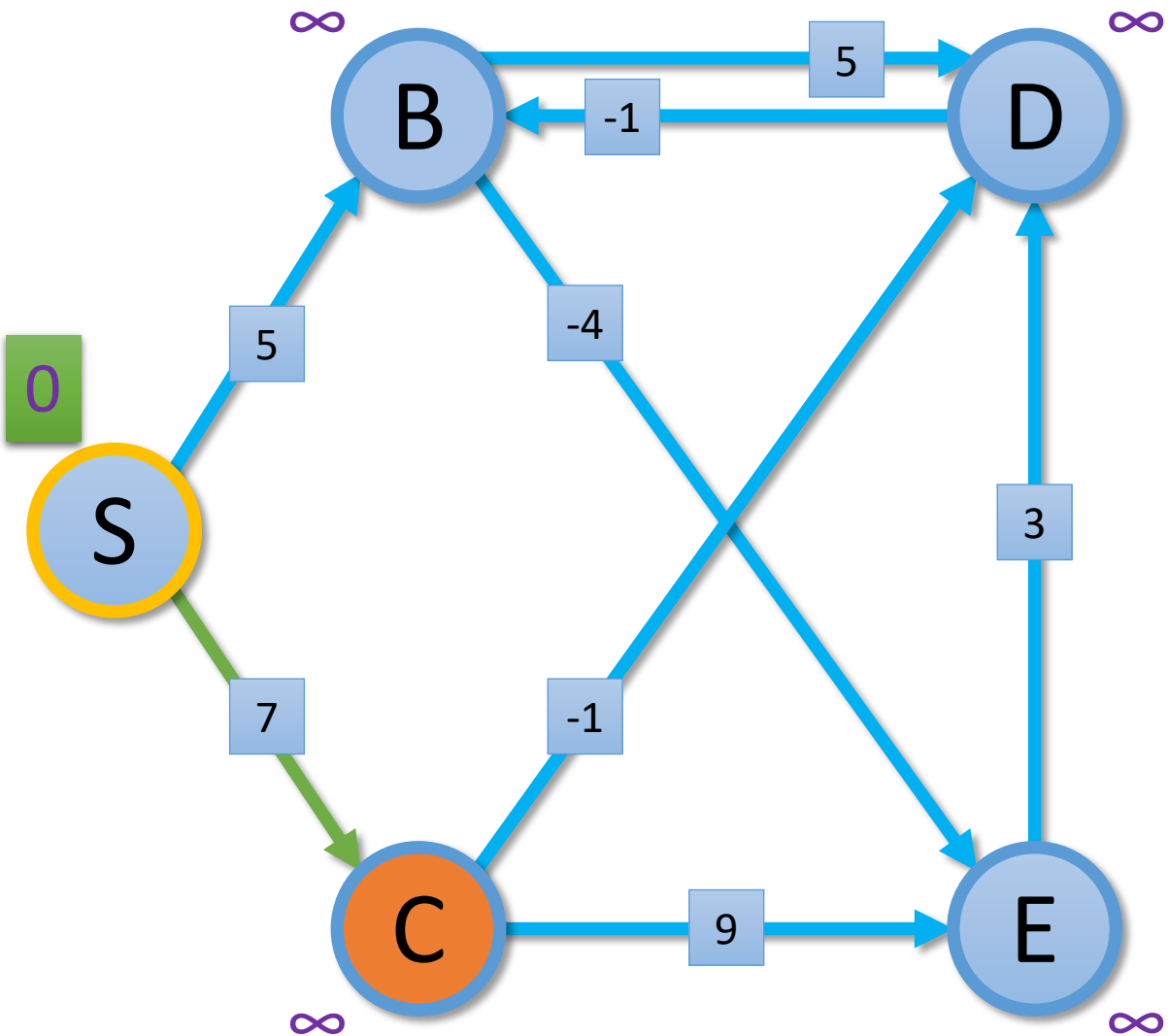


$i = 1$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	S	$\infty$	5
C	None	$\infty$	
D	None	$\infty$	
E	None	$\infty$	

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

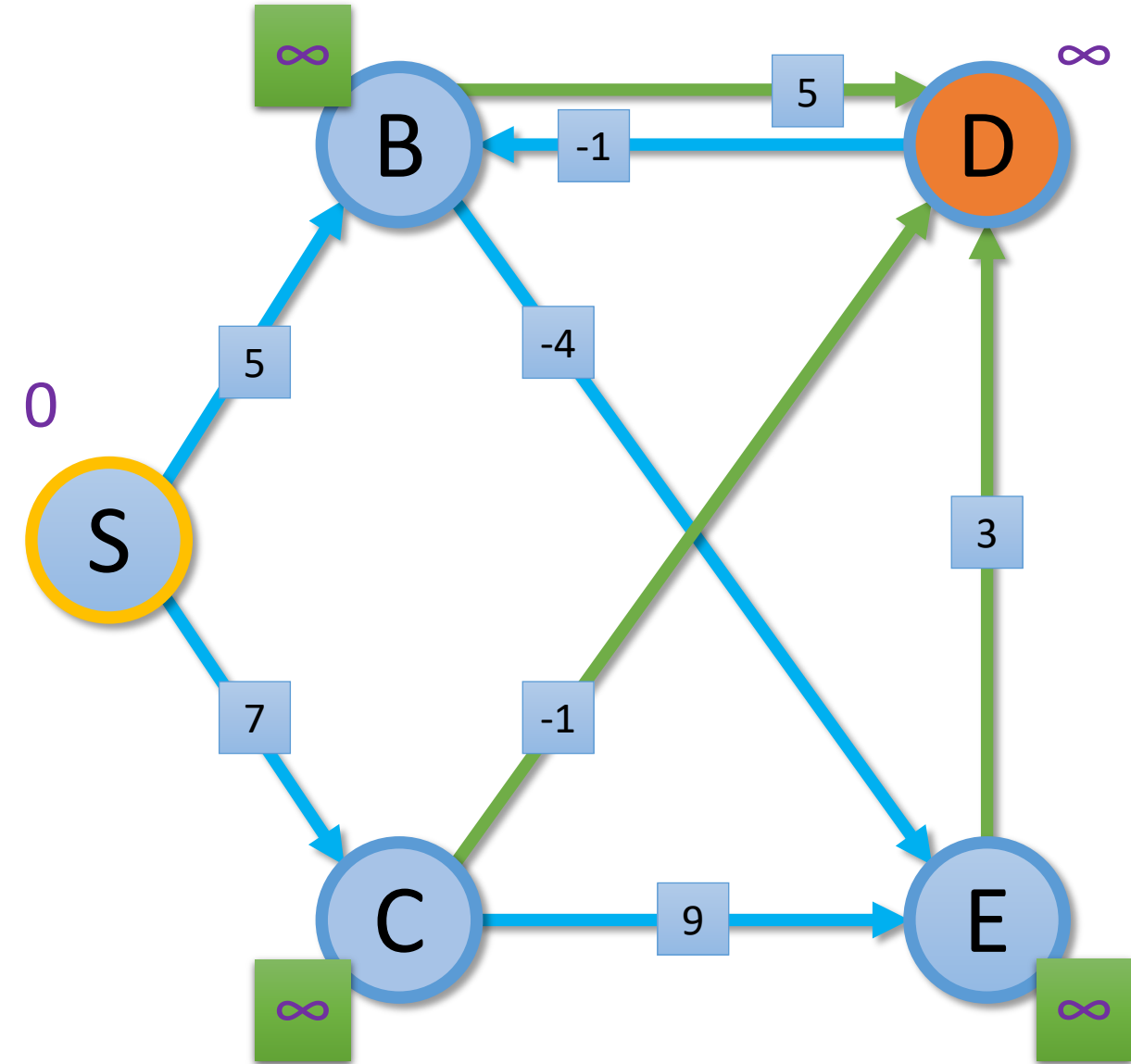


$i = 1$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	S	$\infty$	5
C	S	$\infty$	7
D	None	$\infty$	
E	None	$\infty$	

Table is rotated when compared to previous example (easier to fit on the slide)

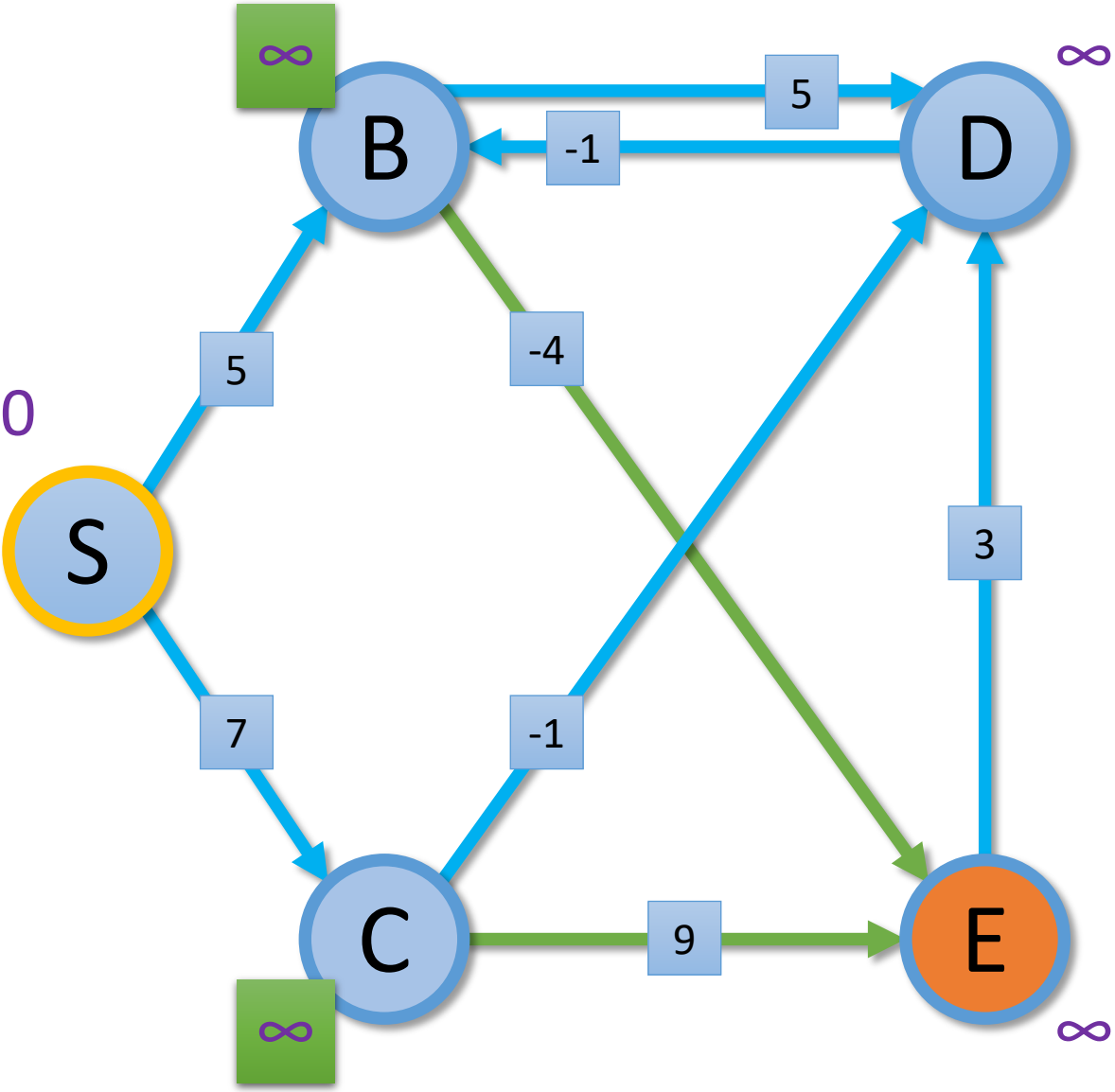
What is the shortest path from S to B?



Vertex	Predecessor	$i-1$	$i$
S	S	0	0
B	S	$\infty$	5
C	S	$\infty$	7
D	None	$\infty$	$\infty$
E	None	$\infty$	

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

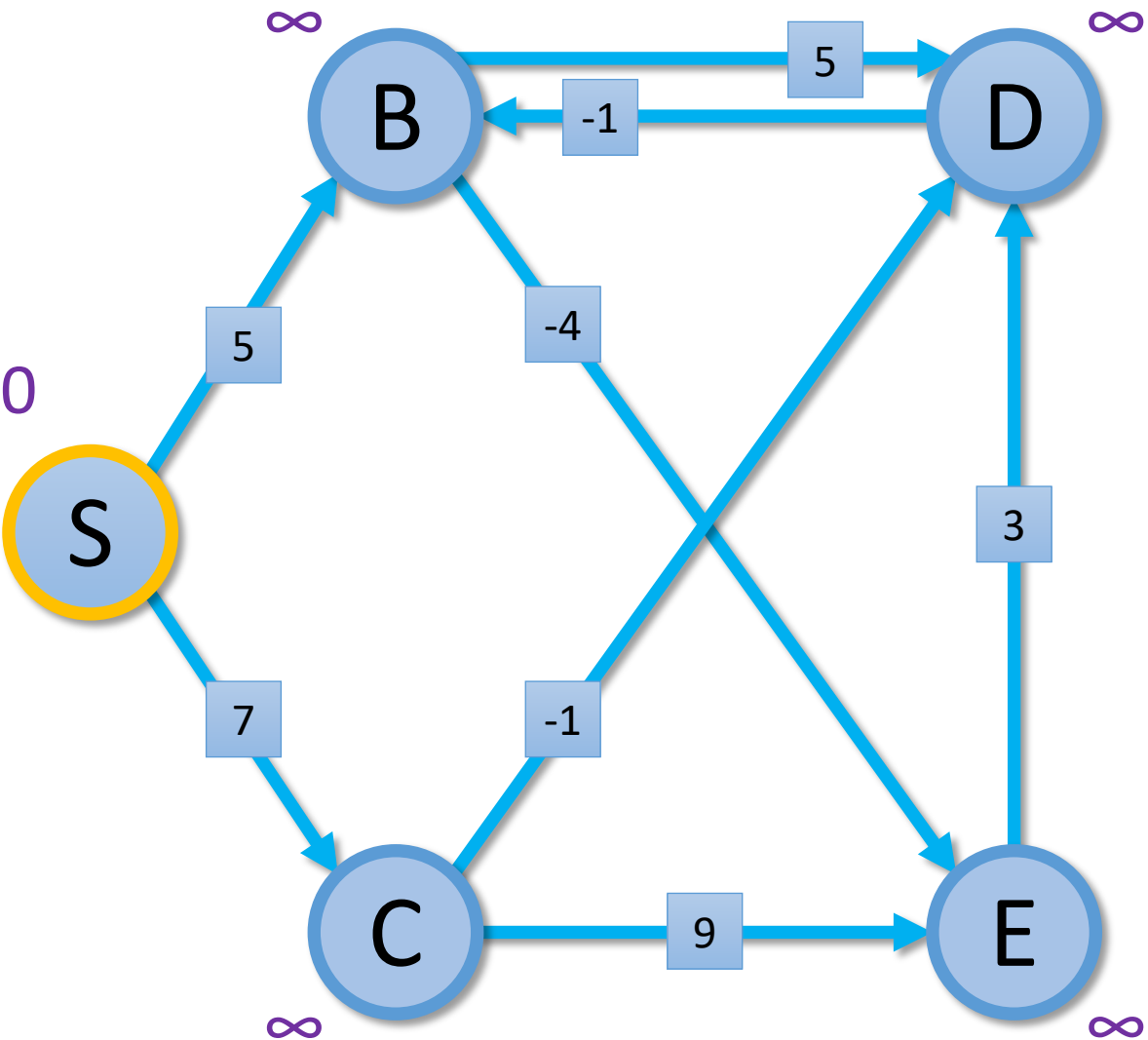


$i = 1$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	S	$\infty$	5
C	S	$\infty$	7
D	None	$\infty$	$\infty$
E	None	$\infty$	$\infty$

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

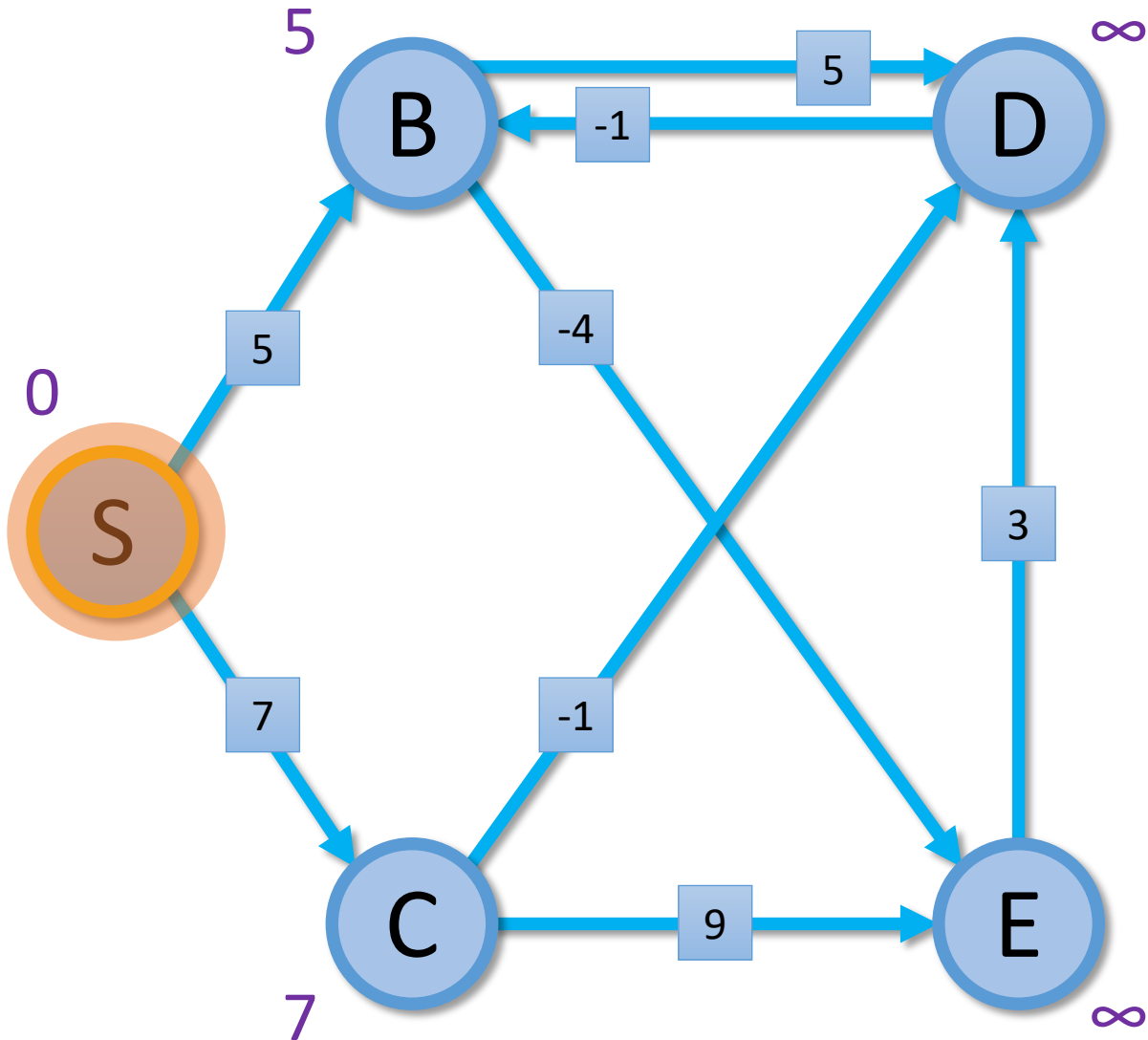


$i = 2$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	S	$\infty$	5
C	S	$\infty$	7
D	None	$\infty$	$\infty$
E	None	$\infty$	$\infty$

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

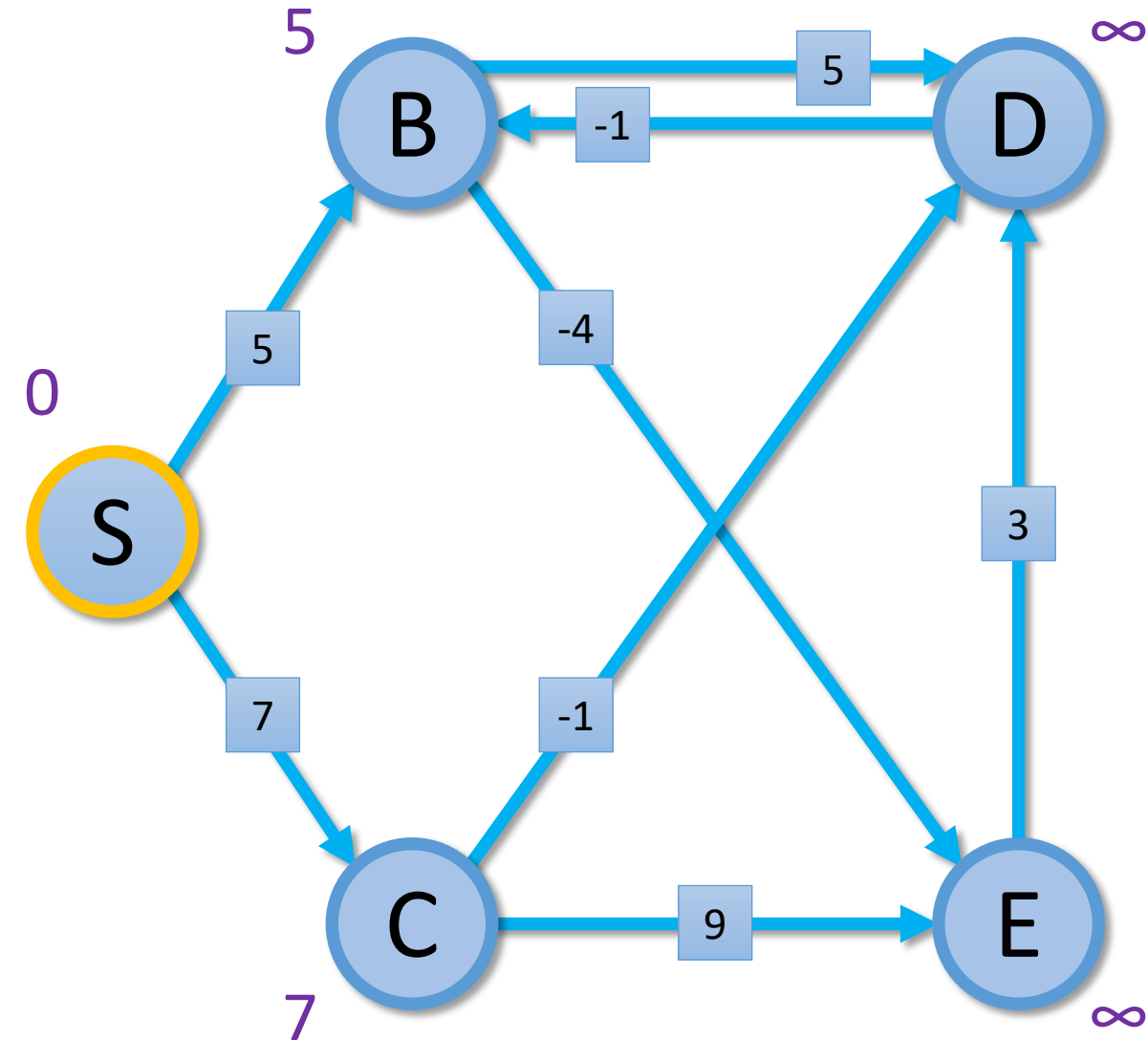


$i = 2$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	S	5	5
C	S	7	7
D	C	$\infty$	6
E	B	$\infty$	1

Table is rotated when compared to previous example (easier to fit on the slide)

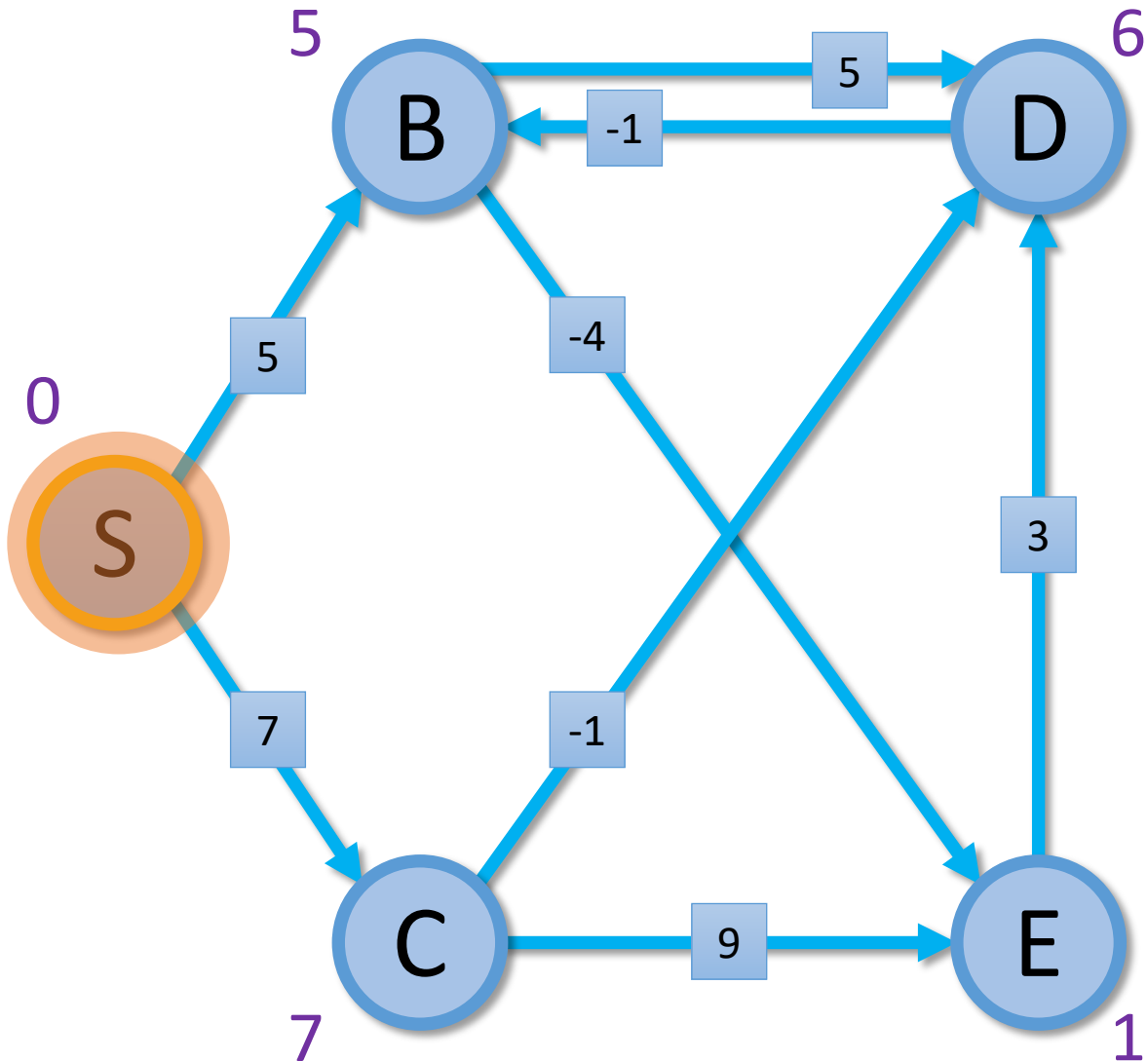
What is the shortest path from S to B?



Vertex	Predecessor	$i-1$		$i$
S	S	0	←	0
B	S	5	←	5
C	S	7	←	7
D	C	$\infty$	←	6
E	B	$\infty$	←	1

Table is rotated when compared to previous example  
(easier to fit on the slide)

What is the shortest path from S to B?



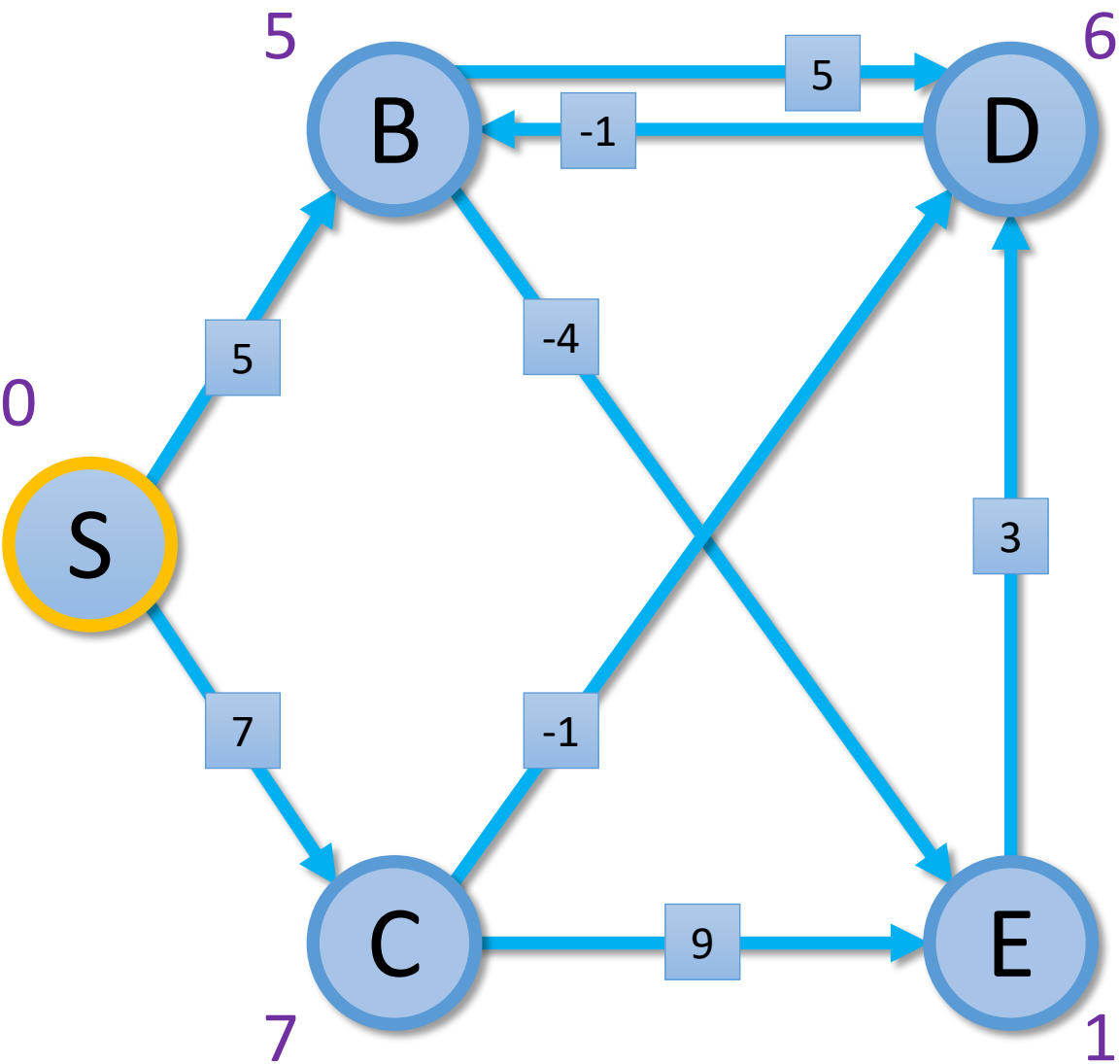
$i = 3$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	S	5	5
C	S	7	7
D	E	6	4
E	B	1	1

Table is rotated when compared to previous example (easier to fit on the slide)



What is the shortest path from S to B?

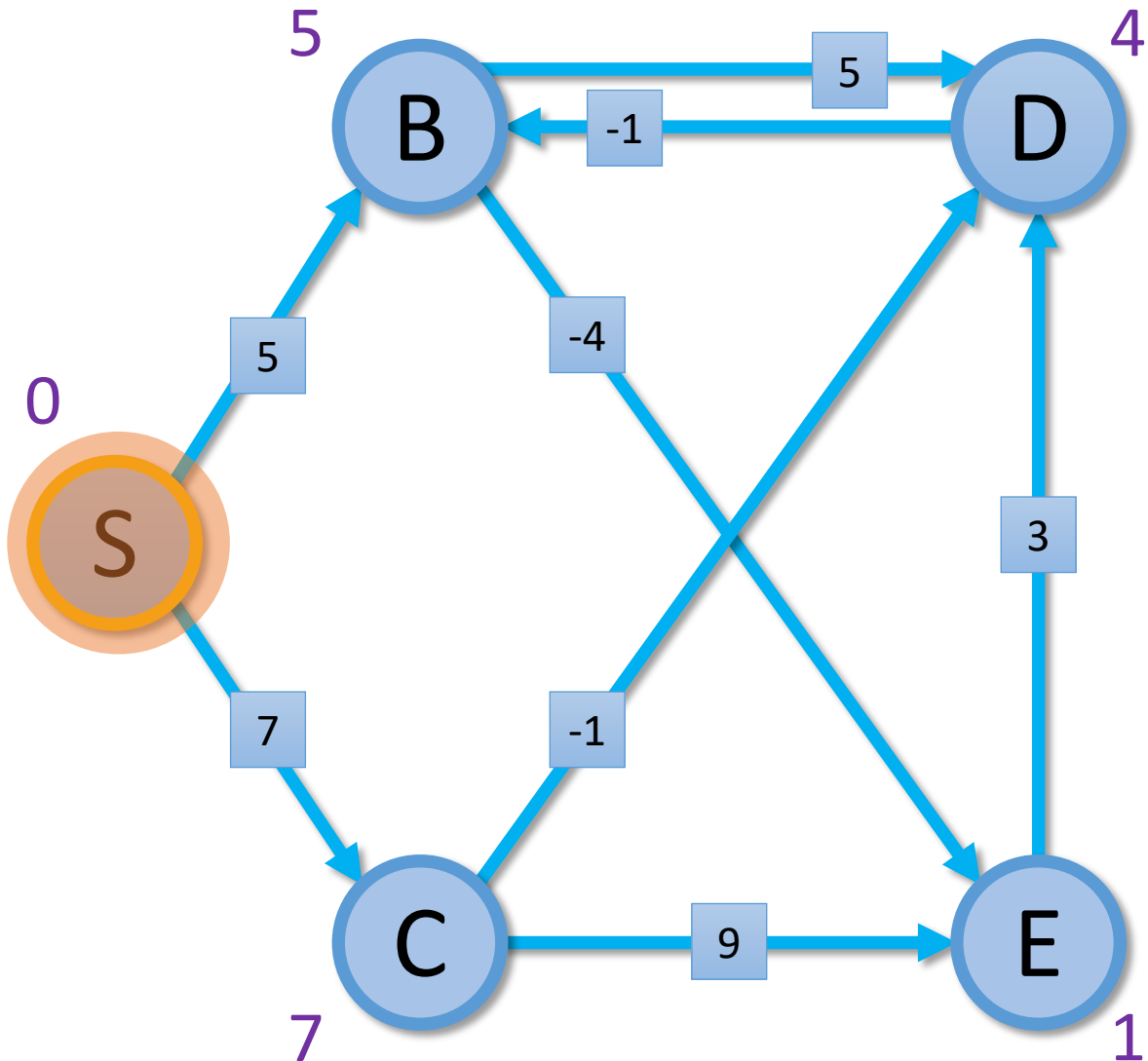


$i = 4$

Vertex	Predecessor	$i - 1$		$i$
S	S	0	←	0
B	S	5	←	5
C	S	7	←	7
D	E	6	←	4
E	B	1	←	1

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

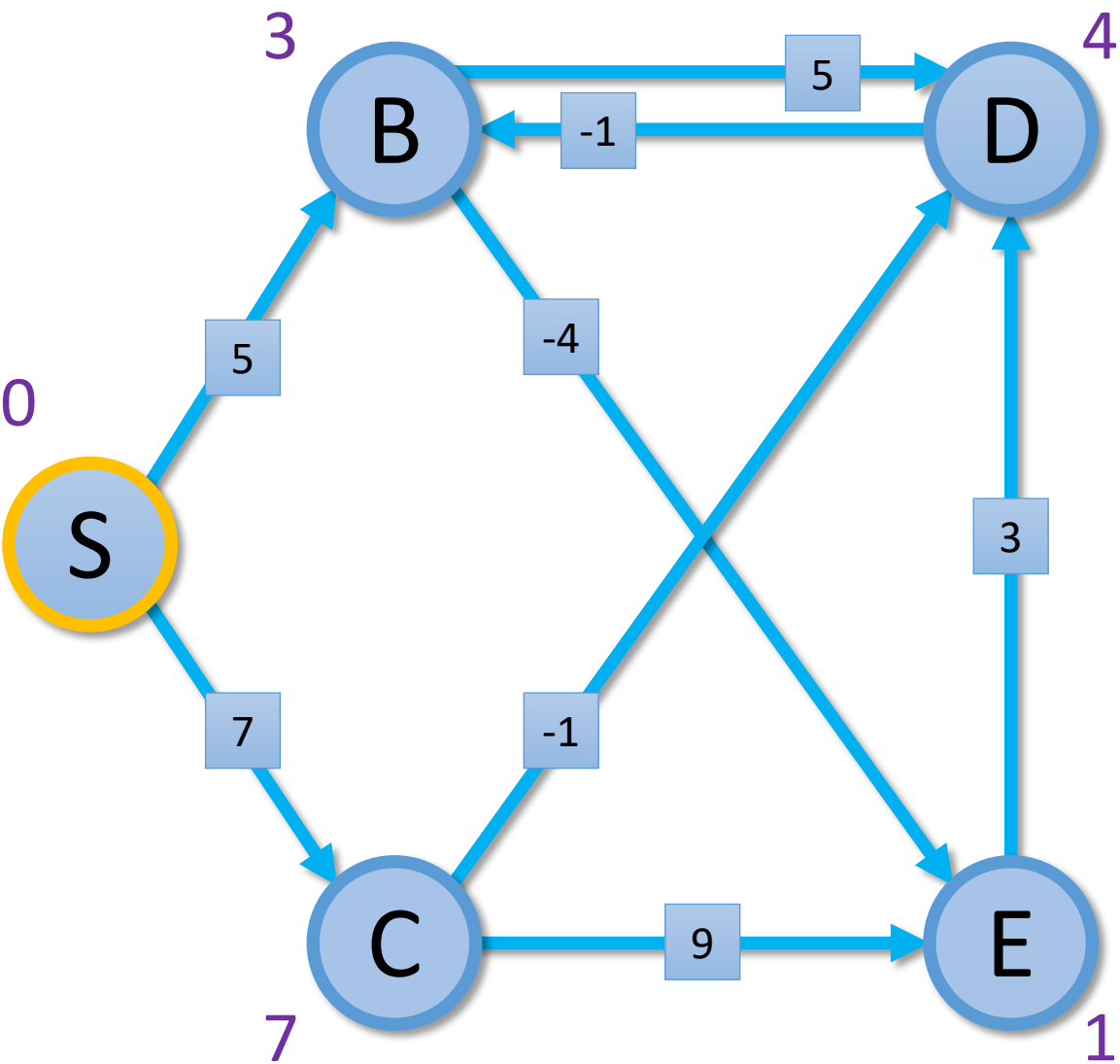


$i = 4$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	D	5	3
C	S	7	7
D	E	4	4
E	B	1	1

Table is rotated when compared to previous example (easier to fit on the slide)

What is the shortest path from S to B?

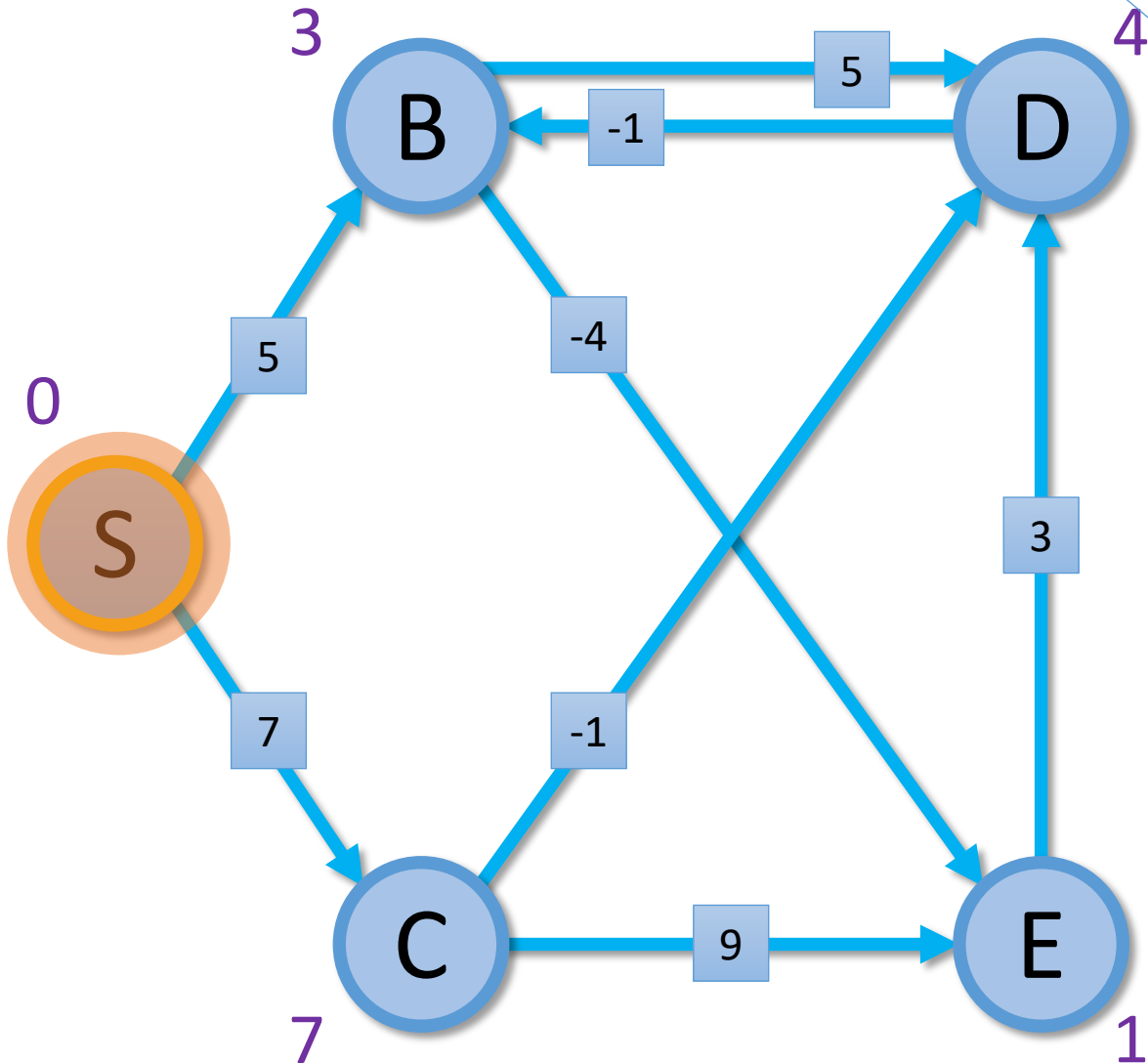


Vertex	Predecessor	$i - 1$		$i$
S	S	0	←	0
B	D	3	←	3
C	S	7	←	7
D	E	4	←	4
E	B	1	←	1

Table is rotated when compared to previous example (easier to fit on the slide)

Last iteration is only to detect negative cycles.

What is the shortest path from S to B?



$i = 5$

Vertex	Predecessor	$i - 1$	$i$
S	S	0	0
B	D	3	3
C	S	7	7
D	E	4	4
E	B	1	-1

Table is rotated when compared to previous example (easier to fit on the slide)

# Summary of Bellman-Ford

- Single-source shortest path problem (like Dijkstra's)
- Running time is  $O(nm)$
- Works with negative weights
- Can detect negative cycles
  - Run the loop  $n$  times and if a path length goes down, then you've found a negative cycle