CENG 218
Design and Analysis of Algorithms

Izmir Institute of Technology

Lecture 13: Shortest paths

Slides were mostly prepared using the material provided by Prof. Charles E. Leiserson and Prof. Erik Demaine from MIT

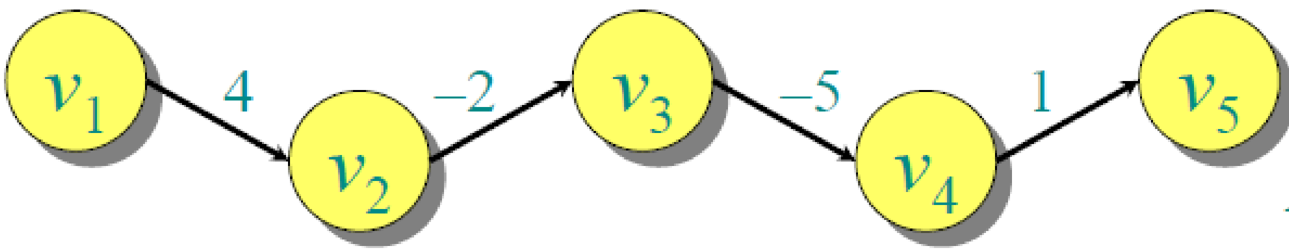
Paths in graphs

Consider a digraph $G = (V, E)$ with edge-weight function $w: E \to \mathbb{R}$.

The ***weight*** of path $p = v_1 \to v_2 \to \dots \to v_k$ can be defined as

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

Example:



$$w(p) = -2$$

Shortest paths

A ***shortest path*** from u to v is a path of minimum weight from u to v. The ***shortest-path weight*** from u to v is defined as

$$\delta(u, v) = \min\{w(p): p \text{ is a path from } u \text{ to } v\}.$$

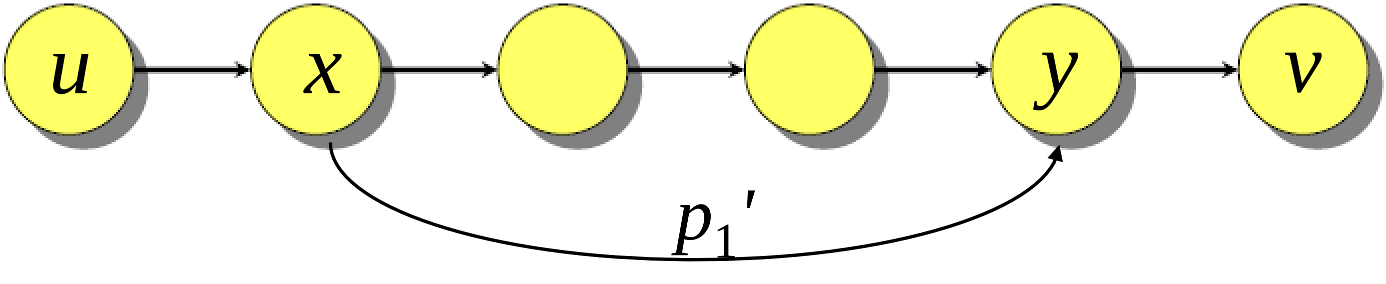
Note: $\delta(u, v) = \infty$ if no path from u to v exists.

Optimal substructure

Theorem. A sub-path of a shortest path is a shortest path.

Proof. Let p be the shortest path from u to v and let p contain p_1 which is the shortest path from x to y.

If there was a p_1' that is shorter than p_1, then $w(p') = \delta(u,x) + w(p_1') + \delta(y,v)$ would be less than $w(p)$, which is a contradiction since p is the shortest path.

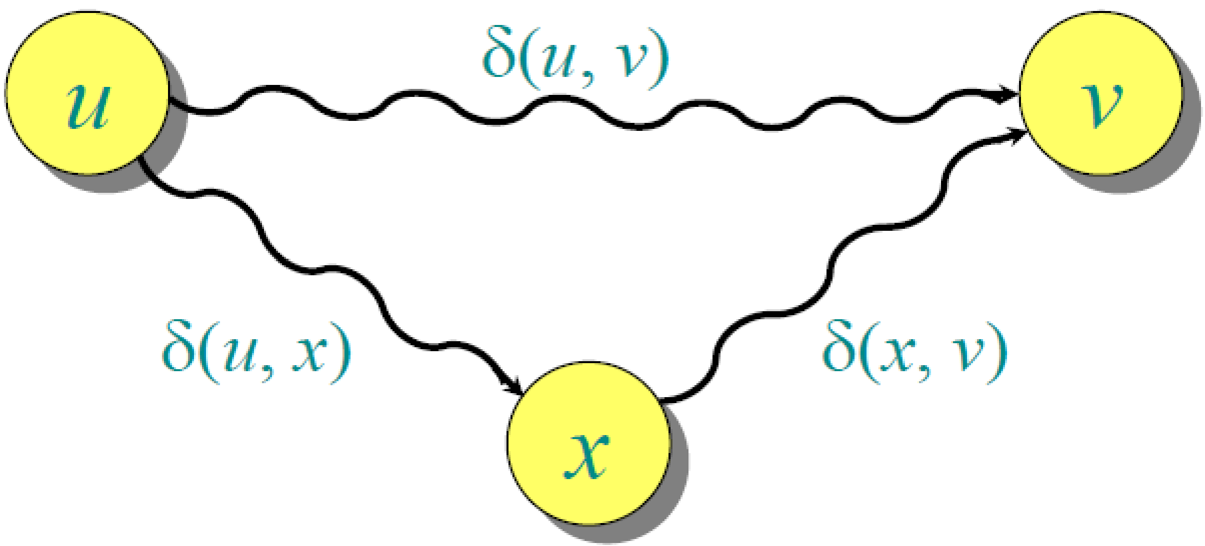


Triangle inequality

Theorem. For all u, v, $x \in V$, we have

$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

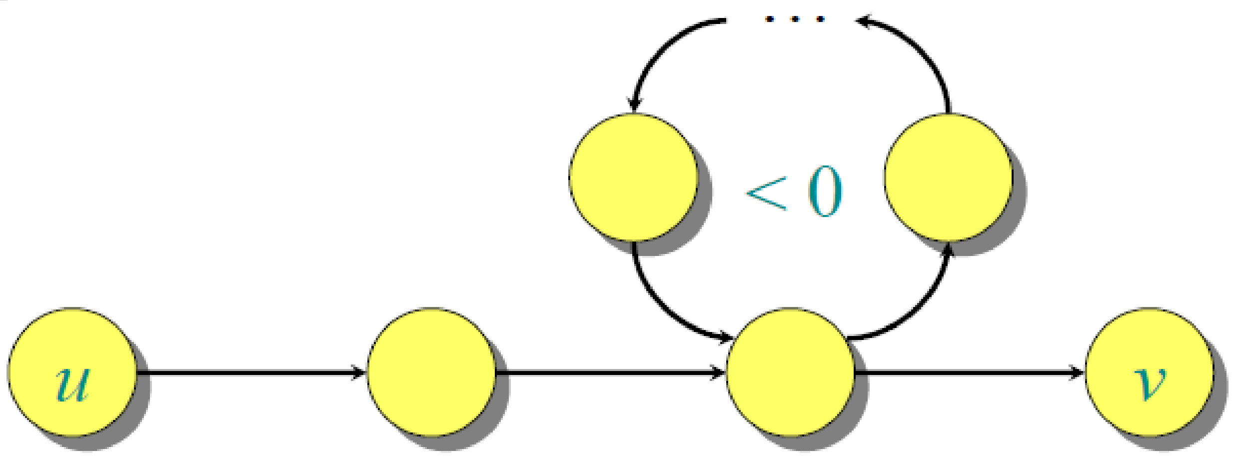
Proof. If $\delta(u, v)$ was bigger than $\delta(u, x) + \delta(x, v)$, then it would no longer be a shortest path from u to v.



When shortest paths do not exist?

If a graph G contains a negative-weight cycle, then some shortest paths may not exist.

Example:



You can keep with the cycle until reaching $-\infty$ weight.

Single-source shortest paths

Problem. From a given source vertex $s \in V$, find the shortest-path weights $\delta(s, v)$ for all $v \in V$. If all edge weights $w(u, v)$ are *nonnegative*, all shortest-paths must exist.

IDEA: Greedy.

1. Maintain a set S of vertices whose shortest-path distances from s are known.
2. At each step add to S the vertex $v \in V - S$ whose distance estimate from s is minimal.
3. Update the distance estimates of vertices adjacent to v.

Dijkstra's algorithm

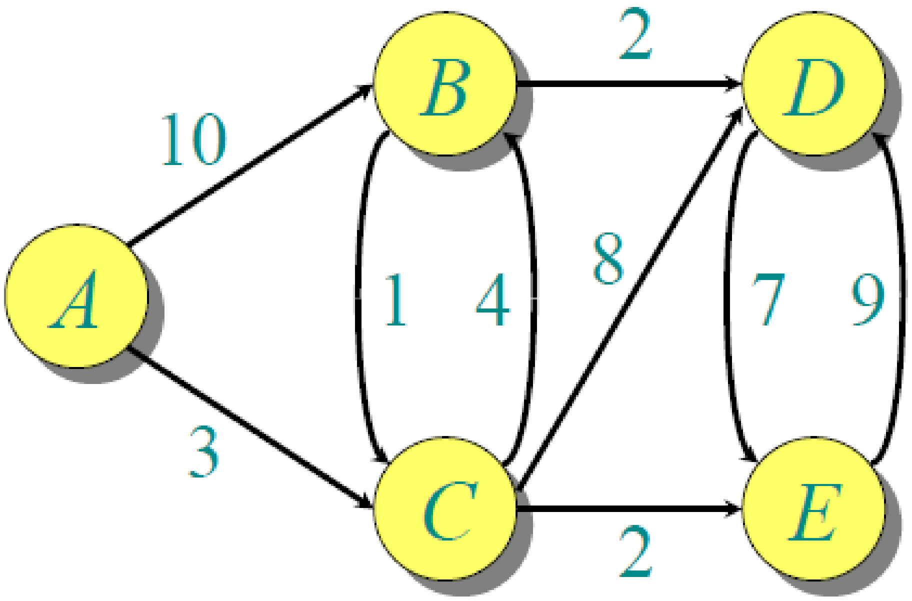
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ▷ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do **if** $d[v] > d[u] + w(u, v)$
          then $d[v] \leftarrow d[u] + w(u, v)$

Relaxation Step

Similar to DECREASE-KEY in Prim's algorithm for MST

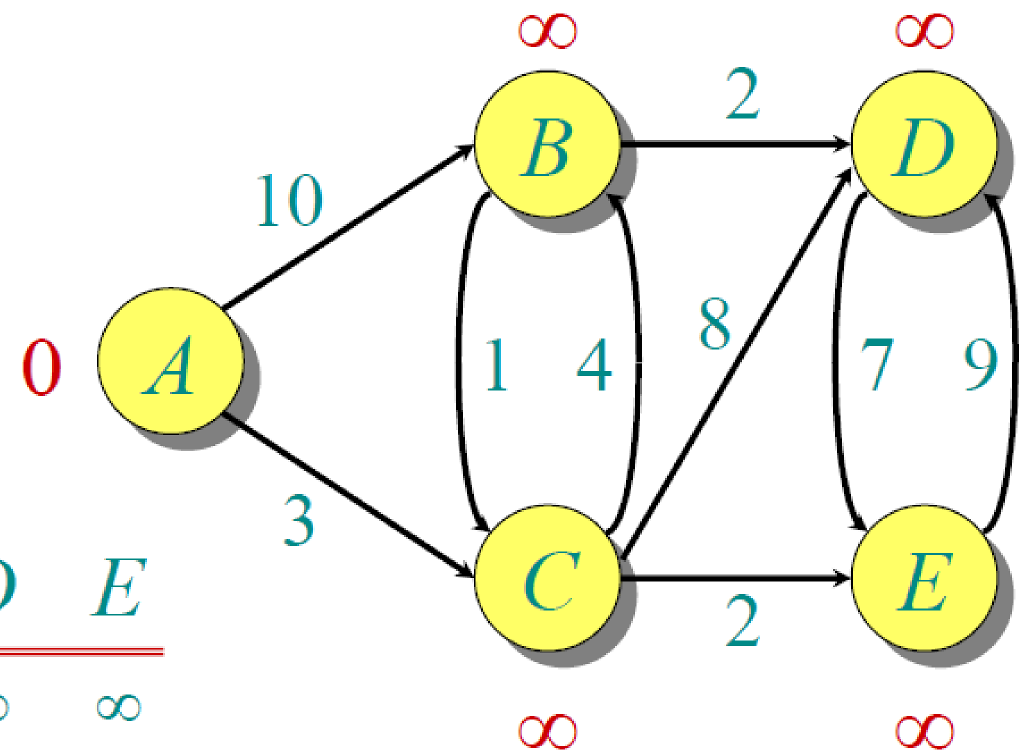
Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's algorithm

Initialize:

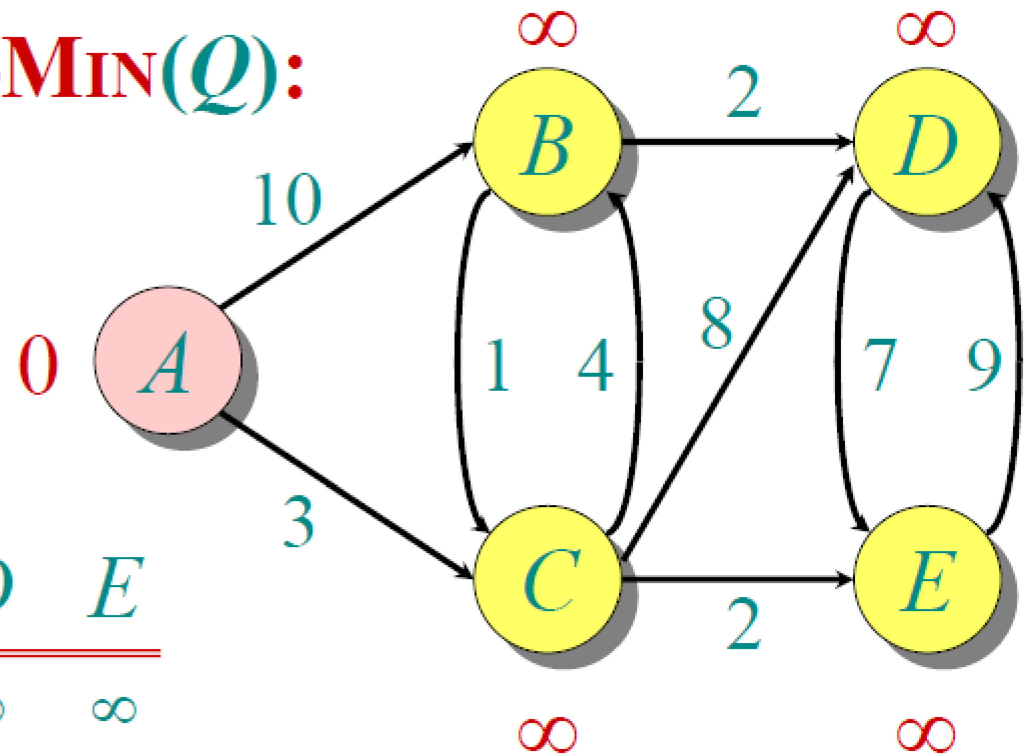


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):

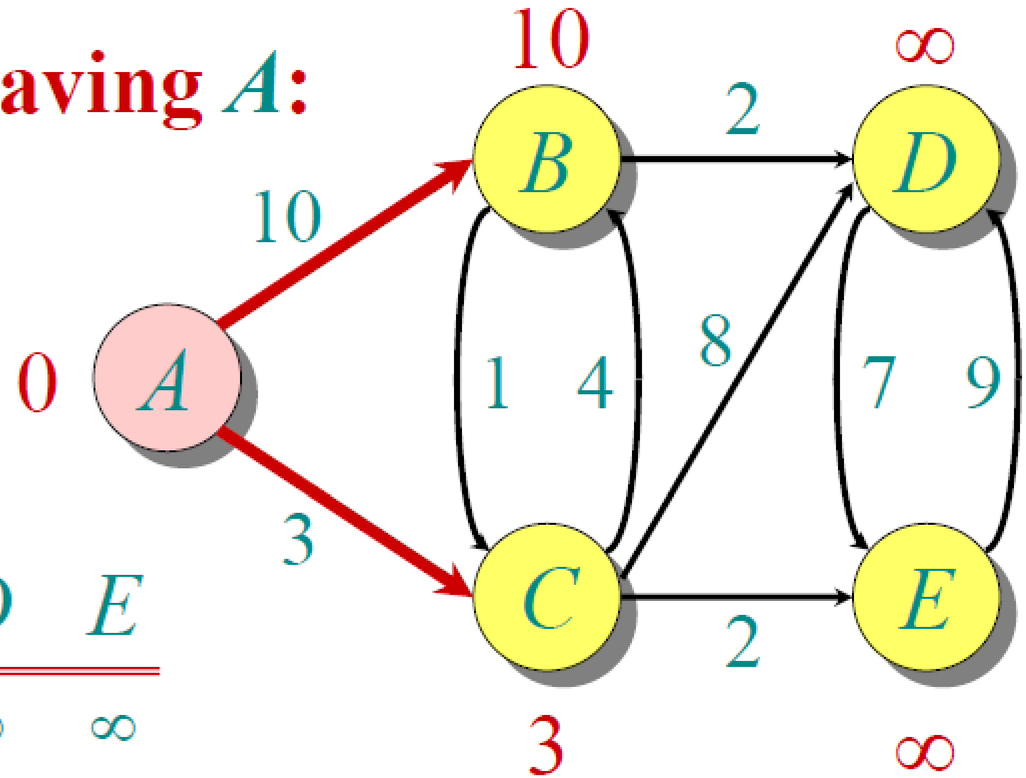


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Example of Dijkstra’s algorithm

Relax all edges leaving *A*:

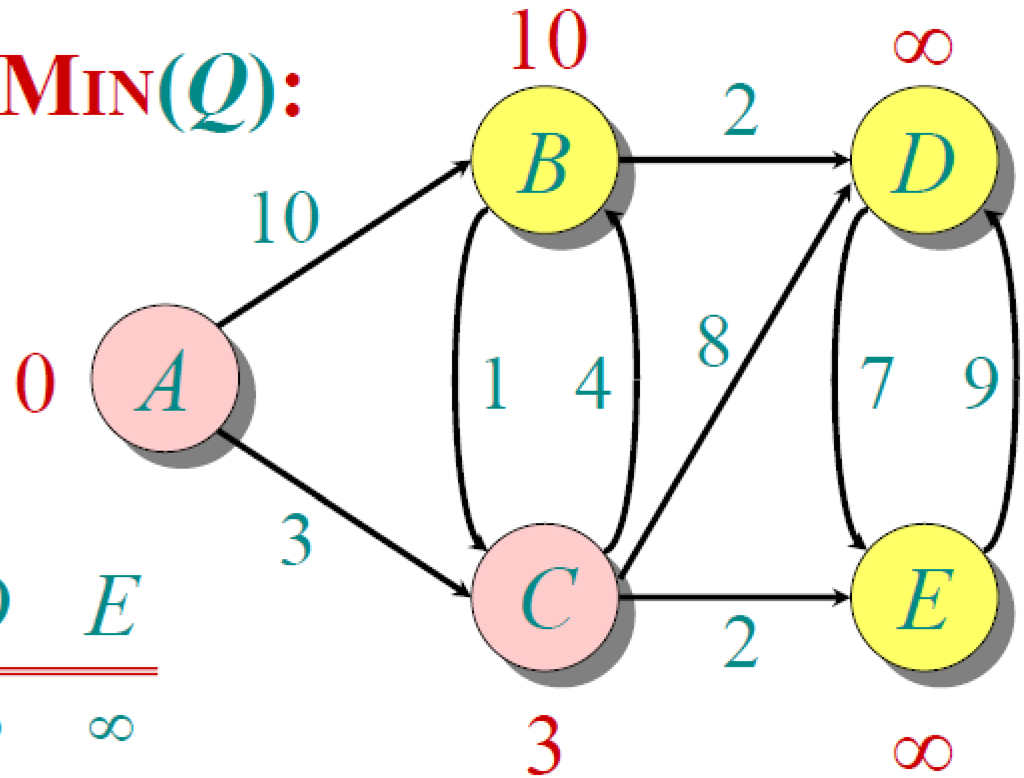


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Example of Dijkstra's algorithm

"*C*" ← EXTRACT-MIN(*Q*):

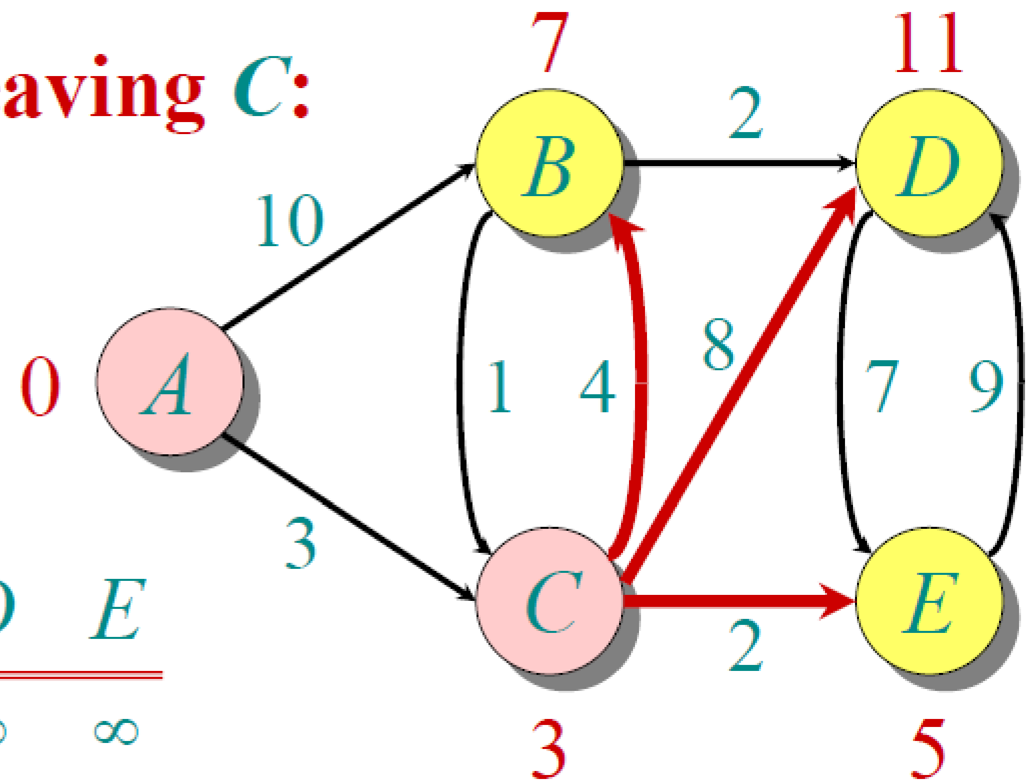


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

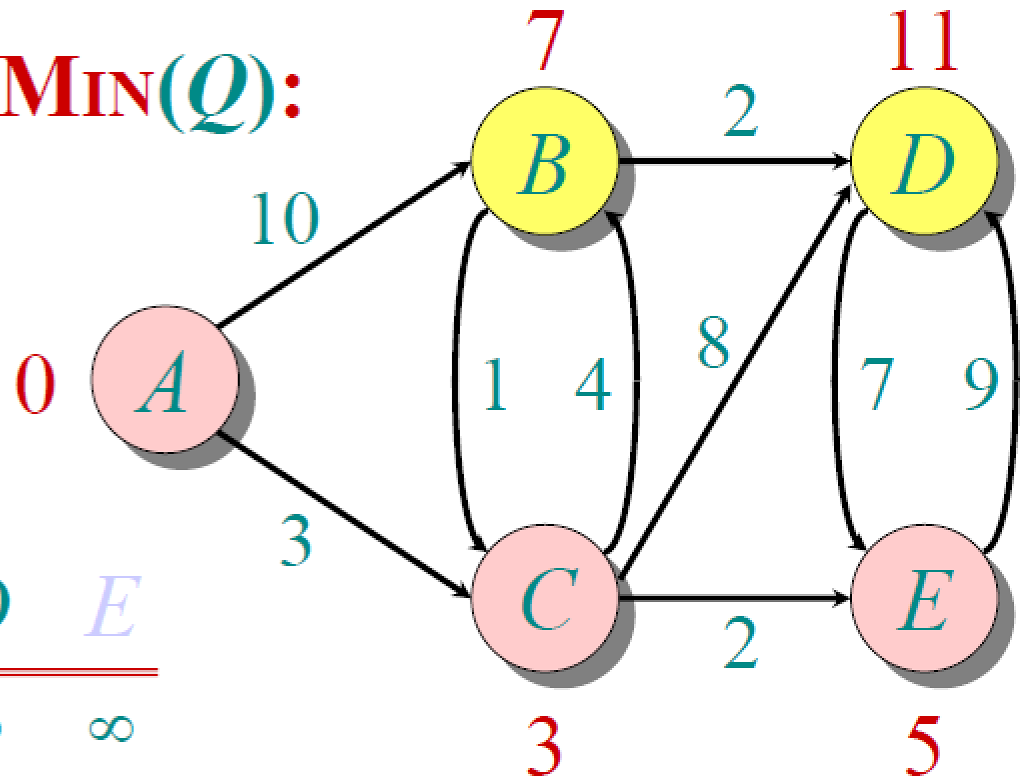


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

Example of Dijkstra's algorithm

"E" $\leftarrow$ **EXTRACT-MIN(Q):**

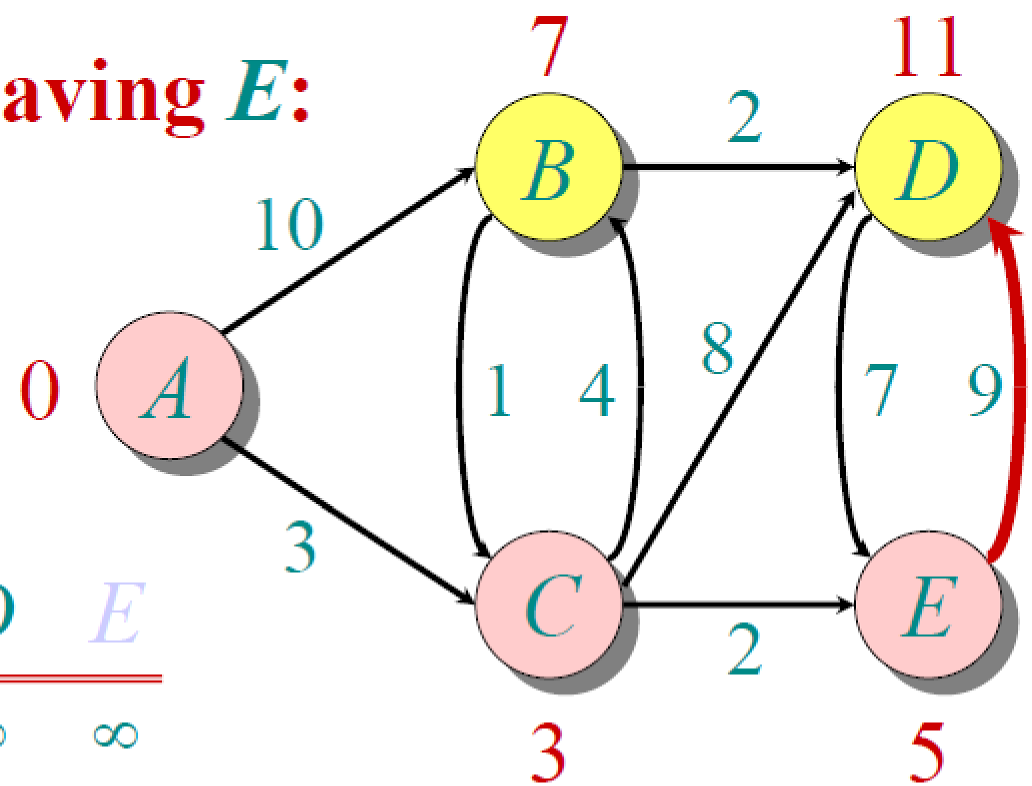


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

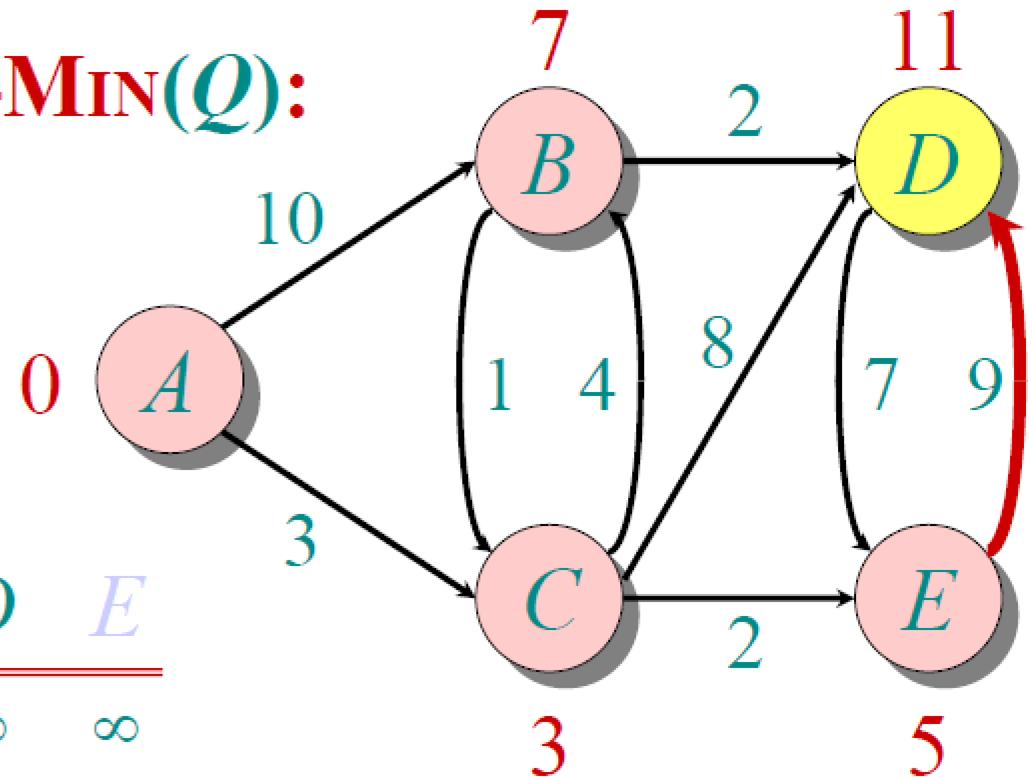


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Example of Dijkstra's algorithm

"B" ← **EXTRACT-MIN(Q):**

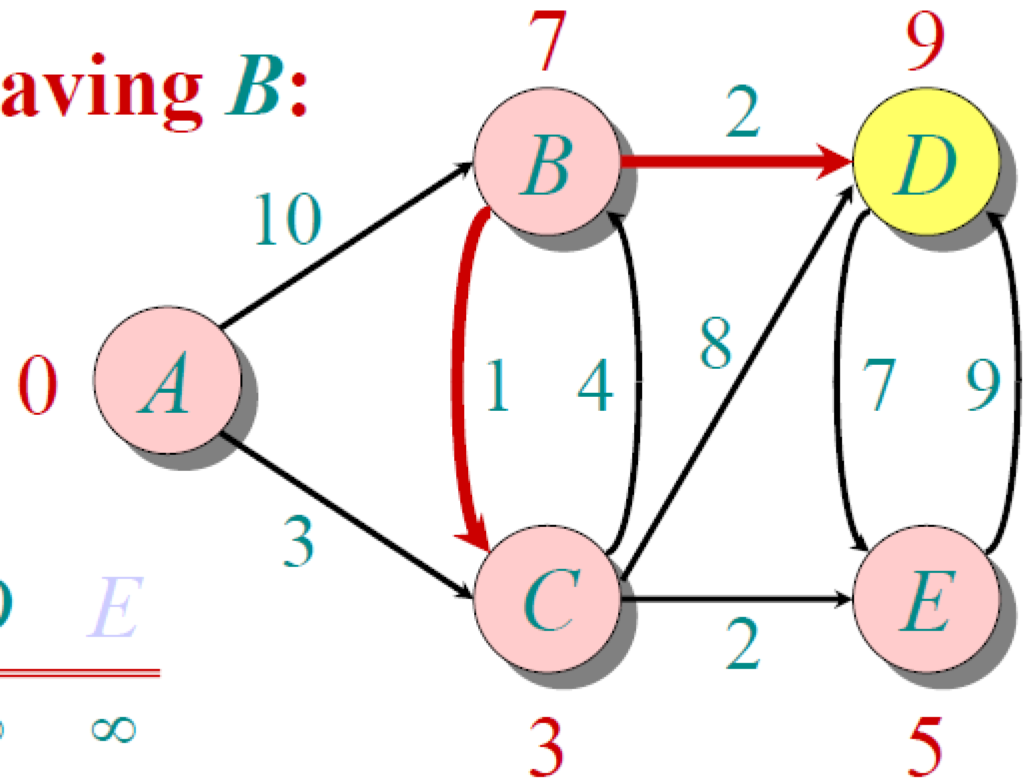


Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	

S: { A, C, E, B }

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

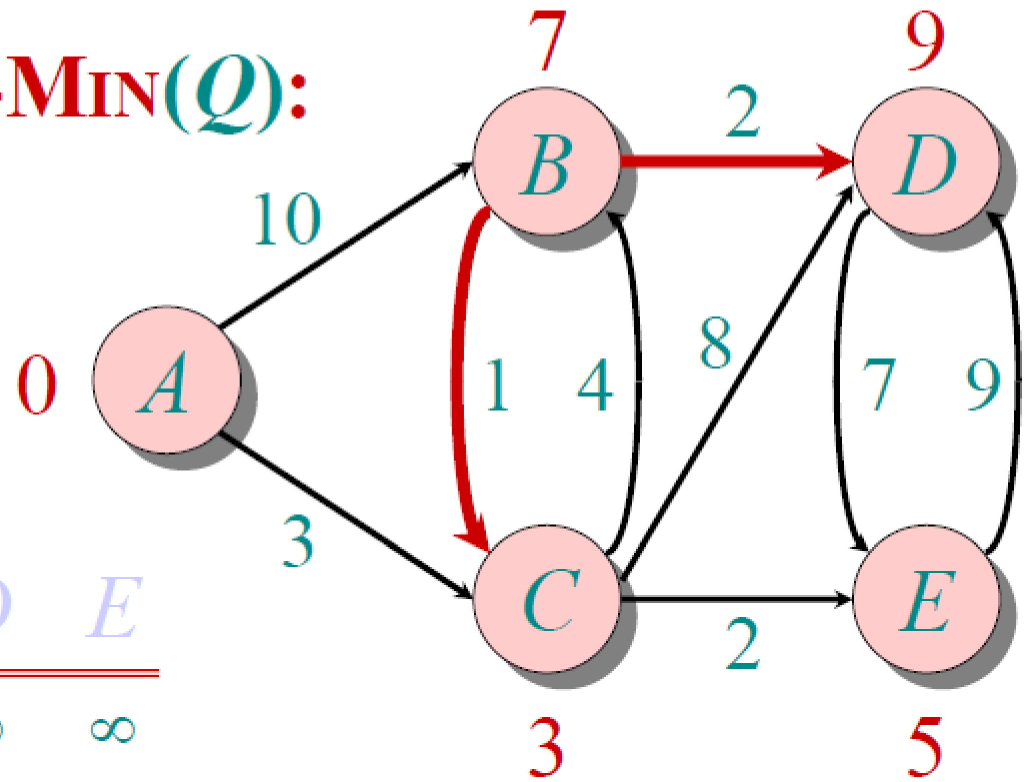


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: $\{ A, C, E, B \}$

Example of Dijkstra's algorithm

"D" $\leftarrow$ EXTRACT-MIN(Q):



Q:

A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

S: { A, C, E, B, D }

Analysis of Dijkstra

$$
\begin{array}{c} |V| \\ \text{times} \end{array}
\left\{
\begin{array}{l}
\quad\quad\quad\quad \textbf{while } Q \neq \emptyset \\
\quad\quad\quad\quad\quad \textbf{do } u \leftarrow \mathrm{EXTRACT\text{-}MIN}(Q) \\
\quad\quad\quad\quad\quad\quad S \leftarrow S \cup \{u\} \\
\begin{array}{c} \mathit{degree}(u) \\ \text{times} \end{array}
\left\{
\begin{array}{l}
\textbf{for each } v \in \mathit{Adj}[u] \\
\quad \textbf{do if } d[v] > d[u] + w(u, v) \\
\quad\quad \textbf{then } d[v] \leftarrow d[u] + w(u, v)
\end{array}
\right.
\end{array}
\right.
$$

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

Time $= \Theta(V \cdot T_{\mathrm{EXTRACT\text{-}MIN}} + E \cdot T_{\mathrm{DECREASE\text{-}KEY}})$

Note: Same as the analysis of Prim's MST algorithm.

Analysis of Dijkstra

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

assume we keep
Q as an array

$$= \Theta(V)\cdot \quad O(V) \quad + O(V^2)\cdot \quad O(1)$$

$$= O(V^2)$$

Analysis of Dijkstra

When it is dense graph, implementing priority queue as an array is OK since the best we can get is $O(V^2)$.

But when the graph is sparse, $E << V^2$, then use binary heap for priority queue of vertices:

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

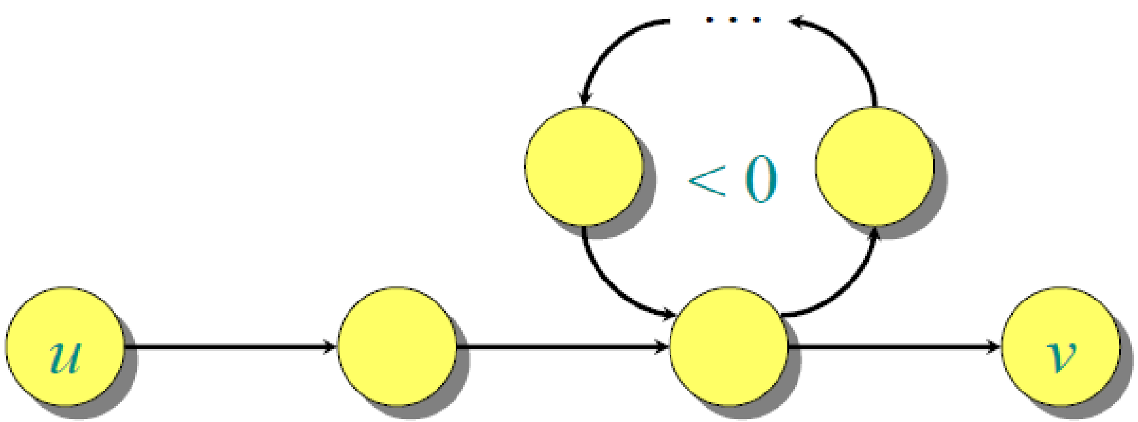
$$= \Theta(V)\cdot \underbrace{\Theta(\log V)}_{\substack{Q \text{ is a binary heap.} \\ \text{Reorganization} \\ \text{after deleting min}}} + \underbrace{\Theta(E)\cdot}\ \underbrace{\Theta(\log V)}_{\substack{\text{Reorganization} \\ \text{of binary heap}}}$$

$$= \Theta(E \log V)$$

Bellman-Ford algorithm

Finds all shortest-path lengths from a ***source*** $s \in V$ to all $v \in V$
OR
reports that a negative-weight cycle exists.

Recall negative-weight cycles:



Bellman-Ford algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$

initialization

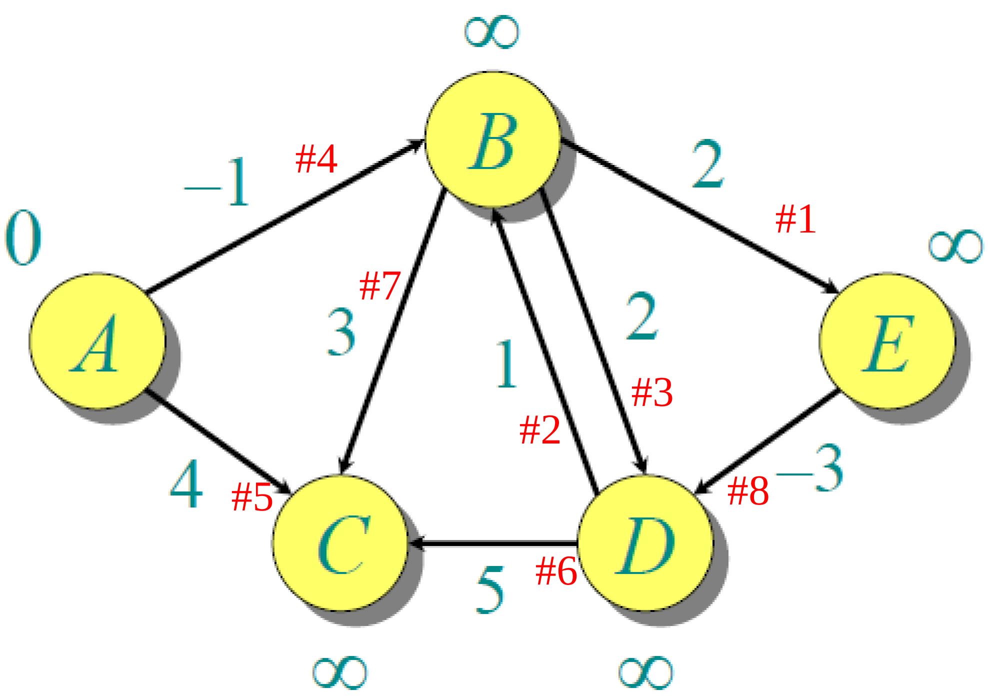
for $i \leftarrow 1$ **to** $|V| - 1$
    do for each edge $(u, v) \in E$
        do if $d[v] > d[u] + w(u, v)$
            then $d[v] \leftarrow d[u] + w(u, v)$

relaxation step

for each edge $(u, v) \in E$
    do if $d[v] > d[u] + w(u, v)$
        then report that a negative-weight cycle exists

At the end, $d[v] = \delta(s, v)$. Running time = $O(VE)$.

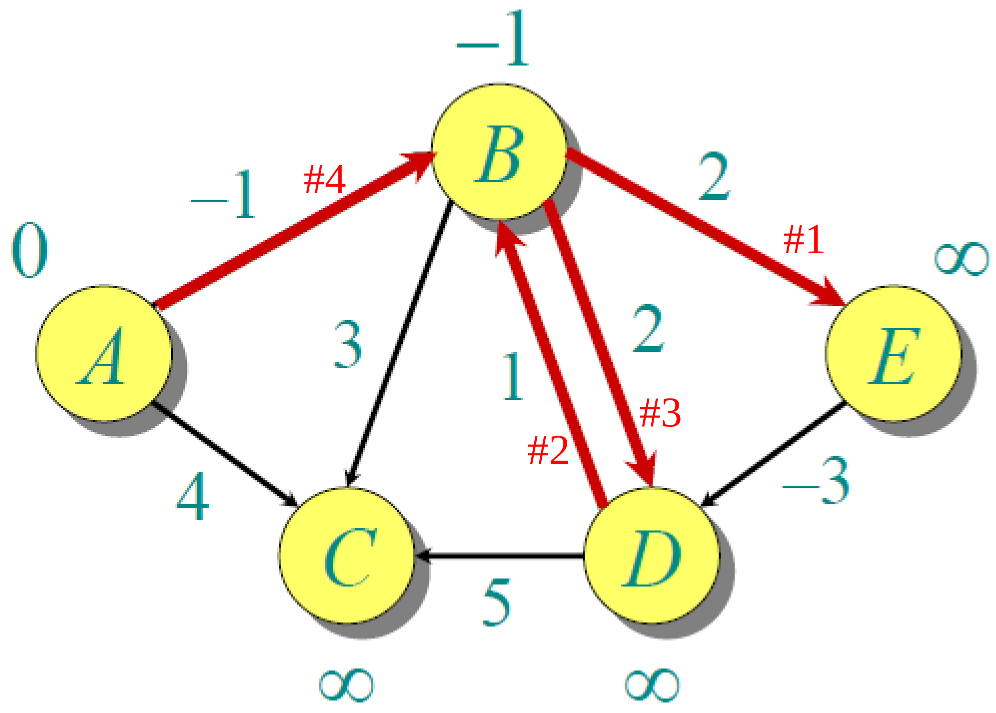
Example of Bellman-Ford



A	B	C	D	E
0	∞	∞	∞	∞

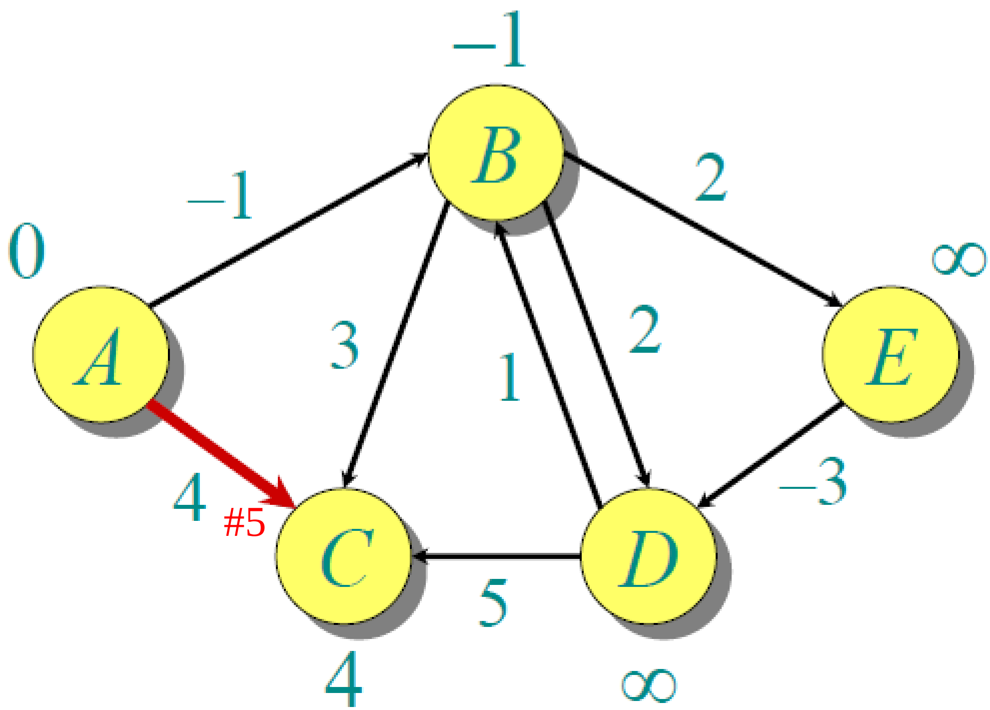
Note: Order of edges has no importance.

Example of Bellman-Ford



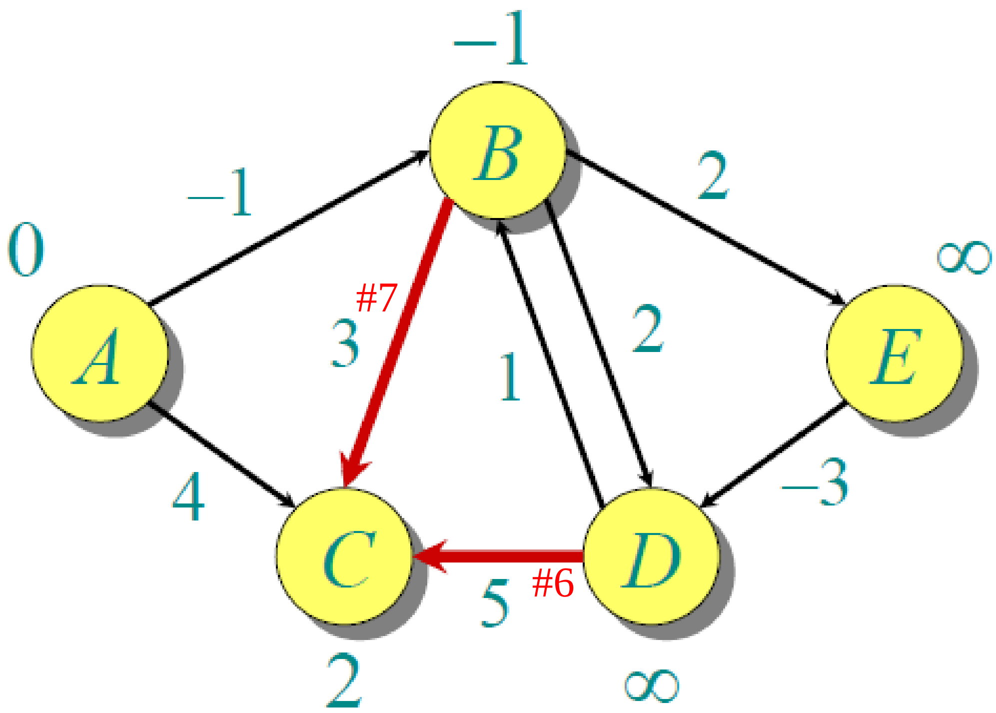
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞

Example of Bellman-Ford



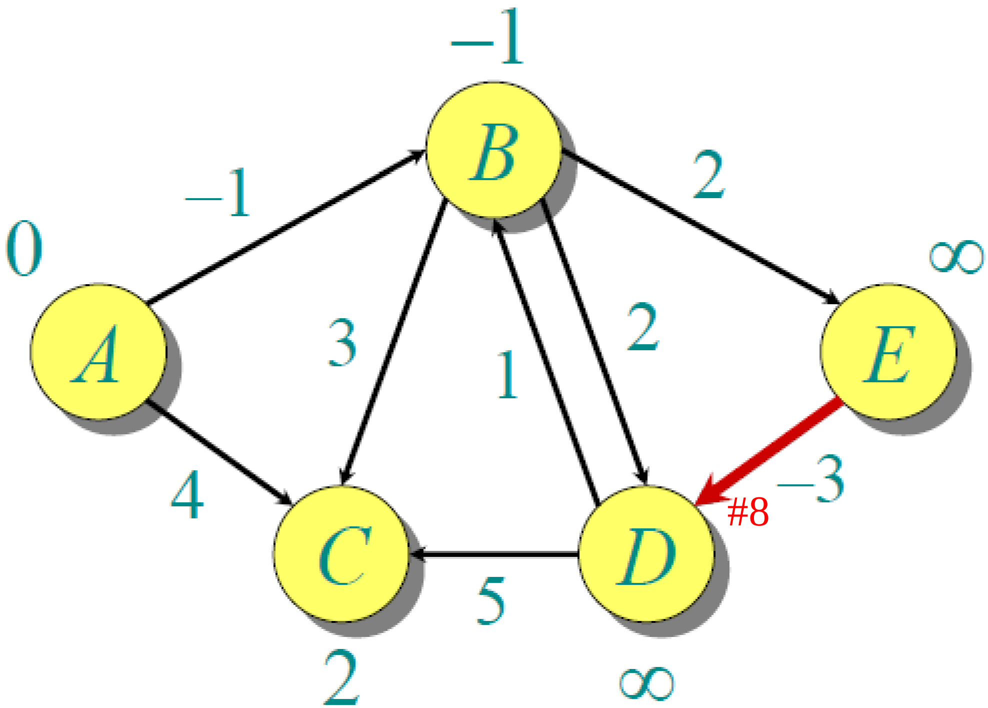
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞

Example of Bellman-Ford



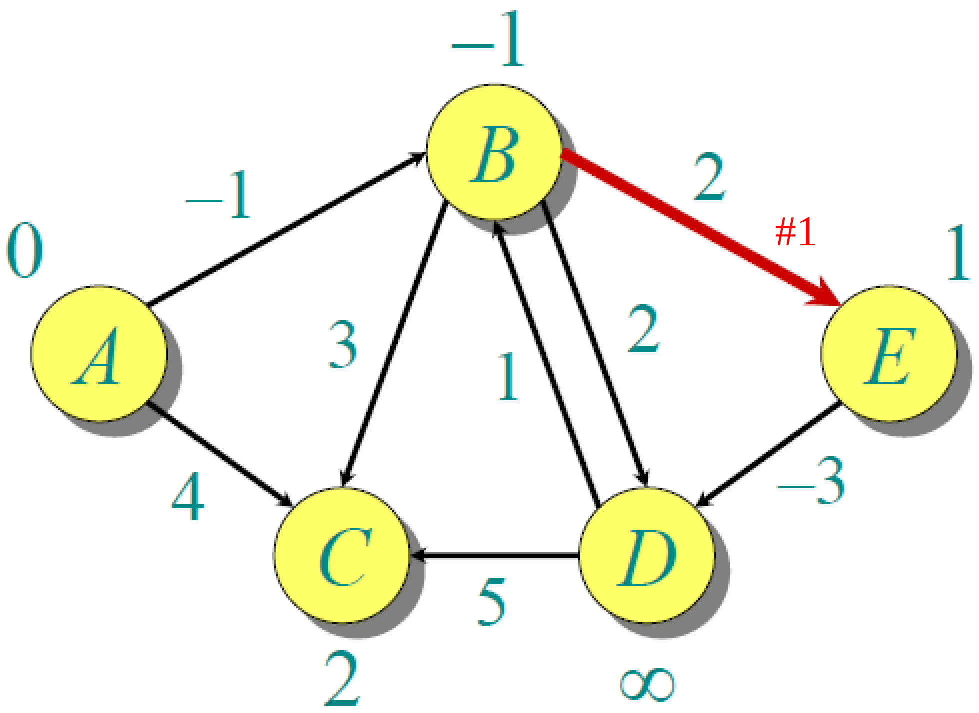
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞

Example of Bellman-Ford



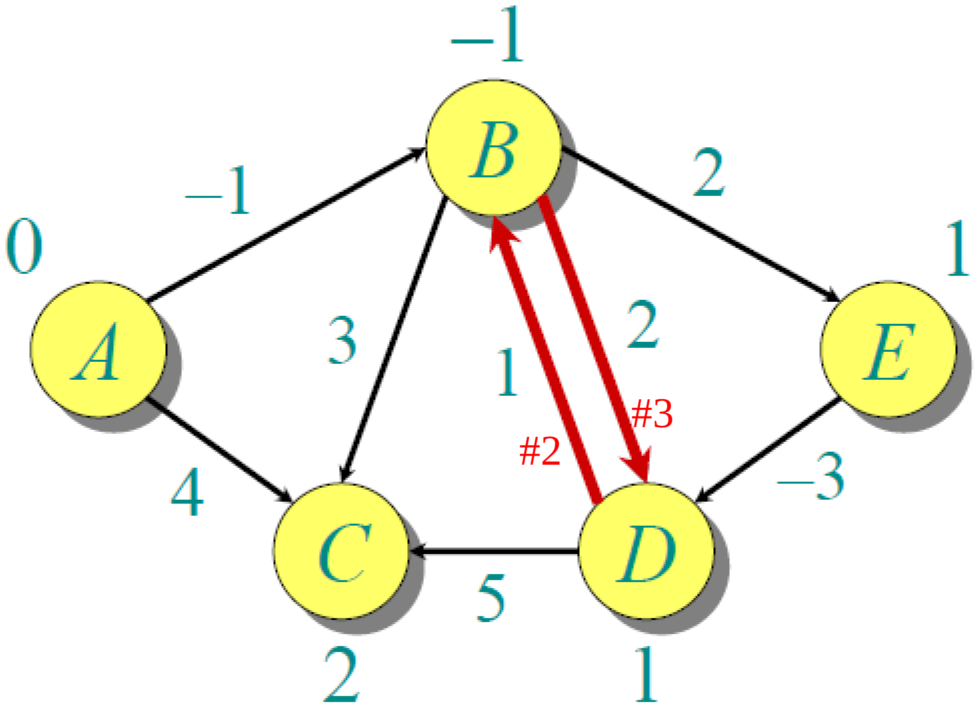
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞

Example of Bellman-Ford



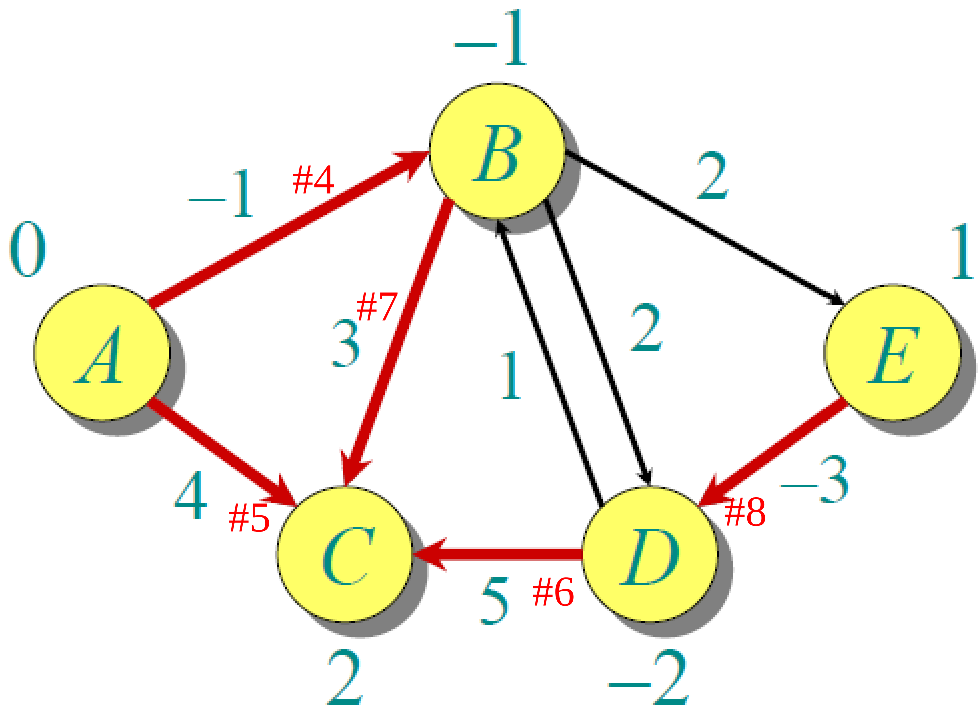
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1

Example of Bellman-Ford



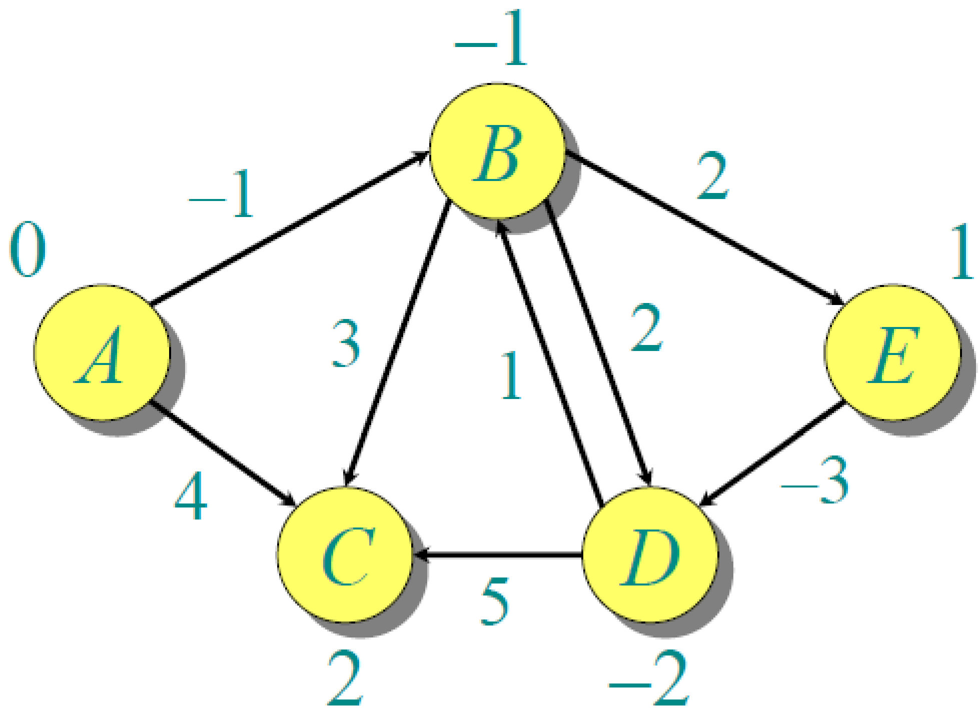
A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1

Example of Bellman-Ford



A	B	C	D	E
0	∞	∞	∞	∞
0	–1	∞	∞	∞
0	–1	4	∞	∞
0	–1	2	∞	∞
0	–1	2	∞	1
0	–1	2	1	1
0	–1	2	–2	1

Example of Bellman-Ford



A	B	C	D	E	
0	∞	∞	∞	∞	
0	−1	∞	∞	∞	i=1
0	−1	4	∞	∞	
0	−1	2	∞	∞	
0	−1	2	∞	1	
0	−1	2	1	1	i=2
0	−1	2	−2	1	

There are two more rounds
here since |V|-1 is 4,
but result does not change.

Correctness

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

Corollary. If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle in G reachable from s.

Conclusion: $O(V \cdot E)$ is slower than Dijkstra's, but we can report if negative weight exists.

The End

Textbook Section 24.1, 24.3.