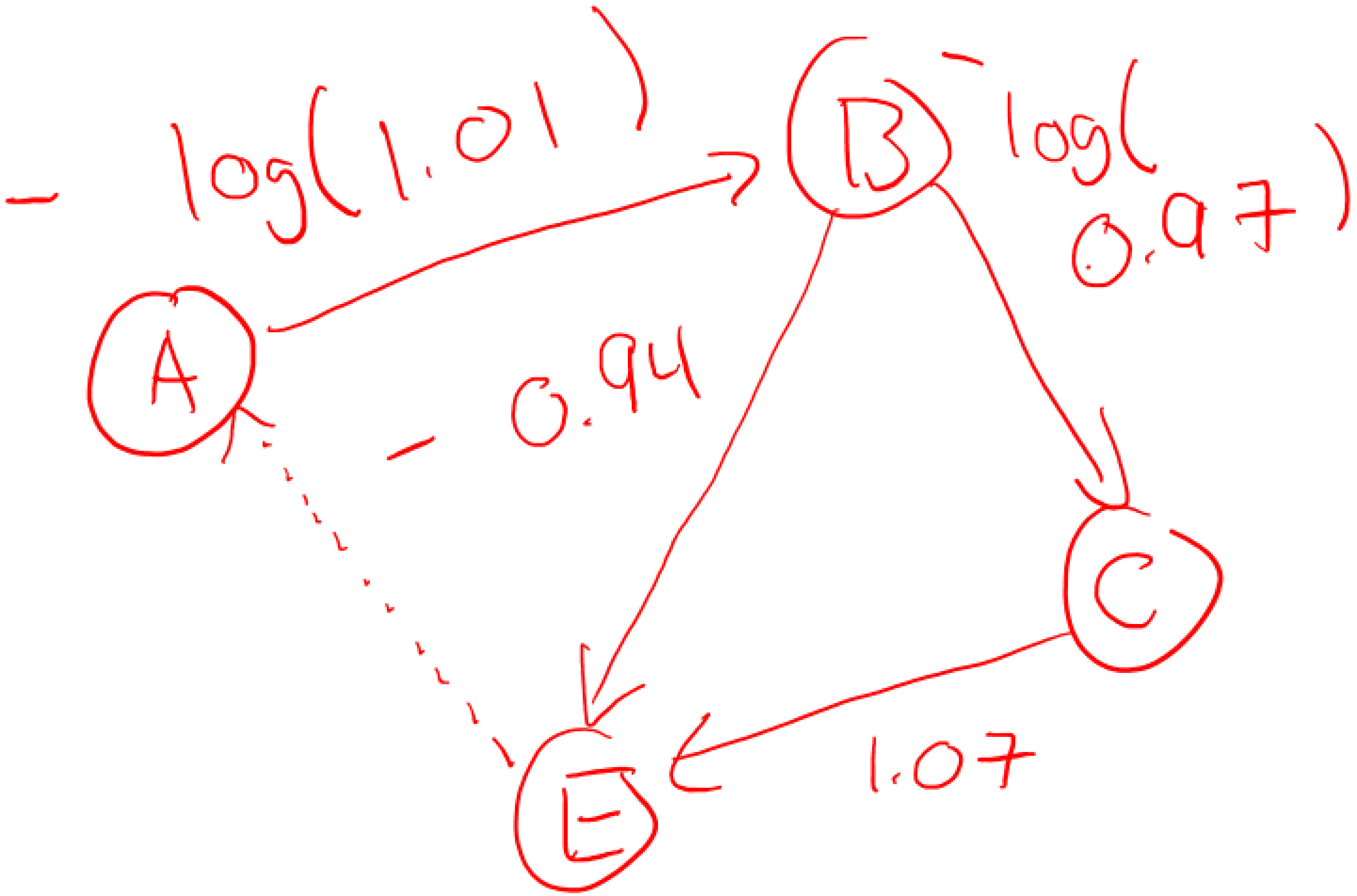


− log(1.01)
− log(0.97)
− 0.94
1.07
A
B
C
E

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

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

Outline

Topics and Learning Objectives

- Discuss and analyze the Bellman-Ford Algorithm

Exercise

- Bellman-Ford Walk-through

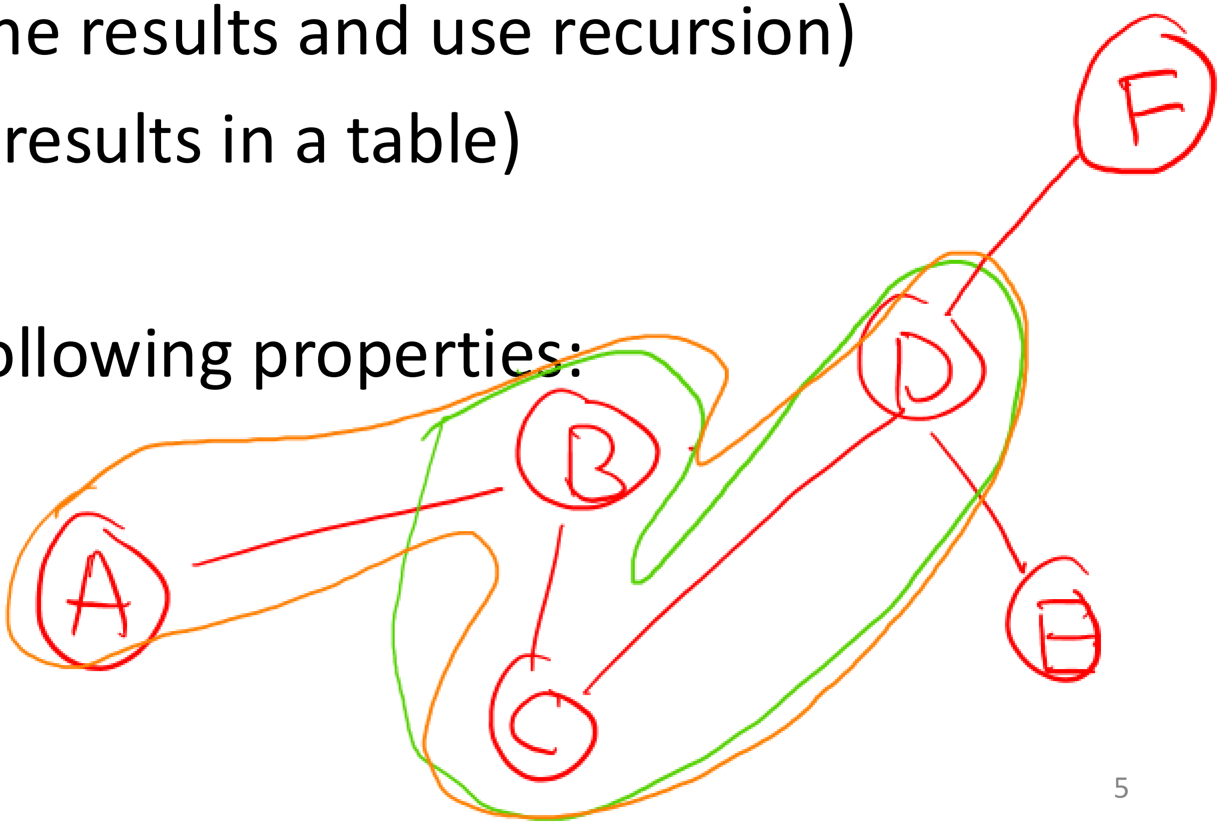
Dynamic Programming

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

<u>Key Idea</u>: leverage overlapping subproblems and optimal substructure.

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

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)

Question

All-Pairs Shortest Path

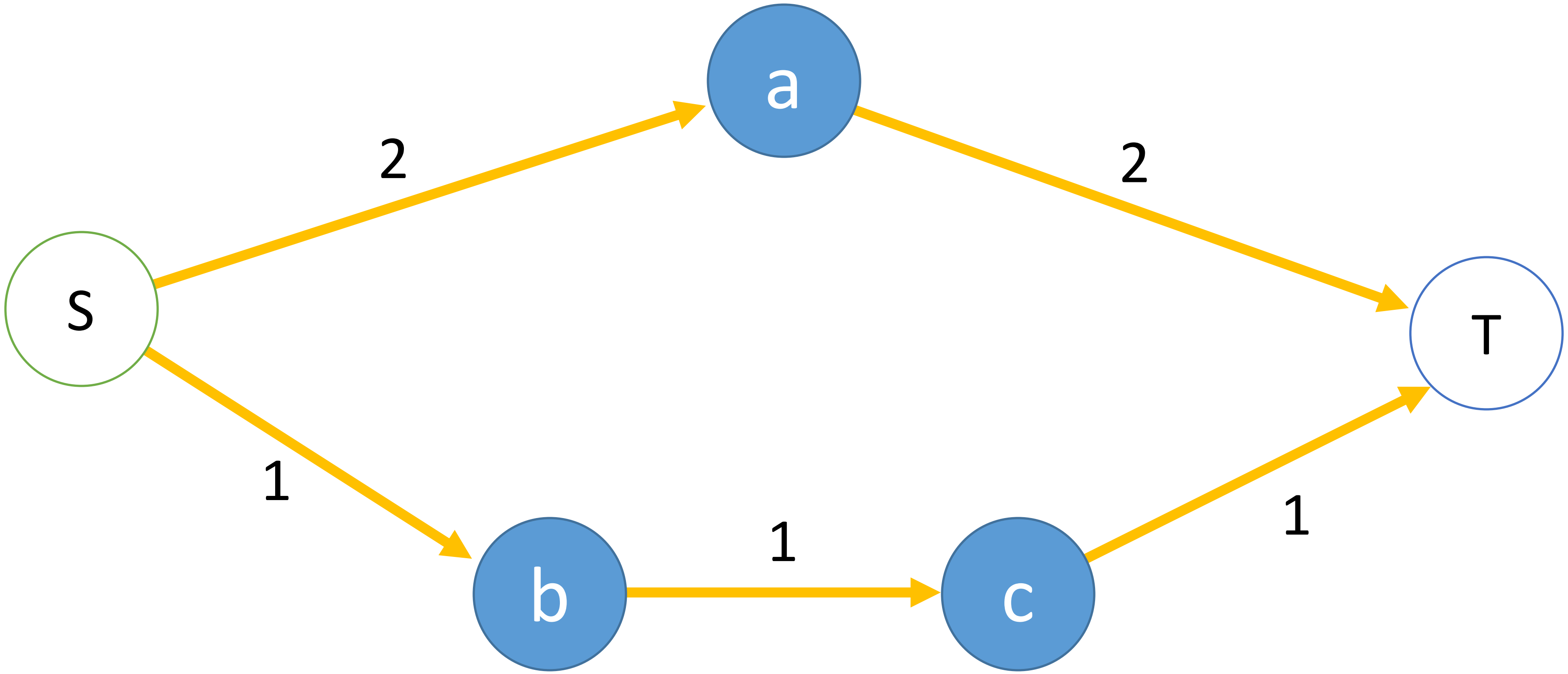
	Sparse Graphs	Dense Graphs
Dijkstra's n times	$O(n^2 \lg n)$	$O(n^3 \lg n)$
Bellman-Ford n times	$O(n^3)$	$O(n^4)$
Floyd-Warshall	$O(n^3)$	$O(n^3)$

Not take into account the distributed nature of the problem

- What algorithm would you choose for sparse graphs?
  - Dijkstra's n times if there are no negative edges, Floyd-Warshall otherwise
- What algorithm would you choose for dense graphs?
  - Always Floyd-Warshall

Example 1

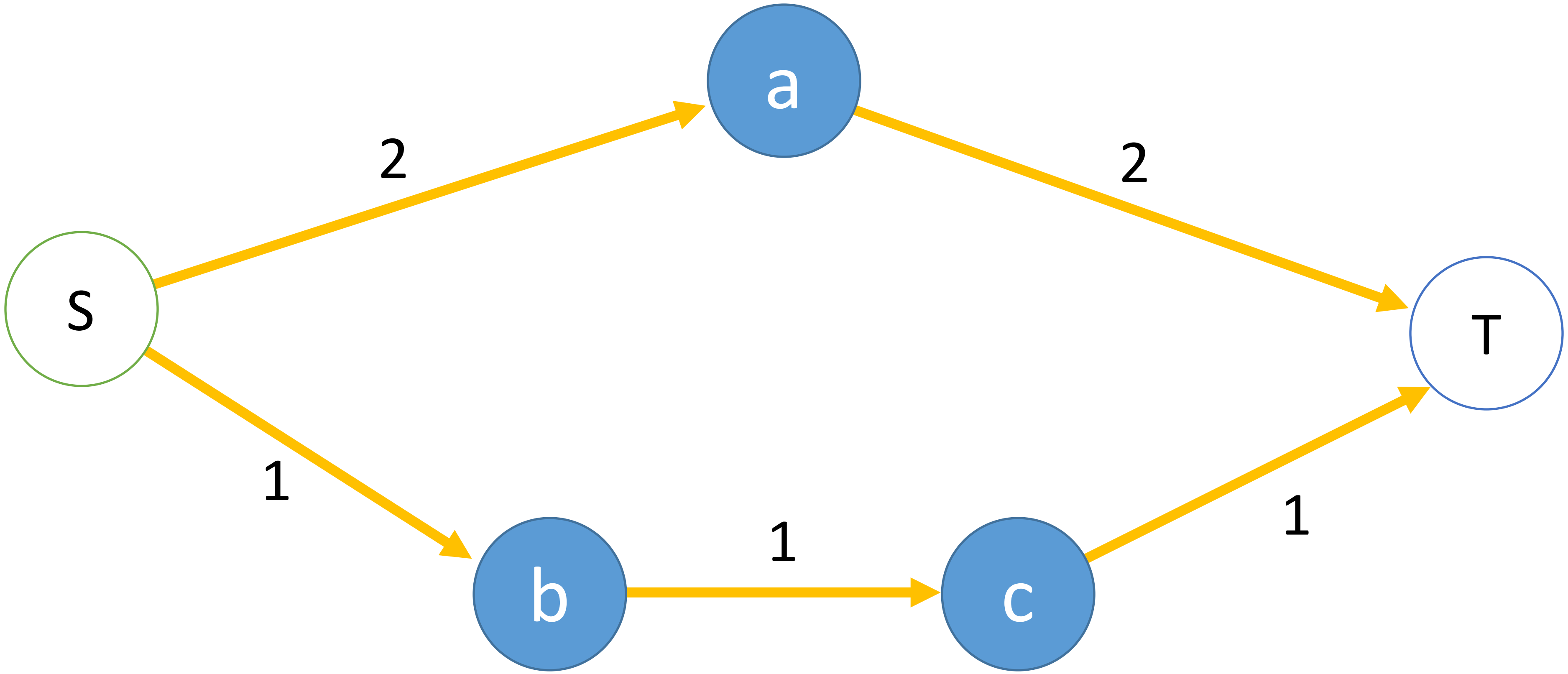
What is the shortest path from S to T using 0 edges?



Subproblem: consider only a subset of the possible paths.

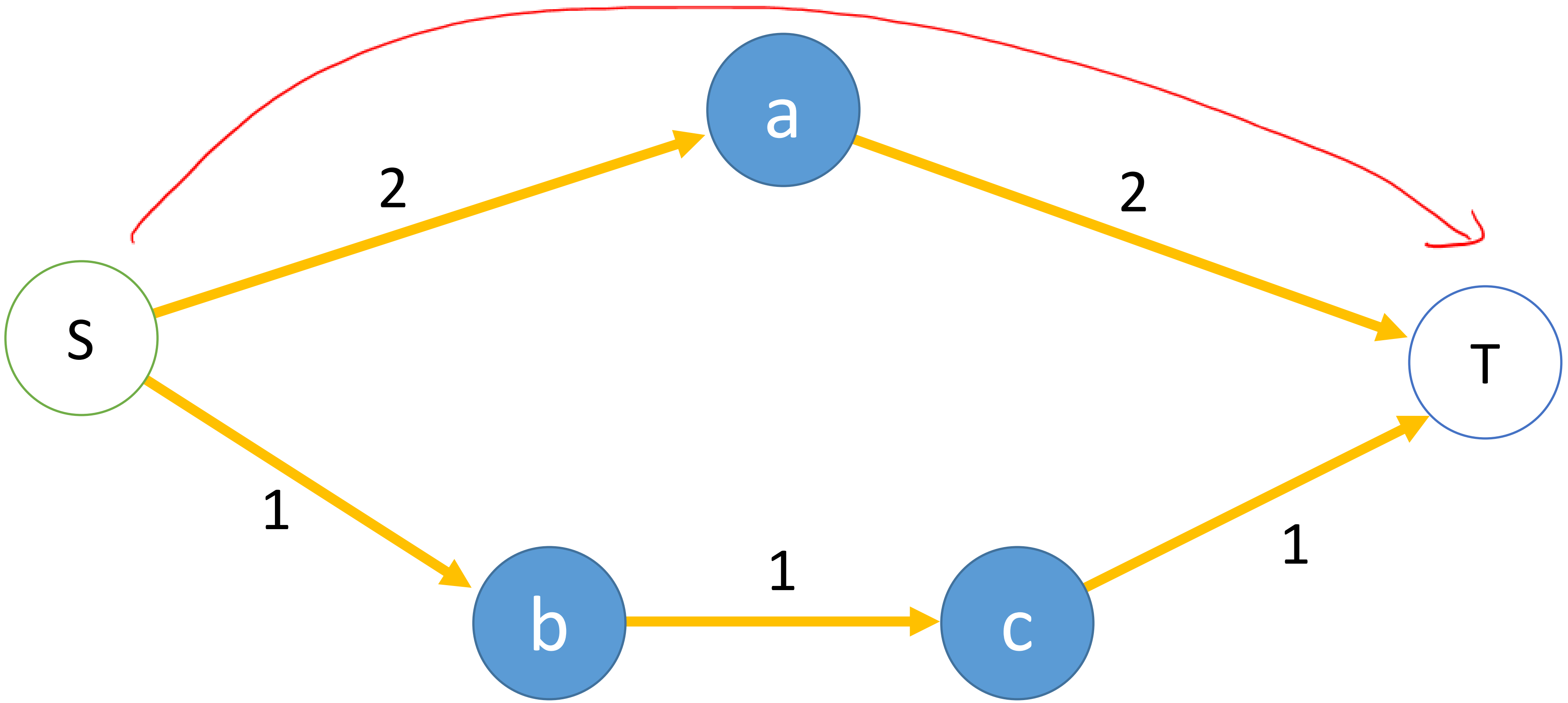
Example 1

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



Example 1

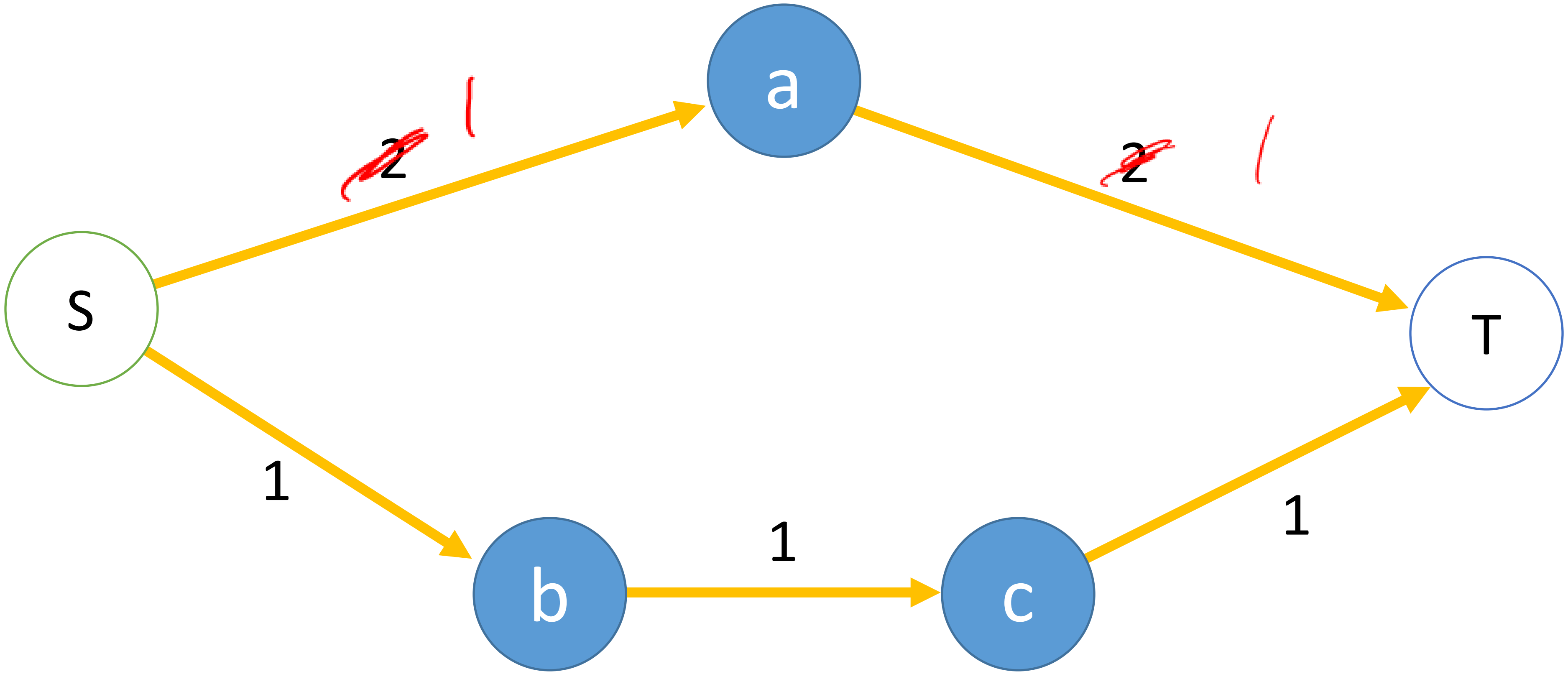
What is the shortest path using 2 edges?



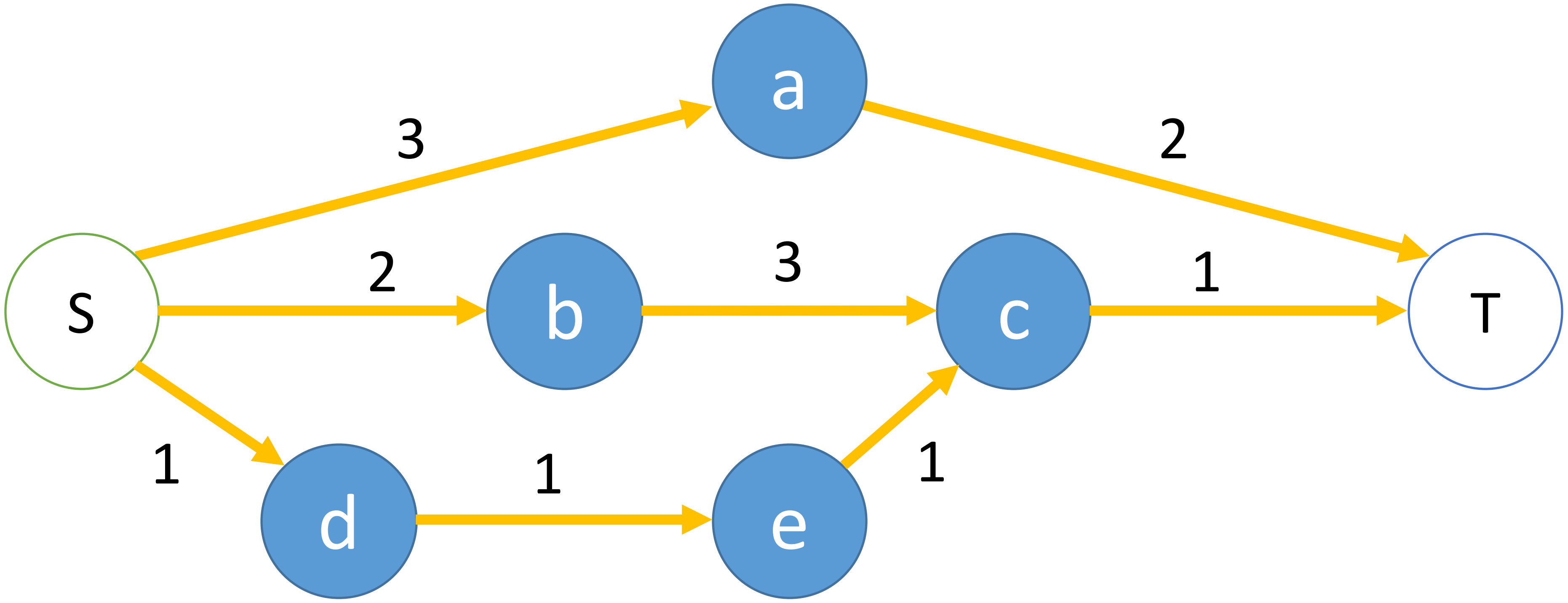
Example 1

What is the shortest path using 3 edges?

What is the shortest path using 2 edges?

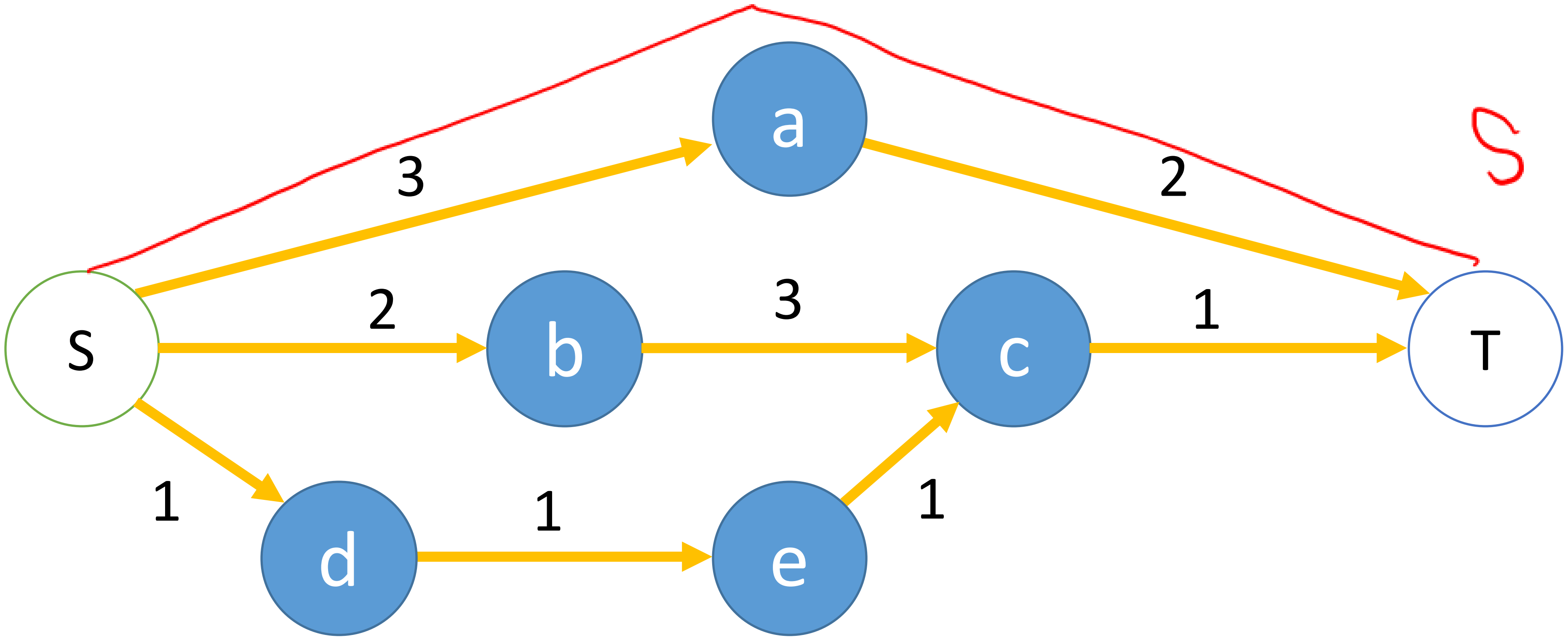


Example 2



What is the shortest path with <u>at most</u> 1 edge?

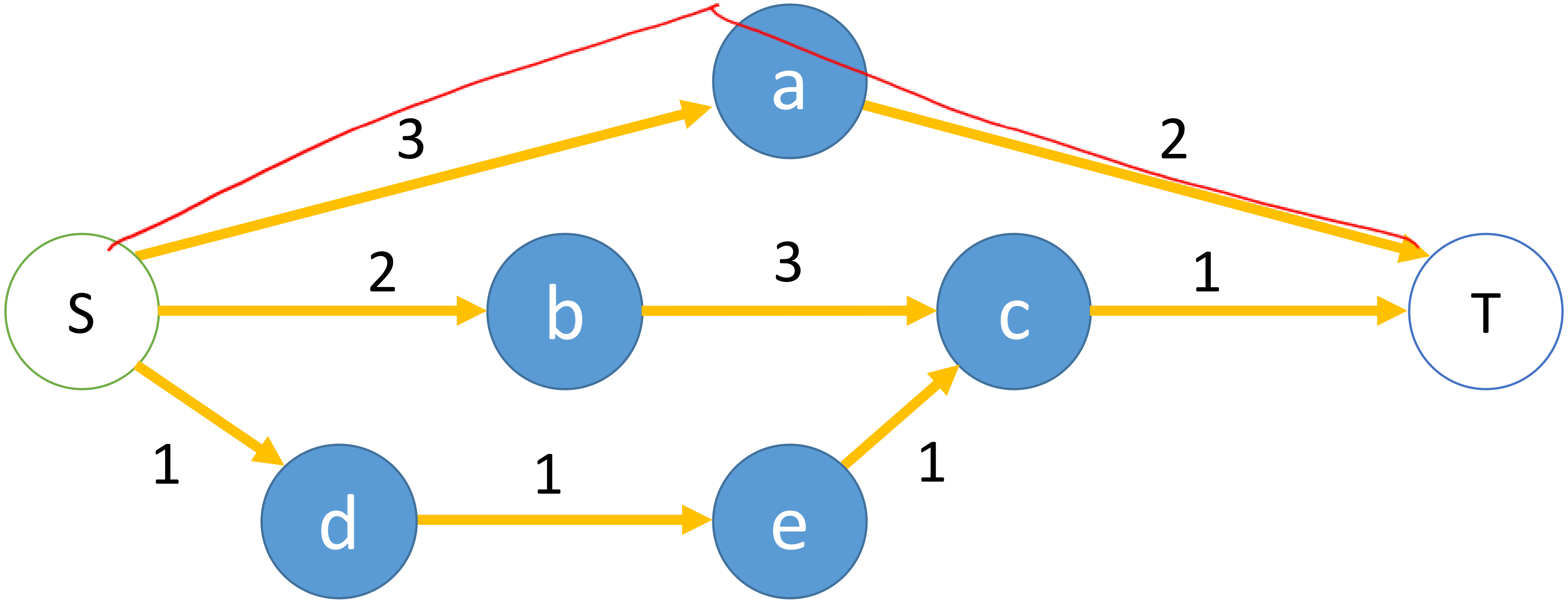
Example 2



Shortest path with at most 2 edges

Example 2

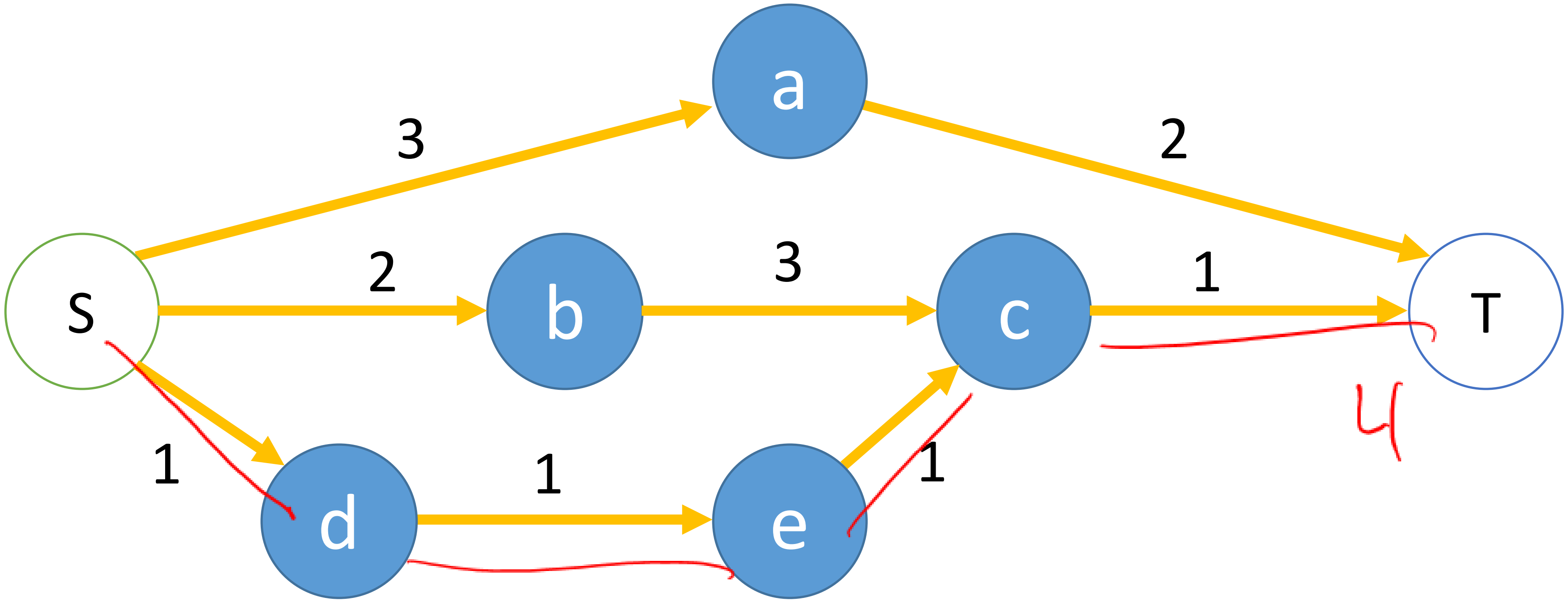
We didn't gain anything by adding the edge



Shortest path with at most 2 edges

Shortest path with at most 3 edges

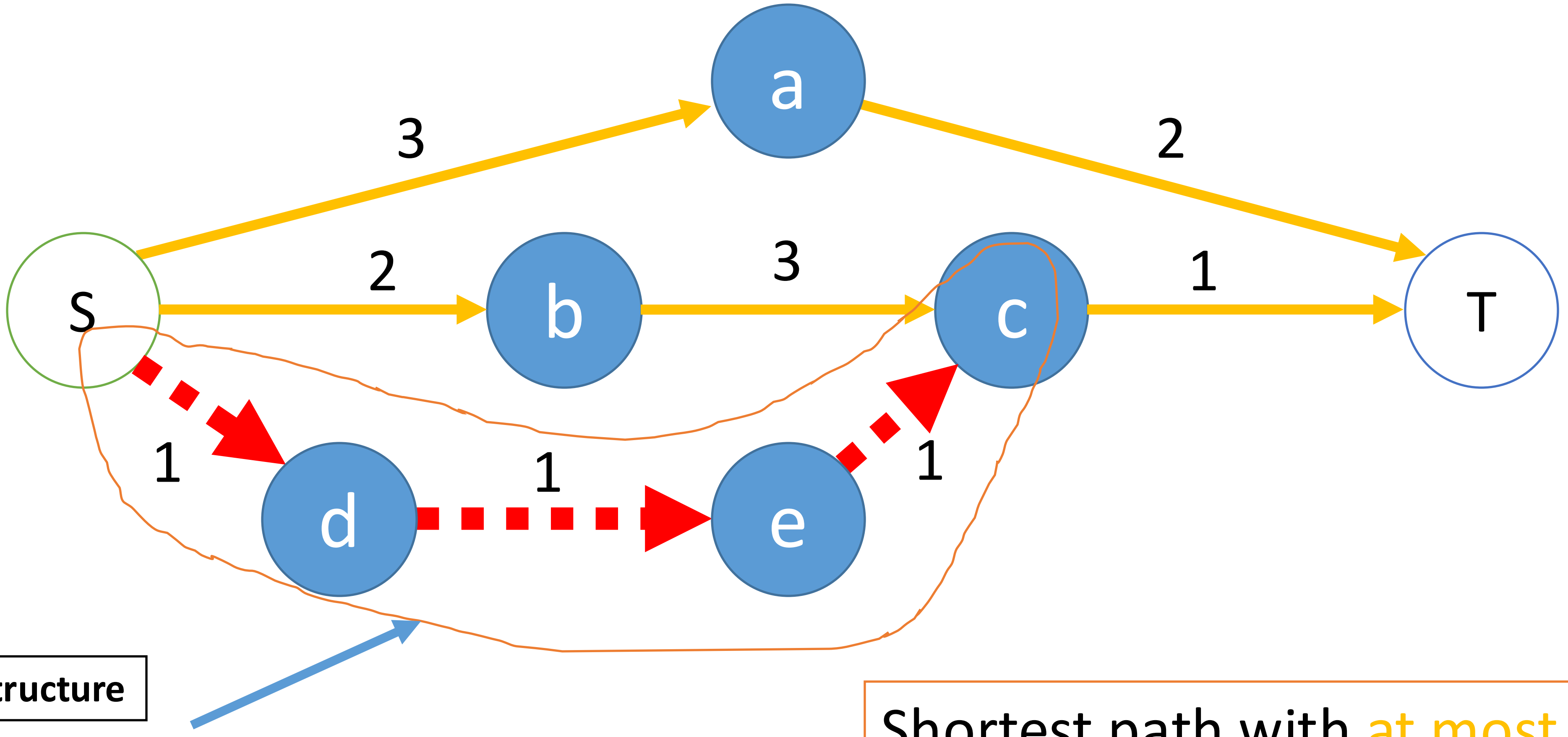
Example 2



Shortest path with at most 4 edges

Example 2

If the path 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.



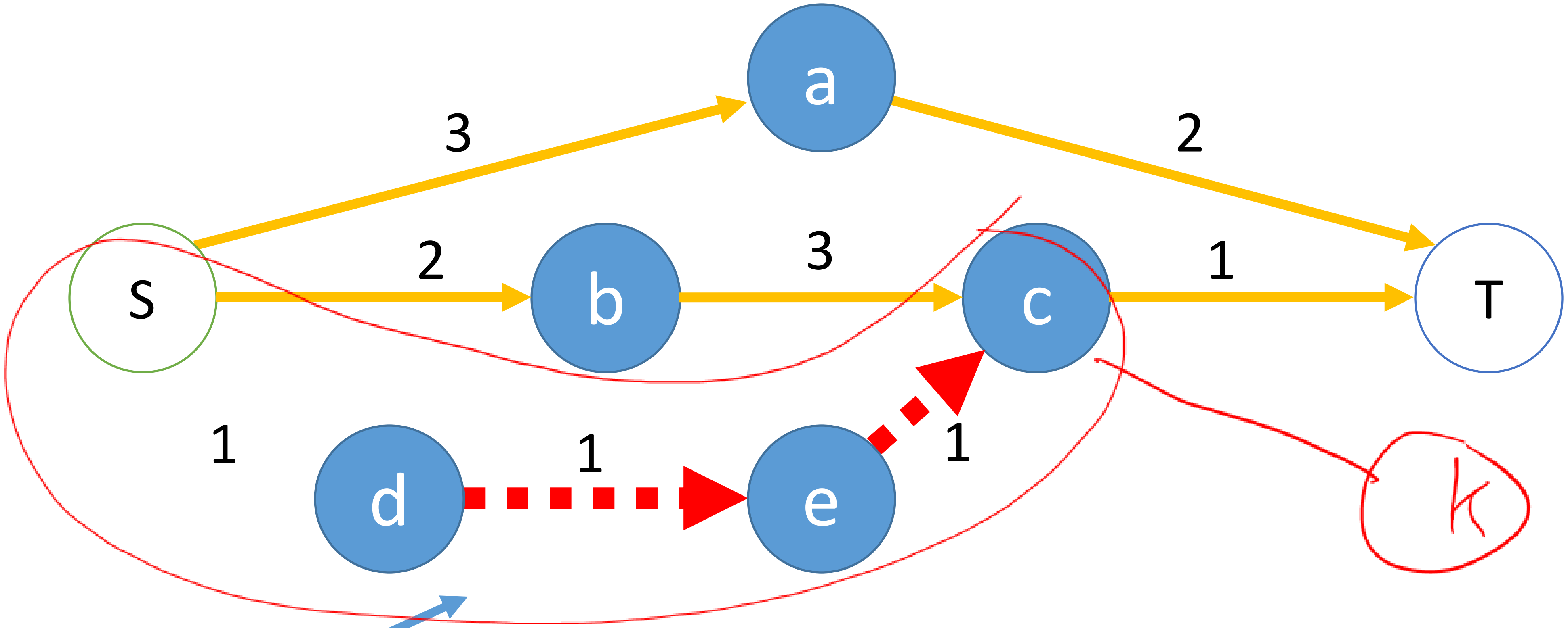
Optimal Substructure

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

Shortest path with at most 4 edges

Example 2

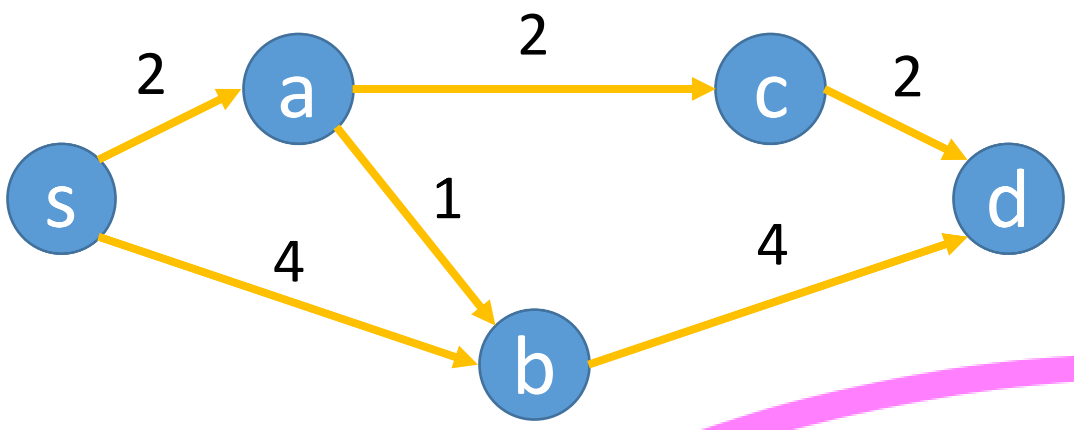
The path from D to C is used as part of the shortest path from S to T. And as part of the shortest path from S to C.



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



Here's the table

Max Number Of Edges On Path

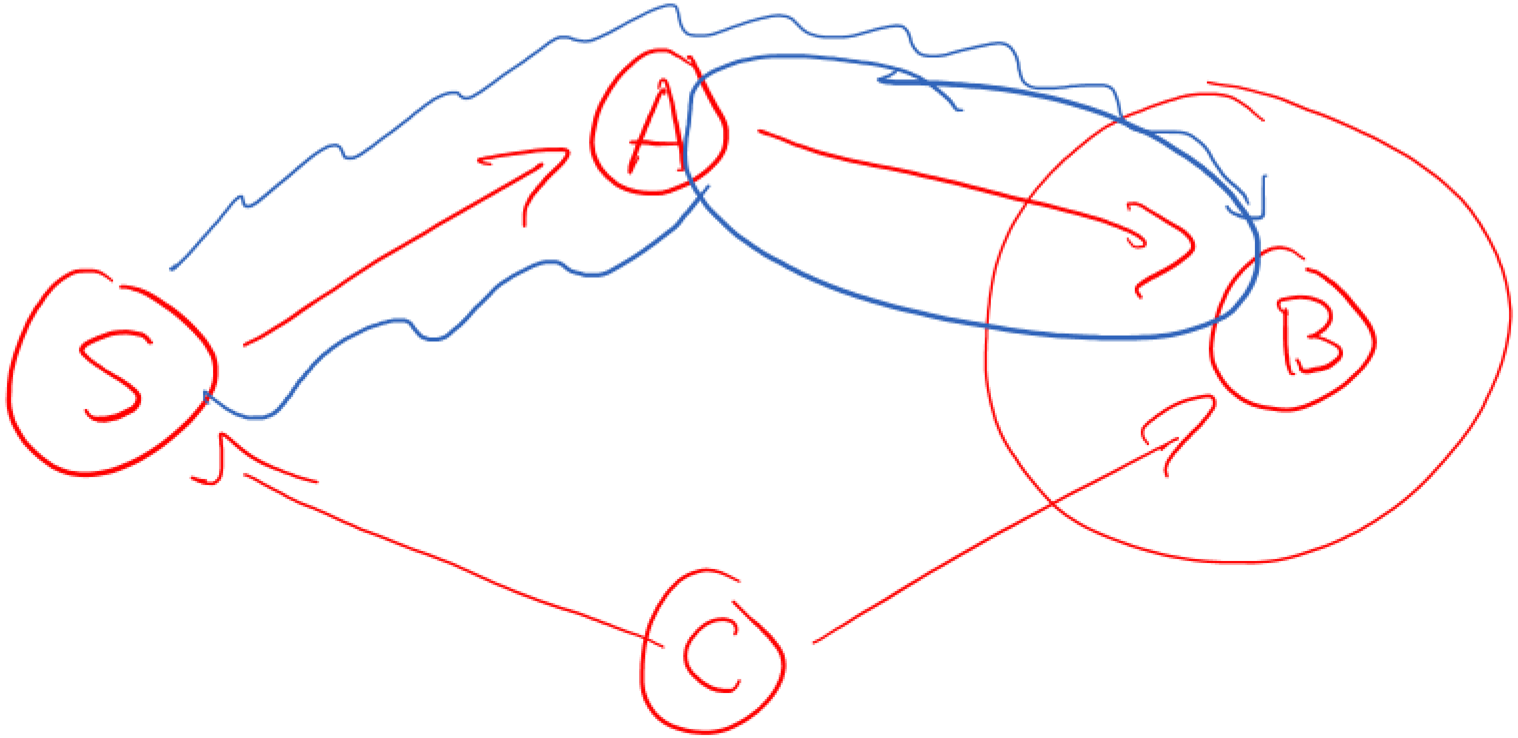
	i					
i	4					
	3					
	2					
	1					
	0					
		s	a	b	c	d
		v				

End Vertex

```
FUNCTION BellmanFord(G, start_vertex)
    n = G.vertices.length
    path_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
    path_lengths[0, start_vertex] = 0
```

```
FUNCTION BellmanFord(G, start_vertex)
    n = G.vertices.length
    path_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
    path_lengths[0, start_vertex] = 0
    FOR num_edges IN [1 ..< n]
```

Why won’t we need more than n-1 edges?



```
FUNCTION BellmanFord(G, start_vertex)
    n = G.vertices.length
    path_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
    path_lengths[0, start_vertex] = 0
    FOR num_edges IN [1 ..< n]
        FOR v IN G.vertices
            min_len = INFINITY
            FOR (vFrom, v) IN G.edges
                len = path_lengths[num_edges - 1, vFrom] + G.edges[vFrom, v].cost
                IF len < min_len
                    min_len = len
            path_lengths[num_edges, v] = min(path_lengths[num_edges - 1, v],
                                             min_len)
```

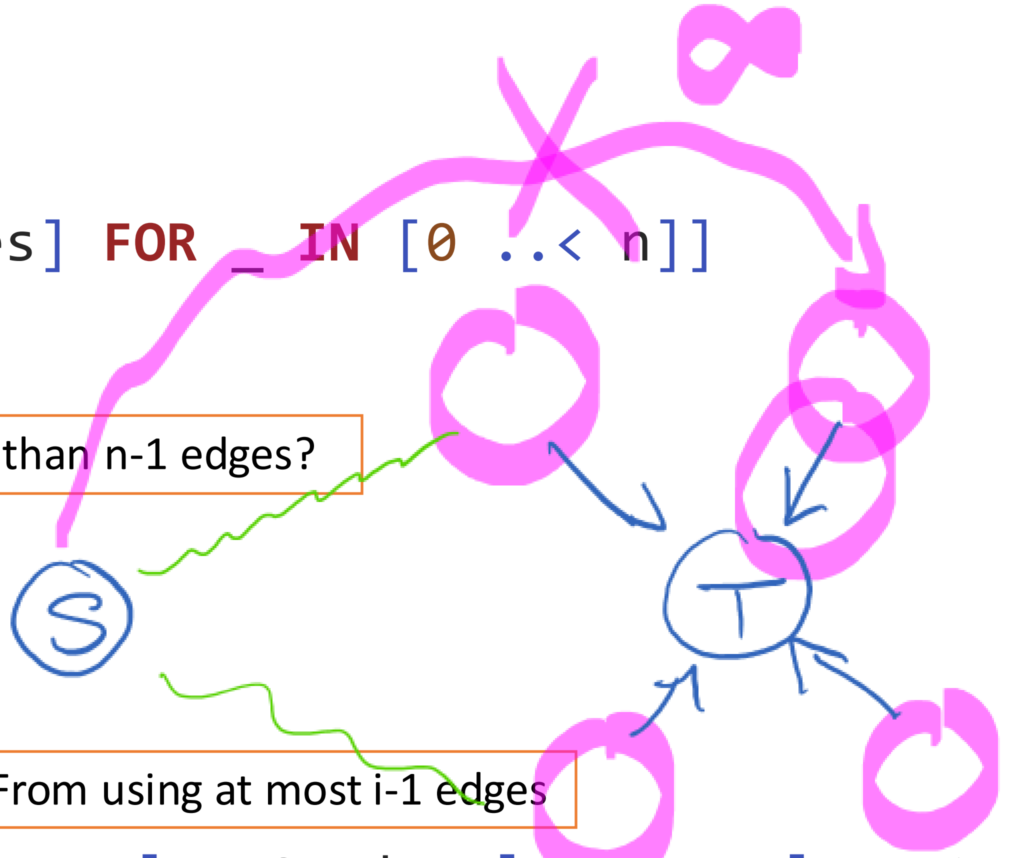
Why won't we need more than n-1 edges?

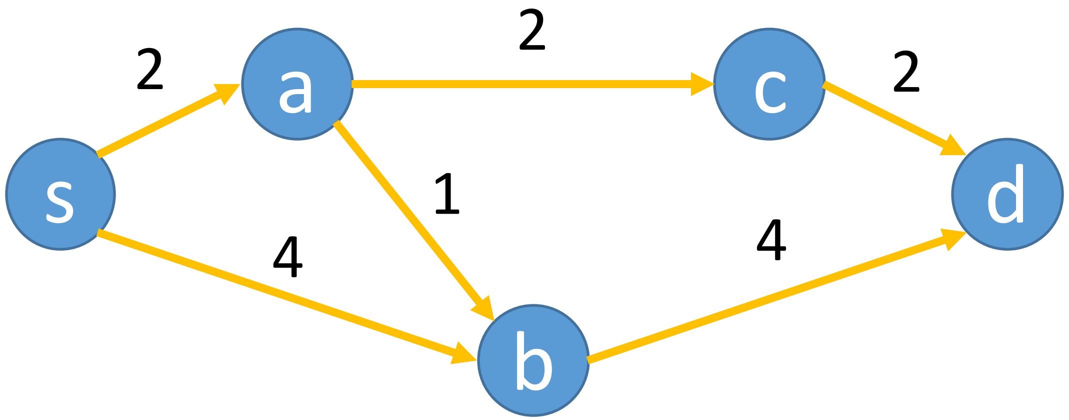
All incoming edges into v

Cost to get to vFrom using at most i-1 edges

Cost using at most num_edges-1 edges

Cost using at most num_edges



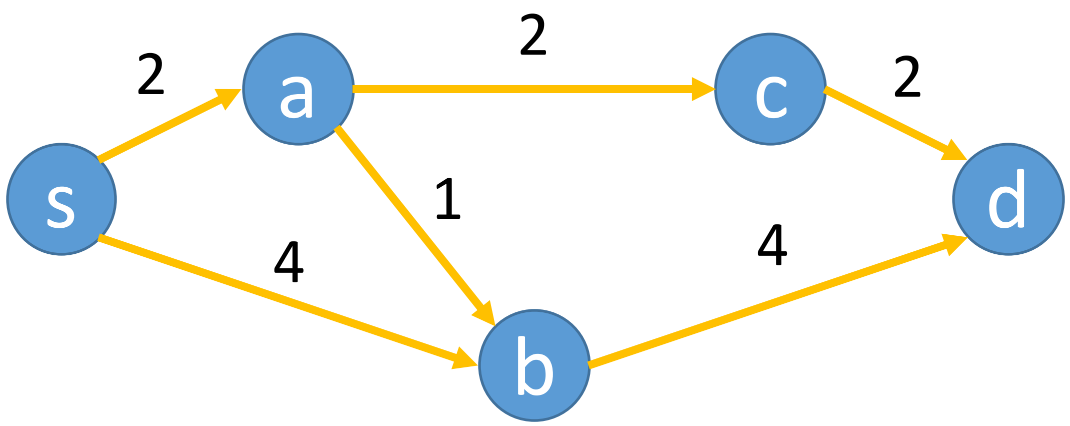



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

Max Number Of Edges On Path

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

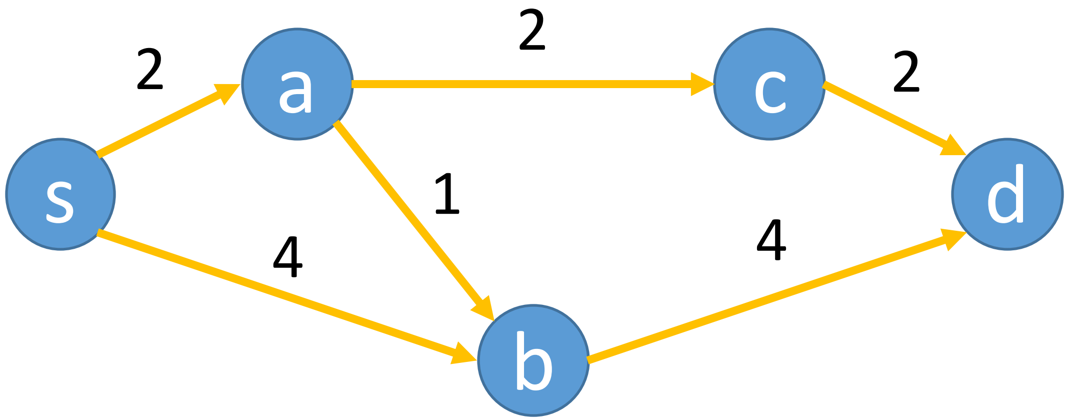
End Vertex




```
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				

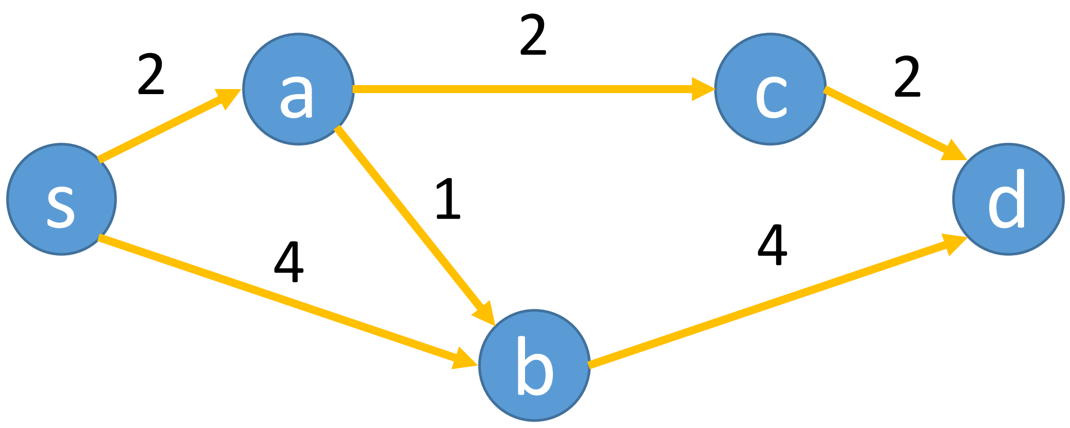



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

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

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

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

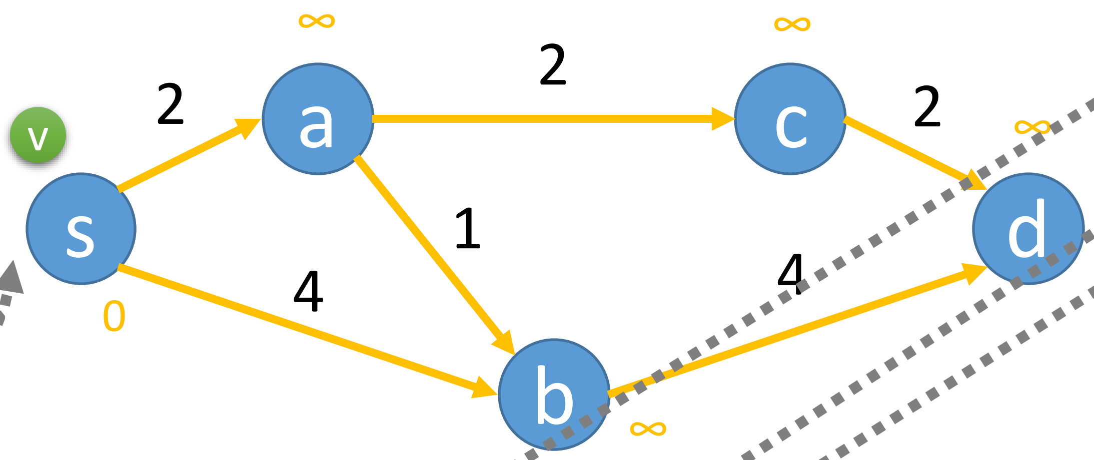



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

```
path_lengths = [[INFINITY FOR v IN G.vertices] FOR _ IN [0 ..< n]]
path_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)
```

skip

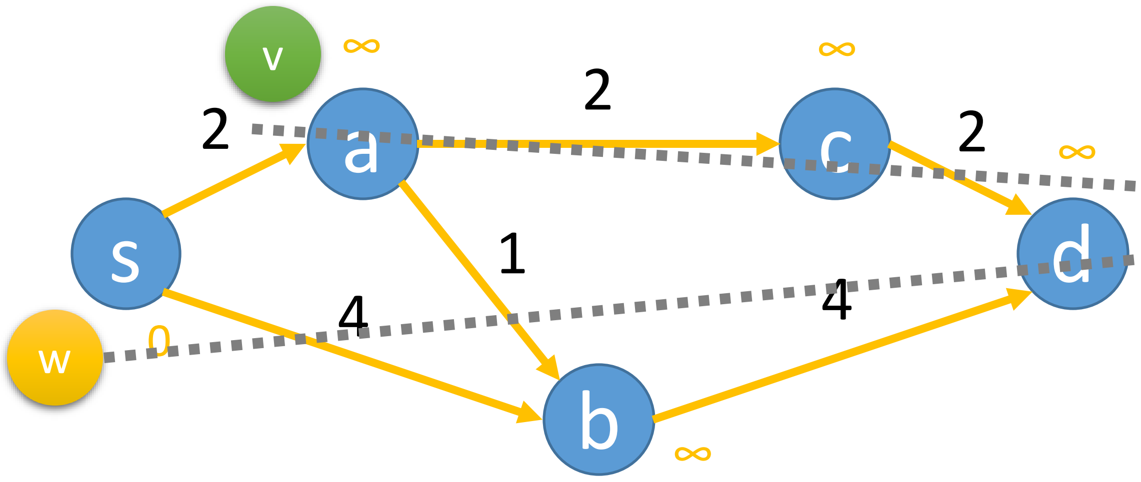
num_edges = 1

v = s

minW = inf

Nothing to loop over

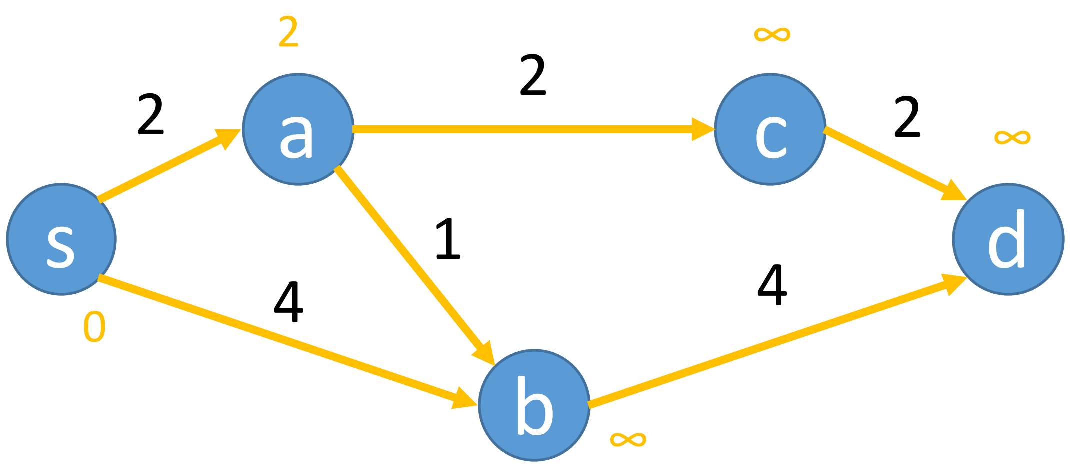
i	4					
	3					
	2					
	1					
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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]
            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

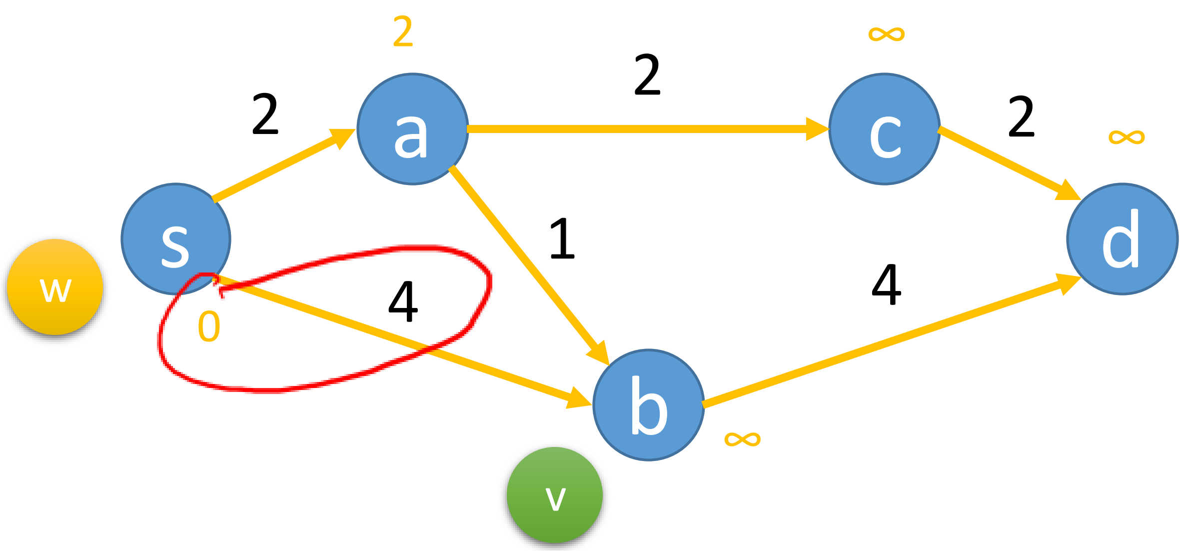
i	4					
	3					
	2					
	1	0				
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

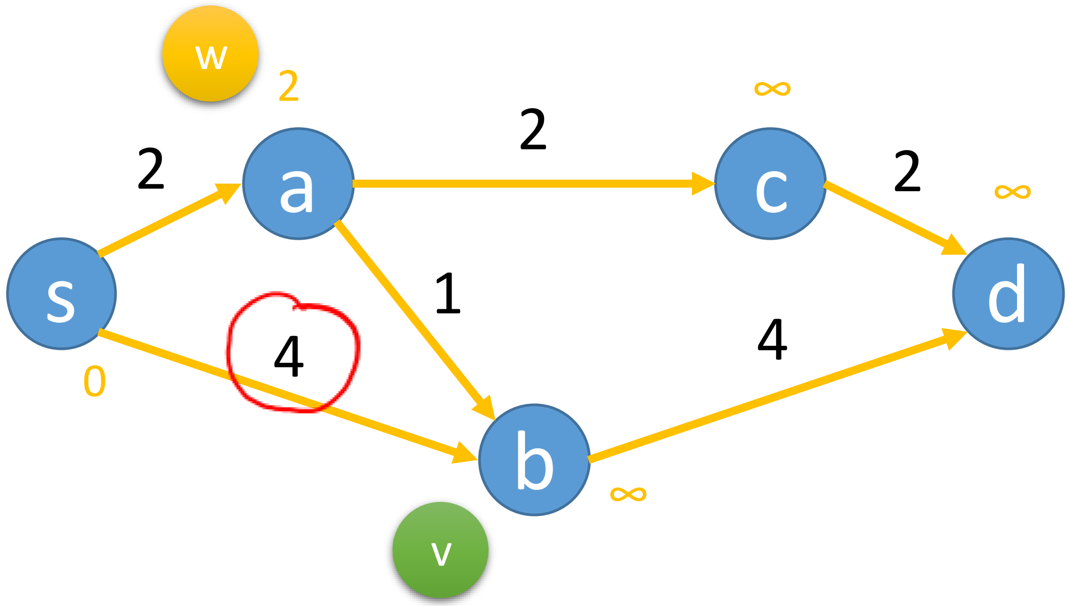
i	4					
	3					
	2					
	1	0	2			
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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				

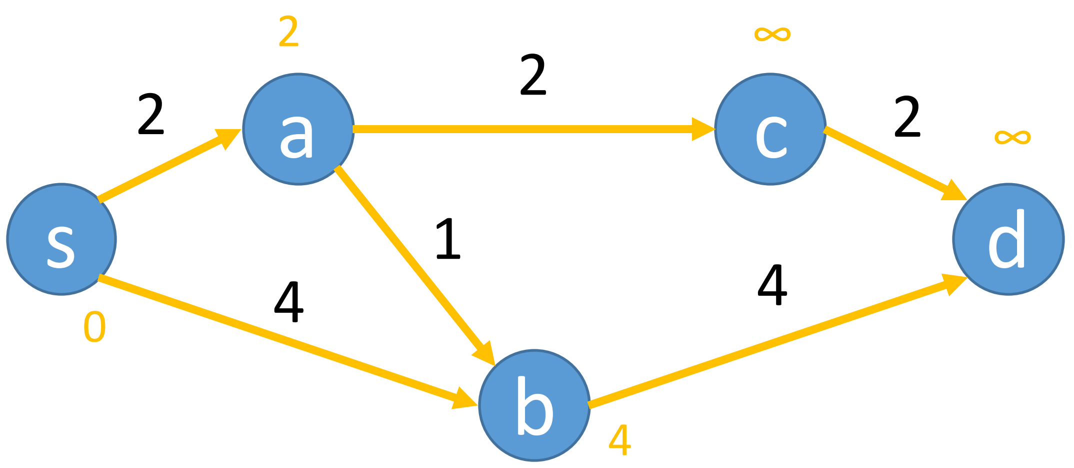



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

num_edges = 1
v = b
minW = inf
minW = 4

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

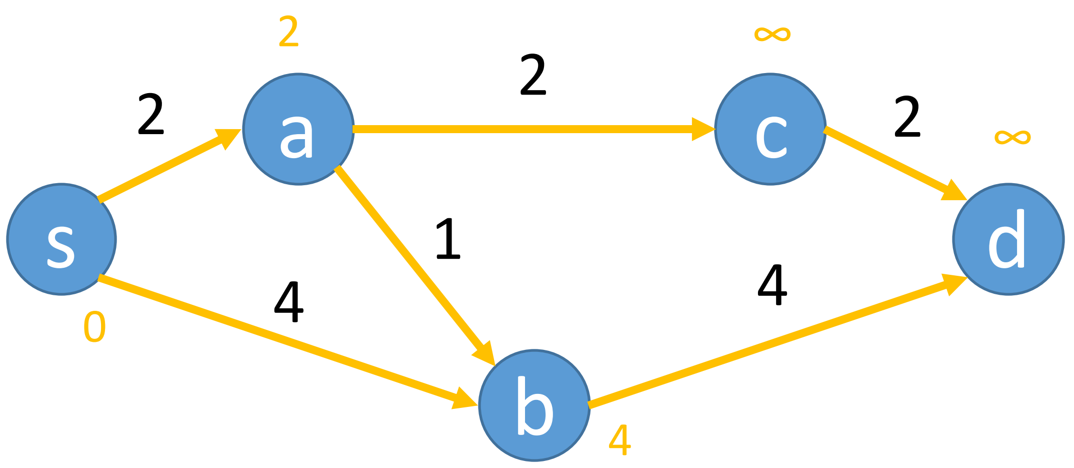
(1) 3
(2) ∞




```
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)
```

num_edges = 1
v = b
minW = inf
minW = 4

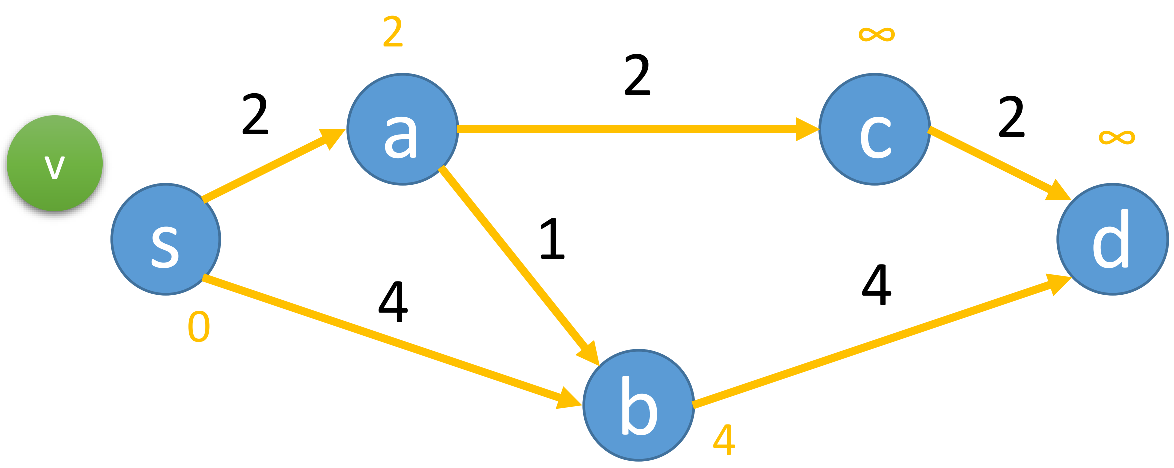
i	4					
	3					
	2					
	1	0	2	4		
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

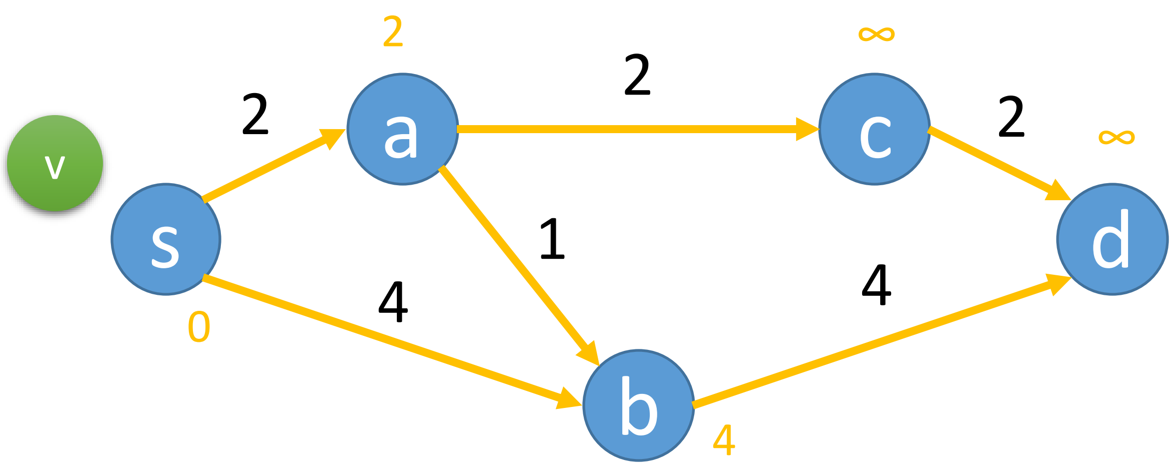
i	4					
	3					
	2					
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

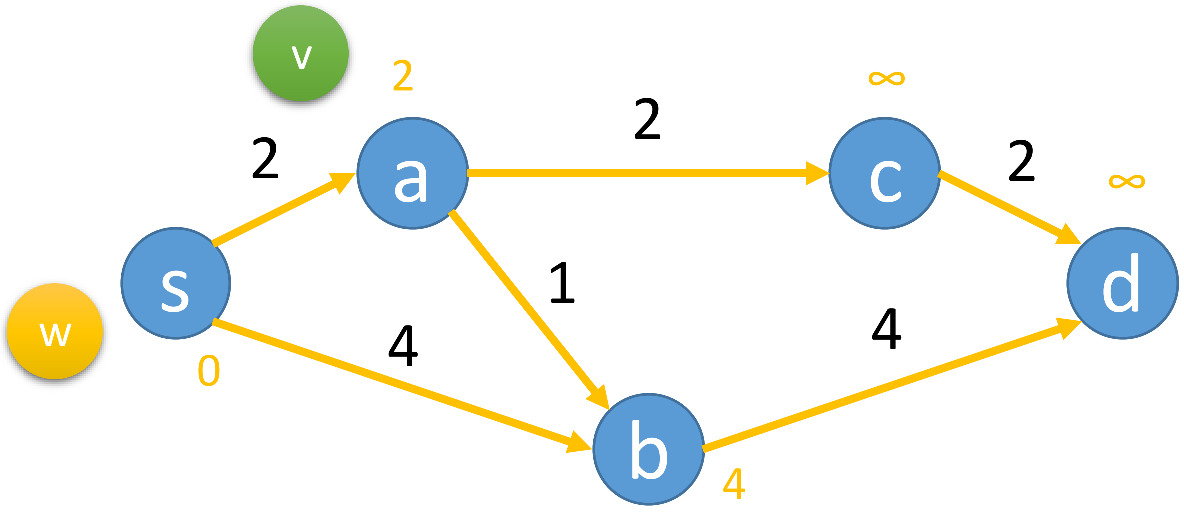
i	4					
	3					
	2					
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

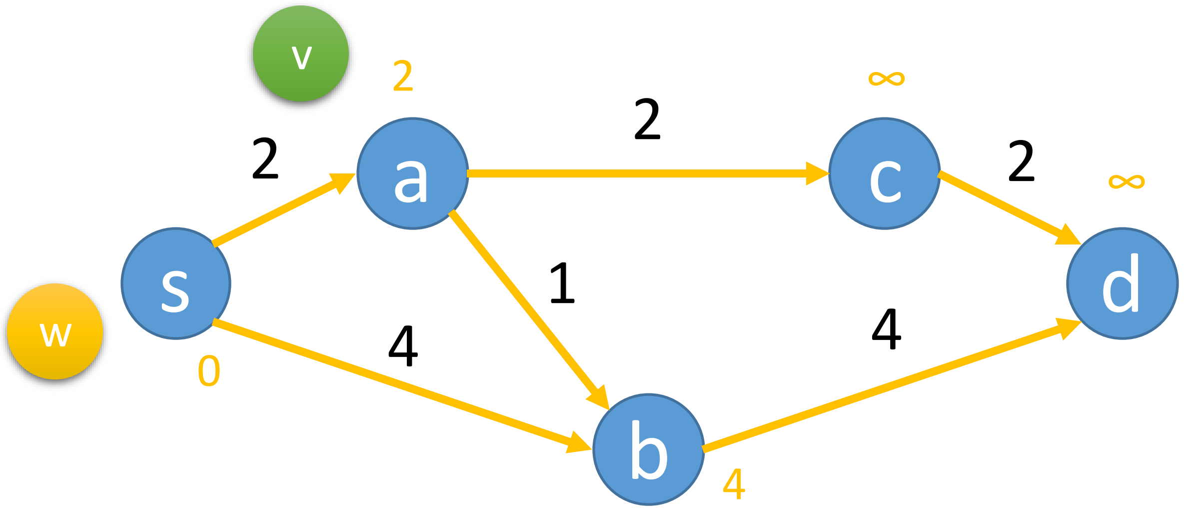
i	4					
	3					
	2	0				
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

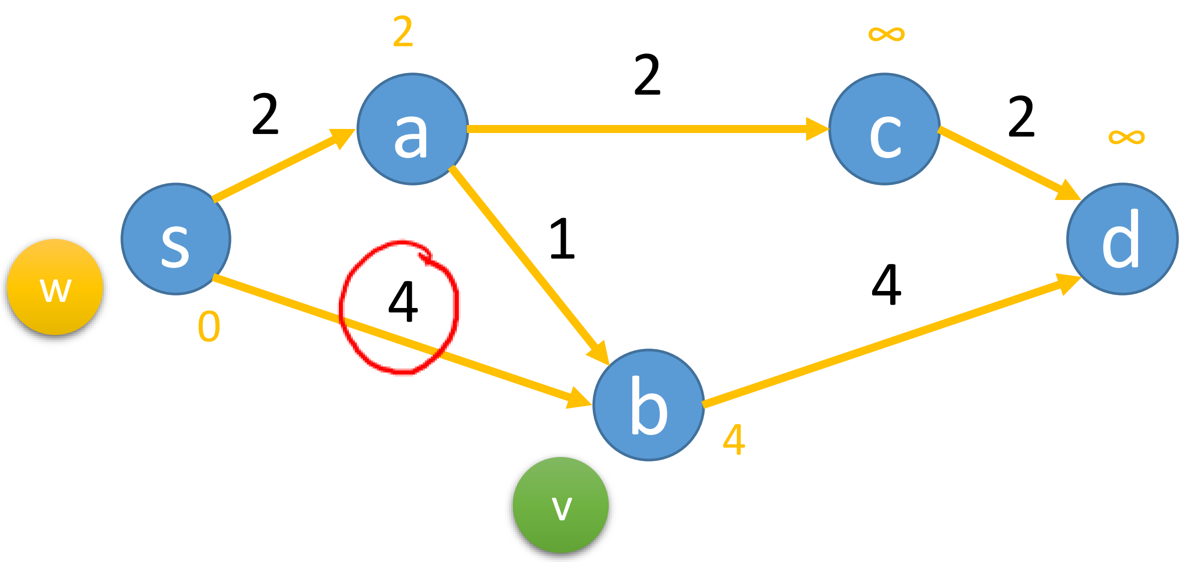
i	4					
	3					
	2	0				
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

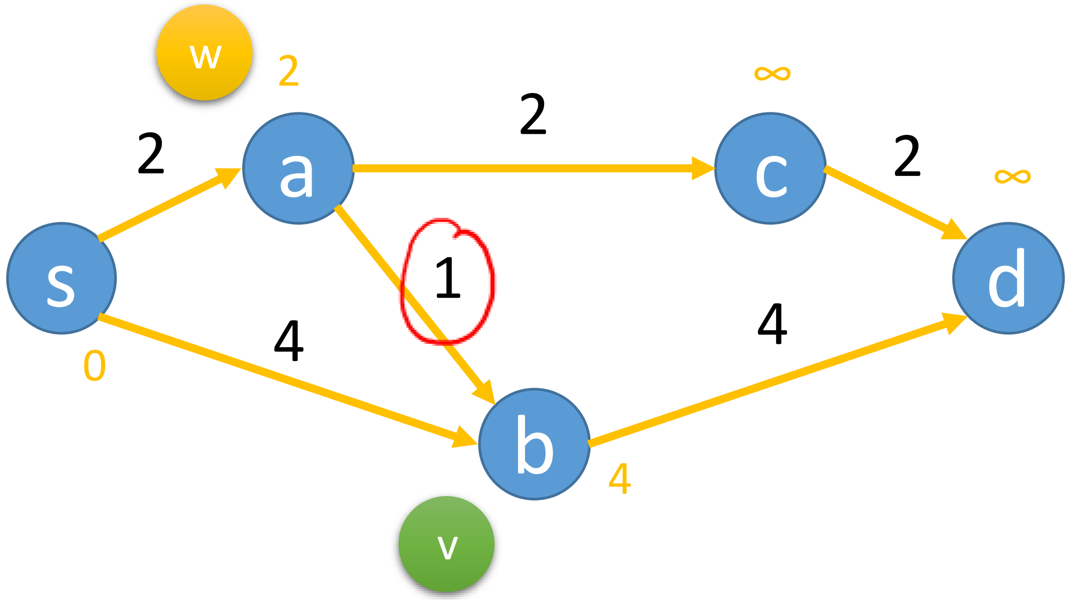
i	4					
	3					
	2	0	2			
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

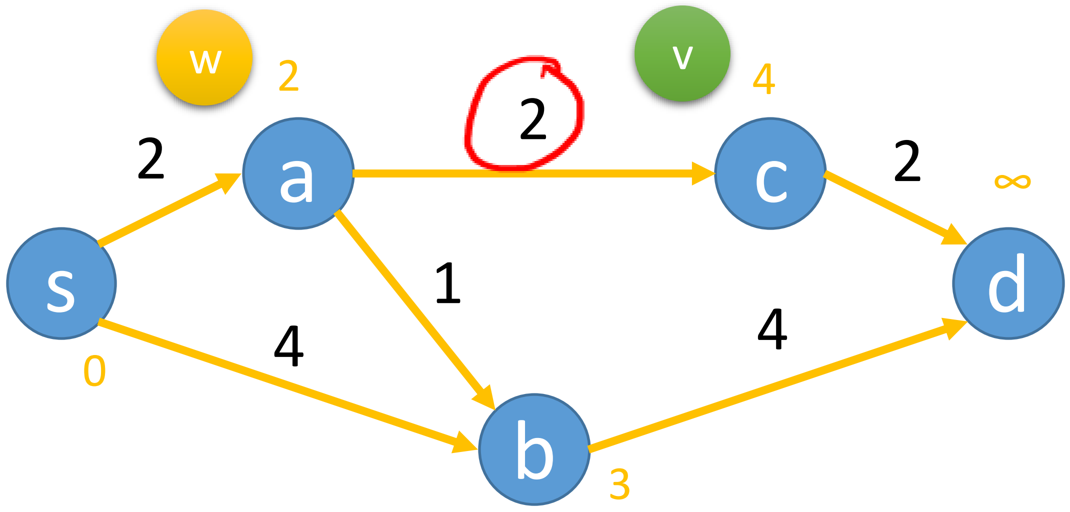
i	4					
	3					
	2	0	2			
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

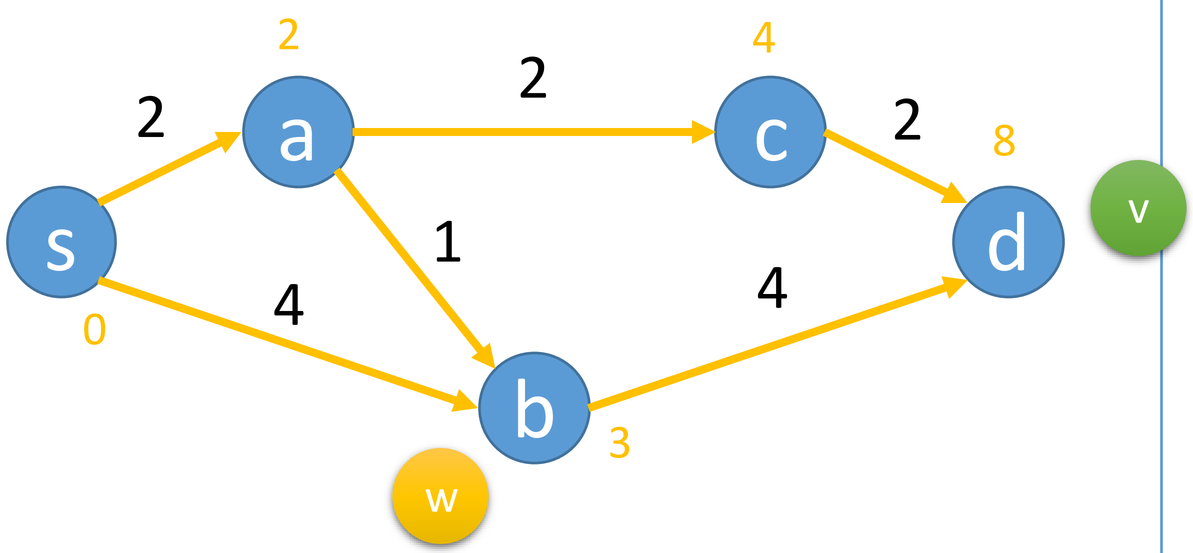
i	4					
	3					
	2	0	2			
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

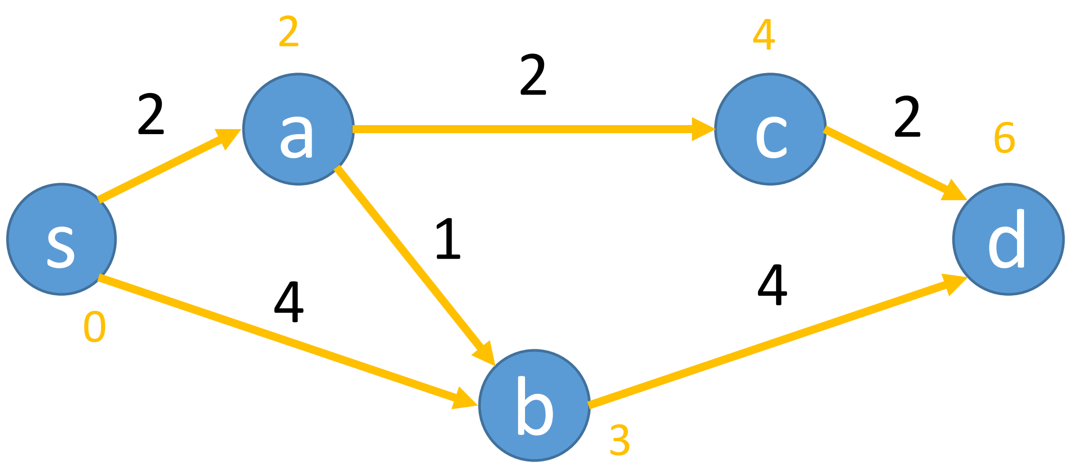
i	4					
	3					
	2	0	2	3	4	
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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

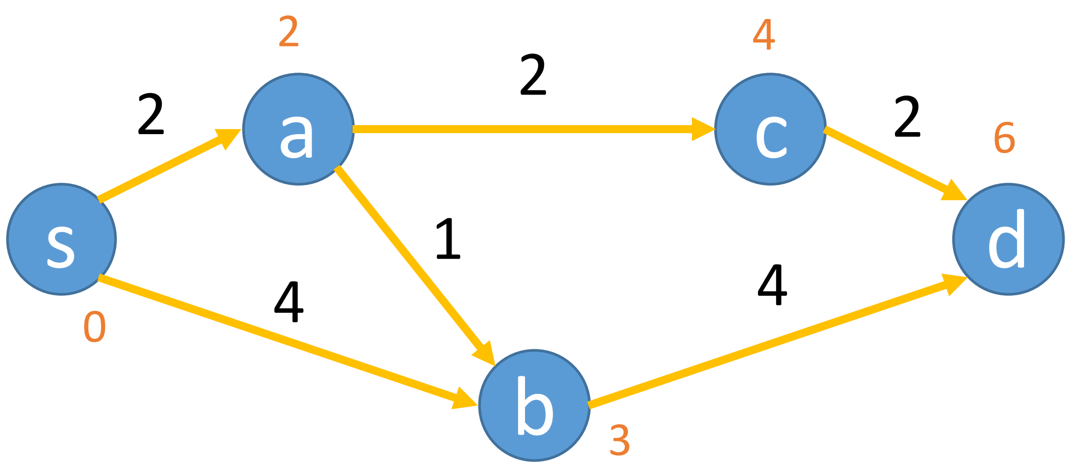
i	4					
	3					
	2	0	2	3	4	8
	1	0	2	4	∞	∞
	0	0	∞	∞	∞	∞
		s	a	b	c	d
		v				




```
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				

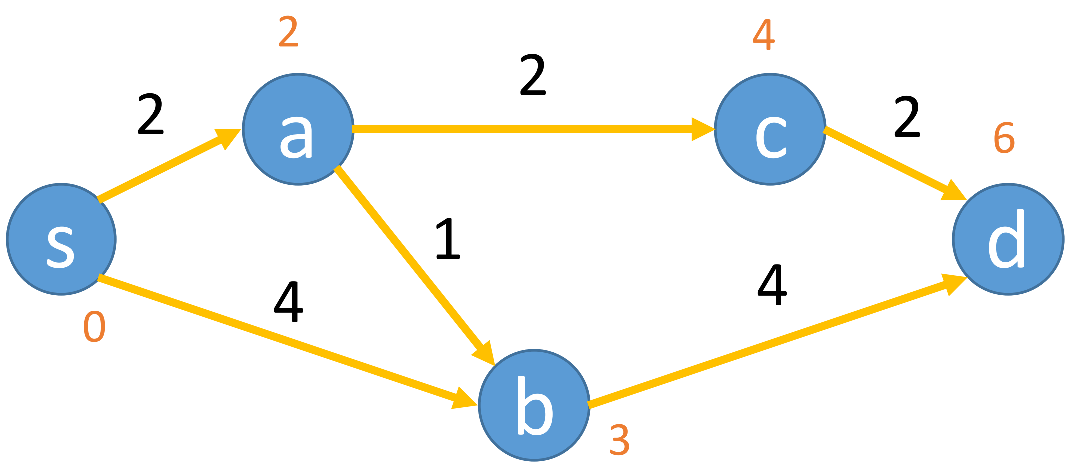



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

What is our output?

We output the shortest paths from s to all other vertices

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				

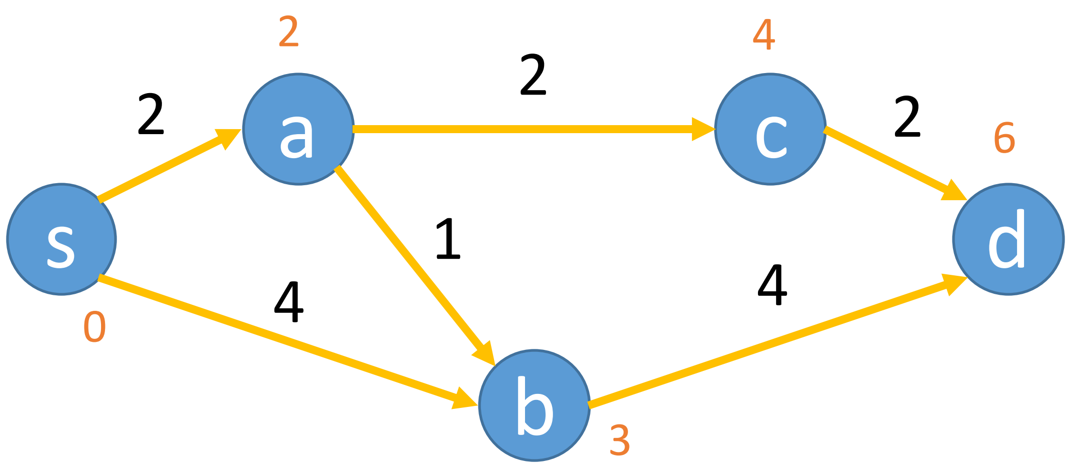



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

What is our output?

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				




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

$O(n^2)$

$O(n)$

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)
```

O(n)

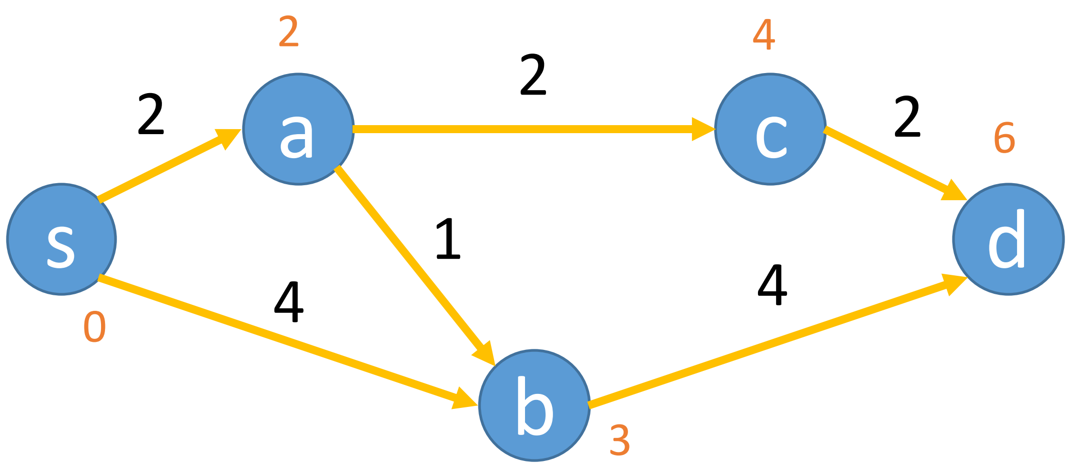
O(m)

not all edges for every v, but every edge just ordered by v

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

$O(n^2) < O(mn) < O(n^3)$ $\qquad O(m^2)$

In general

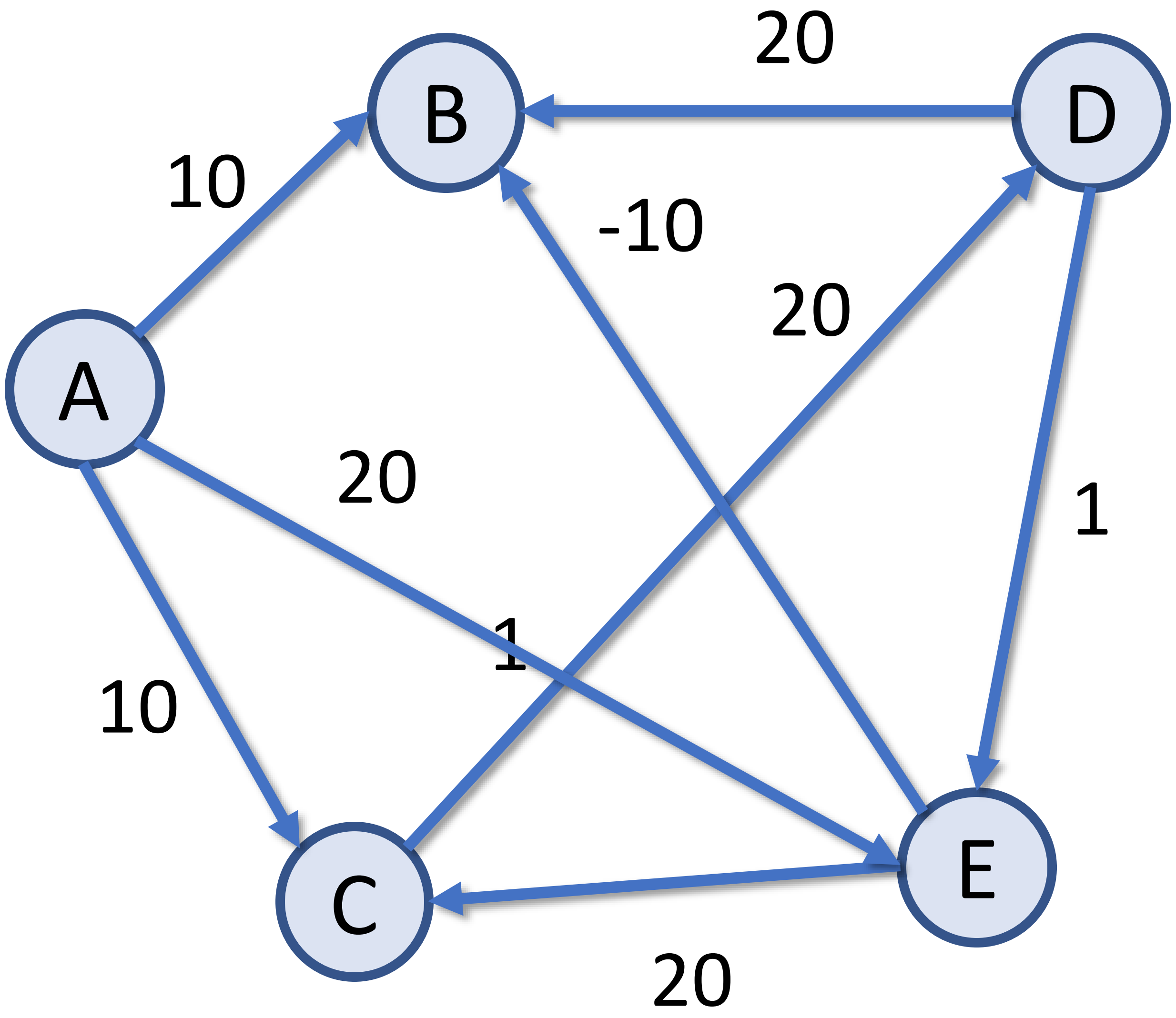



```
FOR num_edges IN [1 ..< n]
    FOR v IN G.vertices
        min_len = INFINITY
        FOR (vFrom, v) IN G.edges
            len = lens[num_edges - 1, vFrom] + c
            IF len < min_len
                min_len = len
        lens[num_edges, v] = min(
            lens[num_edges - 1, v], min_len)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

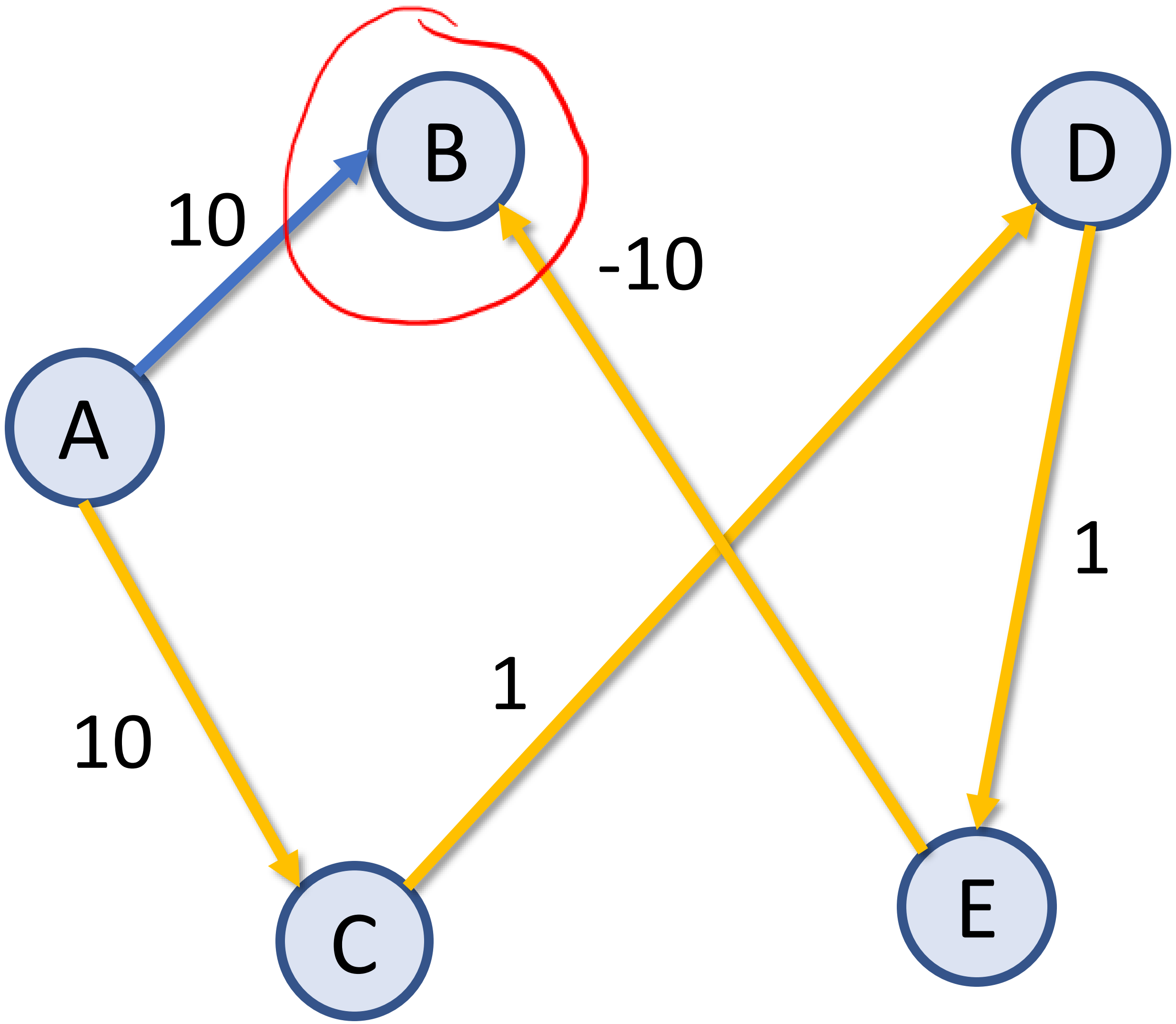
What about negative edges?

Yes it works
because BF conciders all paths

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				



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

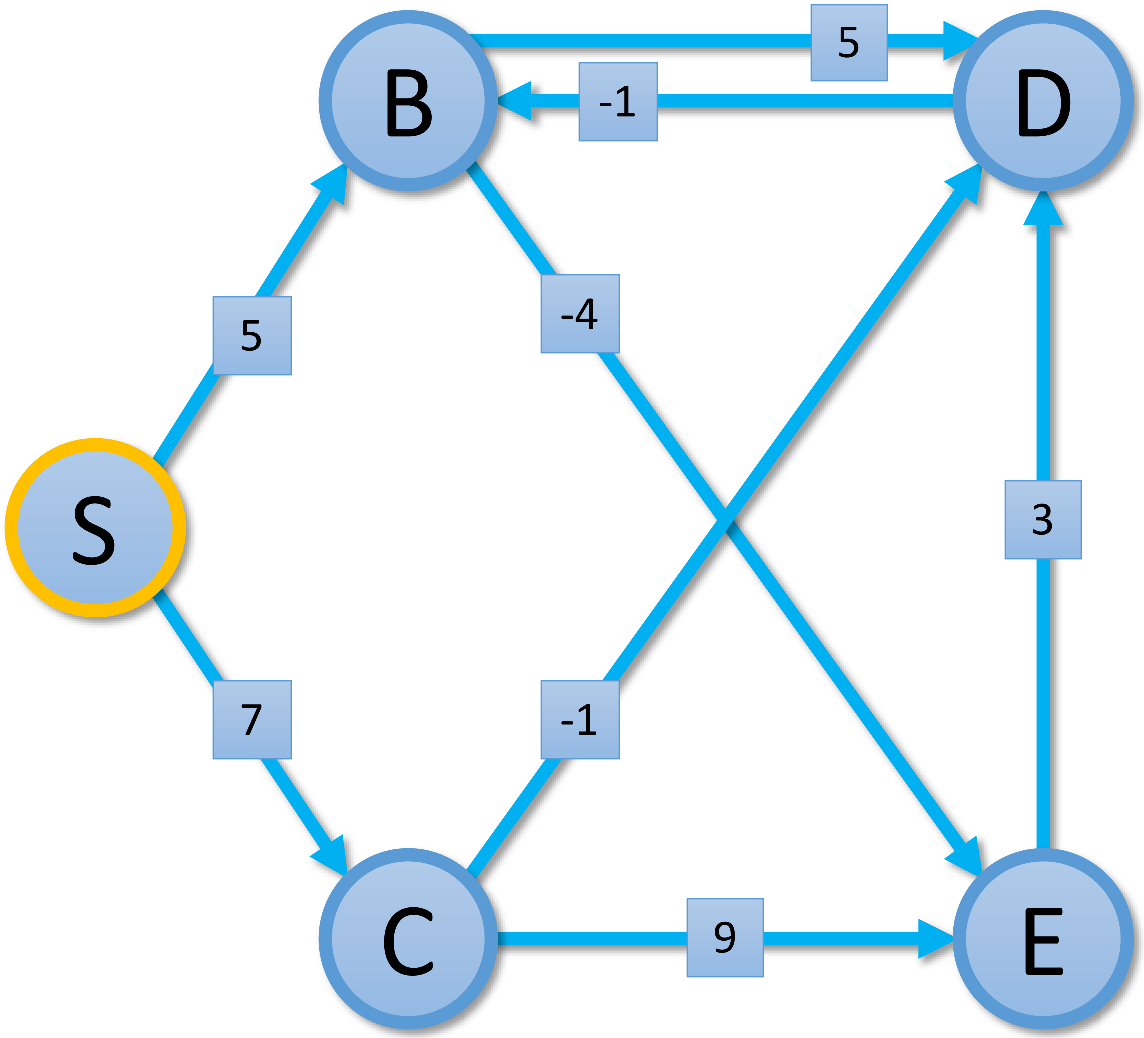


What is the maximum number of edges on any real (not negative infinity) shortest path?

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

Exercise

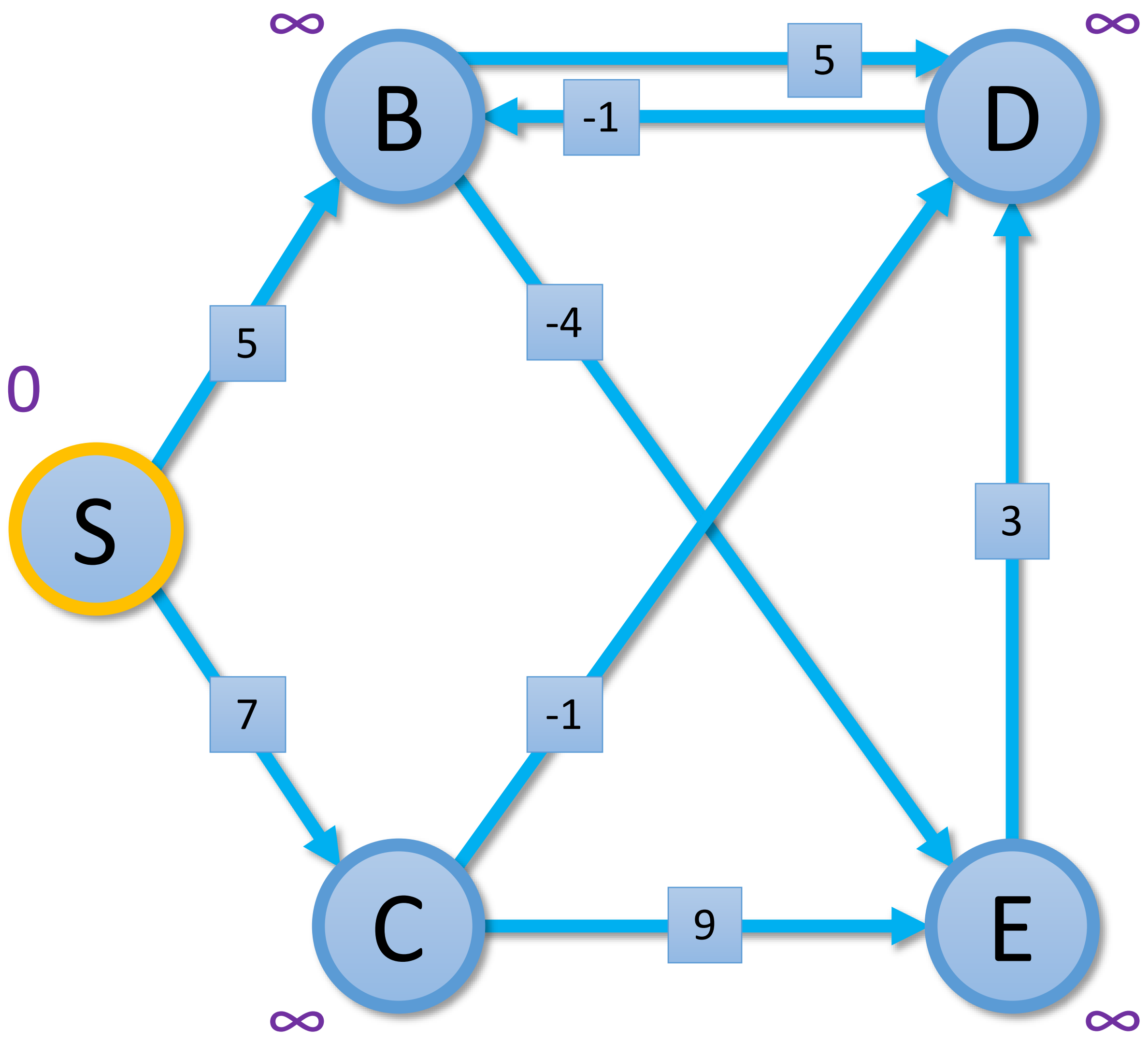
What is the shortest path from S to B?



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)

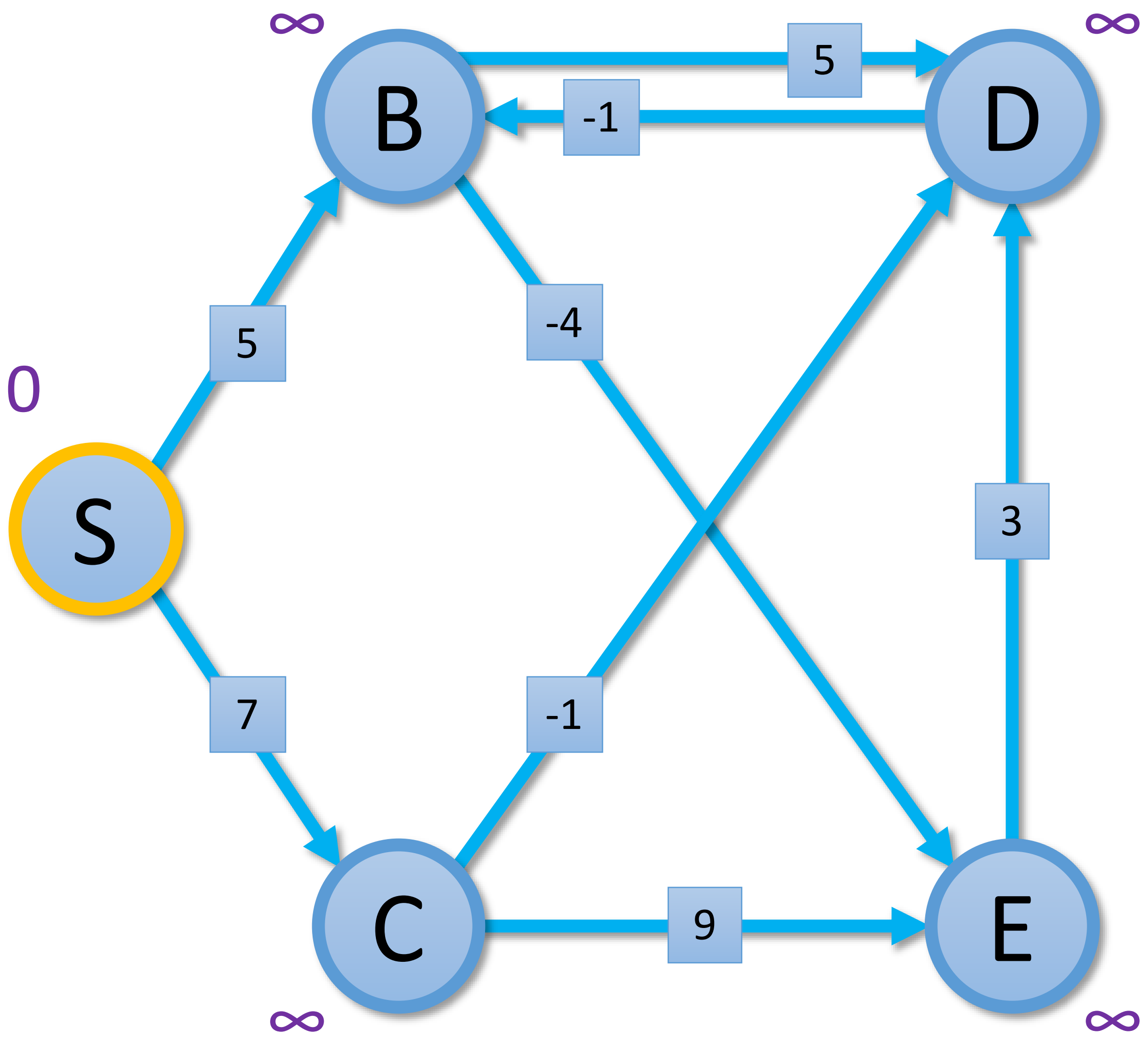


What is the shortest path from S to B?

Initialization

Vertex	Predecessor	i – 1	i
S	S	0	
B	None	∞	
C	None	∞	
D	None	∞	
E	None	∞	

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

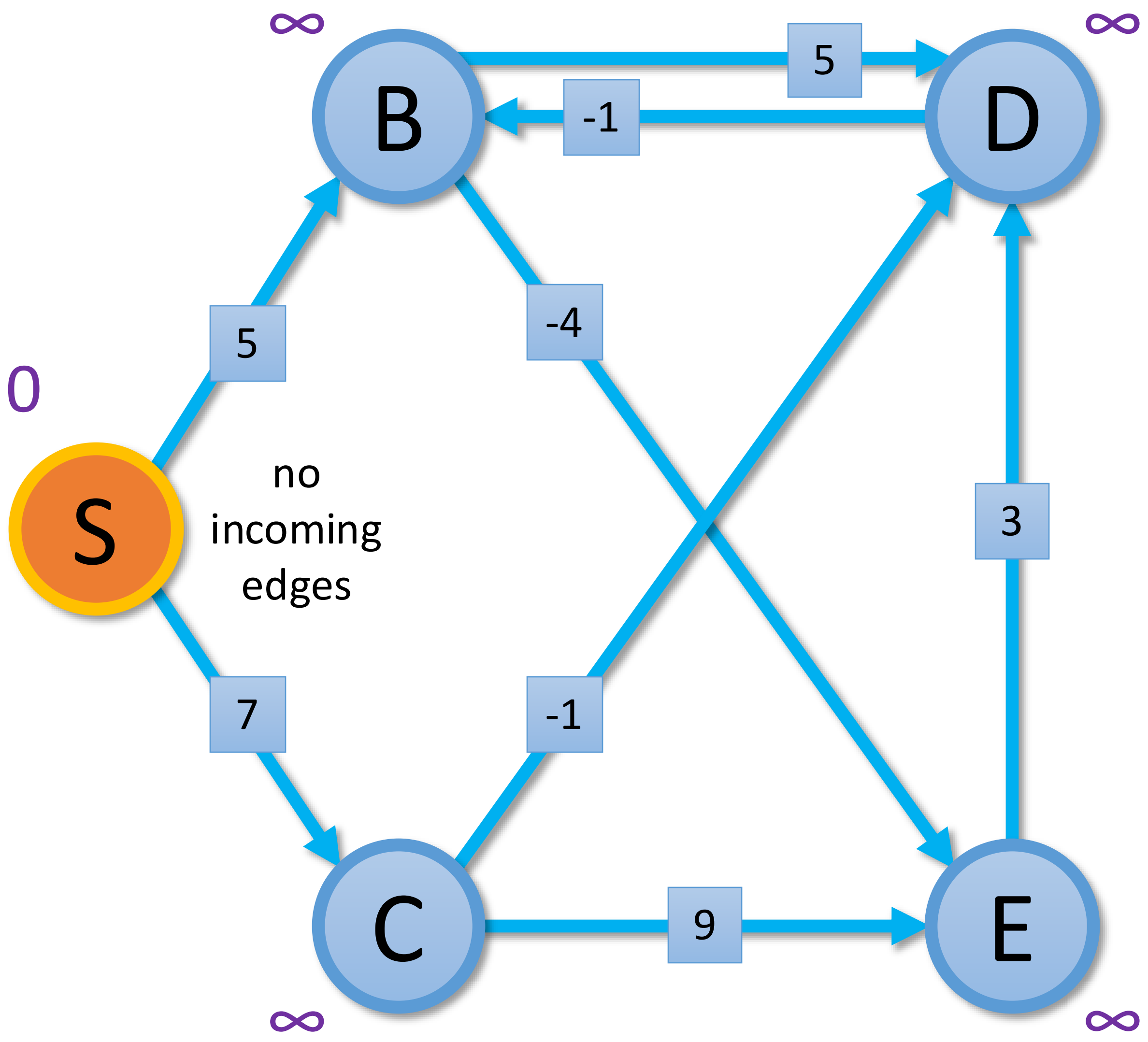


What is the shortest path from S to B?

i = 1

Vertex	Predecessor	i – 1	i
S	S	0	
B	None	∞	
C	None	∞	
D	None	∞	
E	None	∞	

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

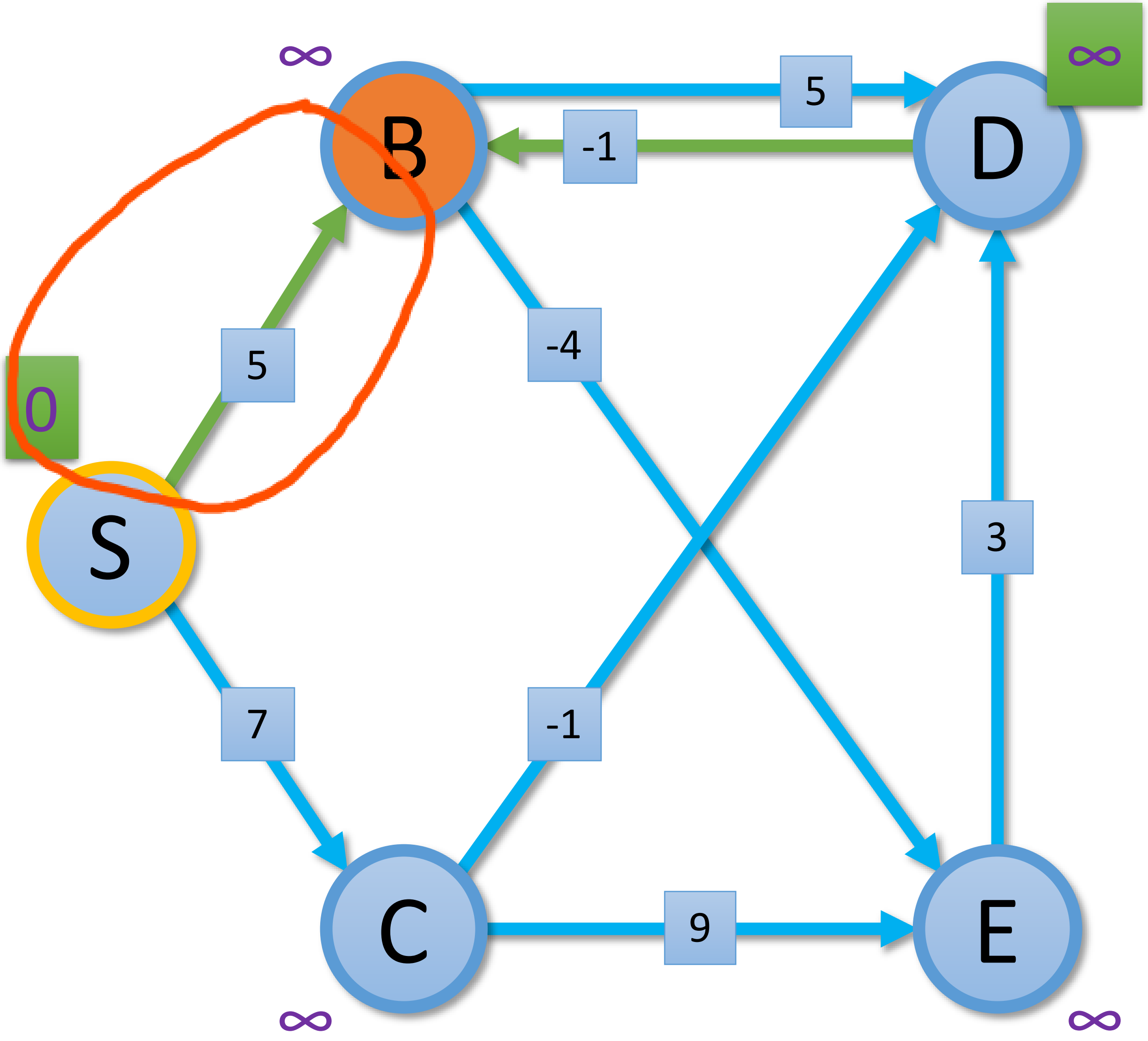


What is the shortest path from S to B?

i = 1

Vertex	Predecessor	i – 1	i
S	S	0	0
B	None	∞	
C	None	∞	
D	None	∞	
E	None	∞	

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

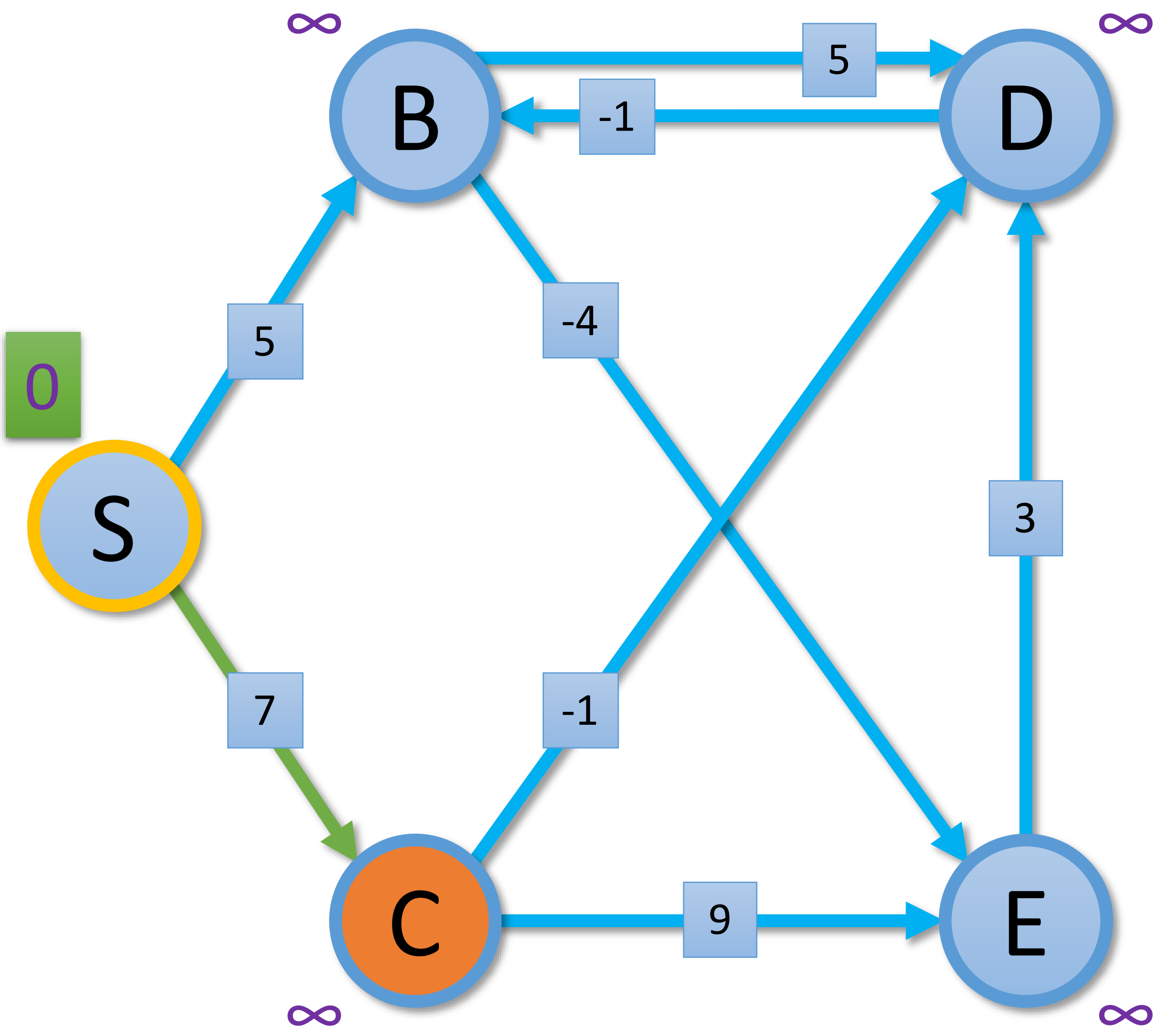


What is the shortest path from S to B?

i = 1

Vertex	Predecessor	i – 1	i
S	S	0	0
B	S	∞	5
C	None	∞	
D	None	∞	
E	None	∞	

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

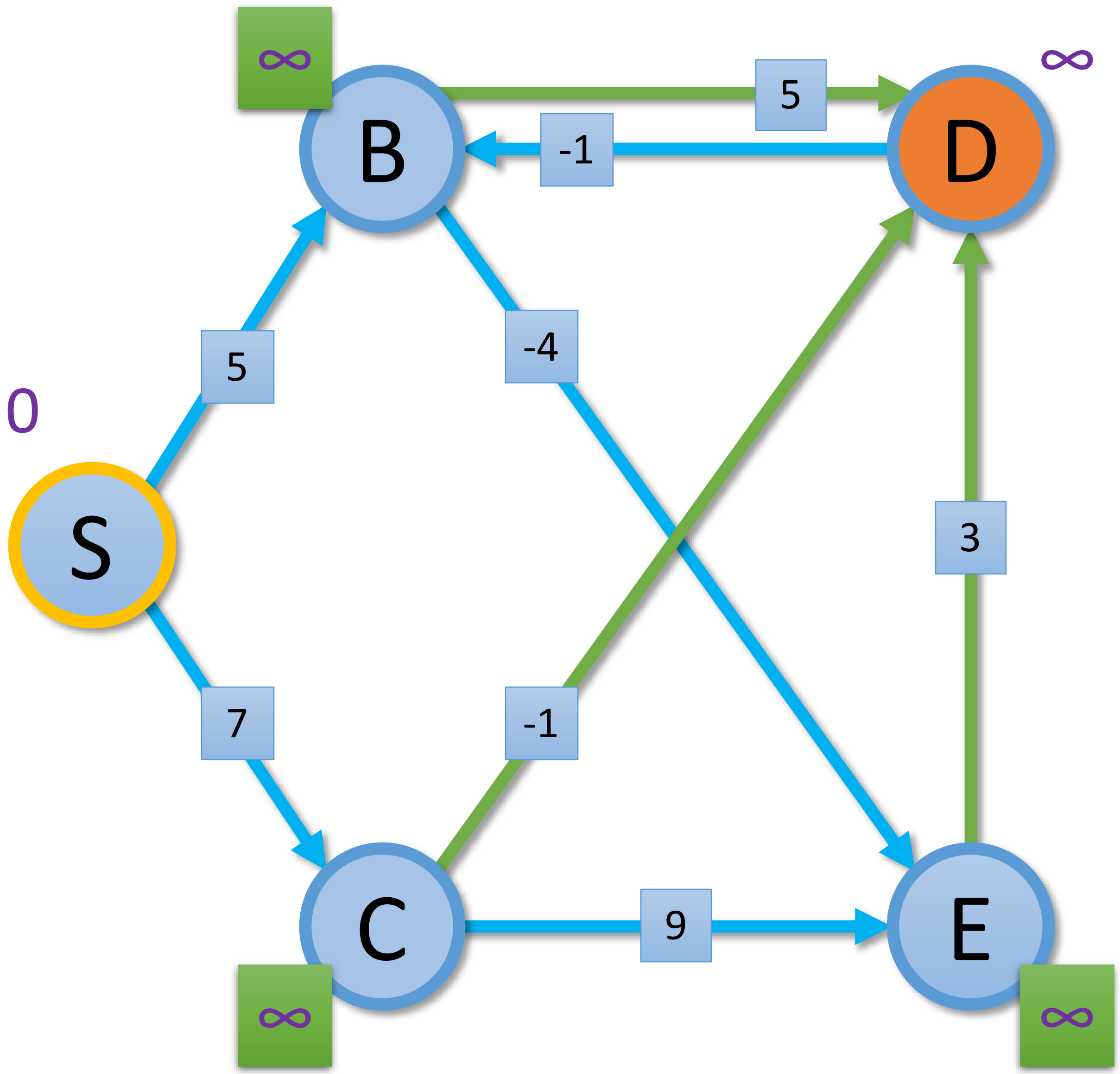


What is the shortest path from S to B?

i = 1

Vertex	Predecessor	i – 1	i
S	S	0	0
B	S	∞	5
C	S	∞	7
D	None	∞	
E	None	∞	

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

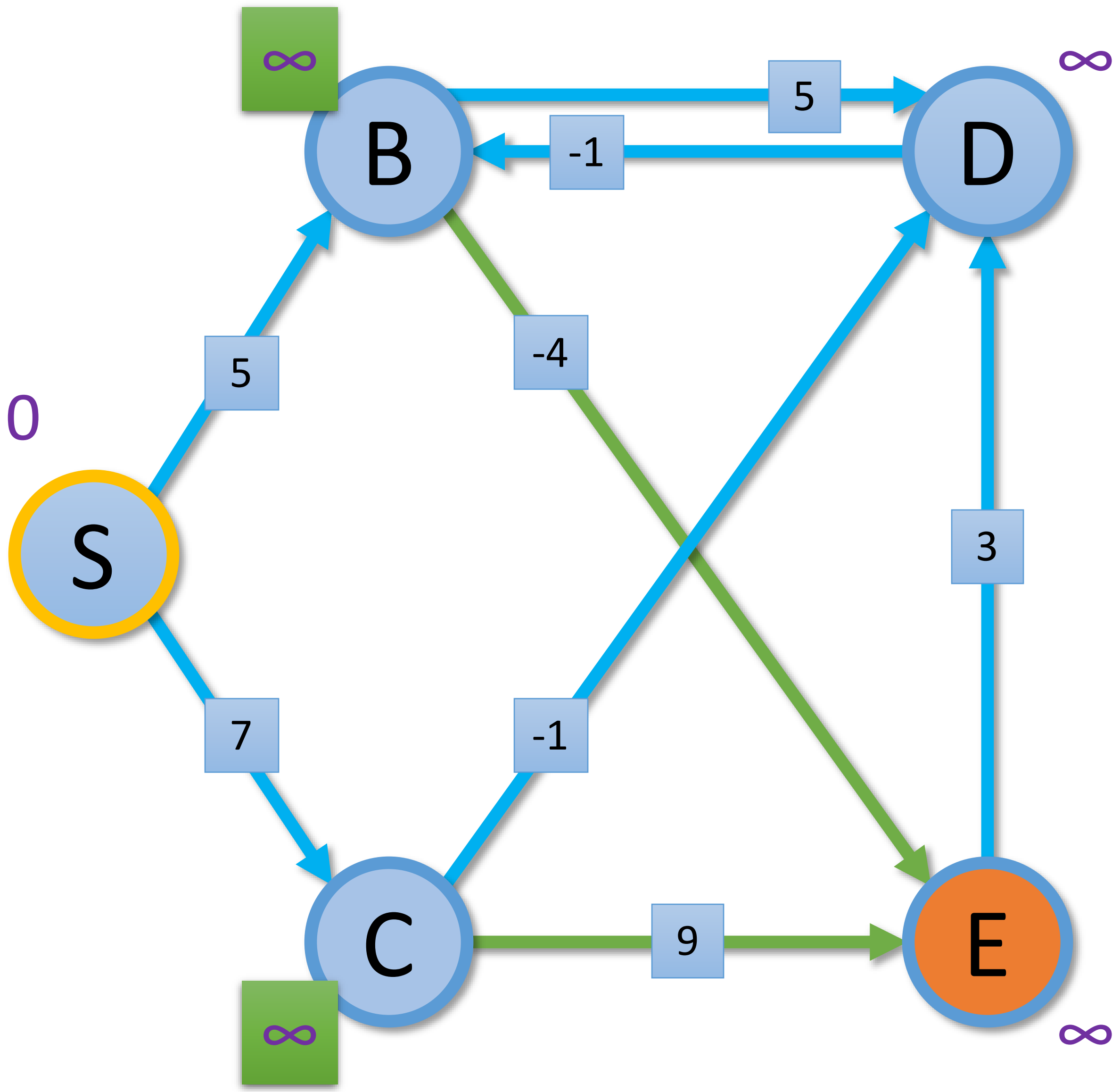


What is the shortest path from S to B?

i = 1

Vertex	Predecessor	i – 1	i
S	S	0	0
B	S	∞	5
C	S	∞	7
D	None	∞	∞
E	None	∞	

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

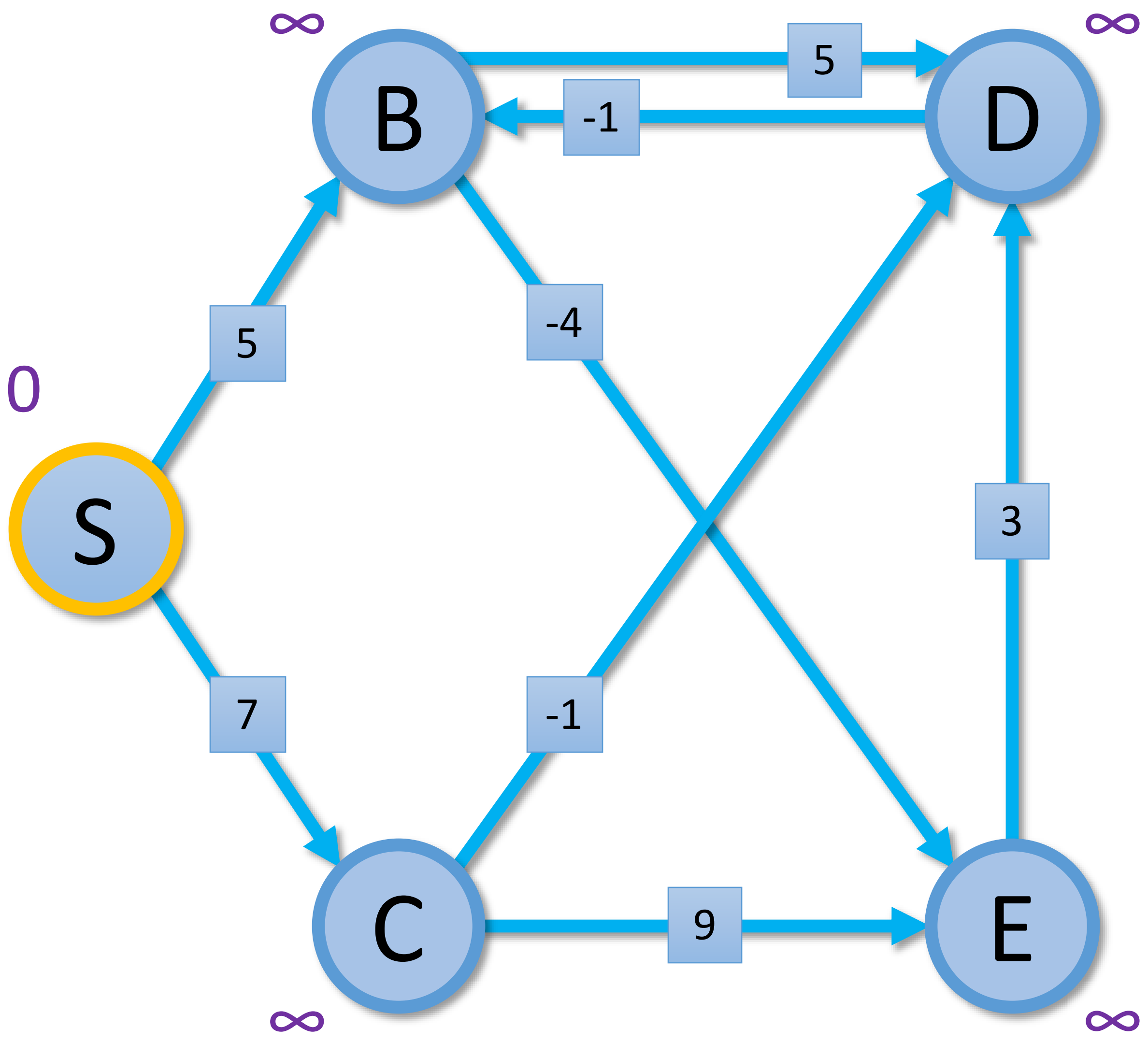


What is the shortest path from S to B?

i = 1

Vertex	Predecessor	i – 1	i
S	S	0	0
B	S	∞	5
C	S	∞	7
D	None	∞	∞
E	None	∞	∞

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

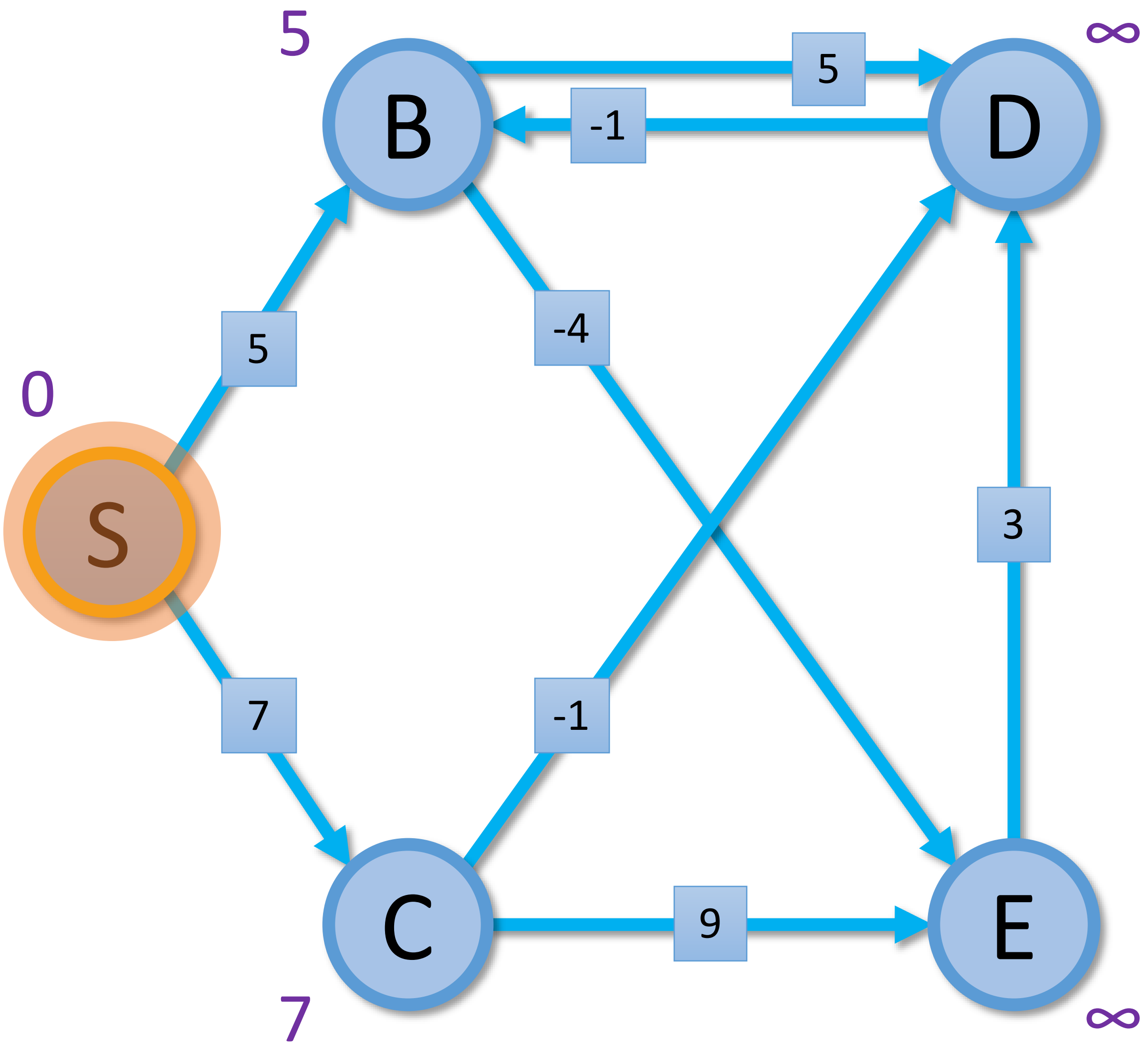


What is the shortest path from S to B?

i = 2

Vertex	Predecessor	i – 1	i
S	S	0 ←	0
B	S	∞ ←	5
C	S	∞ ←	7
D	None	∞ ←	∞
E	None	∞ ←	∞

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

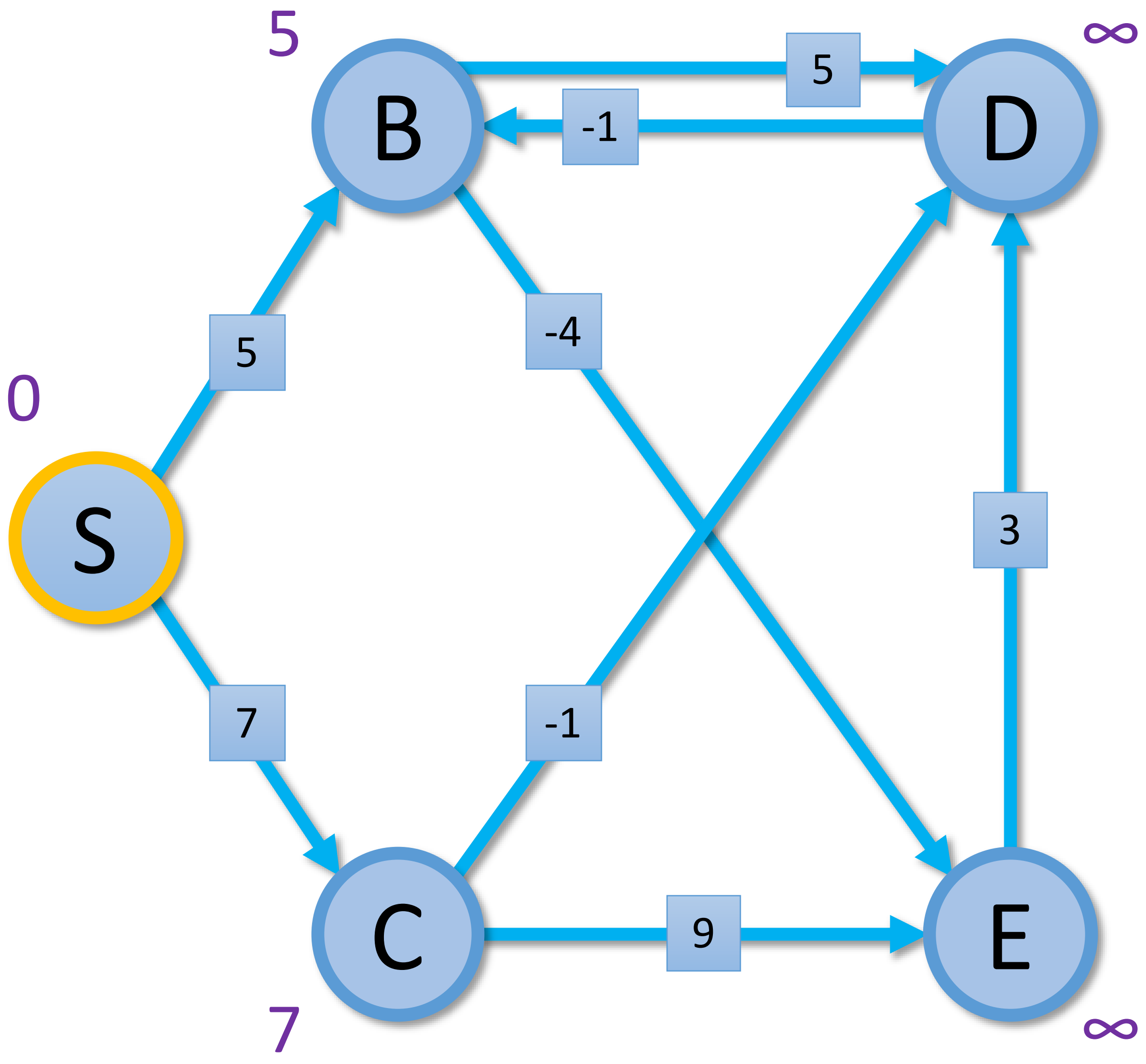


What is the shortest path from S to B?

i = 2

Vertex	Predecessor	i – 1	i
S	S	0	0
B	S	5	5
C	S	7	7
D	C	∞	6
E	B	∞	1

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

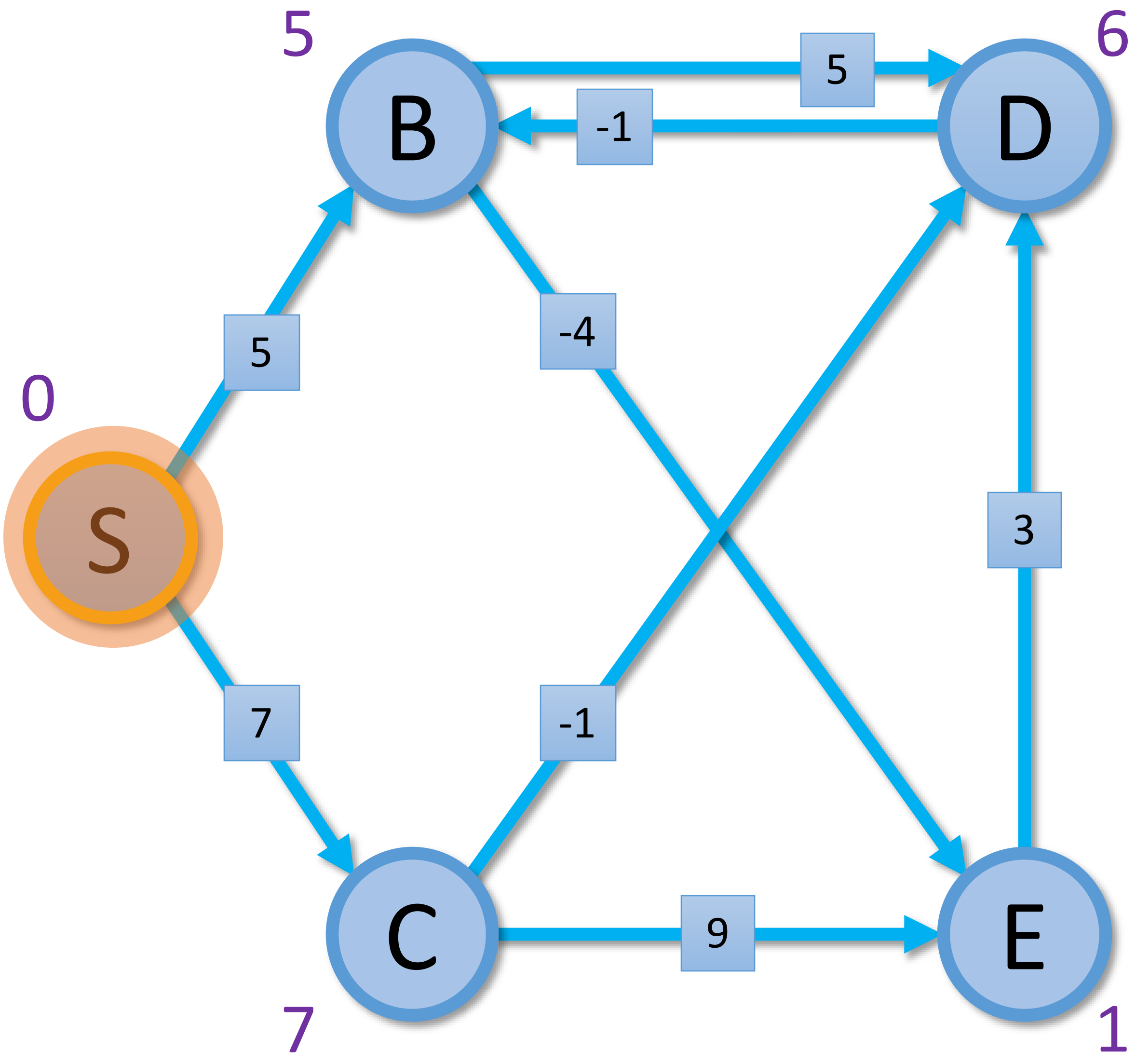


What is the shortest path from S to B?

i = 3

Vertex	Predecessor	i – 1	i
S	S	0	0
B	S	5	5
C	S	7	7
D	C	∞	6
E	B	∞	1

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

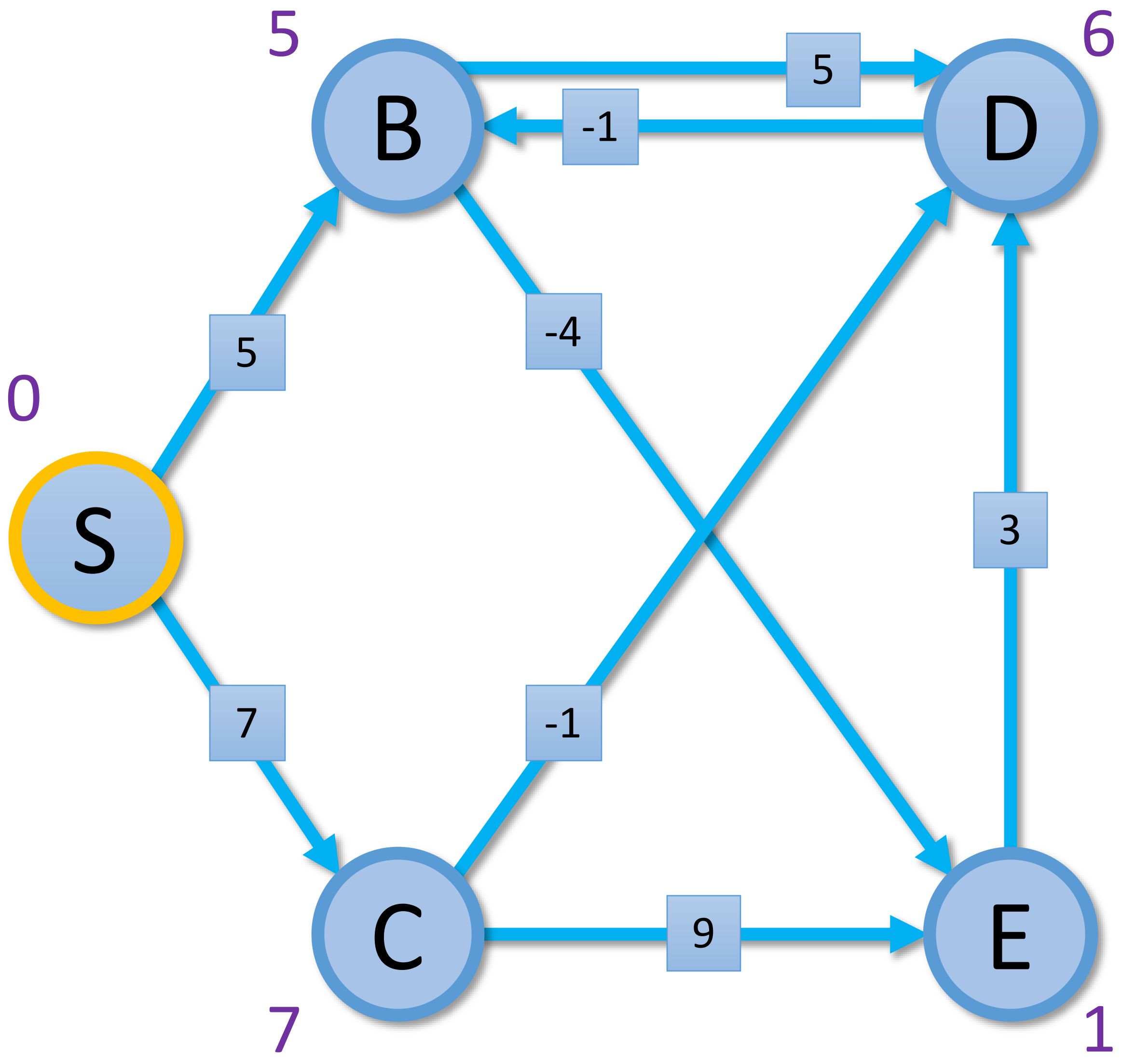


What is the shortest path from S to B?

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)

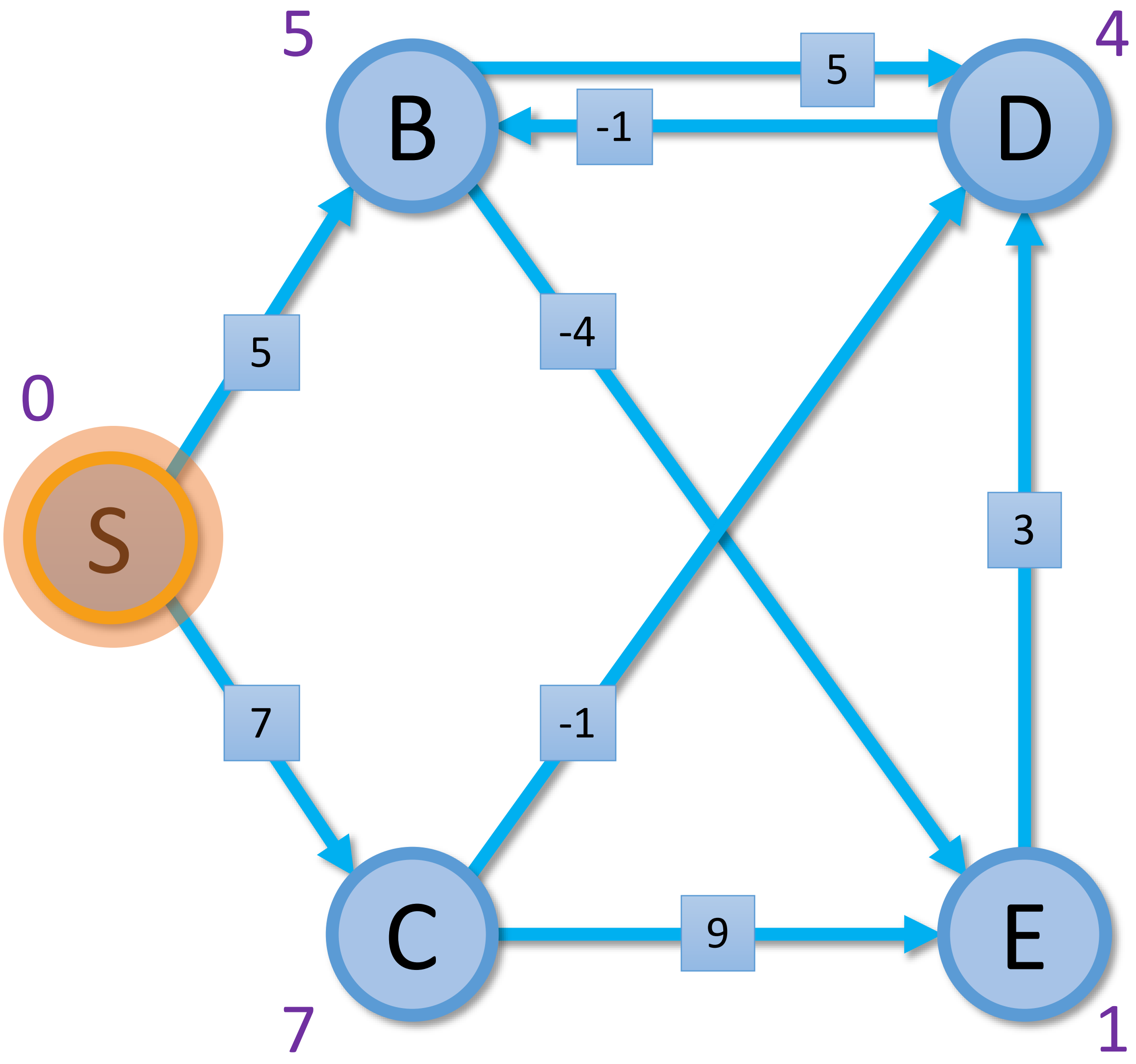


What is the shortest path from S to B?

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)

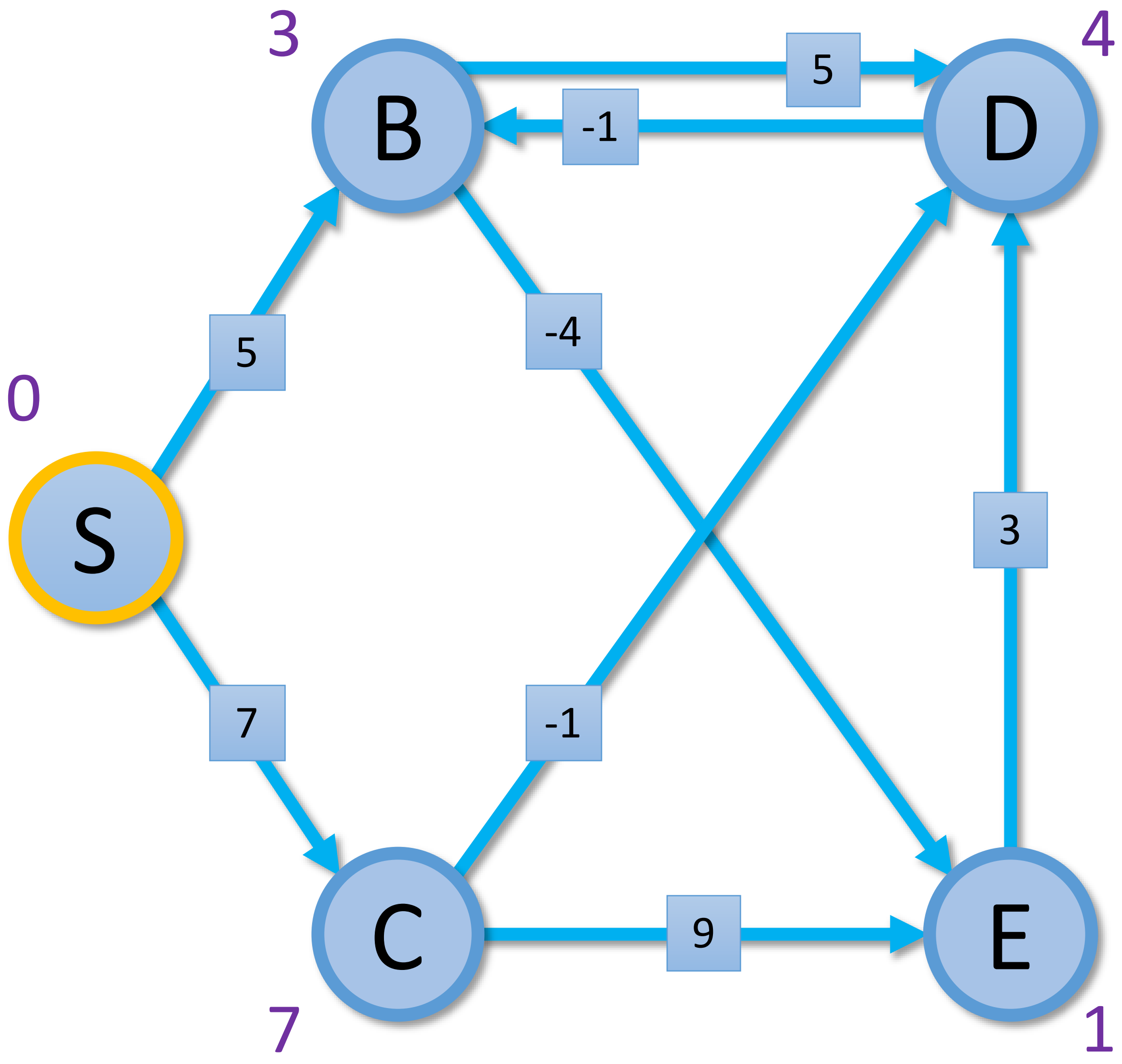


What is the shortest path from S to B?

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)



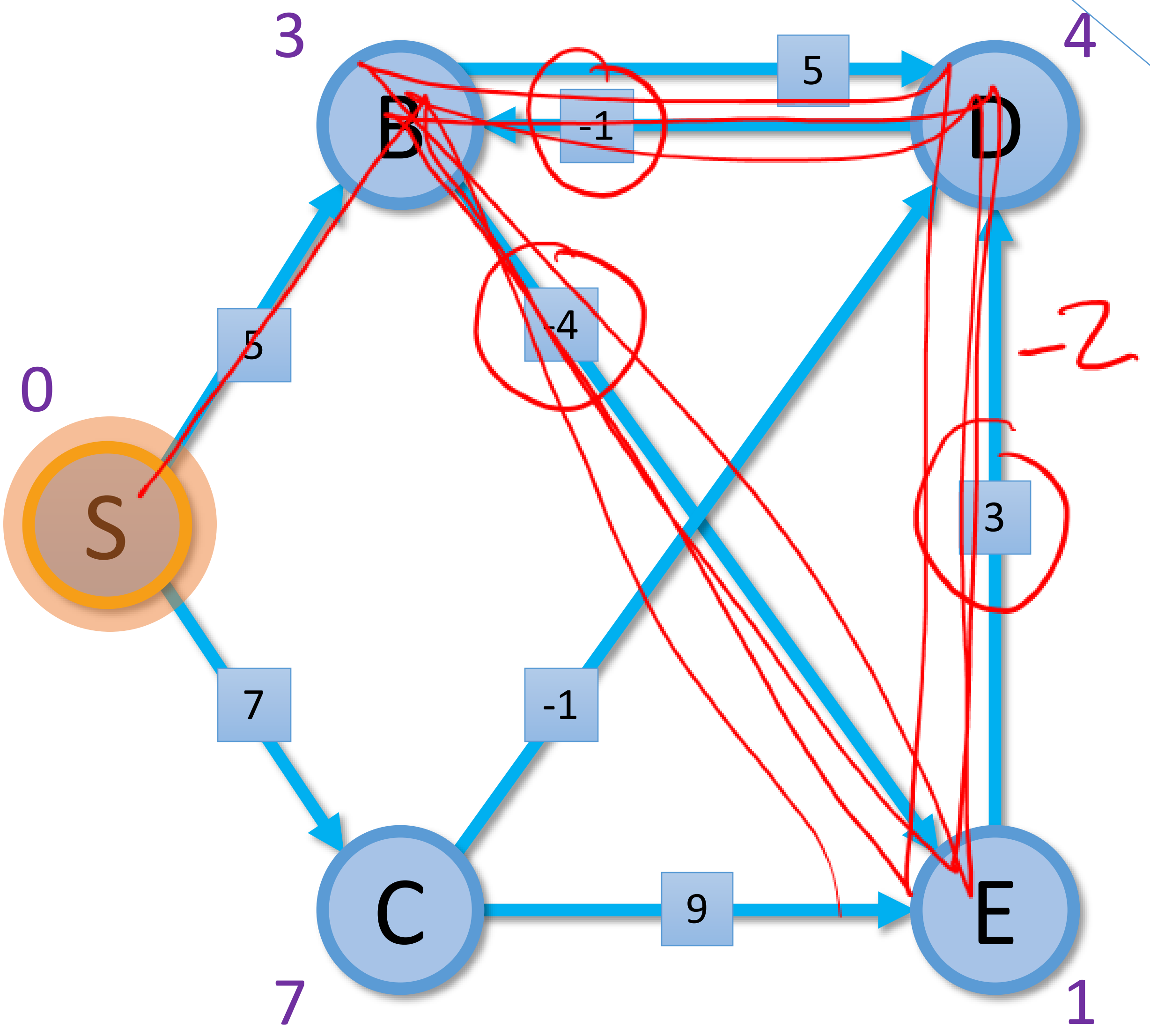
What is the shortest path from S to B?

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)

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