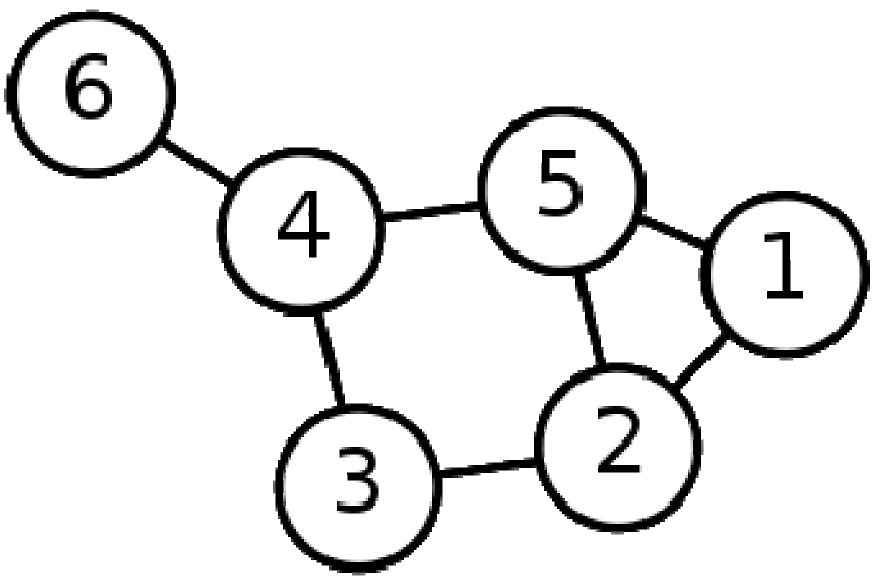
Dijkistra’s algorithm

Single-Source Shortest Path Problem

<u>Single-Source Shortest Path Problem</u> - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

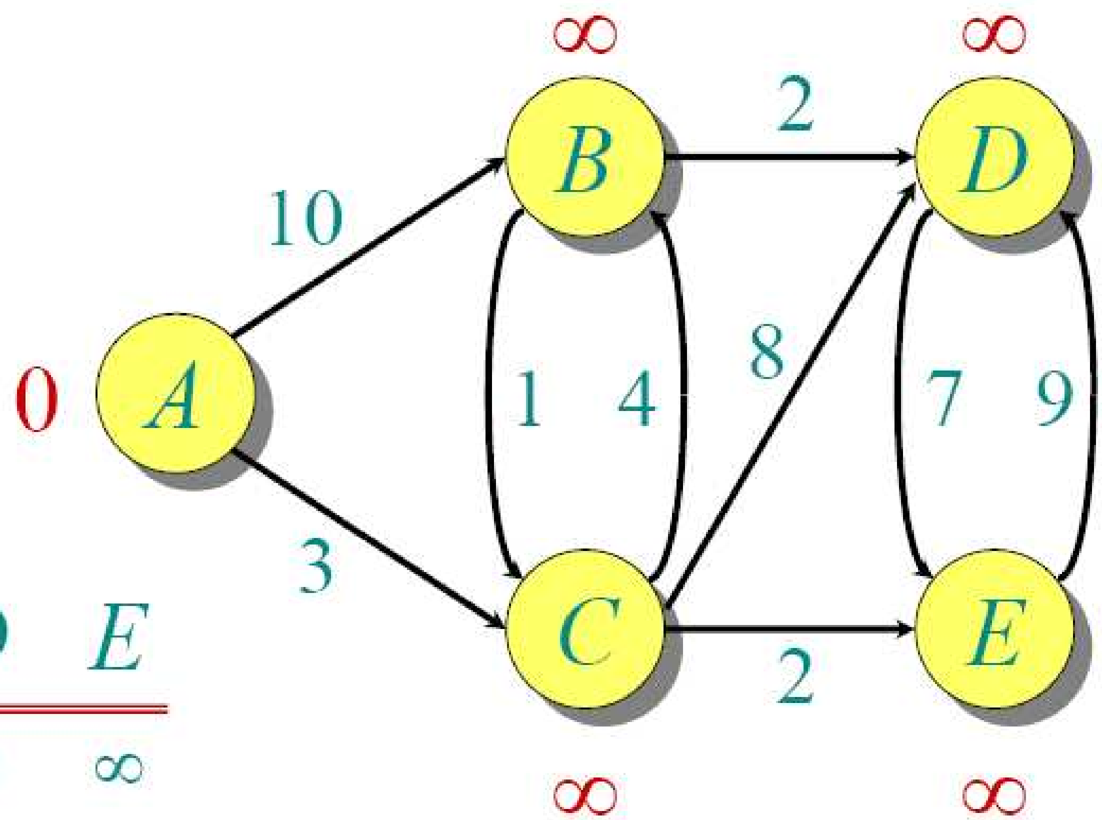
Works on both directed and undirected graphs.

Input: Weighted graph G={E,V} and source vertex $v \in V$, such that all edge weights are nonnegative

Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

Example

Initialize:



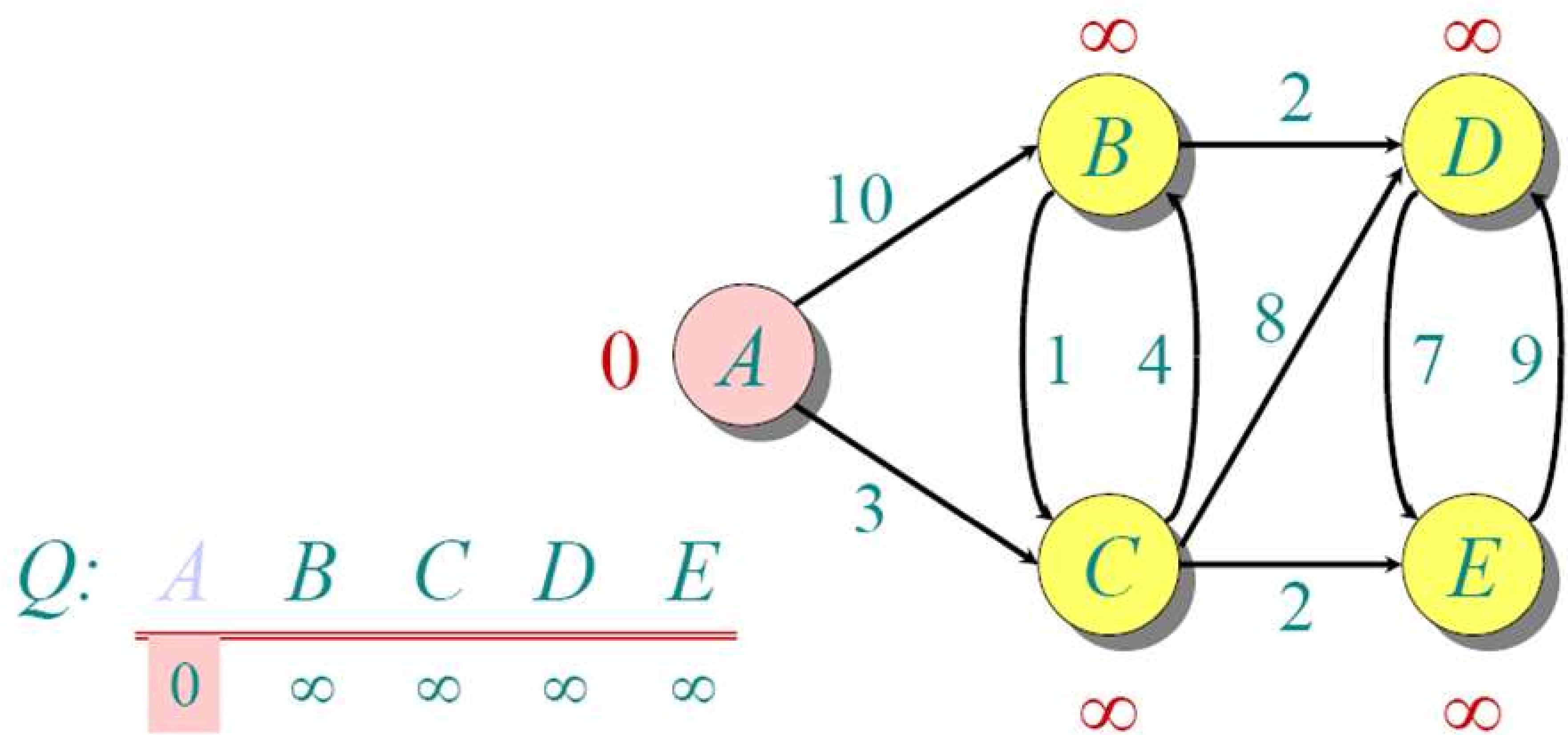
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: $\{\}$

Pre

A	B	C	D	E
-	-	-	-	-

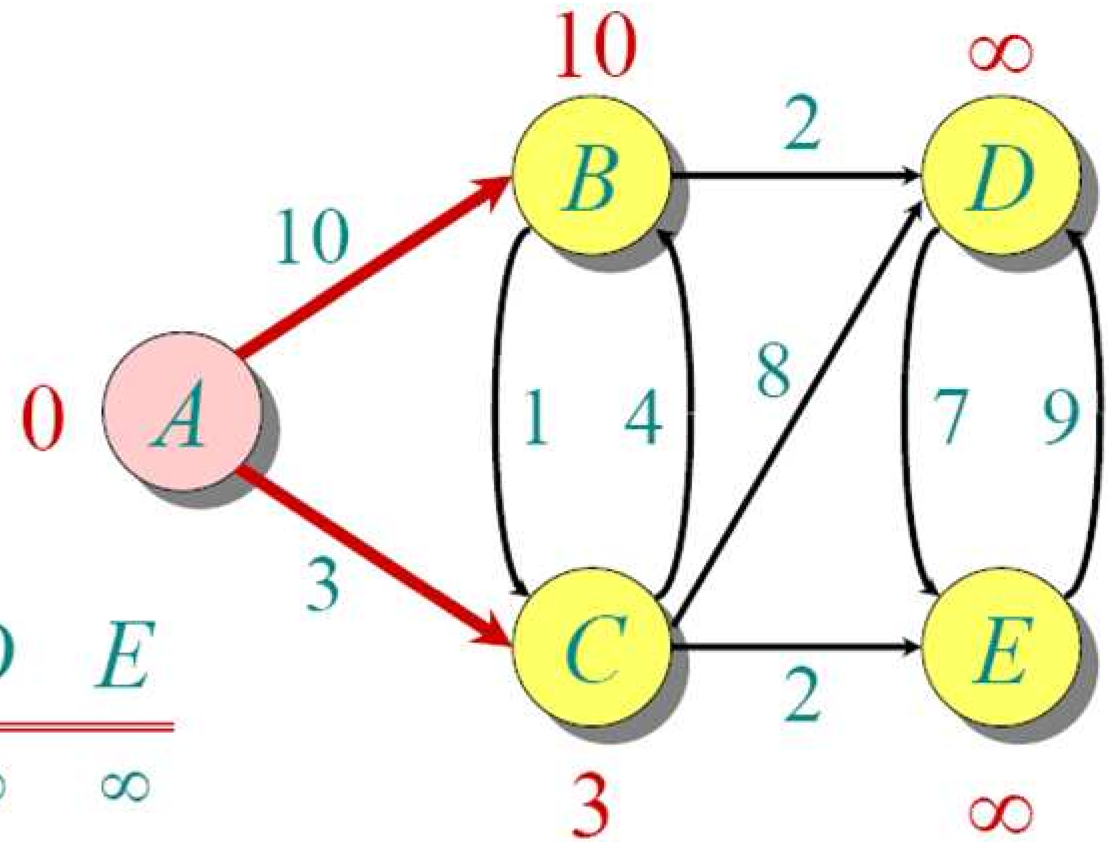
Example (contd…)



Pre

A	B	C	D	E
-	-	-	-	-

Example (contd…)



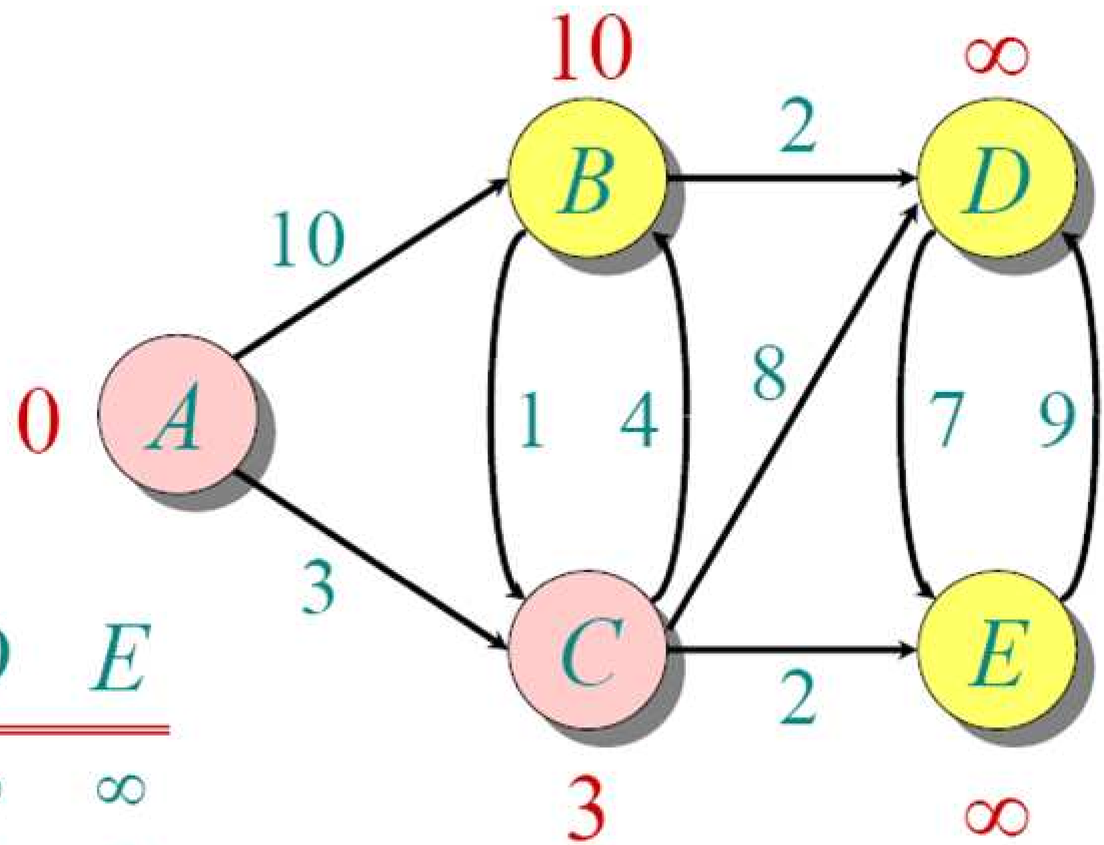
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Pre

A	B	C	D	E
A	A	A	-	-

Example (contd…)



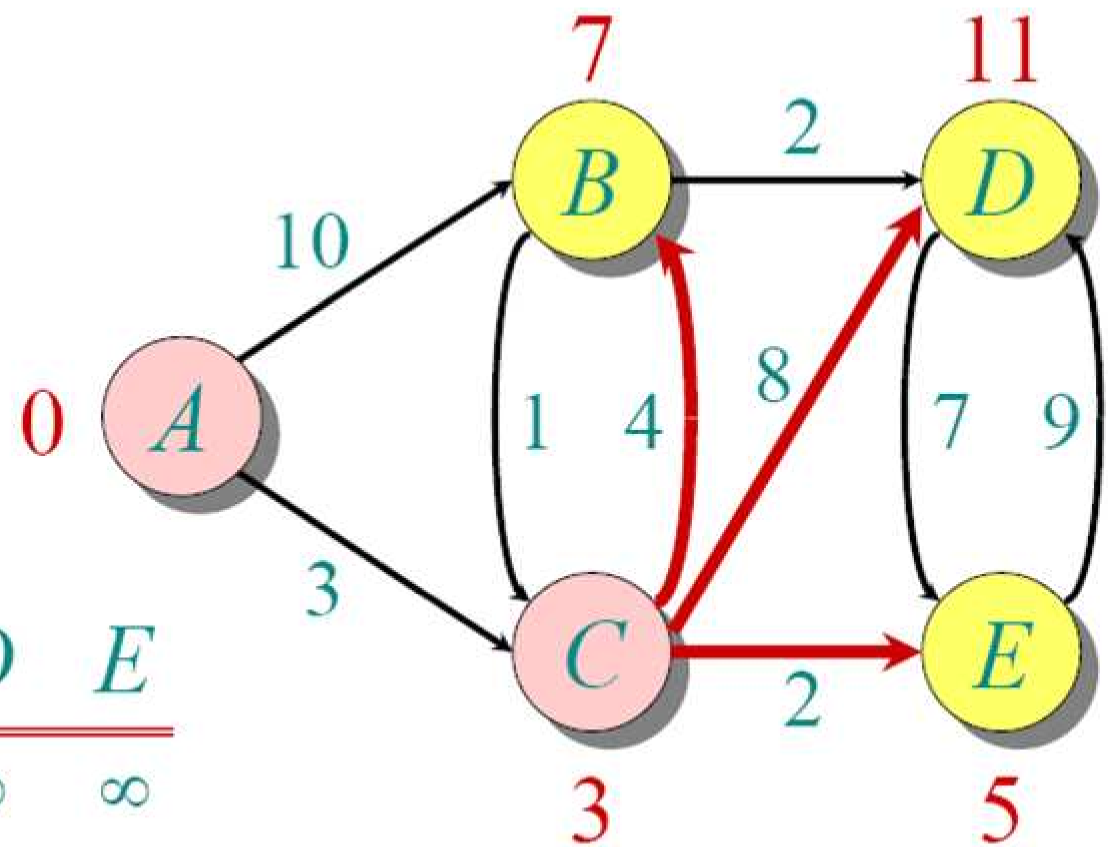
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A*, *C* }

Pre

A	B	C	D	E
A	A	A	-	-

Dijkstra Animated Example



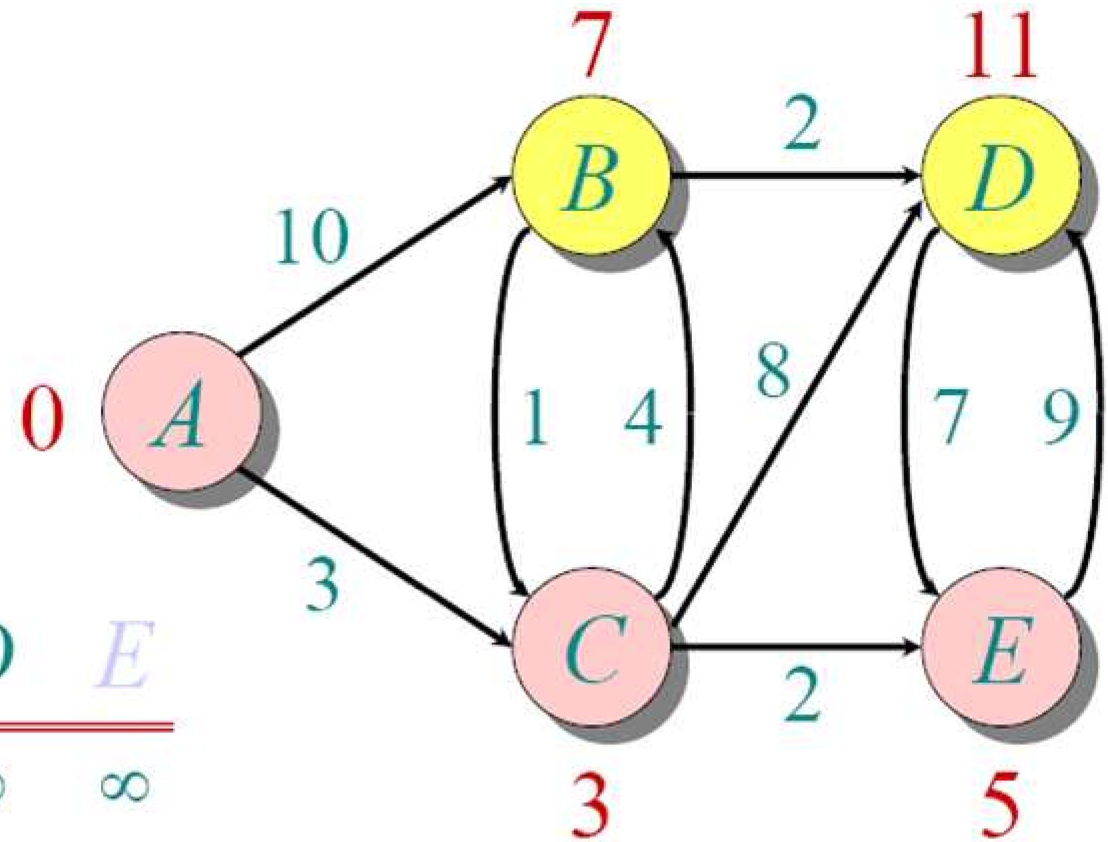
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

Pre

A	B	C	D	E
A	C	A	C	C

Example (contd…)



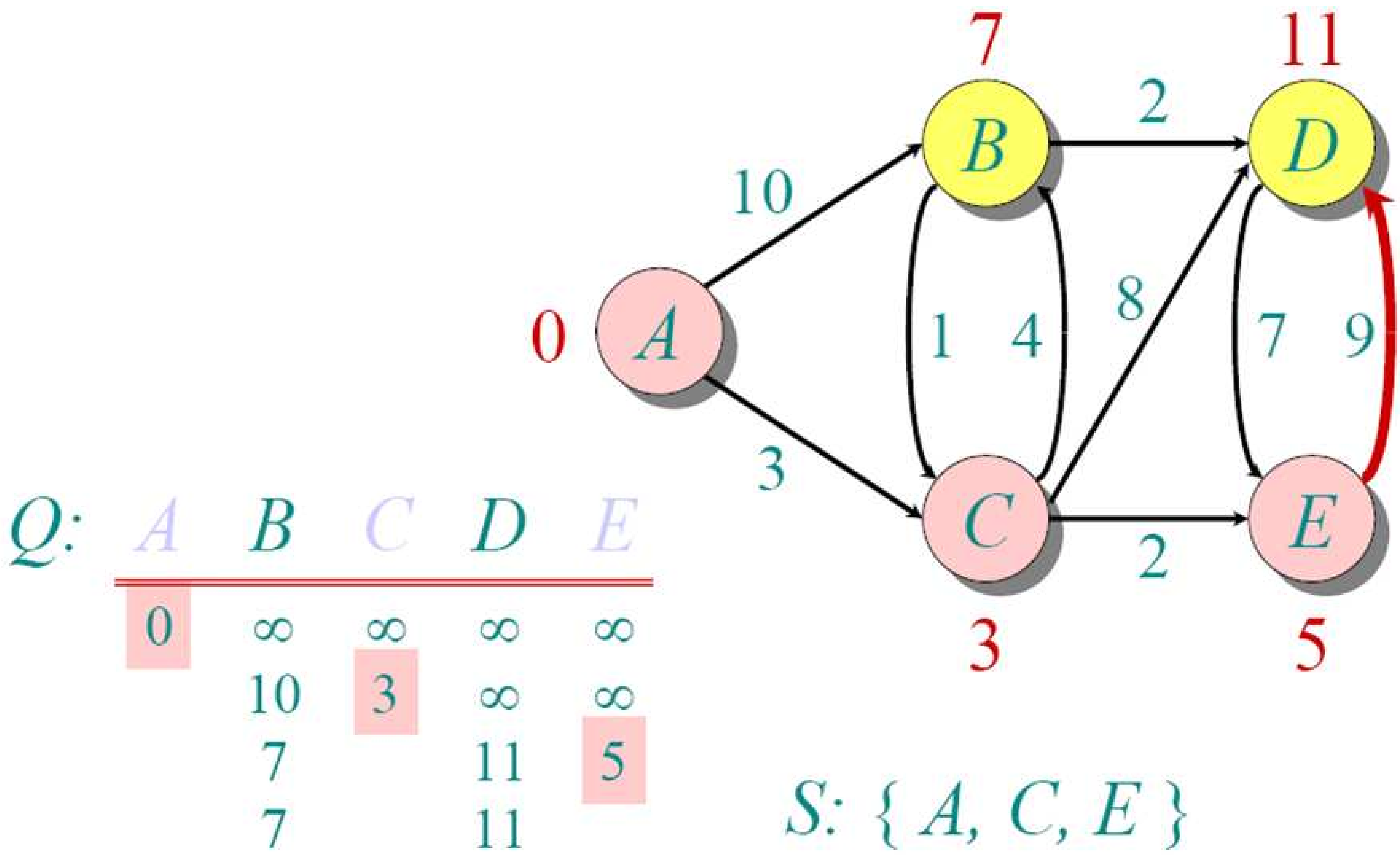
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

Pre

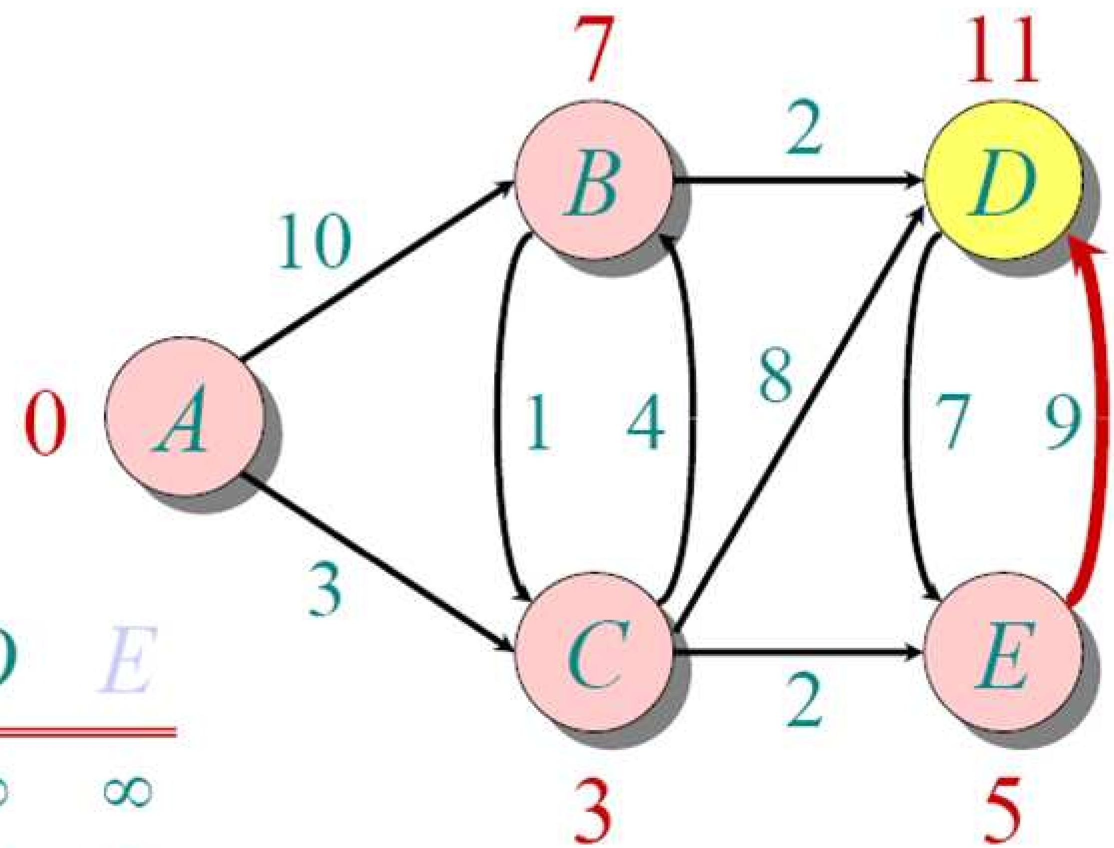
A	B	C	D	E
A	C	A	C	C

Example (contd…)



Pre	A	B	C	D	E
	A	C	A	C	C

Example (contd…)



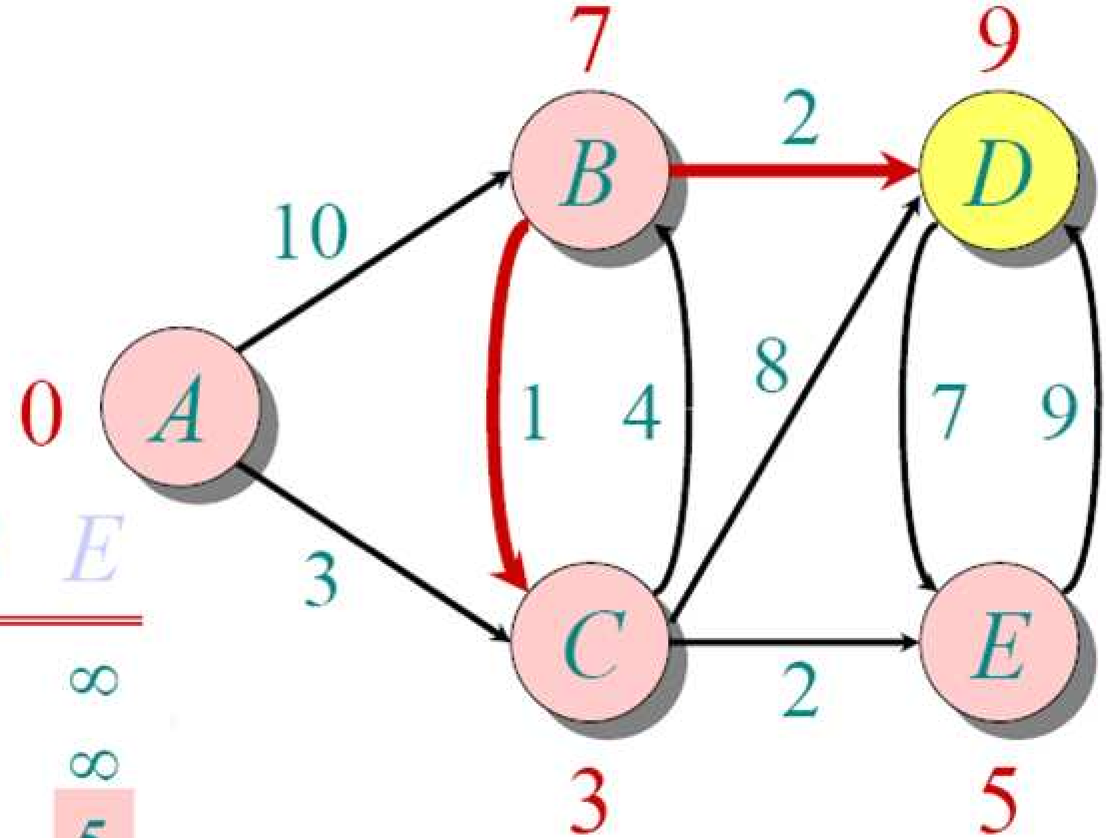
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Pre

A	B	C	D	E
A	C	A	C	C

Dijkstra Animated Example



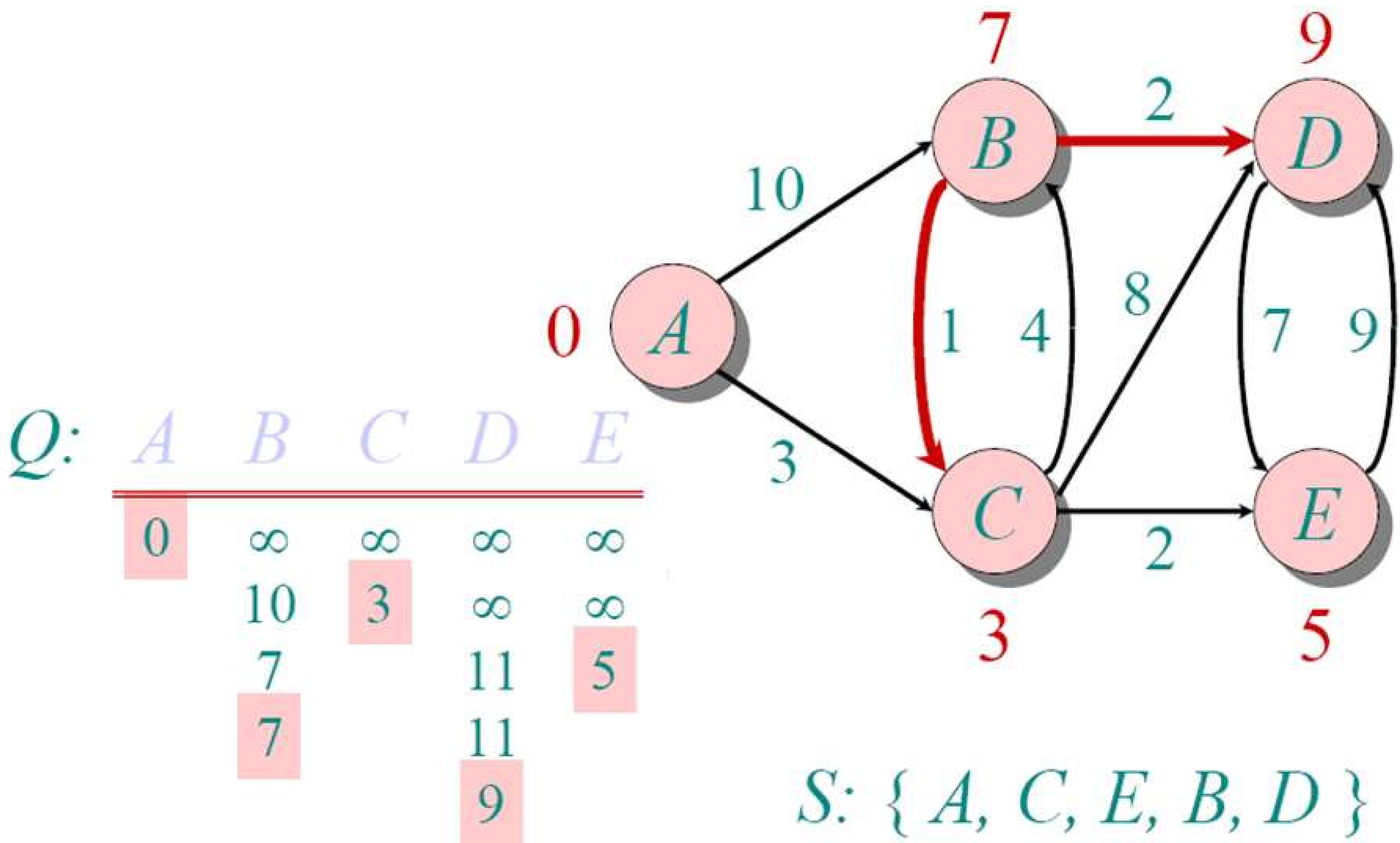
Q: A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

S: { A, C, E, B }

Pre

A	B	C	D	E
A	C	A	B	C

Example (contd…)



Pre

A	B	C	D	E
A	C	A	B	C

Dijkstra's algorithm - Pseudocode

```
Dijkstra(G,s)

dist[s] ←o
for  all v ∈ V–{s}
        do  dist[v] ←∞                (set all other distances to infinity)
               Pre[v] ← '-'
S←Ø                                   (S, the set of visited vertices)
Q←V [G]                               (Q, initially contains all nodes )
while Q ≠Ø                            (while the queue is not empty)
do   u ← mindist(Q, dist)             (select element of Q  with min. dist )
      S←S∪{u}                         (add u to list of visited vertices)
       for all v ∈ Adj[u]
             do  if   dist[v] > dist[u] + w(u, v)
                        then      d[v] ←d[u] + w(u, v)
                                  Pre[v] ← u
return S, dist
```

Implementations and Running Times

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of

O(|V|^2)

For sparse graphs, or graphs with very few edges and many nodes, it can be implemented more efficiently storing the graph in an adjacency list using a binary heap or priority queue. This will produce a running time of

$O((|E|+|V|) \log |V|)$

CONCLUSION

- Dijkstra’s algorithm calculates the shortest path to every vertex.
- However, it is about as computationally expensive to calculate the shortest path from vertex u to every vertex using Dijkstra’s as it is to calculate the shortest path to some particular vertex v.
- Therefore, anytime we want to know the optimal path to some other vertex from a determined origin, we can use Dijkstra’s algorithm.