

# Shortest Path Algorithms

Instructor: Krishna Venkatasubramanian

CSC 212

# Announcements

- Final exam: Thursday (Dec 19) at 11:30am.
  - It will be held in this classroom (WHITE HALL 205)
- Topics for the final exam have been sent to everyone.
- Assignment 3 is due tonight by 11:59pm.

# Shortest-Path Variants



<b>Single-Source Single-Destination (1-1)</b> <ul style="list-style-type: none"><li>- No good solution that beats 1-M variant</li><li>- Thus, this problem is mapped to the 1-M variant</li></ul>	<b>Single-Source All-Destination (1-M)</b> <ul style="list-style-type: none"><li>- Need to be solved (several algorithms)</li><li>- We will study this one</li></ul>
<b>All-Sources Single-Destination (M-1)</b> <ul style="list-style-type: none"><li>- Reverse all edges in the graph</li><li>- Thus, it is mapped to the (1-M) variant</li></ul>	<b>All-Sources All-Destinations (M-M)</b> <ul style="list-style-type: none"><li>- Need to be solved (several algorithms)</li><li>- Will not be covered in this class</li></ul>

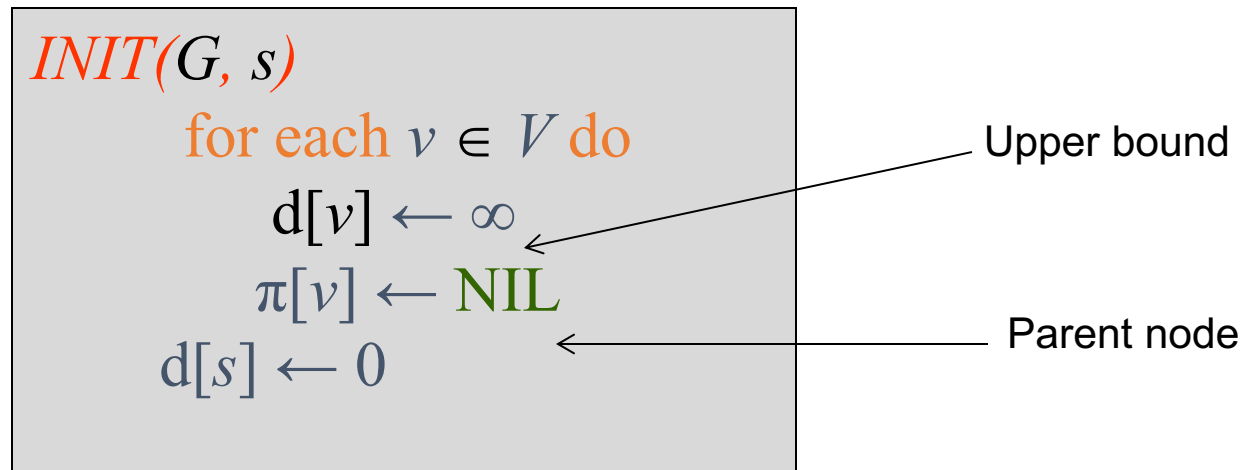
# Shortest Path

**Shortest Path** = Path of minimum weight

$$d(u,v) = \begin{cases} \min\{\omega(p) : u \xrightarrow{p} v\}; & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise.} \end{cases}$$

# Important Idea: Initialization

- Maintain  $d[v]$  for each  $v$  in  $V$
- $d[v]$  is called *shortest-path weight estimate*

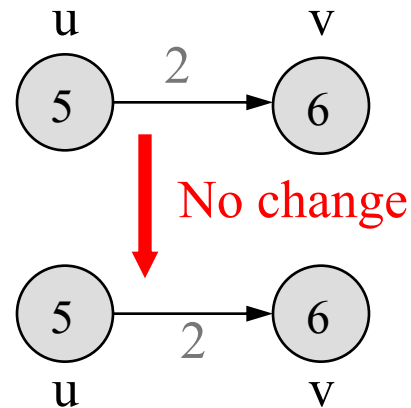
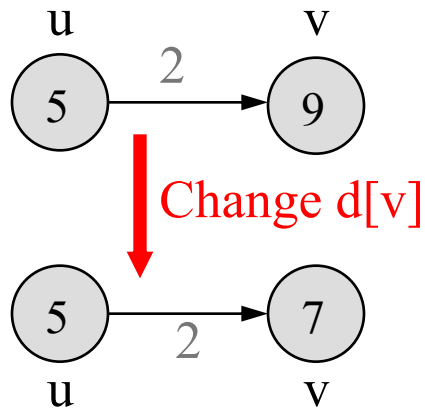


# Important Idea: Relaxation

**RELAX**( $u, v$ )

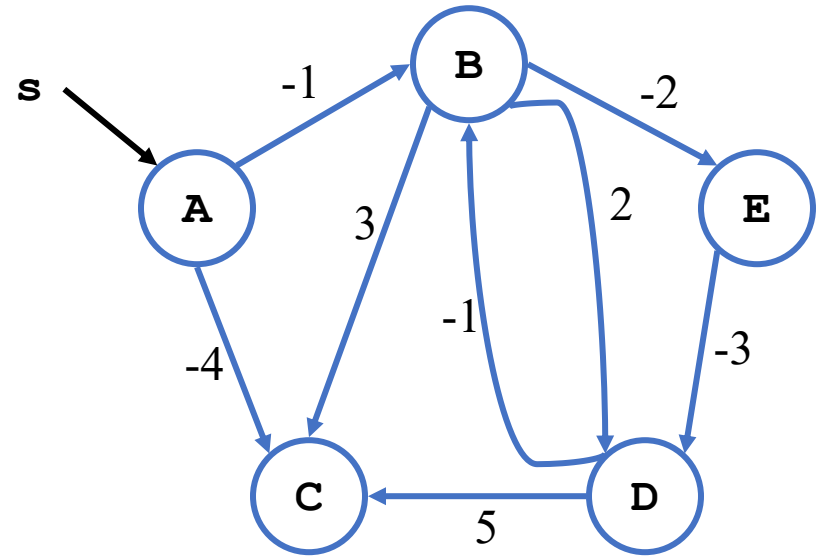
if  $d[v] > d[u] + w(u, v)$  then  
     $d[v] \leftarrow d[u] + w(u, v)$   
     $\pi[v] \leftarrow u$

When you find an edge  $(u, v)$  then  
check this condition and relax  $d[v]$  if  
possible



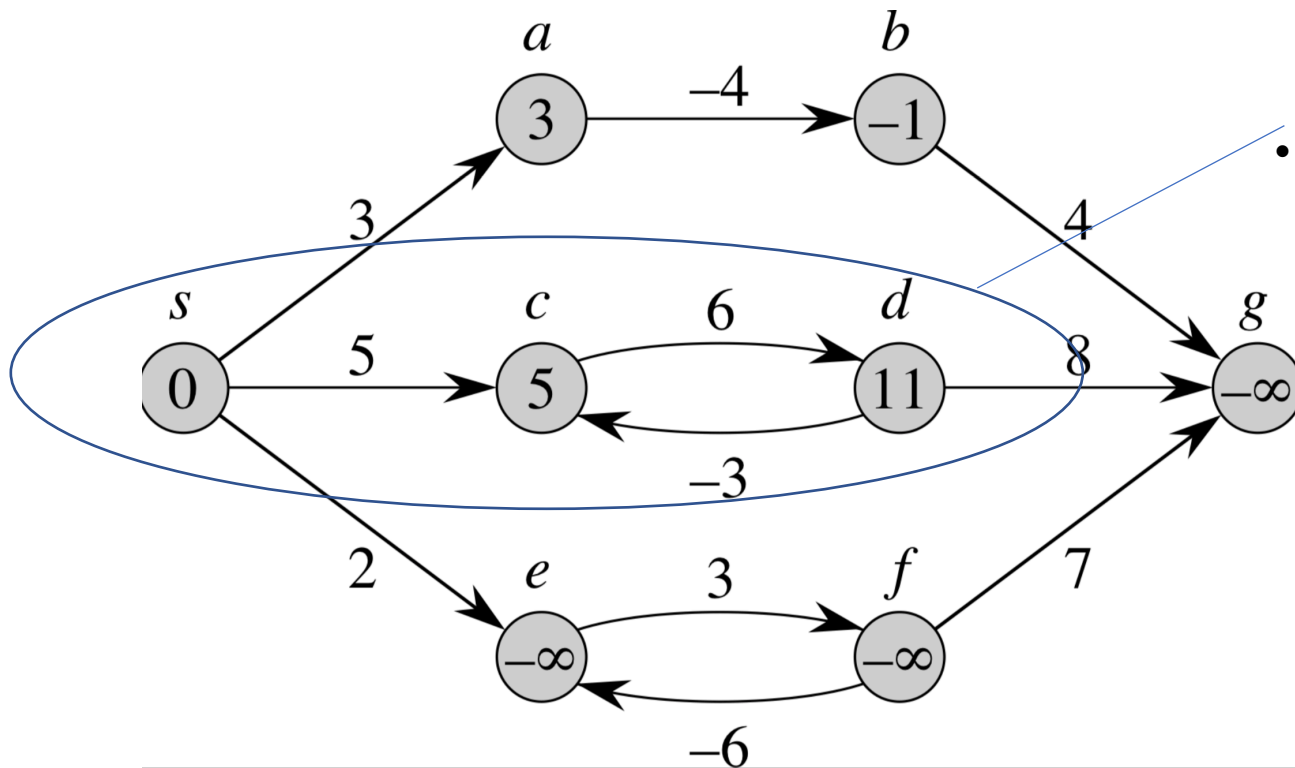
# Graphs Edges and Negative Weights

- Graph edges can have negative weights
- The meaning of the negative weights depends upon the context
  - E.g., decrease in traffic in a network at midnight compared to noon!
- **One can still find 1-M shortest paths to vertices with edges have negative weights**
  - But not always (see next two slides)



# Example

- In graphs with negative edge weights shortest paths can be found:

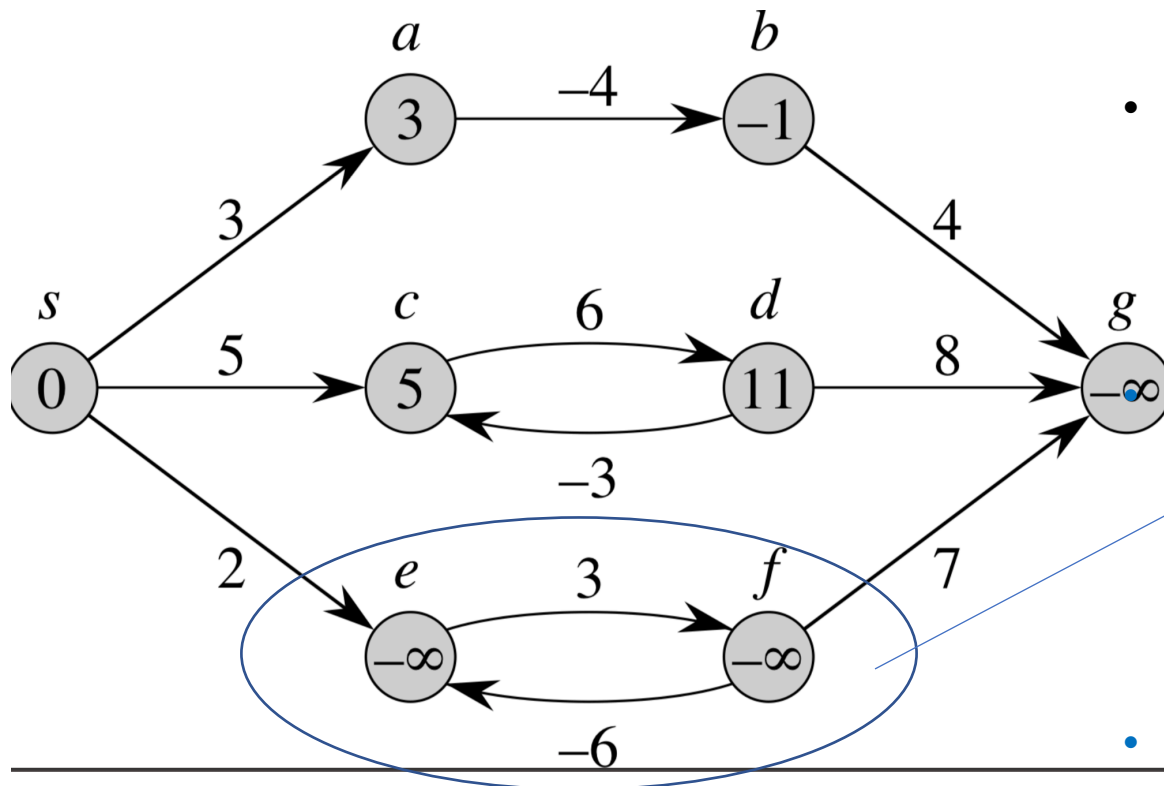


- There are infinitely many paths from  $s$  to  $c$ :  $\langle s, c \rangle$ ,  $\langle s, c, d, c \rangle$ ,  $\langle s, c, d, c, d, c \rangle$ , ...
- However, since the cycle  $\langle c, d, c \rangle$  has a weight of  $6 + (-3) > 0$ ,  $d(c) = 5$  and  $d(d) = 11$  and will not be affected by the negative weights



# Example

- In graphs with negative weight cycles, shortest path to some nodes will not exist:



- There are infinitely many paths from  $s$  to  $e$ :  $\langle s, e \rangle$ ,  $\langle s, e, f, e \rangle$ ,  $\langle s, e, f, e, f, e \rangle$ , ...
- However, since the cycle  $\langle e, f, e \rangle$  has a weight of  $3 + (-6) < 0$ ,  $d(e) = -\infty$  and  $d(d) = -\infty$ .  
**No matter what  $d[e]$  starts with, going to vertex  $f$  and coming back to vertex  $e$  will reduce  $d[e]$  further.**
- The same goes for  $d[f]$**

# Cycles in Shortest Path?

- **Negative weight cycles\* cannot be in a shortest path**

\*That is, the entire cycle path adds up to a negative weight

- **Positive weight cycles will not be in a shortest path**

- You can remove the cycle from the path and it will produce a better shortest path
- See previous slide 8. We just ignore the cycle.

- **IMPORTANT POINT: When we find 1-M shortest path, there are no cycles in any of the paths**

# Shortest Path Algorithms

- **Bellman-Ford**

- Solve single source shortest path algorithm in a graph **with negative edge weights**

- **Dijkstra's**

- Solve single source shortest path algorithm in a graph **with positive edge weights (only)**
- **Faster**

# Bellman-Ford Algorithm

```
BellmanFord()  
  for each  $v \in V$   
     $d[v] = \infty$ ;  
 $d[s] = 0$ ;  
  for  $i=1$  to  $|V|-1$   
    for each edge  $(u,v) \in E$   
      Relax( $u,v, w(u,v)$ );  
  for each edge  $(u,v) \in E$   
    if ( $d[v] > d[u] + w(u,v)$ )  
      return "no solution";
```

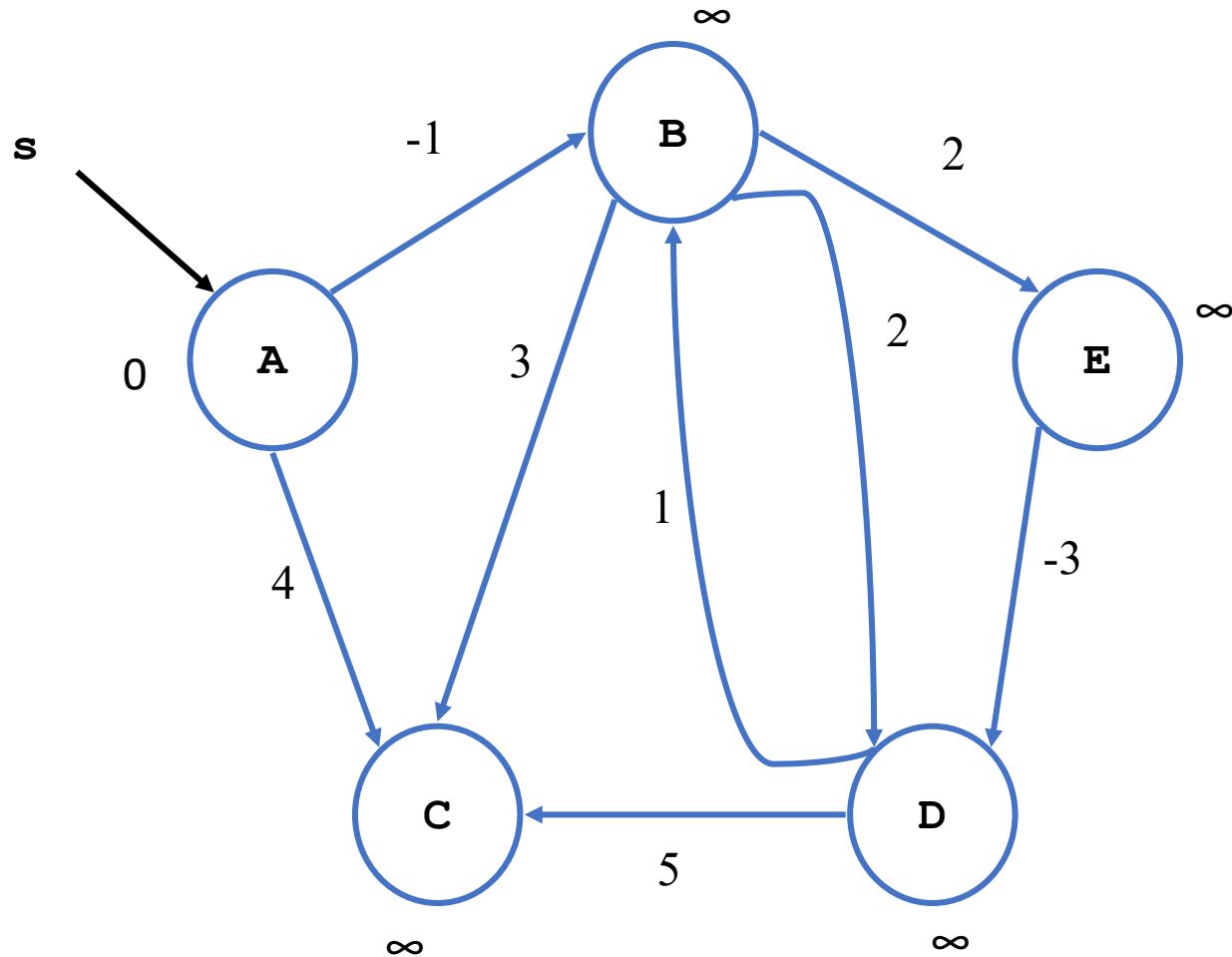
Initialize  $d[]$ , which will converge to shortest-path value  $\delta$

Relaxation:  
Make  $|V|-1$  passes, relaxing each edge

Test for solution  
Under what condition do we get a solution?  
--- NO Cycles in the final shortest path!

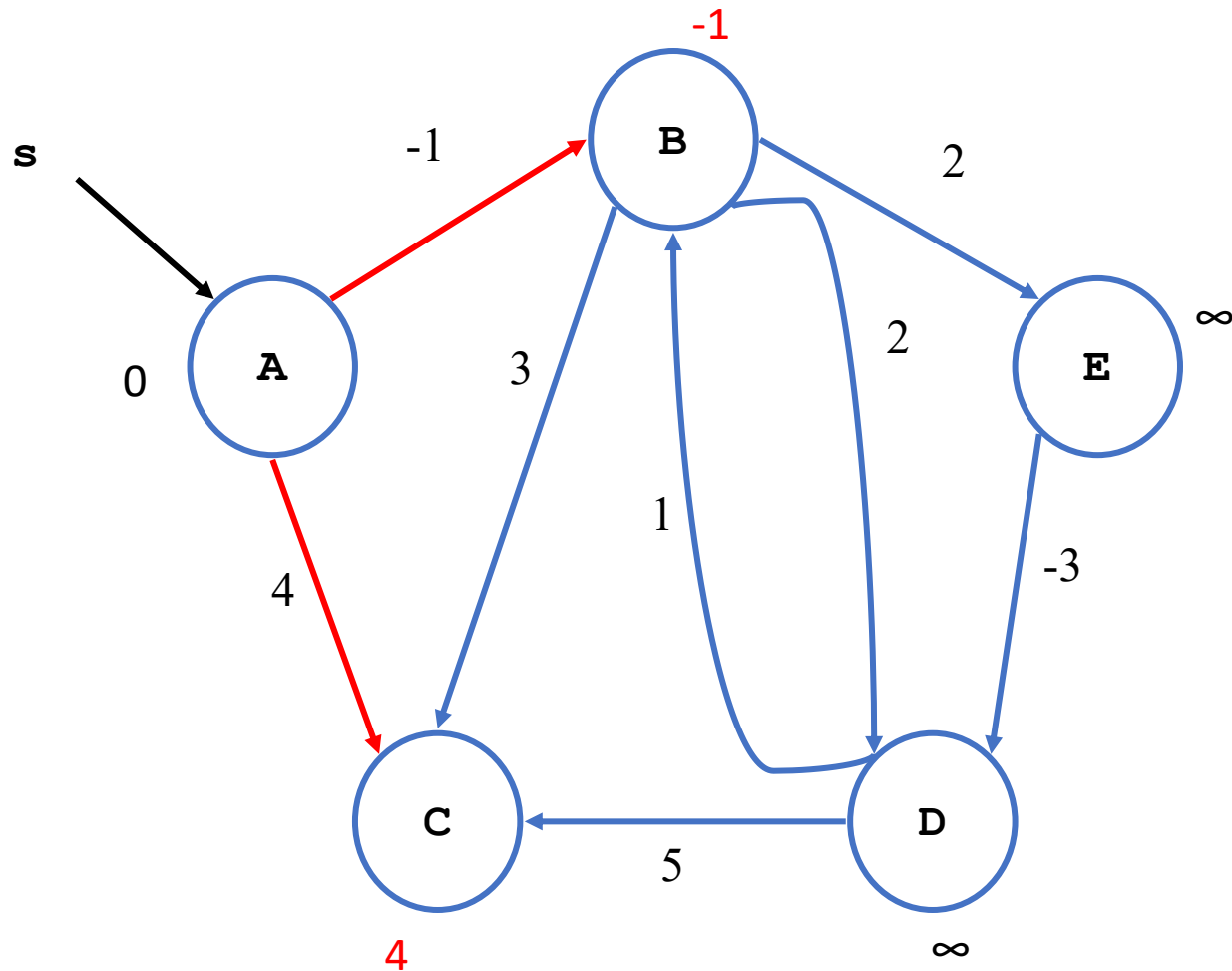
Relax( $u,v,w$ ): if ( $d[v] > d[u]+w$ ) then  $d[v]=d[u]+w$

# Bellman-Ford Algorithm: Example



Order in which edges evaluated:  $\{(A,B),(A,C),(B,C),(B,E),(B,D),(E,D),(D,B),(D,C)\}$

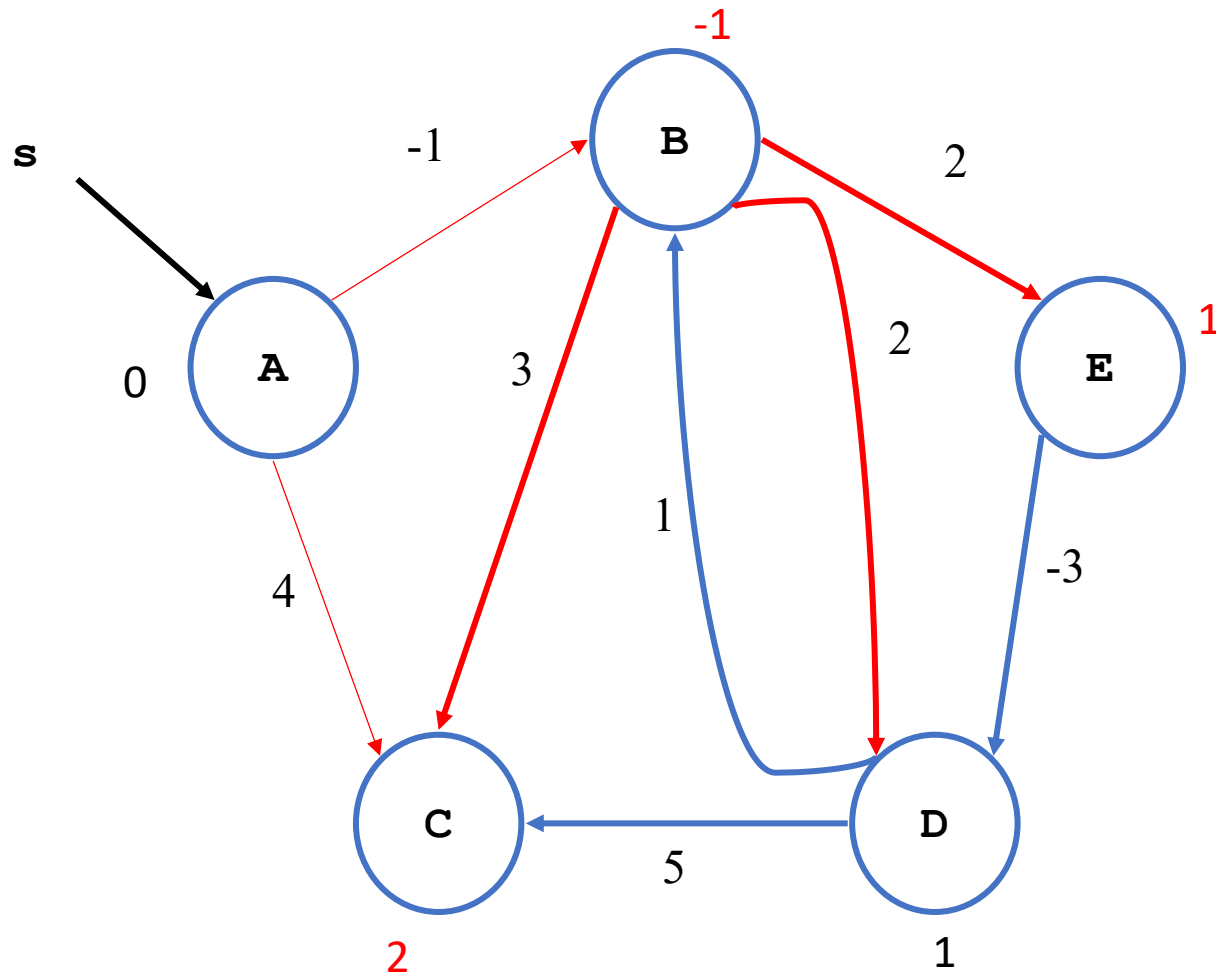
# Bellman-Ford Algorithm: Example



Order in which edges evaluated:  $\{(A,B), (A,C), (B,C), (B,E), (B,D), (E,D), (D,B), (D,C)\}$

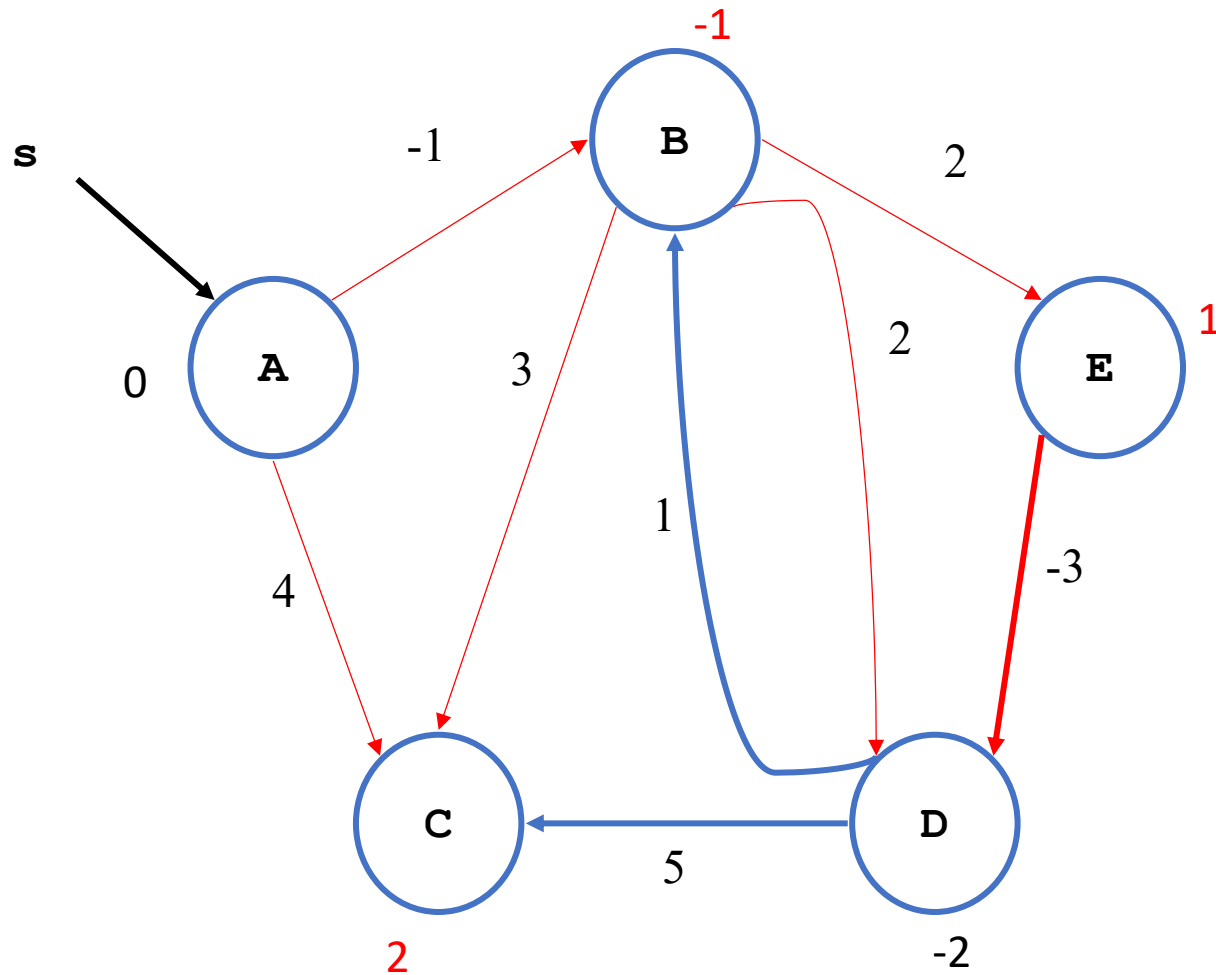
I have shown the relaxation of two edges at a time to speed up the slides. You should look at each edge in the list one by one, and relax it

# Bellman-Ford Algorithm: Example



Order in which edges evaluated: {(A,B),(A,C),(B,C),(B,E),(B,D),(E,D),(D,B),(D,C)}

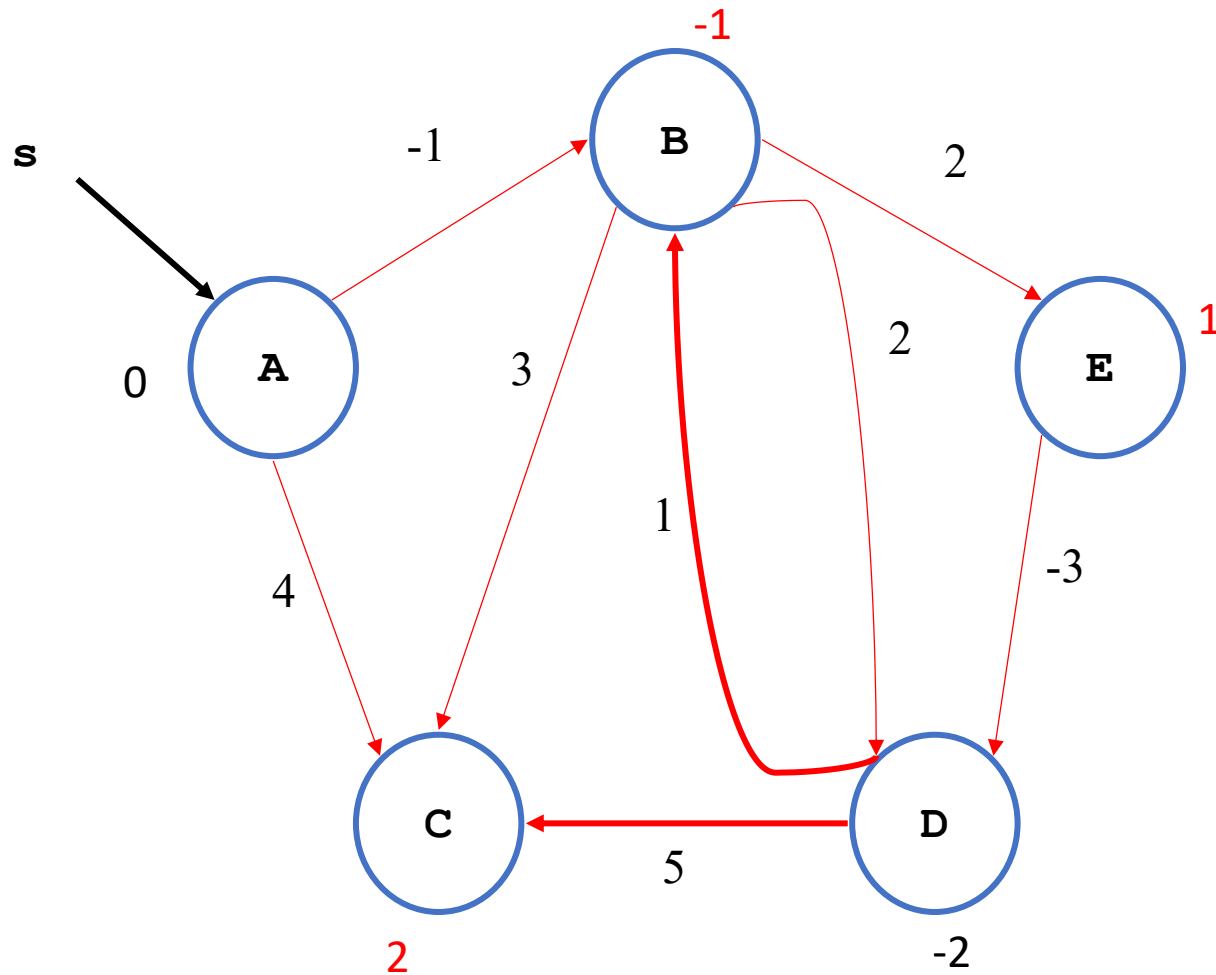
# Bellman-Ford Algorithm: Example



Order in which edges evaluated: {(A,B),(A,C),(B,C),(B,E),(B,D),(E,D),(D,B),(D,C)}

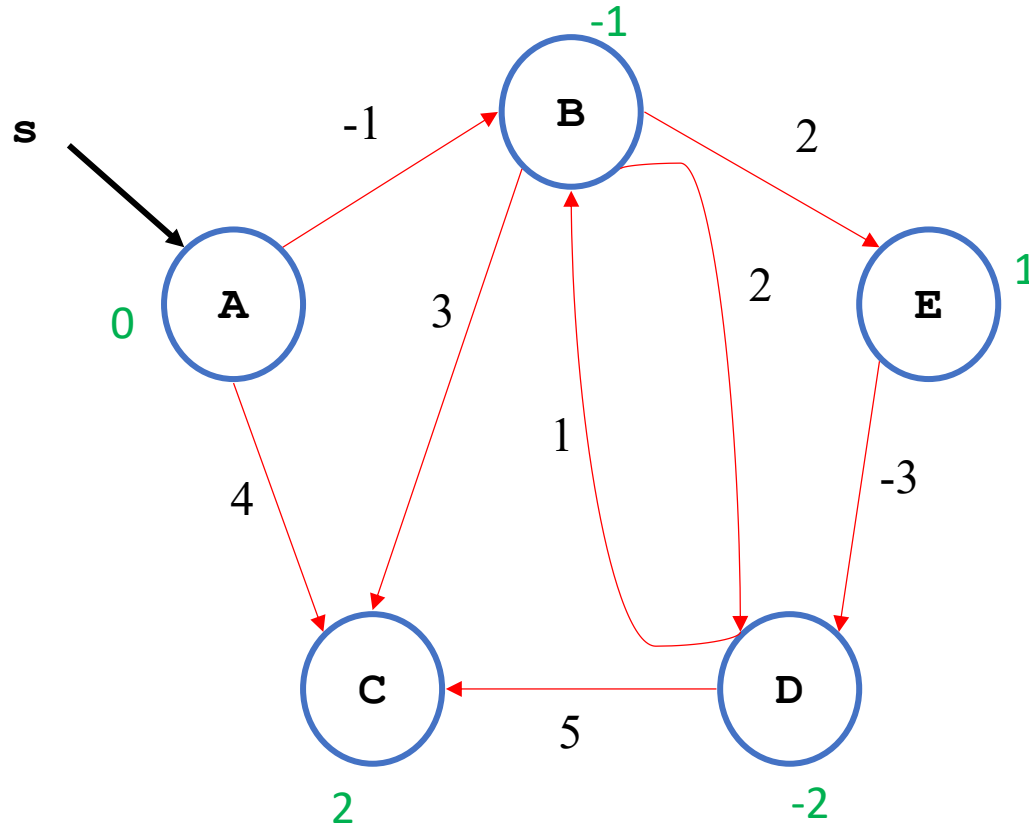


# Bellman-Ford Algorithm: Example



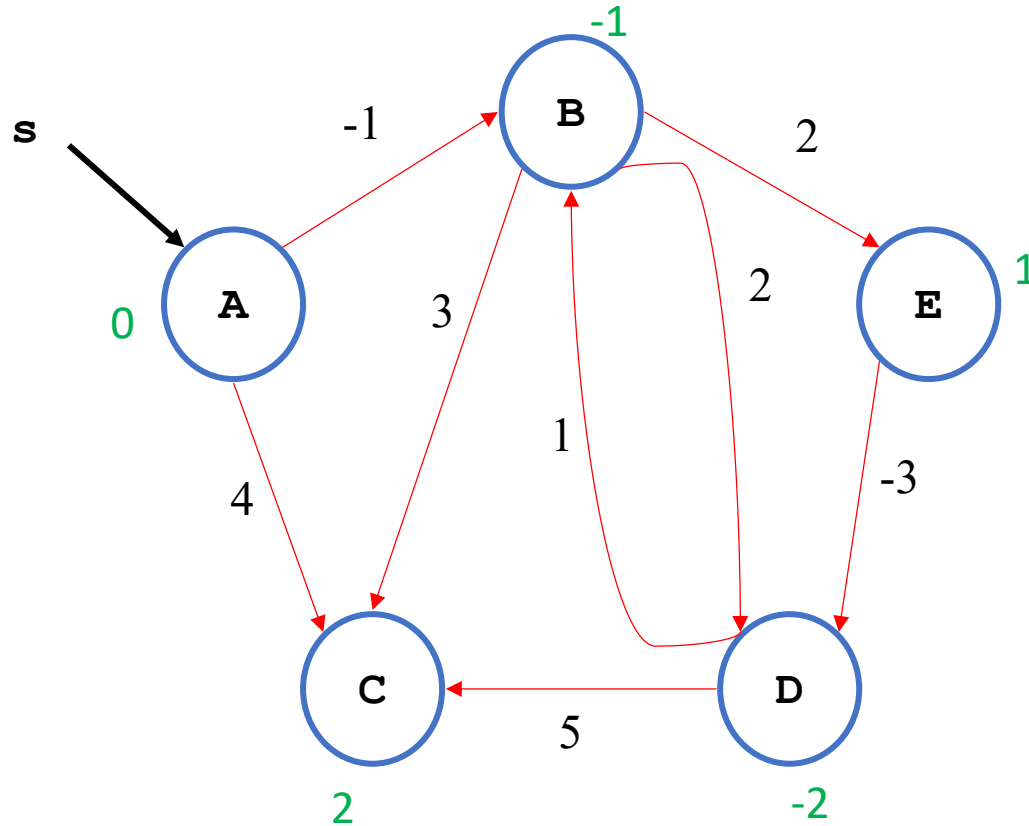
Order in which edges evaluated: {(A,B),(A,C),(B,C),(B,E),(B,D),(E,D),(D,B),(D,C)}

# Bellman-Ford Algorithm: Example



- Do this relaxation  $V-1$  (4) times. Again and again given “weights” in every round
- After  $V-1$  rounds, we will be done.
- **The order of edge evaluation does not matter (try at home). The result (of distance from s) should be the same.**

# Bellman-Ford Algorithm: Example



Are there cycles?  
..Nope

- Check to see if there are cycles in this graph?
- For each edge check
  - if  $d[v] > d[u] + w(u,v)$
  - If so. we say there are cycles in the graph and shortest path cannot be found for all nodes in the graph from 's'.

# Bellman-Ford Algorithm

```
BellmanFord()  
  for each  $v \in V$   
     $d[v] = \infty$ ;  
 $d[s] = 0$ ;  
  for  $i=1$  to  $|V|-1$   
    for each edge  $(u,v) \in E$   
      Relax( $u,v, w(u,v)$ );  
  for each edge  $(u,v) \in E$   
    if ( $d[v] > d[u] + w(u,v)$ )  
      return "no solution";
```

What will be the running time?

A:  $O(VE)$

*We look at all  $E$  edges  $V-1$  times.*

*Can we do better?*

*– Yes, with caveats*

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

# Dijkstra's Algorithm For Shortest Paths

- **Non-negative edge weight**
- **Like BFS:** If all edge weights are equal, then use BFS, otherwise use this algorithm
- Use  $Q$  = min-**priority queue** keyed on  $d[v]$  values

# Dijkstra's Algorithm For Shortest Paths

**DIJKSTRA**(G, s)

**INIT**(G, s)

$S \leftarrow \emptyset$                       > set of discovered nodes

$Q \leftarrow V[G]$                       > put all vertices in a queue

**while**  $Q \neq \emptyset$  **do**

$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

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

**RELAX**( $u, v$ )                      > May cause

   > **DECREASE-KEY**( $Q, v, d[v]$ )

PRIORITY QUEUE OPERATION  
(Yet Again)

# Example: Initialization Step

**DIJKSTRA**(G, s)

**INIT**(G, s)

$S \leftarrow \emptyset$       > set of discovered nodes

$Q \leftarrow V[G]$

**while**  $Q \neq \emptyset$  **do**

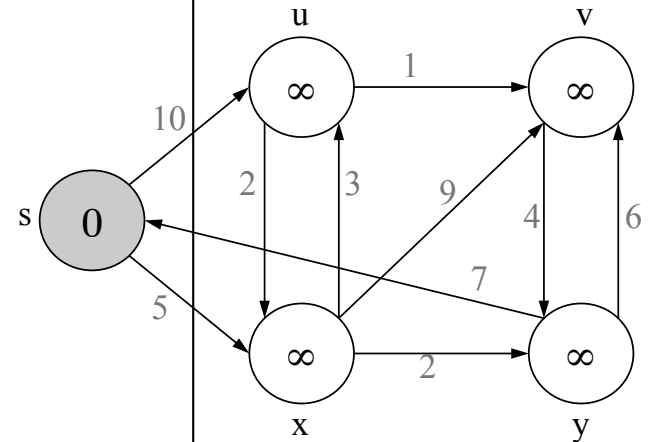
$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

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

**RELAX**( $u, v$ )      > May cause

            > **DECREASE-KEY**( $Q, v, d[v]$ )



# Example

**DIJKSTRA**(G, s)

**INIT**(G, s)

$S \leftarrow \emptyset$       > set of discovered nodes

$Q \leftarrow V[G]$

**while**  $Q \neq \emptyset$  **do**

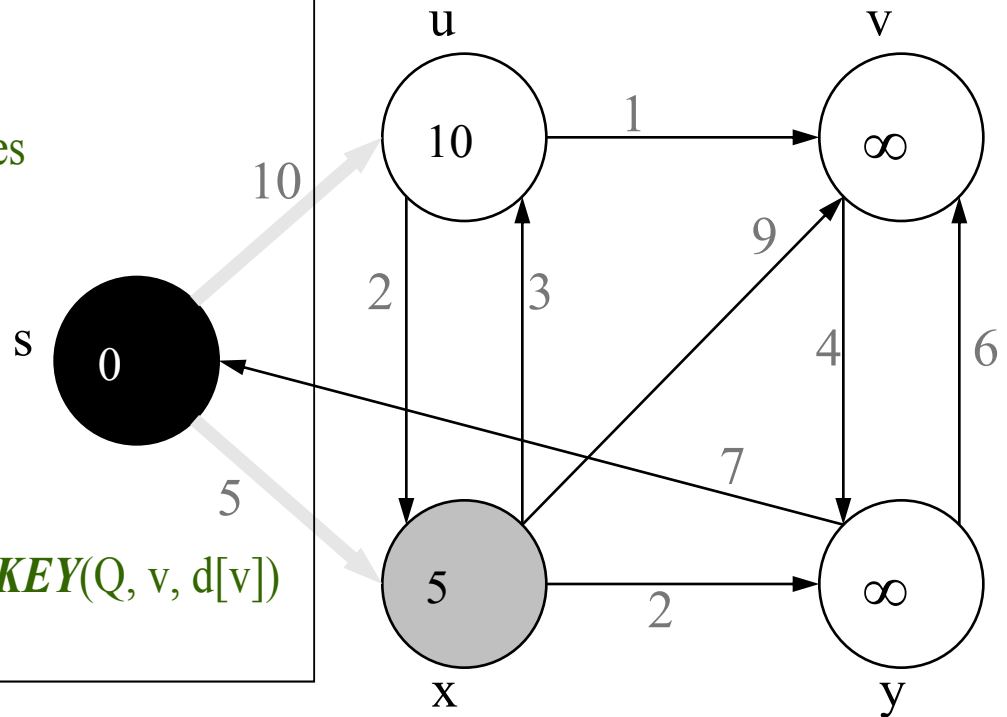
$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

**for each** v in Adj[u] **do**

**RELAX**(u, v) > May cause

        > **DECREASE-KEY**(Q, v, d[v])





# Example

**DIJKSTRA**(G, s)

**INIT**(G, s)

$S \leftarrow \emptyset$       > set of discovered nodes

$Q \leftarrow V[G]$

**while**  $Q \neq \emptyset$  **do**

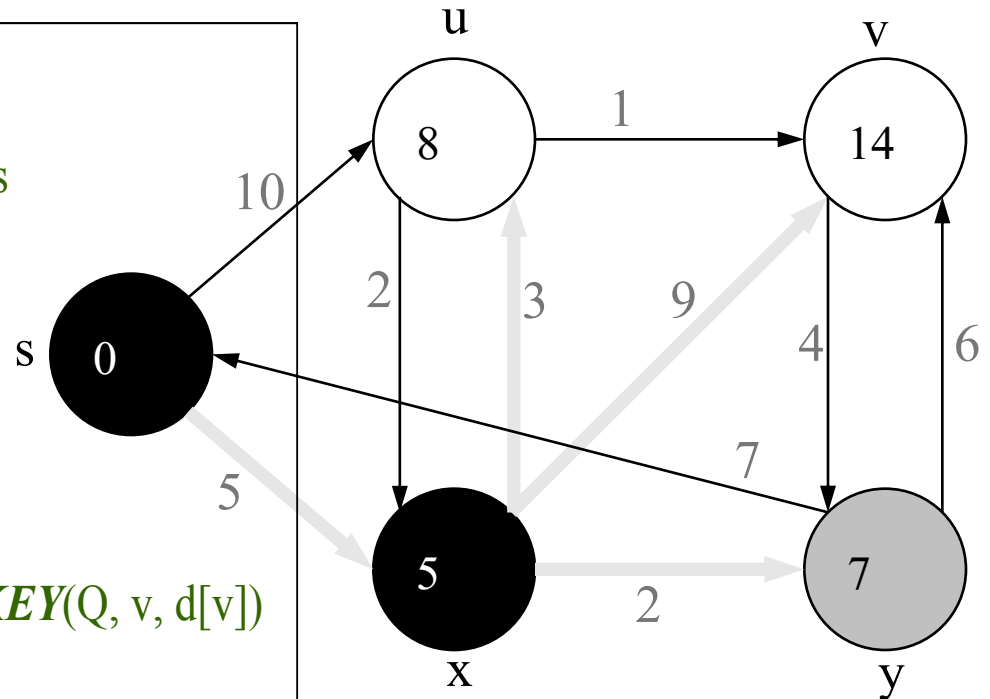
$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

**for each** v in Adj[u] **do**

**RELAX**(u, v) > May cause

        > **DECREASE-KEY**(Q, v, d[v])



# Example

**DIJKSTRA**(G, s)

**INIT**(G, s)

$S \leftarrow \emptyset$       > set of discovered nodes

$Q \leftarrow V[G]$

**while**  $Q \neq \emptyset$  **do**

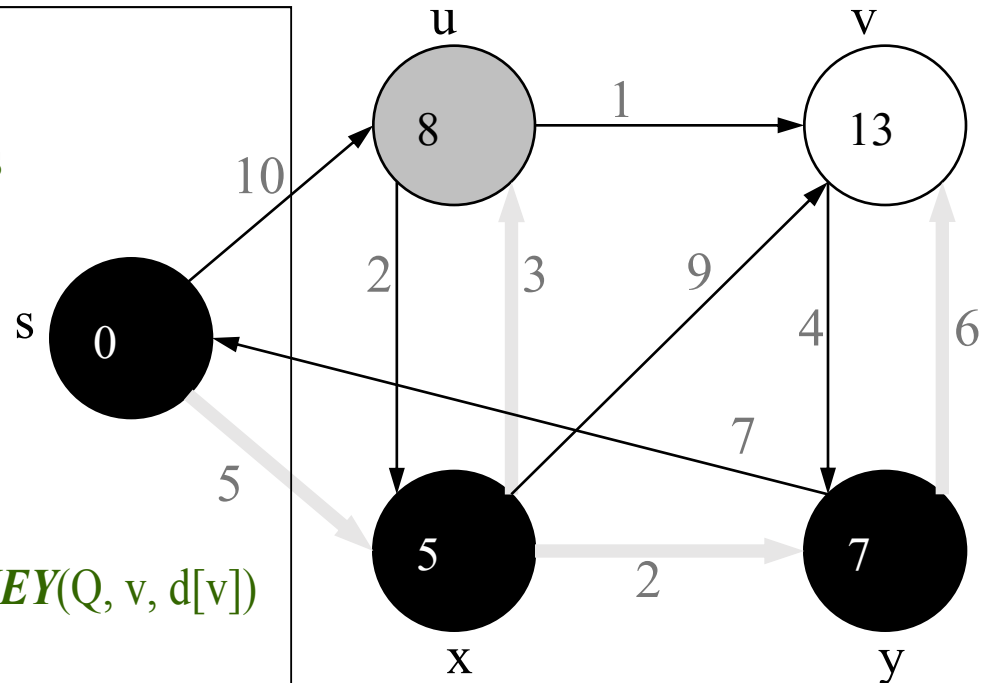
$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

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

**RELAX**(u, v) > May cause

        > **DECREASE-KEY**(Q, v, d[v])



# Example

**DIJKSTRA**(G, s)

**INIT**(G, s)

$S \leftarrow \emptyset$       > set of discovered nodes

$Q \leftarrow V[G]$

**while**  $Q \neq \emptyset$  **do**

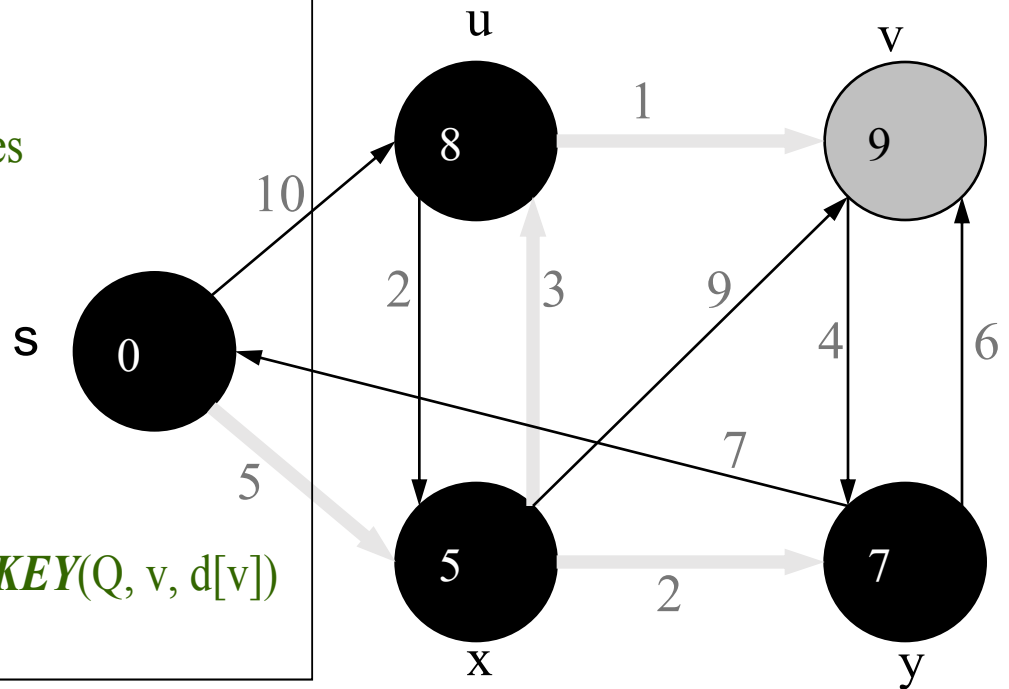
$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S \leftarrow S \cup \{u\}$

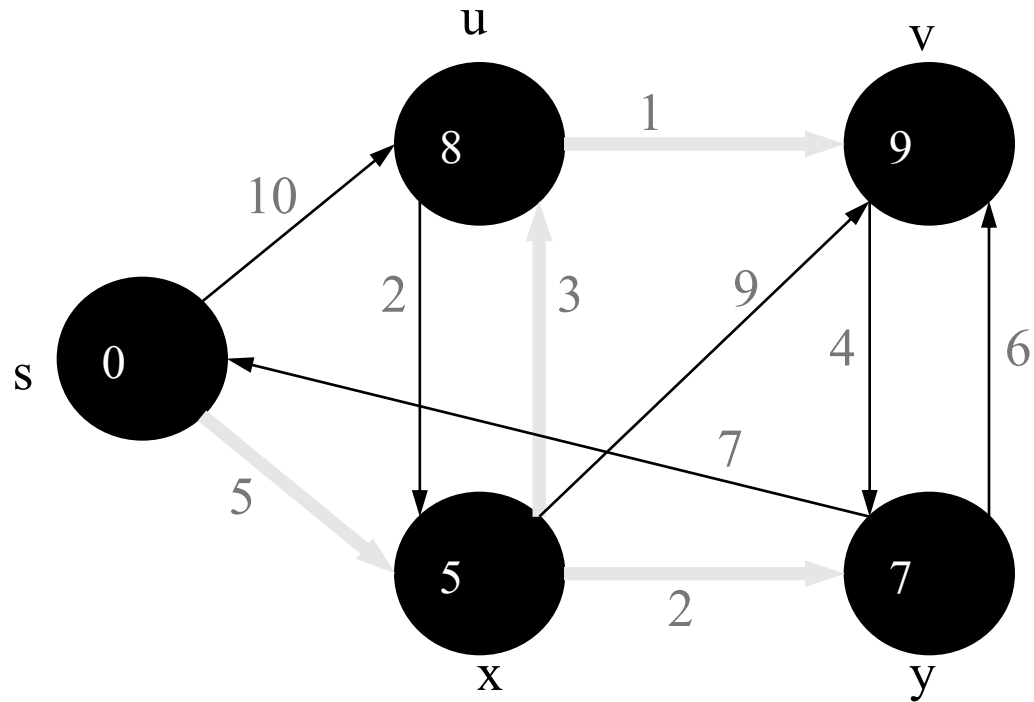
**for** each  $v$  in  $\text{Adj}[u]$  **do**

**RELAX**(u, v) > May cause

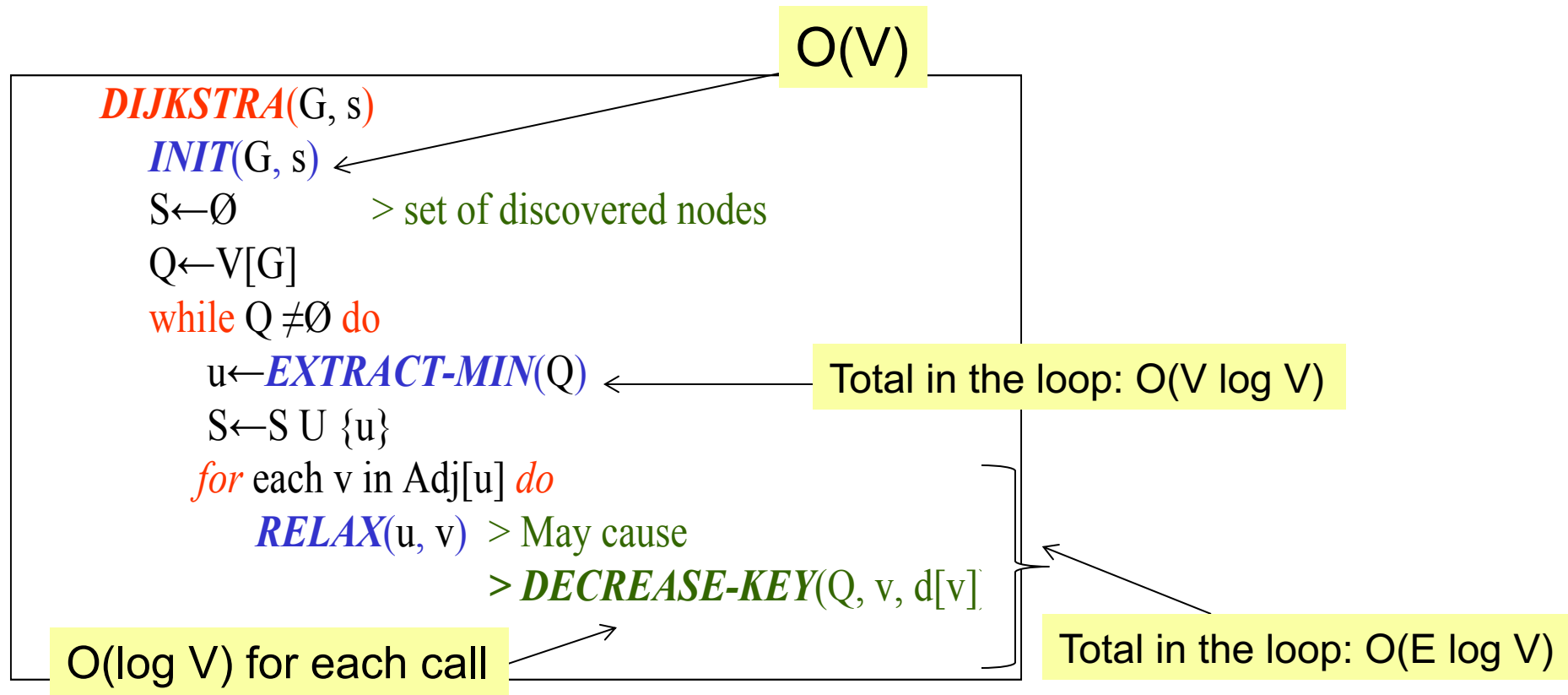
        > **DECREASE-KEY**(Q, v, d[v])



# Example



# Dijkstra's Algorithm Analysis



Time Complexity:  $O(E \log V)$



*That's all Folks!*  
*Any Question?*