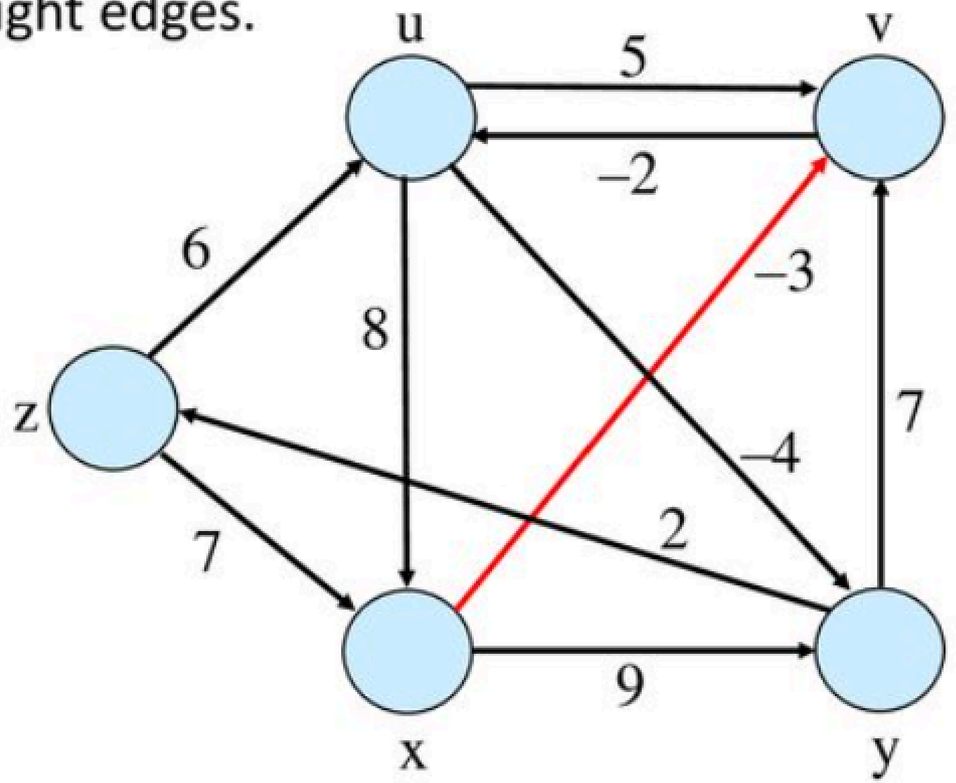
Single-Source Shortest Paths

- **Given:** A single source vertex in a weighted, directed graph.
- Want to compute a shortest path for each possible destination.
  - Similar to BFS.
- We will assume either
  - no negative-weight edges, or
  - no reachable negative-weight cycle that is reachable from source.
- Algorithm will compute a shortest-path tree.
  - Similar to BFS tree.
  - If there is a negative weight cycle, the algorithm indicates that no solution exists.
  - If there is no such cycle, the algorithm produces the shortest paths and their weights.

Negative Edge

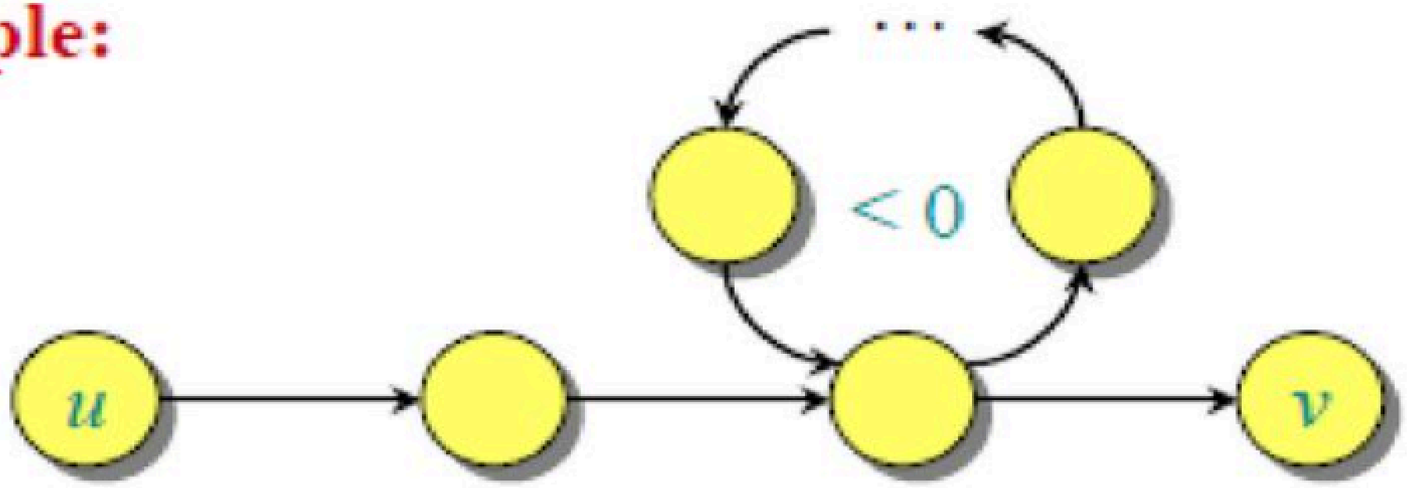
- A negative edge is simply an edge having a negative weight.
  - For example, the edge X-V in the graph is a negative edge. For a negative weight you could simply perform the calculation as you would have done for positive weight edges.



Negative-weight cycles

Recall: If a graph $G = (V, E)$ contains a negative-weight cycle, then some shortest paths may not exist.

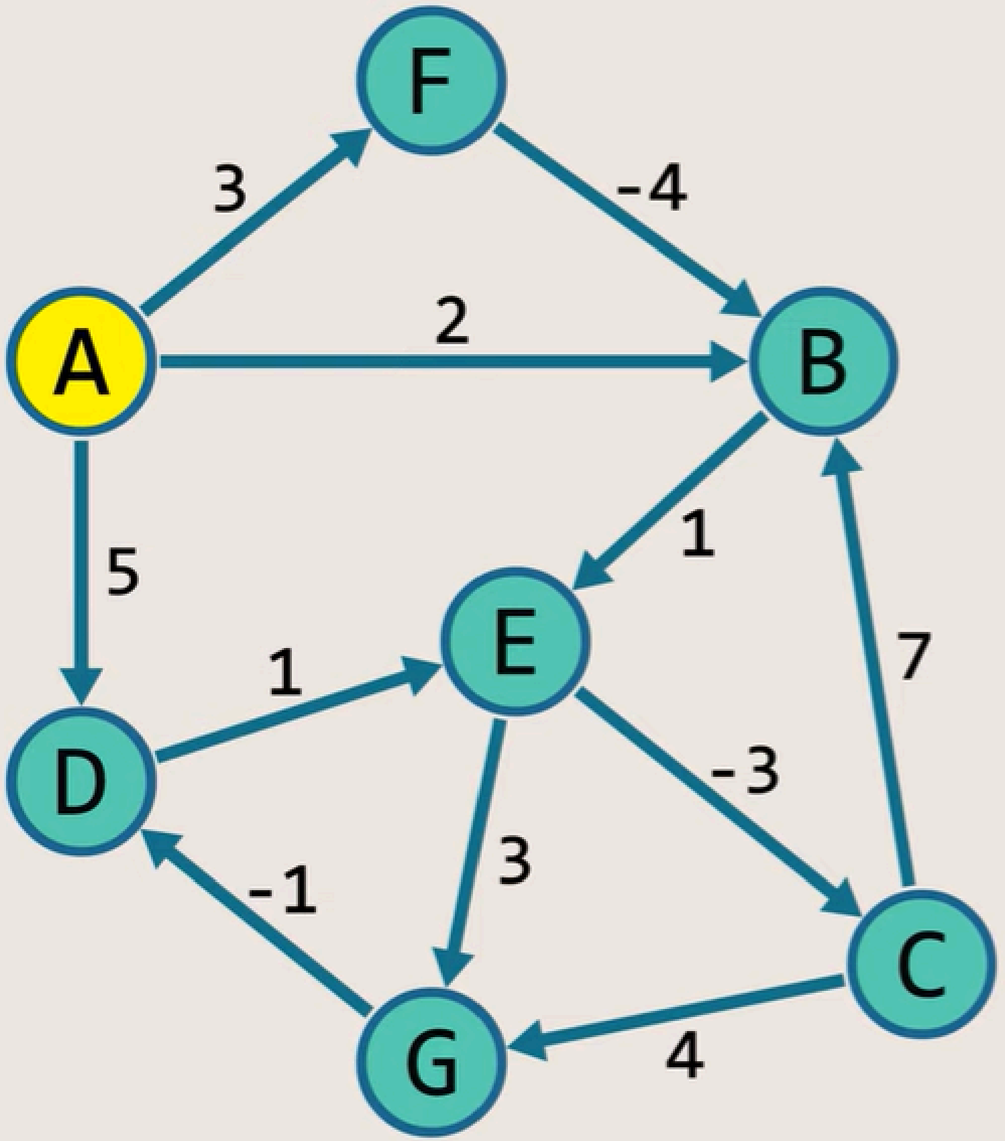
Example:



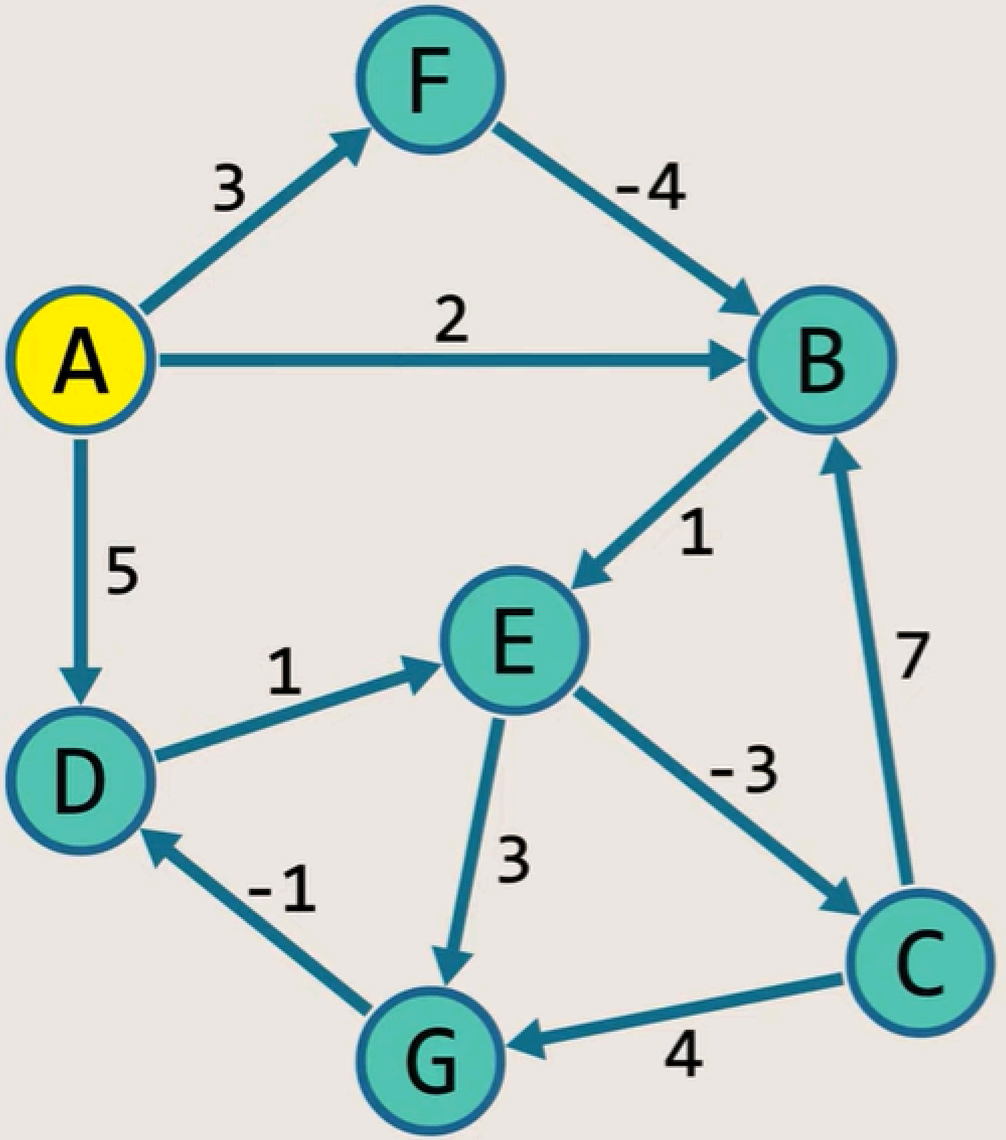
Bellman-Ford algorithm: Finds all shortest-path lengths from a ***source*** $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

BELLMAN–FORD ALGORITHM

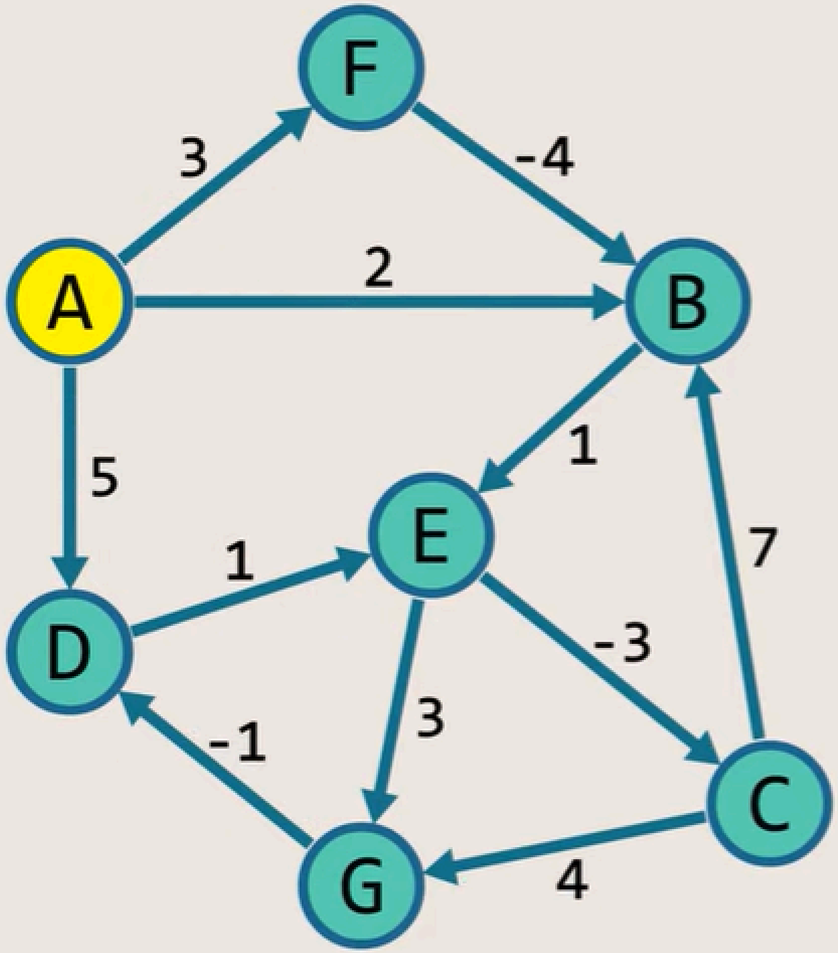
- Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence.



Node	Cost	Previous
A	0	
B		
C		
D		
E		
F		
G		



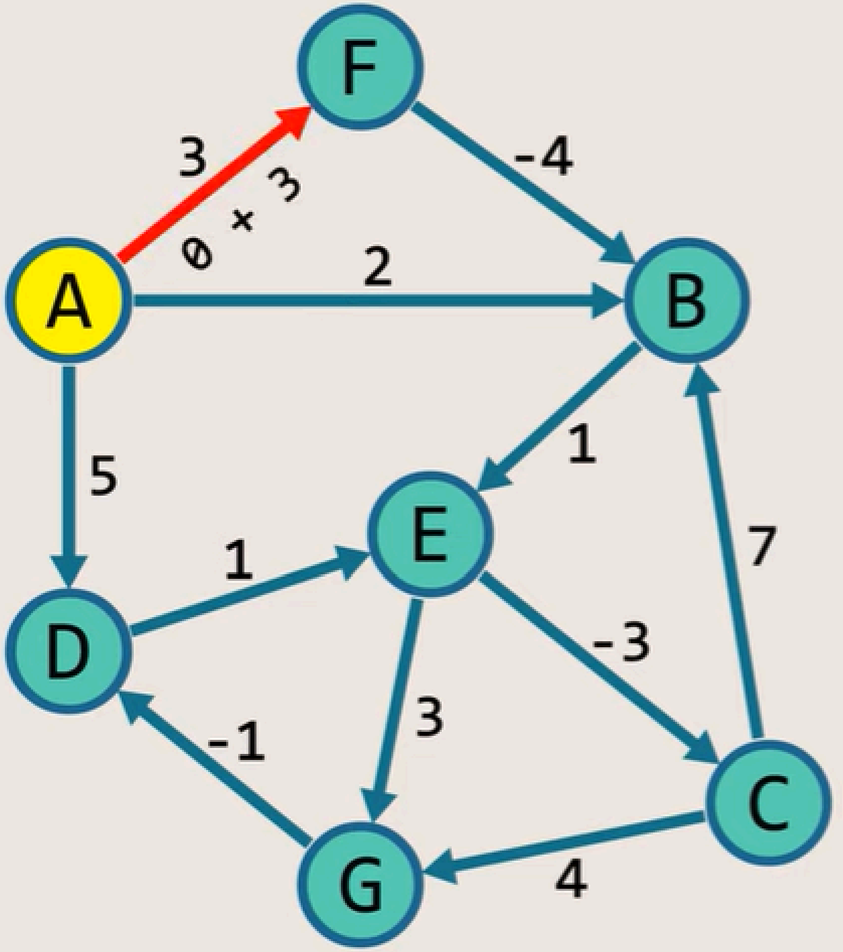
Node	Cost	Previous
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	



Node	Cost	Previous
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	

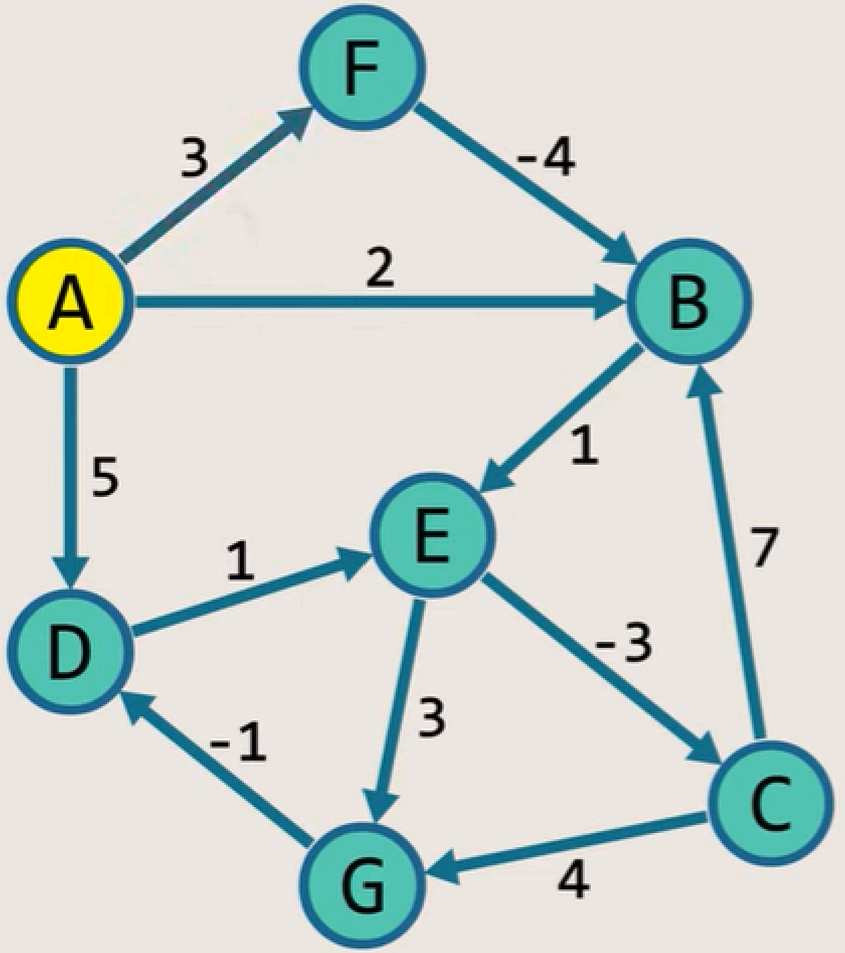
List of edges

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



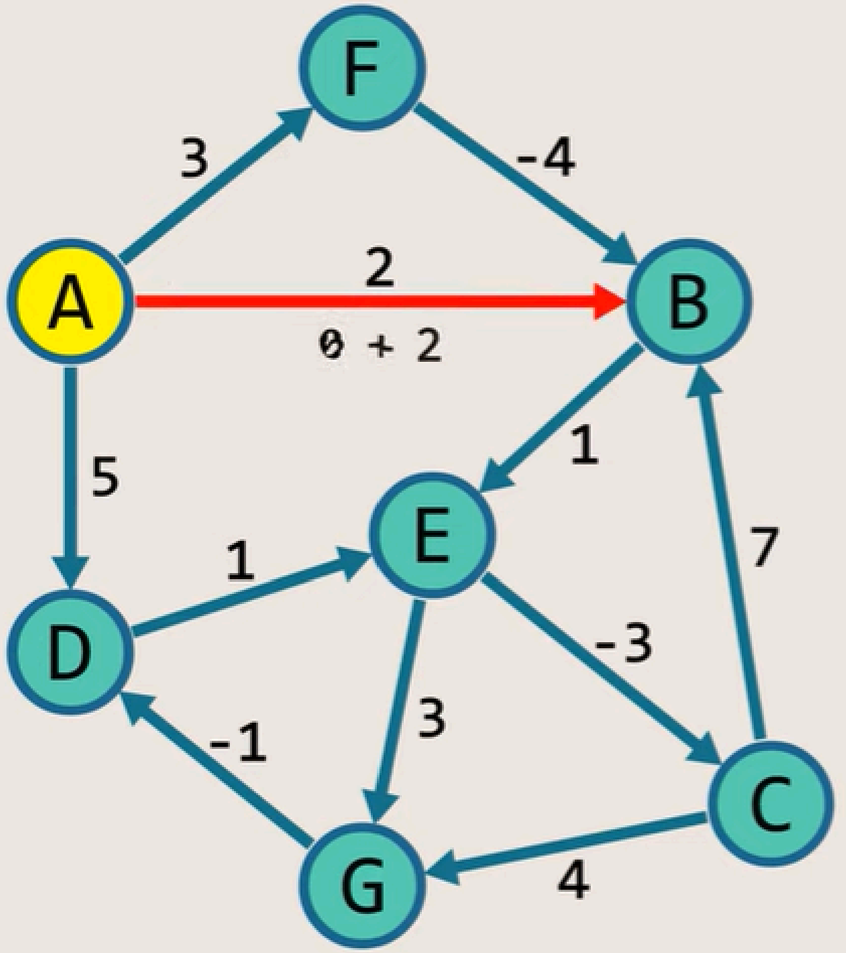
Node	Cost	Previous
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



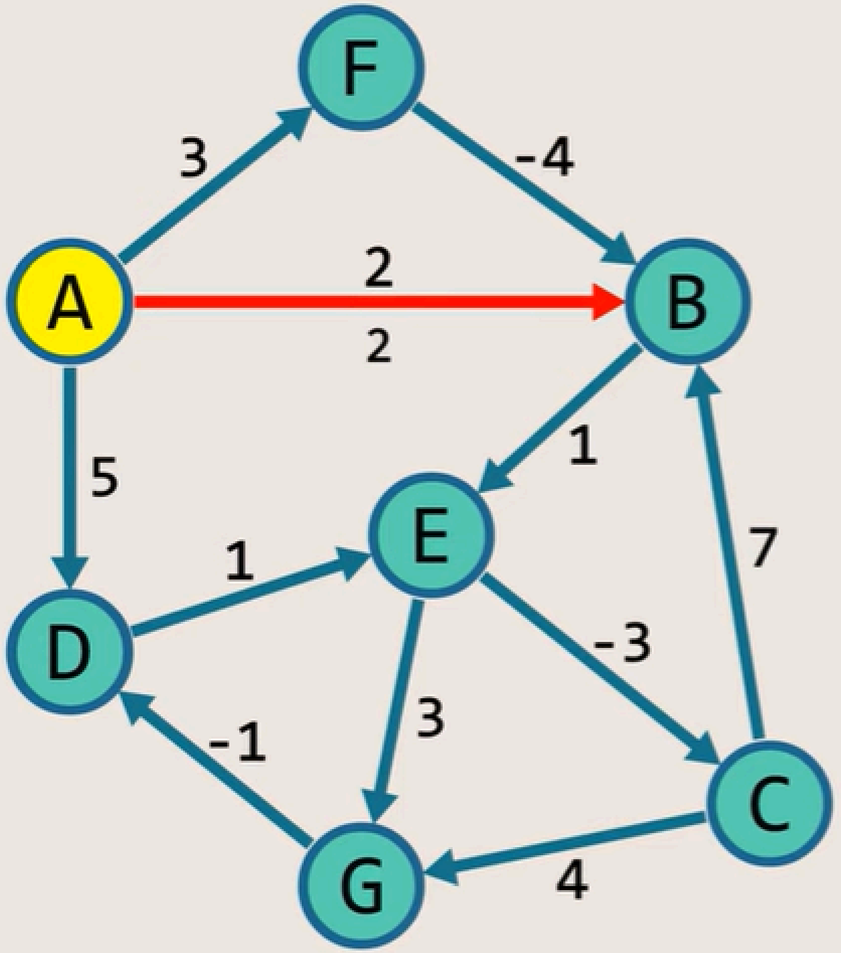
Node	Cost	Previous
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



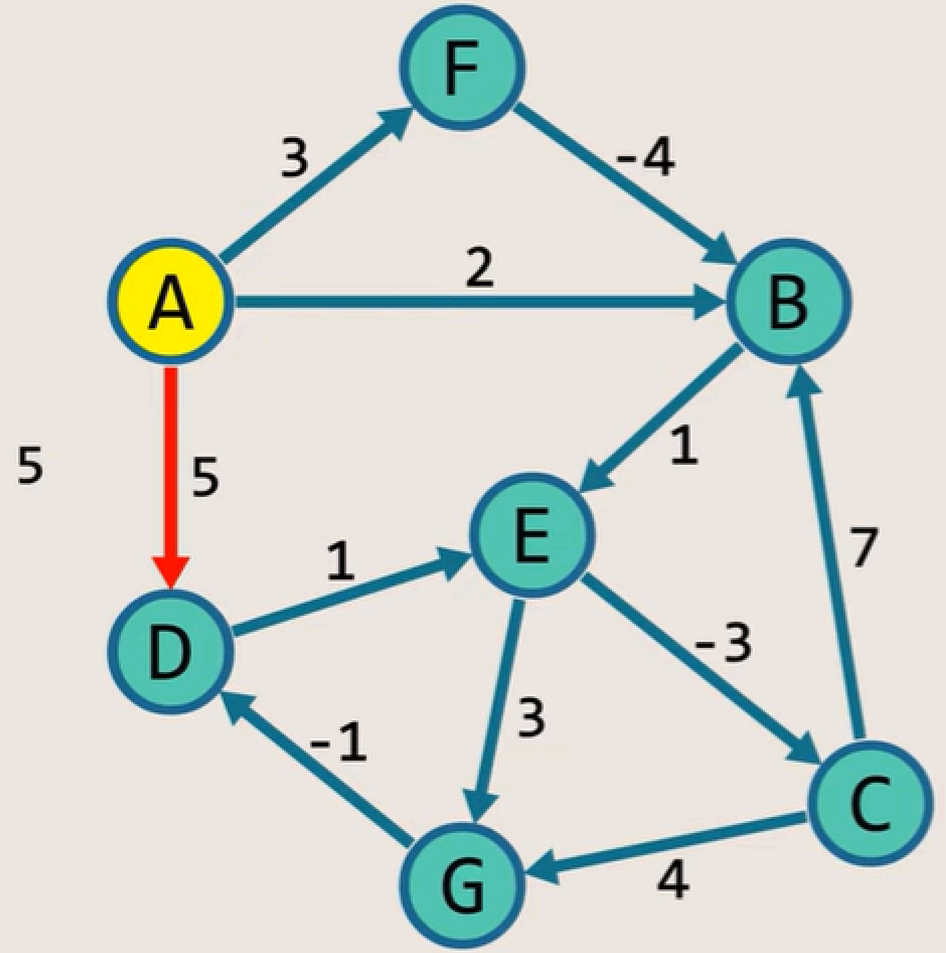
Node	Cost	Previous
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



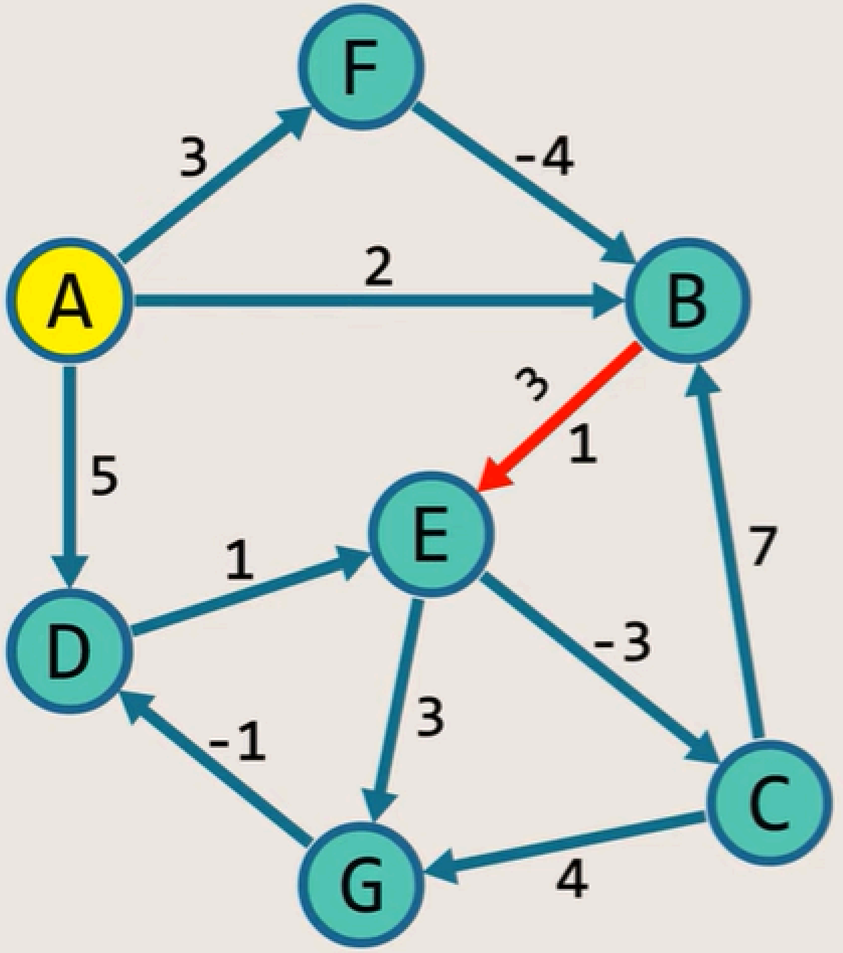
Node	Cost	Previous
A	0	
B	2	A
C	∞	
D	∞	
E	∞	
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



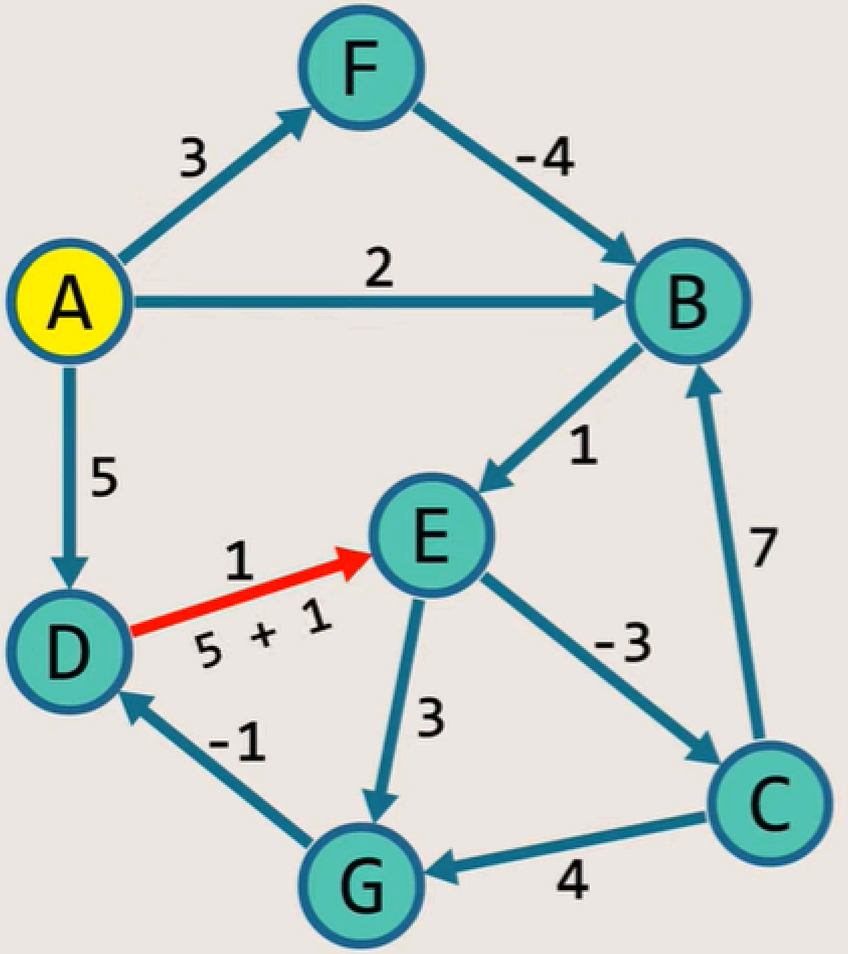
Node	Cost	Previous
A	0	
B	2	A
C	∞	
D	5	A
E	∞	
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



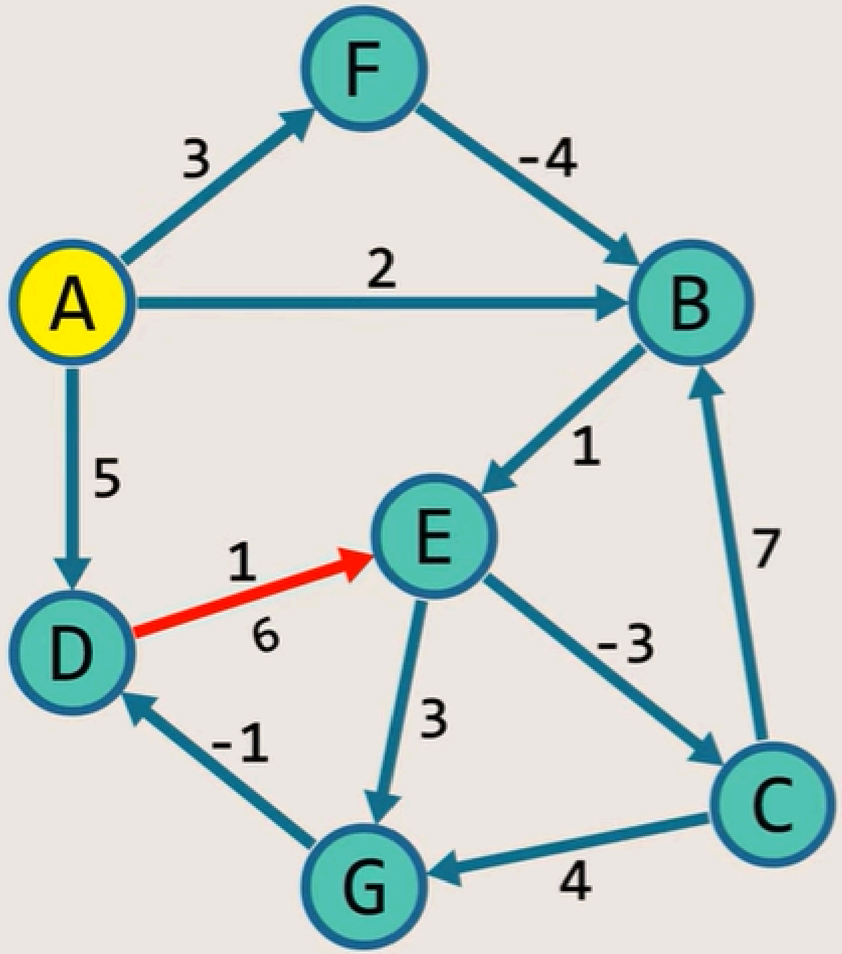
Node	Cost	Previous
A	0	
B	2	A
C	∞	
D	5	A
E	3	B
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	2	A
C	∞	
D	5	A
E	3	B
F	3	A
G	∞	

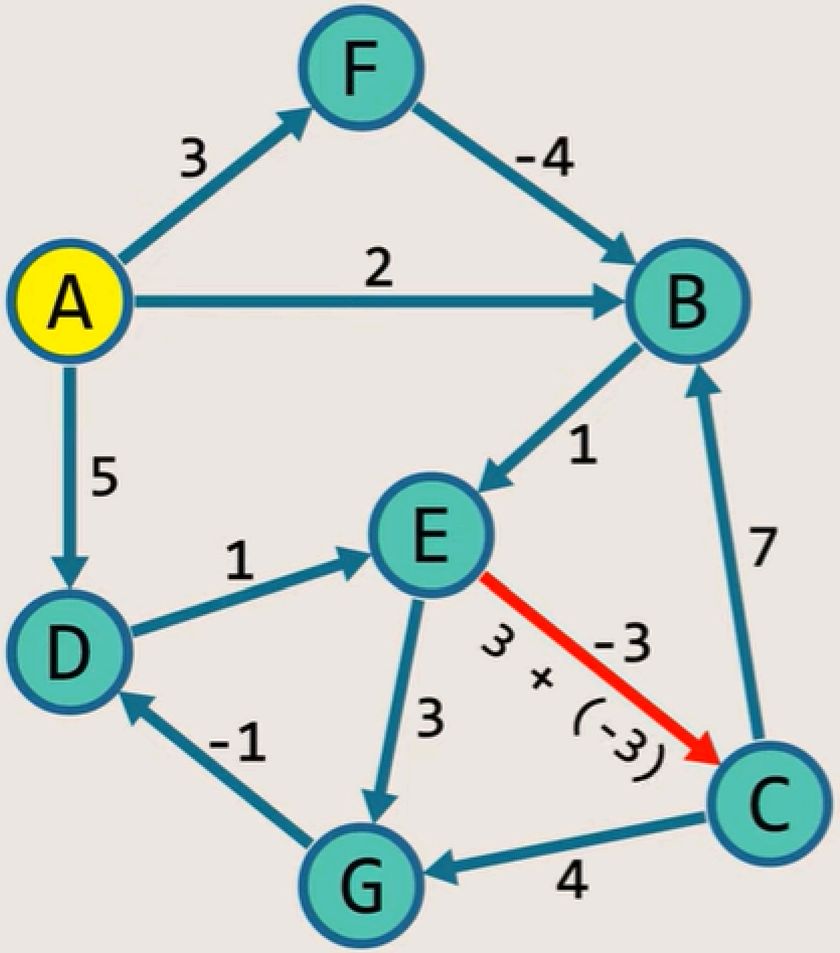
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



6 > 3.
No need to update the cost

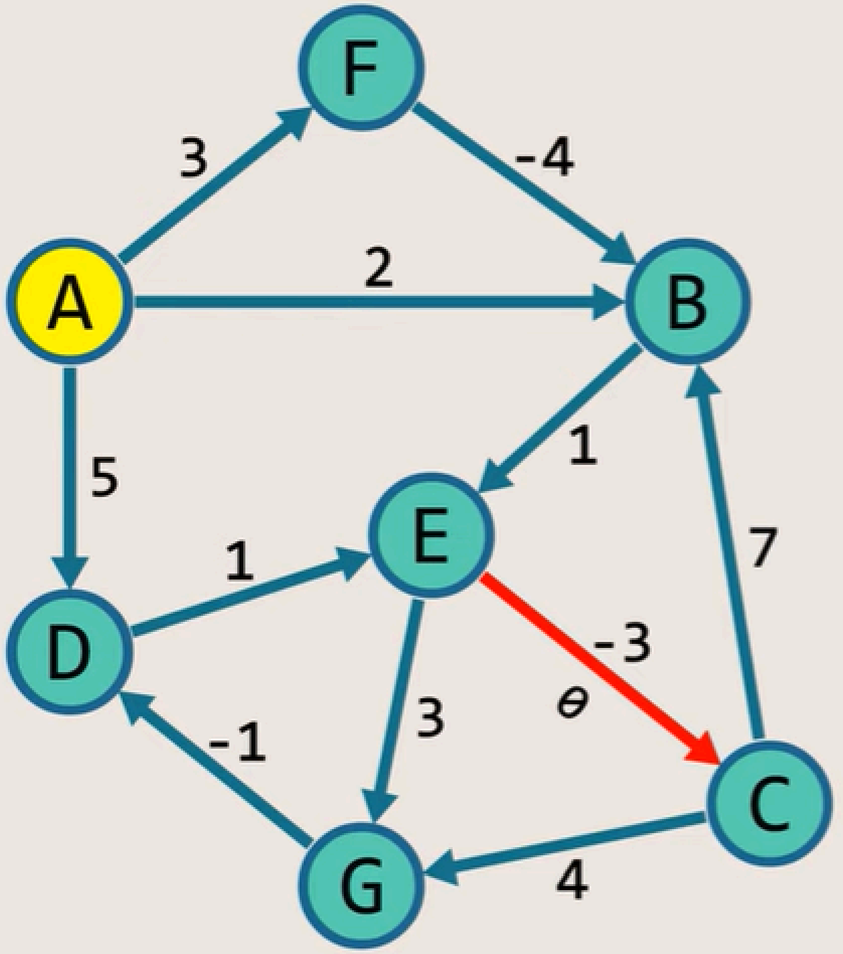
Node	Cost	Previous
A	0	
B	2	A
C	∞	
D	5	A
E	3	B
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



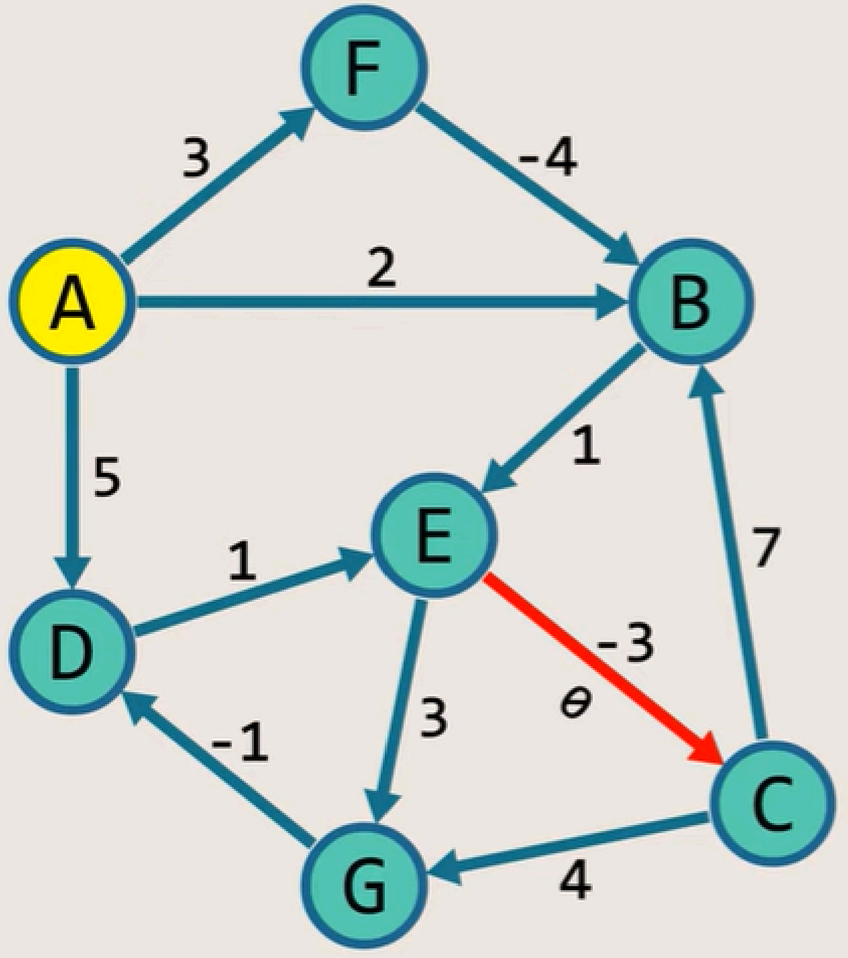
Node	Cost	Previous
A	0	
B	2	A
C	∞	
D	5	A
E	3	B
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



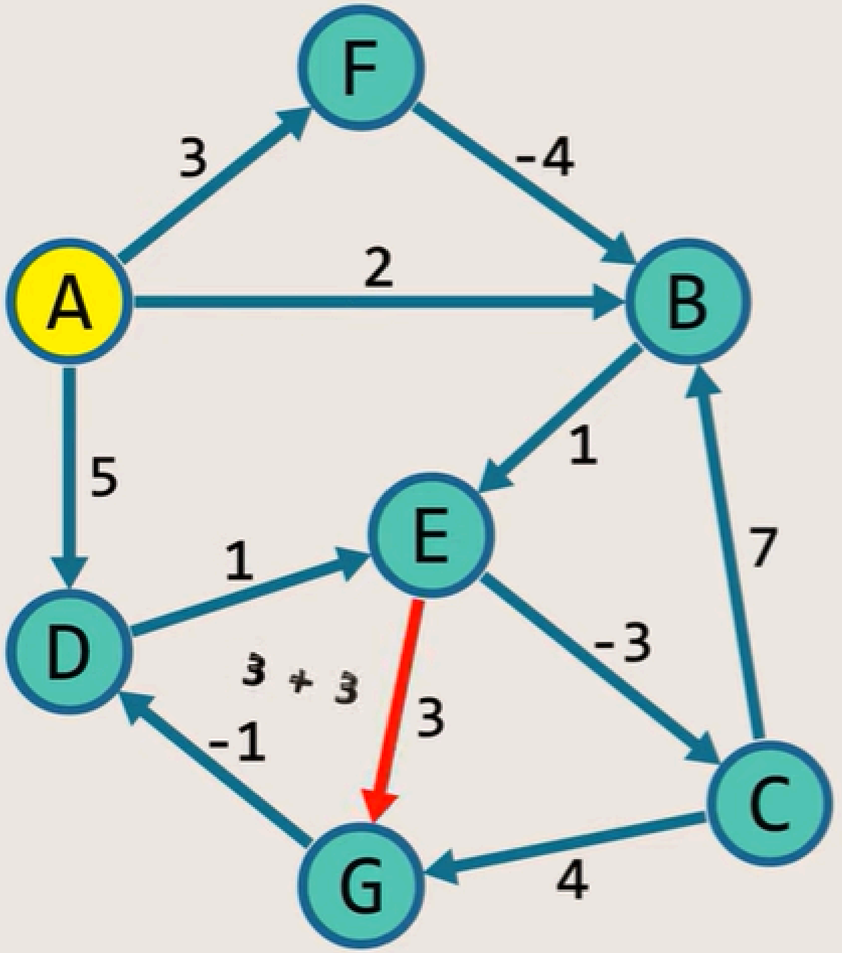
Node	Cost	Previous
A	0	
B	2	A
C	0	
D	5	A
E	3	B
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



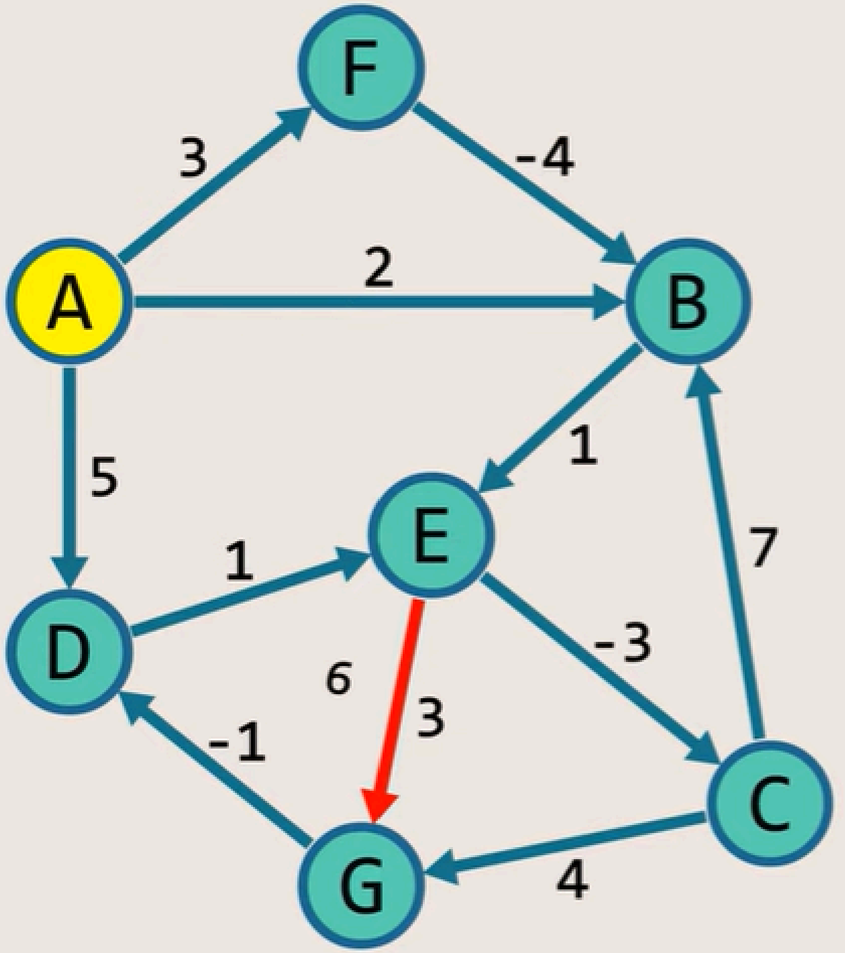
Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



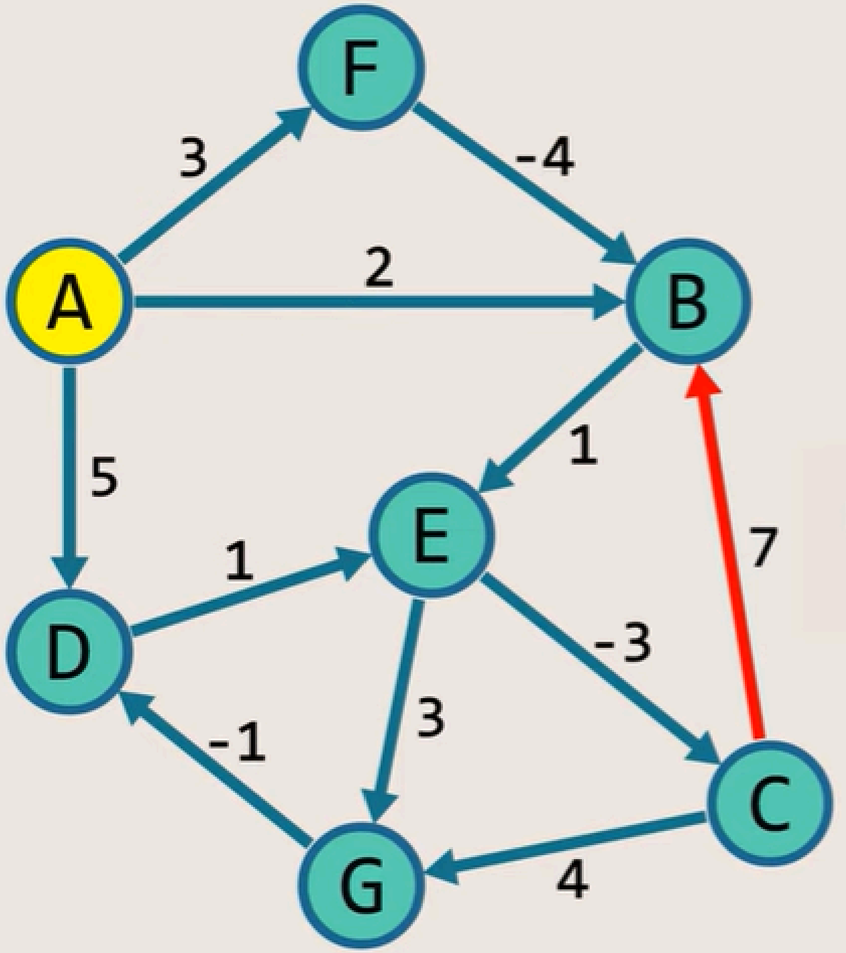
Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	∞	

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	6	E

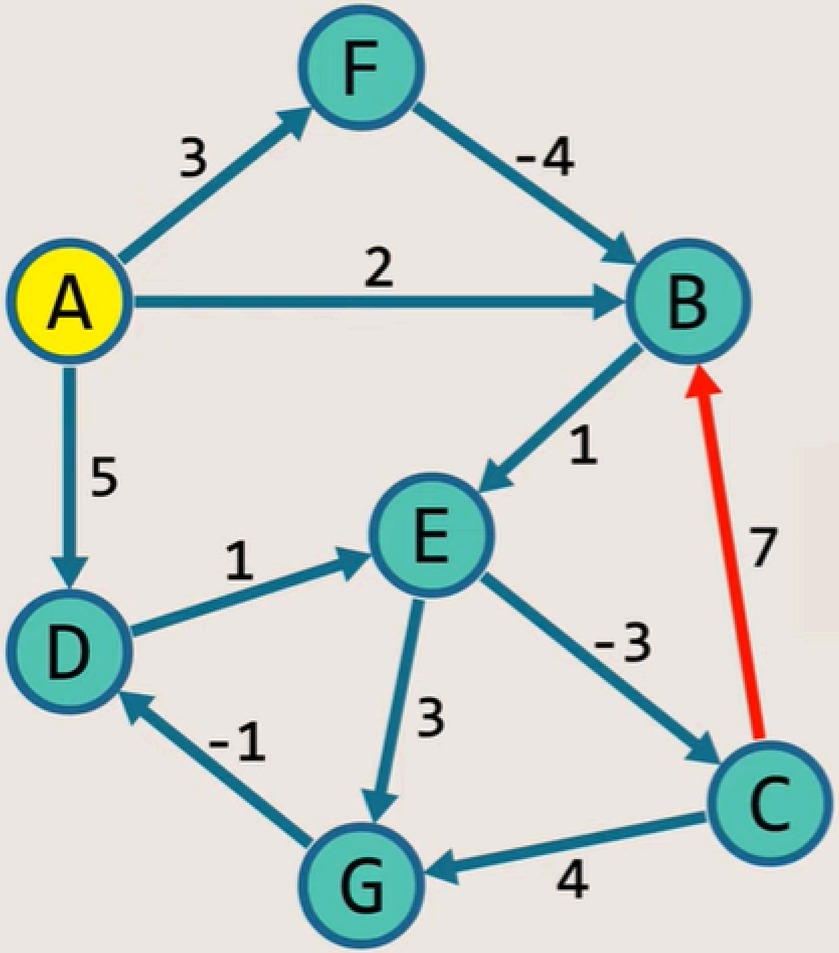
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



0 + 7

Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	6	E

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)

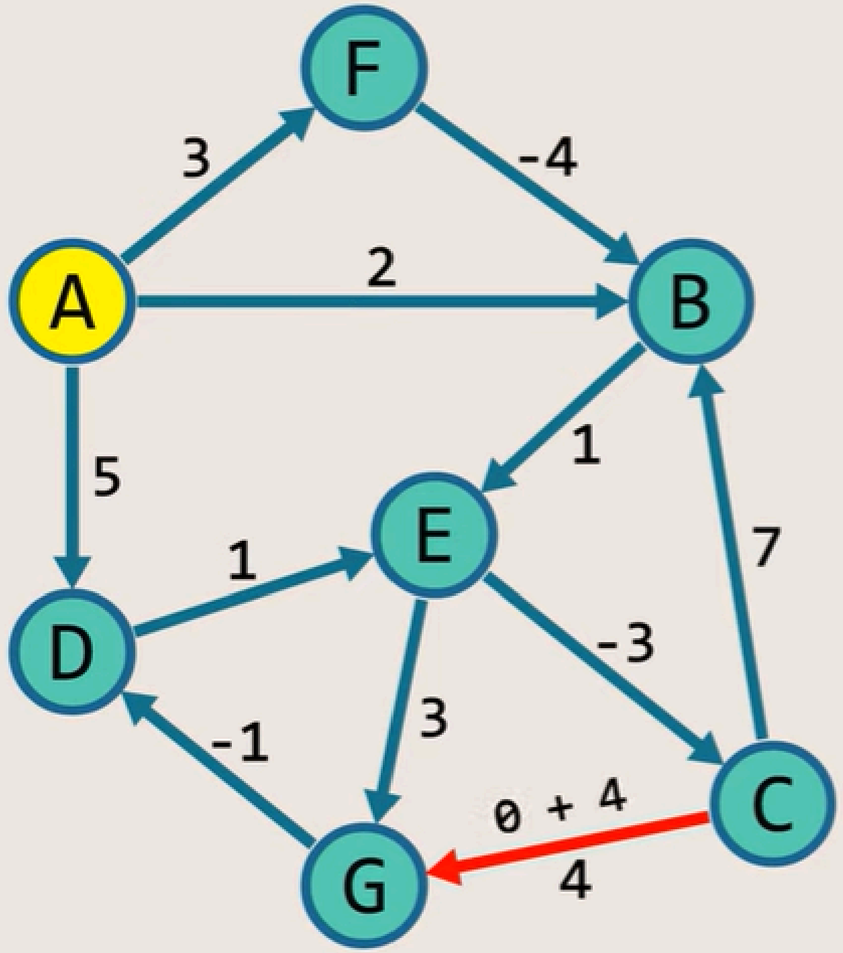


0 + 7

7 > 2.
No need to update the cost

Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	6	E

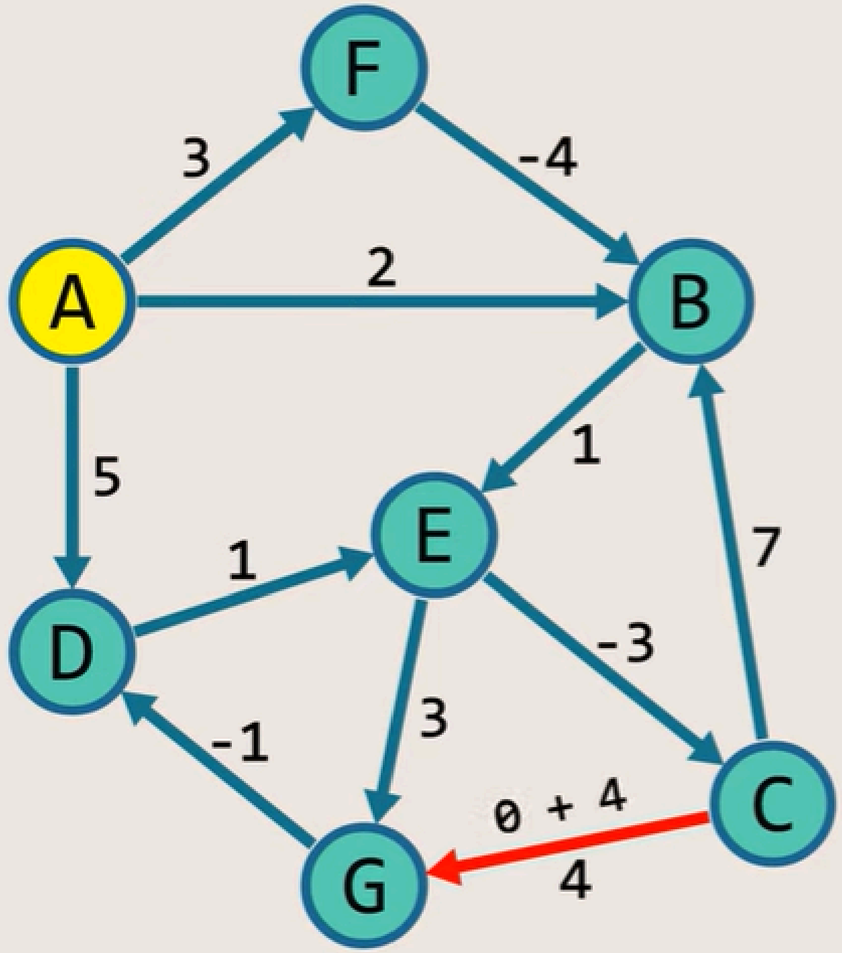
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



4<6
So, update the cost

Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	6	E

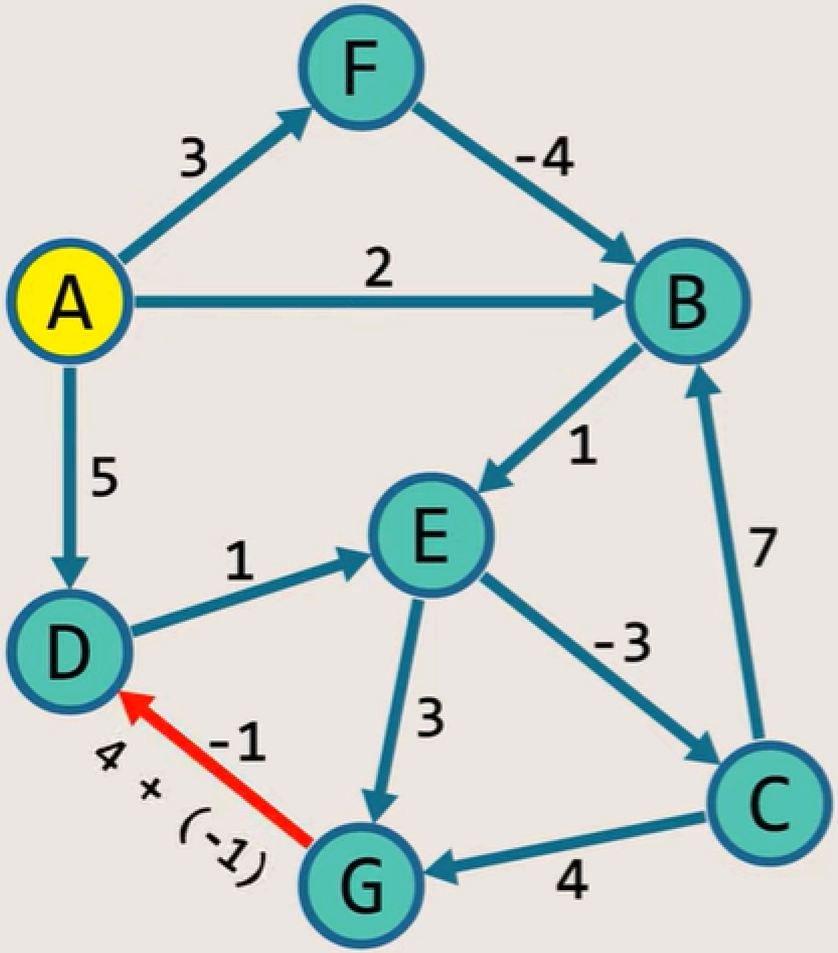
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



4<6
So, update the cost

Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	5	A
E	3	B
F	3	A
G	4	C

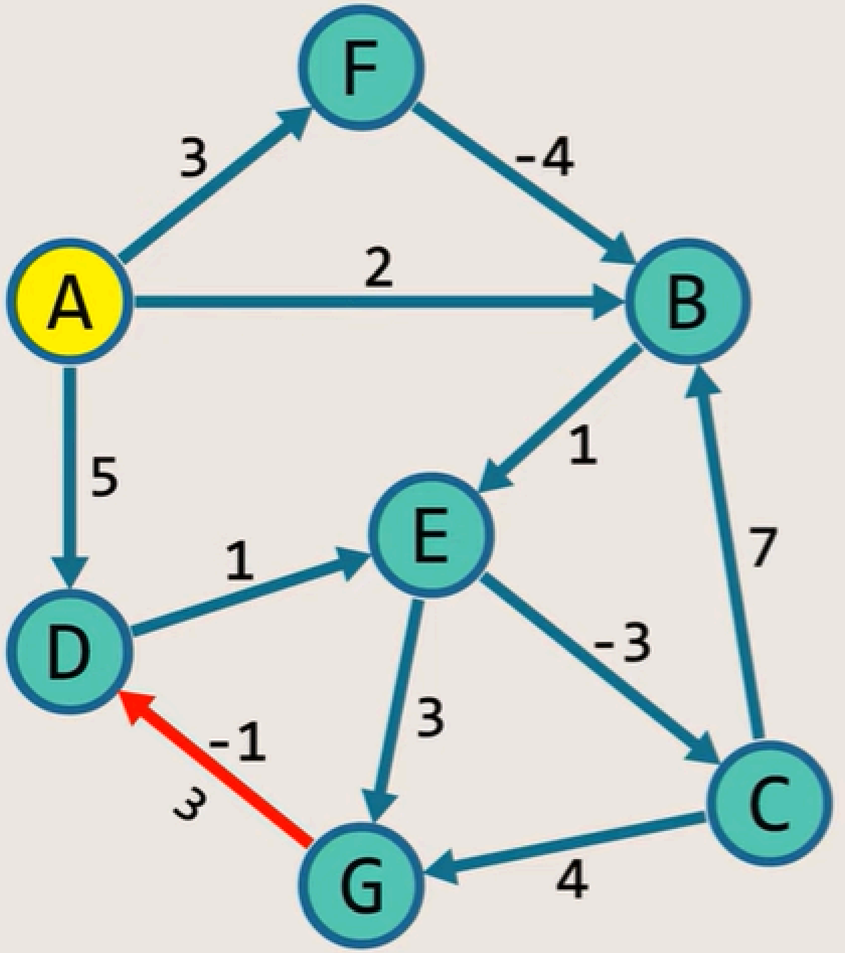
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



3<5
So, update the cost

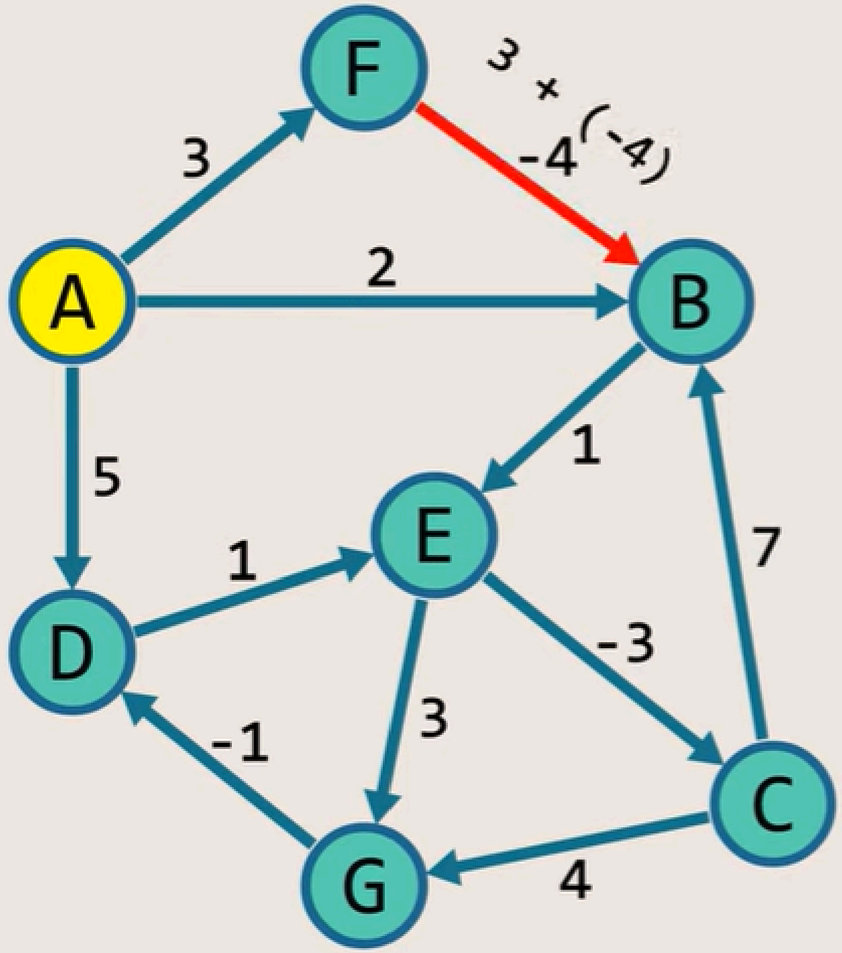
Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	~~5~~	~~A~~
E	3	B
F	3	A
G	4	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	2	A
C	0	E
D	3	G
E	3	B
F	3	A
G	4	C

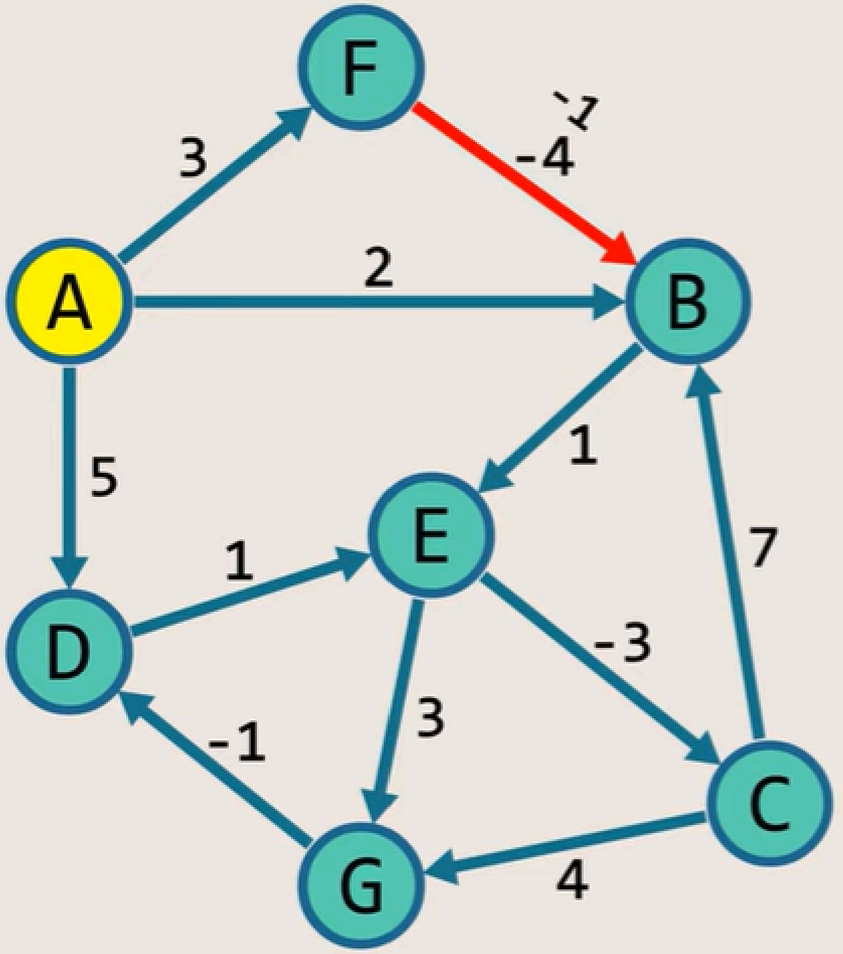
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



3+(-4) = -1
So, update the cost

Node	Cost	Previous
A	0	
B	~~2~~	~~A~~
C	0	E
D	3	G
E	3	B
F	3	A
G	4	C

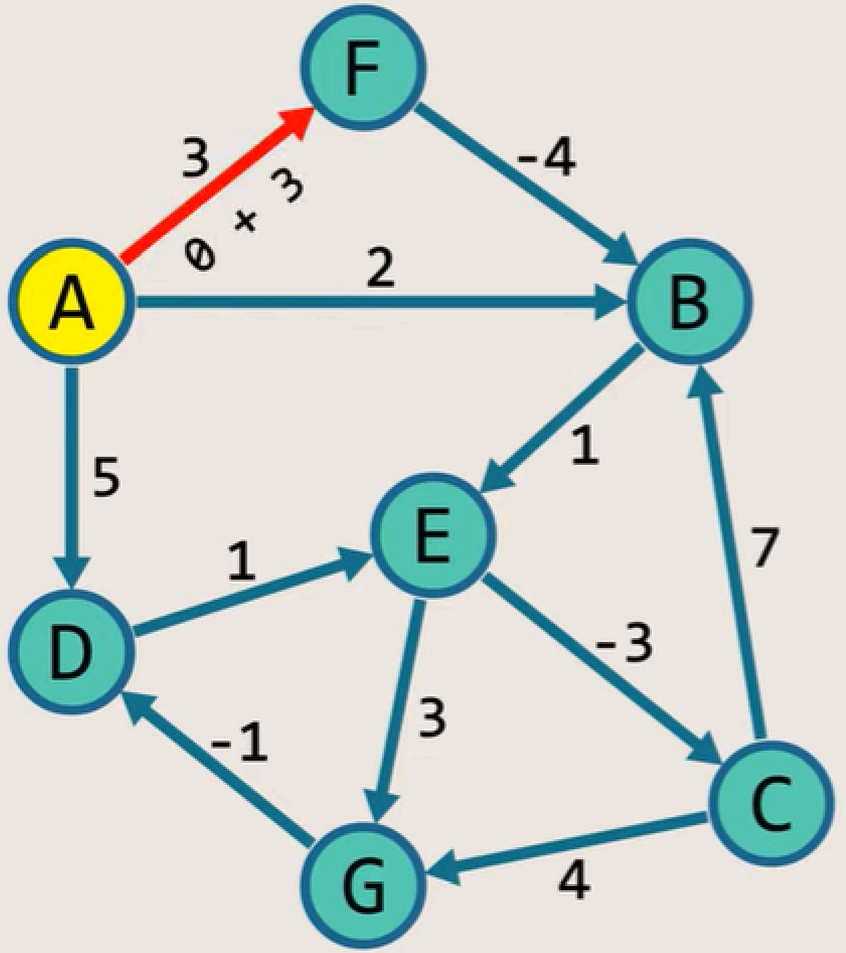
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	-1	F
C	0	E
D	3	G
E	3	B
F	3	A
G	4	C

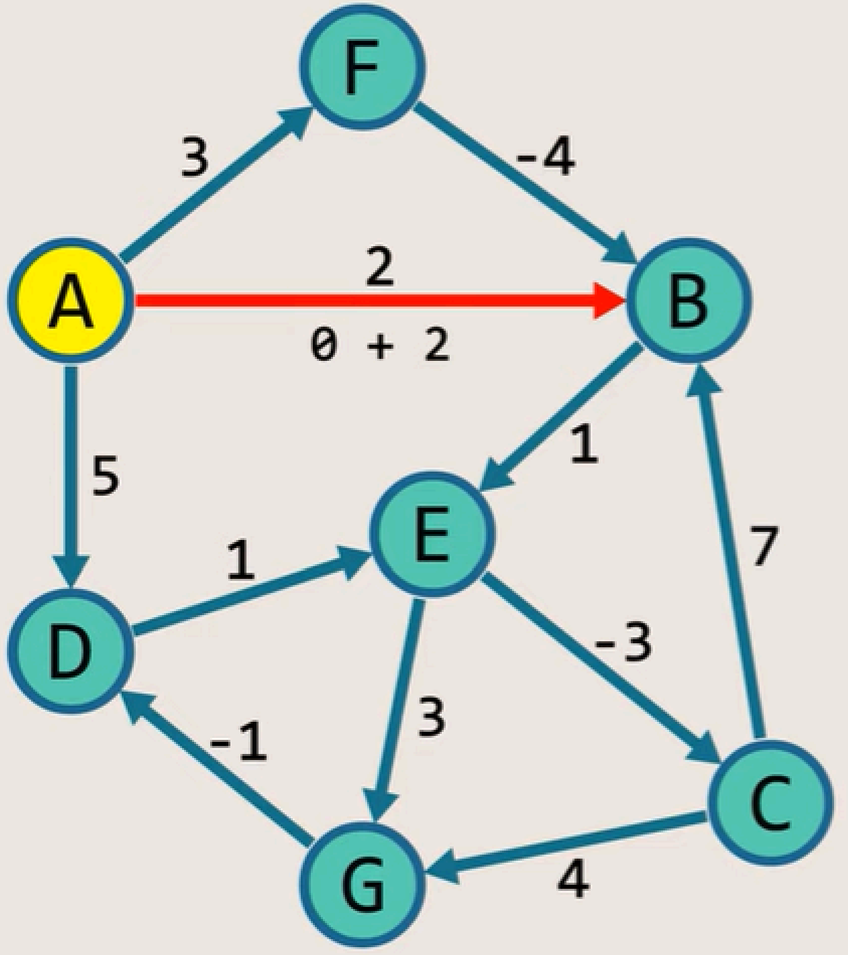
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)

Iteration 2



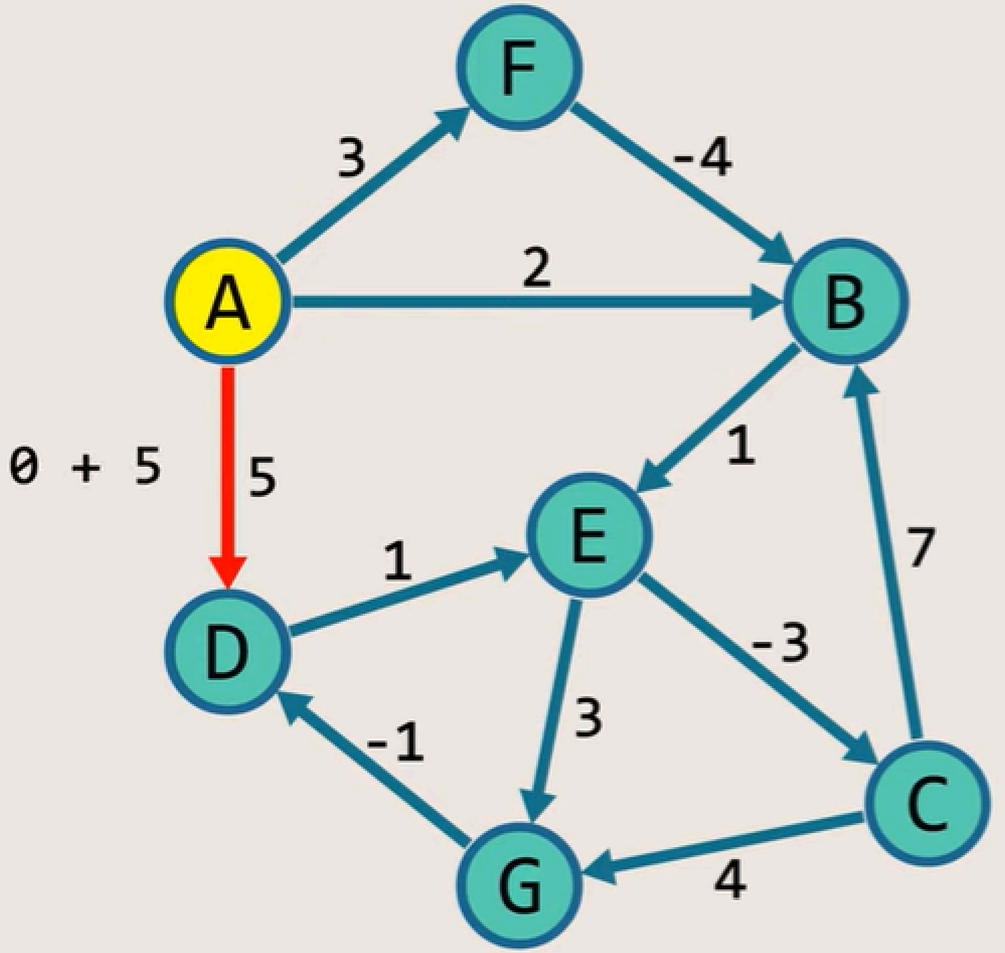
Node	Cost	Previous
A	0	
B	-1	F
C	0	E
D	3	G
E	3	B
F	3	A
G	4	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



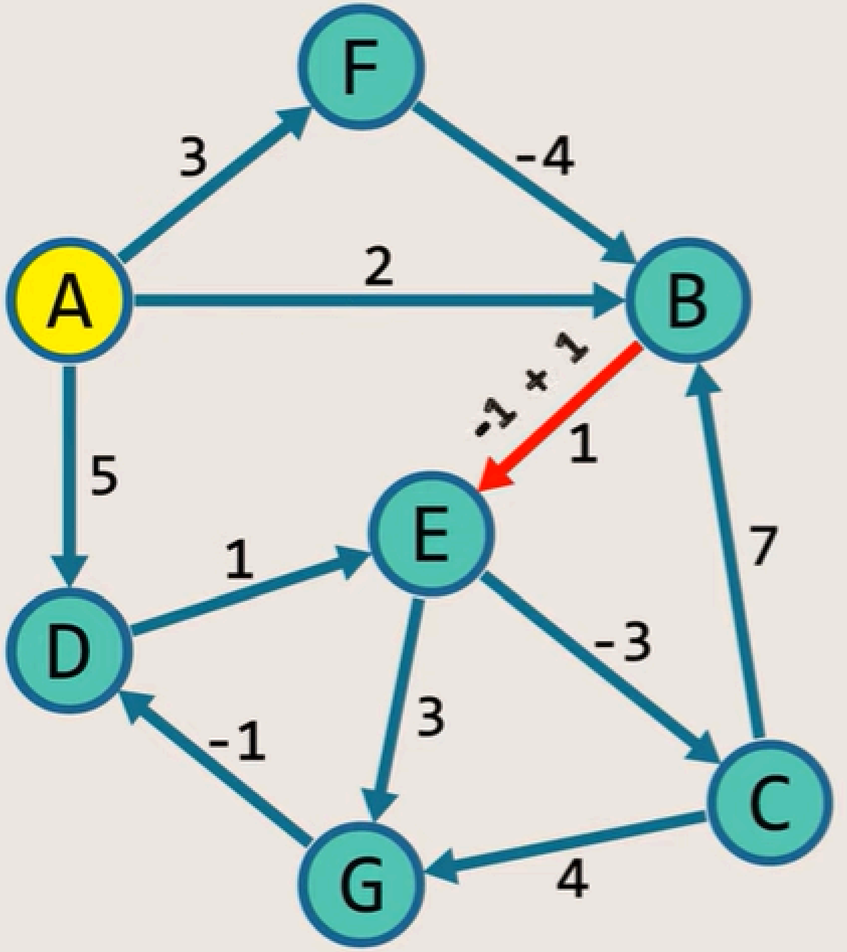
Node	Cost	Previous
A	0	
B	-1	F
C	0	E
D	3	G
E	3	B
F	3	A
G	4	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



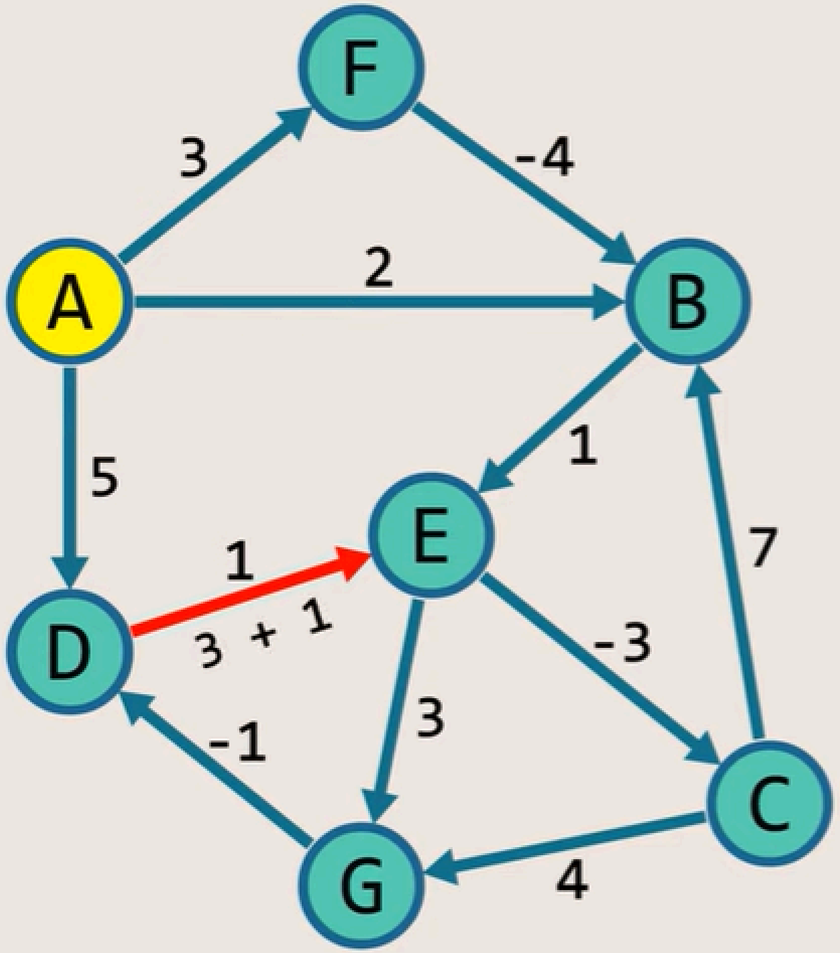
Node	Cost	Previous
A	0	
B	-1	F
C	0	E
D	3	G
E	3	B
F	3	A
G	4	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	-1	F
C	0	E
D	3	G
E	0	B
F	3	A
G	4	C

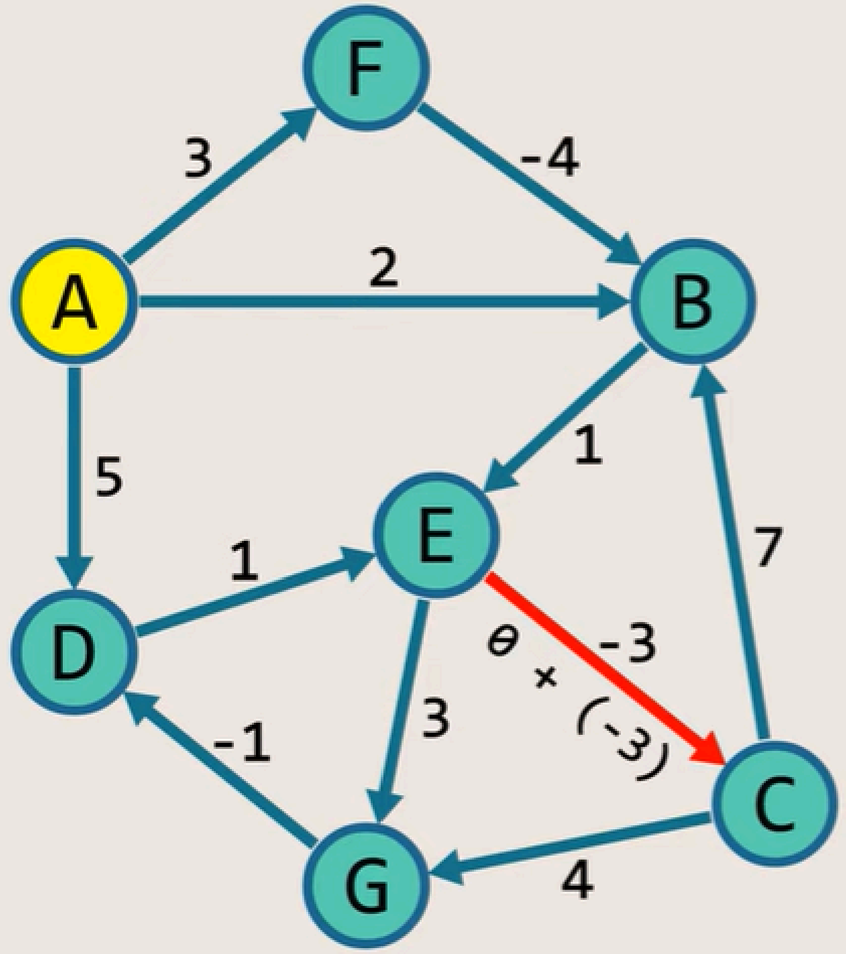
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



4 > 0.
No Update

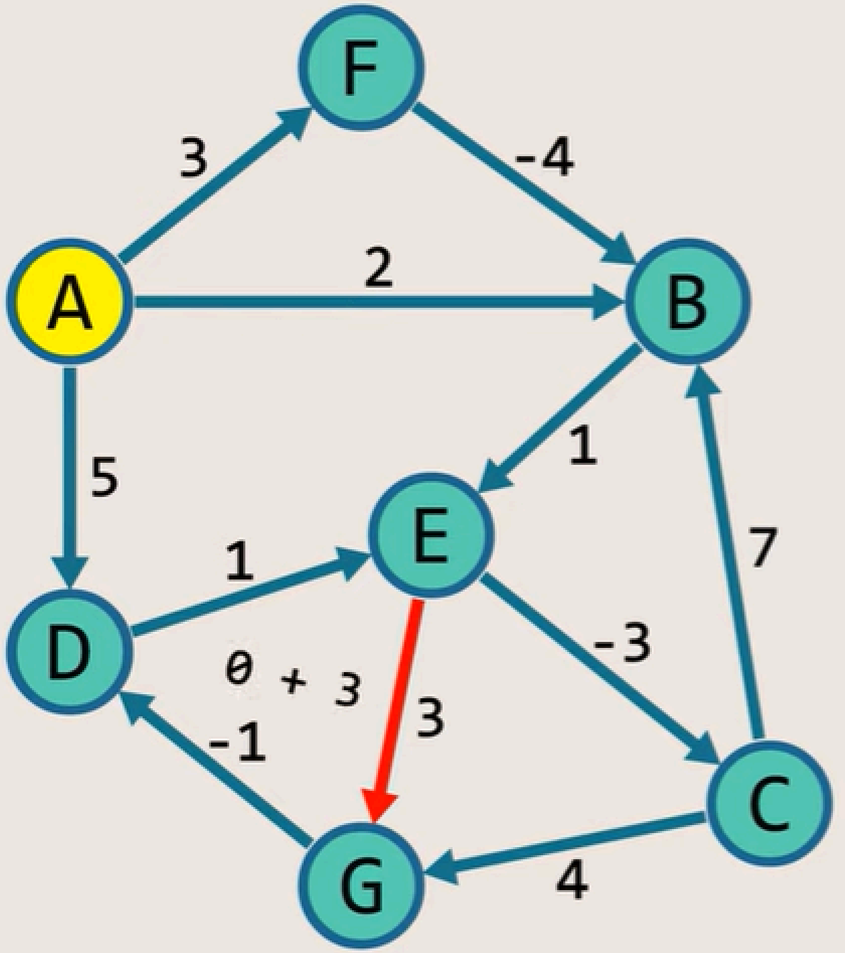
Node	Cost	Previous
A	0	
B	-1	F
C	0	E
D	3	G
E	0	B
F	3	A
G	4	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



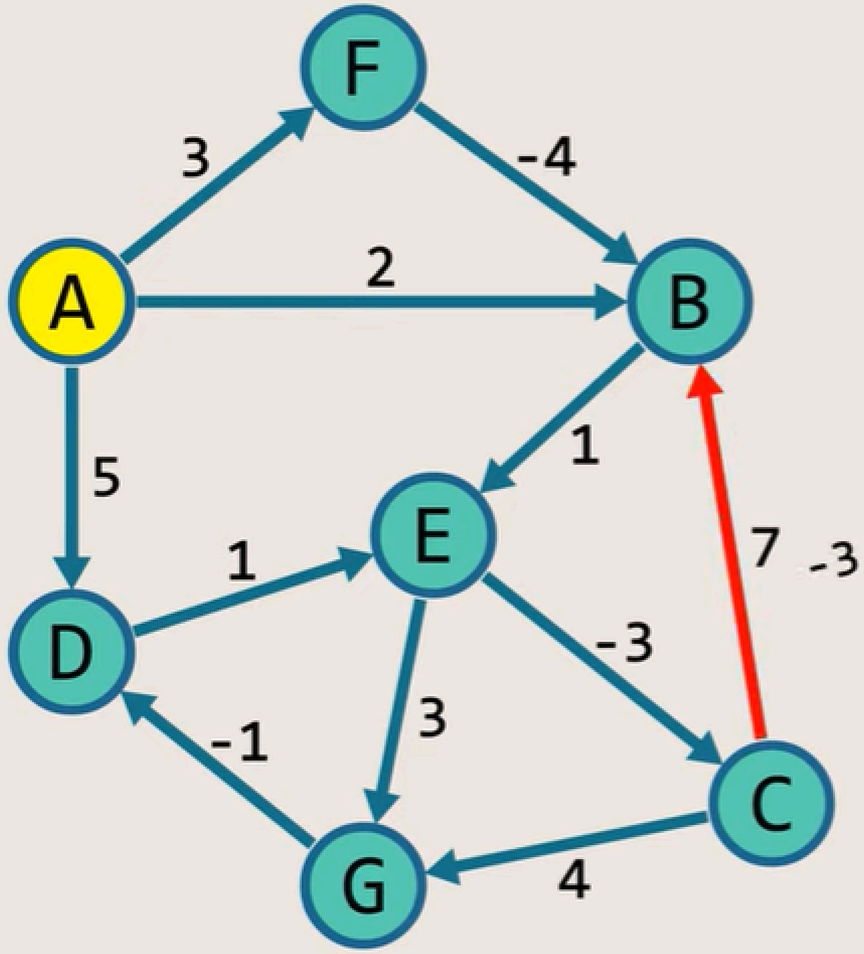
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	3	G
E	0	B
F	3	A
G	4	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	3	G
E	0	B
F	3	A
G	3	E

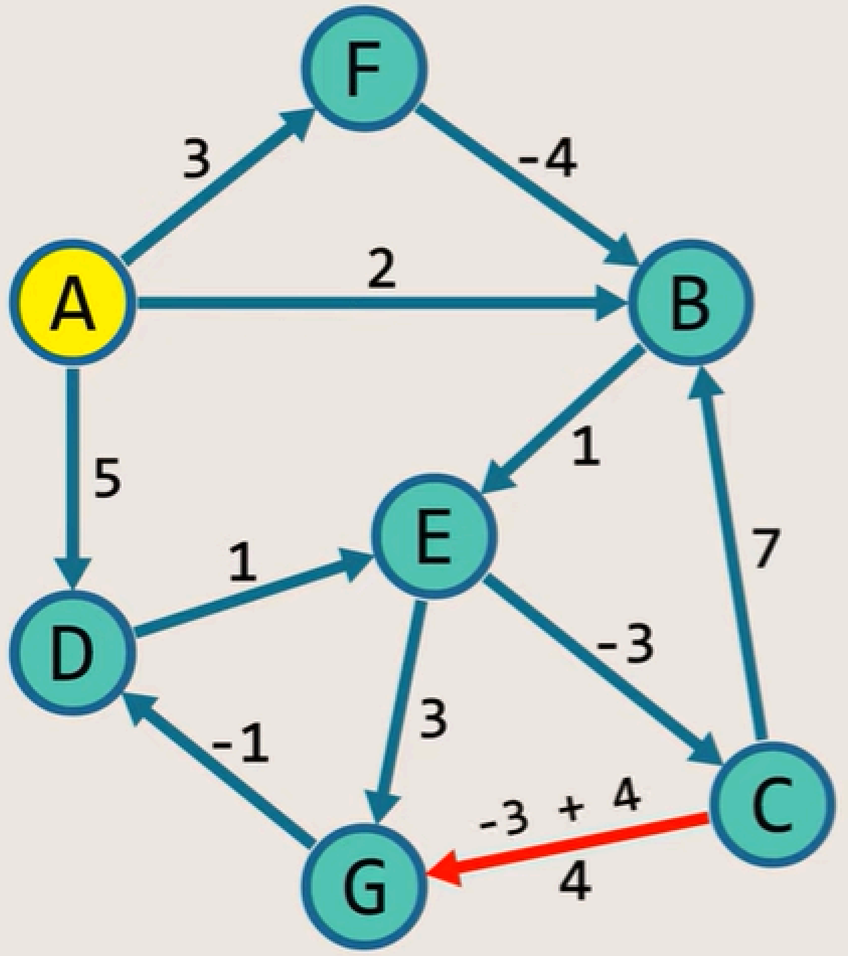
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



7 > -1.
No Update

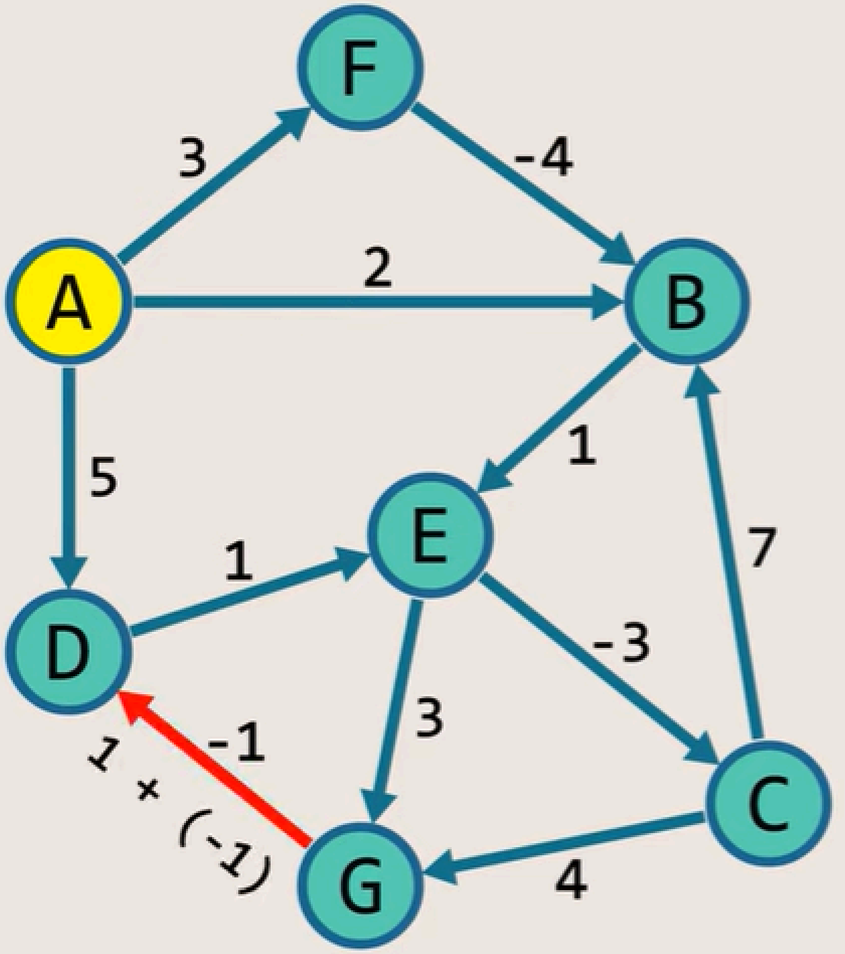
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	3	G
E	0	B
F	3	A
G	3	E

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



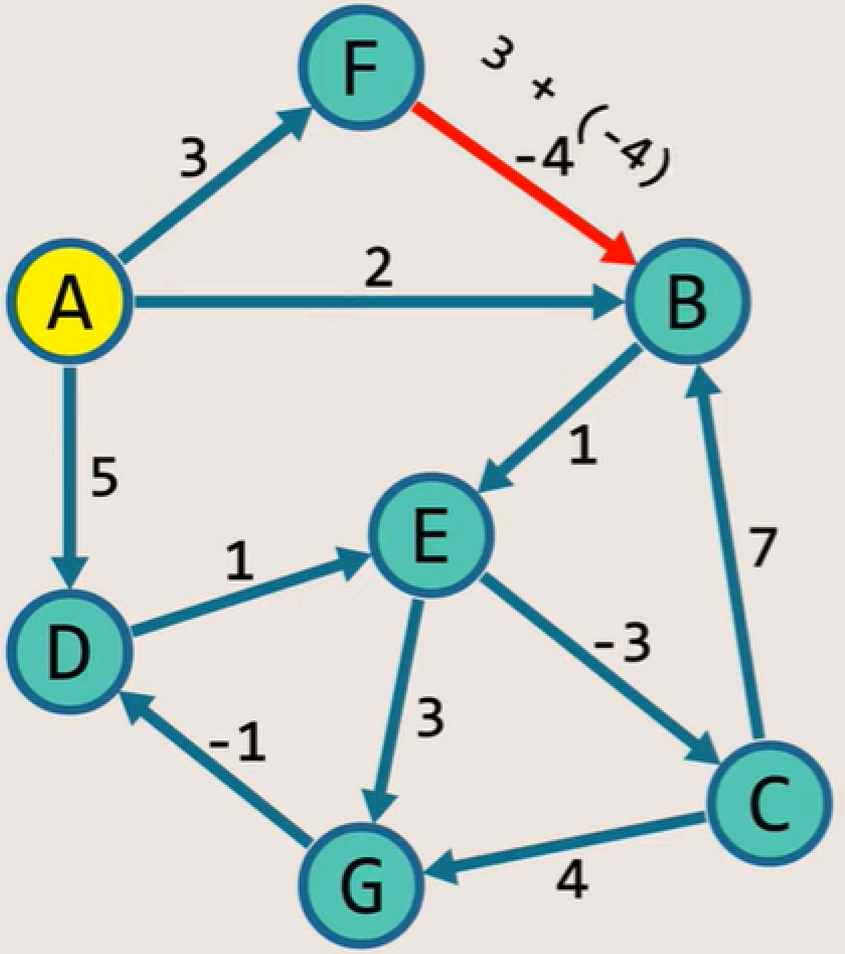
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	3	G
E	0	B
F	3	A
G	1	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

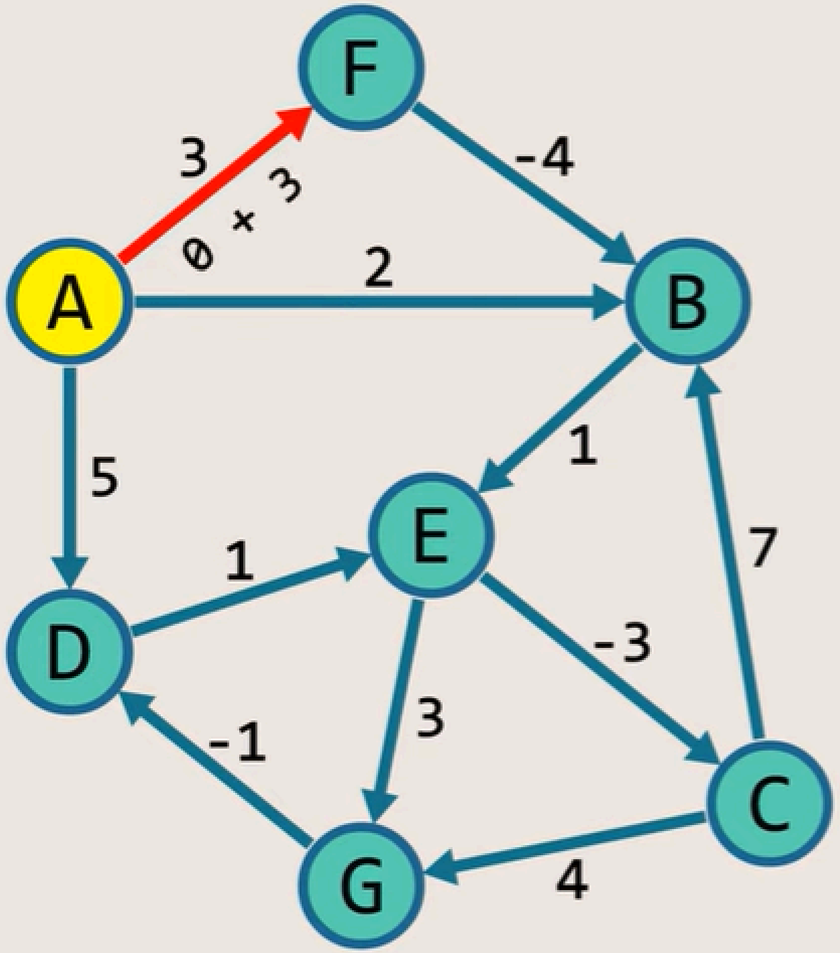
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

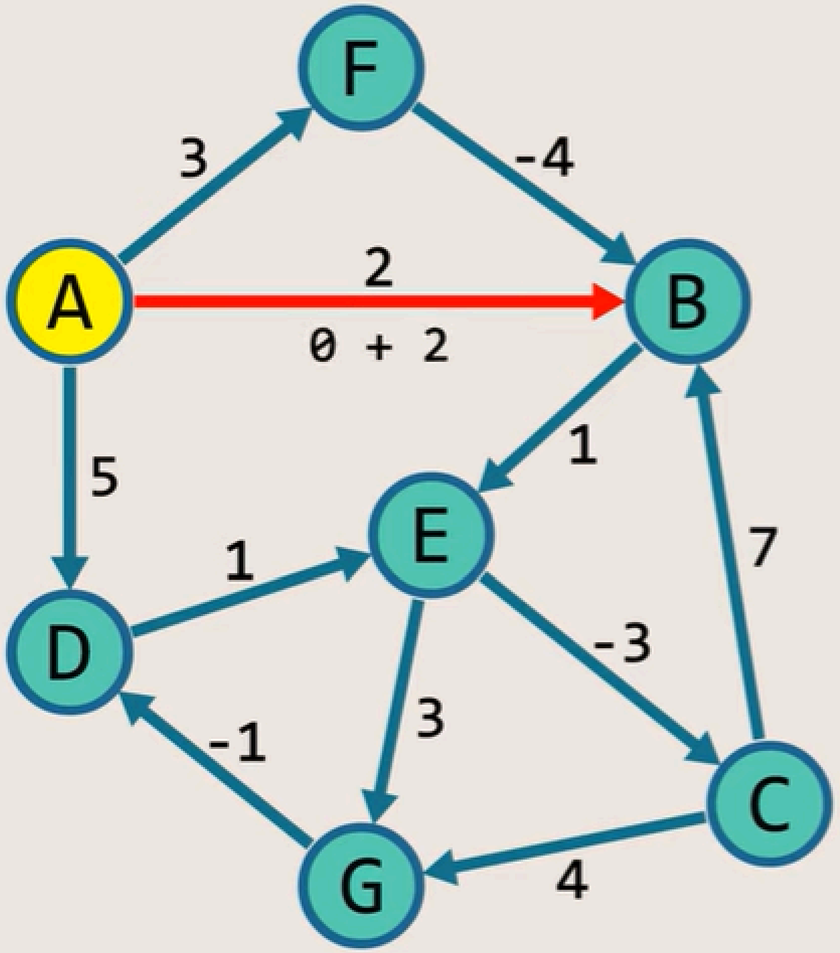
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)

Iteration 3



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

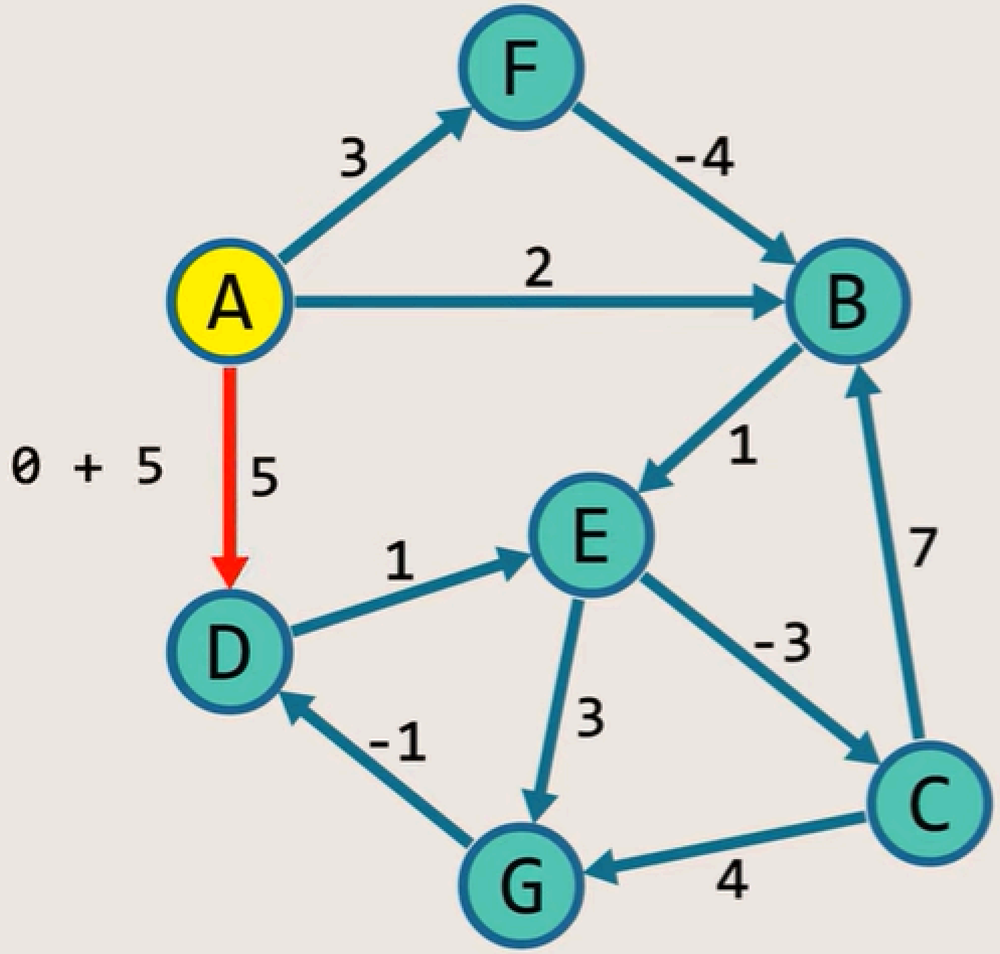
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



2 > -1.
No Update

Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

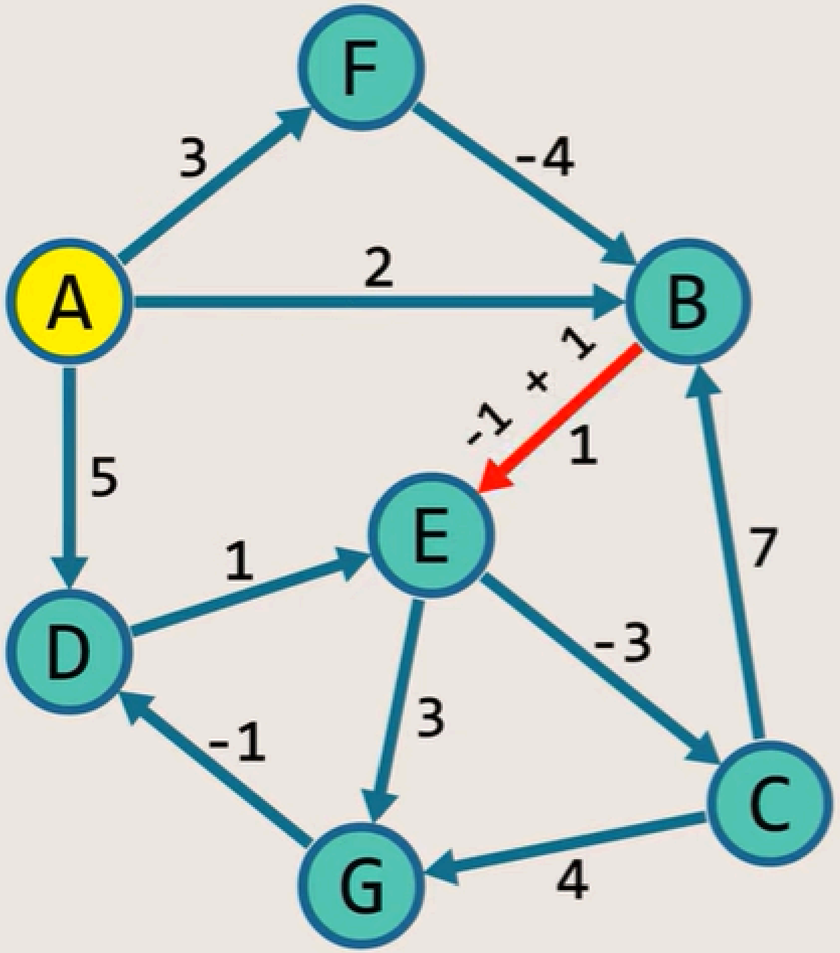
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



5 > 0.
No Update

Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

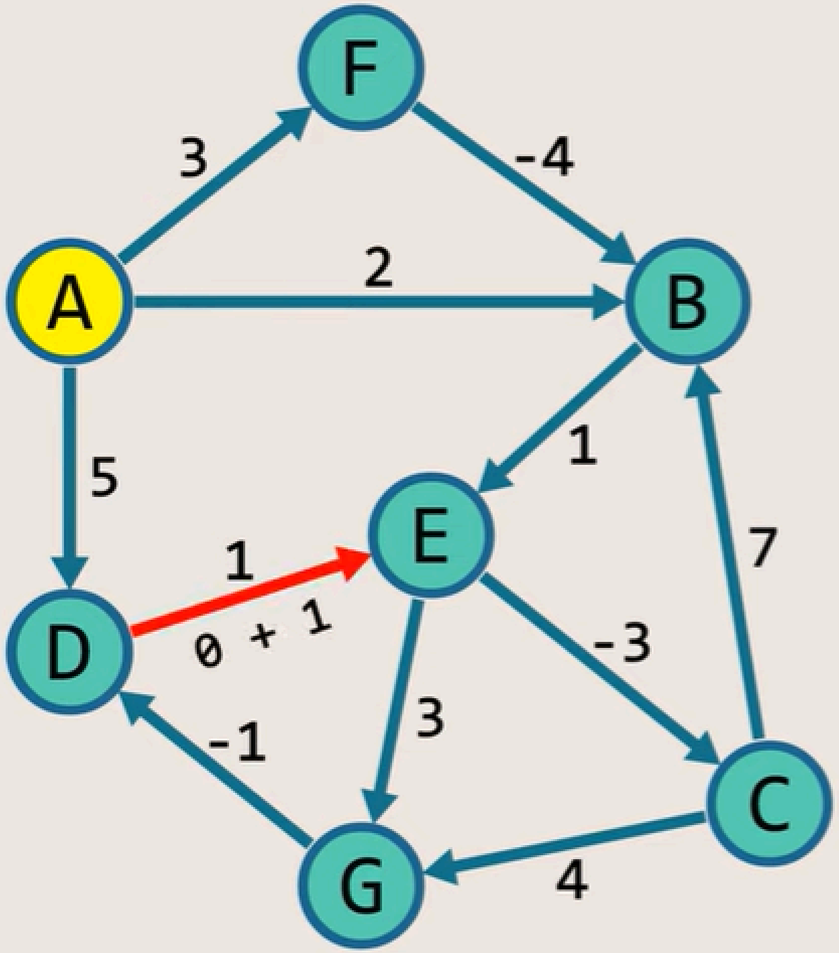
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



0 >= 0.
No Update

Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

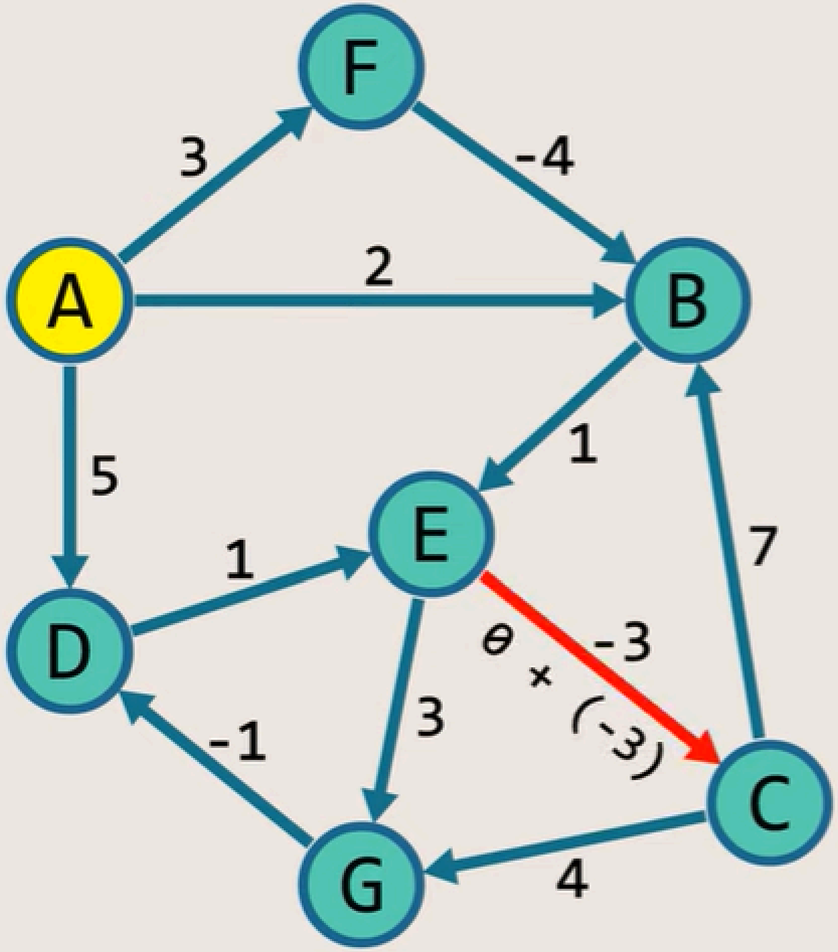
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



1 > 0.
No Update

Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

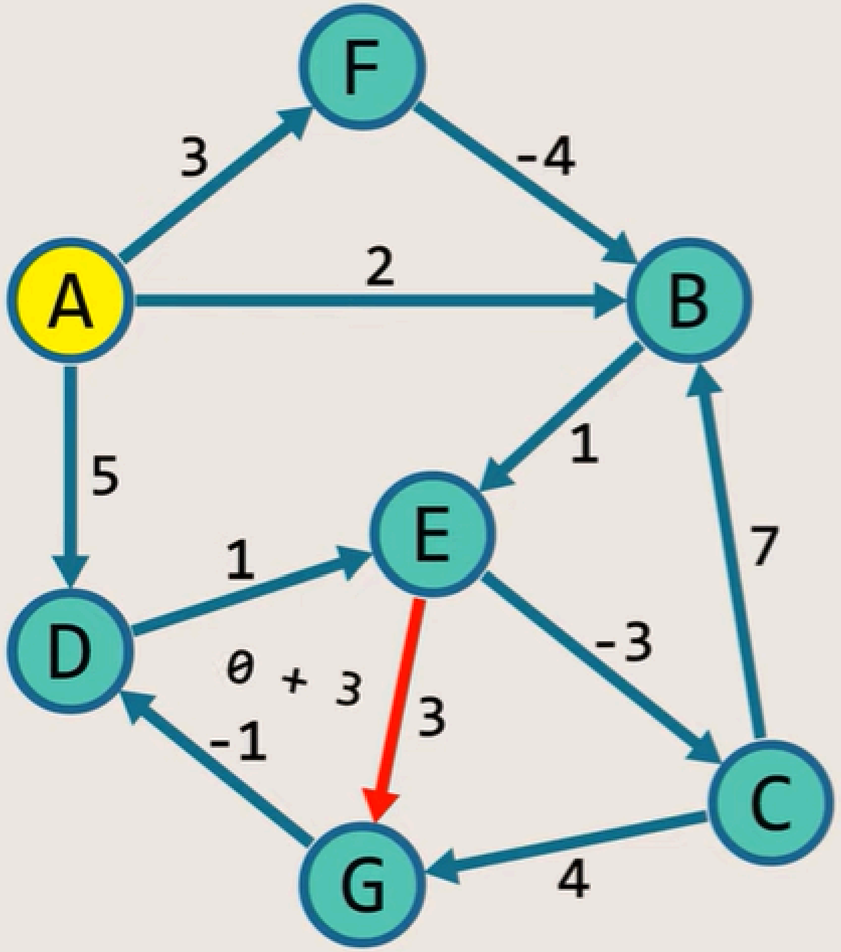
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



-3 >= -3.
No Update

Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

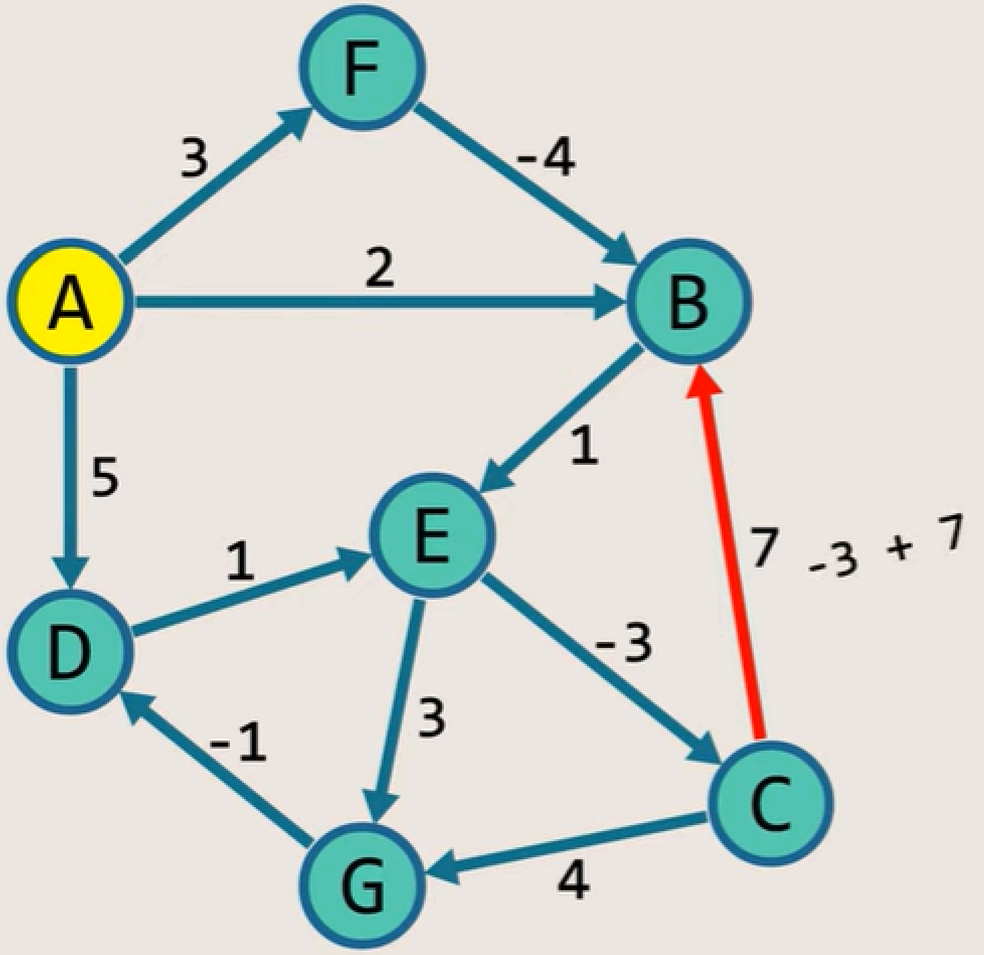
(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



3 > 1.
No Update

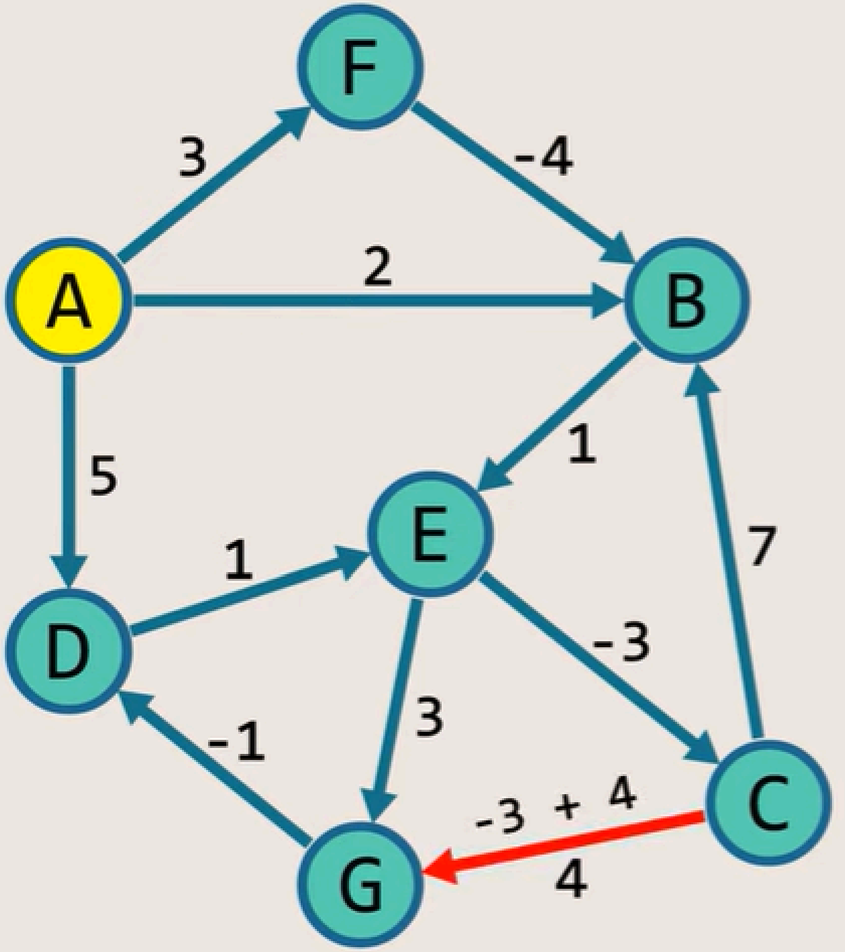
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



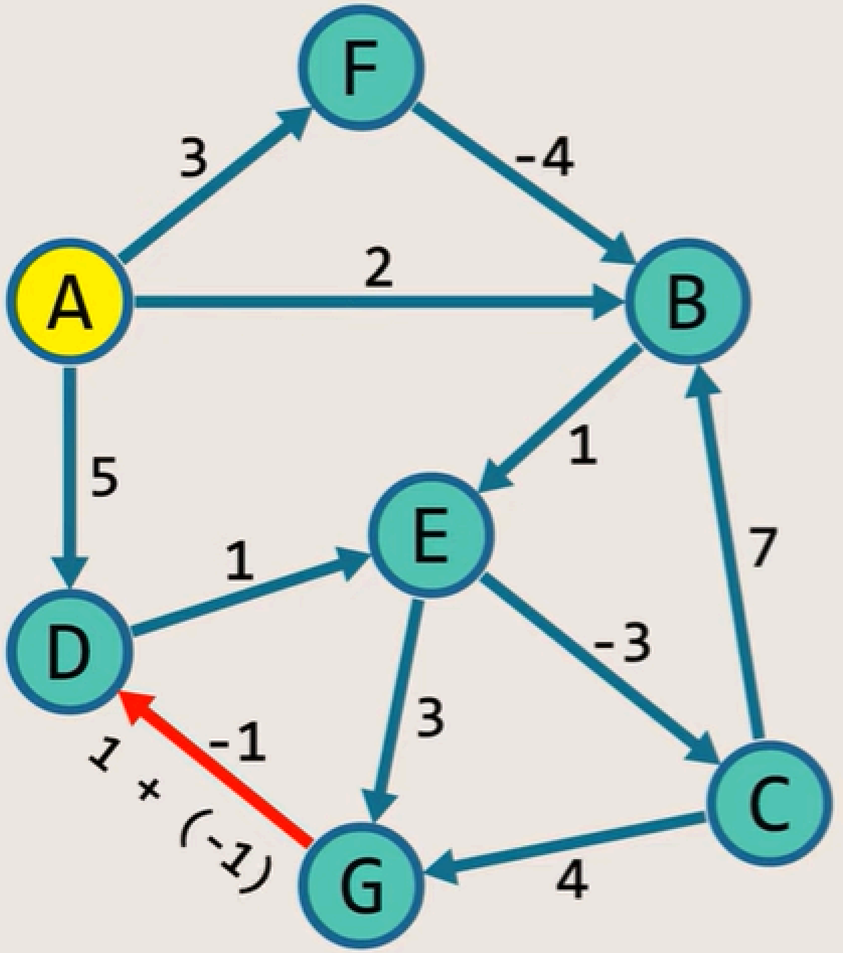
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



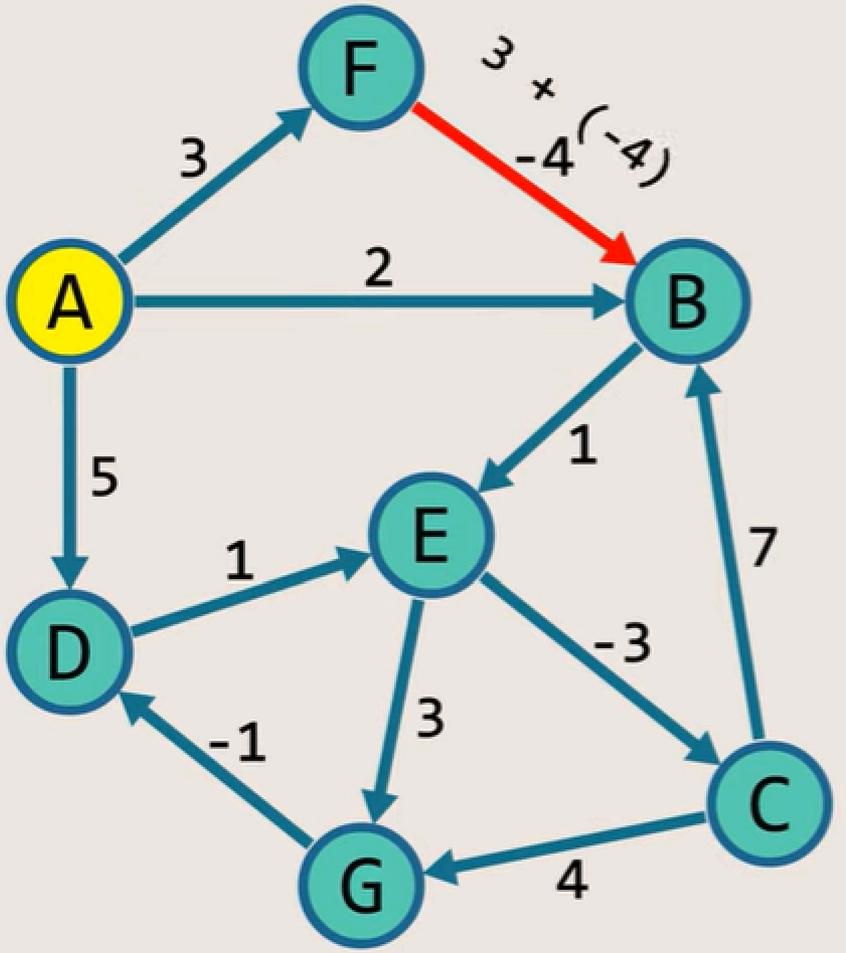
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)



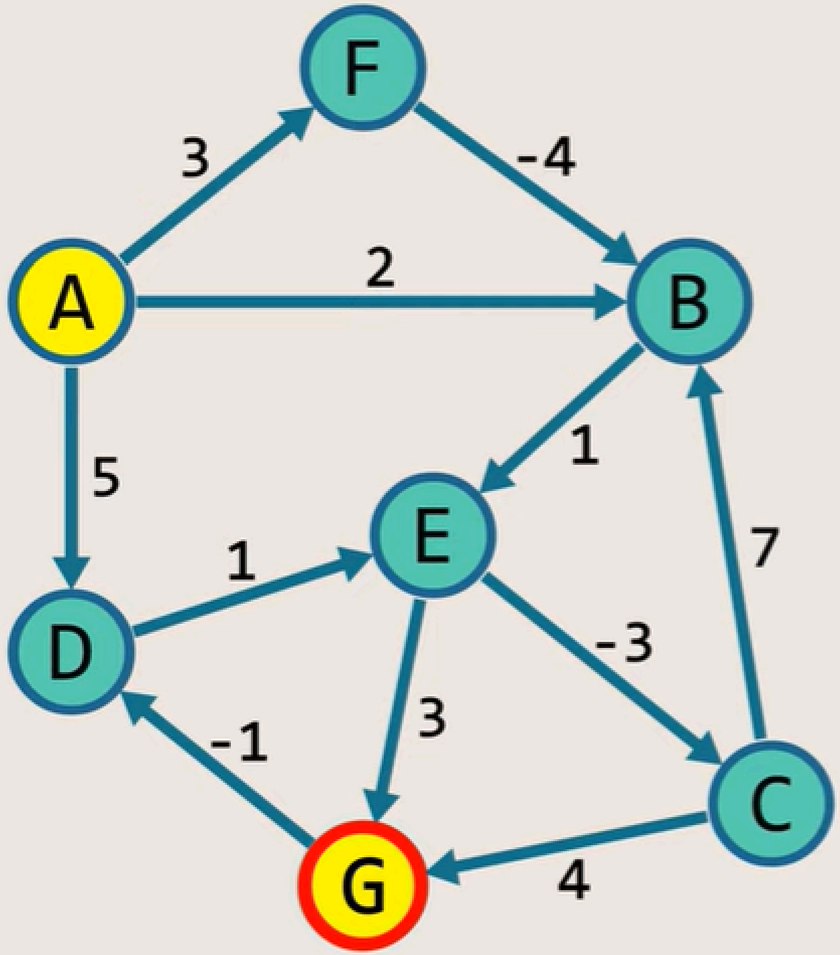
Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

(A-F) (A-B) (A-D) (B-E) (D-E) (E-C) (E-G) (C-B) (C-G) (G-D) (F-B)

Iteration 3 is Over.

- In this Iteration, no further updates occur.
- Means, there **w'ont** be any more relaxations in the later iterations and **no updates** during the negative weight cycle detection either
- So there is no need to check further.
- We can stop here and return the results.

- Lets use this result to find the shortest path from A to G

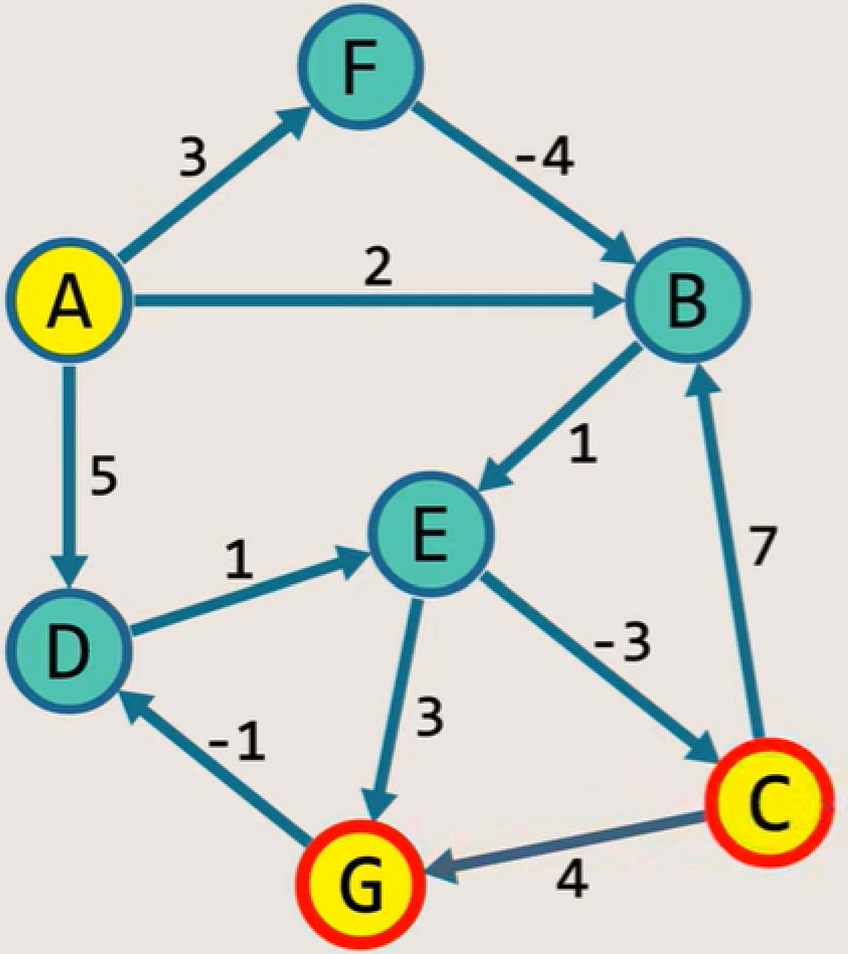


Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

Check the Previous node of G

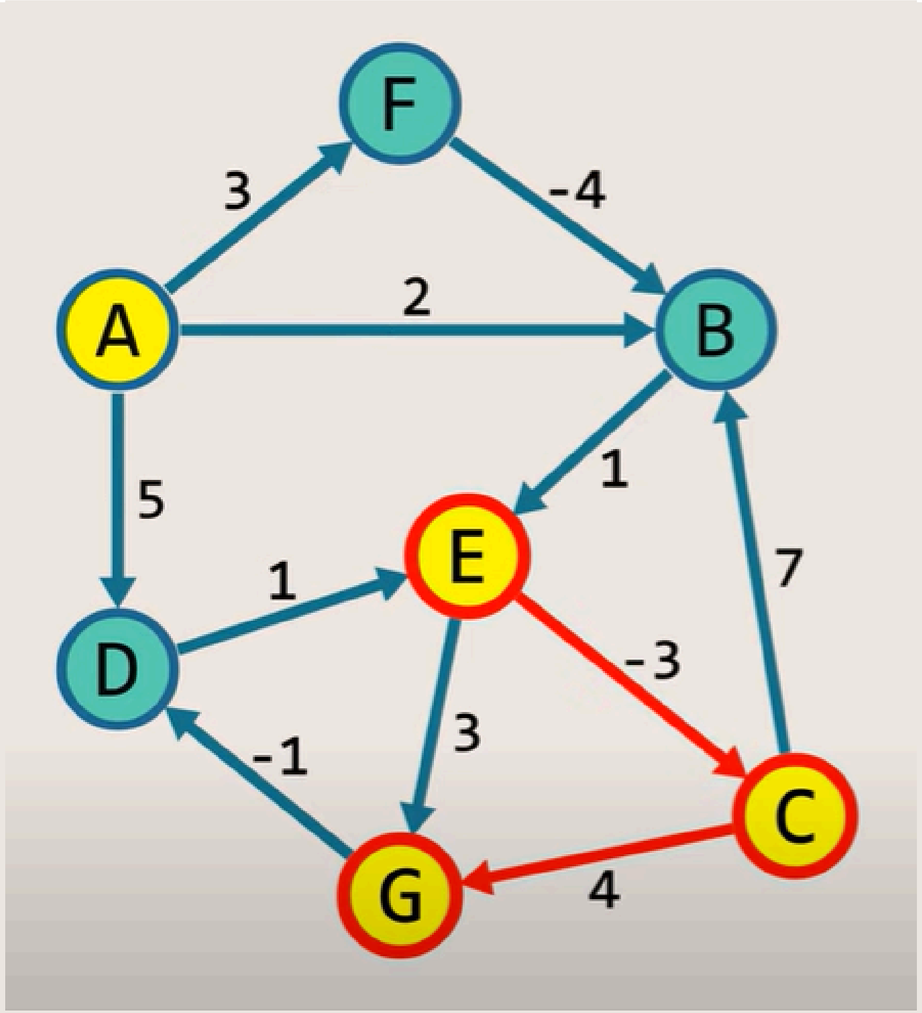
G

To find the shortest path from A to G



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

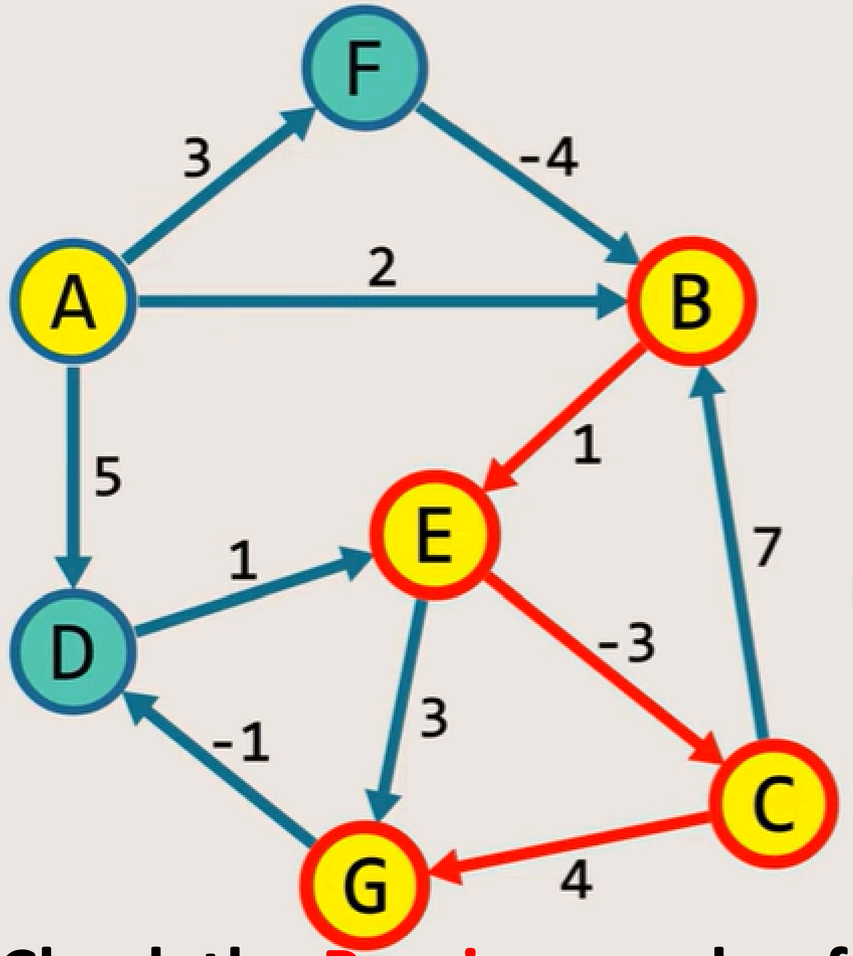
C G



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

Check the Previous node of C

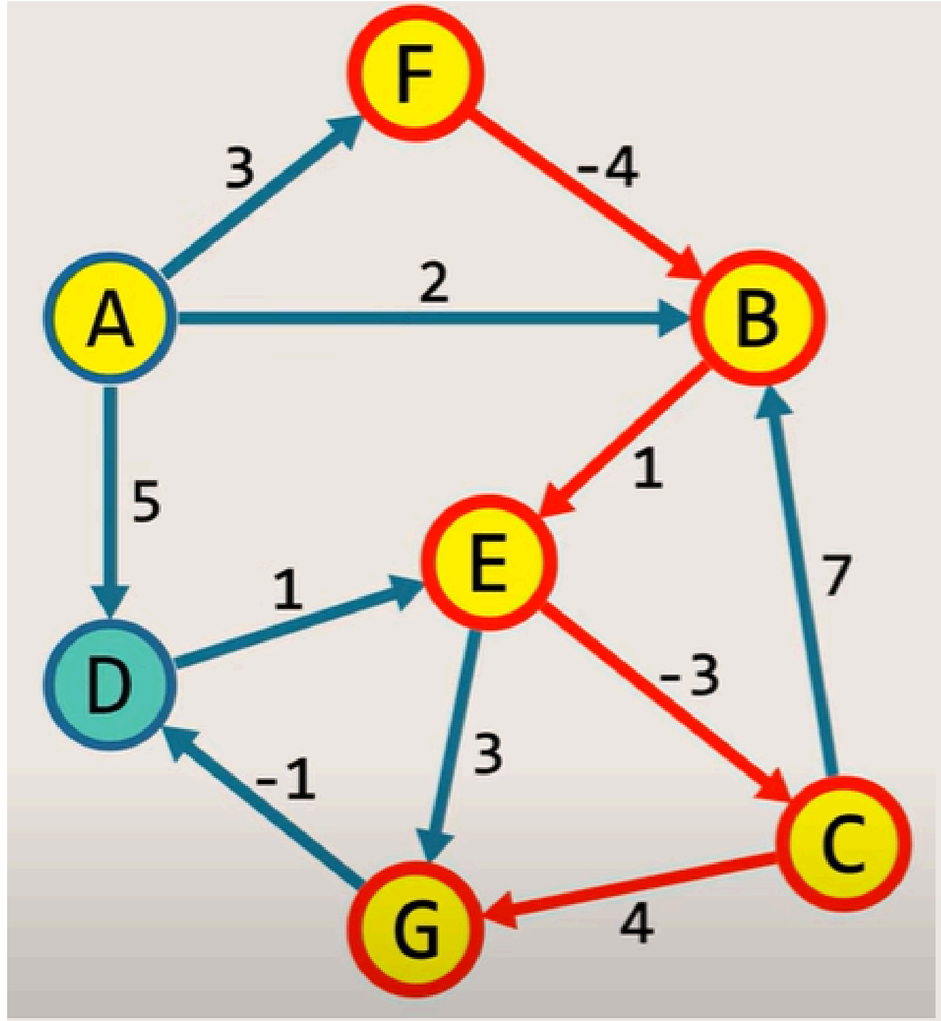
E C G



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

Check the Previous node of E

B E C G

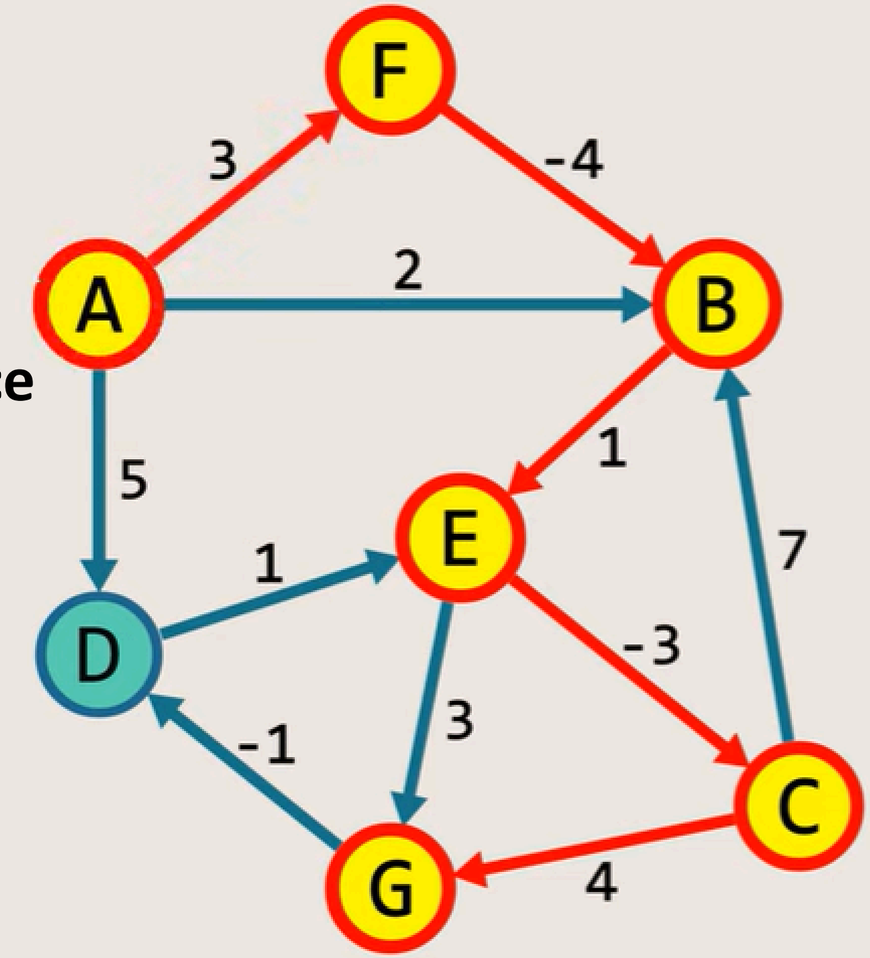


Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

F B E C G

Check the Previous node of B

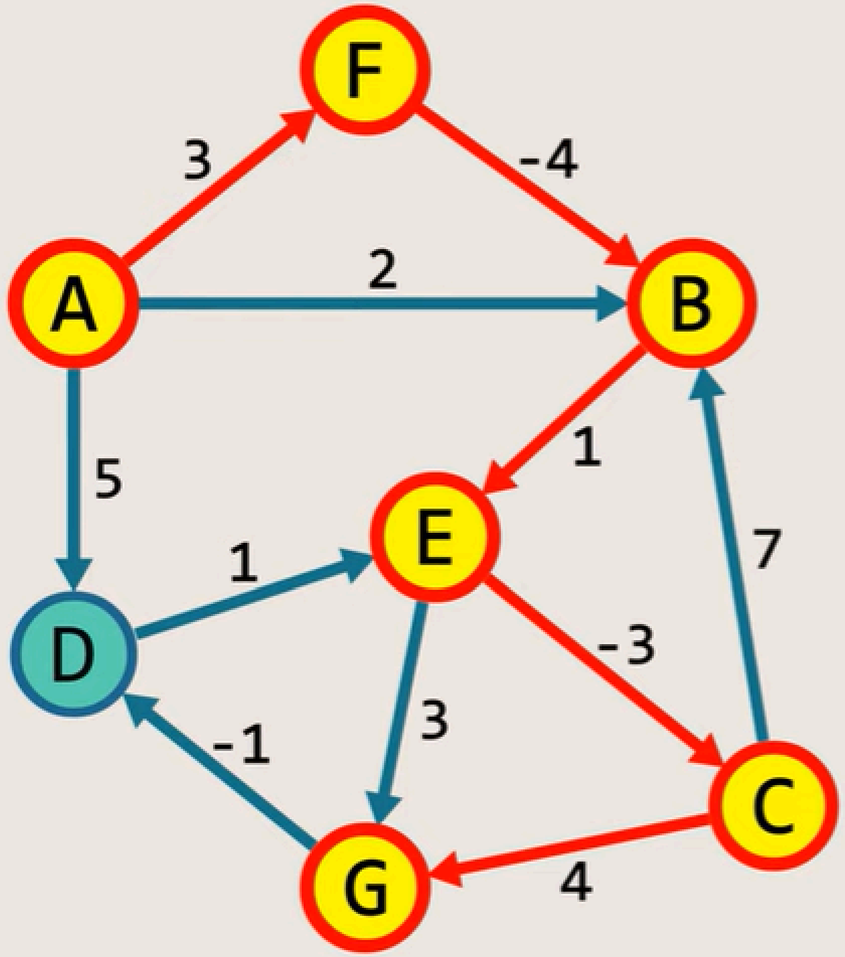
Source



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

A F B E C G

Since we reach the Source node, Stop here



Node	Cost	Previous
A	0	
B	-1	F
C	-3	E
D	0	G
E	0	B
F	3	A
G	1	C

A → F → B → E → C → G

Shortest Path from node A to G

TIME COMPLEXITY

$$O((V-1)\cdot E)$$

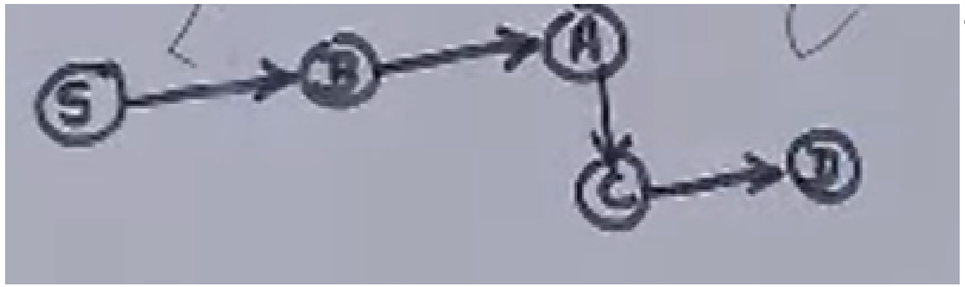
Algorithm goes through all the edges V-1 times to perform the relaxation
and
One more time to check for negative weight cycles

Hence, the time complexity becomes ?

This is another example: Consider **the last iteration** as **final table** and *draw* shortest path tree

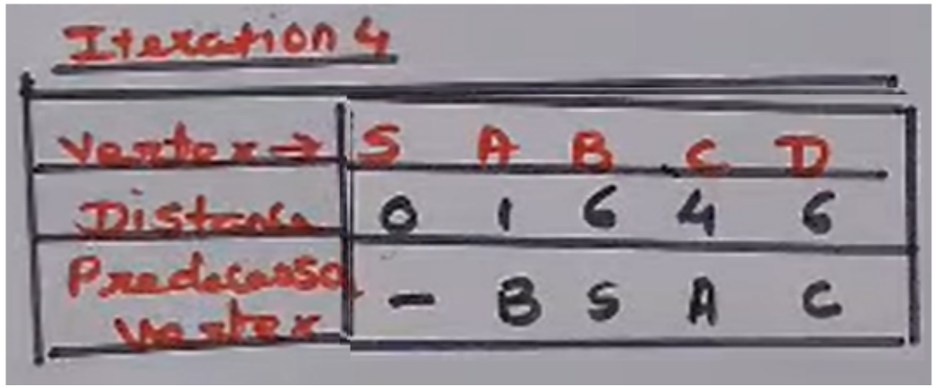
Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

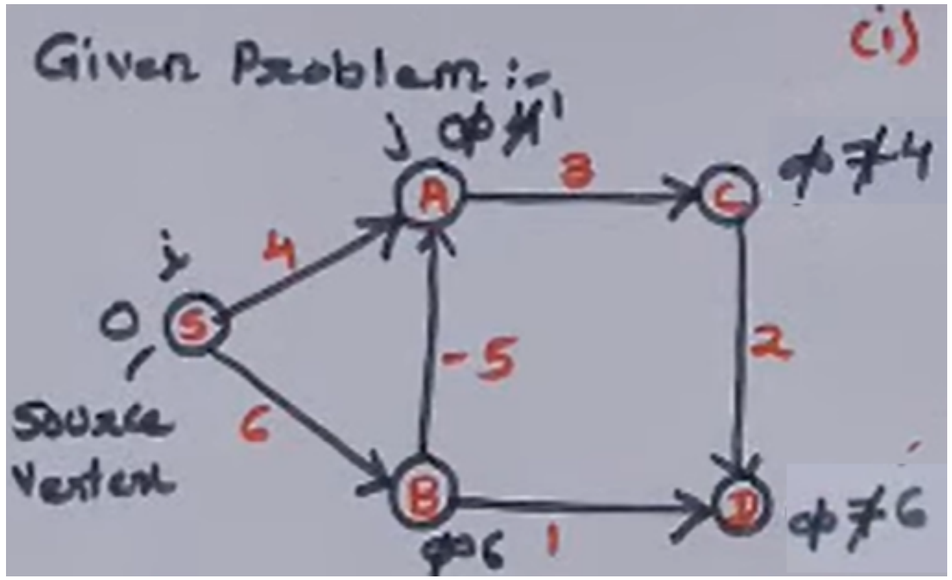


1. Source vertex: S
2. From the table, find which entry has S as the predecessor
   Here S is the Predecessor to B
3. Search Table: B is the predecessor to A
4. Search Table: A is the predecessor to C
5. Search Table: C is the predecessor to D

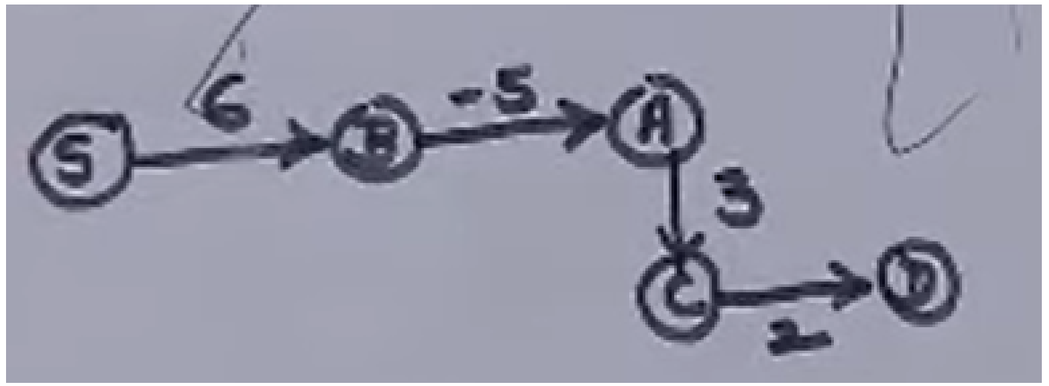
Consider **the last iteration** as **final table** and *draw* shortest path tree



1. Source vertex: S
2. From the table, find which entry has S as the predecessor
Here S is the Predecessor to B
3. Search Table: B is the predecessor to A
4. Search Table: A is the predecessor to C
5. Search Table: C is the predecessor to D



Fill this graph **with weights of original Graph.**



Fill the **shortest Path table** with **Final iteration**

Iteration 4

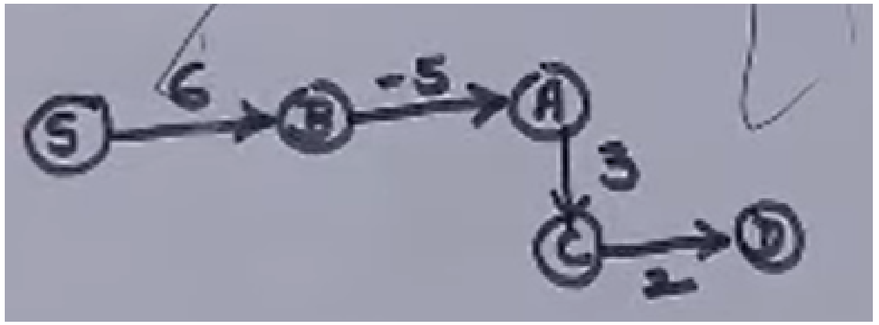
Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

Path	Shortest Distance	Shortest Path
S-A		
S-B		
S-C		
S-D		

Fill the **shortest Path table** with **Final iteration**

Iteration 4

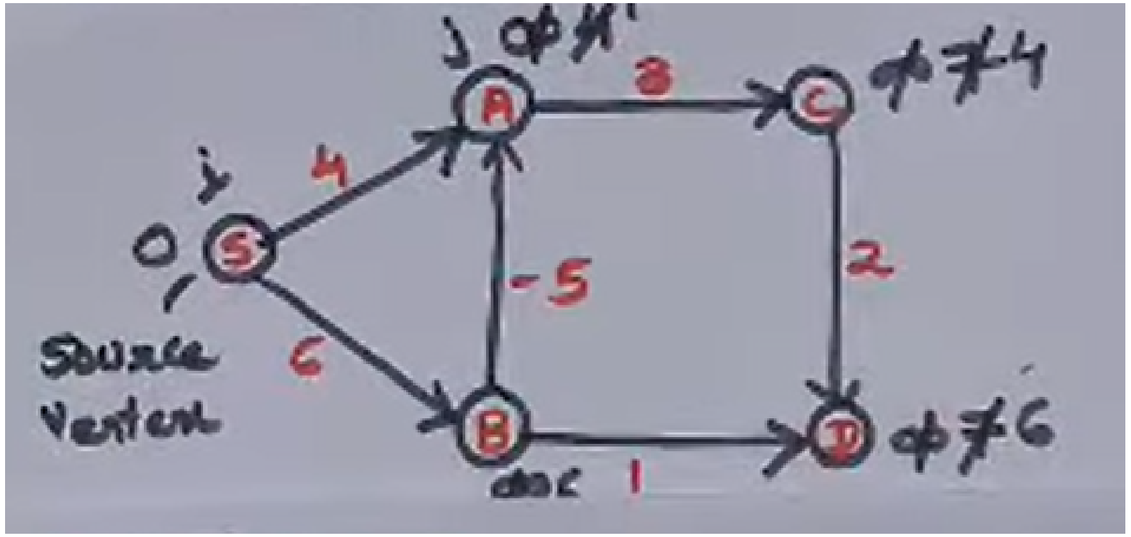
Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	-	B	S	A	C

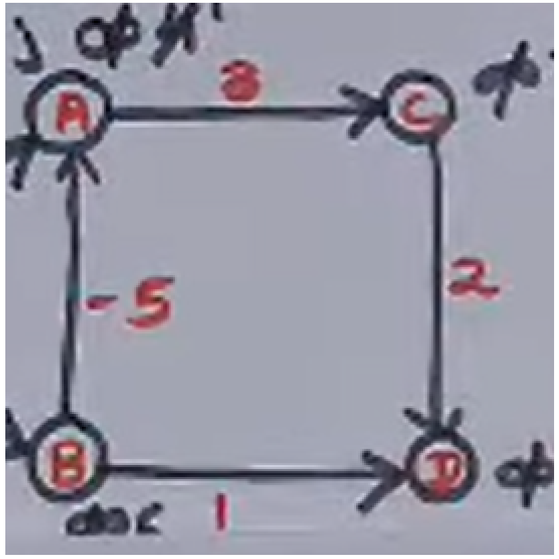


Path	Shortest Distance	Shortest Path
S-A	1	S-B-A (6-5)=1
S-B	6	S-B (6)
S-C	4	S-B-A-C (6-5+3) = 4
S-D	6	S-B-A-C-D

(6-5+3+3)
≅ 6

Next Step: Ensure that **no Negative cycles** in the graph :– **Approach 1**





Consider one of the loops existing:-
Add all the weights = -5 + 3 + 2 + 1 = **1**

They should be positive. (i.e. Non-negative)

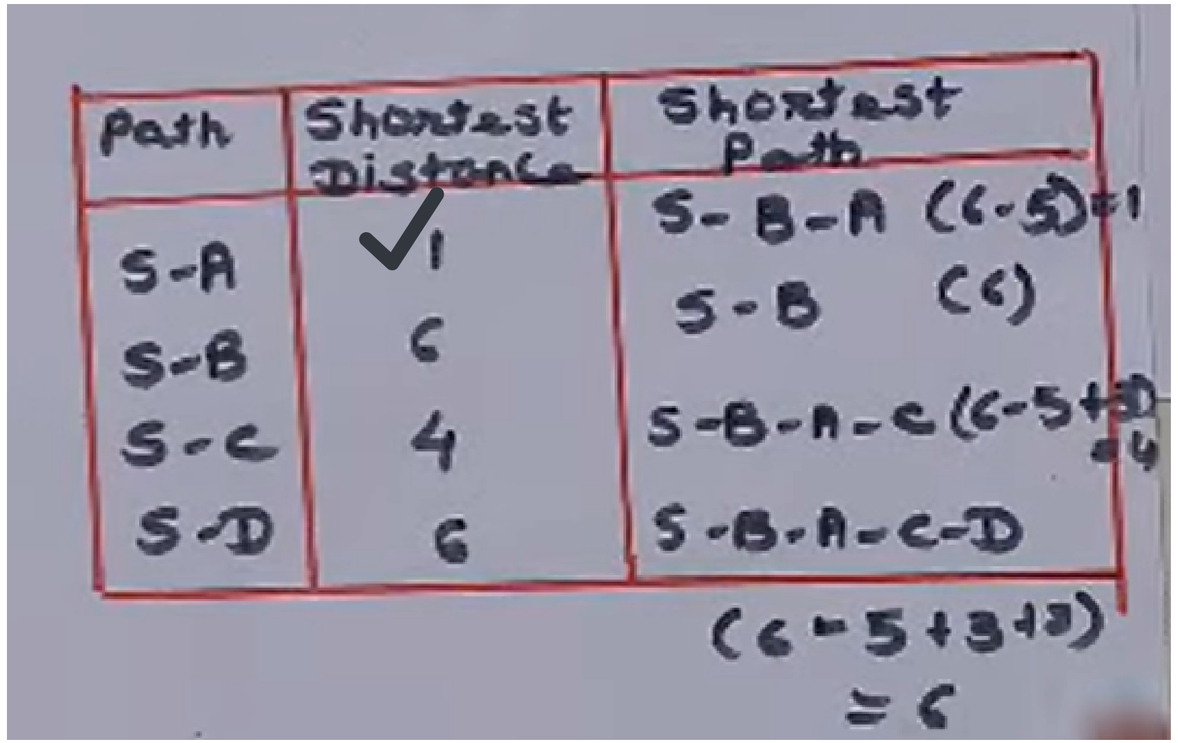
Therefore, there is no negative cycle exists in the graph.

Next Step: Ensure that **no Negative cycles** in the graph :– **Approach 2**

Note: **If there is a negative edge cycle, then our problem solution is not correct.**

1. **Consider edge A->C**
2. **dc <= dA + CAC**
3. **4 <= 1 + 3**
4. **If this condition is satisfied for every edge, then, there is no negative weight cycle that is existing.**

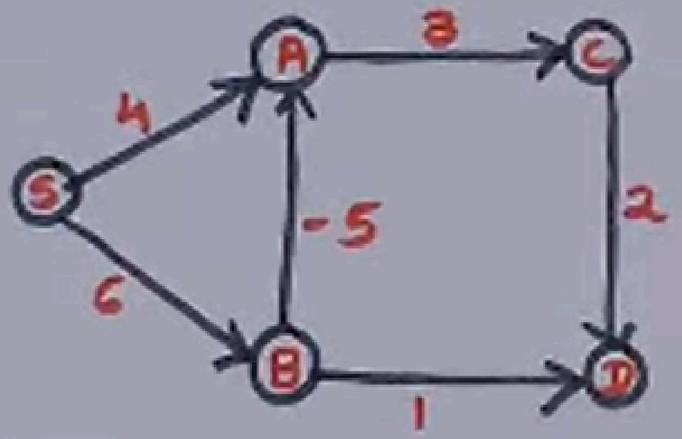
Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6



Bellman Ford Algorithm

(i) List all the Edges

Given Problem :-

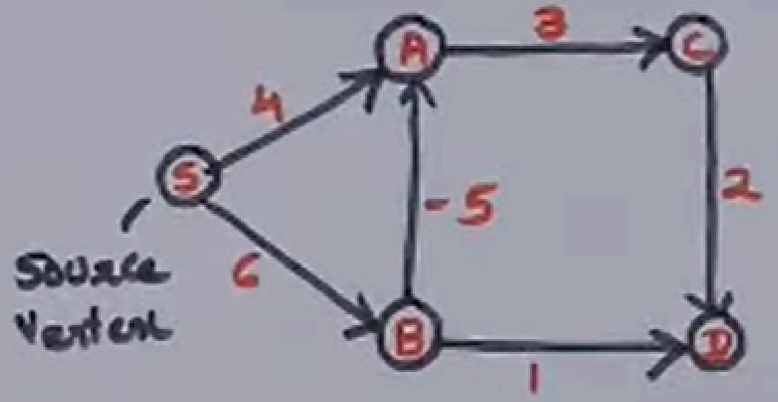


Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

No of Iterations
= n - 1
n → No of Vertices

No of Iterations = 5-1 = 4

Given Problem :-



(i) List all the Edges

Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

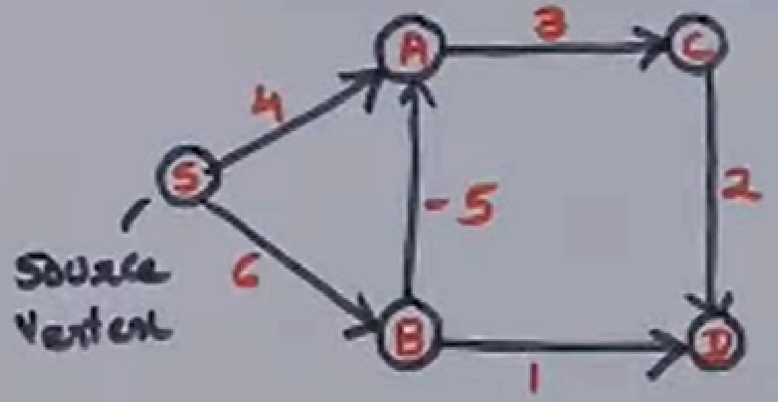
No of Iterations
= n - 1
n → No of Vertices

5 - 1 = 4 Iterations

Initial

VERTEX	S	A	B	C	D
Distance					
Predecessor Vertex					

Given Problem :-



(i) List all the Edges

Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

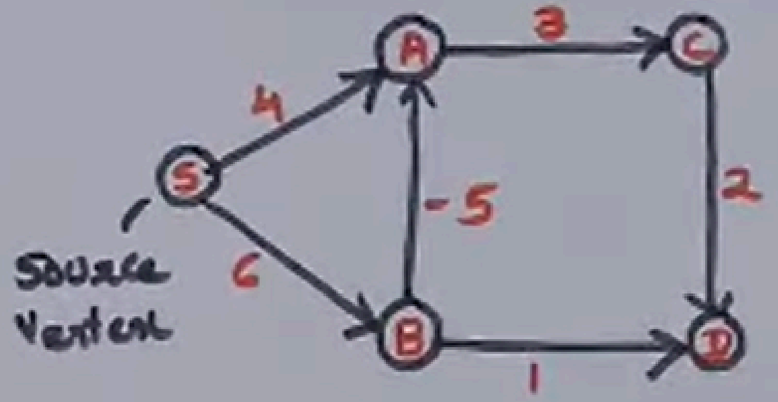
No of Iterations
= n - 1
n → No of Vertices

5 - 1 = 4 Iterations

Initial

VERTEX	S	A	B	C	D
Distance	0	∞	∞	∞	∞
Predecessor Vertex	-	-	-	-	-

Given Problem :-



(i) List all the Edges

Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

No of Iterations
= n - 1
n → No of vertices

5 - 1 = 4 Iterations

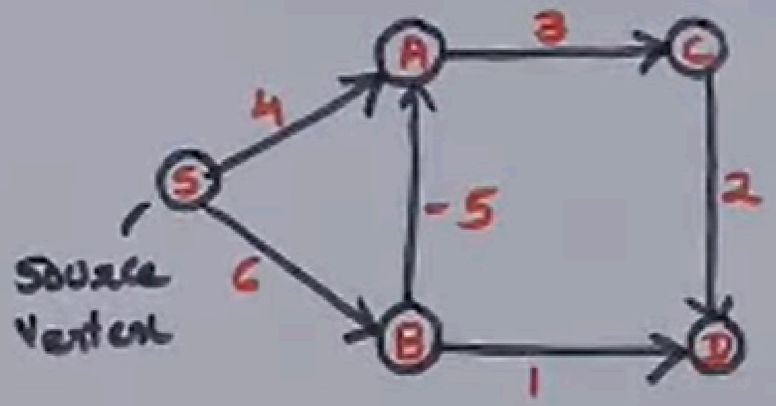
Initial

VERTEX→	S	A	B	C	D
Distance	0	∞	∞	∞	∞
Predecessor Vertex	-	-	-	-	-

Iteration 1

VERTEX→	S	A	B	C	D
Distance					
Predecessor vertex					

Given Problem :-



(i) List all the Edges

Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Formula to calculate the shortest path between any 2 vertices

$d(i) + c(i,j) < d(j)$
then
$d(j) = d(i) + c(i,j)$

No of Iterations
$= n - 1$
$n \rightarrow$ No of vertices

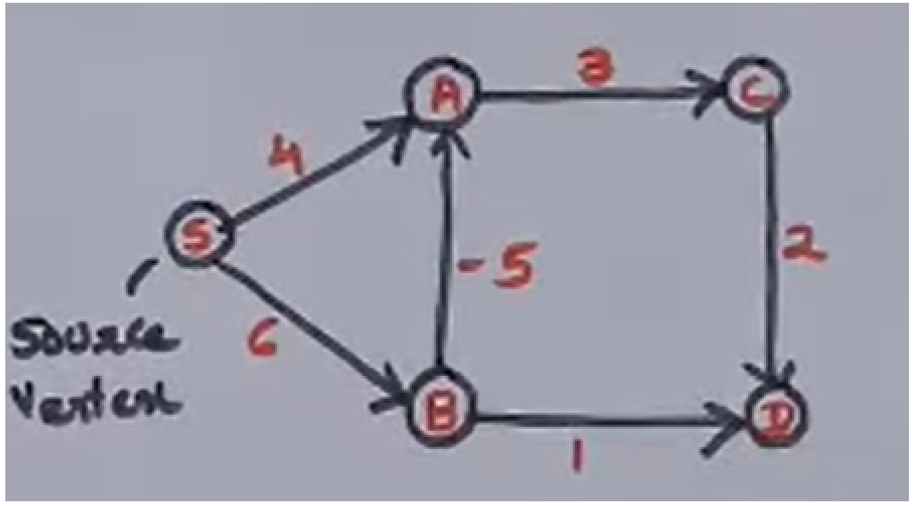
$5 - 1 = 4$ Iterations

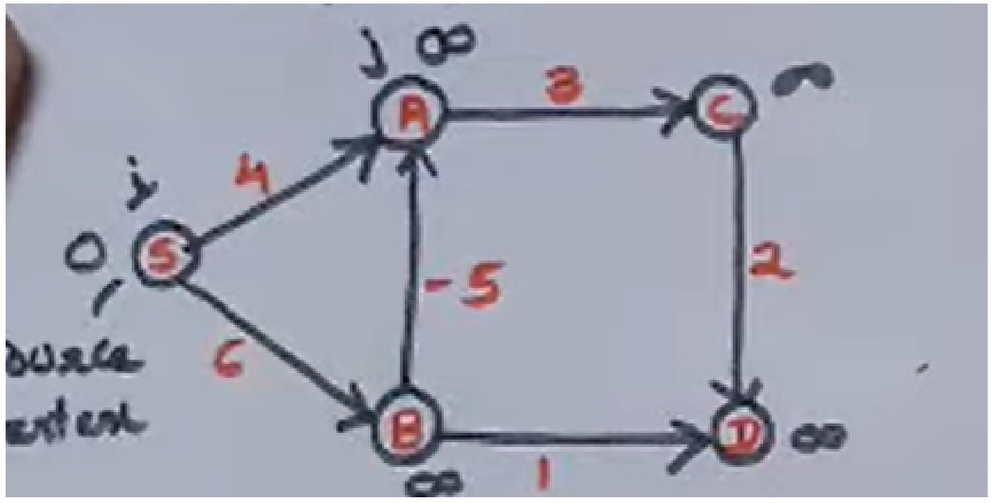
Initial

VERTEX→	S	A	B	C	D
Distance	0	∞	∞	∞	∞
Predecessor Vertex	-	-	-	-	-

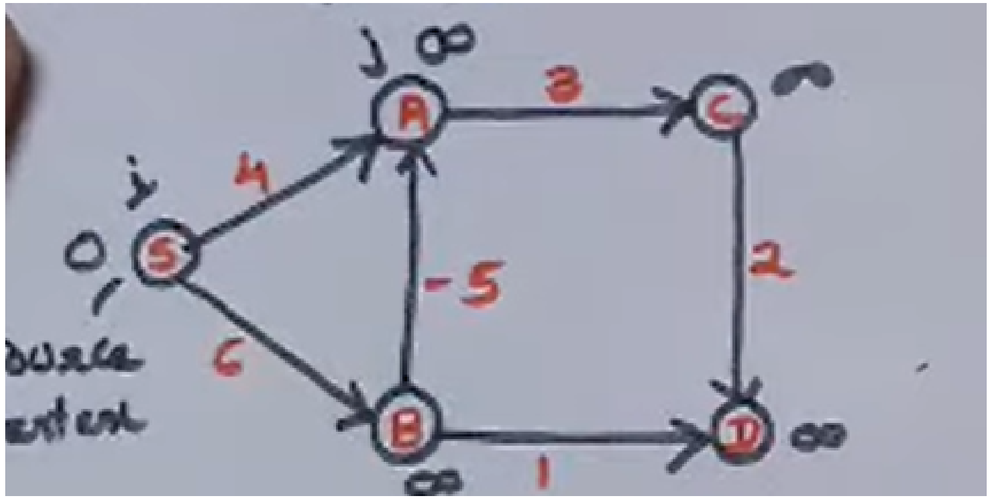
Iteration 1

VERTEX→	S	A	B	C	D
Distance					
Predecessor vertex					





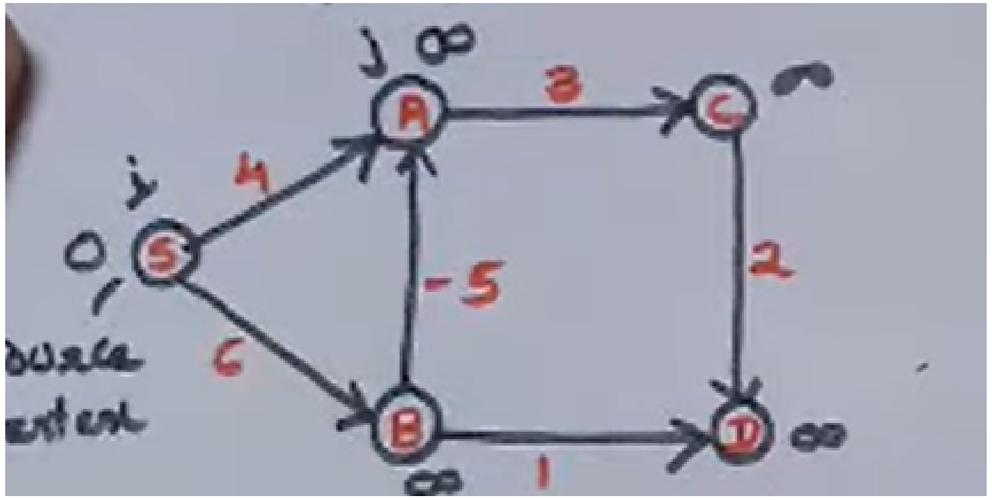
Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

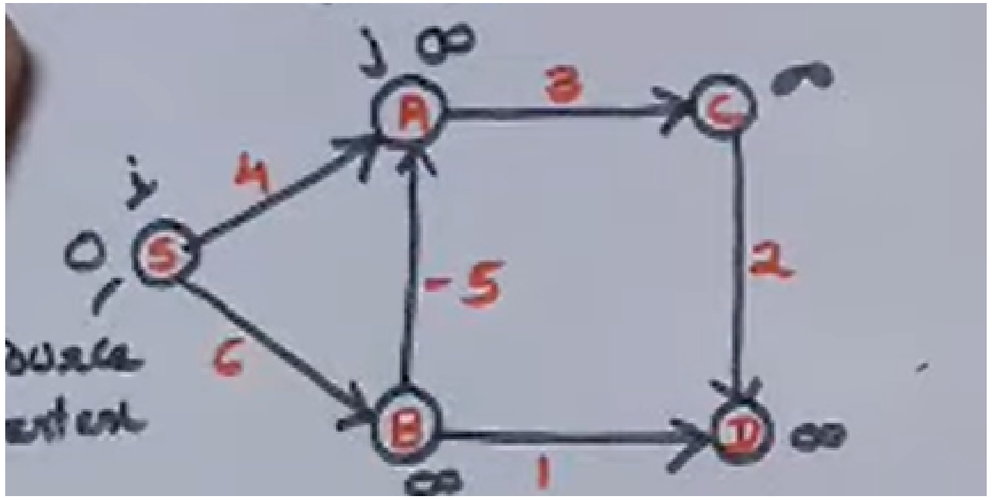
VERTEX →	S	A	B	C	D
Distance				∞	
Predecessor vertex				-	



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

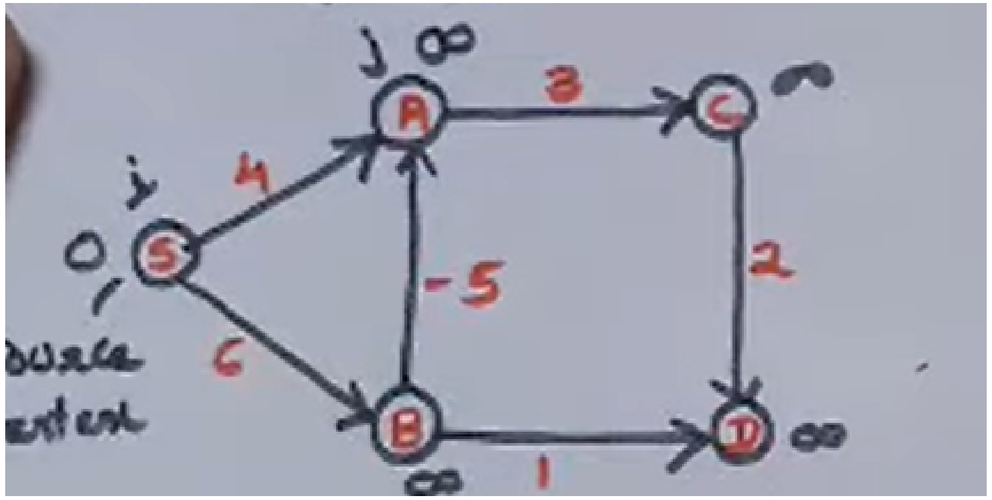
VERTEX →	S	A	B	C	D
Distance		∞		∞	
Predecessor vertex		-		-	



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance		∞		∞	∞
Predecessor vertex		-		-	-

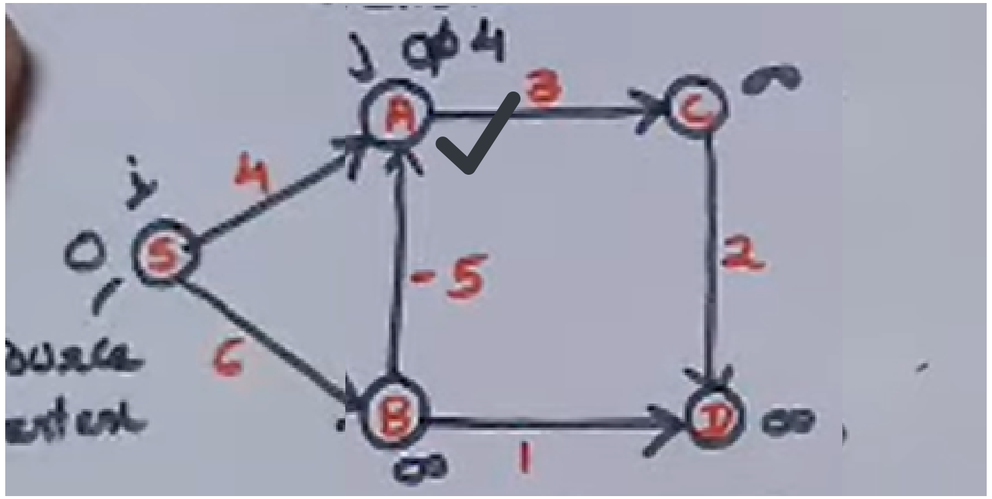


Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance		∞		∞	∞
Predecessor vertex		—		—	—

Its ∞. Already written.

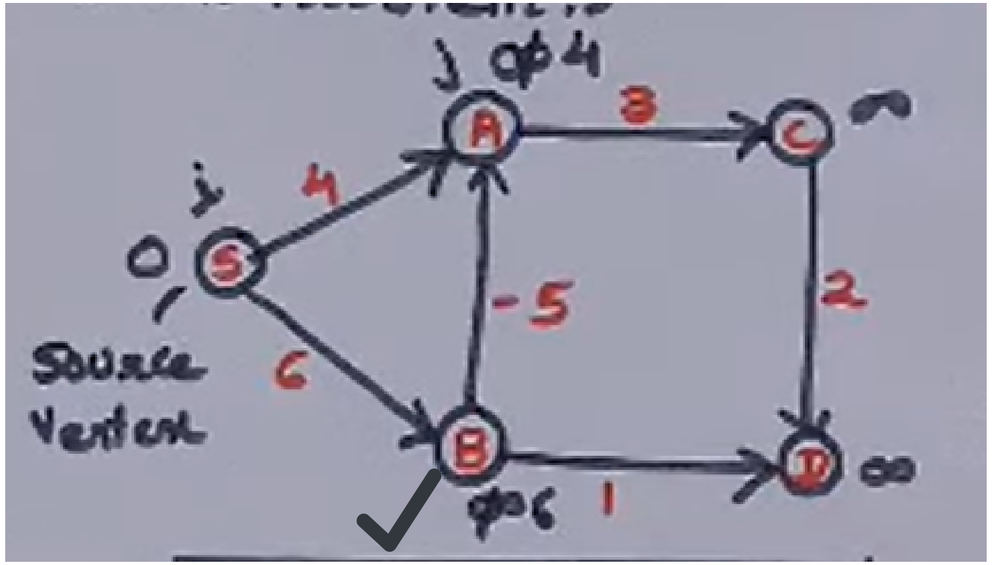


Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance		∞		∞	∞
Predecessor vertex		—		—	—

Its ∞. Already written.

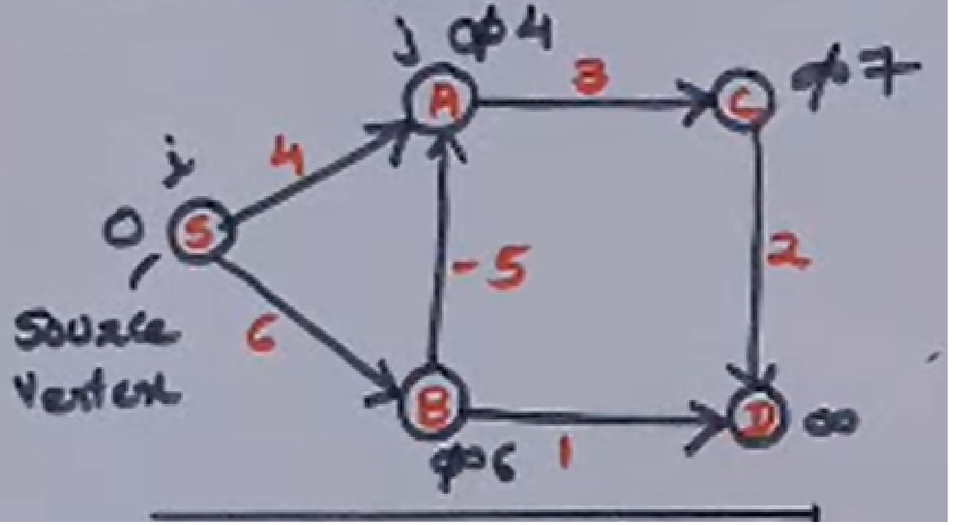


Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
✓ S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	~~4~~	6	∞	∞
Predecessor vertex	-	S	S	-	-

Given Problem :-



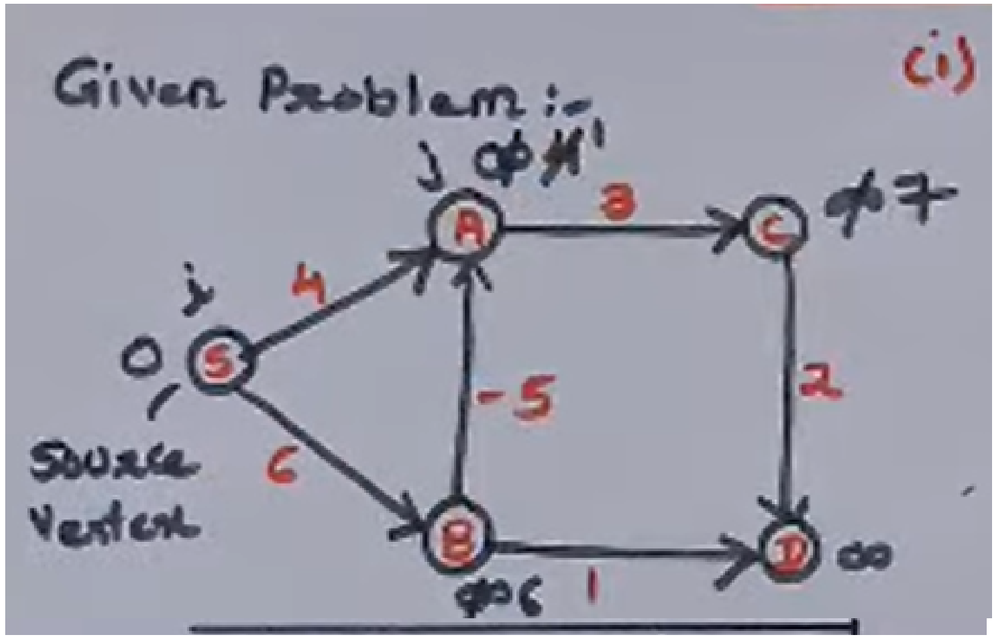
Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0			7	
Predecessor vertex	-			A	



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

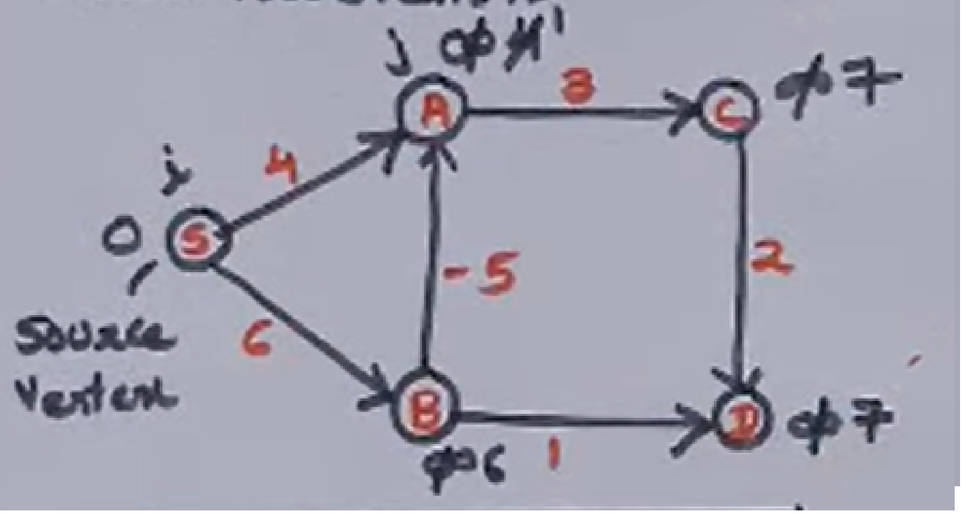
VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1		7	
Predecessor vertex	-	B		A	

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

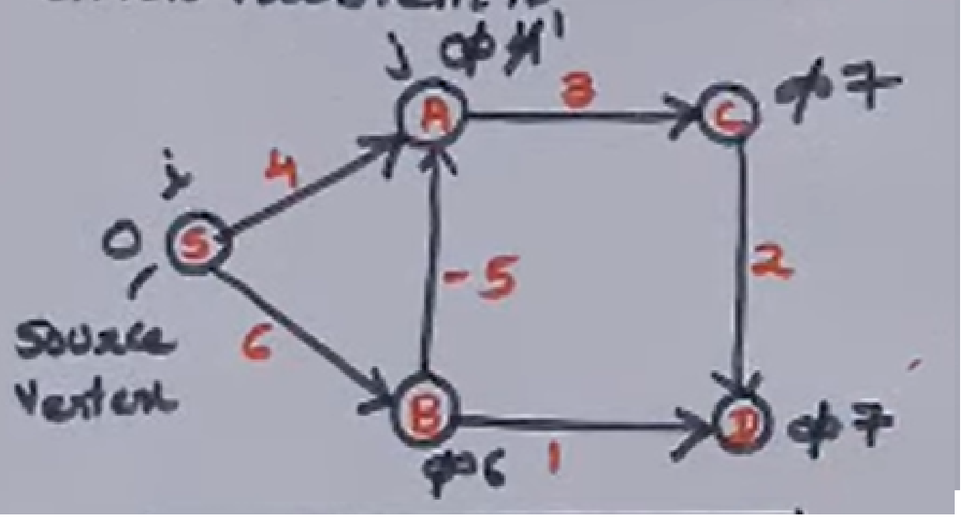
VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1		7	7
Predecessor vertex	-	B		A	B

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

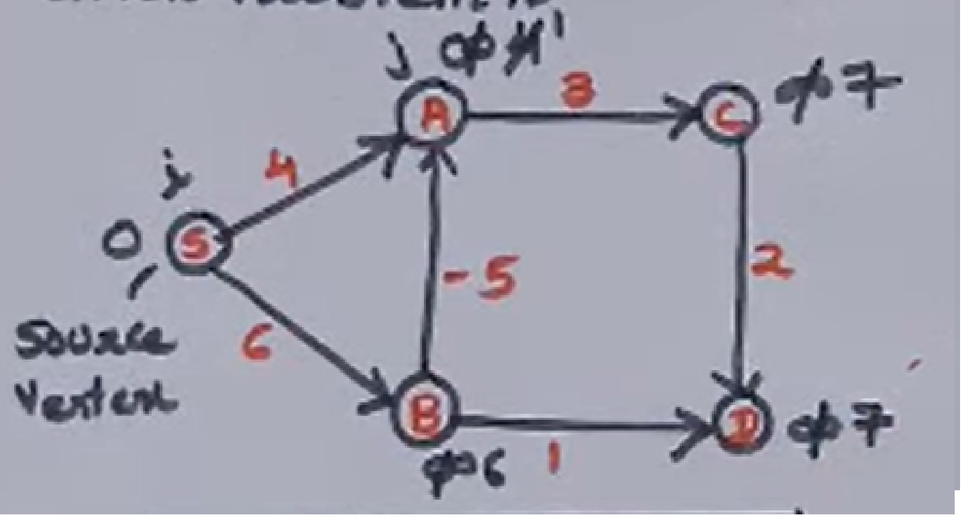
Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1		7	7
Predecessor vertex	-	B		A	B

No change, already
7 < 9 (7+2)

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

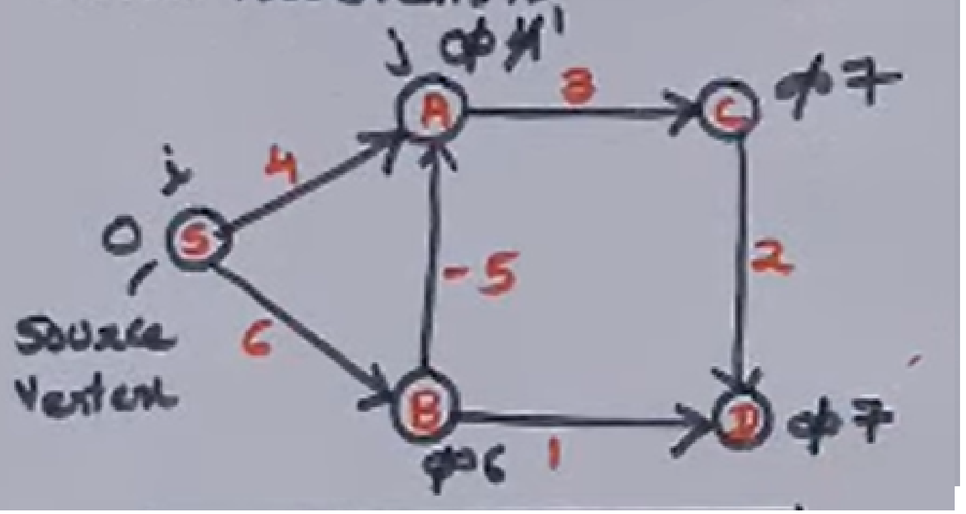
Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1		7	7
Predecessor vertex	-	B		A	B

No change, already
1 < 4 (0+4)

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
✓ S-B	6

Iteration 1

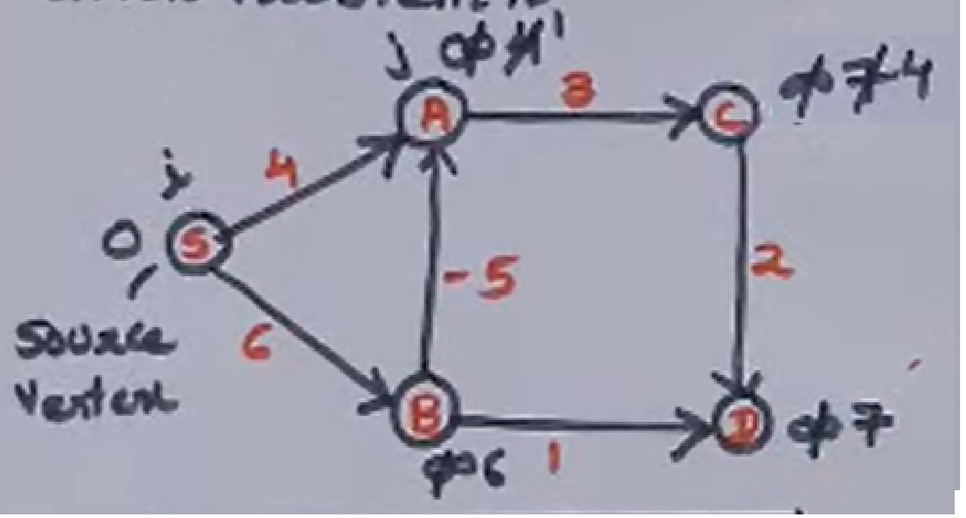
VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	∞ 4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

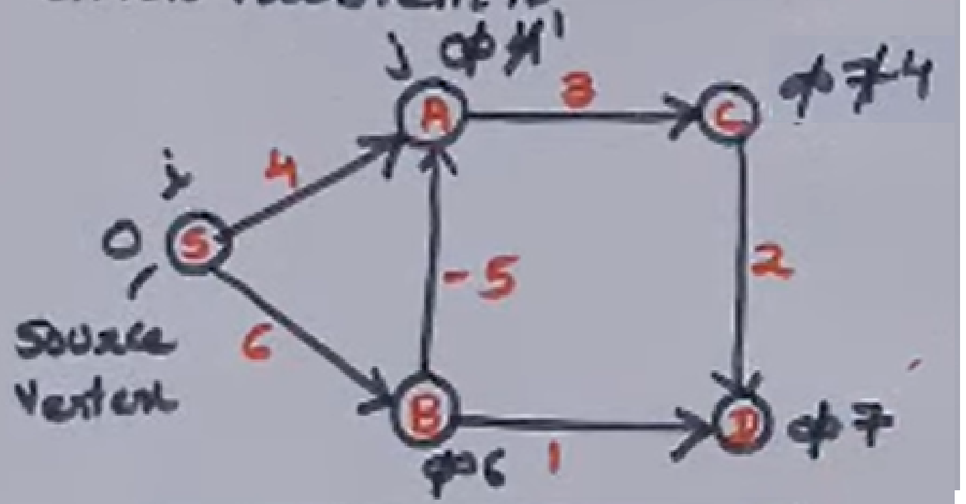
Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0			4	
Predecessor vertex	-			A	

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

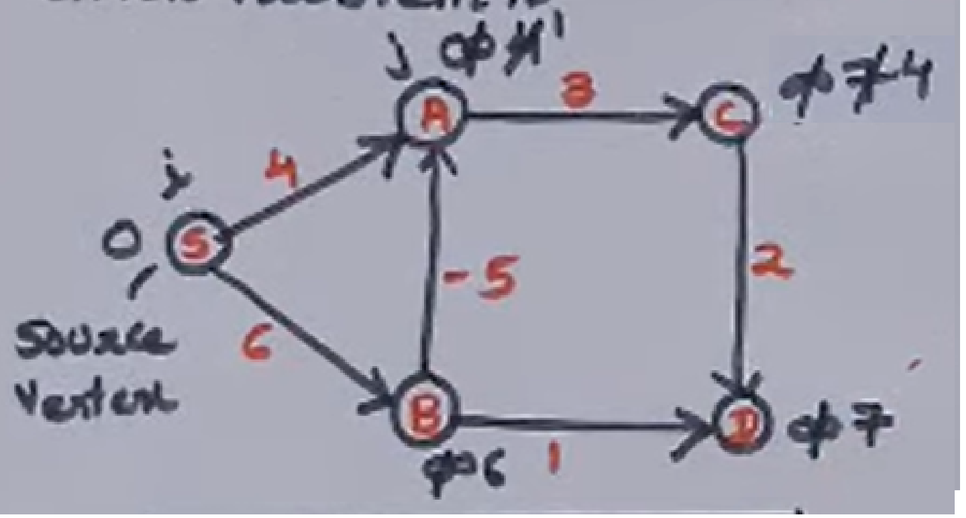
Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1		4	
Predecessor Vertex	-	B		A	

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

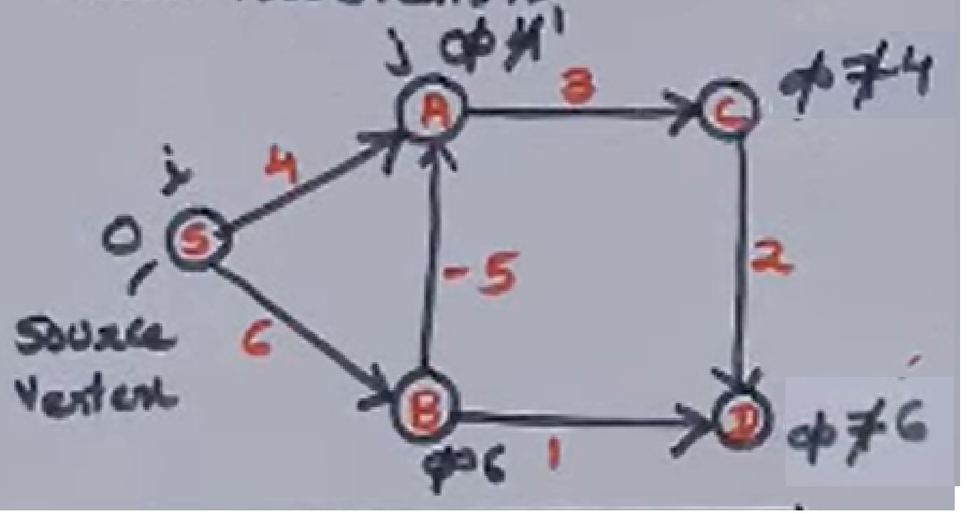
Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1		4	7
Predecessor Vertex	-	B		A	B

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

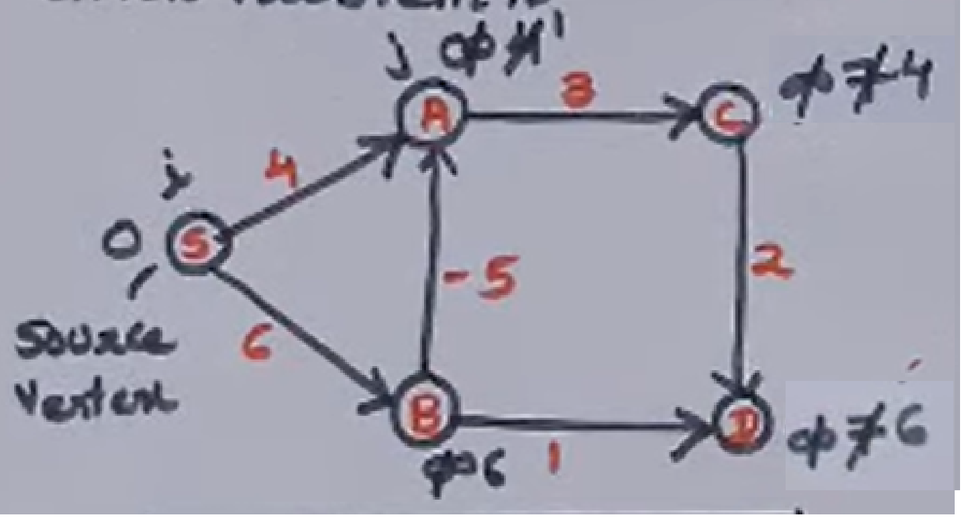
Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1		4	7
Predecessor Vertex	-	B		A	B

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

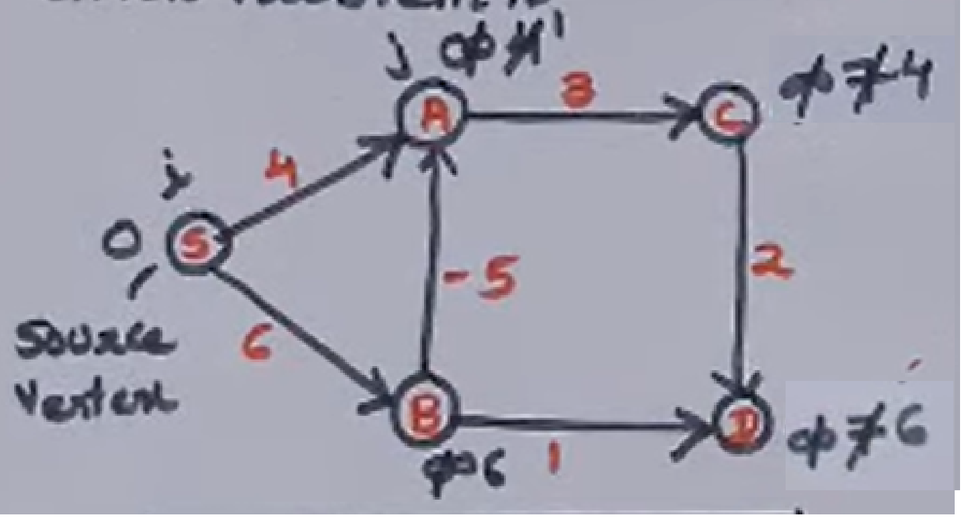
Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1		4	7
Predecessor Vertex	-	B		A	B

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

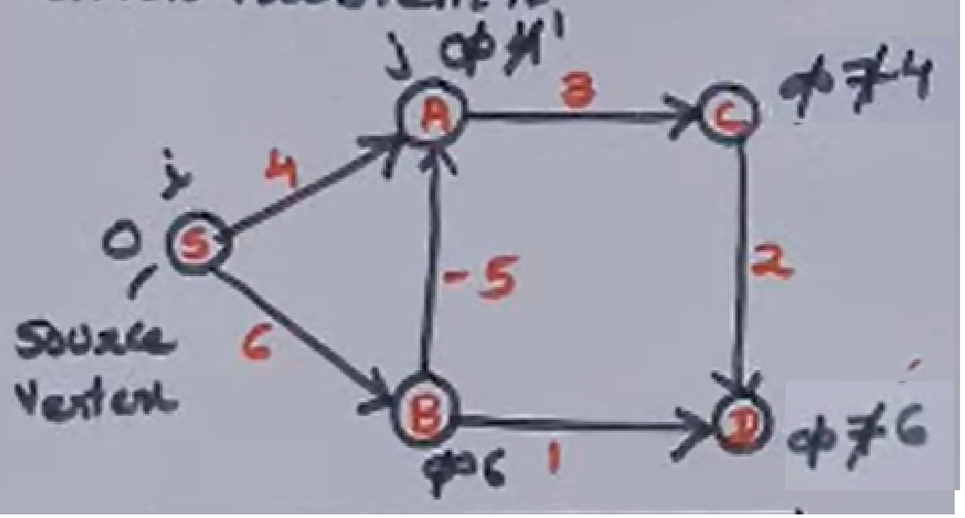
Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	-	B	S	A	B

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

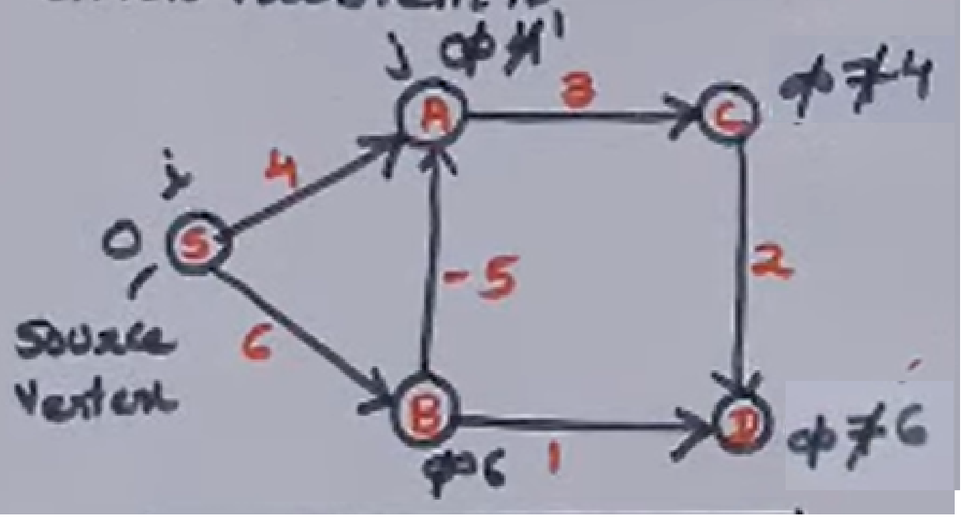
Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

Iteration 4

Vertex →	S	A	B	C	D
Distance	0				
Predecessor vertex	-				

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

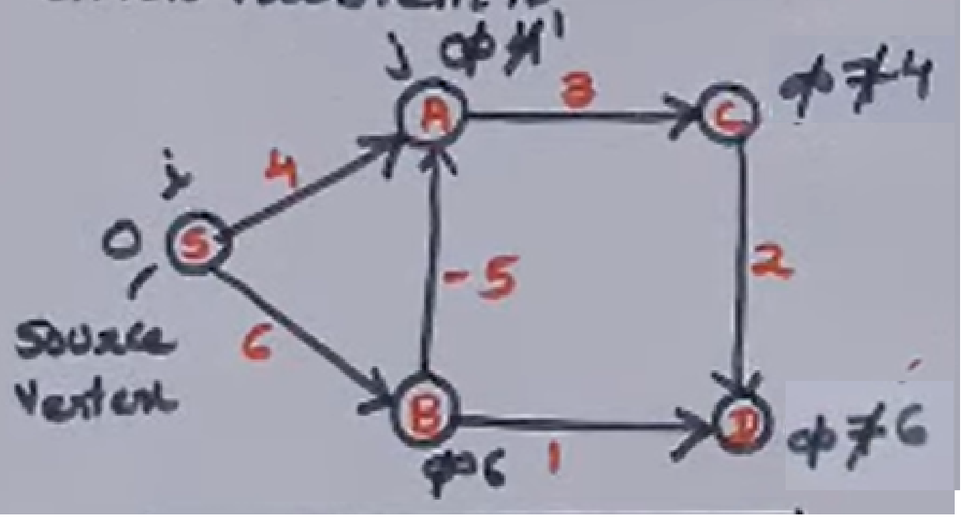
Iteration 4

Vertex →	S	A	B	C	D
Distance	0			4	
Predecessor vertex	-			A	

No Change

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

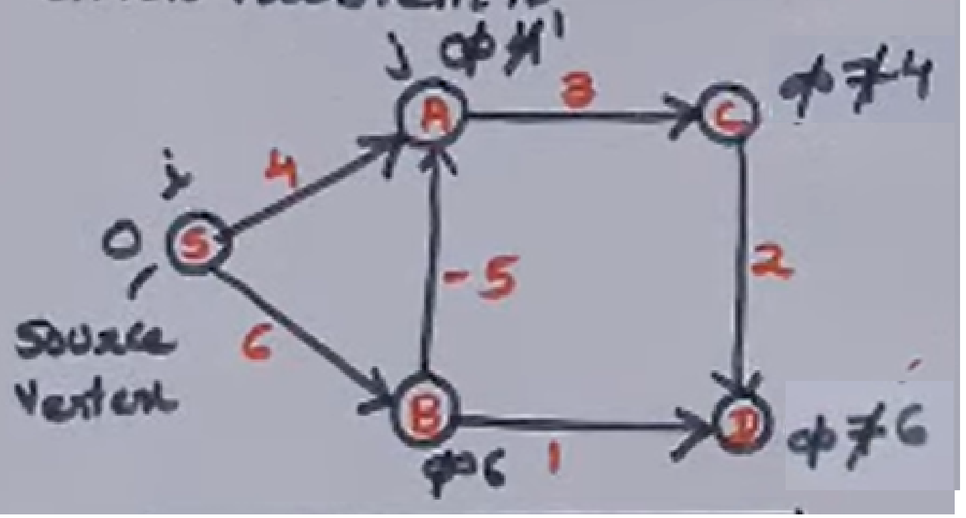
Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1		4	
Predecessor vertex	-	B		A	

No Change

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

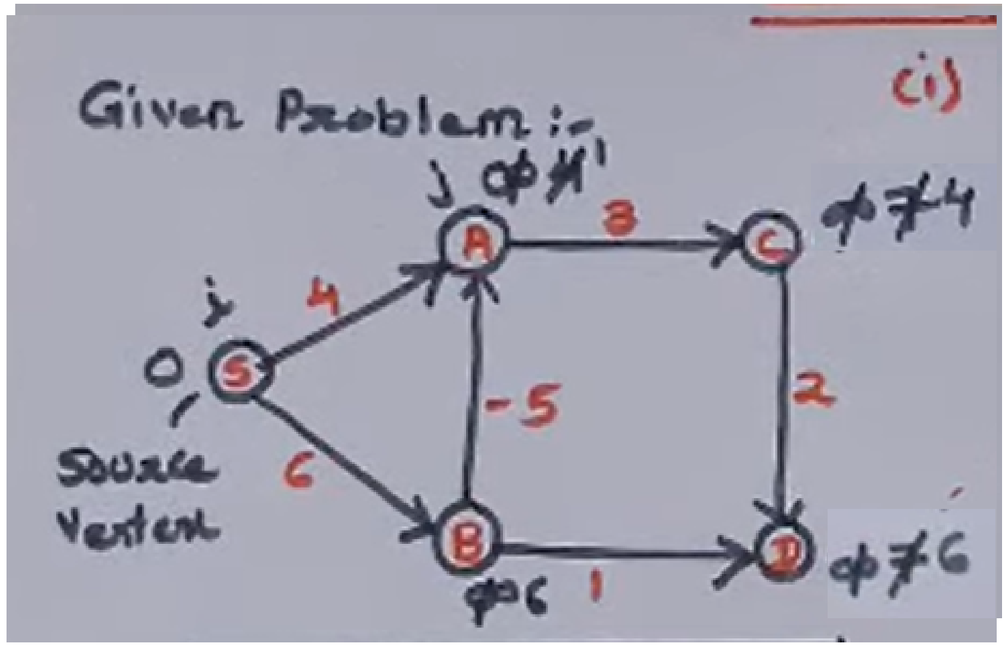
Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1		4	6
Predecessor vertex	-	B		A	C

No Change



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

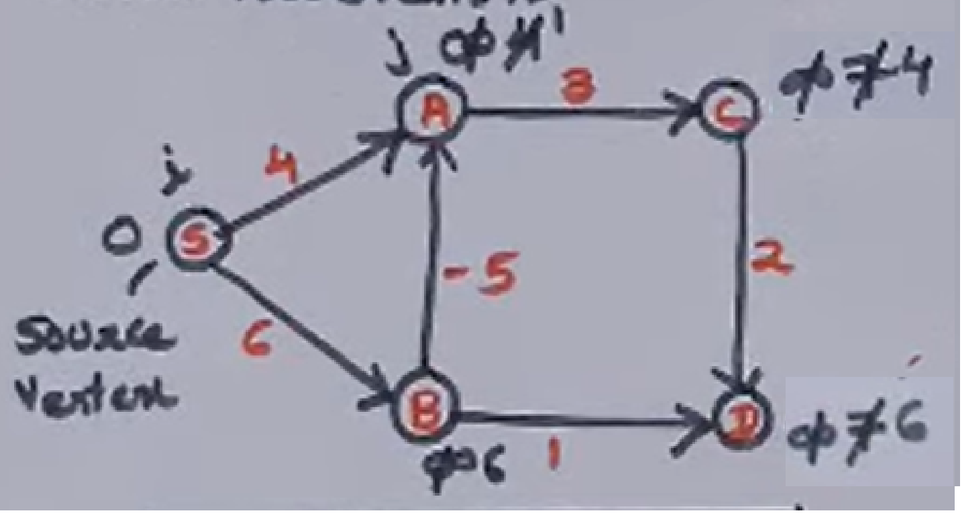
Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1		4	6
Predecessor vertex	-	B		A	C

Already considered this.

(i)

Given Problem :-



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

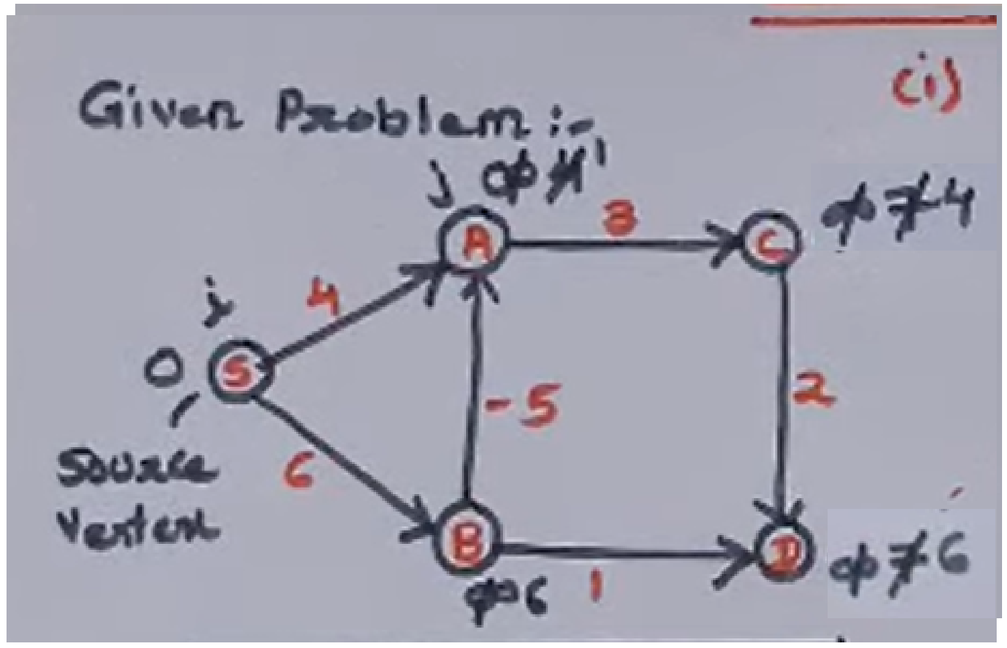
Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1		4	6
Predecessor vertex	-	B		A	C

No change



Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
✓ S-B	6

Iteration 1

VERTEX →	S	A	B	C	D
Distance	0	∞ 4	6	∞	∞
Predecessor vertex	-	S	S	-	-

Iteration 2

Vertex →	S	A	B	C	D
Distance	0	1	6	7	7
Predecessor vertex	-	B	S	A	B

Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor vertex	-	B	S	A	C

No change

Iteration 3

Vertex →	S	A	B	C	D
Distance	0	1	6	4	~~7~~ 6
Predecessor vertex	-	B	S	A	~~B~~ C

Iteration 3 and Iteration 4 remains same

Consider **Iteration 4** as **final table** and *draw* shortest path tree

Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

1. Source vertex: S

Consider **Iteration 4** as **final table** and *draw* shortest path tree

Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

1. Source vertex: S
2. From the table, find which entry has S as the predecessor

Here S is the Predecessor to B

Consider **Iteration 4** as **final table** and *draw* shortest path tree

Iteration 4

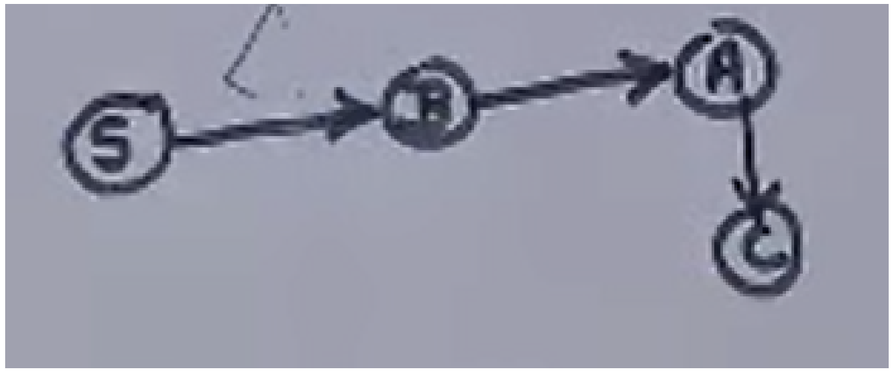
Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

1. Source vertex: S
2. From the table, find which entry has S as the predecessor
Here S is the Predecessor to B
3. Search Table: B is the predecessor to A

Consider **Iteration 4** as **final table** and *draw* shortest path tree

Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

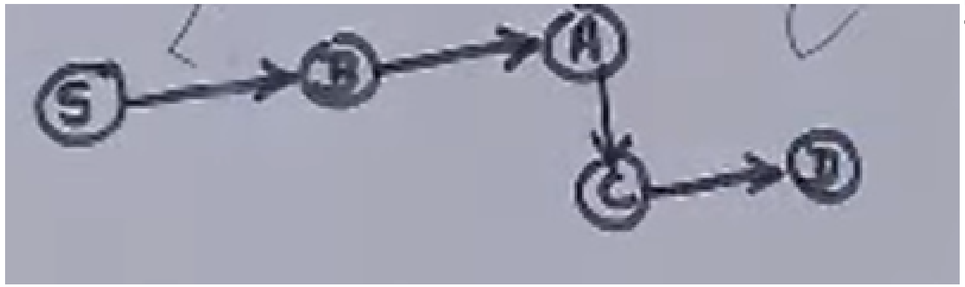


1. Source vertex: S
2. From the table, find which entry has S as the predecessor
Here S is the Predecessor to B
3. Search Table: B is the predecessor to A
4. Search Table: A is the predecessor to C

Consider **Iteration 4** as **final table** and *draw* shortest path tree

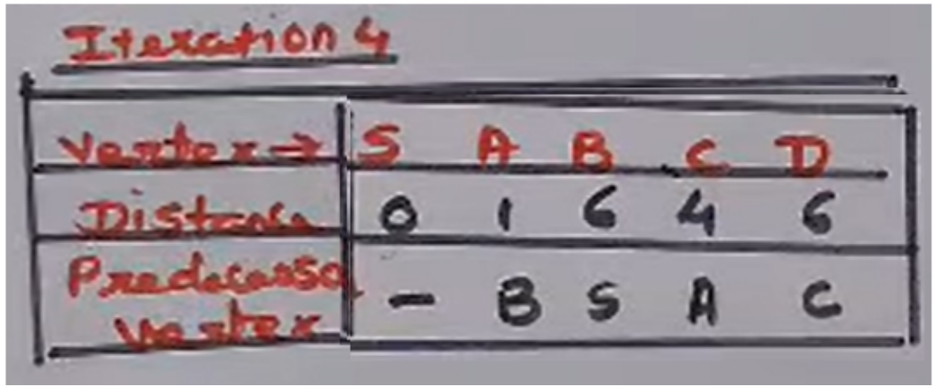
Iteration 4

Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

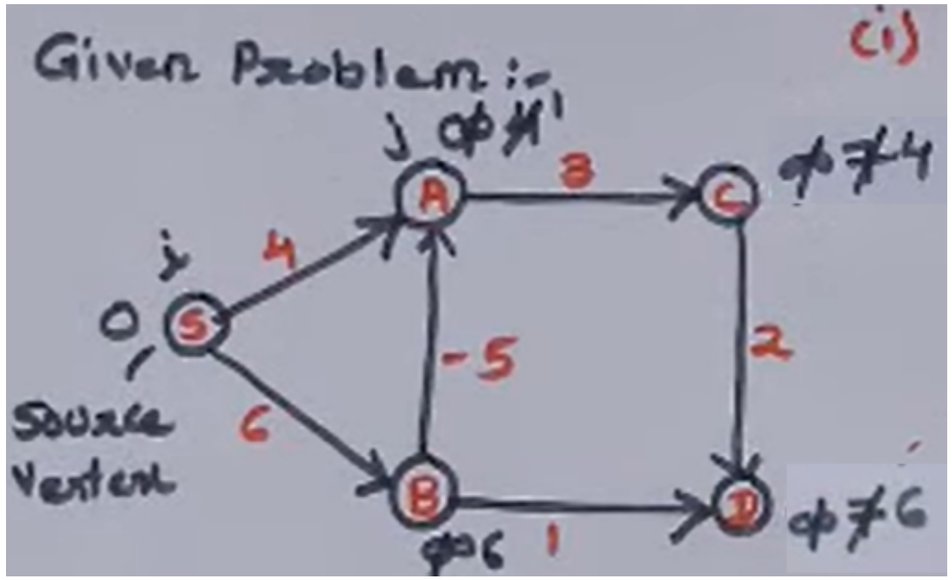


1. Source vertex: S
2. From the table, find which entry has S as the predecessor
Here S is the Predecessor to B
3. Search Table: B is the predecessor to A
4. Search Table: A is the predecessor to C
5. Search Table: C is the predecessor to D

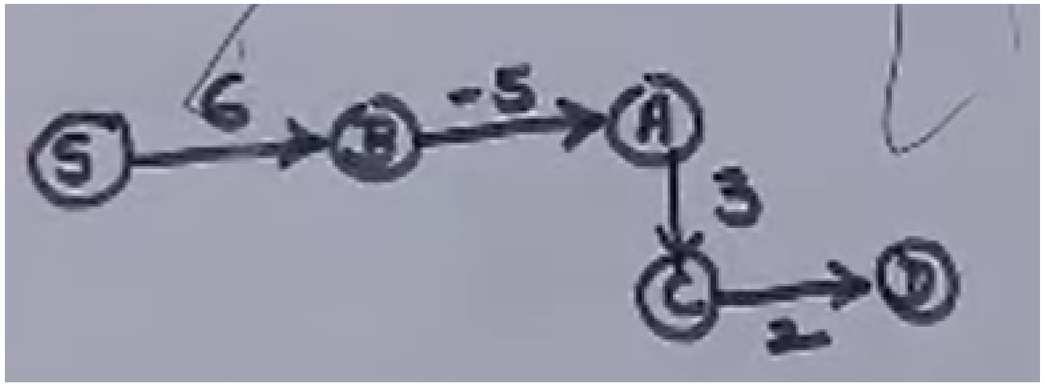
Consider **Iteration 4** as **final table** and *draw* shortest path tree



1. Source vertex: S
2. From the table, find which entry has S as the predecessor
Here S is the Predecessor to B
3. Search Table: B is the predecessor to A
4. Search Table: A is the predecessor to C
5. Search Table: C is the predecessor to D



Fill this graph **with weights of original Graph.**



Fill the **shortest Path table** with **Iteration 4**

Iteration 4

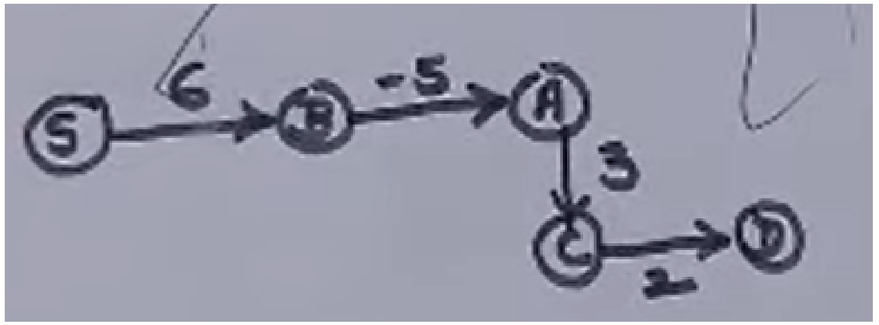
Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	–	B	S	A	C

Path	Shortest Distance	Shortest Path
S-A		
S-B		
S-C		
S-D		

Fill the **shortest Path table** with **Iteration 4**

Iteration 4

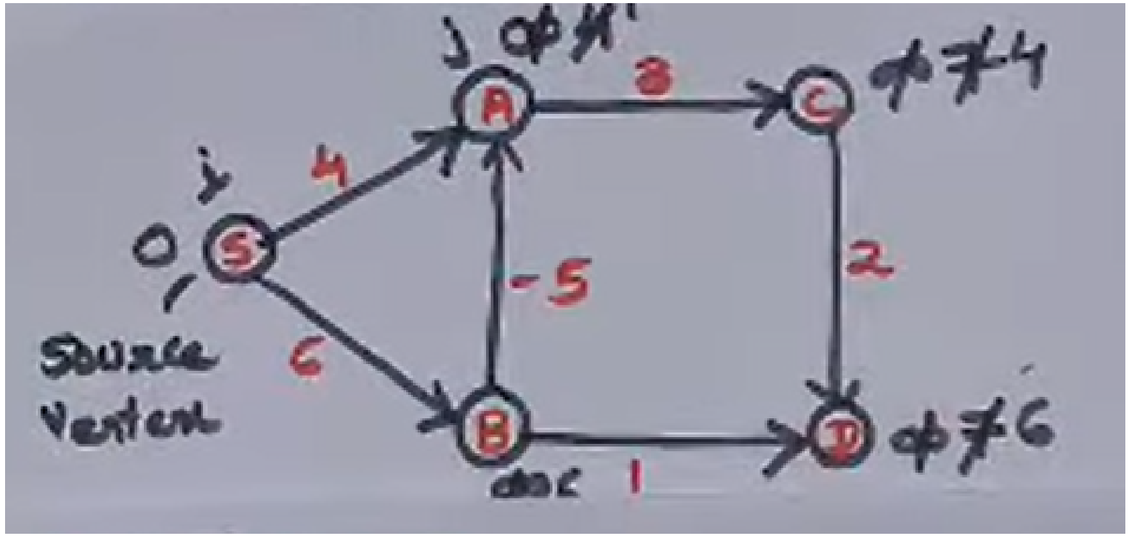
Vertex →	S	A	B	C	D
Distance	0	1	6	4	6
Predecessor Vertex	-	B	S	A	C

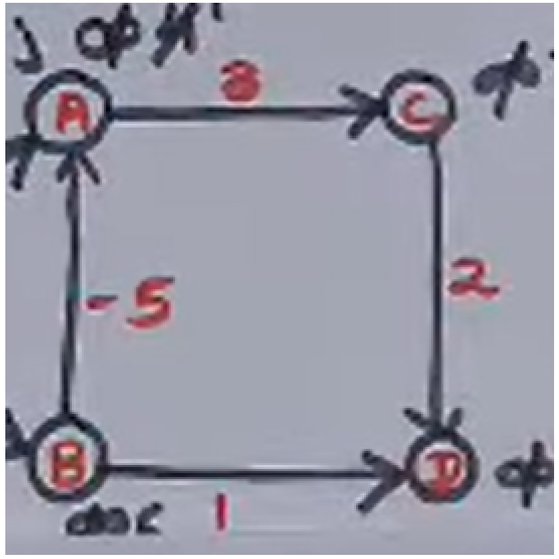


Path	Shortest Distance	Shortest Path
S-A	1	S-B-A (6-5)=1
S-B	6	S-B (6)
S-C	4	S-B-A-C (6-5+3) = 4
S-D	6	S-B-A-C-D

(6-5+3+2)
= 6

Next Step: Ensure that **no Negative cycles** in the graph :– **Approach 1**





Consider one of the loops existing:-
Add all the weights = -5 + 3 + 2 + 1 = **1**

They should be positive. (i.e. Non-negative)

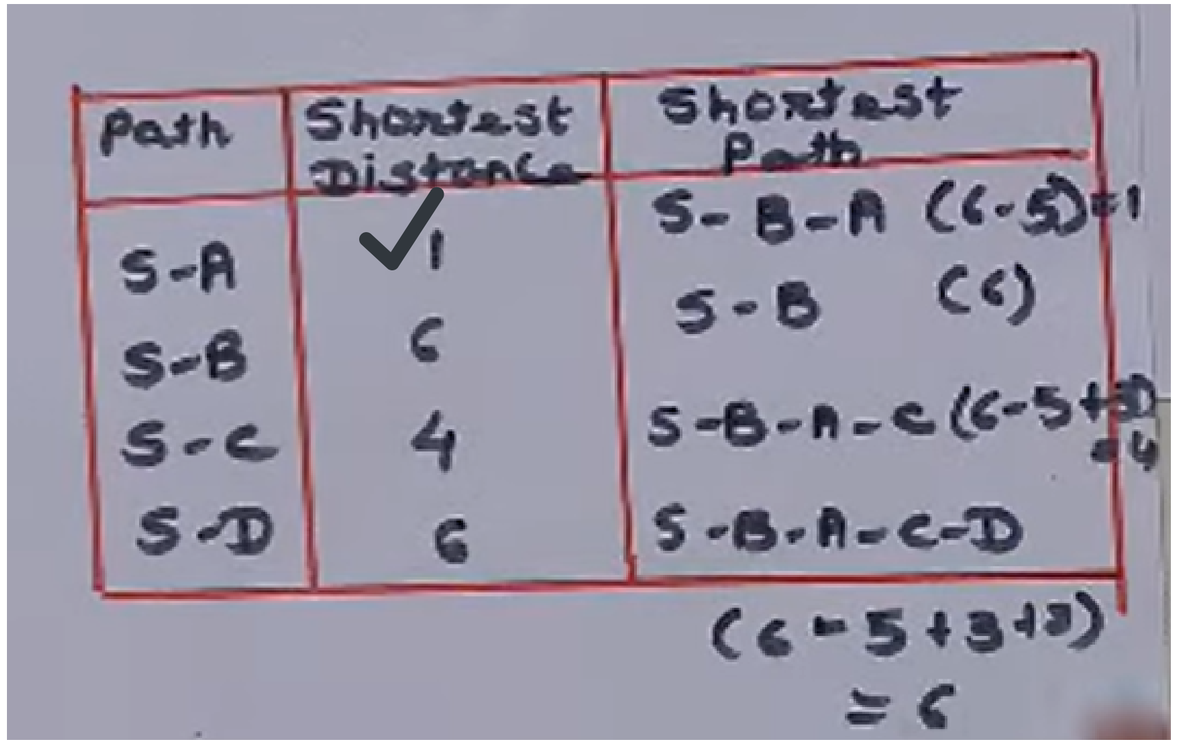
Therefore, there is no negative cycle exists in the graph.

Next Step: Ensure that **no Negative cycles** in the graph :– **Approach 2**

Note: **If there is a negative edge cycle, then our problem solution is not correct.**

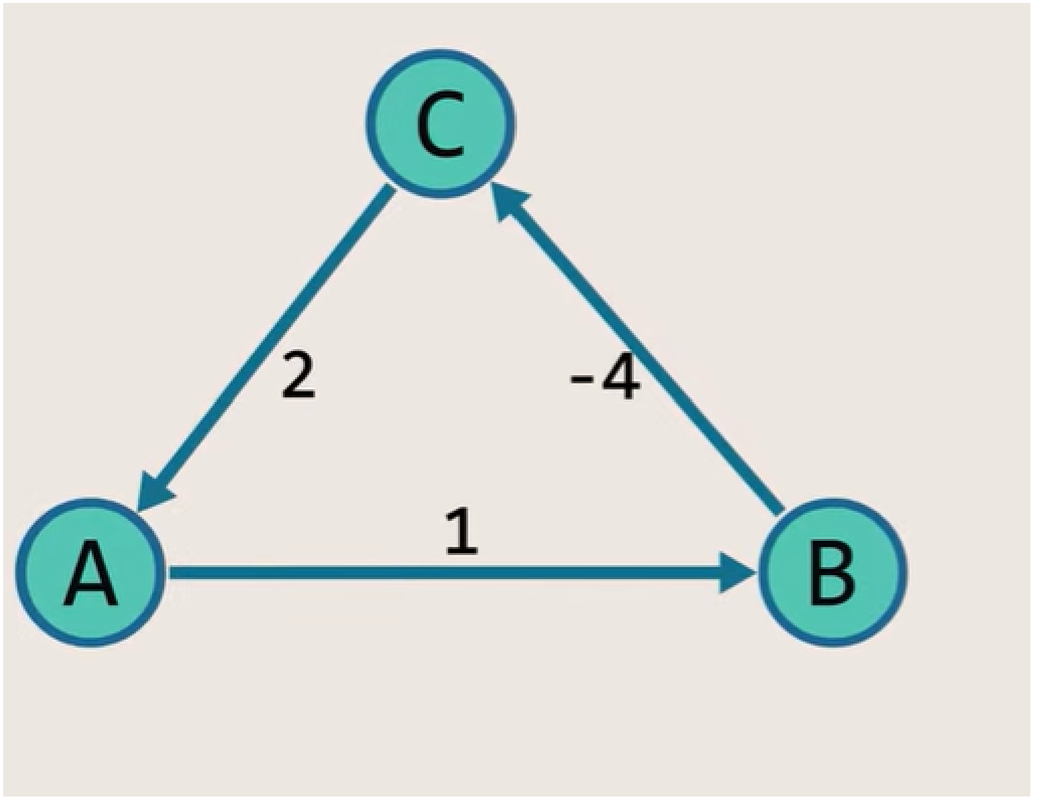
1. **Consider edge A->C**
2. **dc <= dA + CAC**
3. **4 <= 1 + 3**
4. **If this condition is satisfied for every edge, then, there is no negative weight cycle that is existing.**

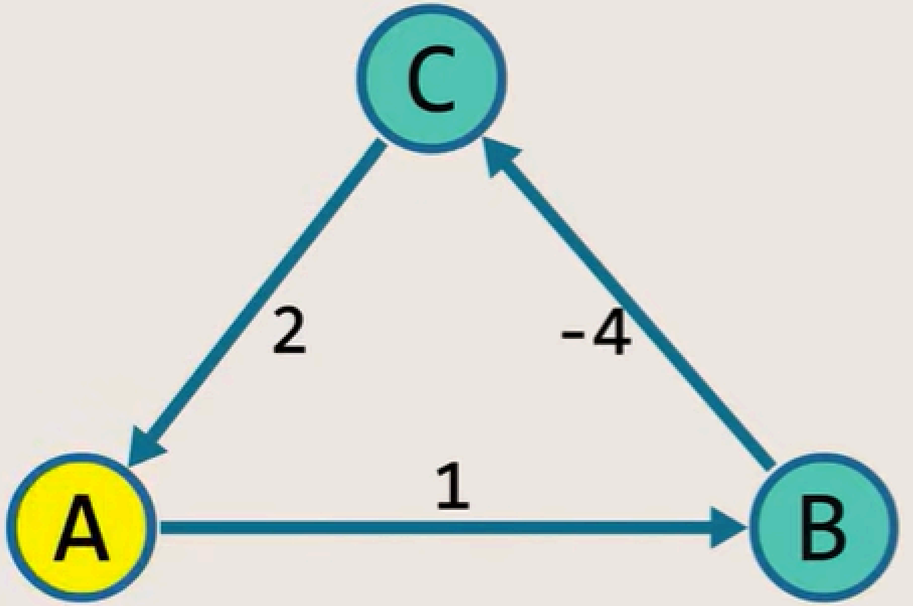
Edges	Cost
A-C	3
B-A	-5
B-D	1
C-D	2
S-A	4
S-B	6



Approach-3 To find the Negative Weight Cycle

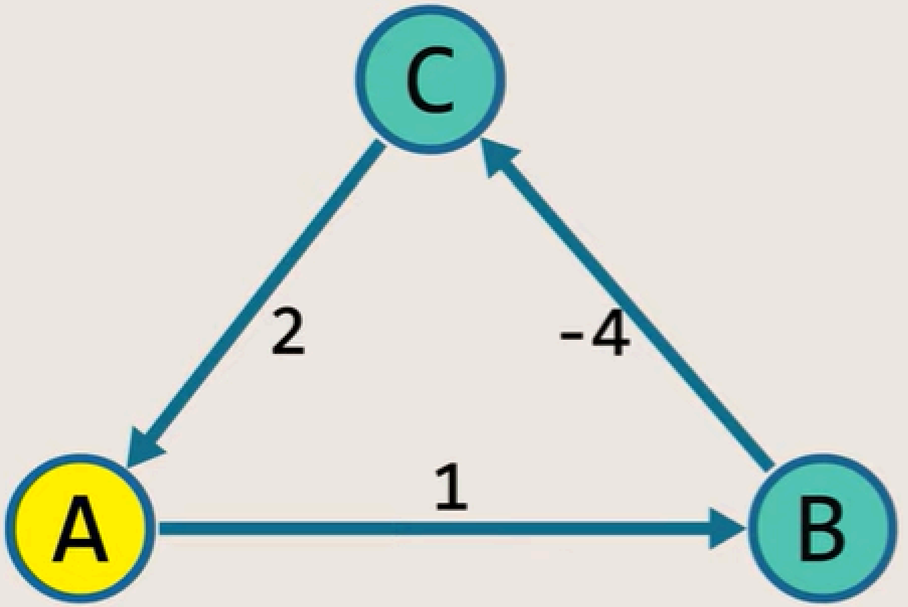
- Repeat the same procedure


C
2
-4
A
1
B



Node	Cost	Previous
A		
B		
C		

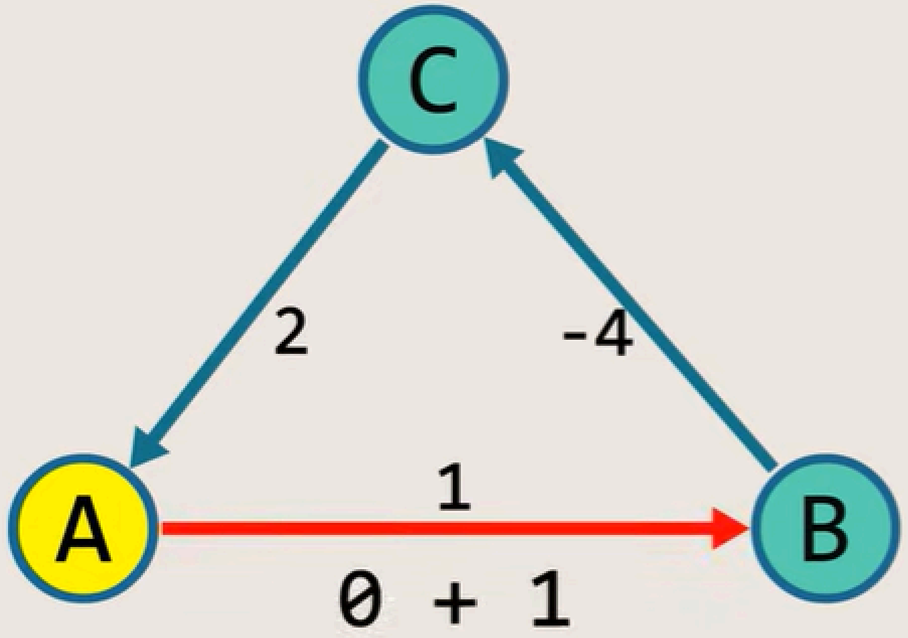
(A-B) (B-C) (C-A)



Node	Cost	Previous
A	0	
B	∞	
C	∞	

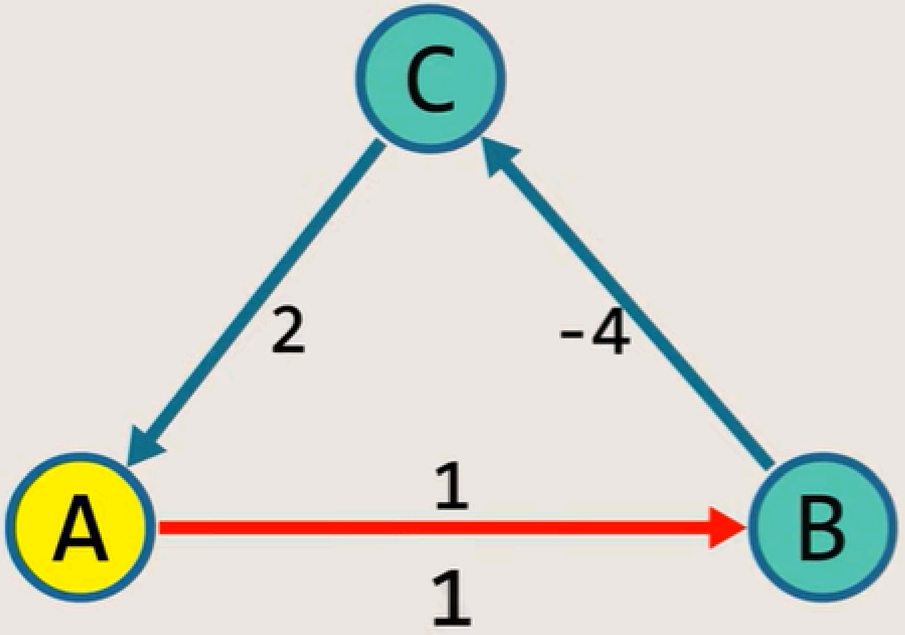
(A-B) (B-C) (C-A)

Iteration 1



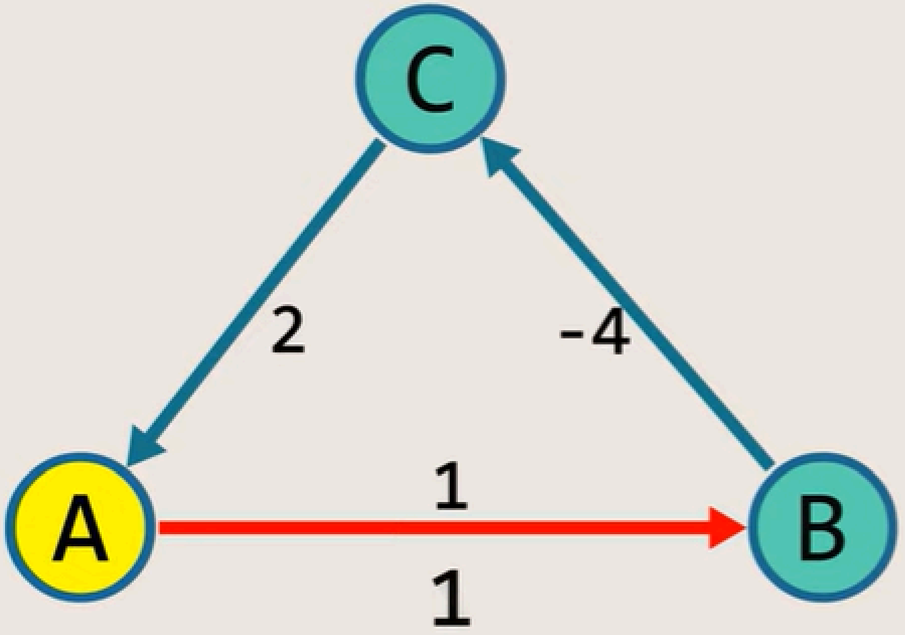
Node	Cost	Previous
A	0	
B	∞	
C	∞	

(A-B) (B-C) (C-A)



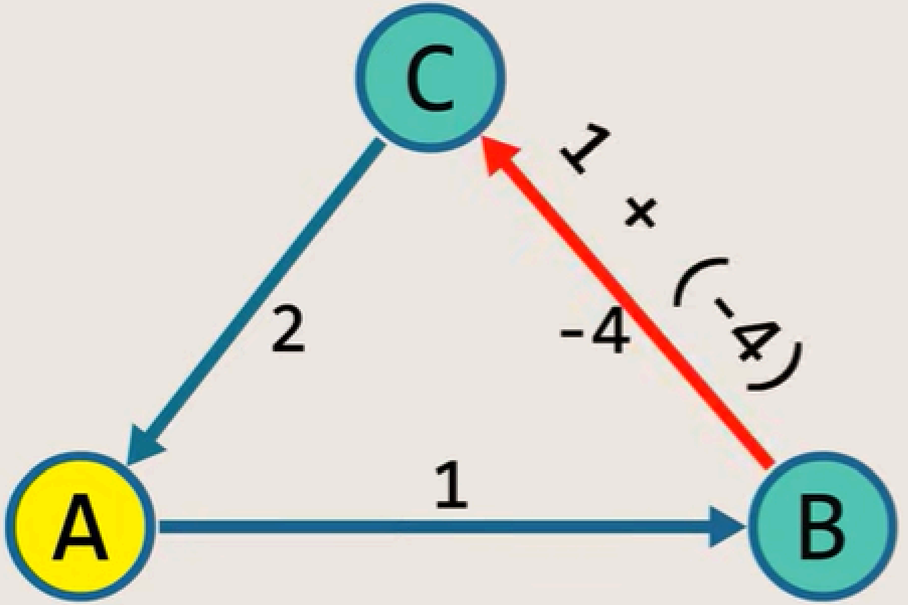
Node	Cost	Previous
A	0	
B	∞	
C	∞	

(A-B) (B-C) (C-A)



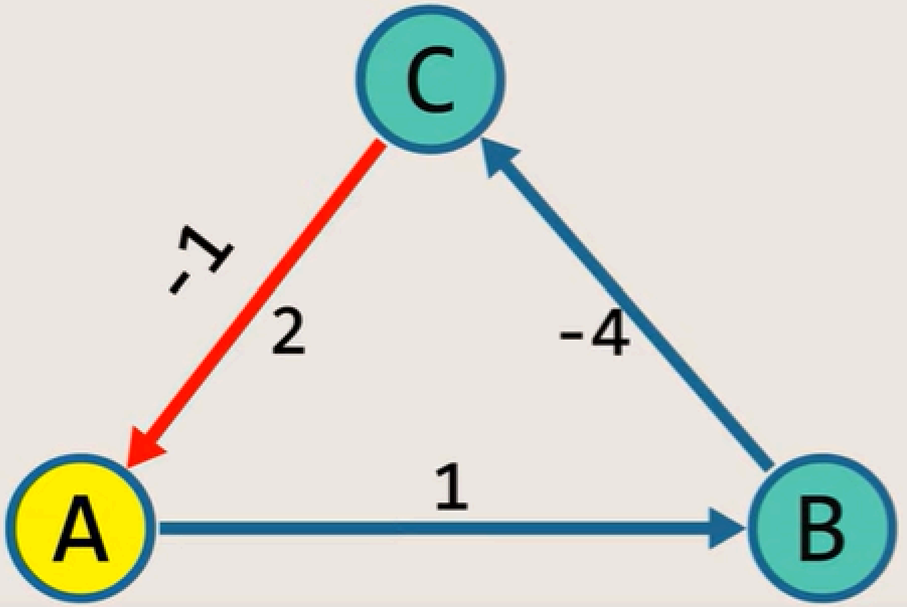
Node	Cost	Previous
A	0	
B	1	A
C	∞	

(A-B) (B-C) (C-A)



Node	Cost	Previous
A	0	
B	1	A
C	-3	B

(A-B) **(B-C)** (C-A)

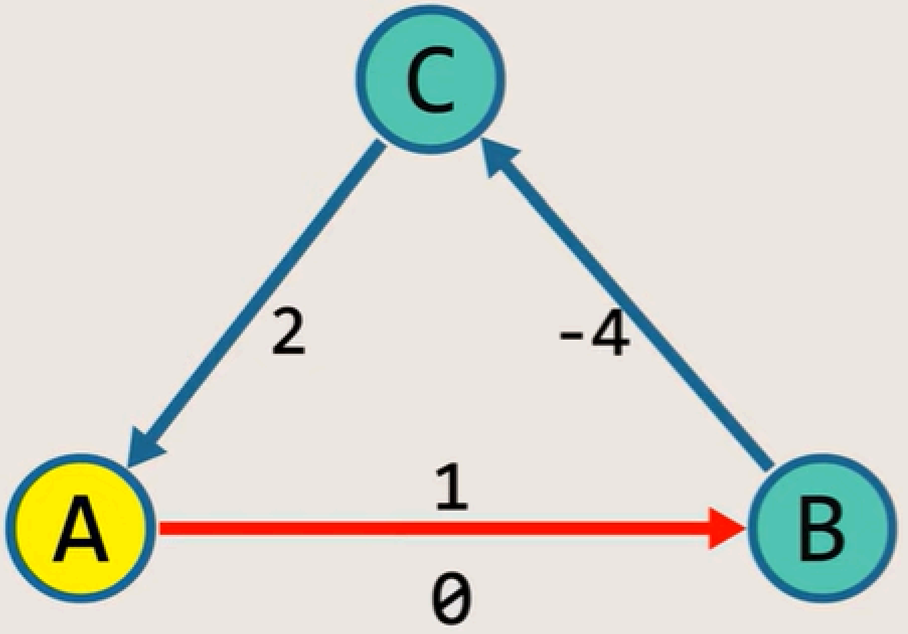


Node	Cost	Previous
A	-1	C
B	1	A
C	-3	B

(A-B) (B-C) (C-A)

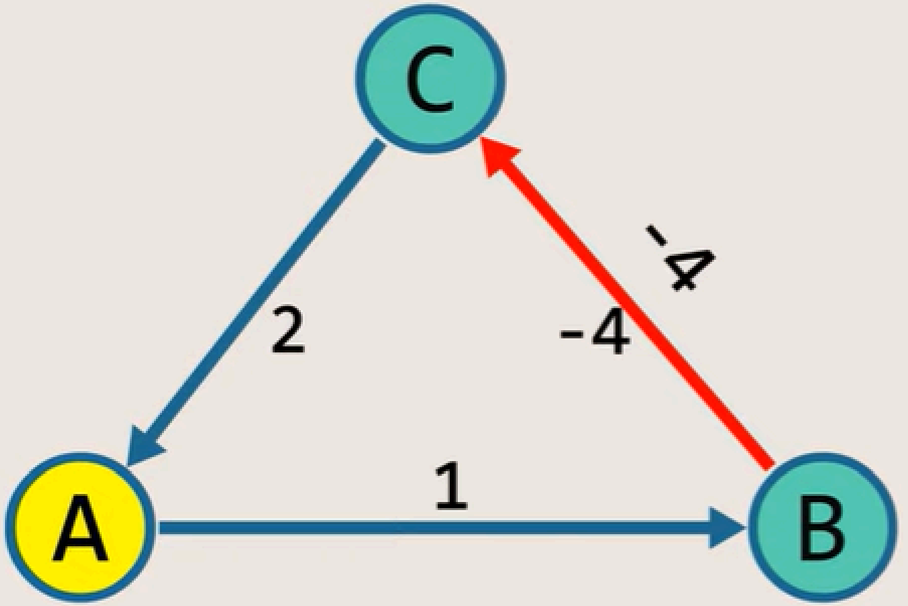
Iteration 2

- The Main reason that the algorithm fails in the presence of negative weight cycles is that it keeps finding smaller and smaller paths indefinitely
- It will continue converging lower and lower cost forever without ever stabilizing



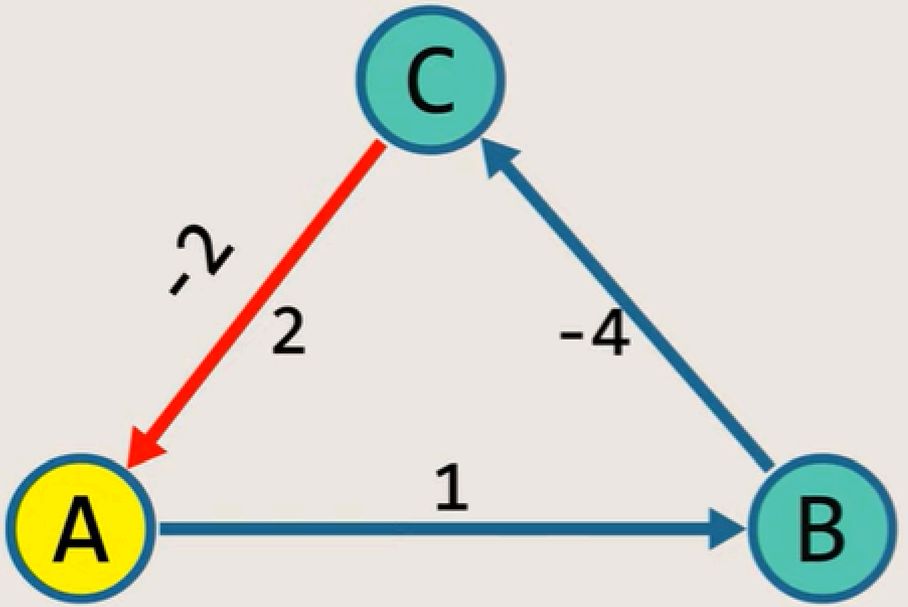
Node	Cost	Previous
A	-1	C
B	0	A
C	-3	B

(A-B) (B-C) (C-A)



Node	Cost	Previous
A	-1	C
B	0	A
C	-4	B

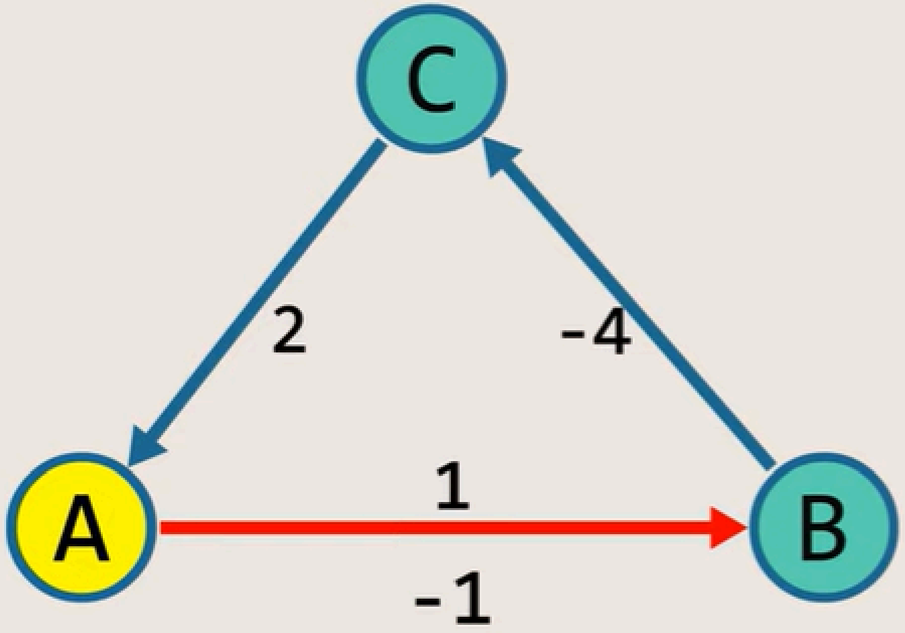
(A-B) **(B-C)** (C-A)



Node	Cost	Previous
A	-2	C
B	0	A
C	-4	B

(A-B) (B-C) (C-A)

Iteration 3



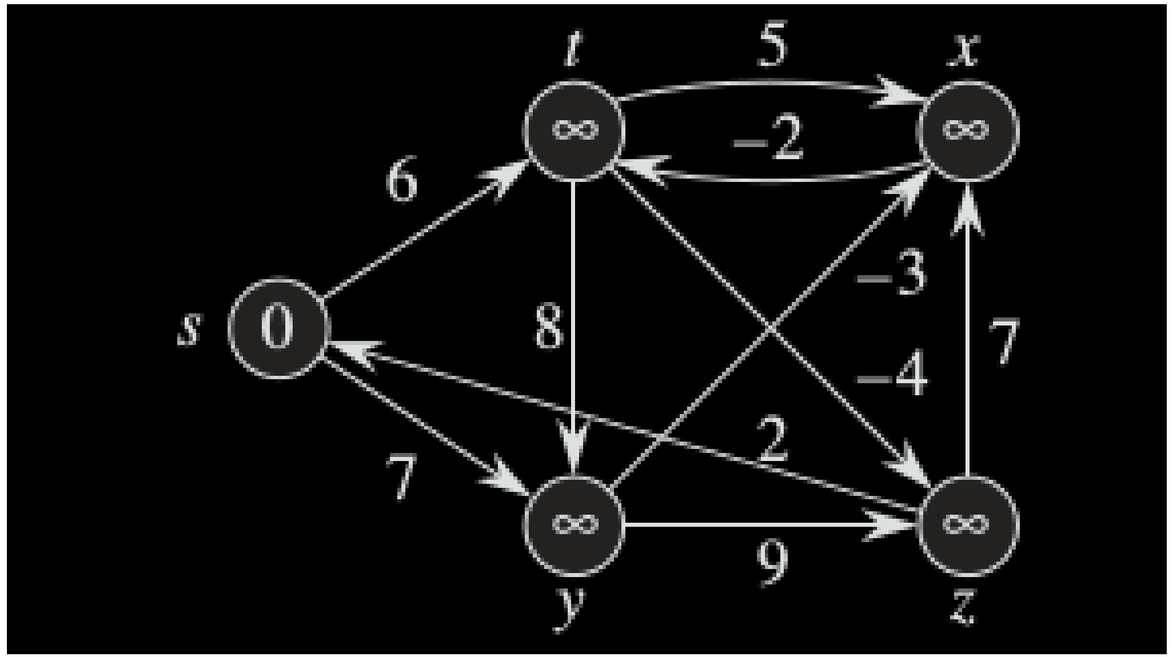
-1 < 0 (cur_cost of B)

Node	Cost	Previous
A	-2	C
B	0	A
C	-4	B

(A-B) (B-C) (C-A)

Indicating that updates are still occurring which should not happen if there were no negative weight cycle.
This confirms the presence of negative weight cycle causing the algorithm to fail at this point.

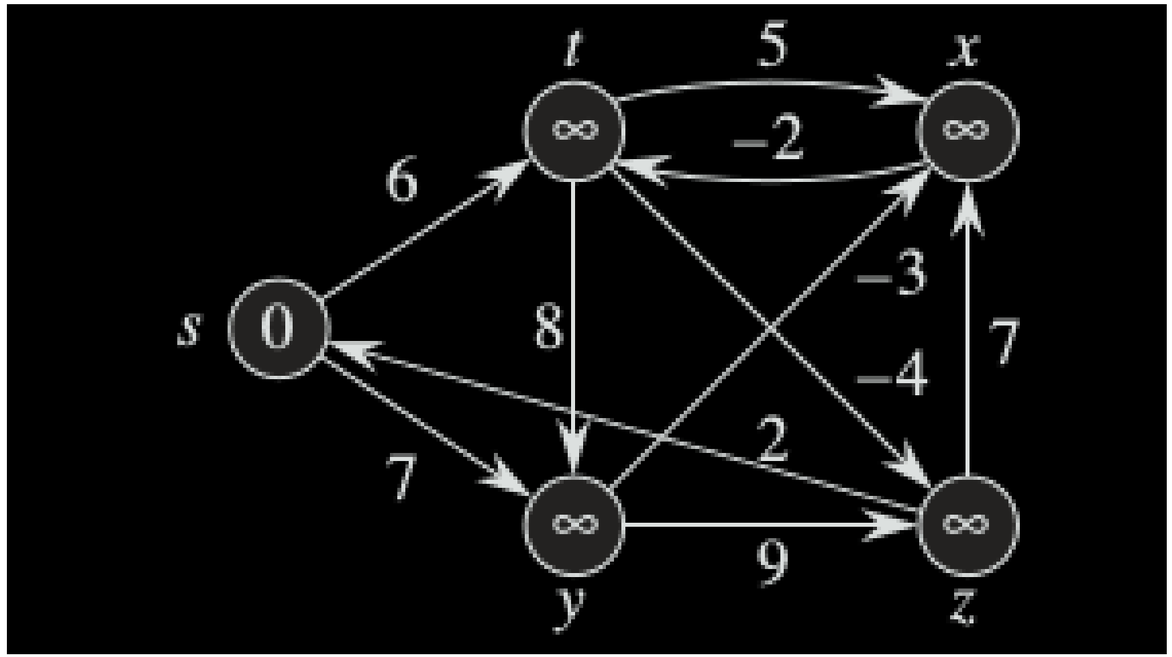
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	
x	∞	∞
y	∞	
z	∞	

(t, x)									

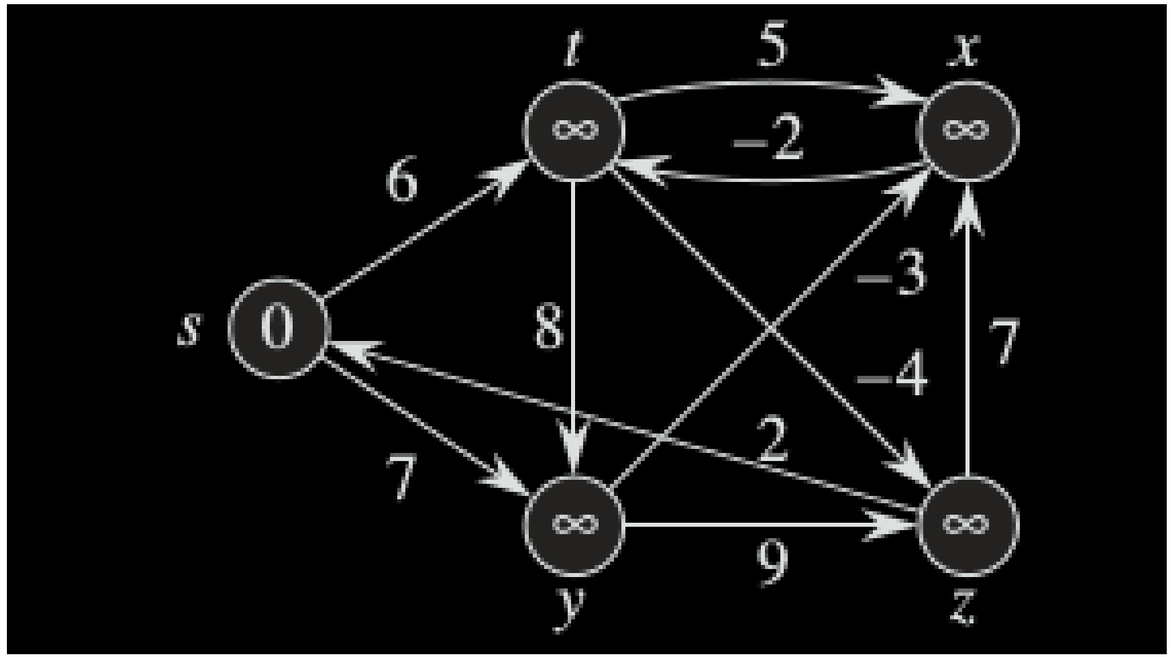
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	
x	∞	∞
y	∞	∞
z	∞	

(t, x)	(t, y)								

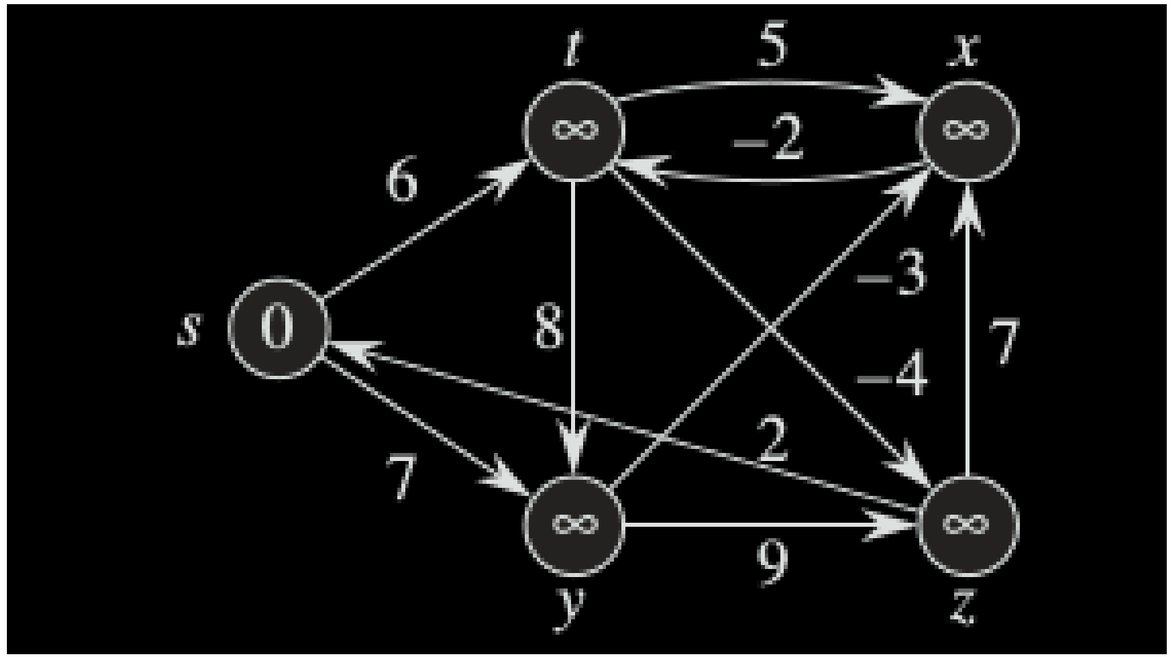
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	
x	∞	∞
y	∞	∞
z	∞	∞

(t, x)	(t, y)	(t, z)							

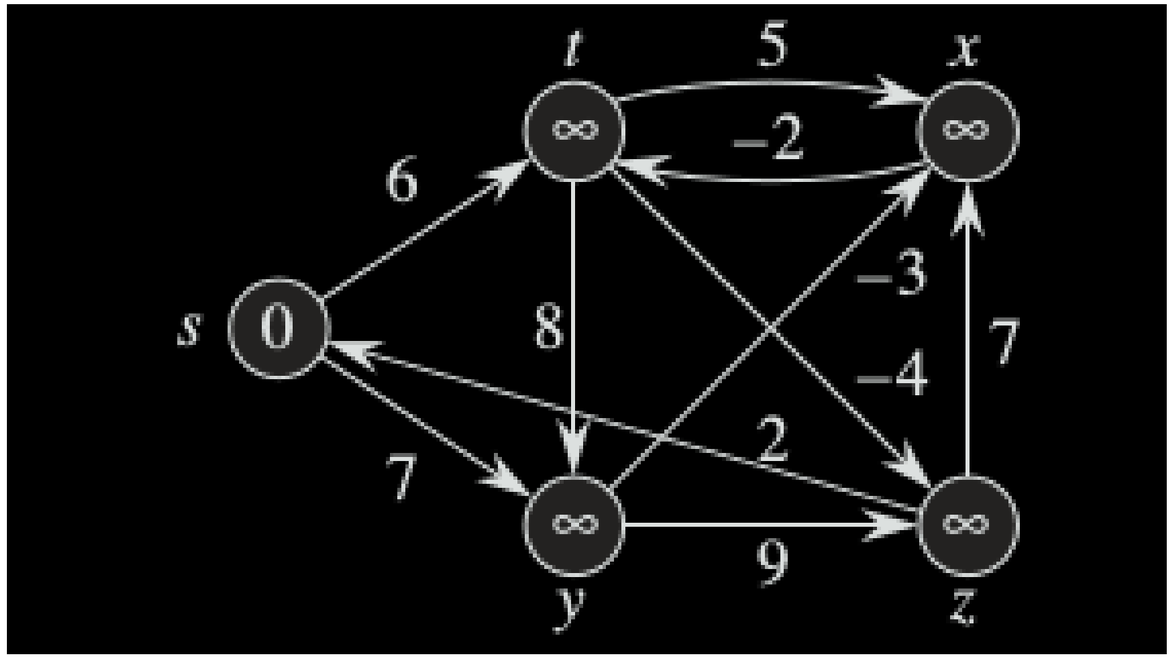
Bellman-Ford algorithm



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	∞
x	∞	∞
y	∞	∞
z	∞	∞

(t, x)	(t, y)	(t, z)	(x, t)						

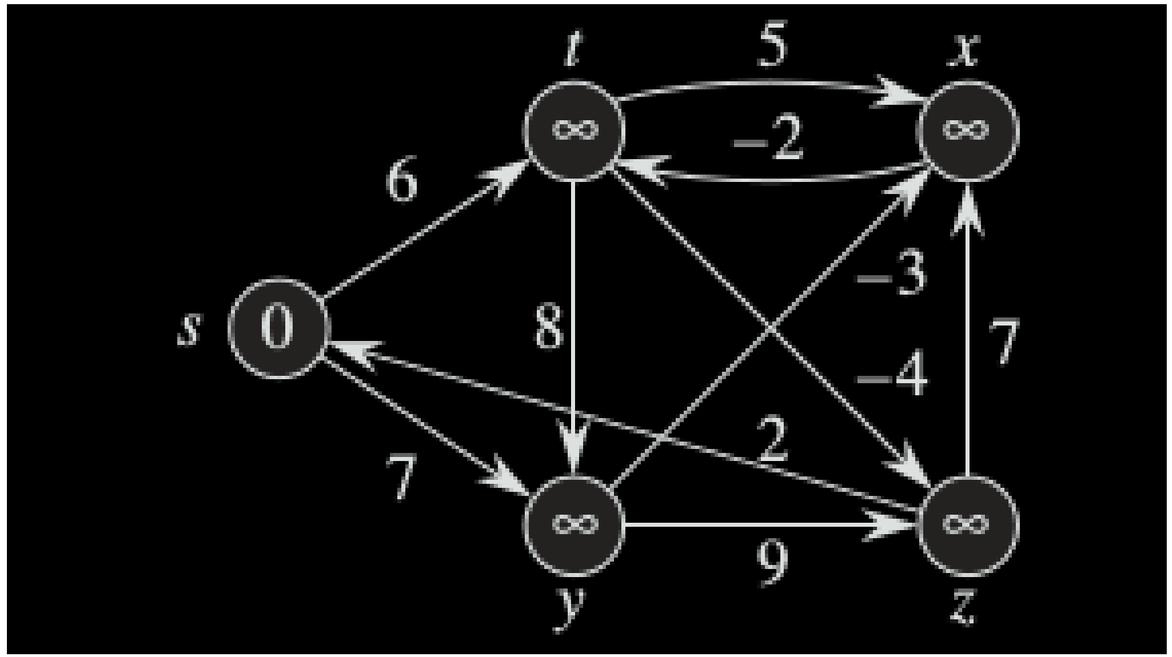
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	∞
x	∞	∞
y	∞	∞
z	∞	∞

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)					

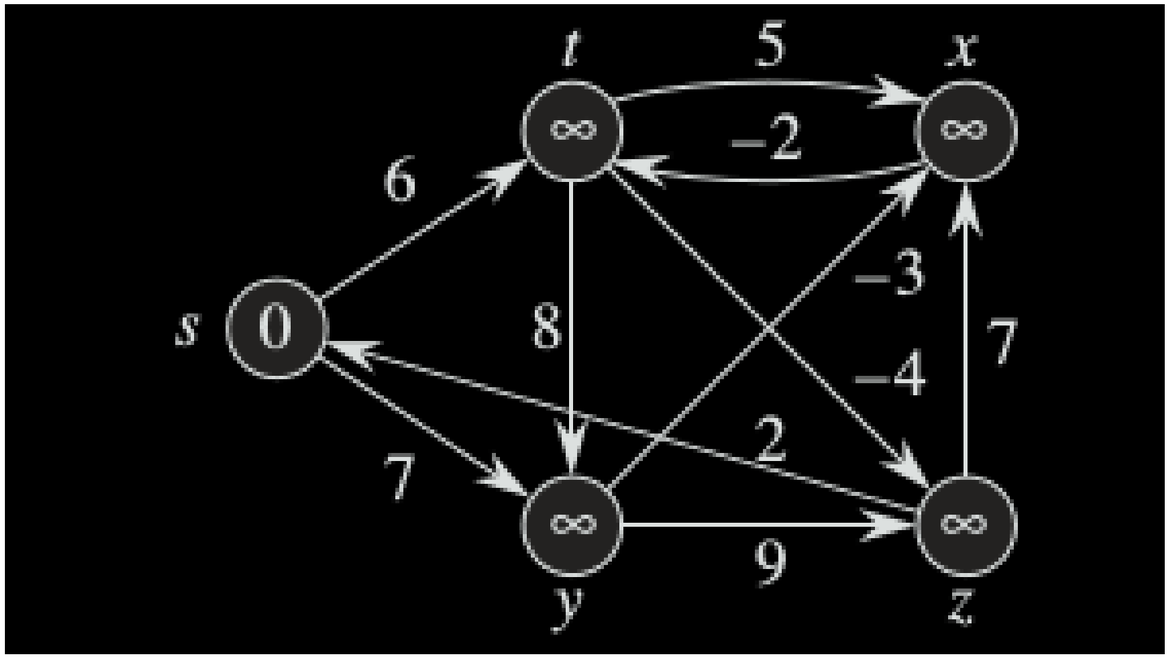
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	∞
x	∞	∞
y	∞	∞
z	∞	∞

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)				

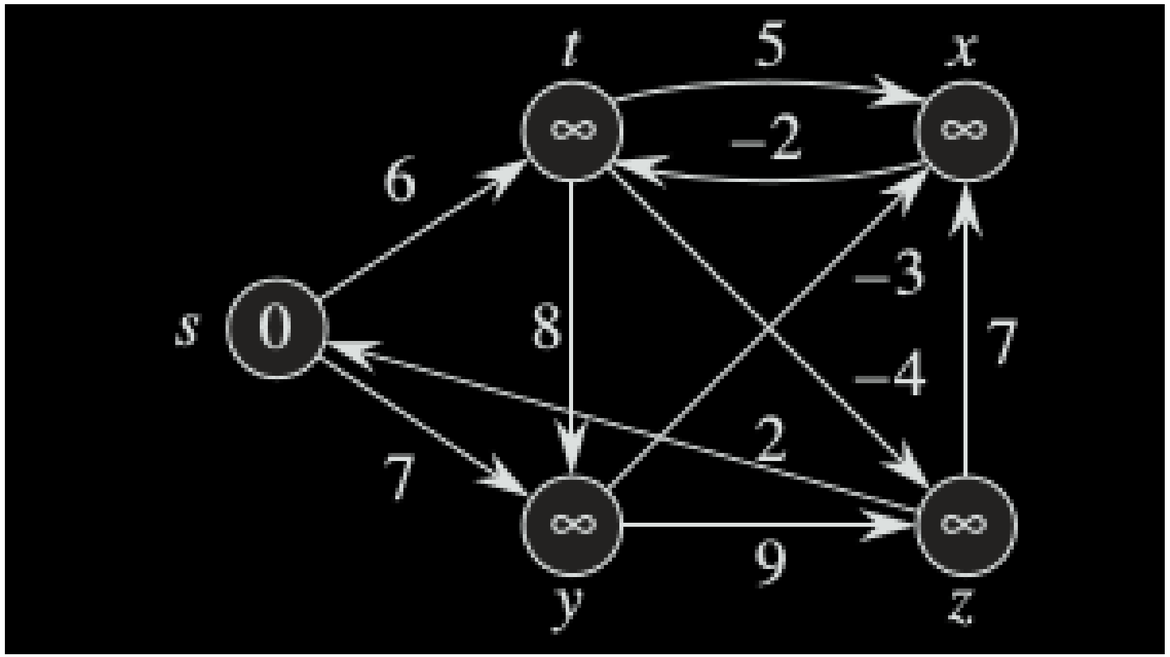
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	∞
x	∞	∞
y	∞	∞
z	∞	∞

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)			

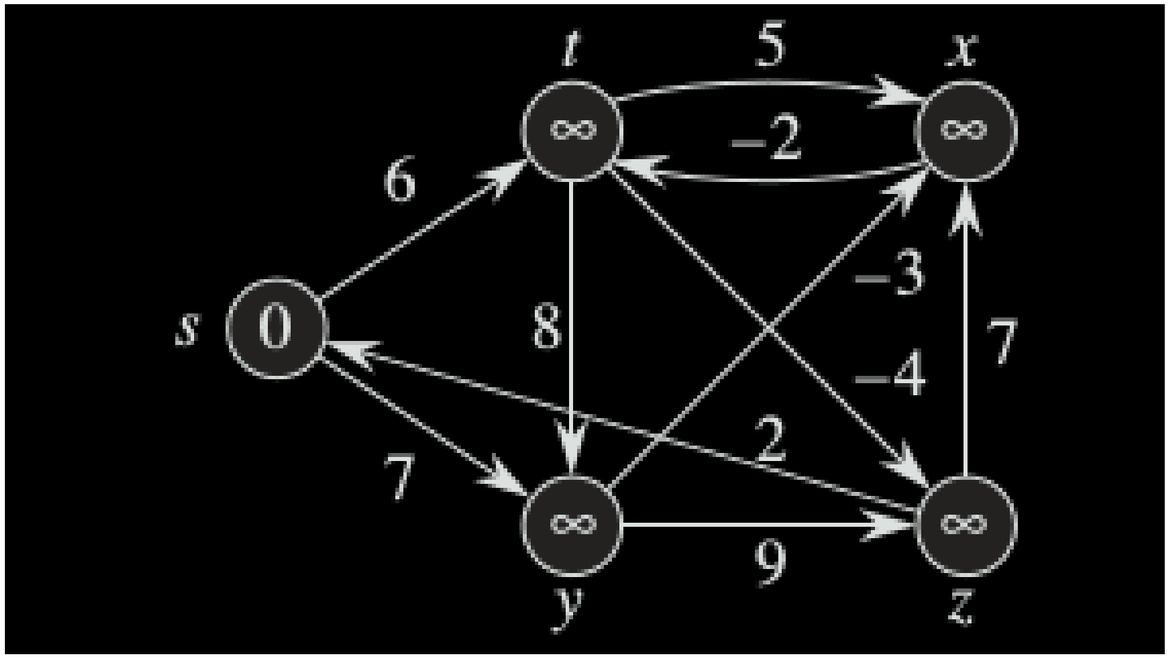
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	∞
x	∞	∞
y	∞	∞
z	∞	∞

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)		

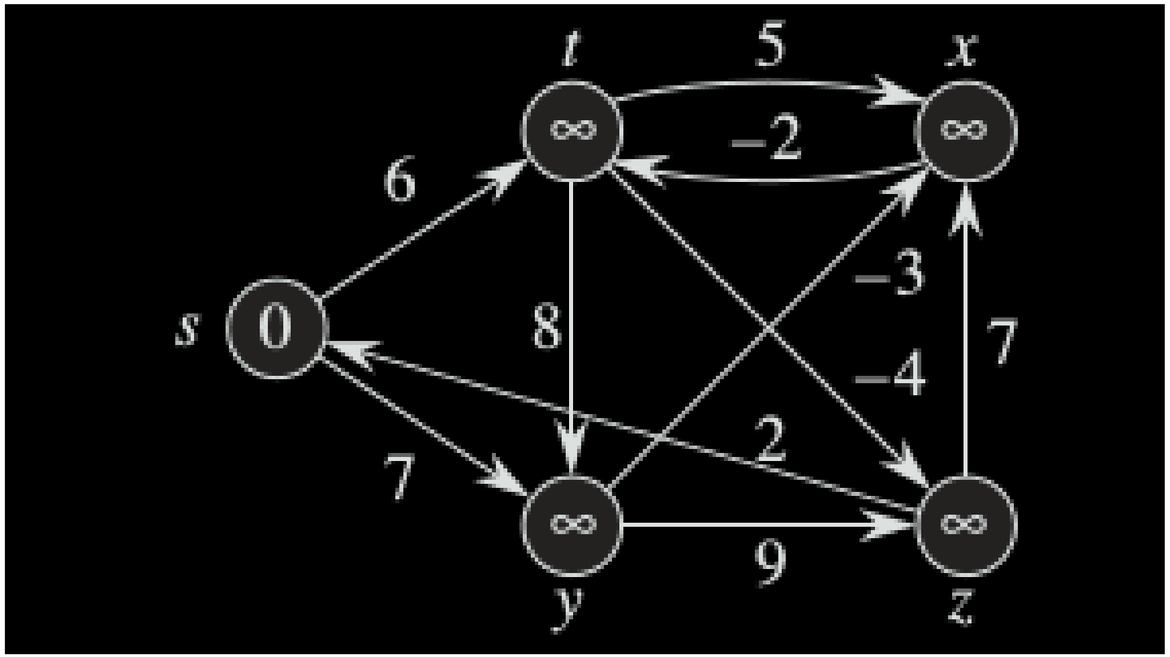
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	**0**	**0**
t	**∞**	**6**
x	**∞**	**∞**
y	**∞**	**∞**
z	**∞**	**∞**

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	

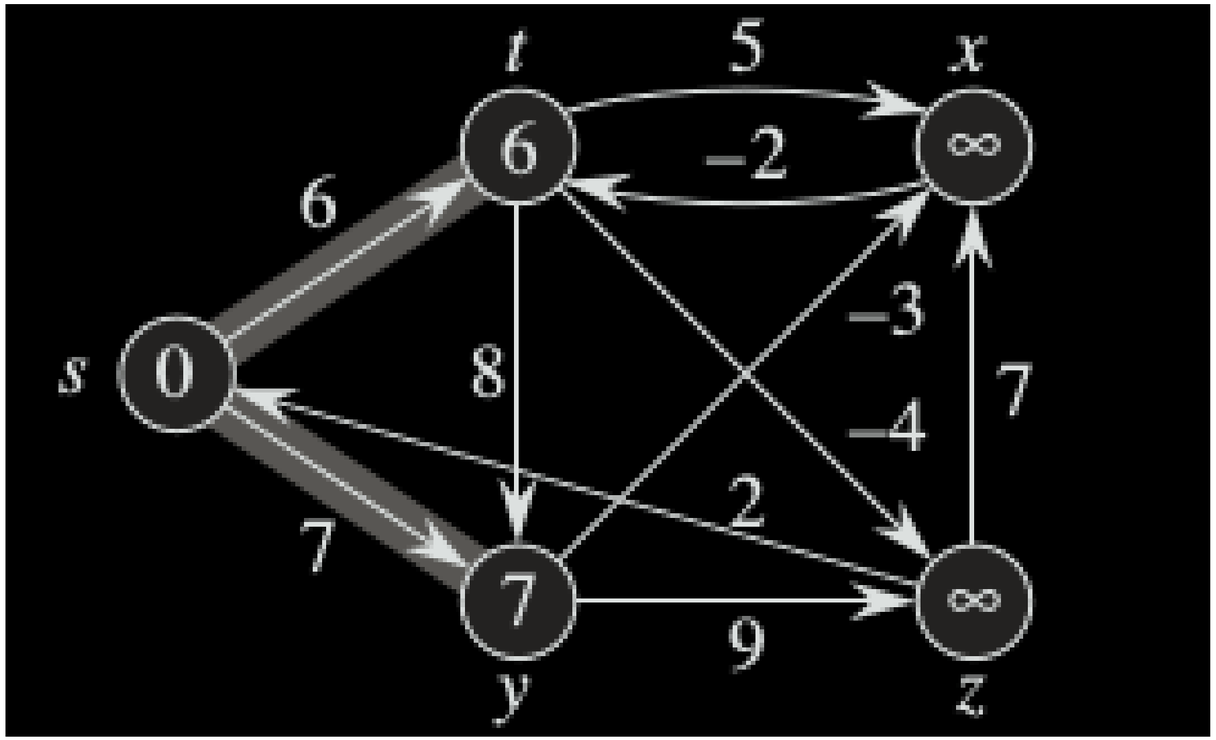
Bellman-Ford algorithm – Iteration 1



Edge Name	Old Cost	Updated Cost
s	0	0
t	∞	6
x	∞	∞
y	∞	7
z	∞	∞

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	(s,y)

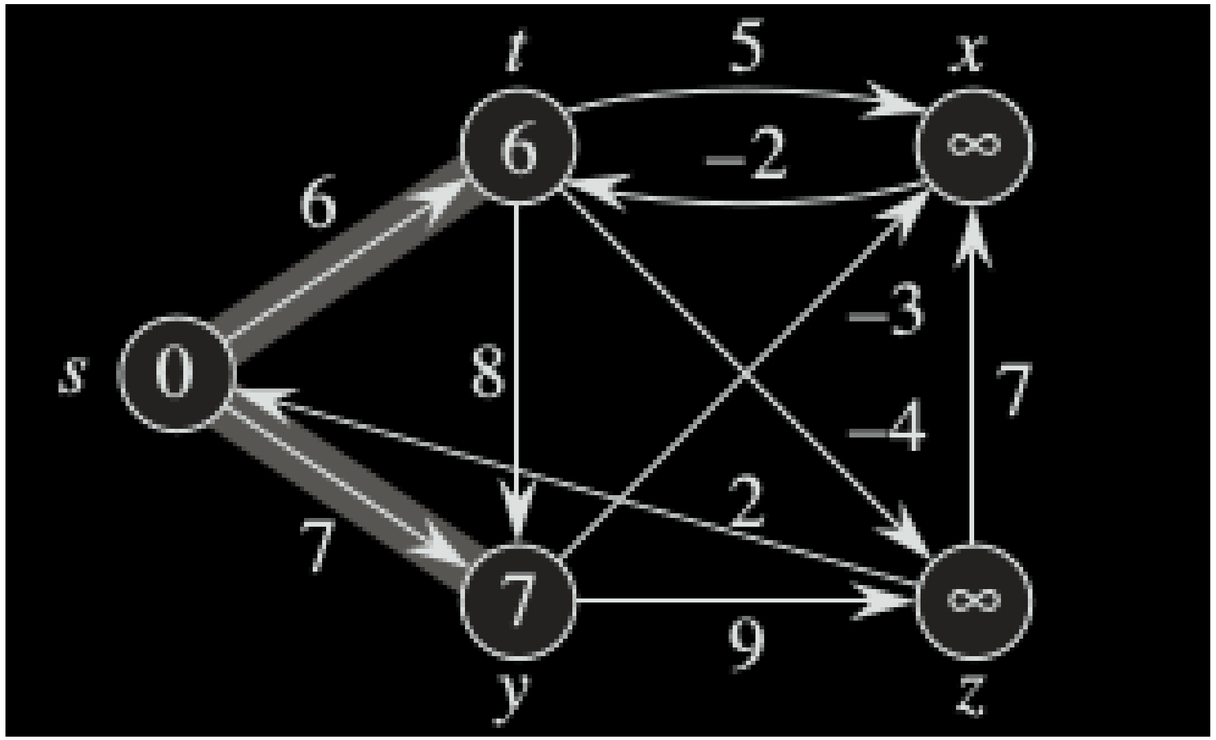
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	
x	∞	11
y	7	
z	∞	

(t, x)									

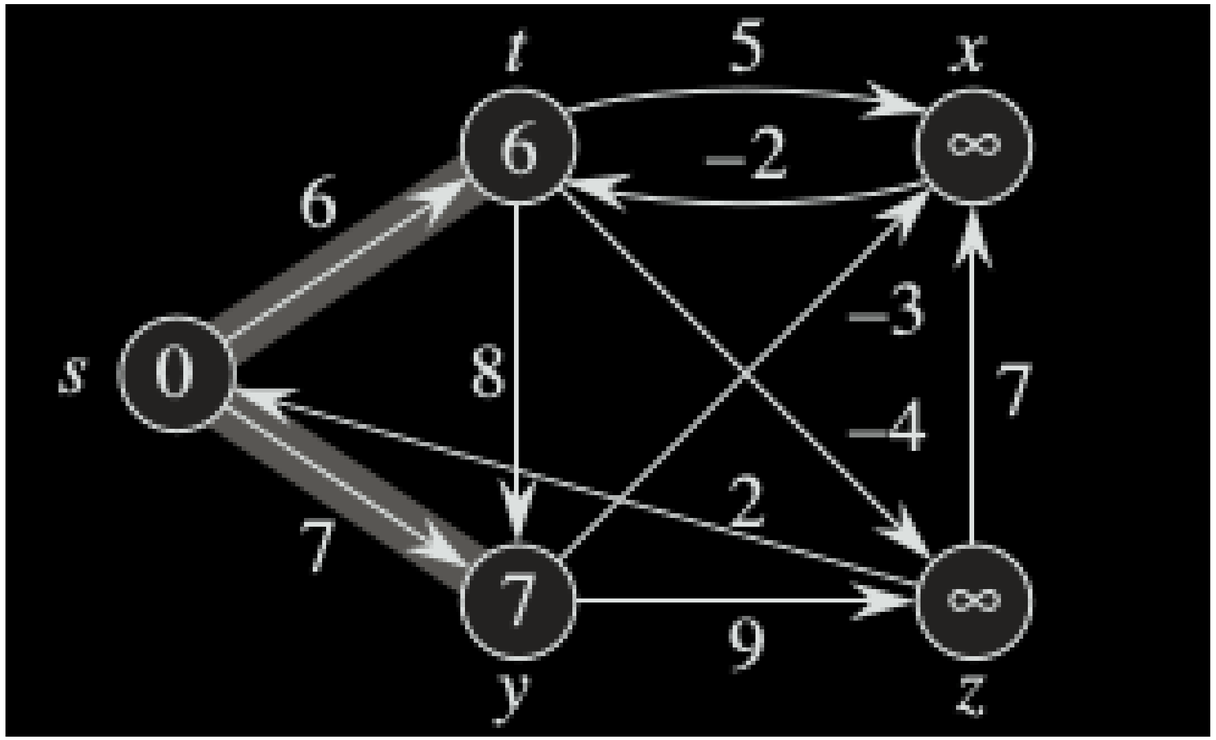
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	
x	∞	11
y	7	~~14~~ 7
z	∞	

(t, x)	(t, y)								

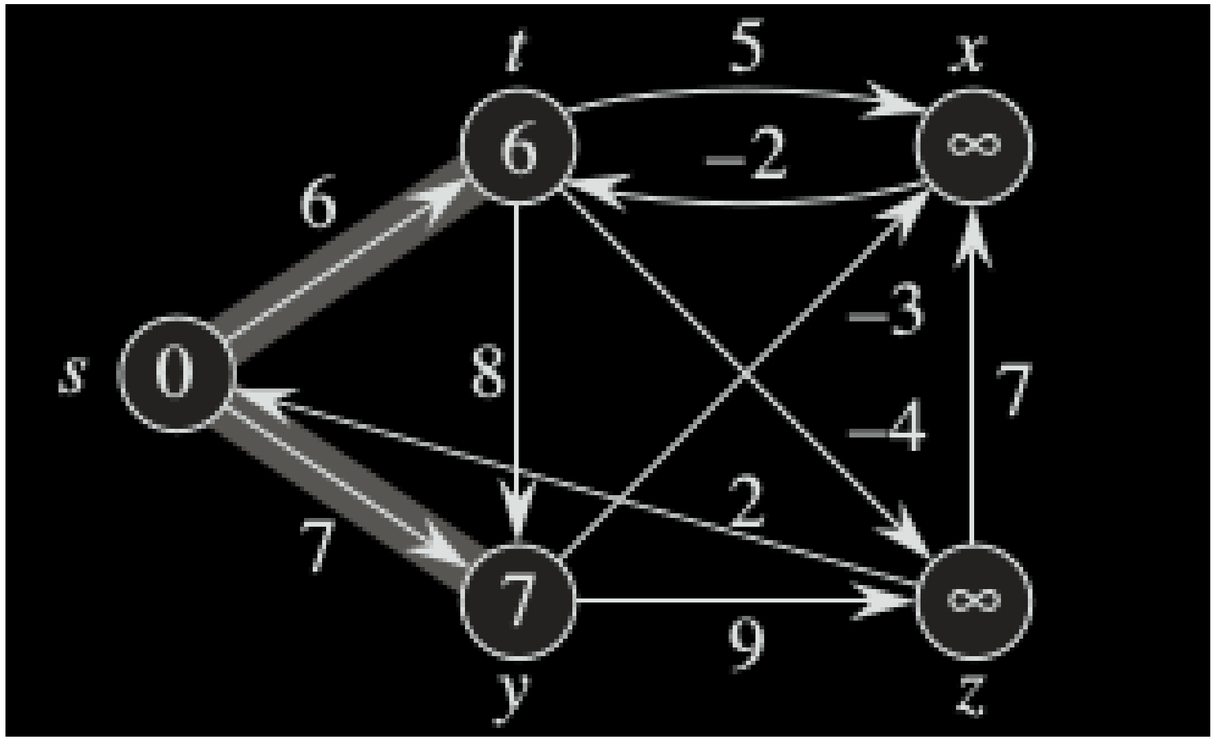
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	
x	∞	11
y	7	~~14~~ 7
z	∞	2

(t, x)	(t, y)	(t, z)							

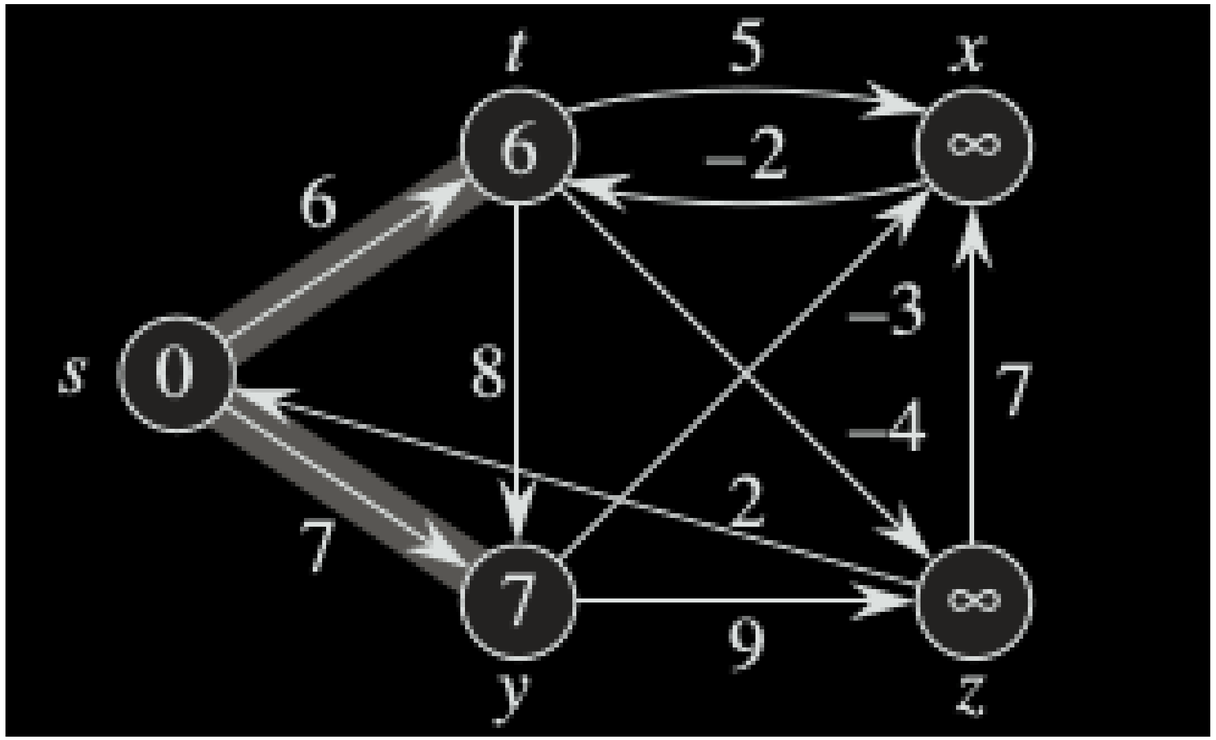
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	∞	11
y	7	~~14~~ 7
z	∞	2

(t, x)	(t, y)	(t, z)	(x, t)						

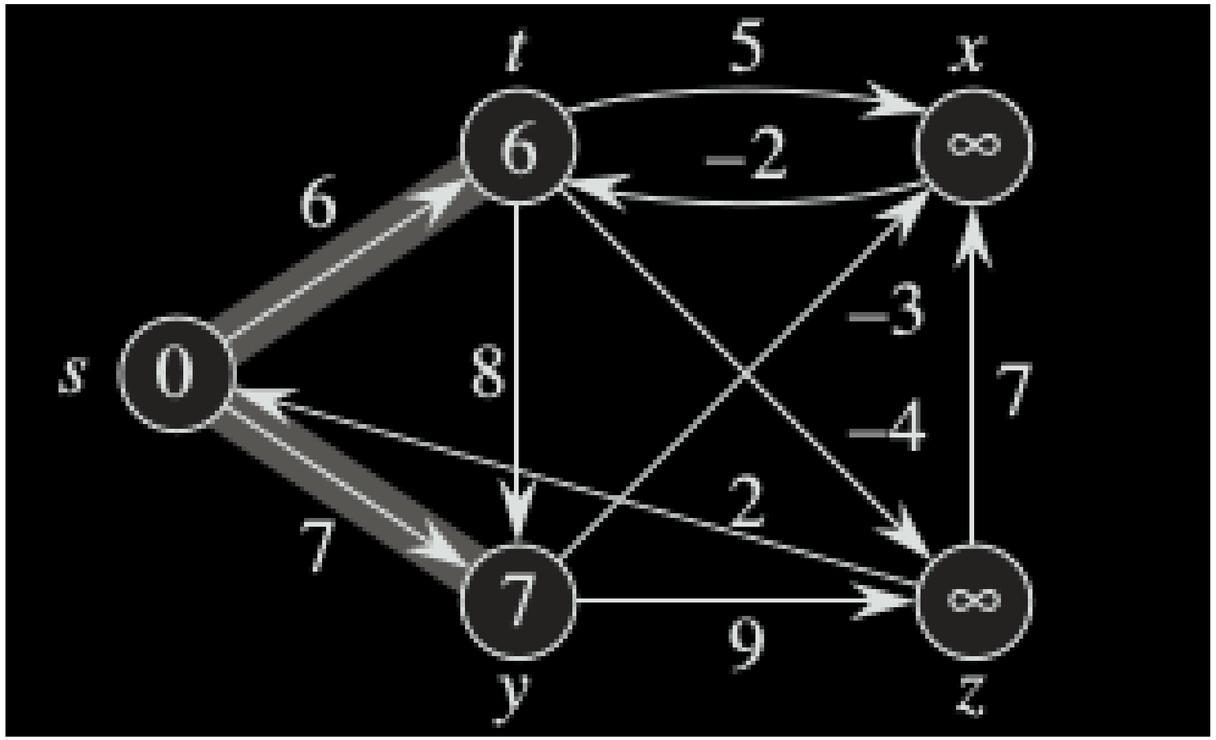
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	11	4
y	7	~~14~~ 7
z	∞	2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)					

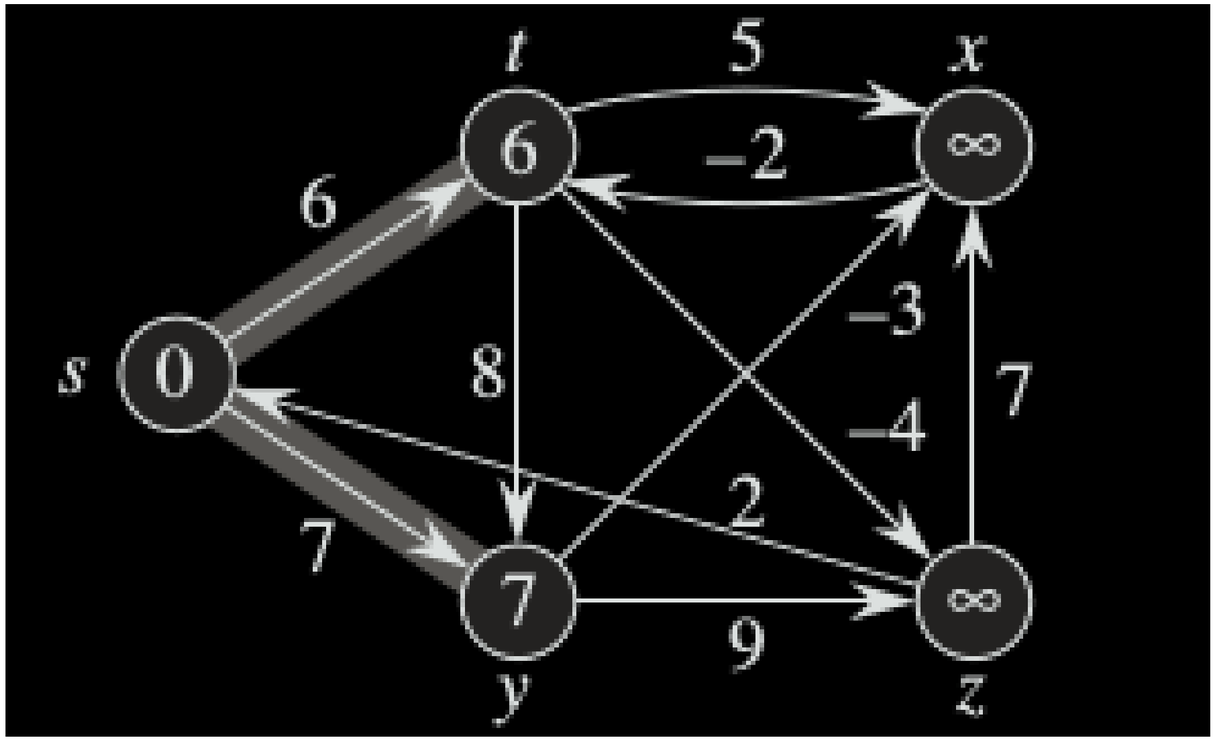
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	11	4
y	7	~~14~~ 7
z	∞	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)				

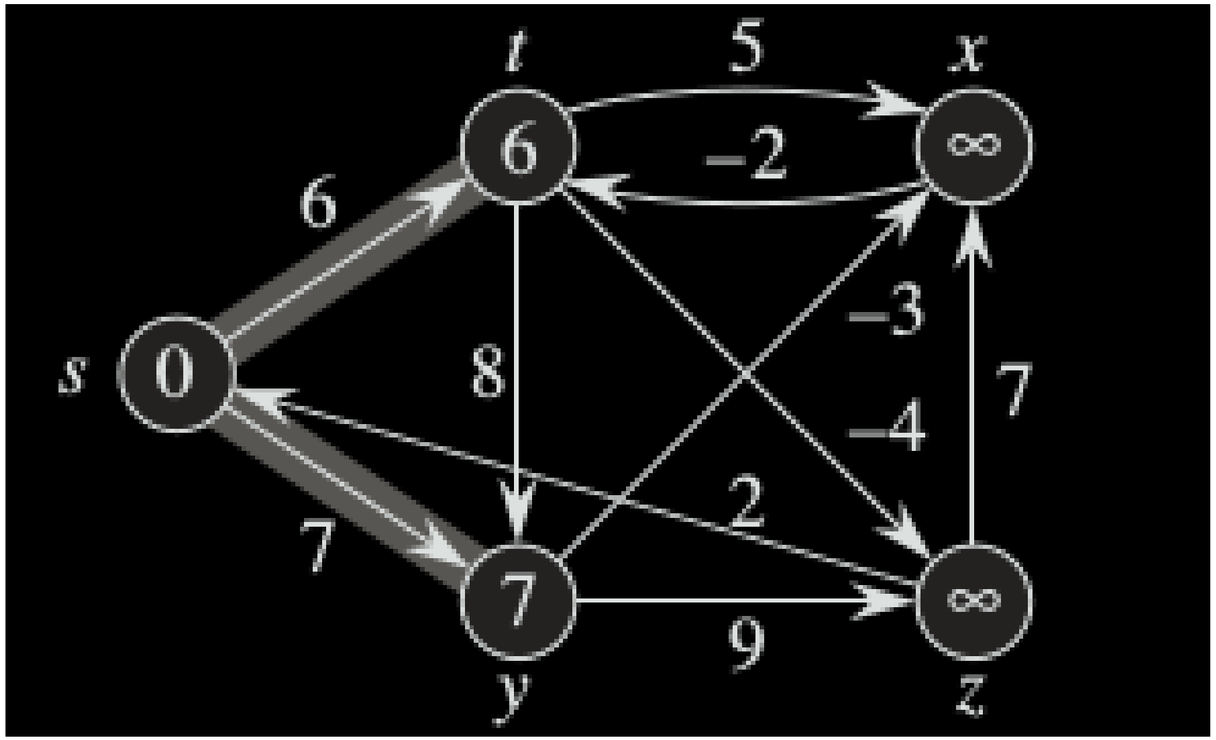
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	11	~~9~~ 4
y	7	~~14~~ 7
z	∞	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)			

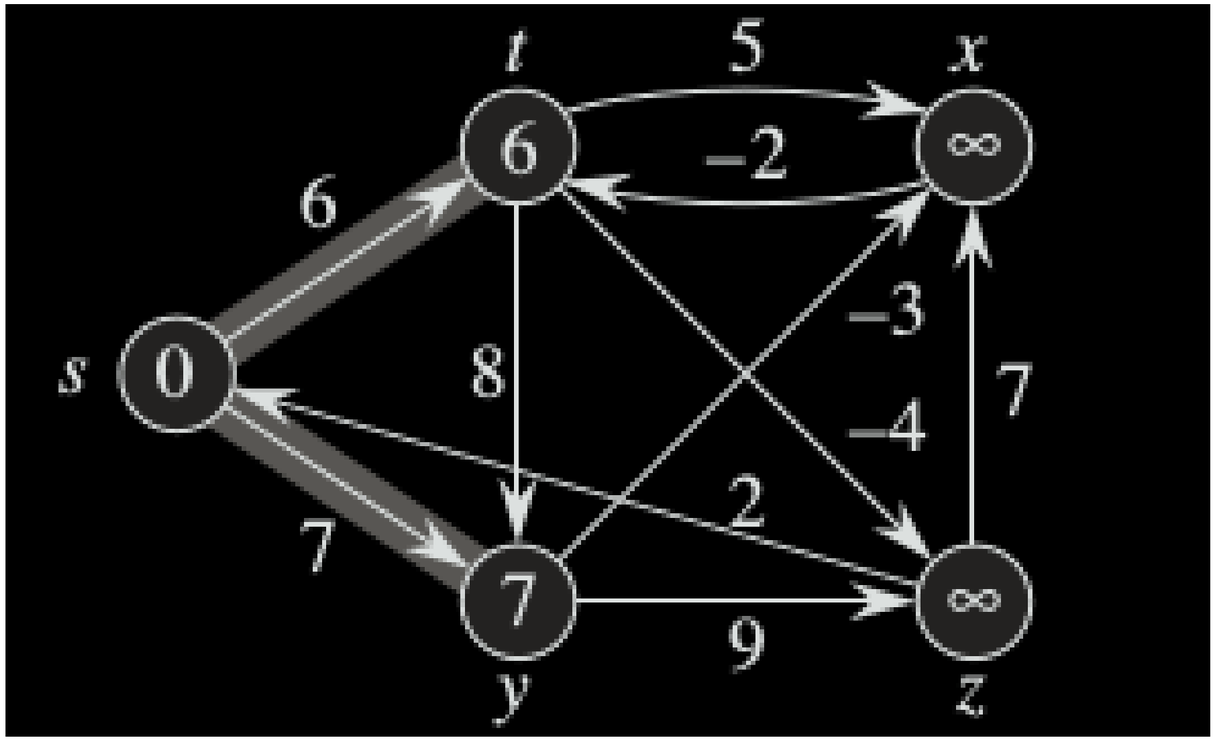
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	11	~~9~~ 4
y	7	~~14~~ 7
z	∞	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)		

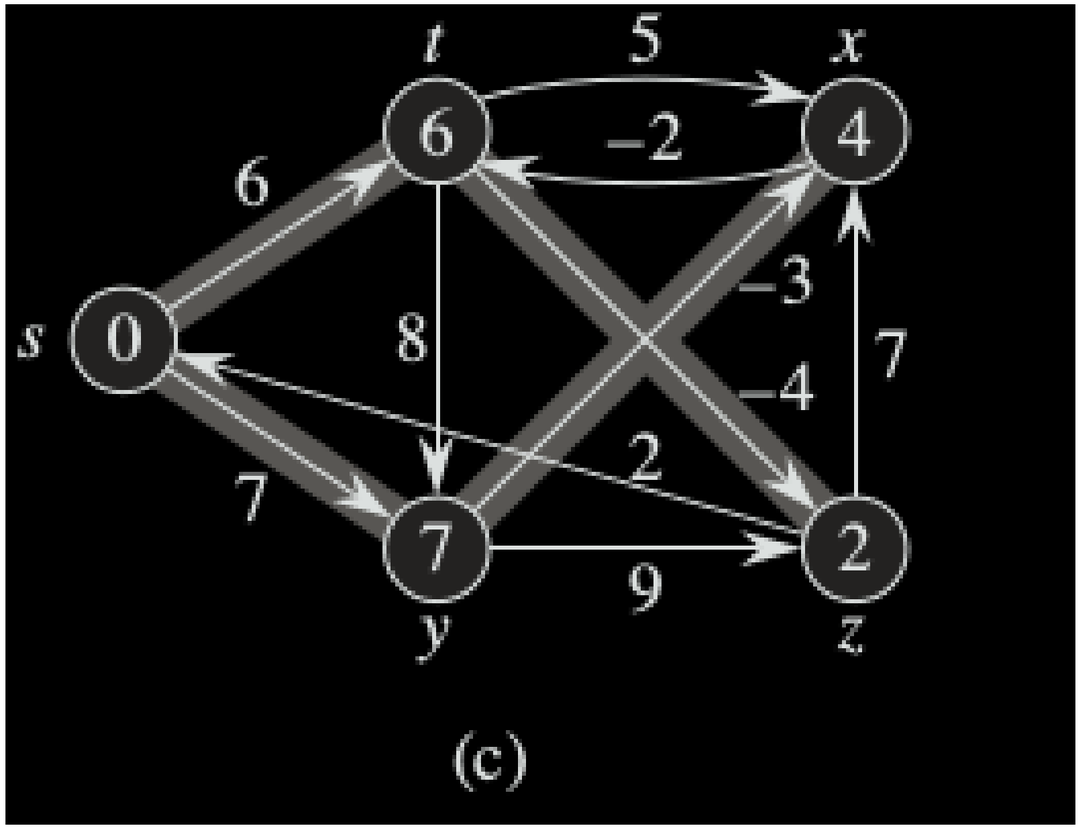
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	11	~~9~~ 4
y	7	~~14~~ 7
z	∞	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	

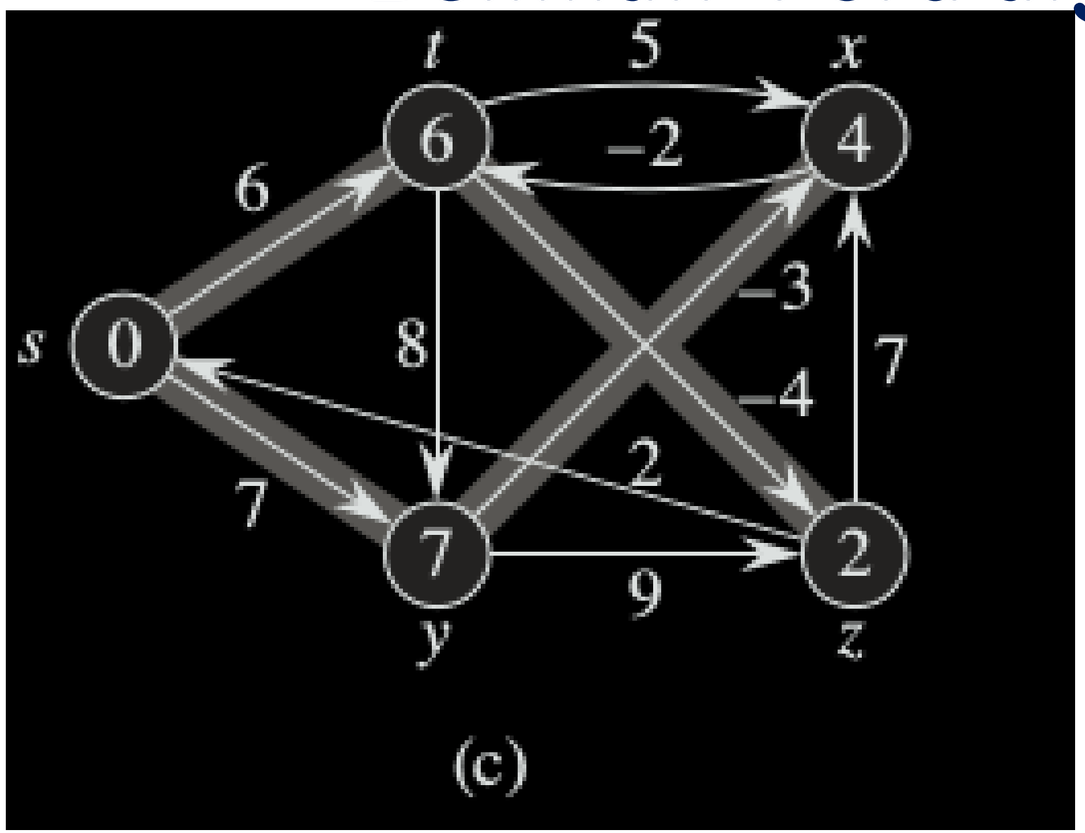
Bellman-Ford algorithm – Iteration 2



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	~~9~~ 6
x	11	~~9~~ 4
y	7	~~14~~ 7
z	∞	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	(s,y)

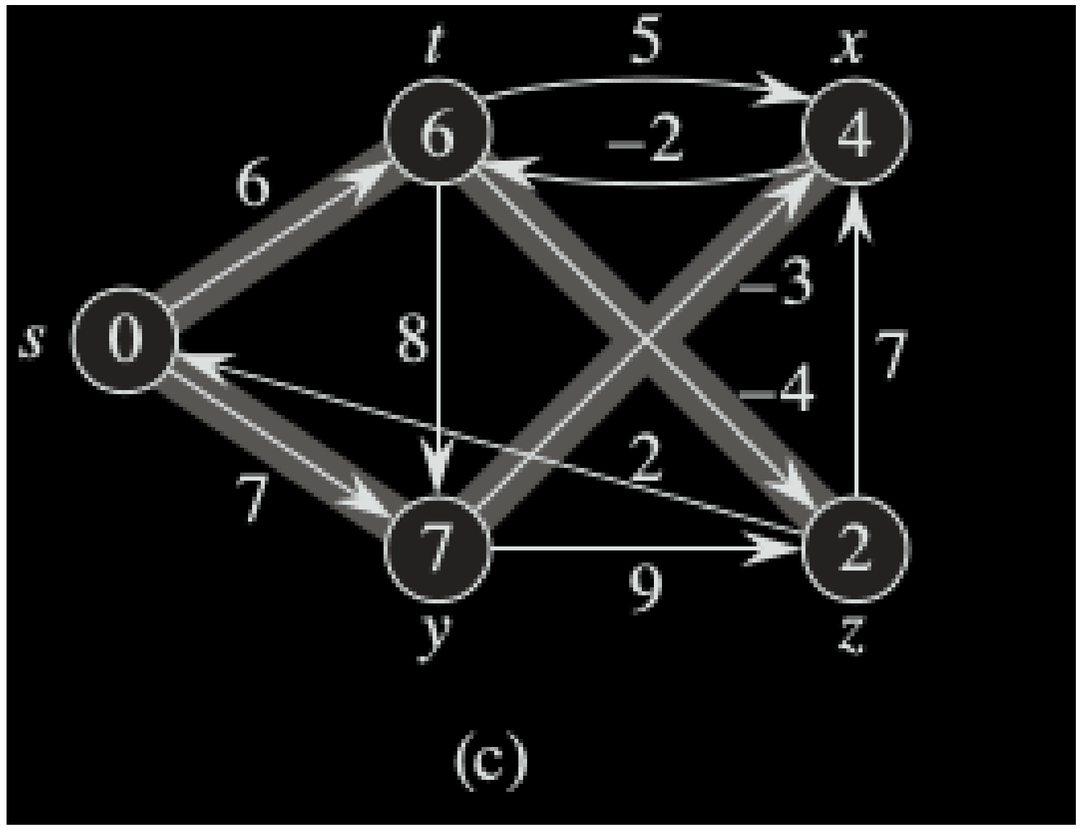
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	6
x	4	~~11~~ 4
y	7	7
z	2	2

(t, x)									

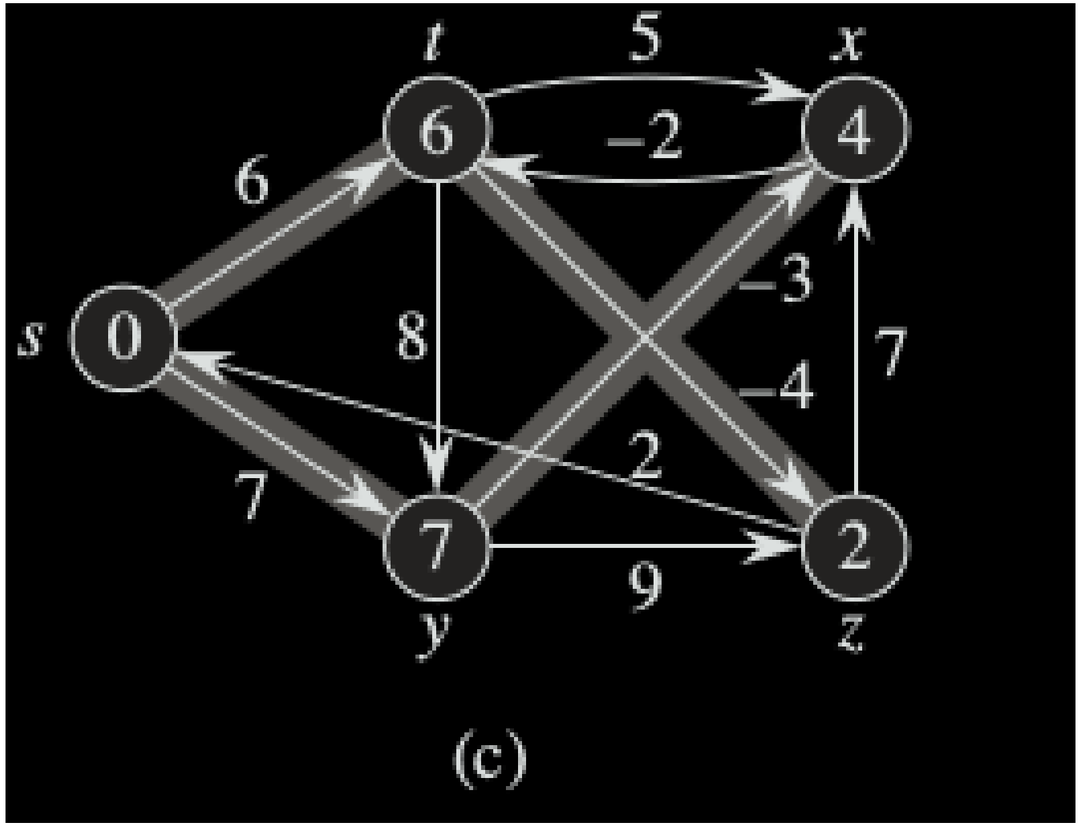
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	6
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	2

(t, x)	(t, y)								

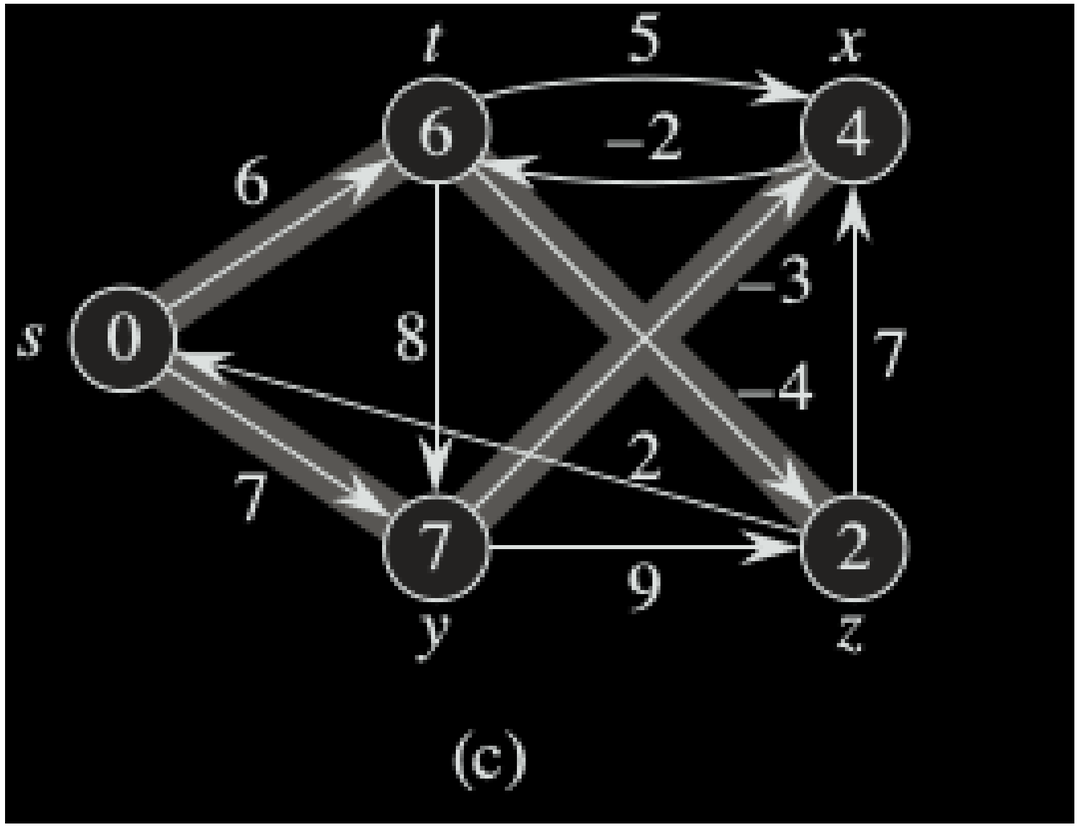
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	6
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	2

(t, x)	(t, y)	(t, z)							

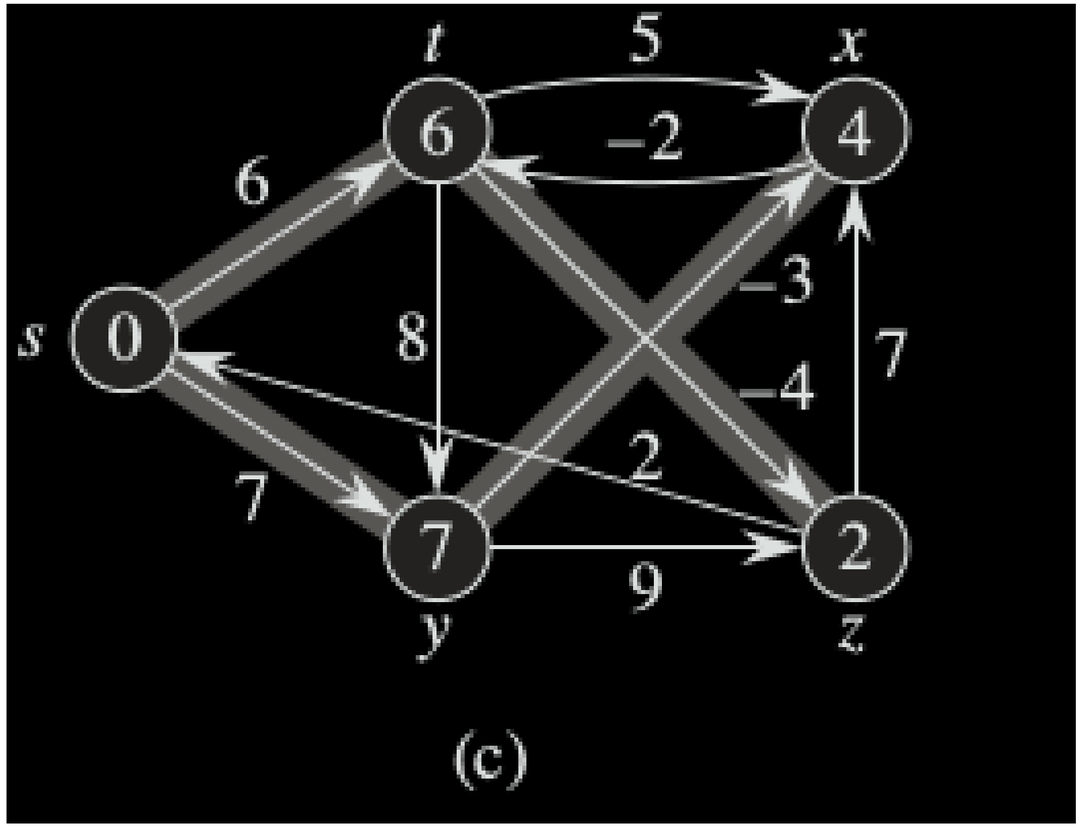
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	2

(t, x)	(t, y)	(t, z)	(x, t)						

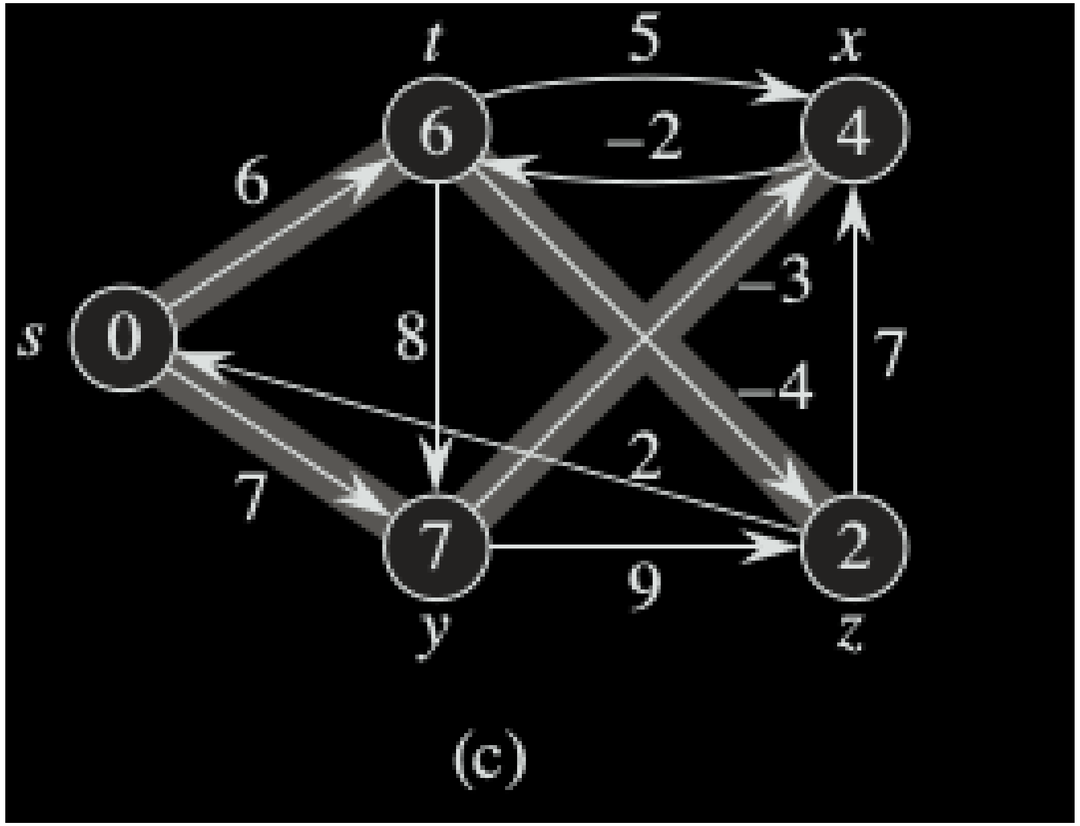
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)					

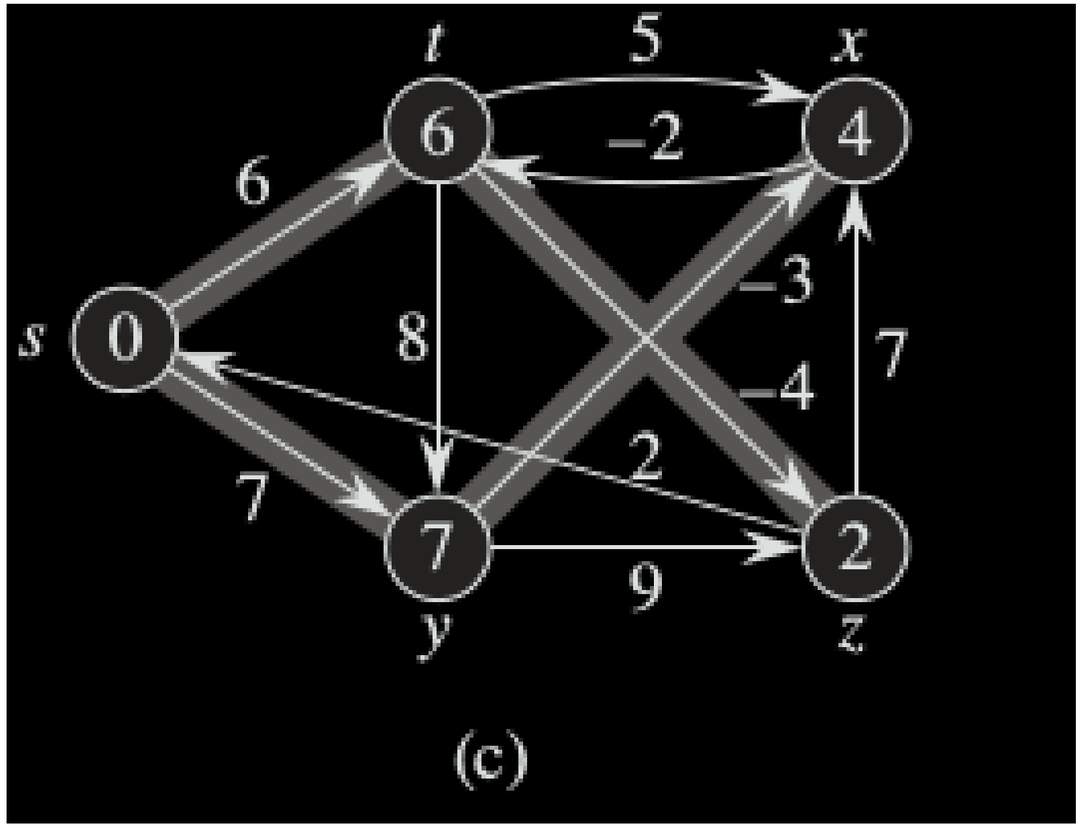
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)				

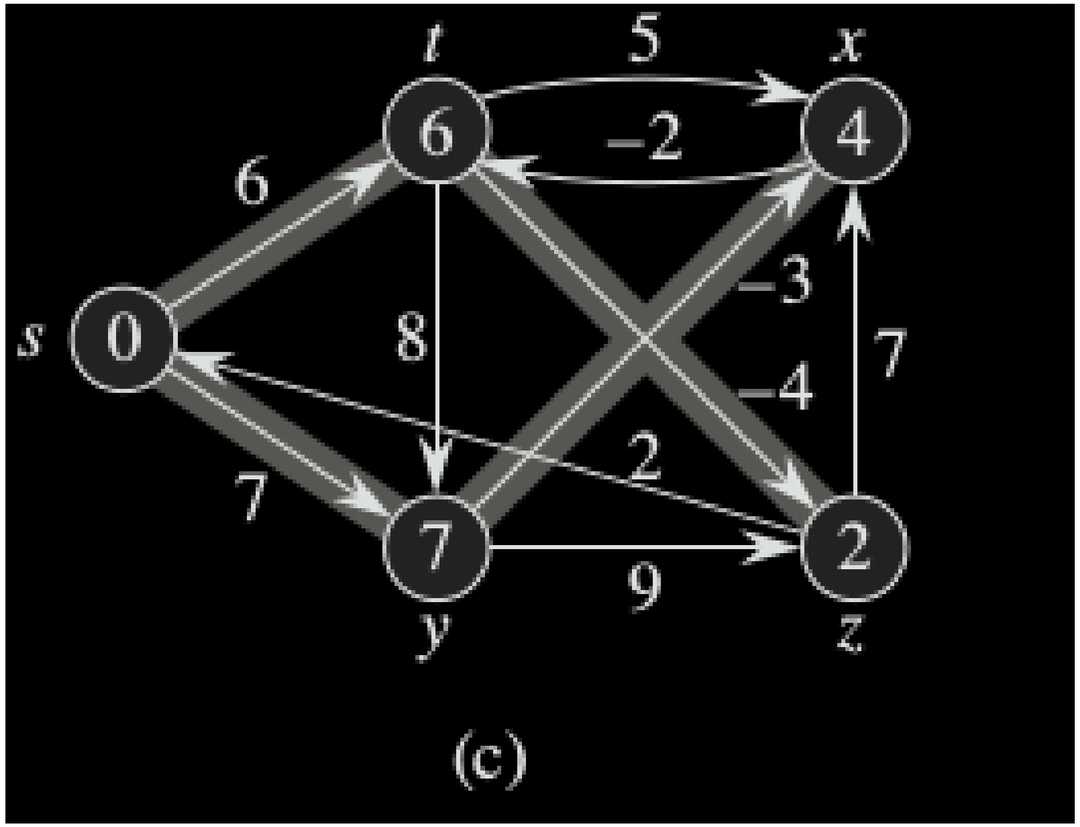
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)			

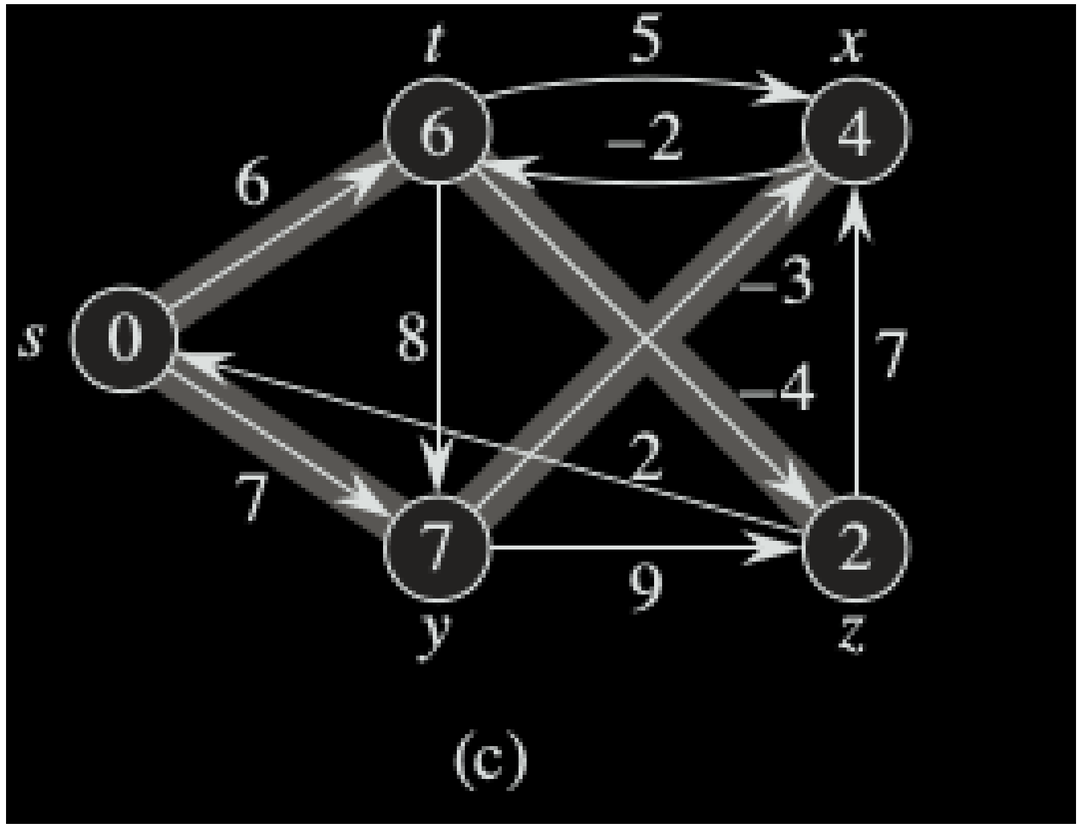
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)		

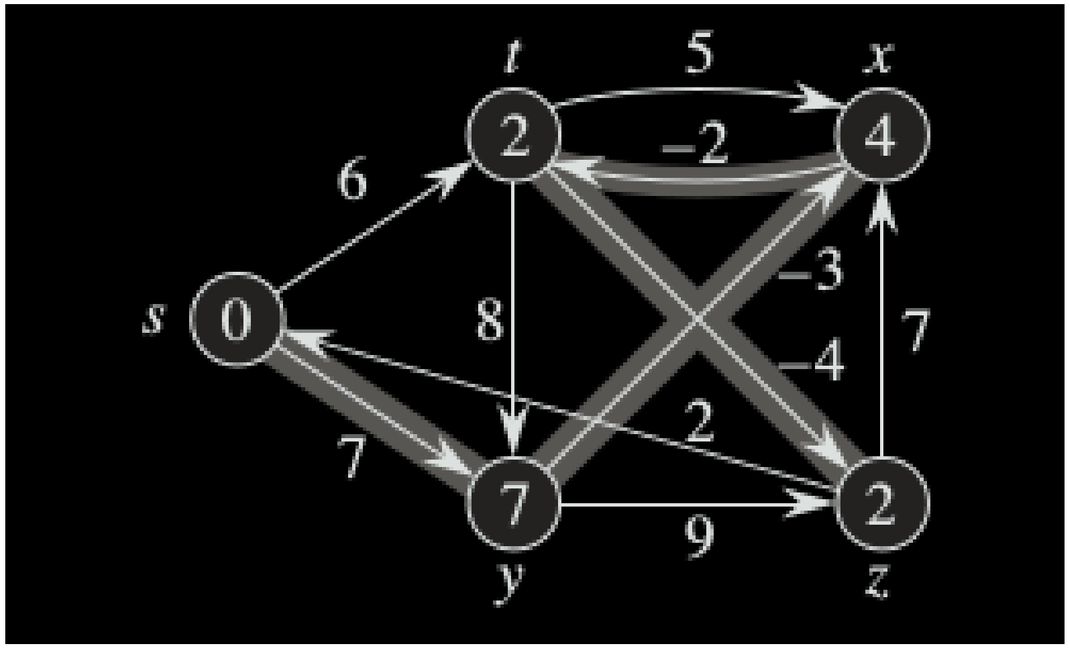
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	

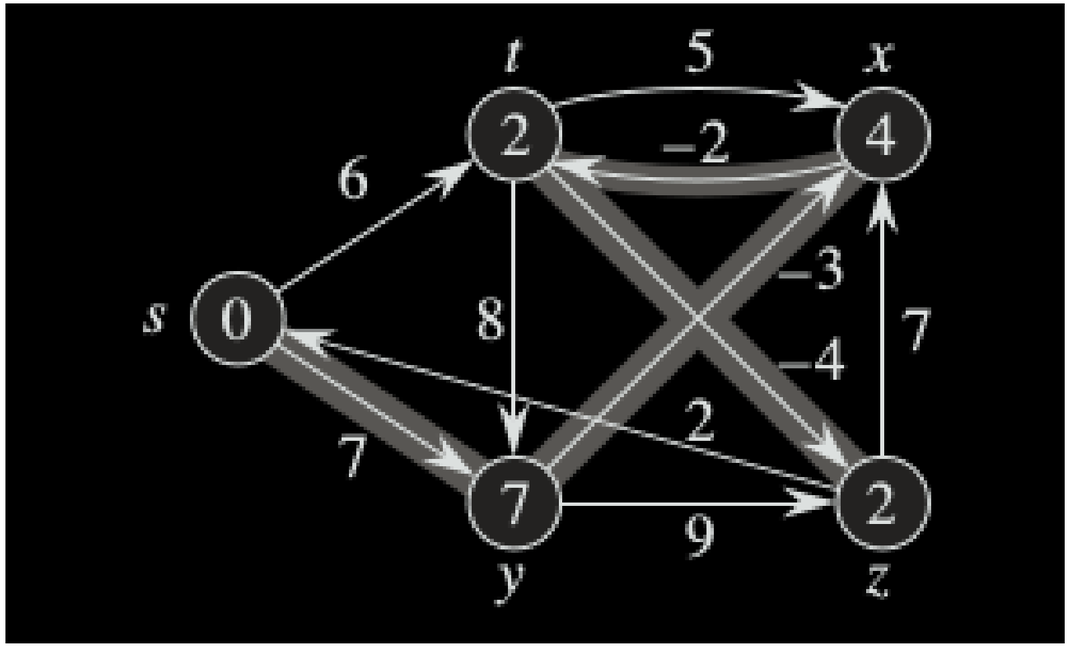
Bellman-Ford algorithm – Iteration 3



Edge Name	Old Cost	Updated Cost
s	0	0
t	6	2
x	4	~~11~~ 4
y	7	~~14~~ 7
z	2	~~16~~ 2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	(s,y)

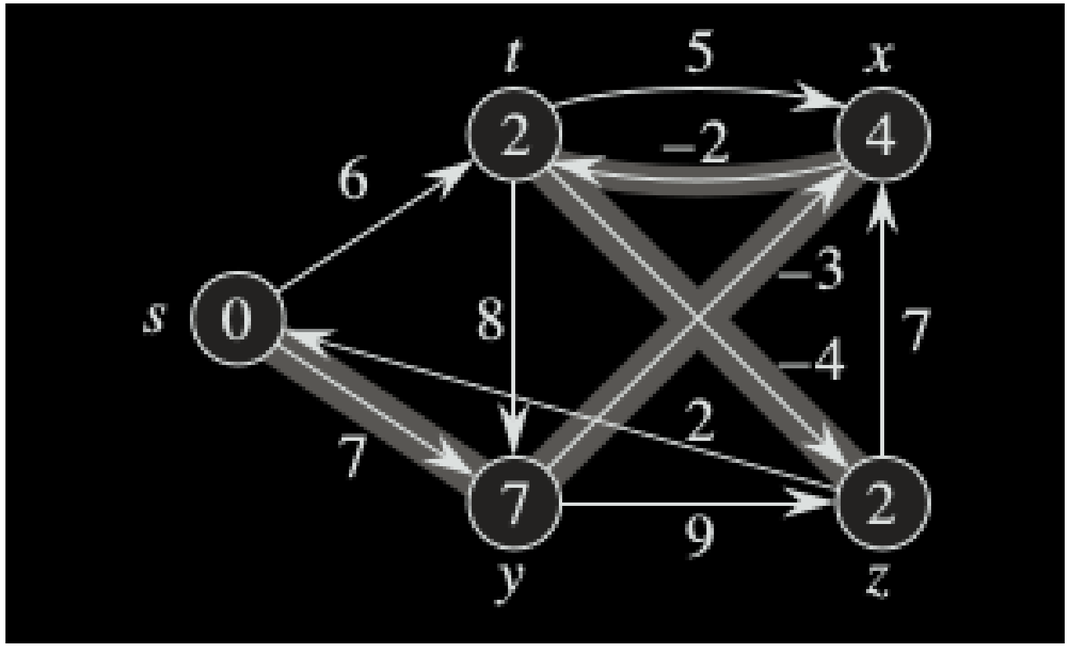
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	
x	4	~~7~~ 4
y	7	
z	2	

(t, x)									

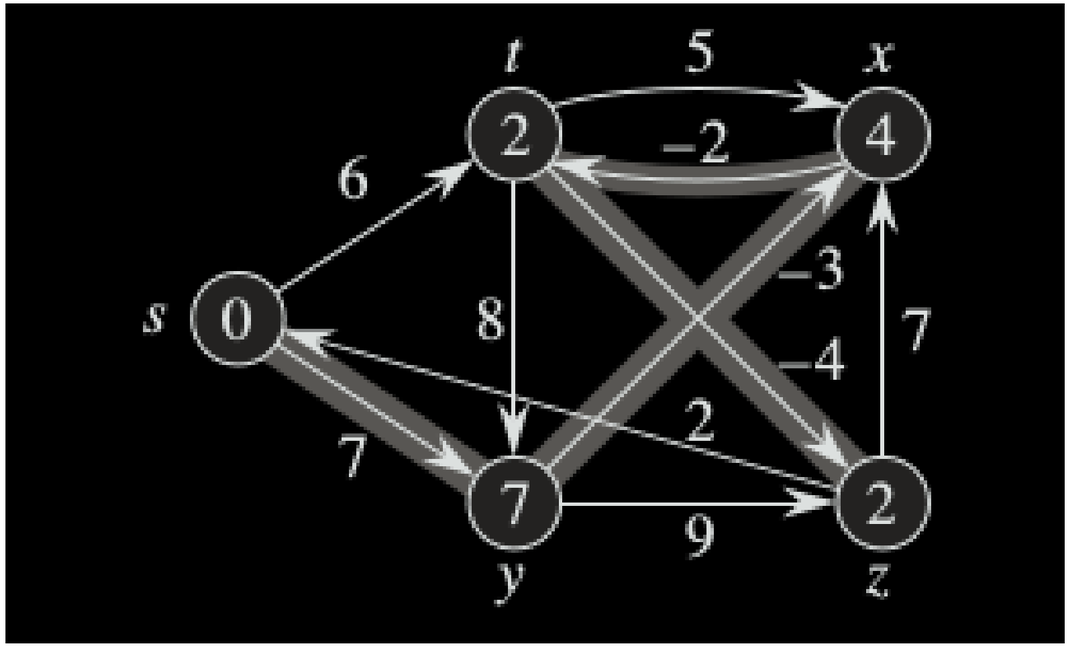
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	

(t, x)	(t, y)								

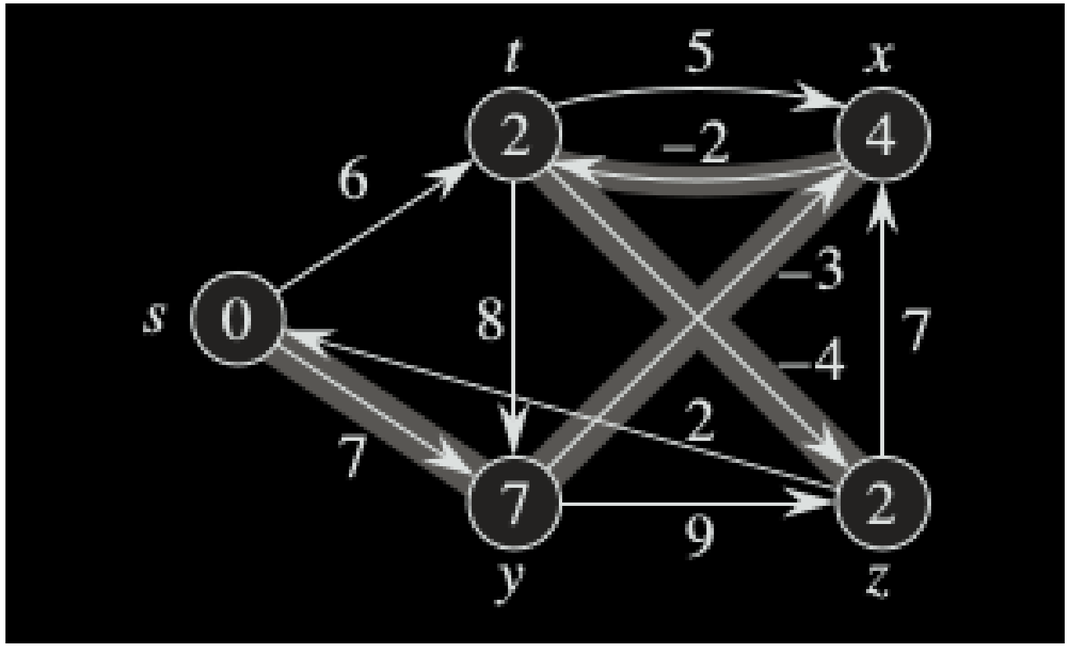
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	-2

(t, x)	(t, y)	(t, z)							

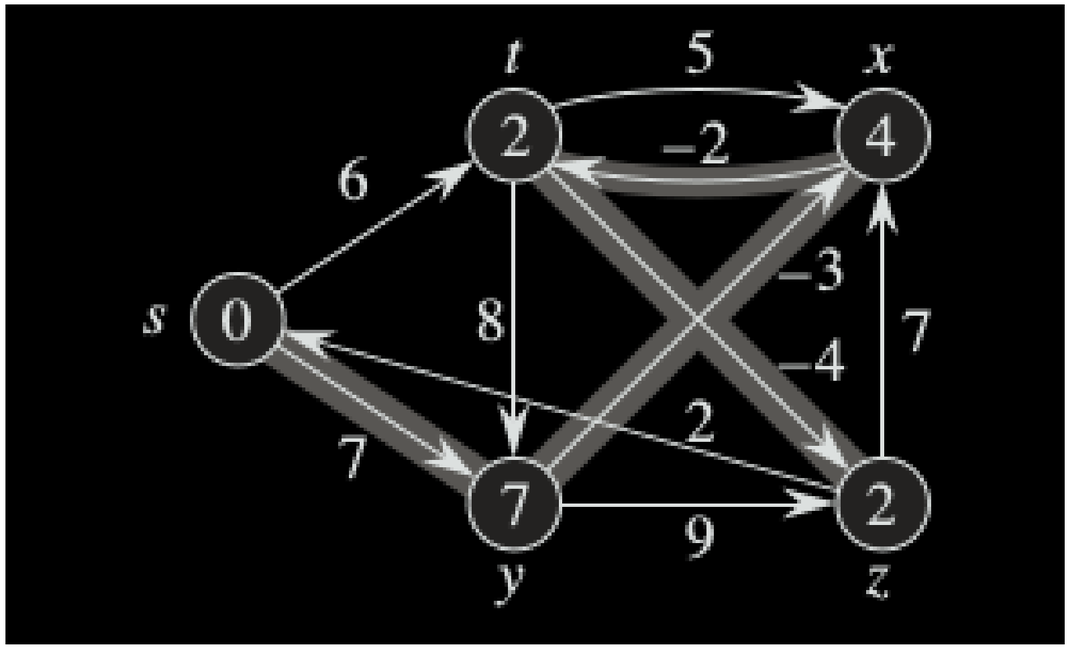
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	-2

(t, x)	(t, y)	(t, z)	(x, t)						

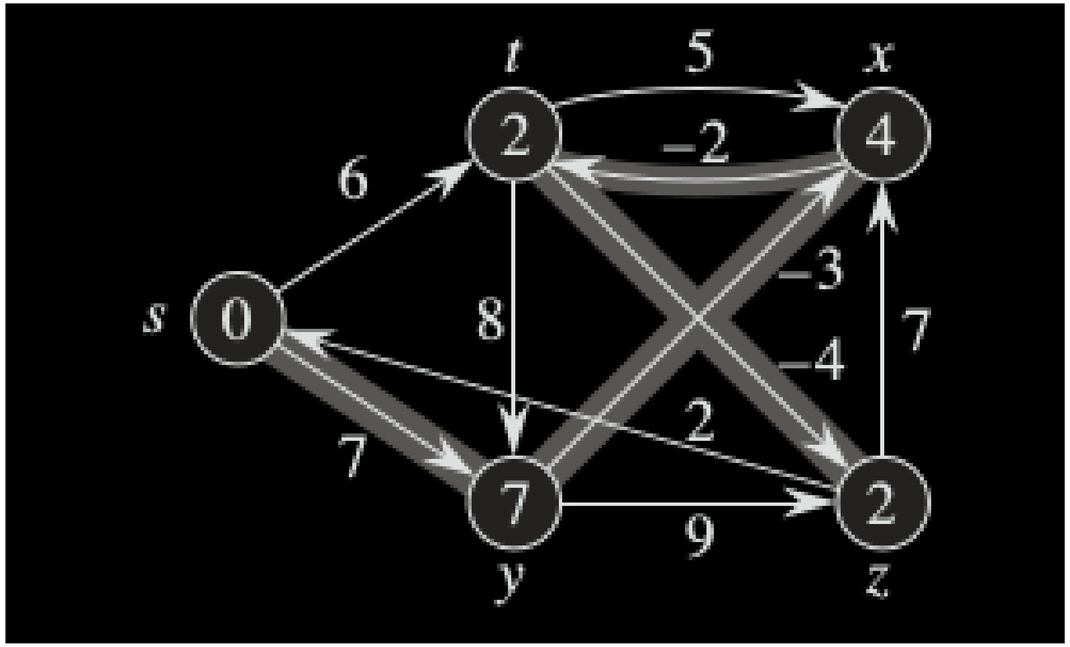
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	-2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)					

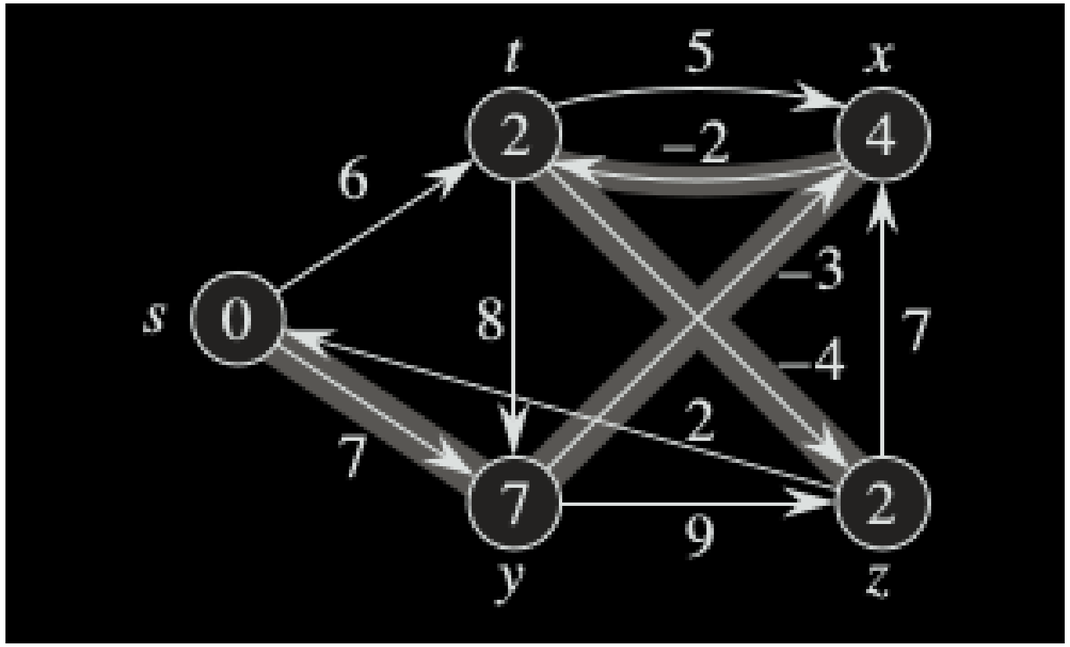
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)				

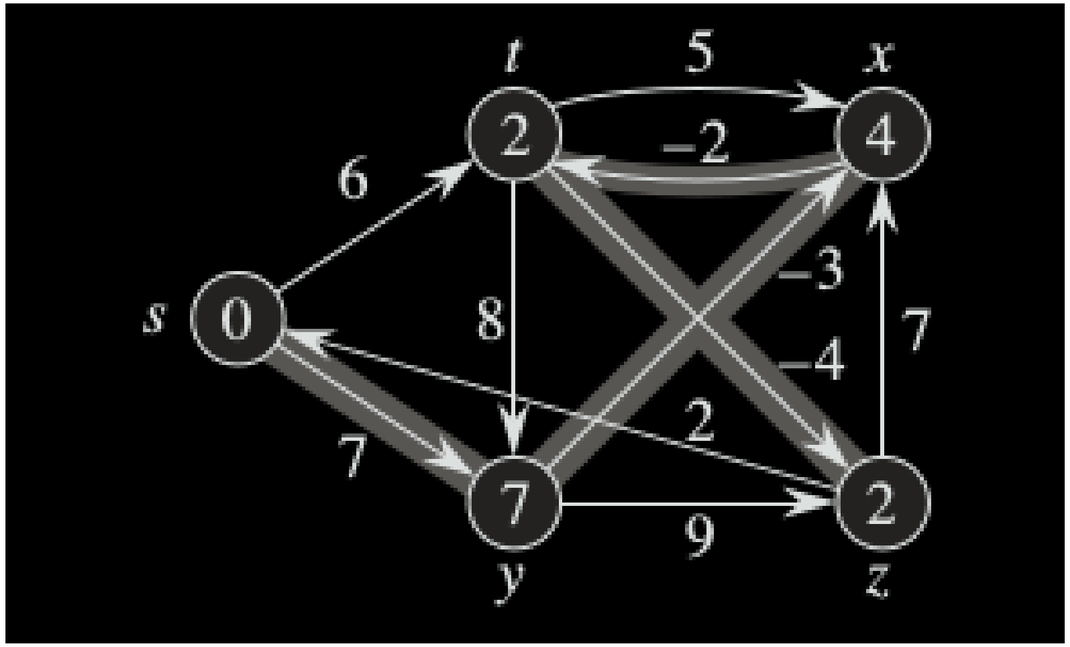
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)			

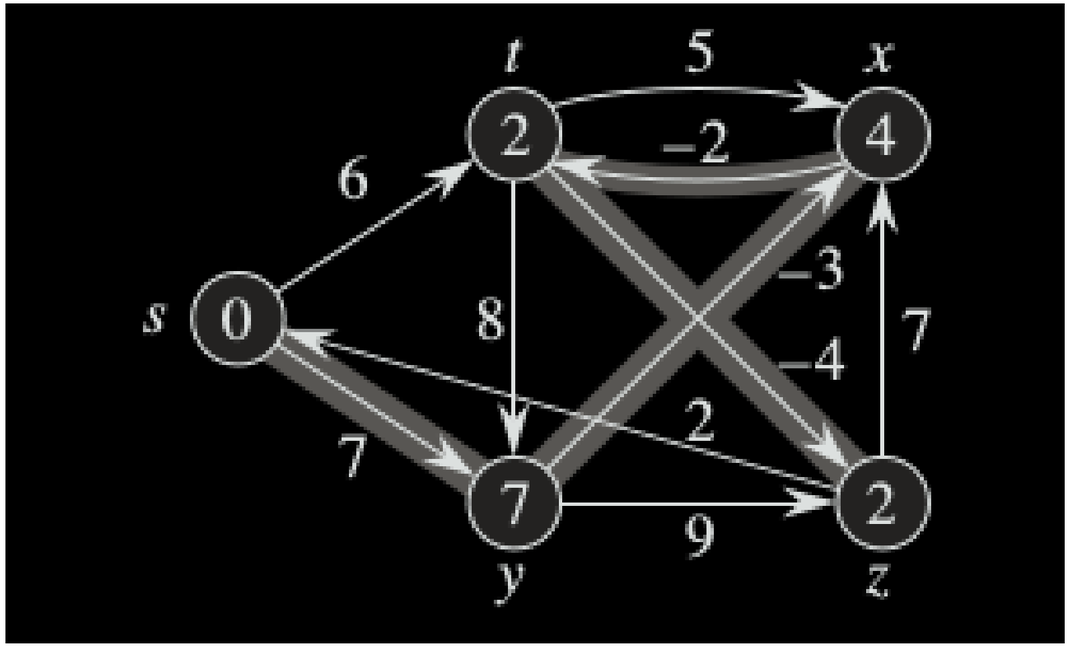
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)		

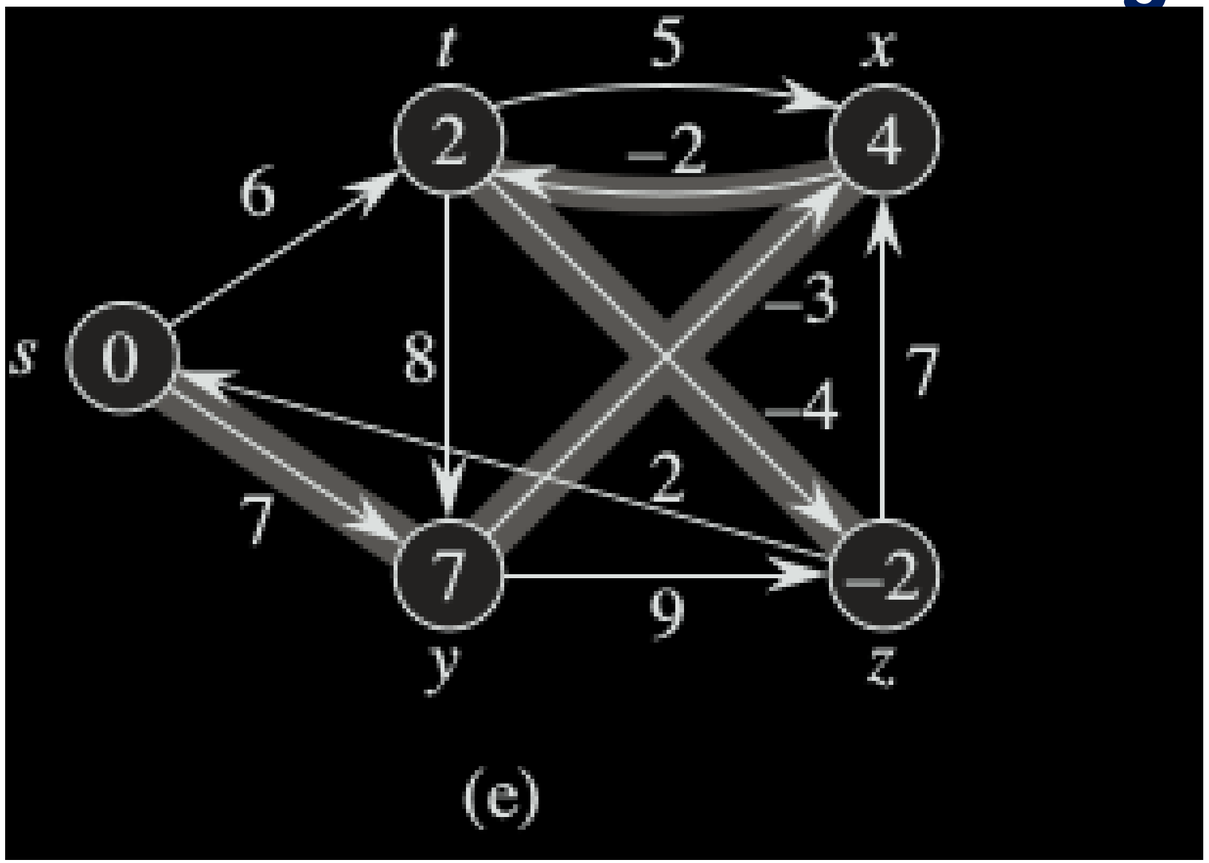
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	

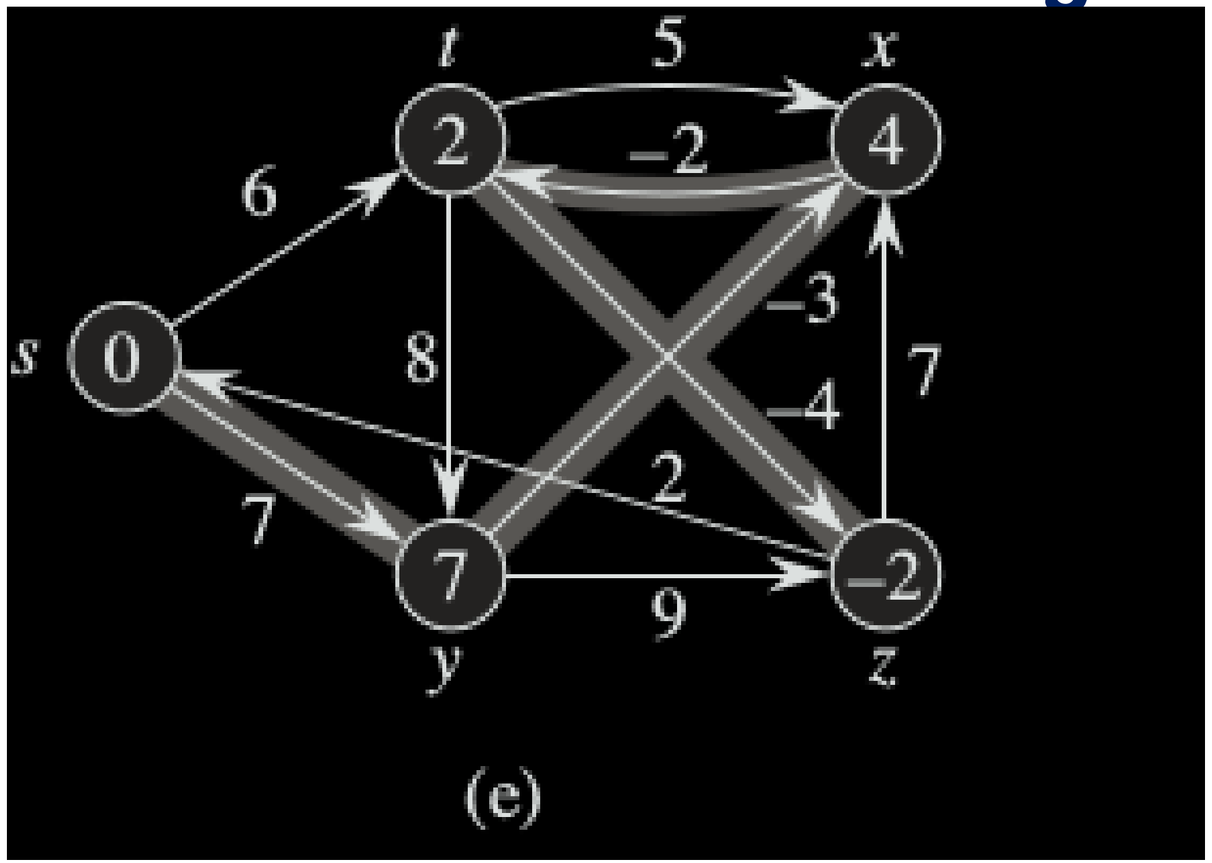
Bellman-Ford algorithm – Iteration 4



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	(s,y)

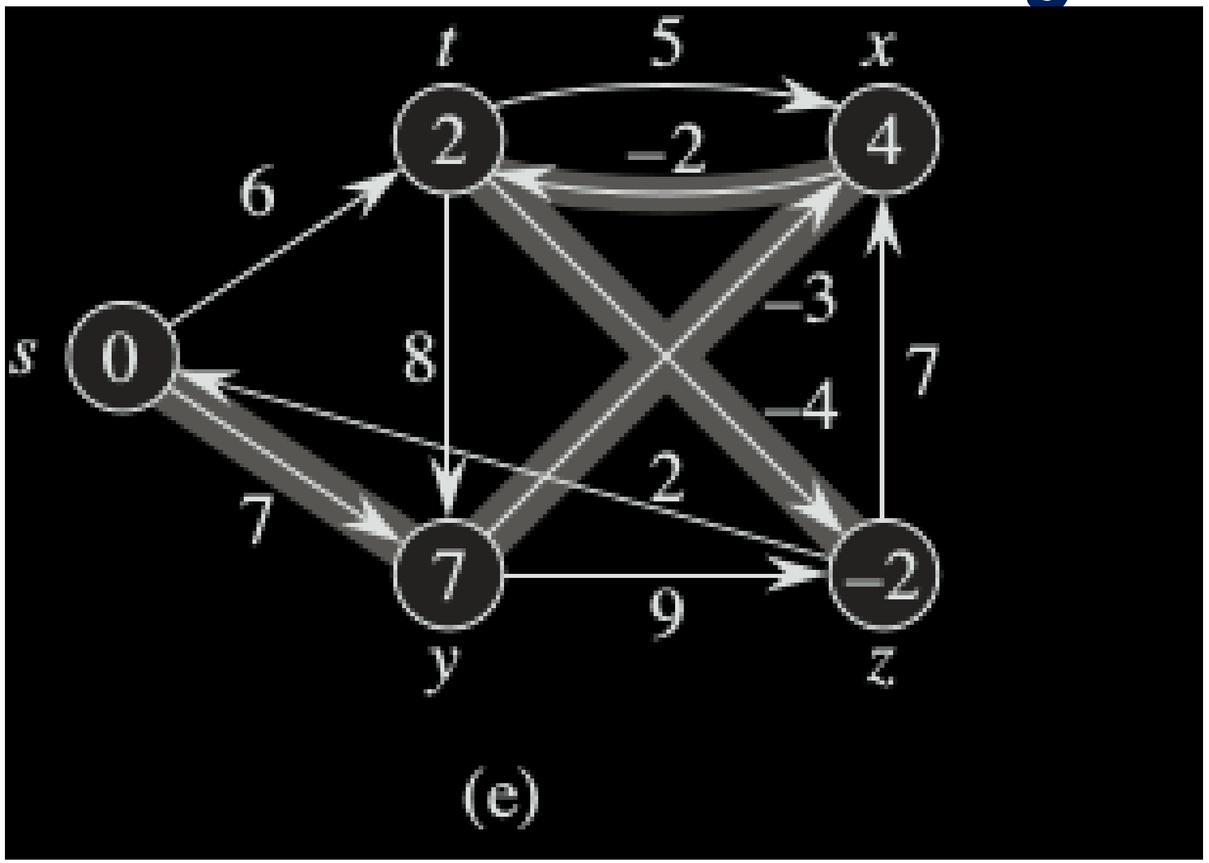
Bellman-Ford algorithm – Check for Cycles



Edge Name	Old Cost	Updated Cost
s	0	
t	2	
x	4	~~7~~ 4
y	7	
z	-2	

(t, x)									

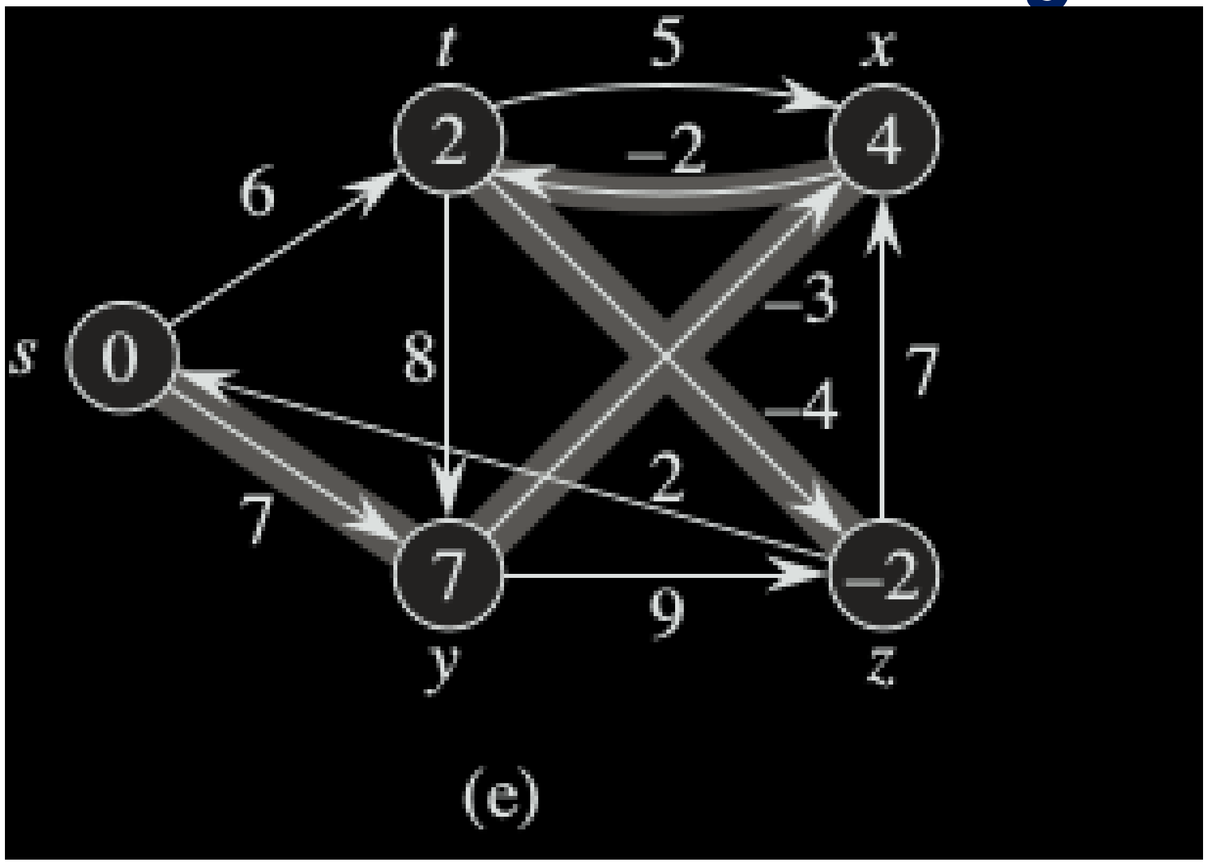
Bellman-Ford algorithm – Check for Cycles



Edge Name	Old Cost	Updated Cost
s	0	
t	2	
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	

(t, x)	(t, y)								

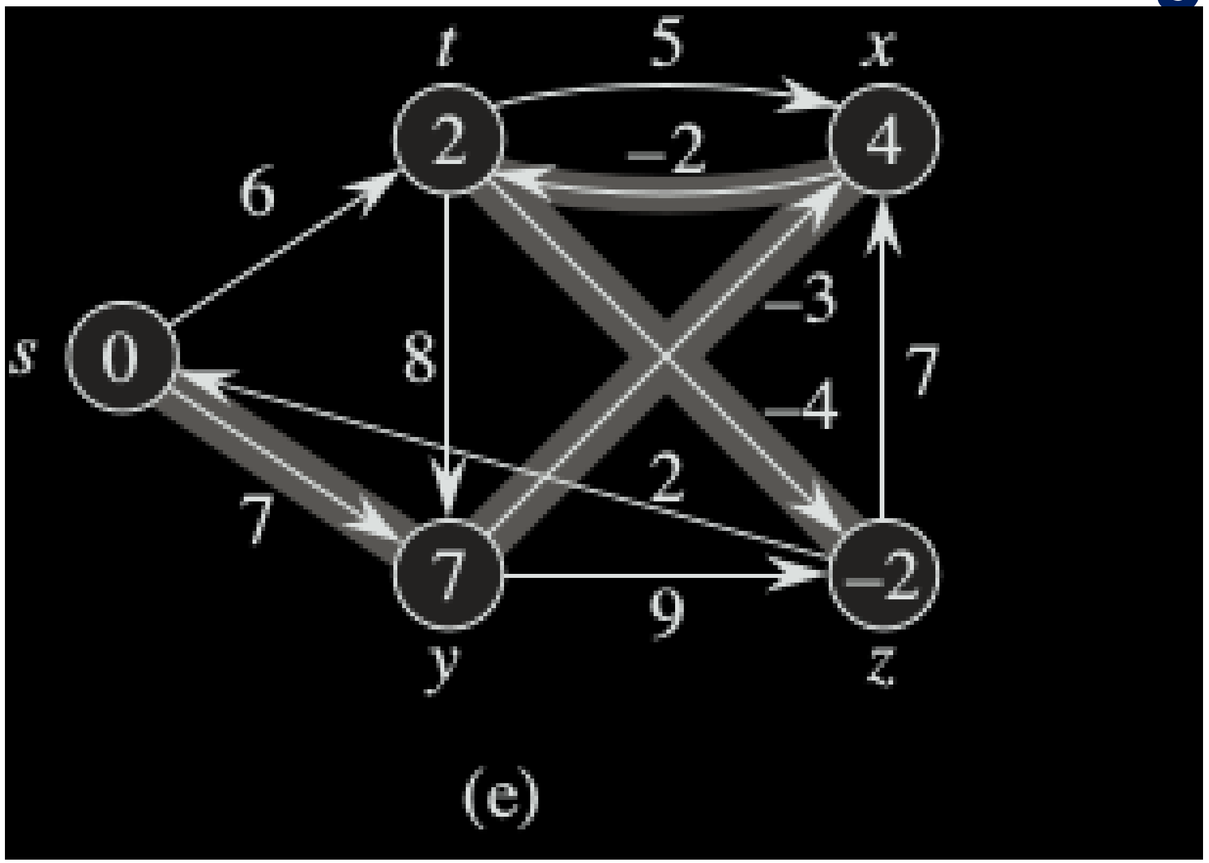
Bellman-Ford algorithm – Check for Cycles



Edge Name	Old Cost	Updated Cost
s	0	
t	2	
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	-2

(t, x)	(t, y)	(t, z)							

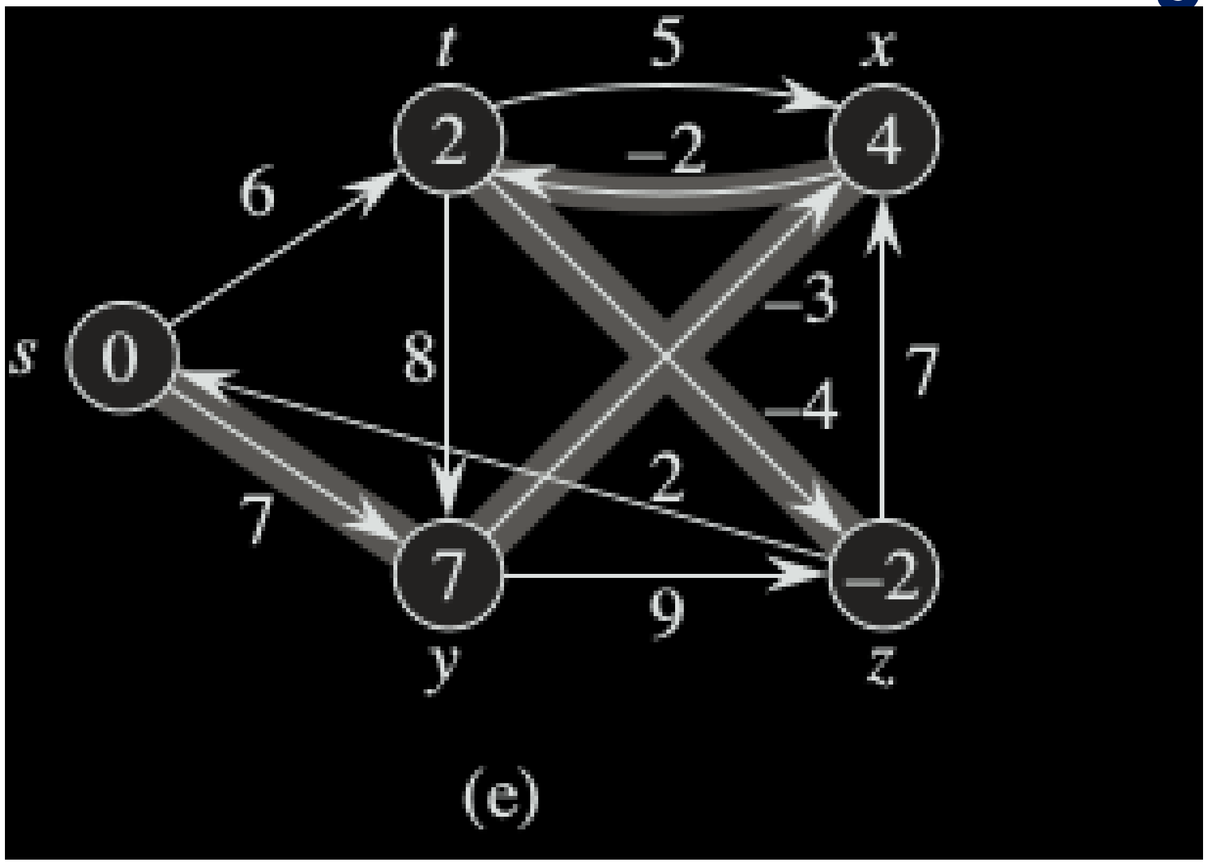
Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	-2

(t, x)	(t, y)	(t, z)	(x, t)						

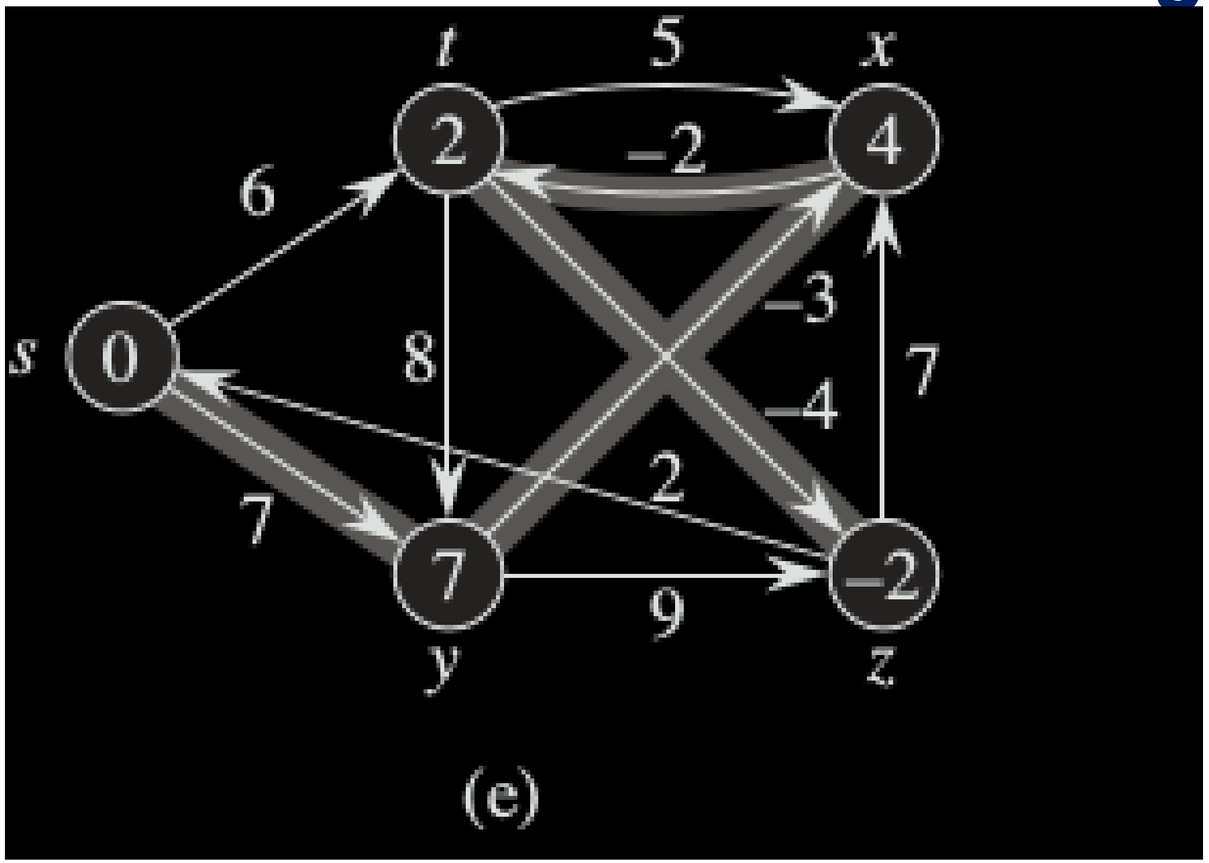
Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	-2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)					

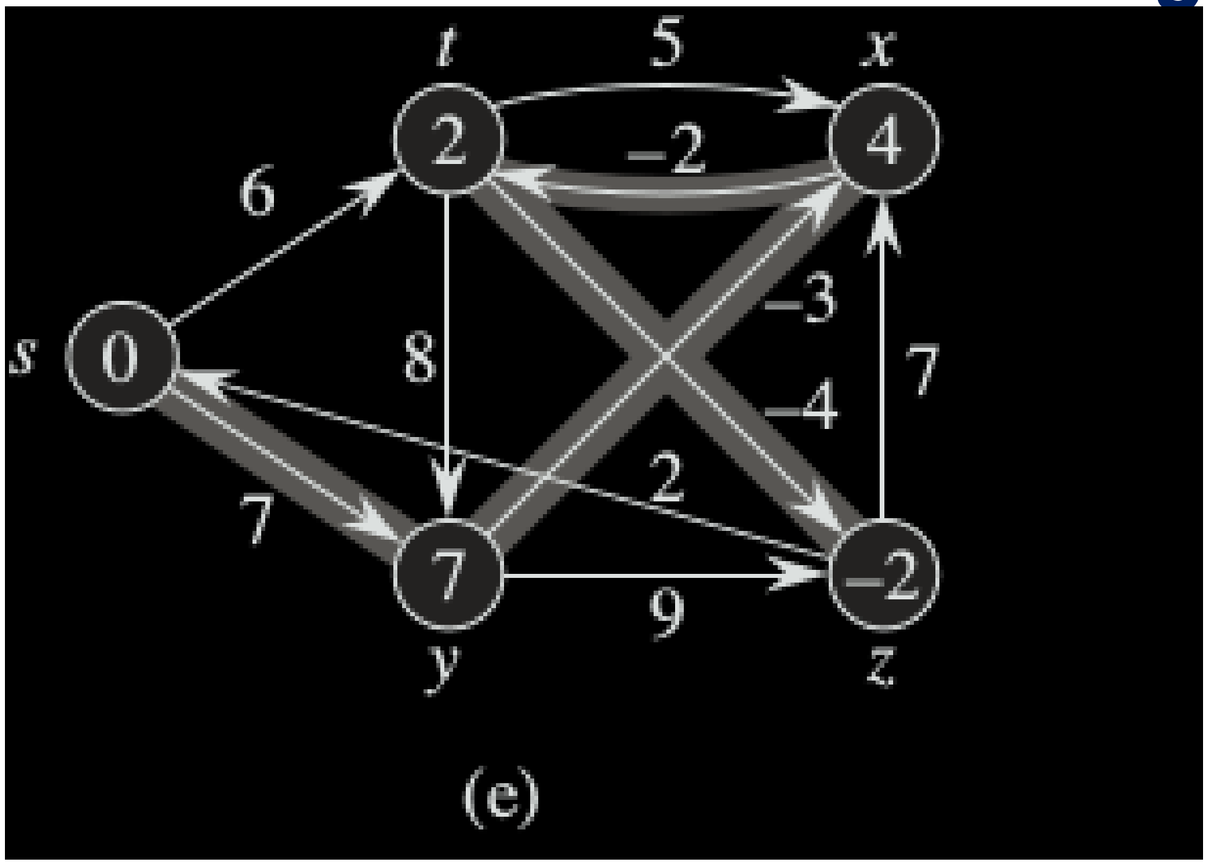
Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	-2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)				

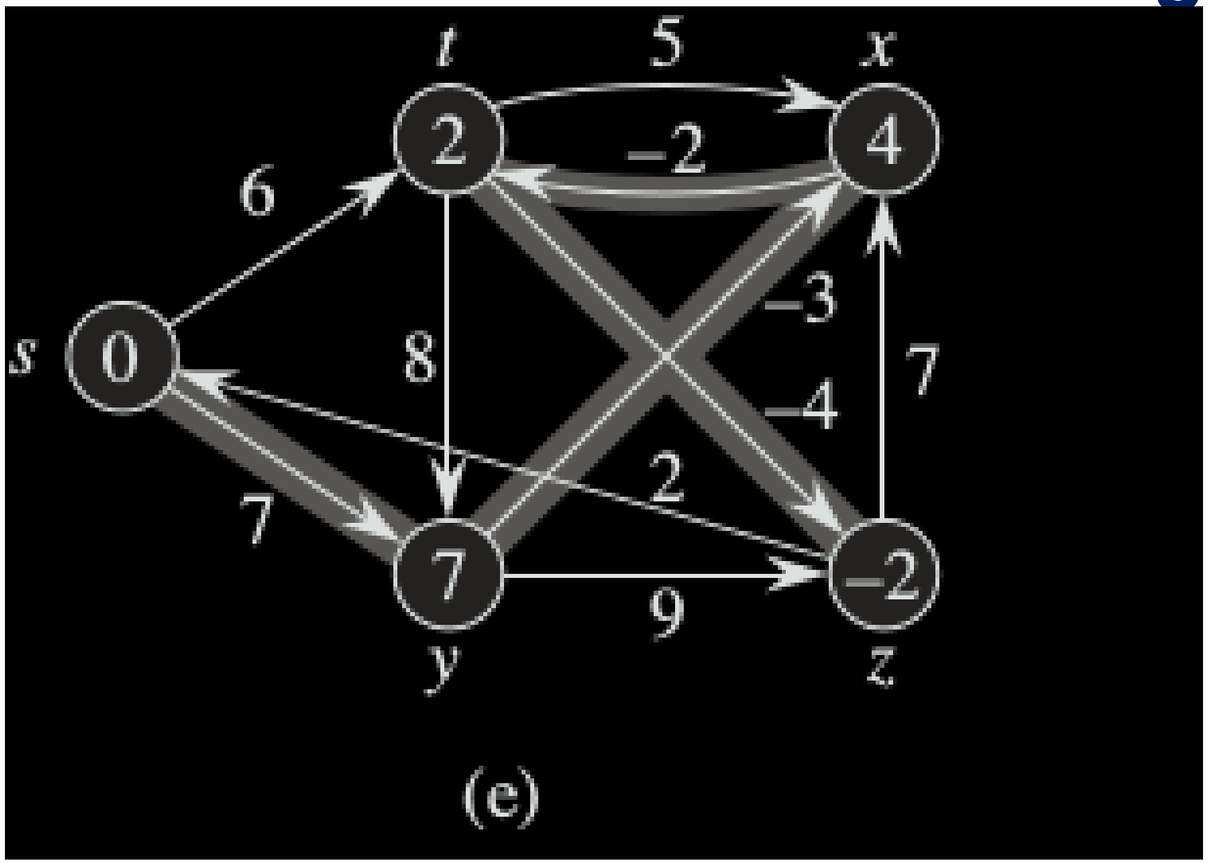
Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)			

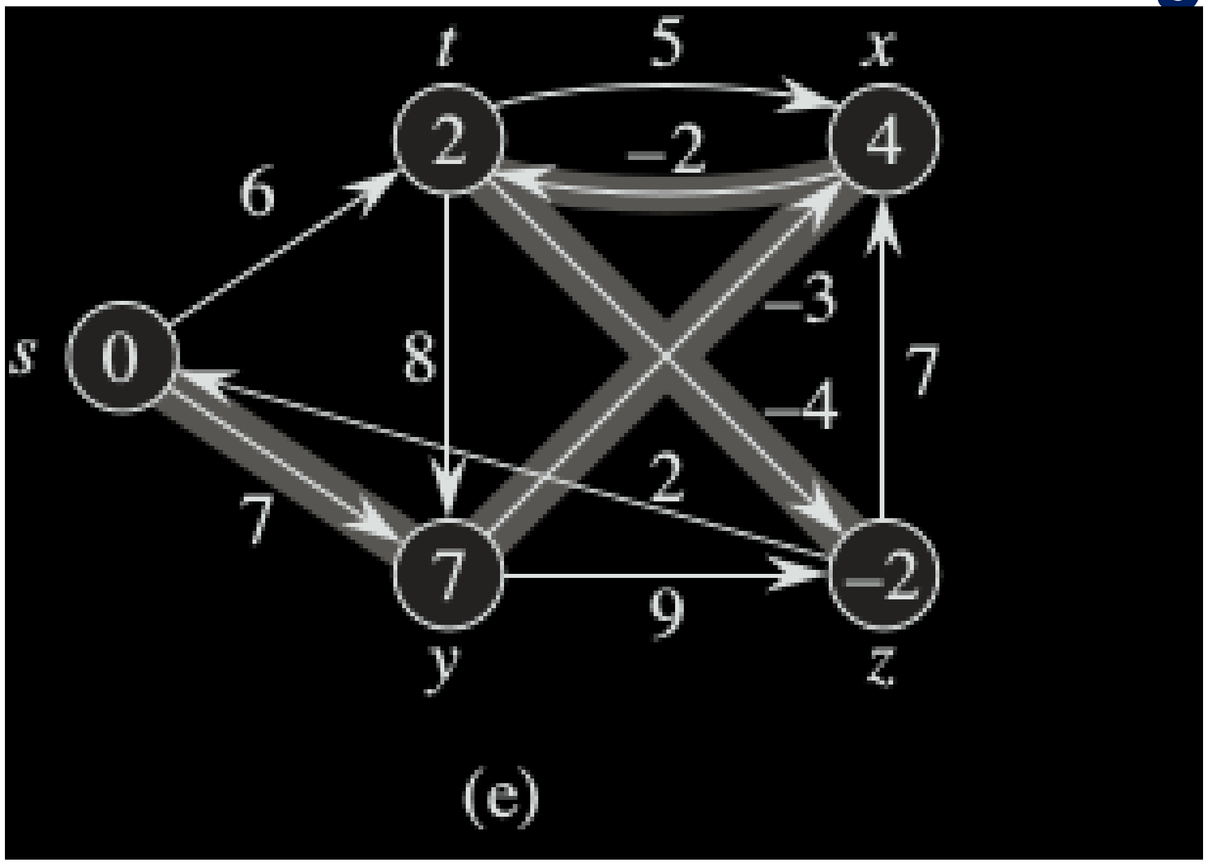
Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)		

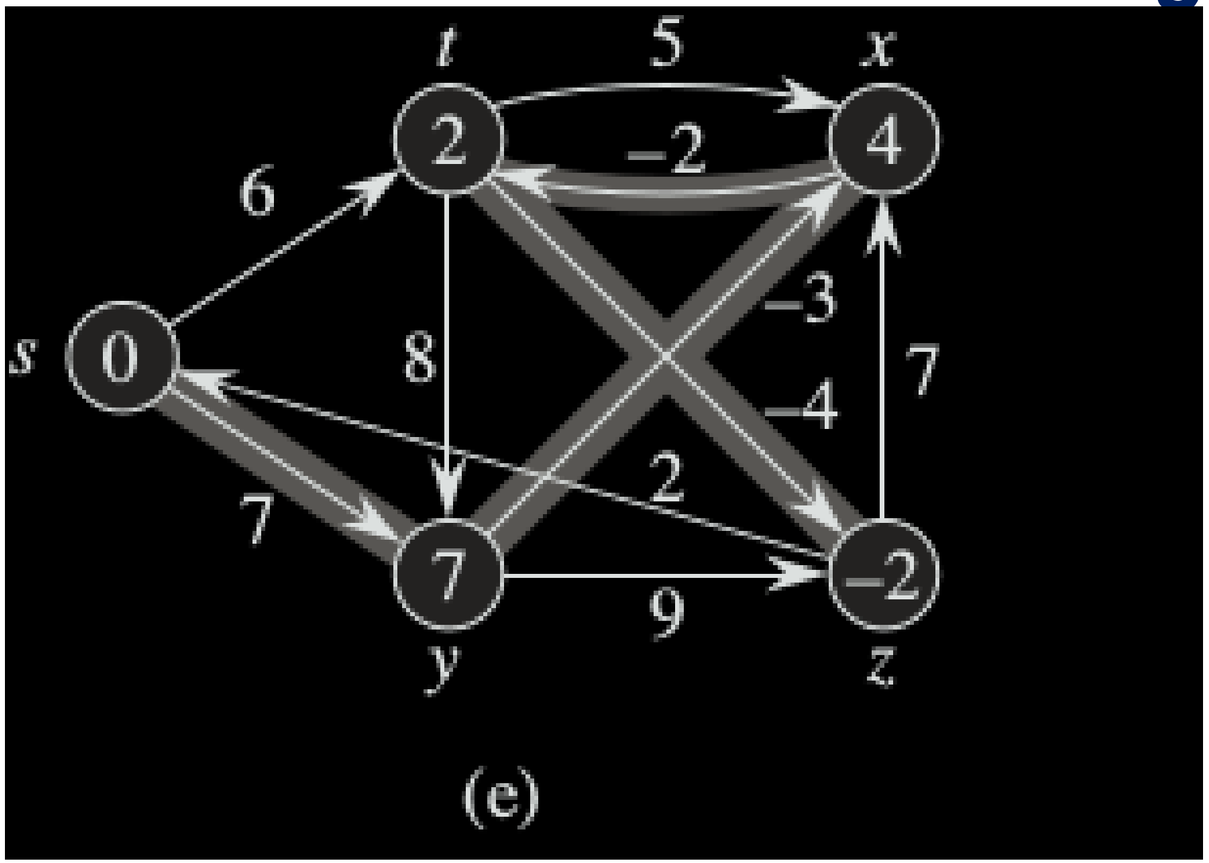
Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	

Bellman-Ford algorithm – Iteration 5



Edge Name	Old Cost	Updated Cost
s	0	0
t	2	2
x	4	~~7~~ 4
y	7	~~10~~ 7
z	-2	~~16~~ -2

(t, x)	(t, y)	(t, z)	(x, t)	(y, x)	(y,z)	(z, x)	(z,s)	(s,t)	(s,y)

Algorithm

```
procedure BellmanFord(Graph, source):

    // Step 1: Initialization
    for each vertex v in Graph:
        distance[v] = ∞
        predecessor[v] = null
    distance[source] = 0

    // Step 2: Relax edges repeatedly (|V| - 1 times)
    for i = 1 to |V| - 1:
        for each edge (u, v) with weight w in Graph:
            if distance[u] + w < distance[v]:
                distance[v] = distance[u] + w
                predecessor[v] = u

    // Step 3: Check for negative weight cycles
    for each edge (u, v) with weight w in Graph:
        if distance[u] + w < distance[v]:
            print("Graph contains a negative weight cycle")
            return

    return distance[], predecessor[]
```

TIME COMPLEXITY

$$O(V \cdot E)$$

TIME COMPLEXITY

$$O(V \cdot E)$$

SPACE COMPLEXITY

$$O(V) + O(V + E)$$

Store Previous nodes information which depends on no. of nodes

Also Stores graph which is proportional to no. of nodes and edges. So, ?