Single-Source Shortest Path Algorithm

MD. MUKTAR HOSSAIN

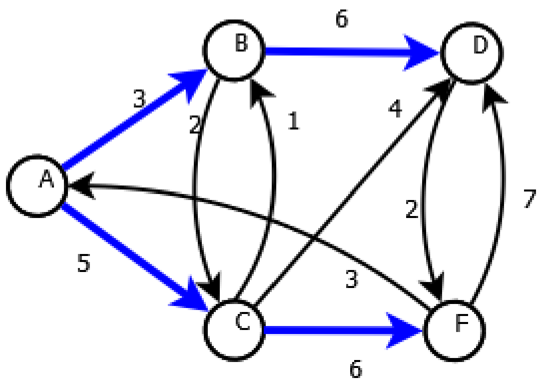
LECTURER

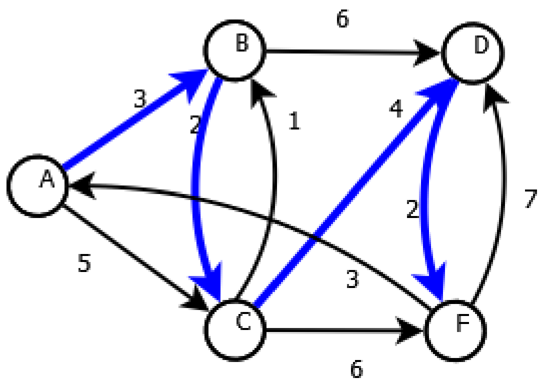
DEPT. OF CSE

VARENDRA UNIVERSITY

Single-Source Shortest Path Problem

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





Relaxation

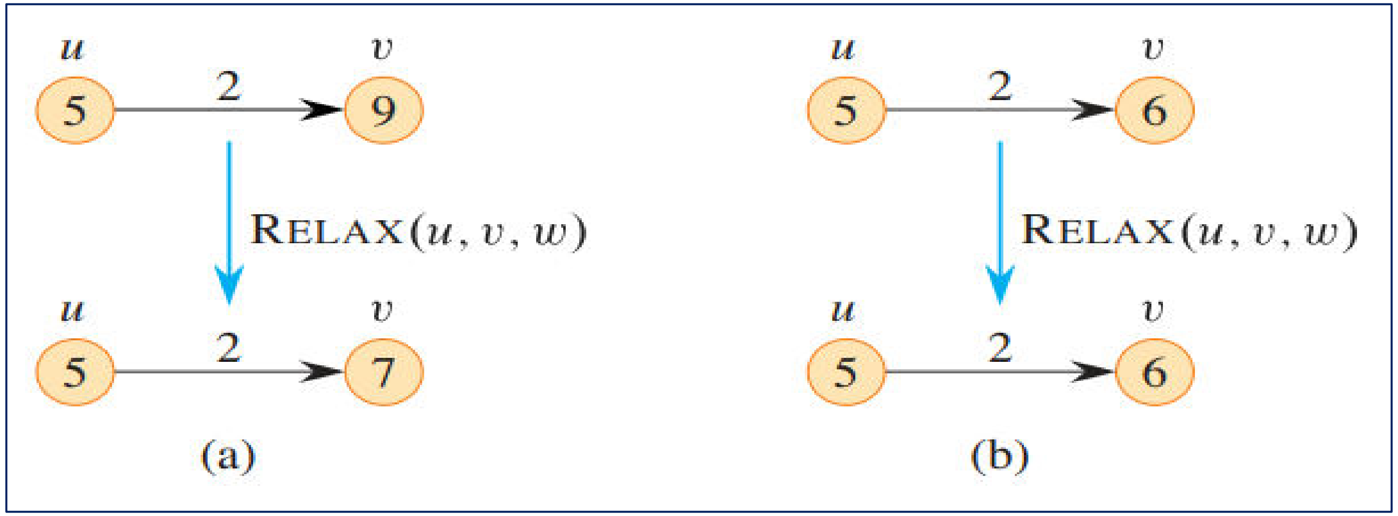
For an edge ***(u,v)*** with weight ***w***, relaxation checks if the known shortest path to ***v*** can be improved by going through ***u***.

RELAX(u, v, w)

if $v.d > u.d + w(u, v)$

  $v.d = u.d + w(u, v)$

  $v.\pi = u$



Introduction

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

Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

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

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

Applications of Dijkstra's Algorithm

1. **Network Routing Protocols**:
   - ✓ Used in **OSPF (Open Shortest Path First)** and **IS-IS** routing algorithms to compute the shortest path tree for routing packets.
2. **Mapping and GPS Systems**:
   - ✓ Calculating the quickest route between two locations using road network data.
3. **Game Development**:
   - ✓ Pathfinding for characters or AI agents (though A* is often preferred for efficiency in certain cases).
4. **Telecommunications**:
   - ✓ Optimizing data flow through networks by finding the least-cost path.
5. **Compiler Design**:
   - ✓ Used in **data-flow analysis** and **code optimization** to calculate the shortest distance in control flow graphs.
6. **Robot Motion Planning**:
   - ✓ In robotics, Dijkstra helps robots find the shortest path in a known environment.
7. **Dependency Resolution**:
   - ✓ Used in systems like **package managers** to resolve dependency chains efficiently.

Dijkstra's Algorithm

1. Initialize distances from the source to all nodes as infinity, except the source (0).
2. Use a priority queue (min-heap) to repeatedly select the node with the smallest tentative distance.
3. Update the distances to neighboring nodes if a shorter path is found.
4. Repeat until all nodes are visited or the destination is reached.

```
function Dijkstra(Graph, source):
create a priority queue Q
    for each vertex v in Graph:
        dist[v] := infinity
        prev[v] := undefined
        add v to Q
    dist[source] := 0
```

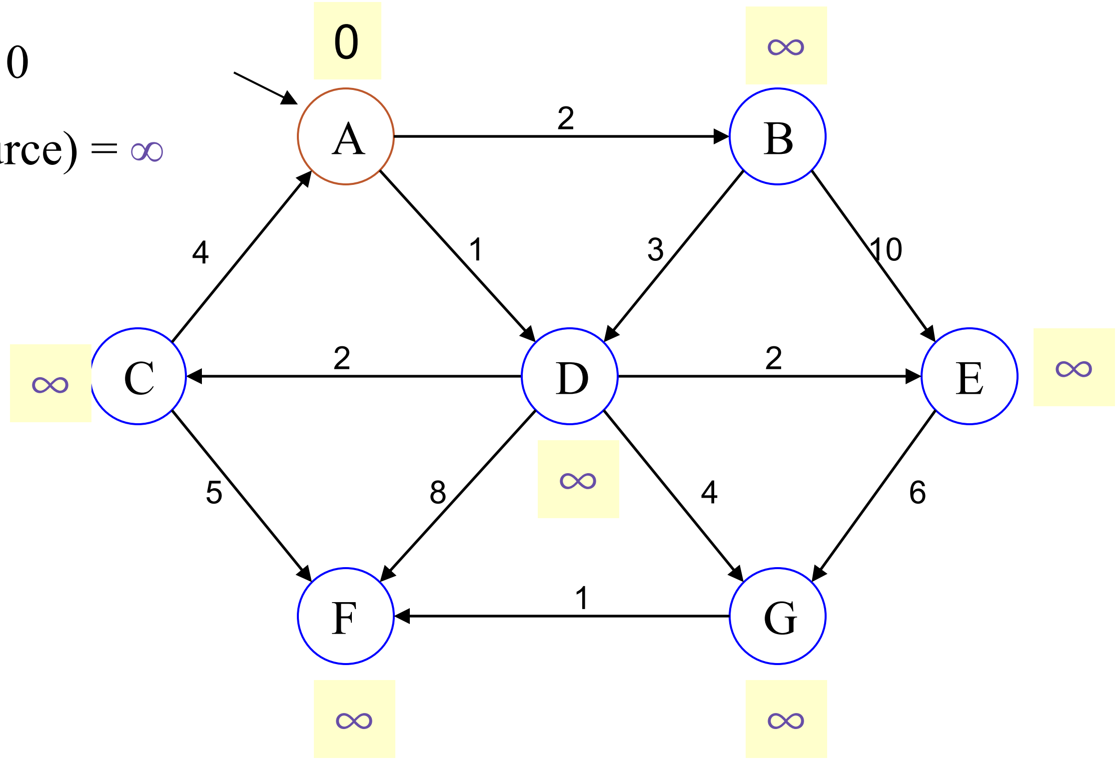
```
while Q is not empty:
     u := vertex in Q with smallest dist[u]
     remove u from Q
     for each neighbor v of u:
         alt := dist[u] + length(u, v)
         if alt < dist[v]:
             dist[v] := alt
             prev[v] := u

    return dist, prev
```

Example: Initialization

Distance(source) = 0

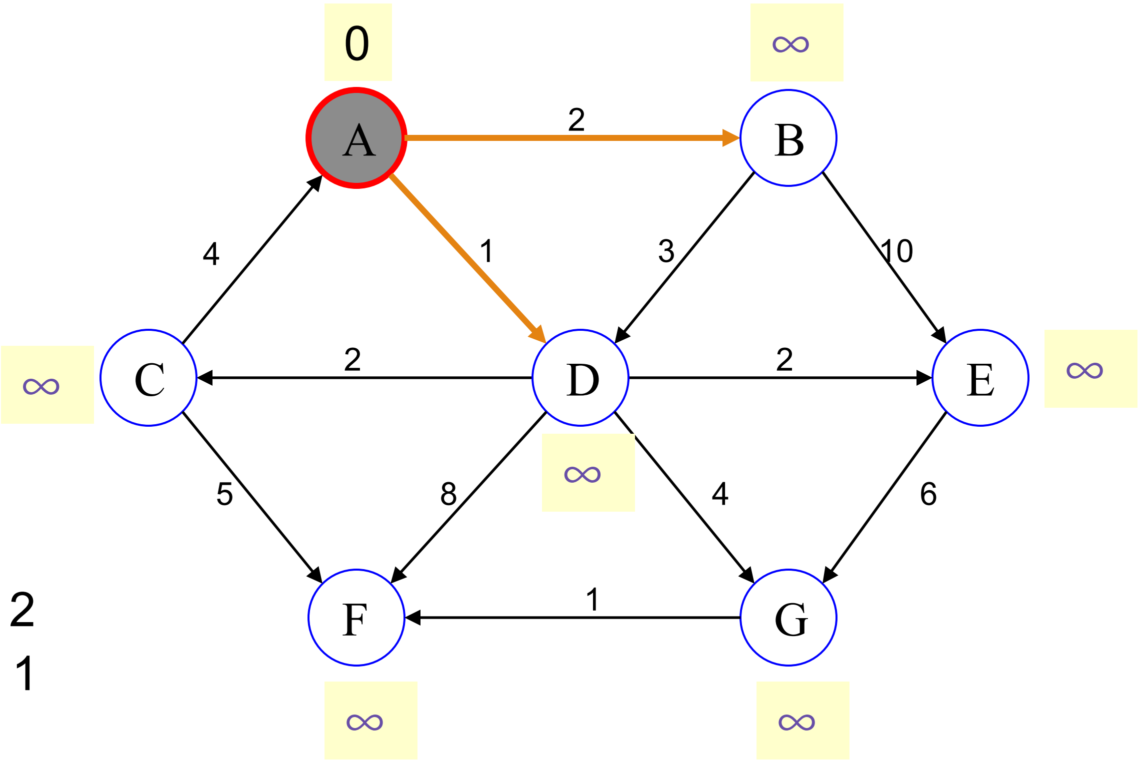
Distance (all vertices but source) = ∞



Node	Distance	Parent
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	

Pick vertex in List with minimum distance.

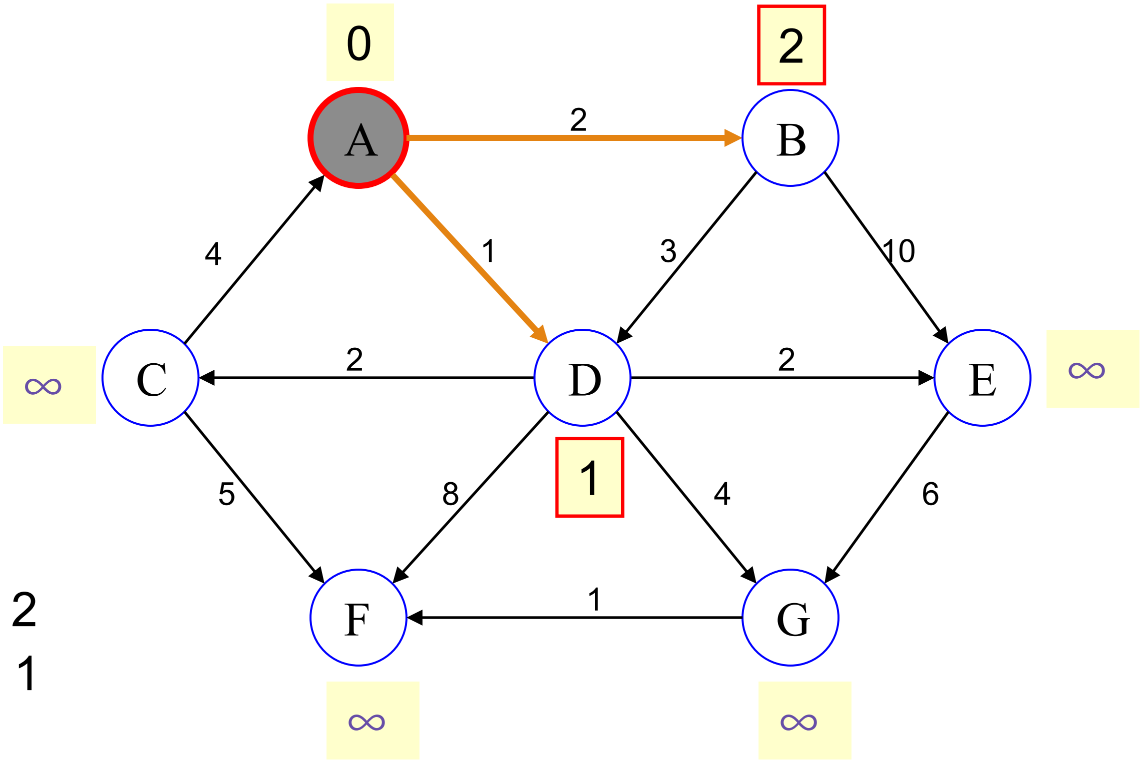
Example: Update neighbors' distance



Distance(B) = 2
Distance(D) = 1

Node	Distance	Parent
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	

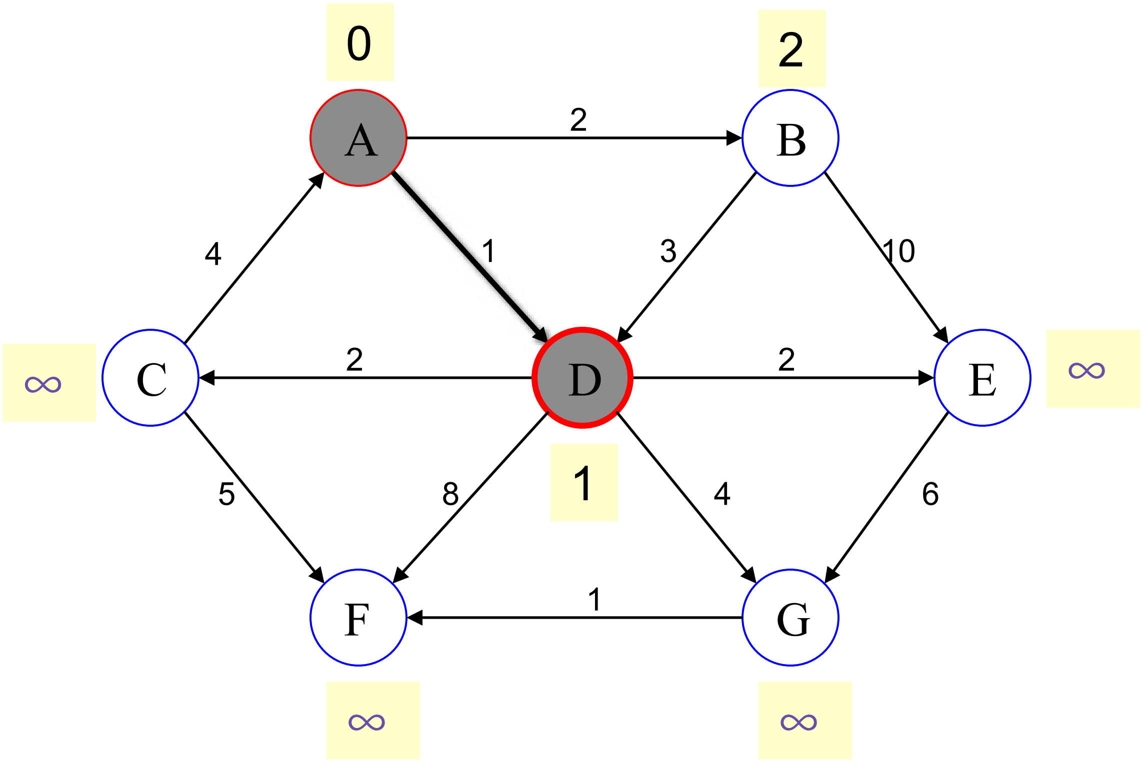
Example: Update neighbors' distance



Node	Distance	Parent
A	0	
B	2	A
C	∞	
D	1	A
E	∞	
F	∞	
G	∞	

Distance(B) = 2
Distance(D) = 1

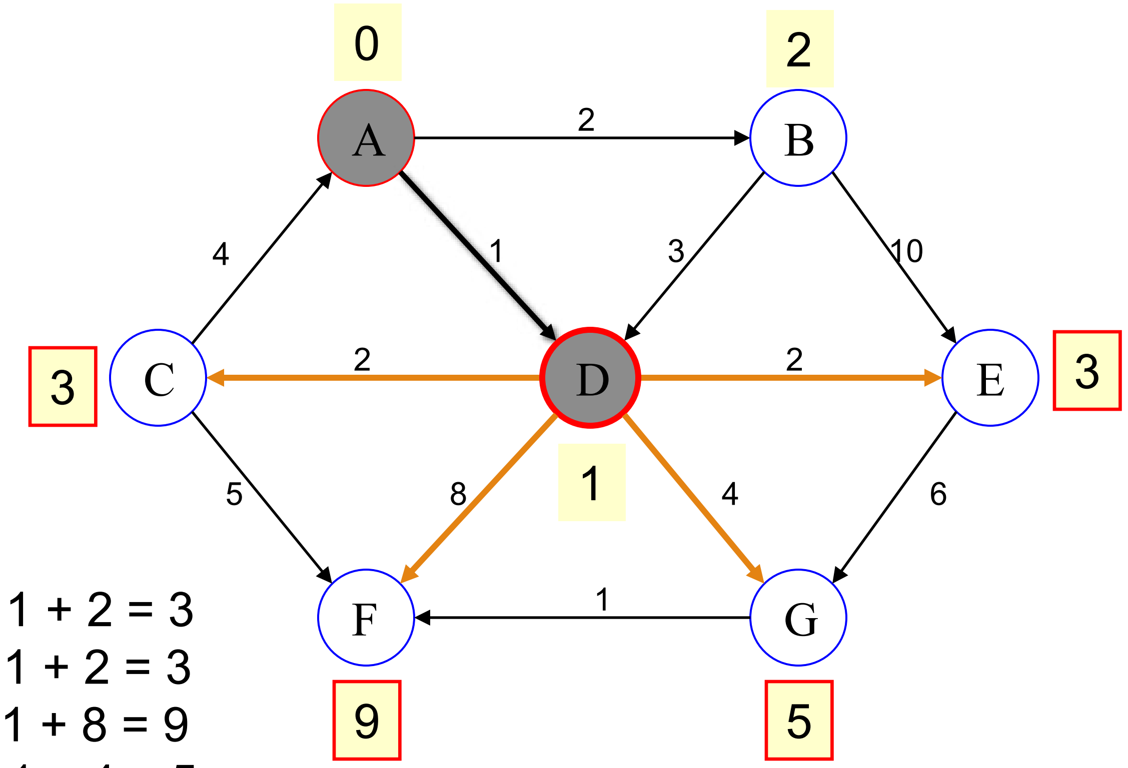
Example: Pick vertex with minimum distance



Node	Distance	Parent
A	0	
B	2	A
C	∞	
D	1	A
E	∞	
F	∞	
G	∞	

Pick vertex in List with minimum distance, i.e., D

Example: Update neighbors

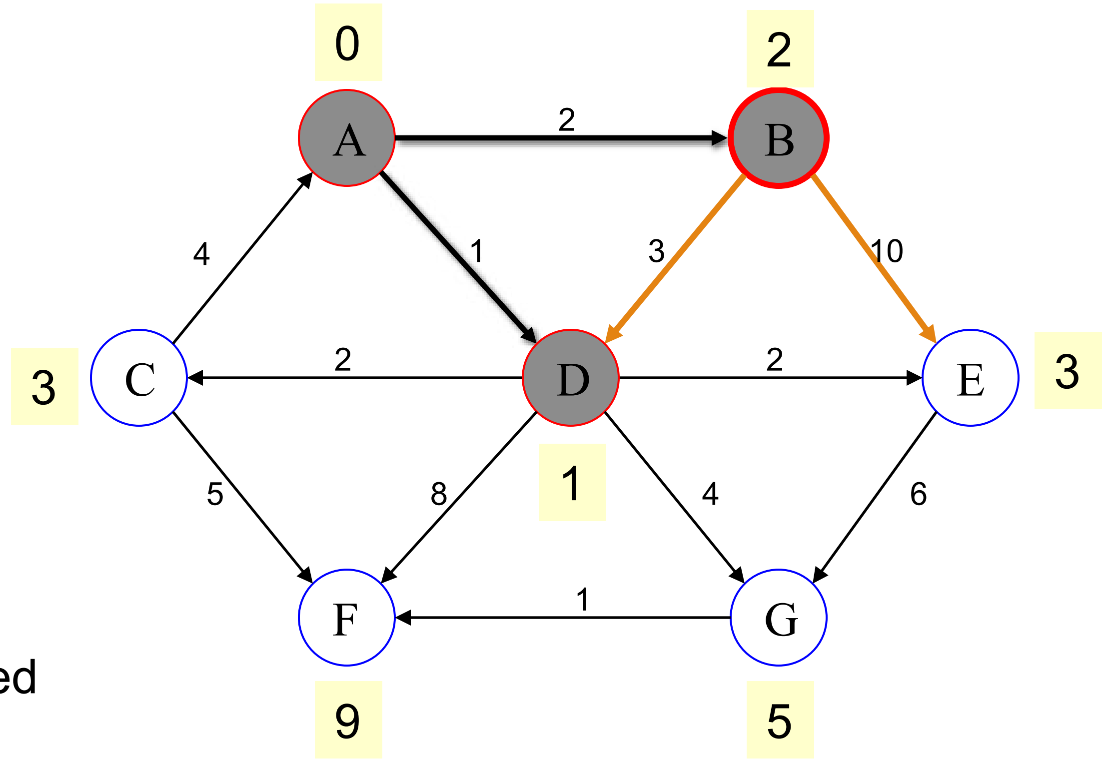


Node	Distance	Parent
A	0	
B	2	A
C	3	D
D	1	A
E	3	D
F	9	D
G	5	D

Distance(C) = 1 + 2 = 3
Distance(E) = 1 + 2 = 3
Distance(F) = 1 + 8 = 9
Distance(G) = 1 + 4 = 5

Example: Continued...

Pick vertex in List with minimum distance (B) and update neighbors

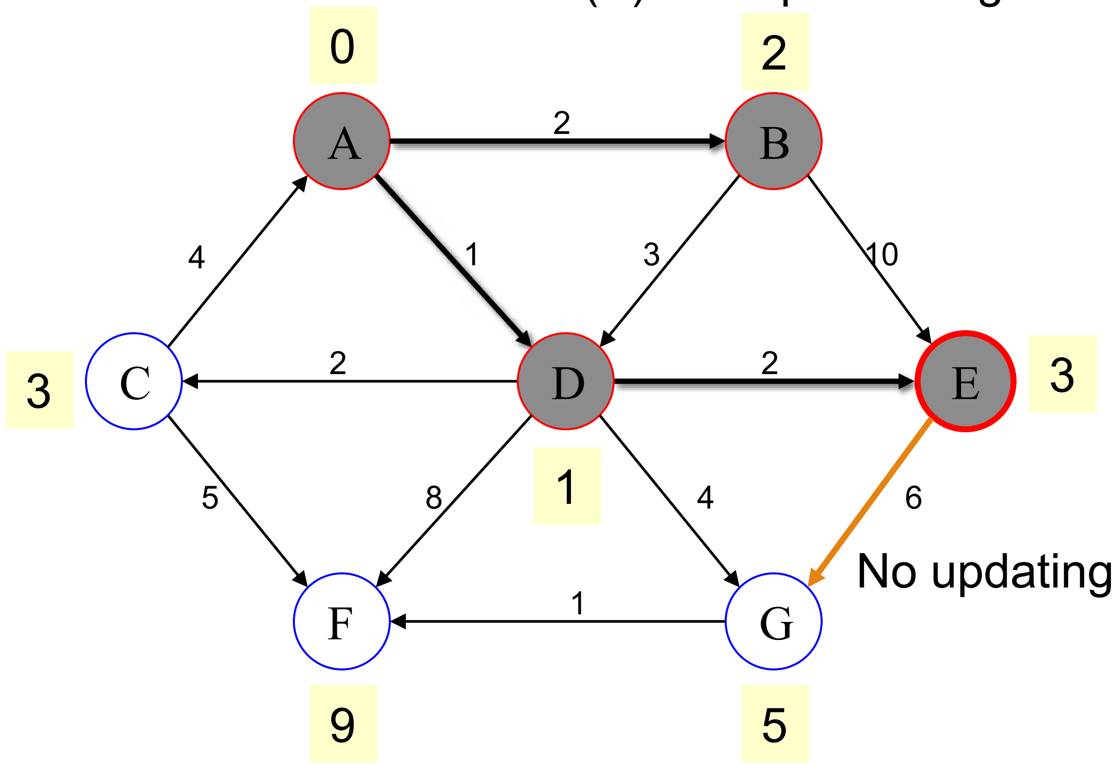


Node	Distance	Parent
A	0	
B	2	A
C	3	D
D	1	A
E	3	D
F	9	D
G	5	D

Note : distance(D) not updated since D is already known and distance(E) not updated since it is larger than previously computed

Example: Continued...

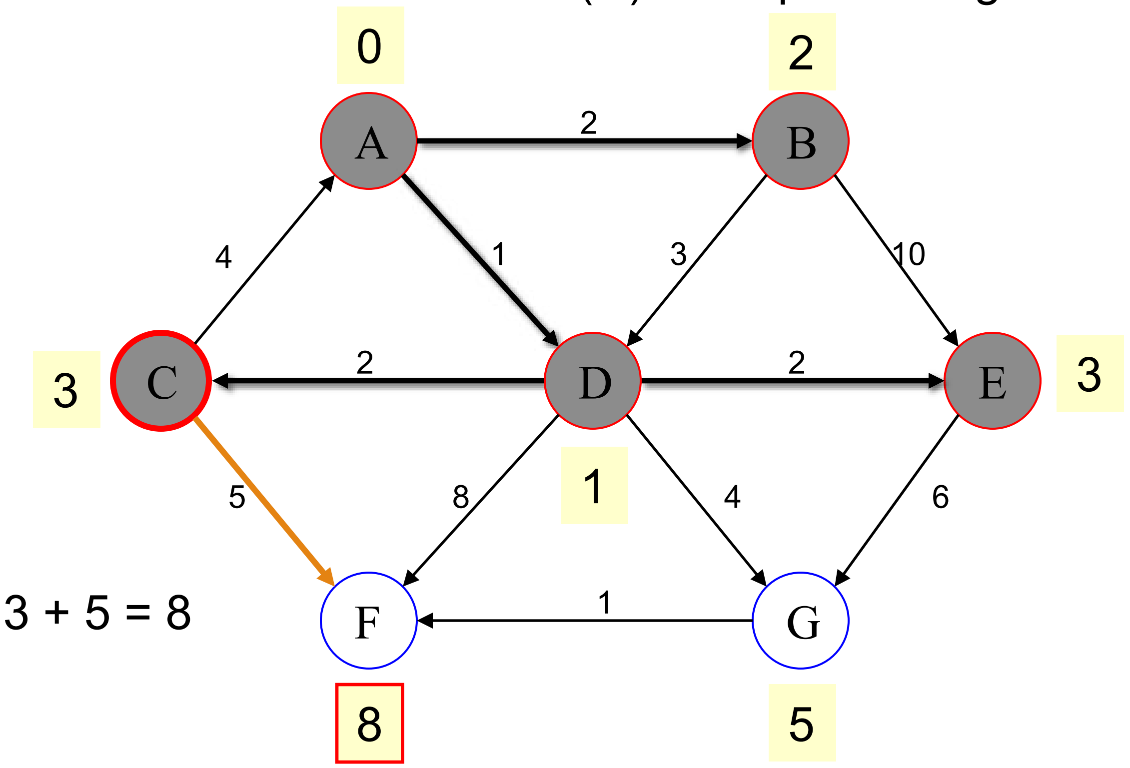
Pick vertex List with minimum distance (E) and update neighbors



Node	Distance	Parent
A	0	
B	2	A
C	3	D
D	1	A
E	3	D
F	9	D
G	5	D

Example: Continued...

Pick vertex List with minimum distance (C) and update neighbors

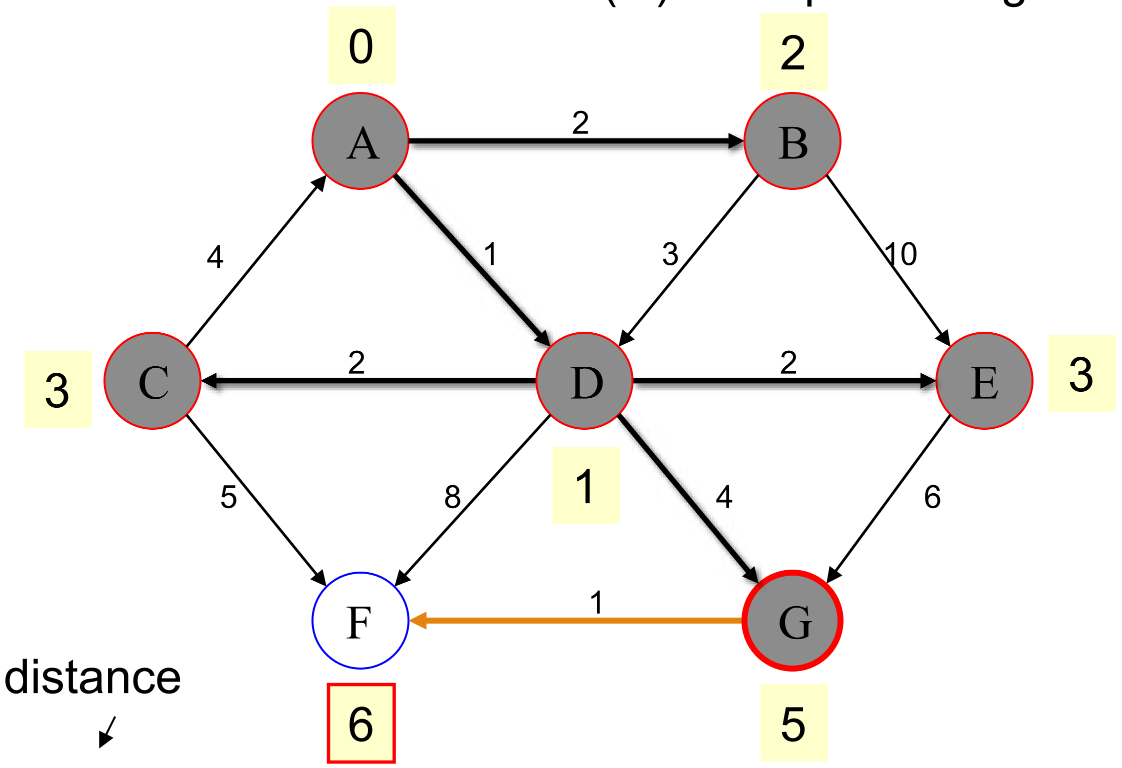


Distance(F) = 3 + 5 = 8

Node	Distance	Parent
A	0	
B	2	A
C	3	D
D	1	A
E	3	D
F	8	C
G	5	D

Example: Continued...

Pick vertex List with minimum distance (G) and update neighbors

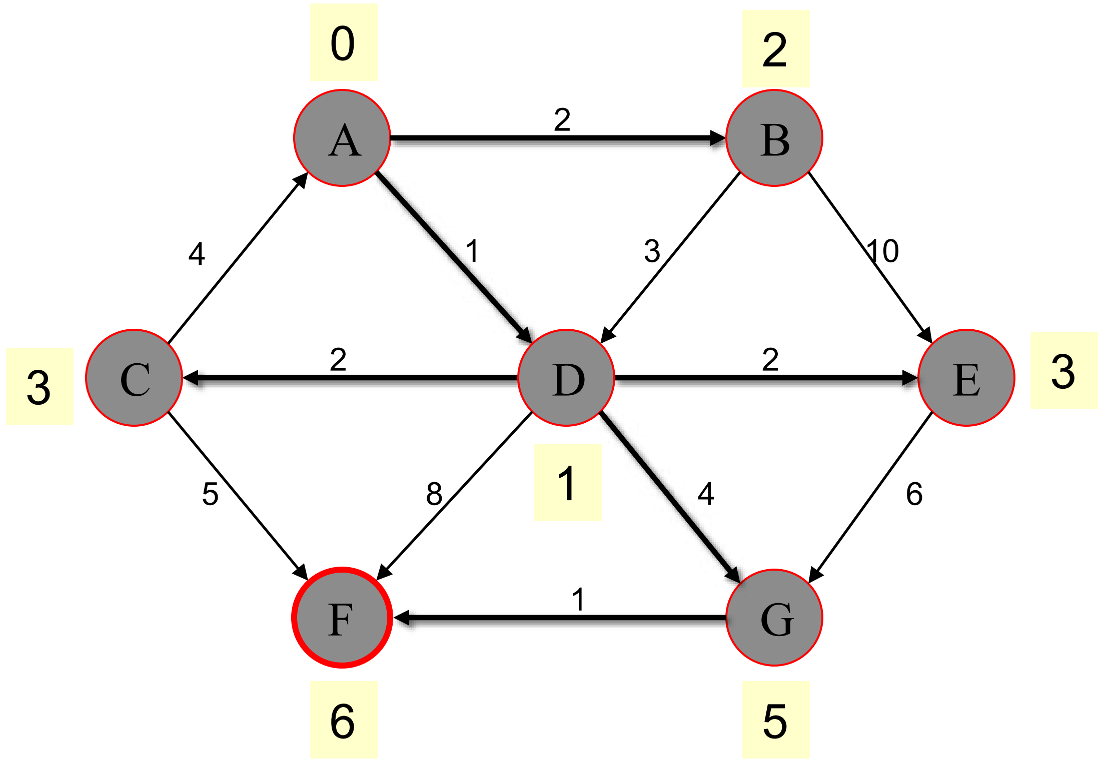


Node	Distance	Parent
A	0	
B	2	A
C	3	D
D	1	A
E	3	D
F	6	G
G	5	D

Previous distance

Distance(F) = min (8, 5+1) = 6

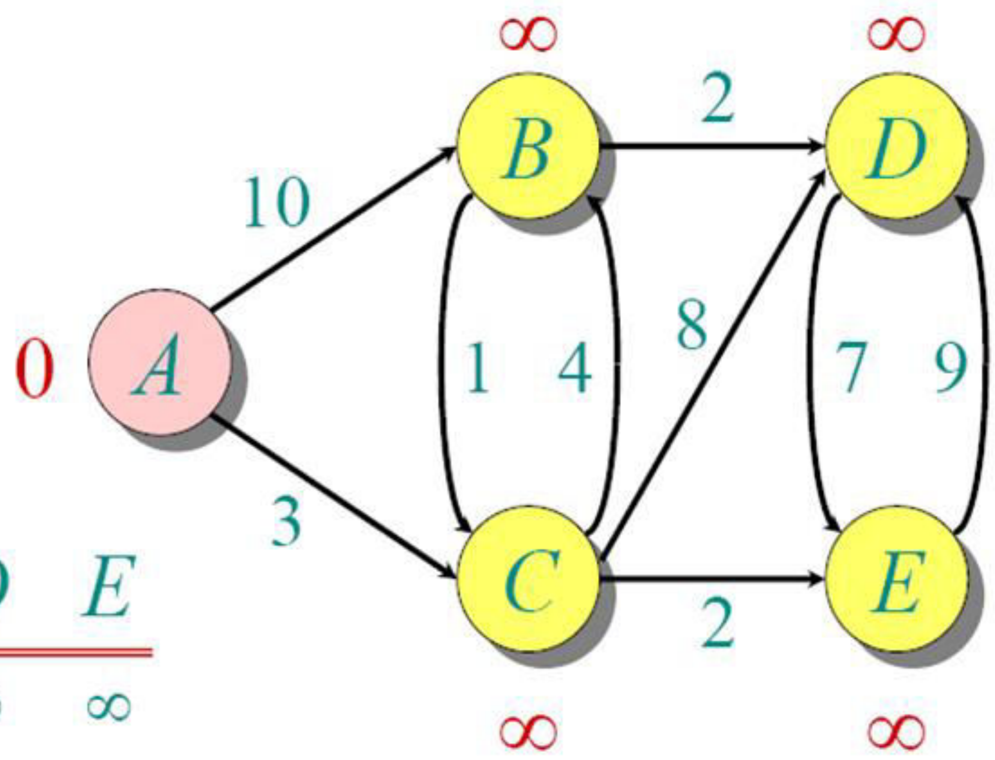
Example (end)



Node	Distance	Parent
A	0	
B	2	A
C	3	D
D	1	A
E	3	D
F	6	G
G	5	D

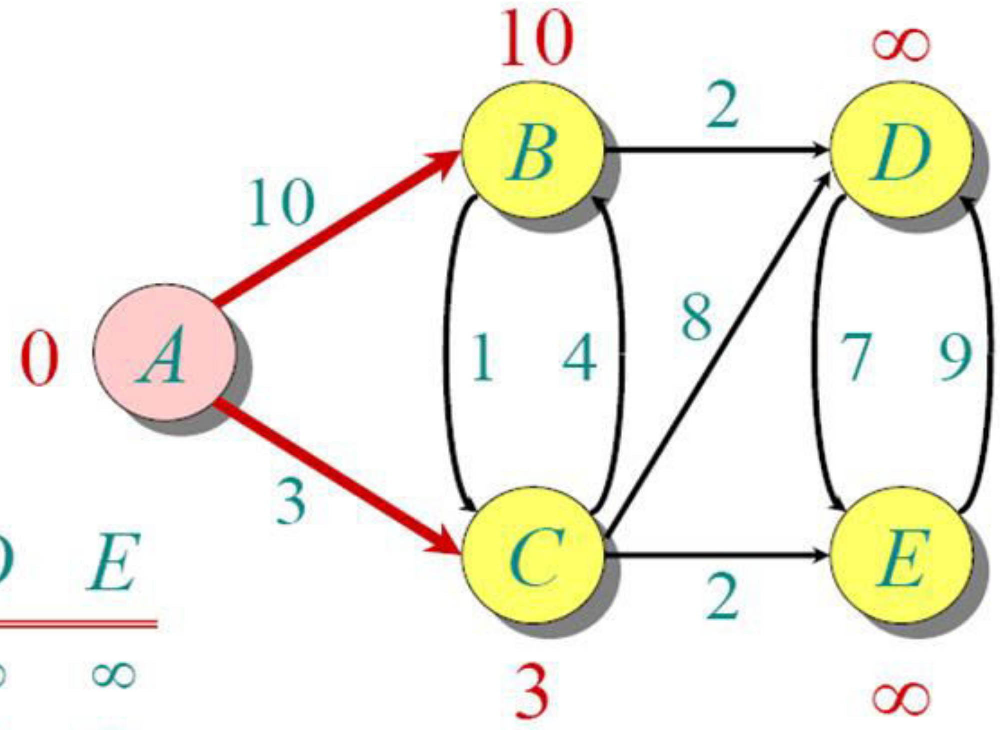
Pick vertex not in S with lowest cost (F) and update neighbors

Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

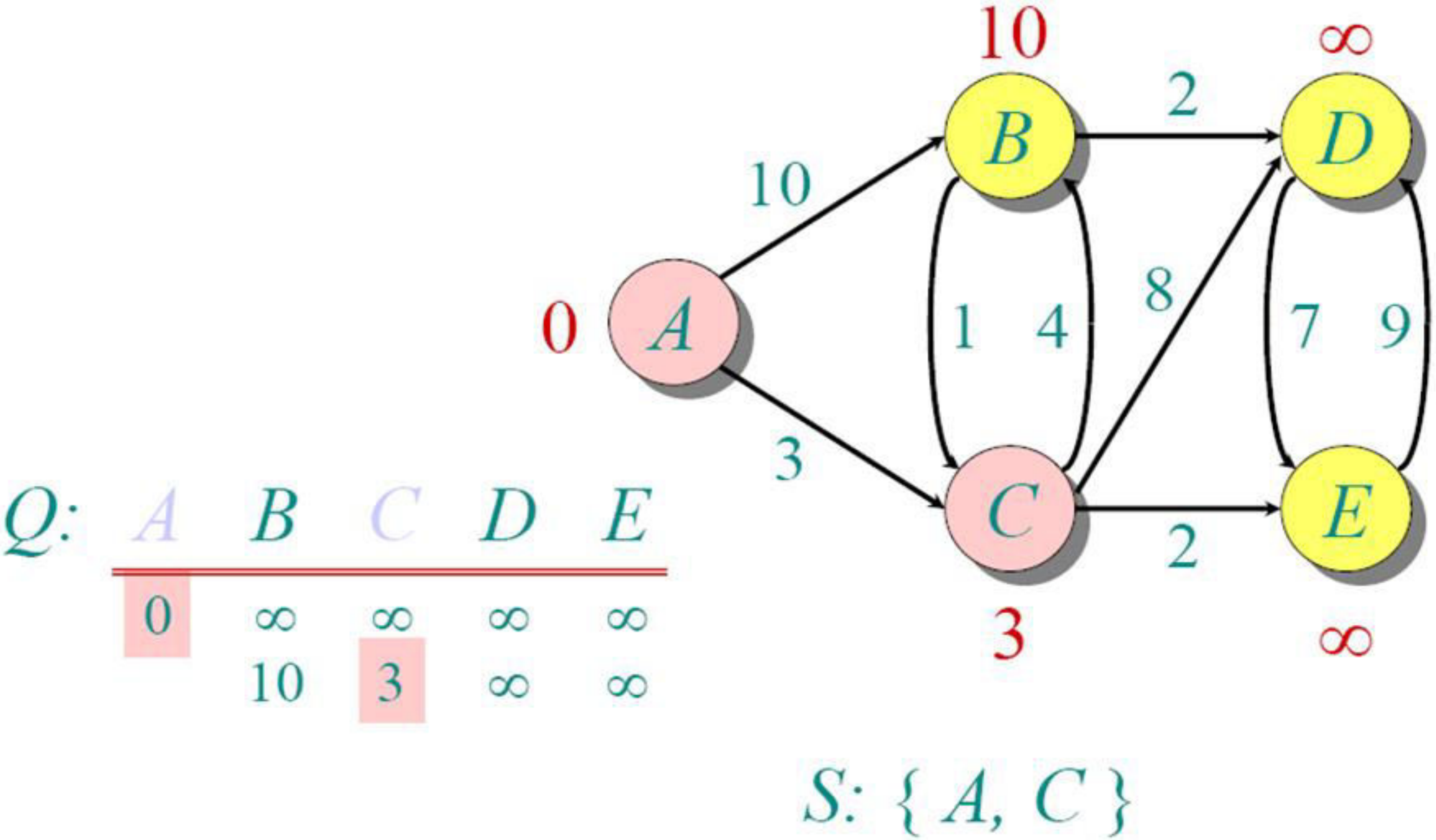
Example: Continued...



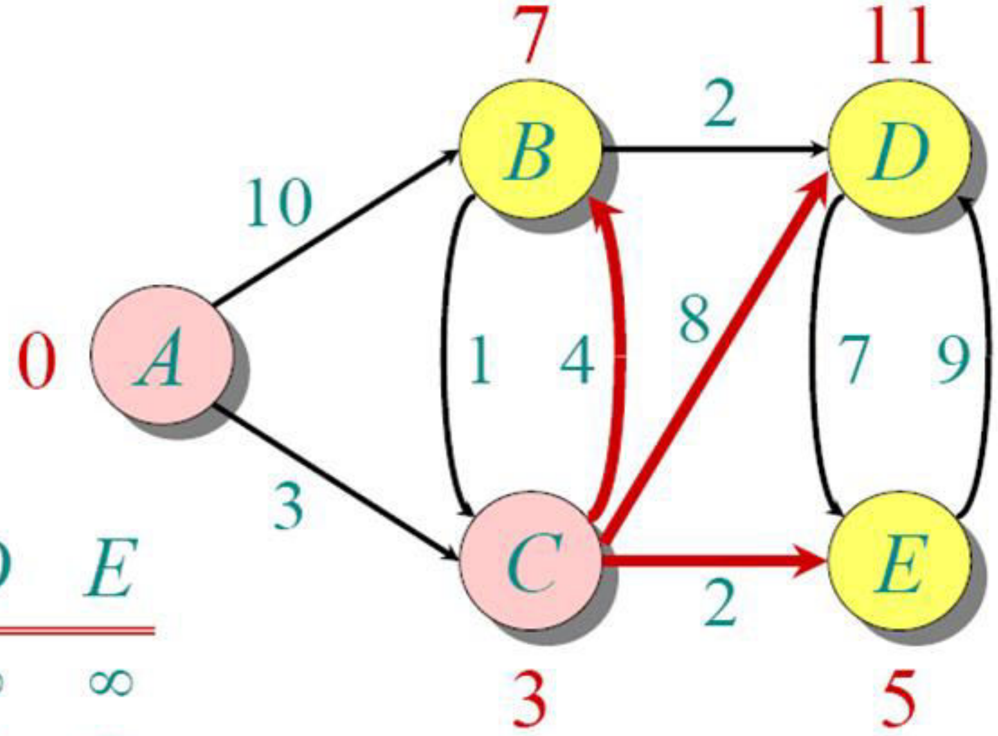
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Example: Continued...



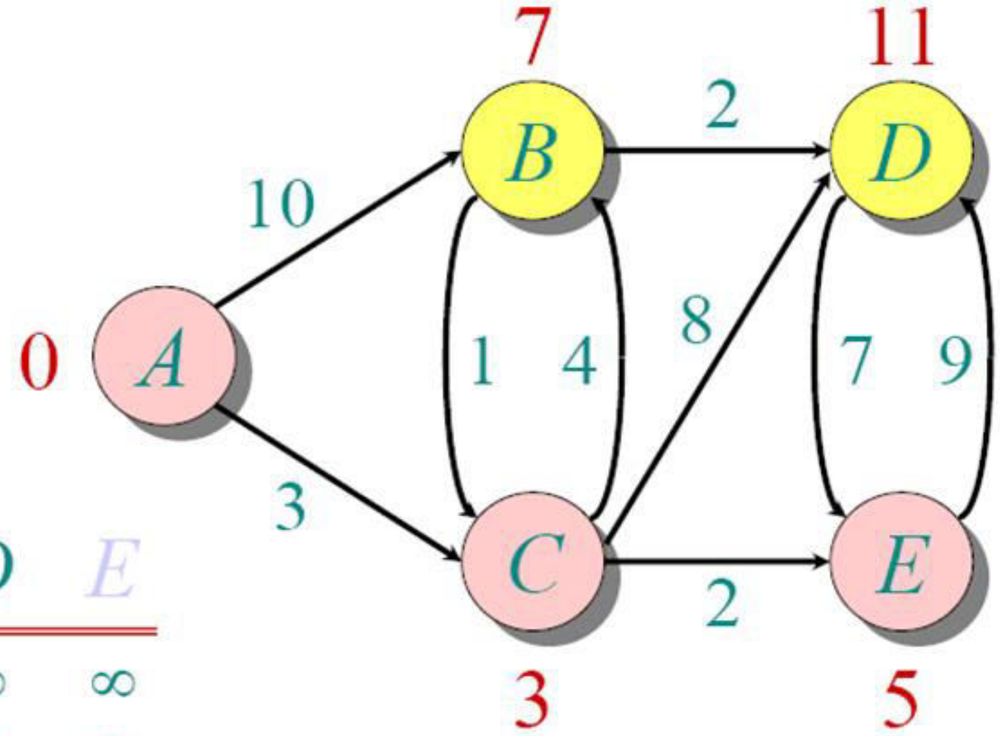
Example: Continued...



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

S: { A, C }

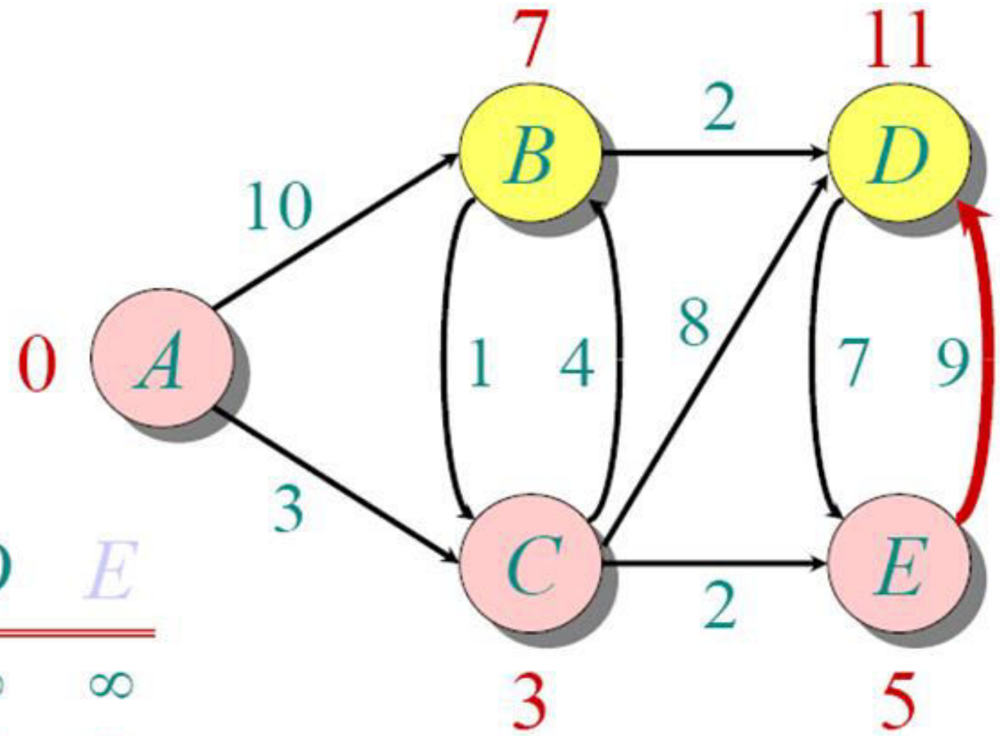
Example: Continued...



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

S: { A, C, E }

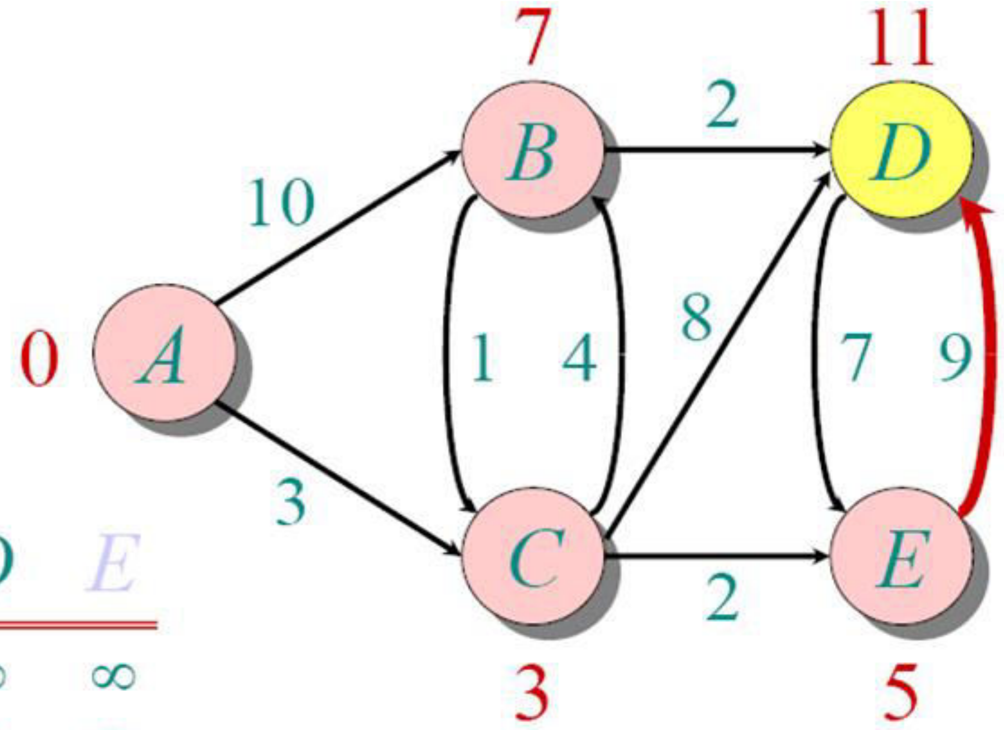
Example: Continued...



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

S: { A, C, E }

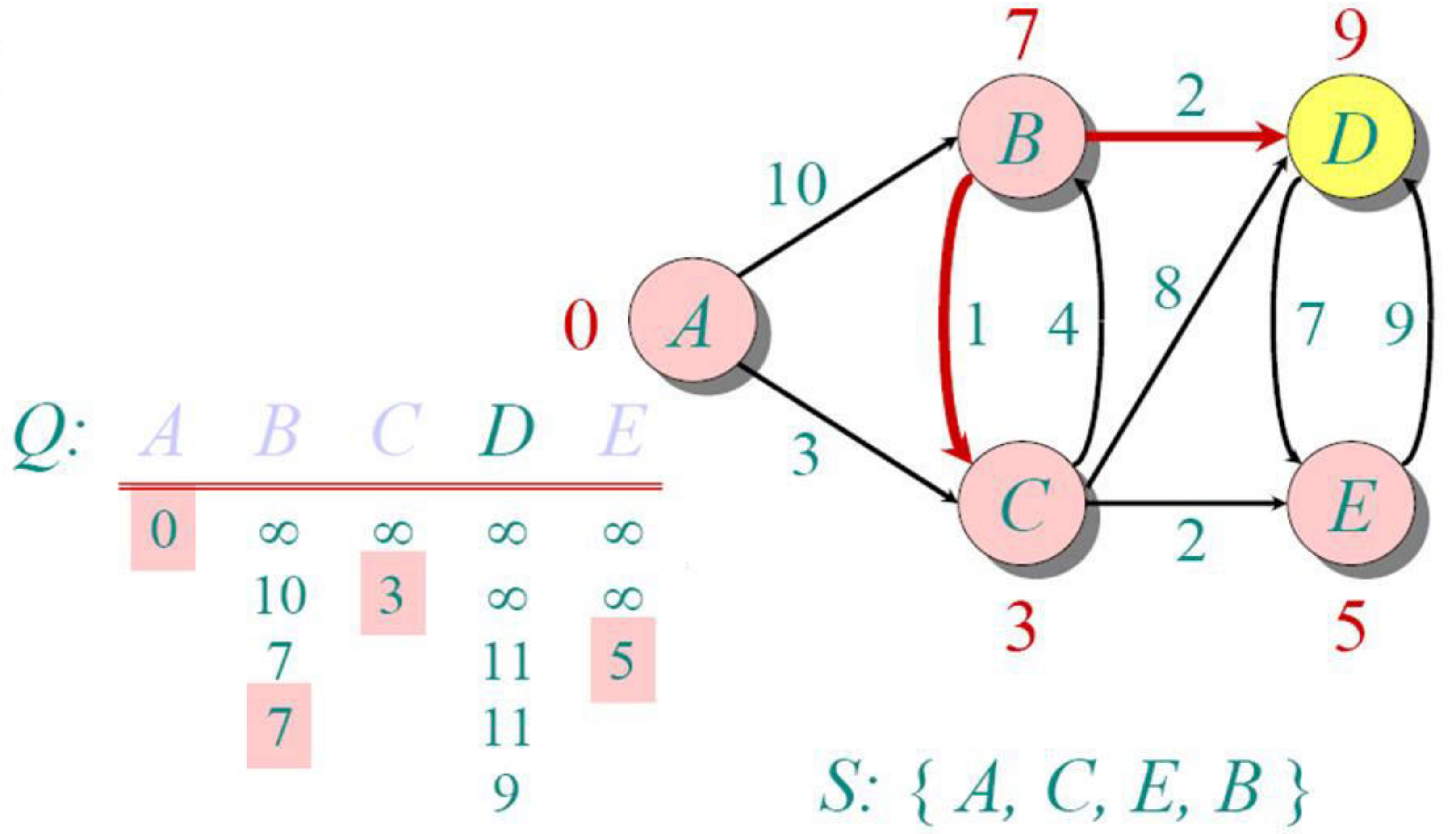
Example: Continued...



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

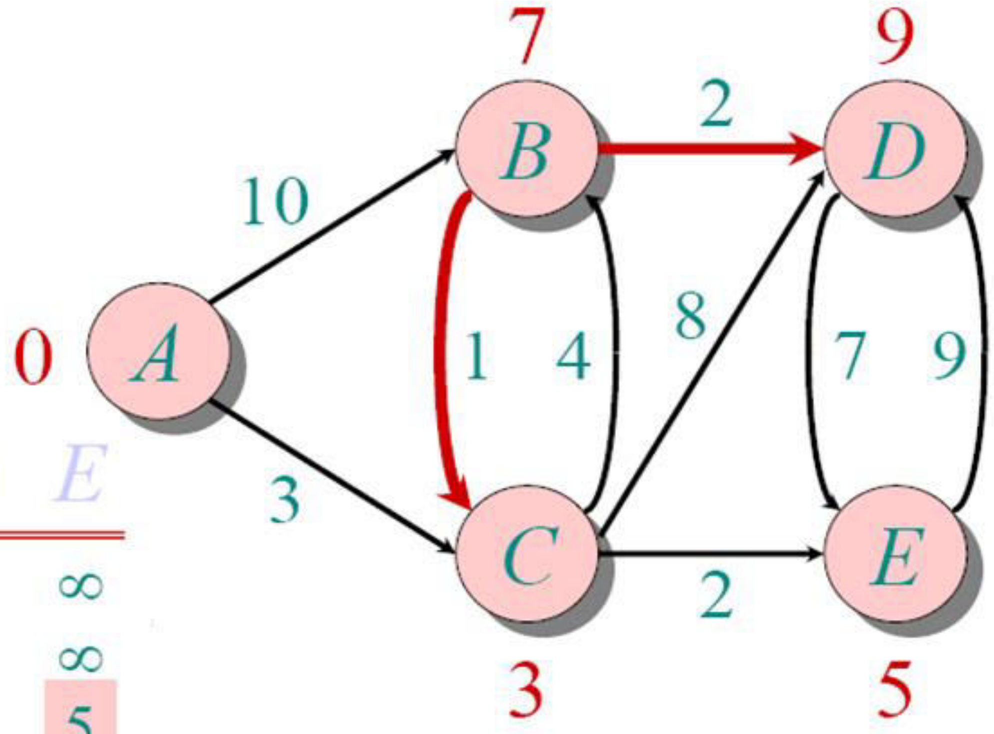
S: { A, C, E, B }

Example: Continued...



Example: Continued...

Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				$\boxed{9}$	

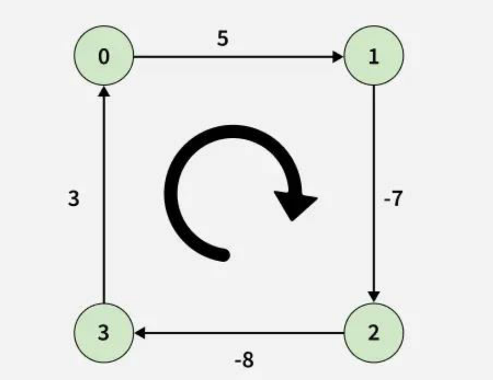


S: { A, C, E, B, D }

What if Negative Edge/Cycle??

Dijkstra is not suitable when the graph consists of negative edges. The reason is, it doesn’t revisit those nodes which have already been marked as visited. If a shorter path exists through a longer route with negative edges, Dijkstra’s algorithm will fail to handle it.

A negative weight cycle is a cycle in a graph, whose sum of edge weights is negative. If you traverse the cycle, the total weight accumulated would be less than zero.



Cycle weight 5 + (-7) + (-8) + 3 = -7

In the presence of negative weight cycle in the graph, the shortest path doesn’t exist because with each traversal of the cycle shortest path keeps decreasing.

Bellman–Ford Algorithm

Bellman-Ford** is a **single source** shortest path algorithm. It effectively works in the cases of negative edges and is able to detect negative cycles as well. It works on the principle of **relaxation of the edge

```
function BellmanFord(G, source):
   for each vertex v in G:
       distance[v] = ∞
   distance[source] = 0

   for i from 1 to |V| - 1:
      for each edge (u, v) with weight w in G:
          if distance[u] + w < distance[v]:
              distance[v] = distance[u] + w

   for each edge (u, v) with weight w in G:
      if distance[u] + w < distance[v]:
          report "Negative-weight cycle detected"
          return
   return distance
```

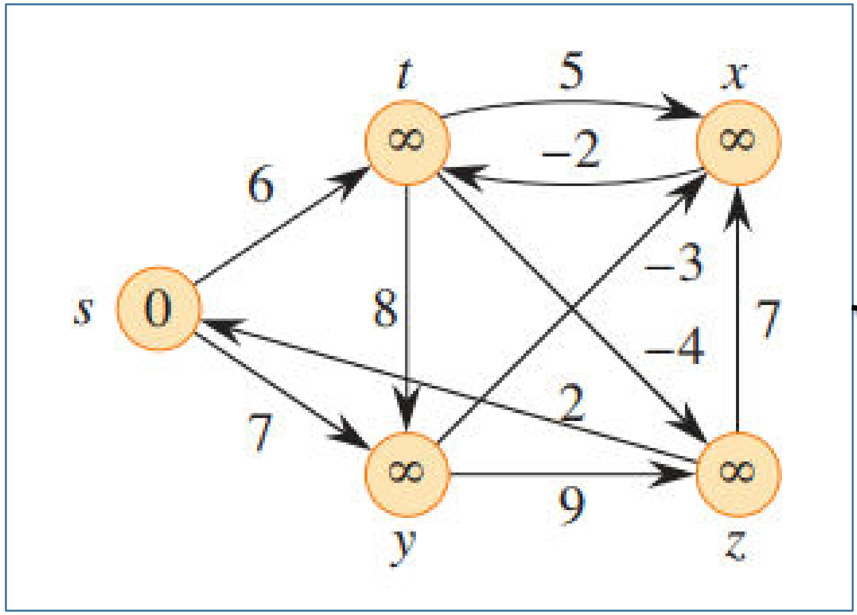
Set the distance to all other nodes as ∞ (infinity).

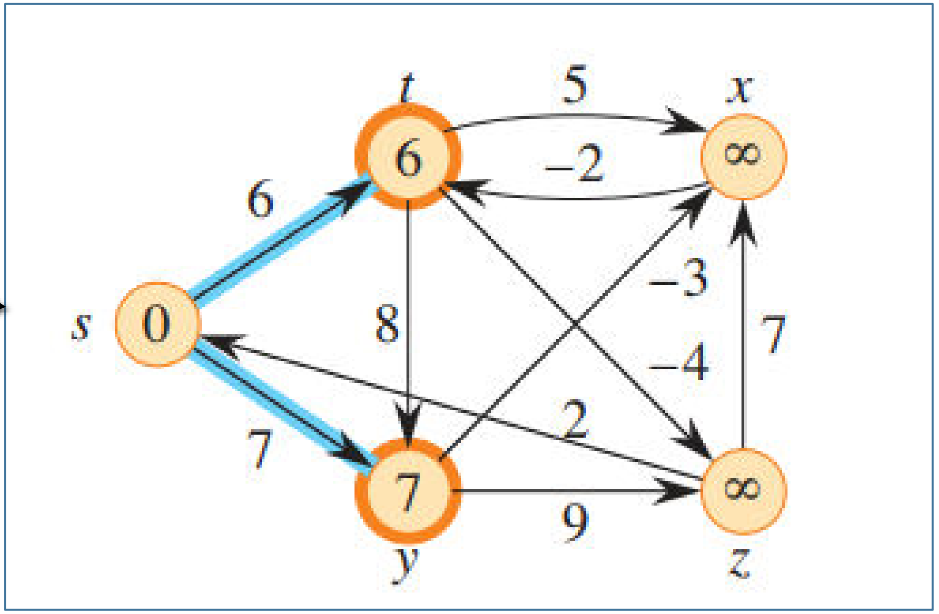
Set the distance to the source node as 0.

Relax all edges (V - 1) times

Check for negative-weight cycles (optional but important)

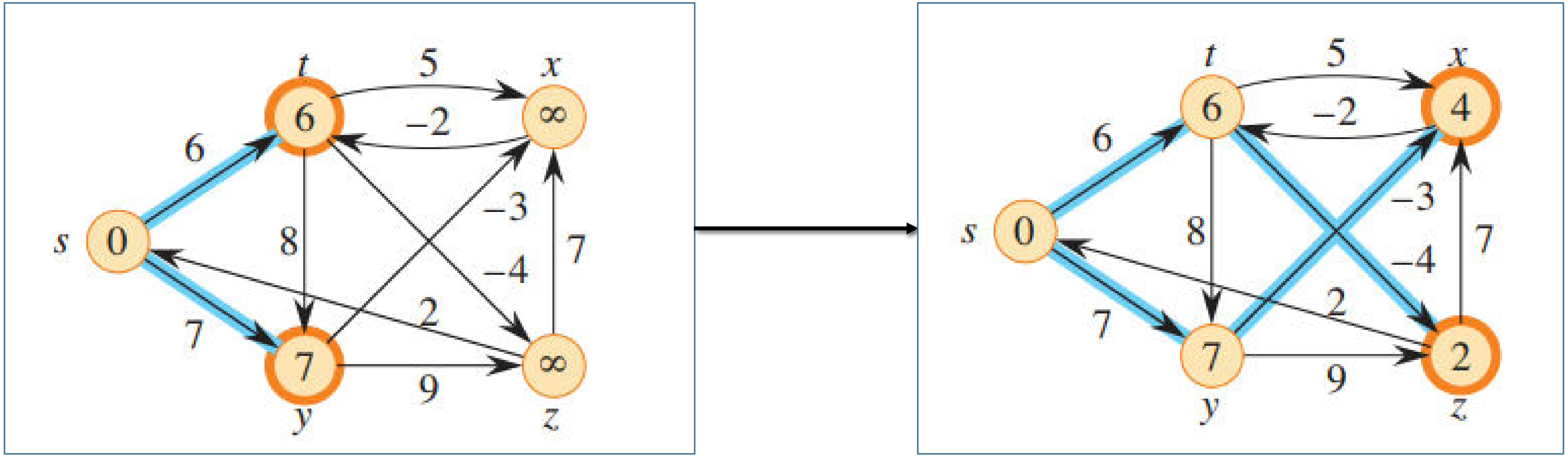
Bellman–Ford Algorithm - Example





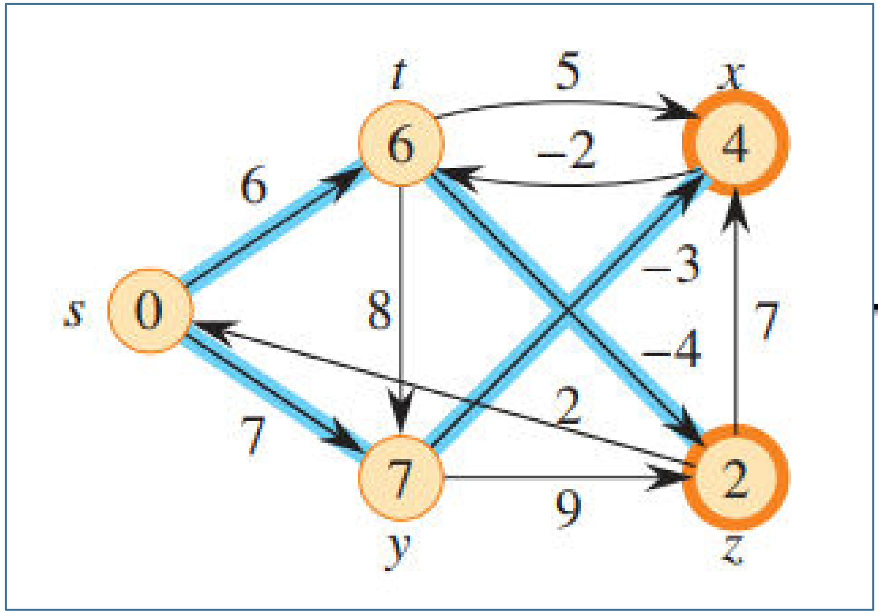
First Relaxation

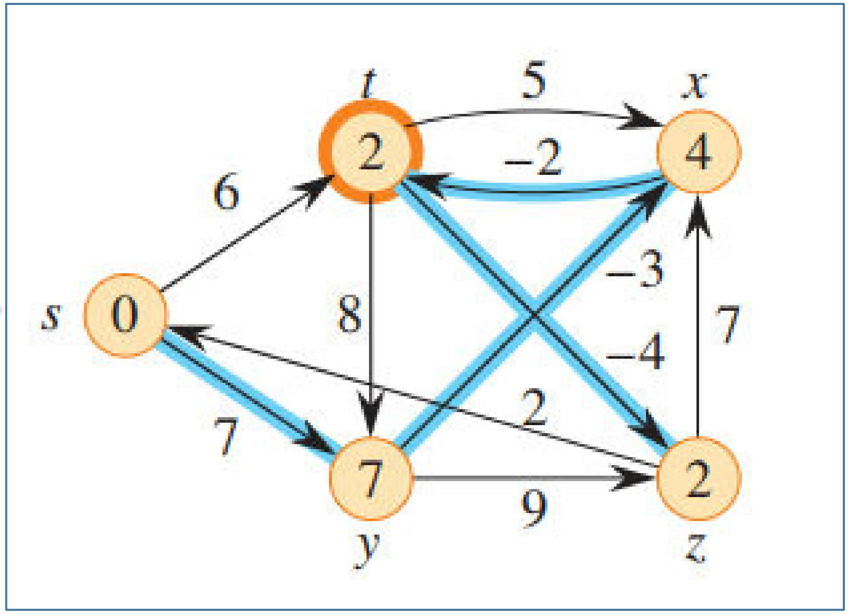
Example Continued…



Second Relaxation

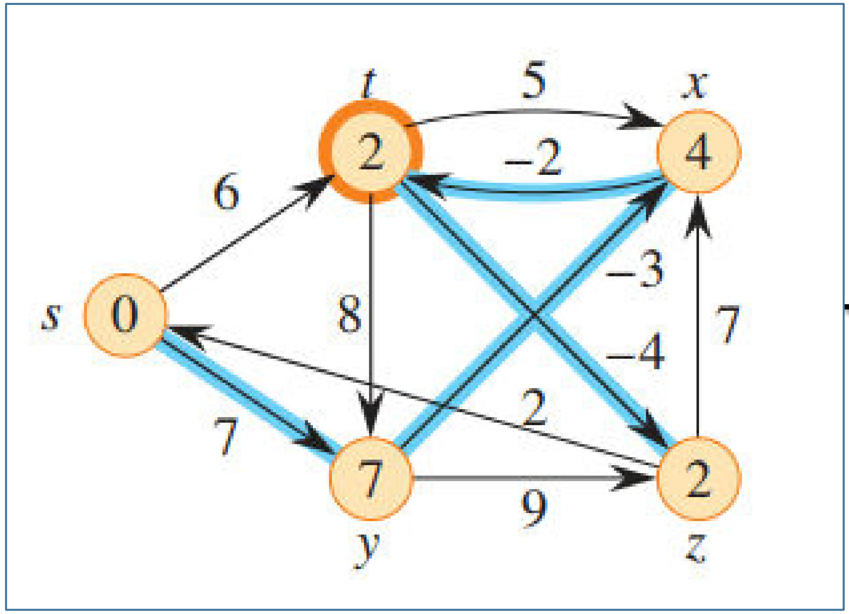
Example Continued…

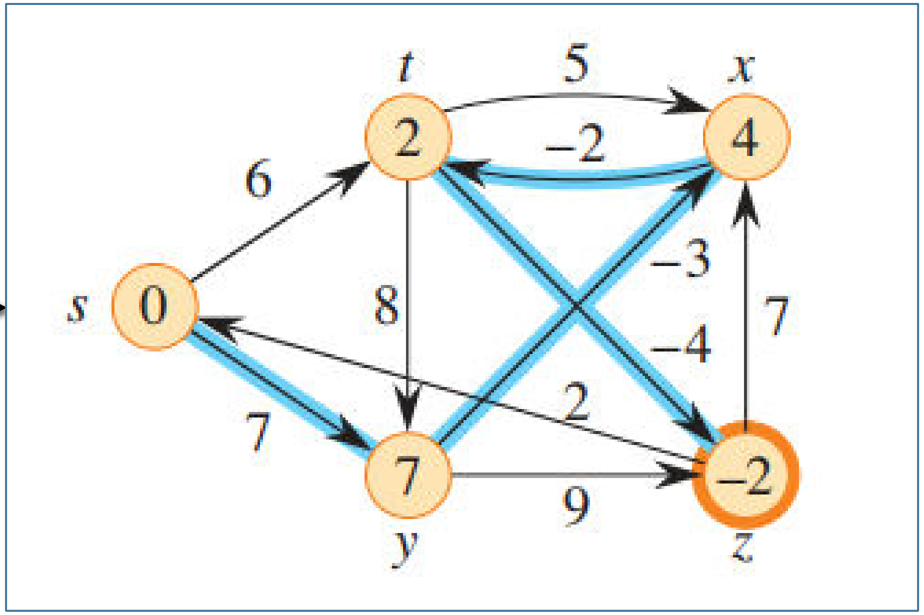




Third Relaxation

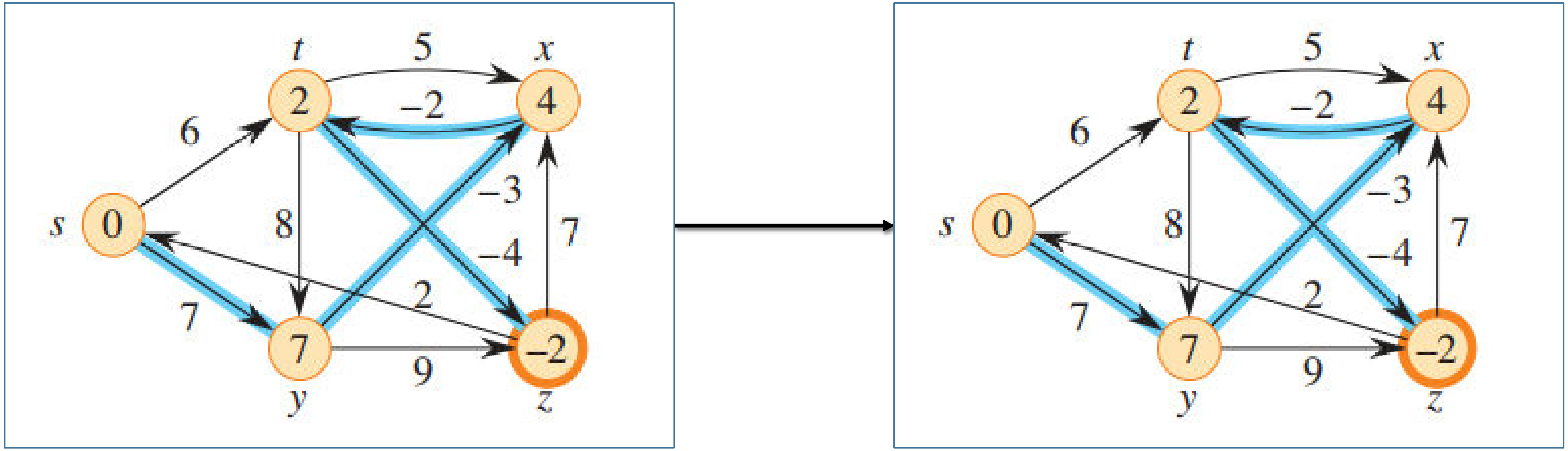
Example Continued…





Fourth Relaxation

Example Continued…

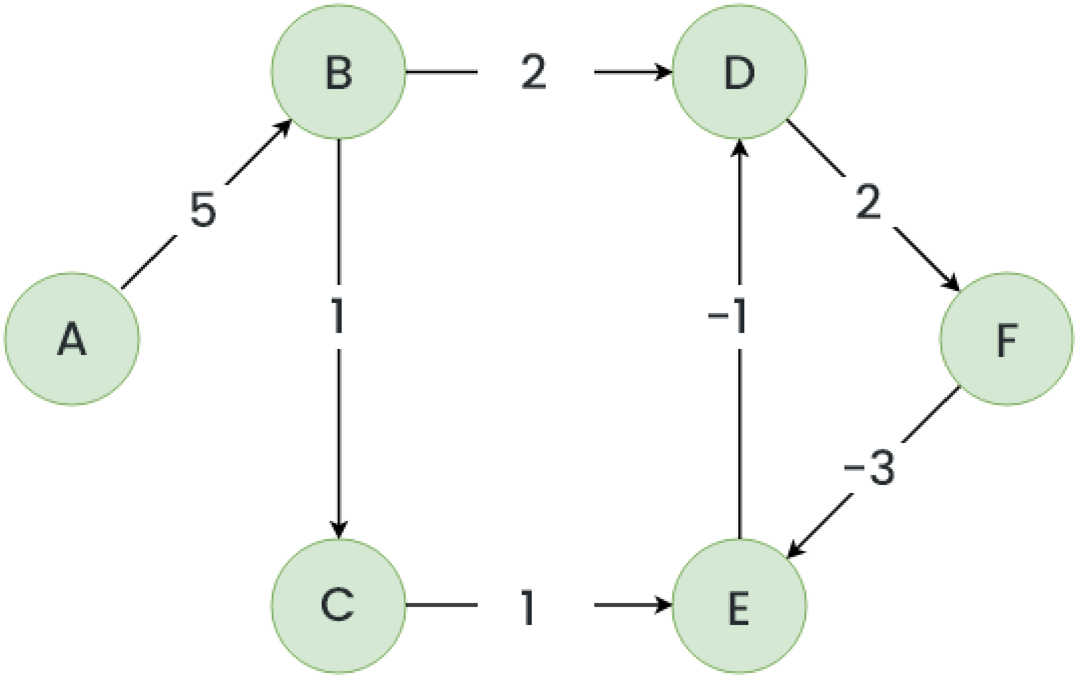


Fifth Relaxation

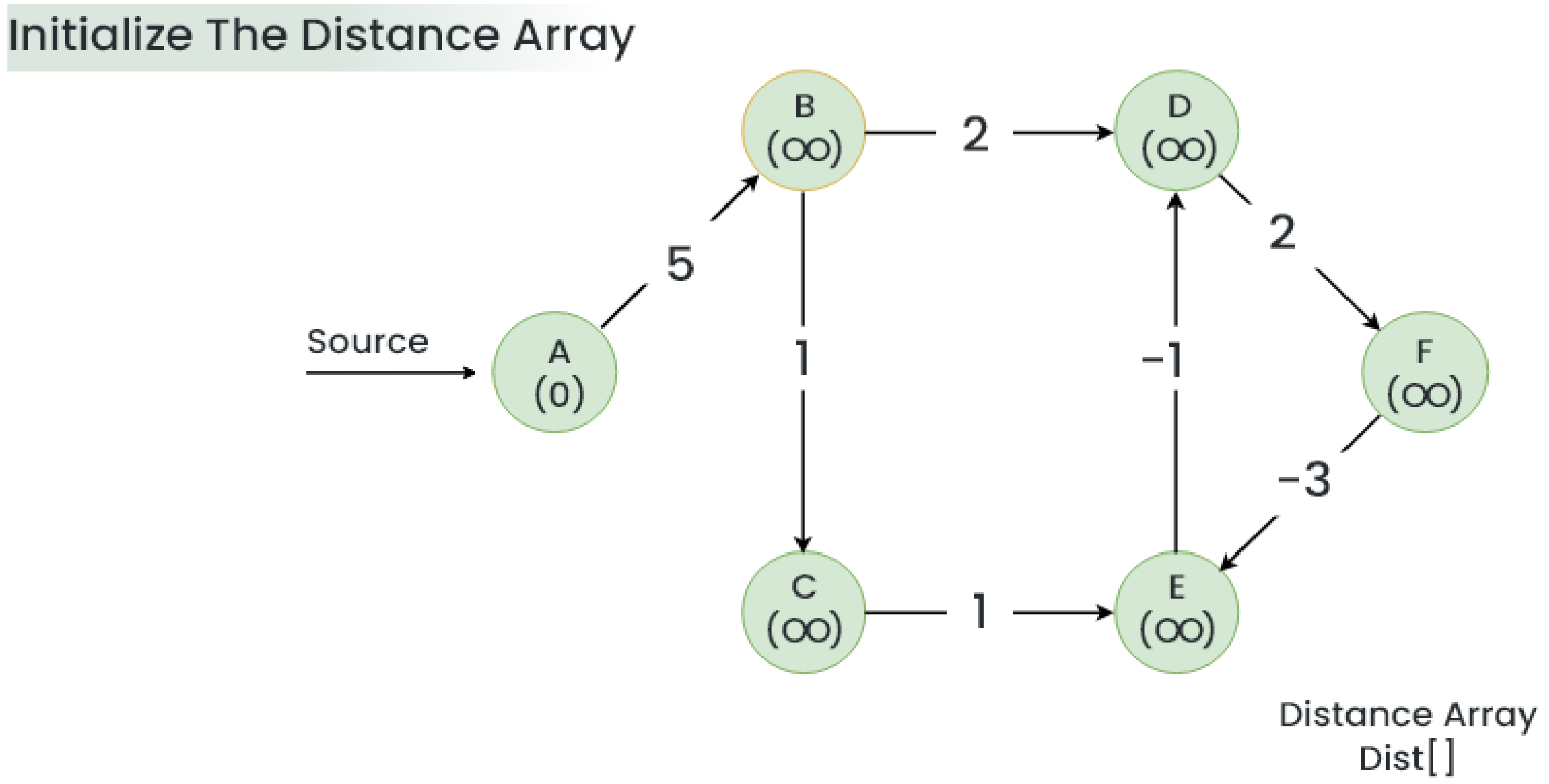
No updates found. Hence, no negative edge cycle present in the graph.

Another Example

Let's suppose we have a graph which is given below and we want to find whether there exists a negative cycle or not using Bellman-Ford.



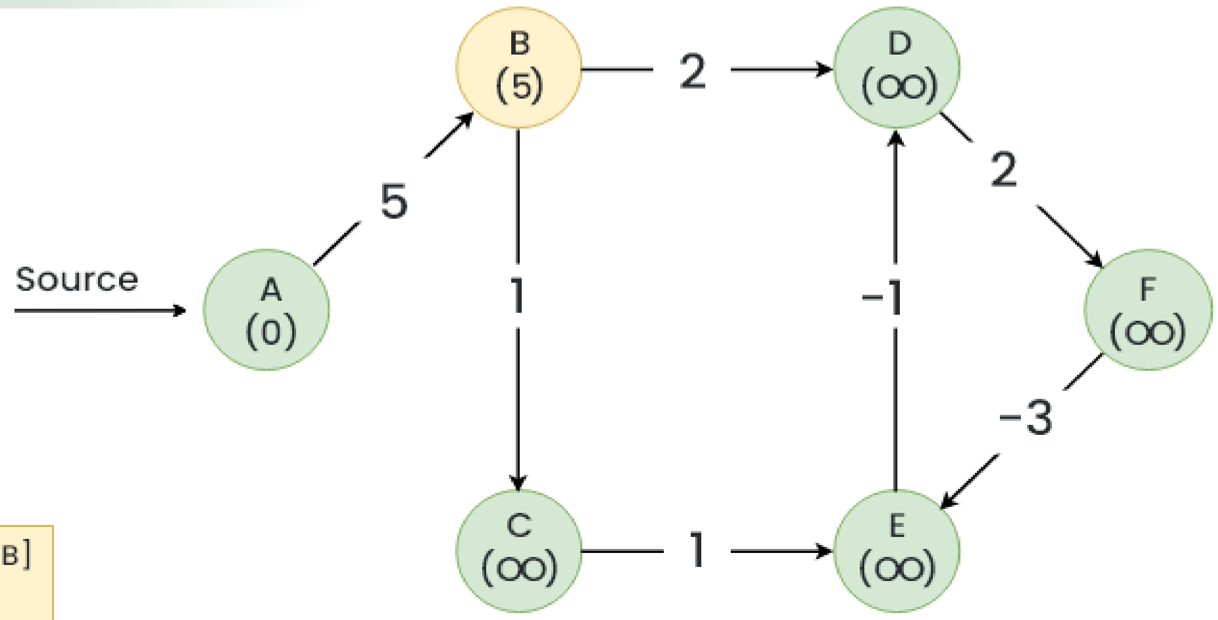
Example Continued…



A	B	C	D	E	F
0	∞	∞	∞	∞	∞

Example Continued…

1st Relaxation Of Edges



Dist [A] + 5 <Dist[B]
0+5<(∞)
Dist[B] = 5

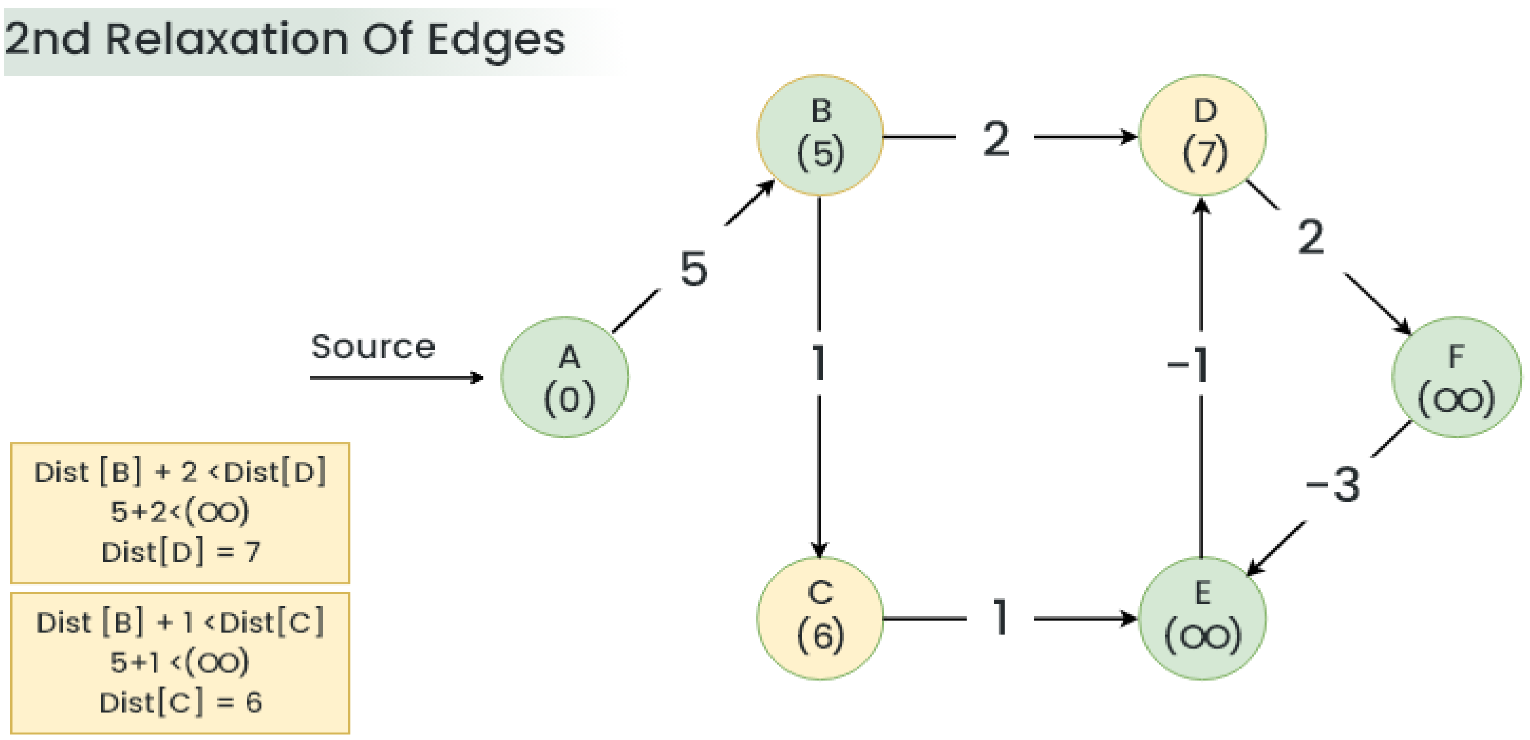
Distance Array

A	B	C	D	E	F
0	∞	∞	∞	∞	∞

↓

A	B	C	D	E	F
0	5	∞	∞	∞	∞

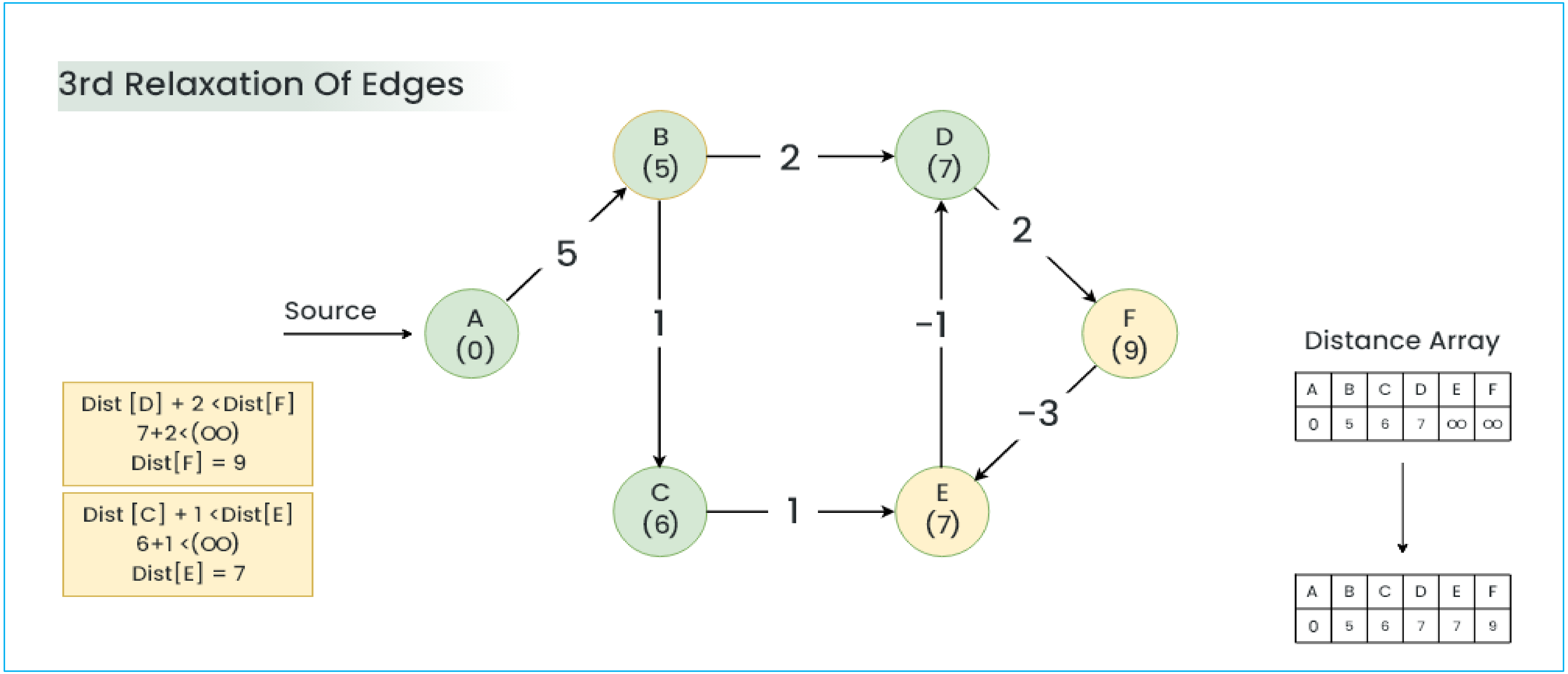
Example Continued…



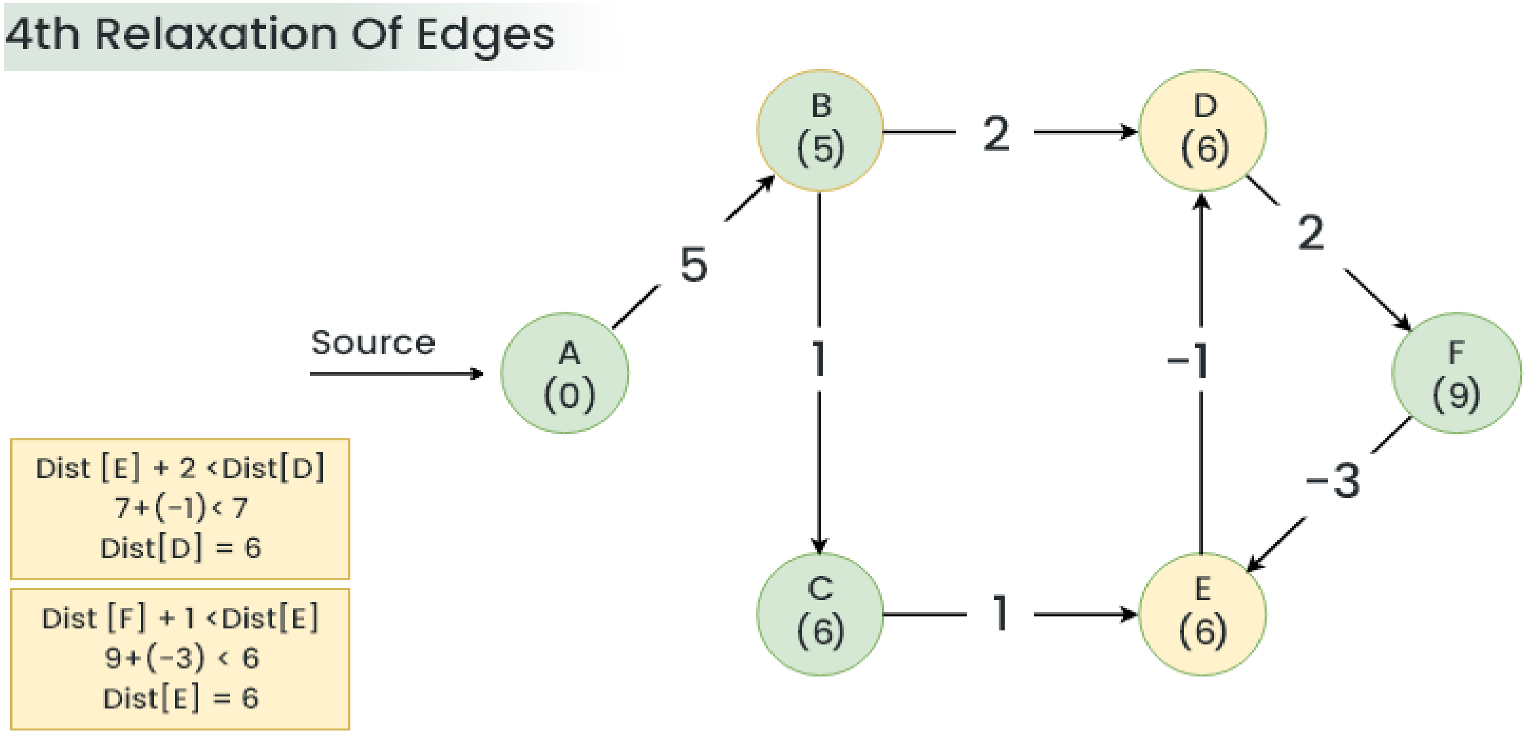
A	B	C	D	E	F
0	5	∞	∞	∞	∞

A	B	C	D	E	F
0	5	6	7	∞	∞

Example Continued…



Example Continued…

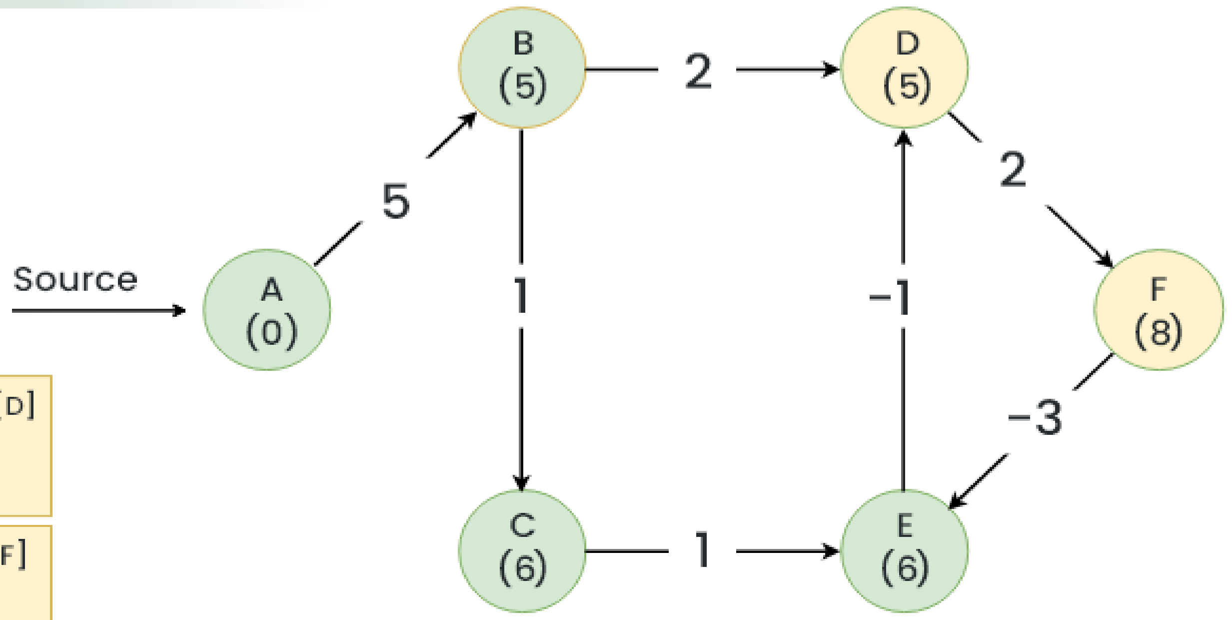


A	B	C	D	E	F
0	5	6	7	7	9

A	B	C	D	E	F
0	5	6	6	6	9

Example Continued…

5th Relaxation Of Edges



Dist [E] + (-1) < Dist[D]
6+(-1) < 6
Dist[D] = 5

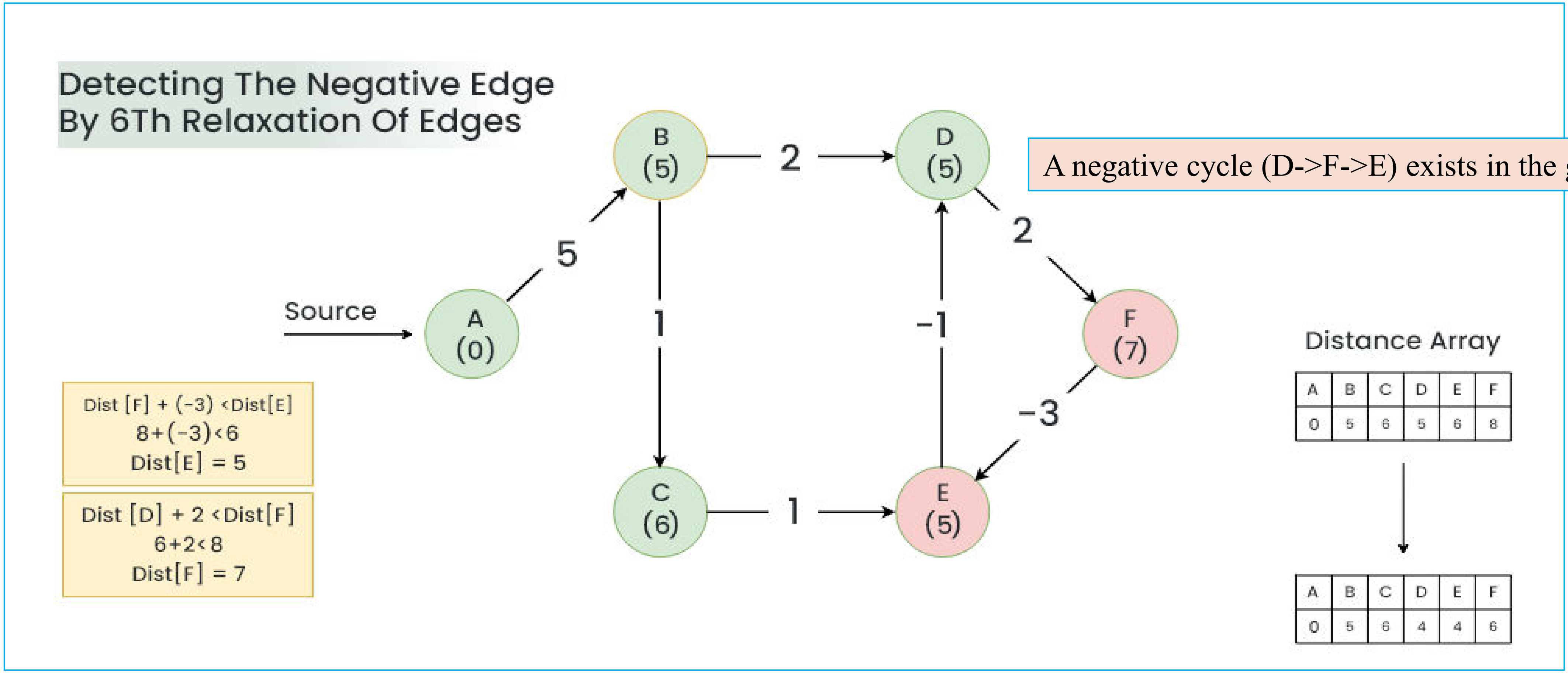
Dist [D] + 2 < Dist[F]
6+2 < 9
Dist[F] = 8

Distance Array

A	B	C	D	E	F
0	5	6	6	6	9

A	B	C	D	E	F
0	5	6	5	6	8

Example Continued…



A negative cycle (D->F->E) exists in the graph.

THANK YOU