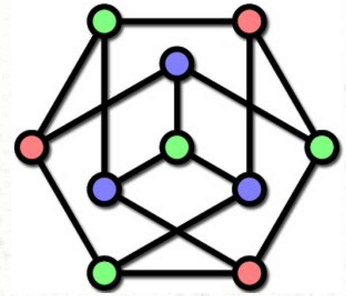


APPLIED GRAPH THEORY AND ALGORITHMS (CSC4066)

Bellman Ford Algorithm

Dr. Hasin A Ahmed
Assistant Professor
Department of Computer Science
Gauhati University

Complexity of Floyd Warshall Algorithm

- Time complexity of Floyd Warshall algorithm is $O(n^3)$
- Floyd Warshall Algorithm is best suited for dense graphs
- This is because its complexity depends only on the number of vertices in the given graph
- For sparse graphs, Johnson's Algorithm is more suitable

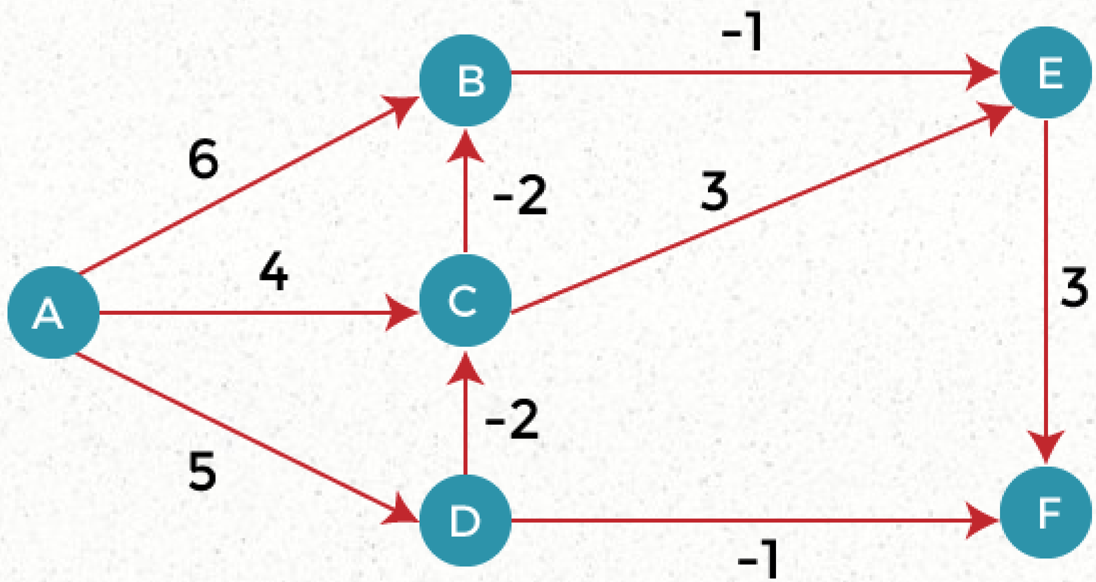
Bellman Ford Algorithm

- It computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph
- It is slower than Dijkstra's algorithm, but more versatile, as it is capable of handling negative weight edge
- Bellman–Ford algorithm can detect and report the negative cycle

Bellman Ford Algorithm

- Rule:
  - We will go on relaxing all the edges (n - 1) times where,
  - n = number of vertices

Bellman Ford Algorithm

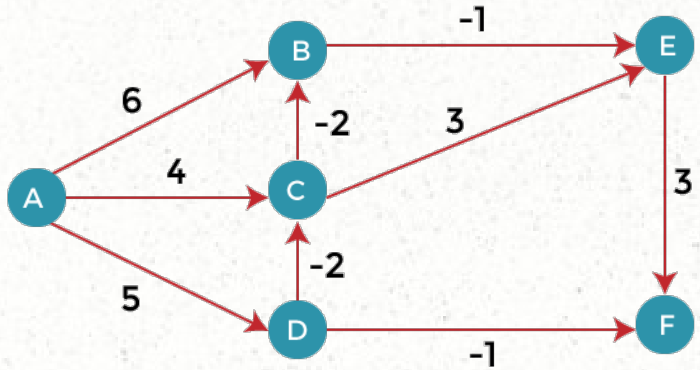


The above graph contains 6 vertices so we will relax all the edges 5 times

Bellman Ford Algorithm

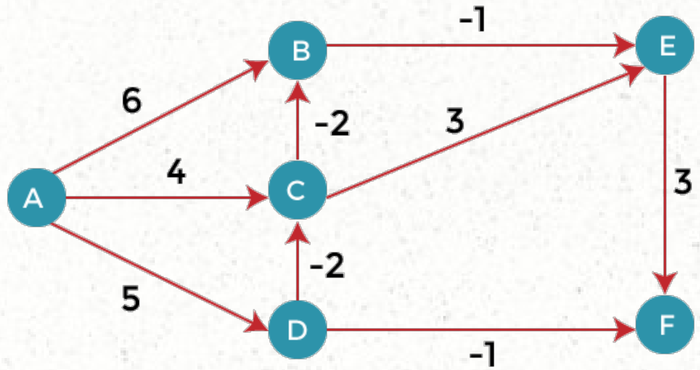
- Relaxing means:
  - If $(d(u) + c(u , v) < d(v))$
  - $d(v) = d(u) + c(u , v)$

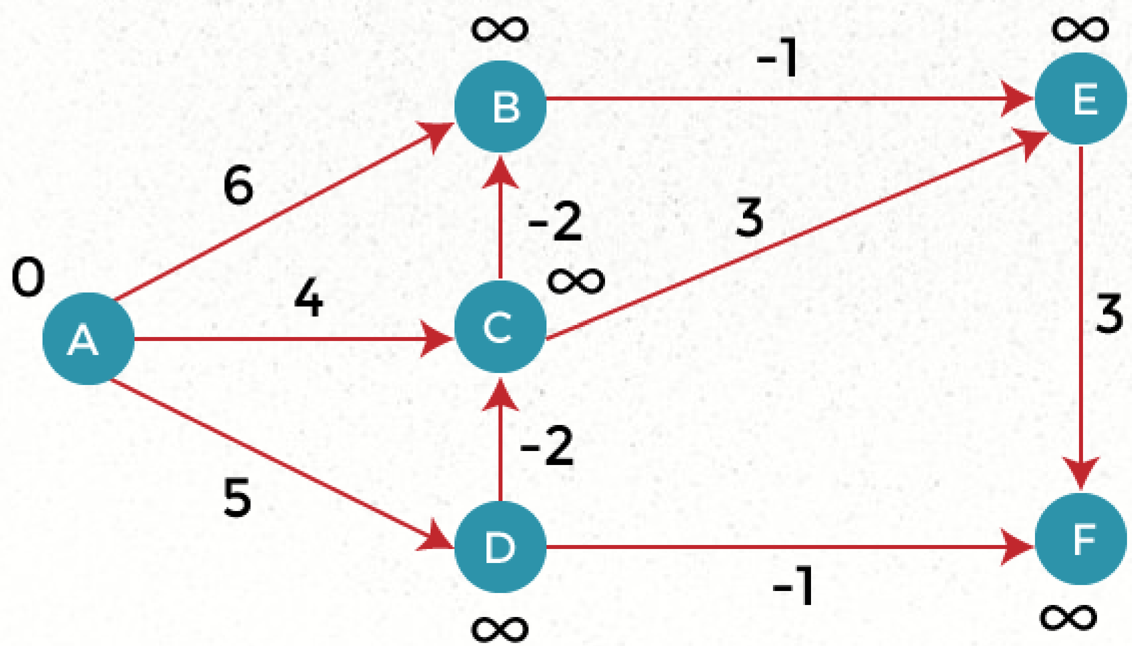
Bellman Ford Algorithm



(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)

Bellman Ford Algorithm

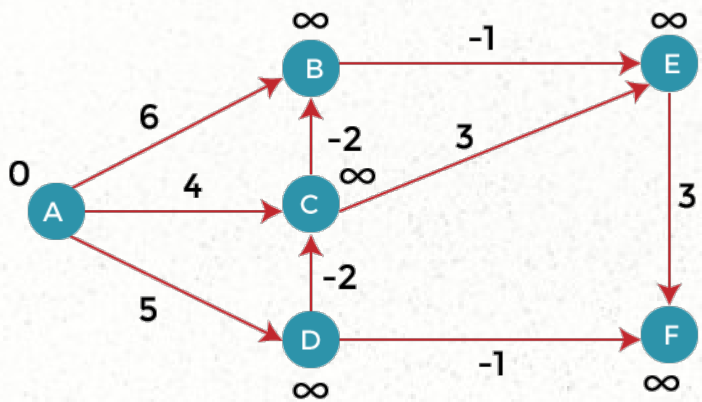


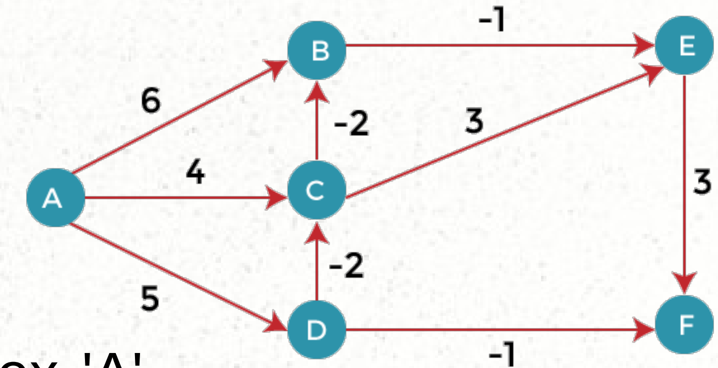


(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)

Iteration 1

Bellman Ford Algorithm





Consider the edge (A, B). Denote vertex 'A' as 'u' and vertex 'B' as 'v'. Now use the relaxing formula

$d(u) = 0$

$d(v) = \infty$

$c(u , v) = 6$

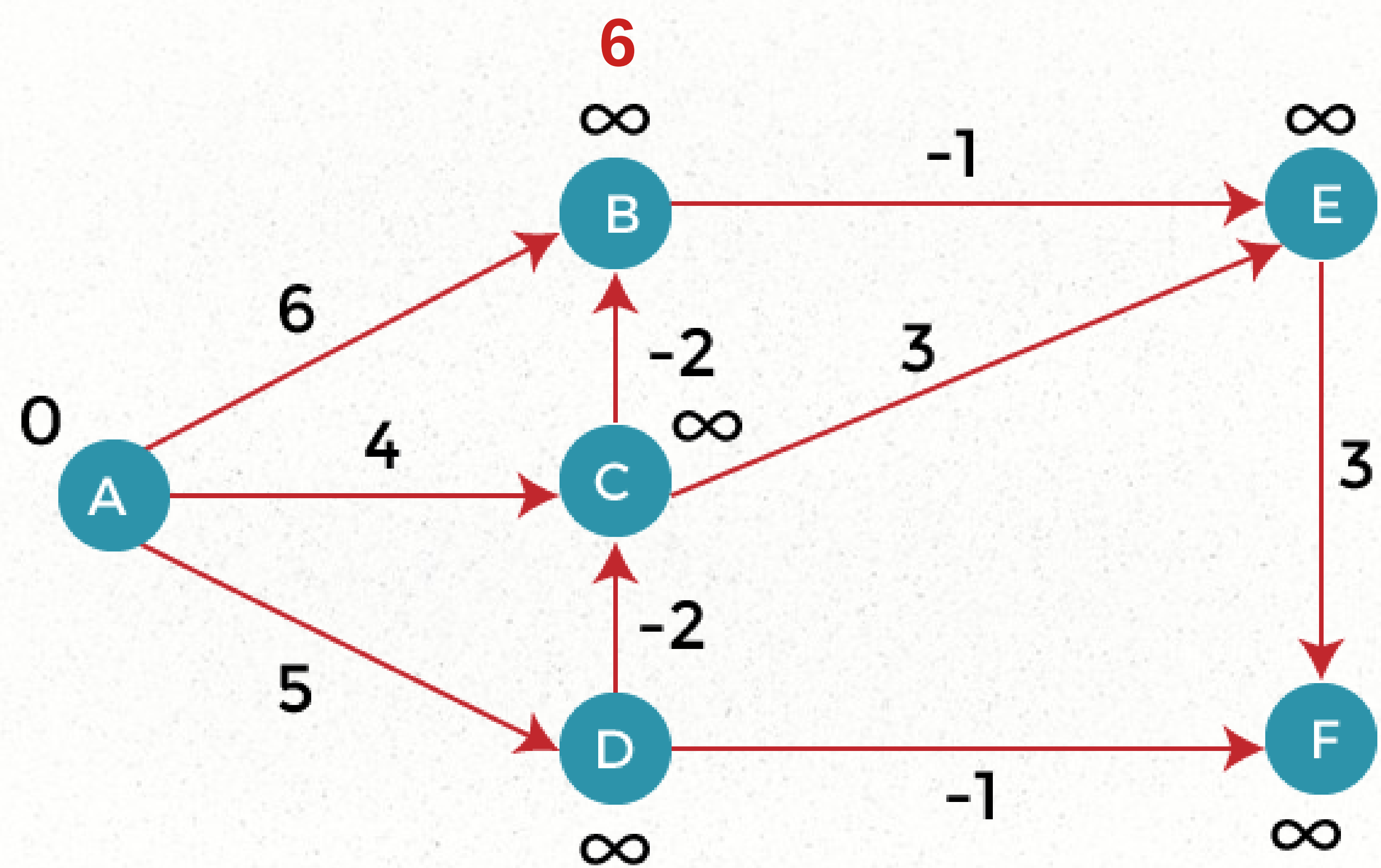
Since $(0 + 6)$ is less than ∞, so update

$d(v) = d(u) + c(u , v)$

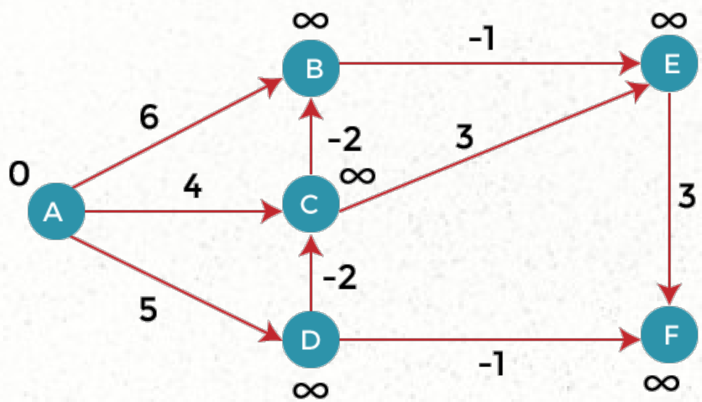
$d(v) = 0 + 6 = 6$

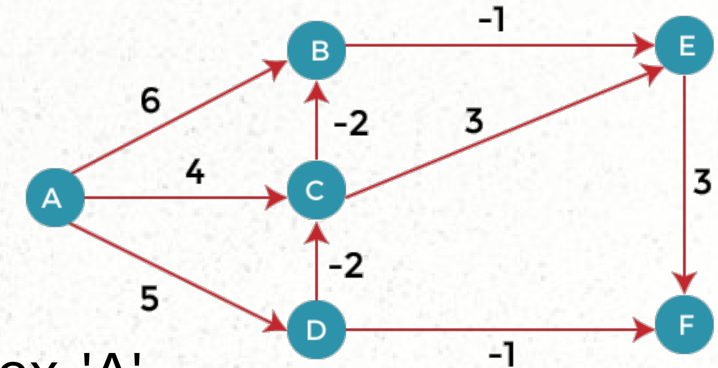
Therefore, the distance of vertex B is 6

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)


6
∞
-1
∞
B
E
6
-2
3
0
∞
4
A
C
3
-2
5
D
F
-1
∞
∞

Bellman Ford Algorithm





Consider the edge (A, C). Denote vertex 'A' as 'u' and vertex 'C' as 'v'. Now use the relaxing formula:

d(u) = 0

d(v) = ∞

c(u , v) = 4

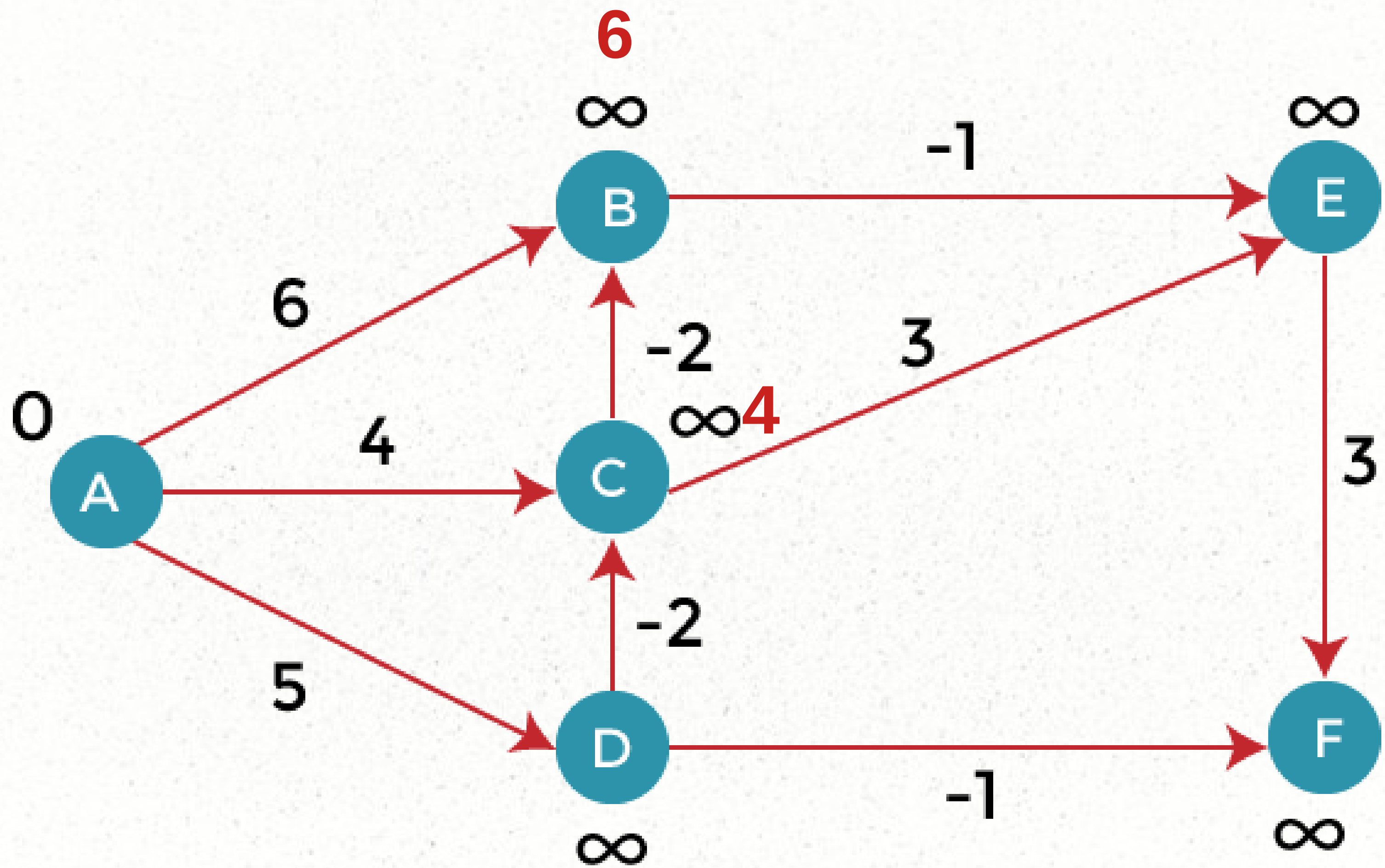
Since (0 + 4) is less than ∞, so update

d(v) = d(u) + c(u , v)

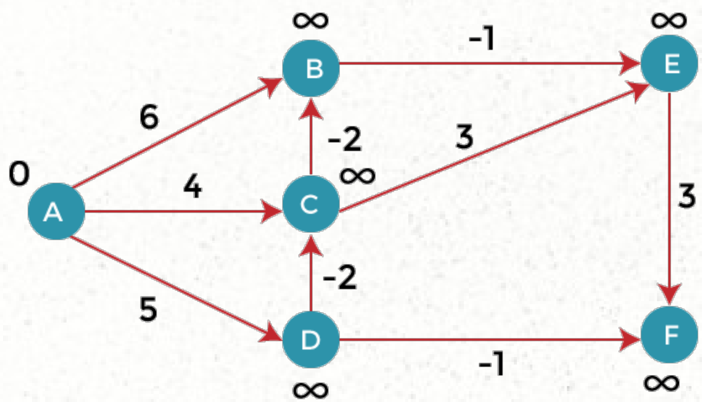
d(v) = 0 + 4 = 4

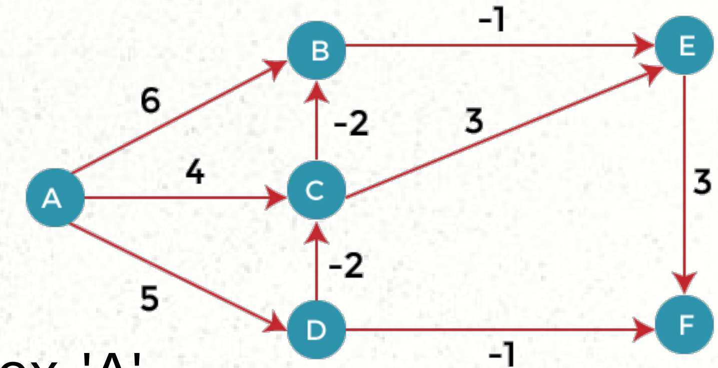
Therefore, the distance of vertex C is 4.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)


6
∞
-1
∞
B
E
6
-2
3
0
4
∞4
3
A
C
-2
5
D
F
-1
∞
∞

Bellman Ford Algorithm





Consider the edge (A, D). Denote vertex 'A' as 'u' and vertex 'D' as 'v'. Now use the relaxing formula:

d(u) = 0

d(v) = ∞

c(u , v) = 5

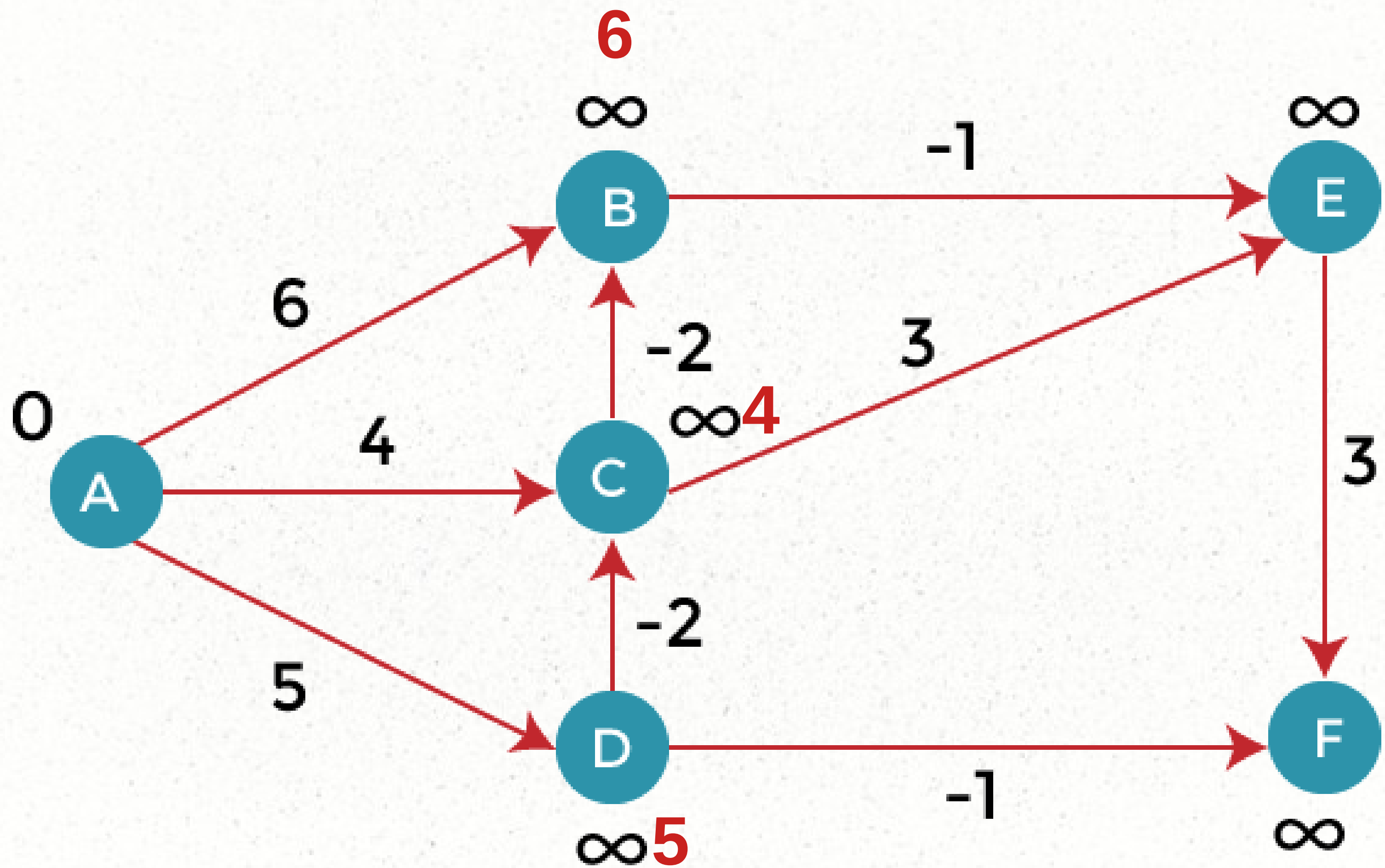
Since (0 + 5) is less than ∞, so update

d(v) = d(u) + c(u , v)

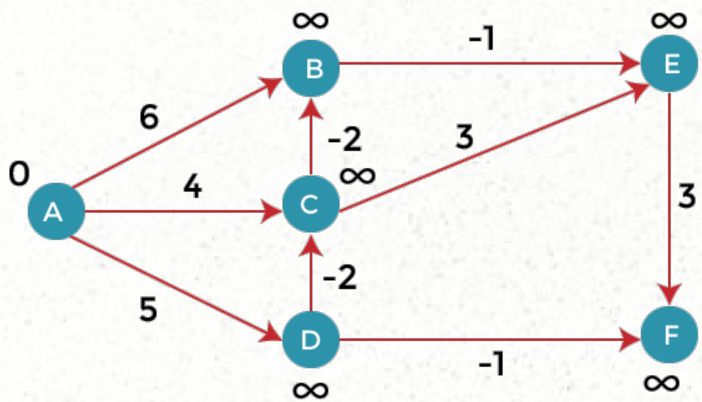
d(v) = 0 + 5 = 5

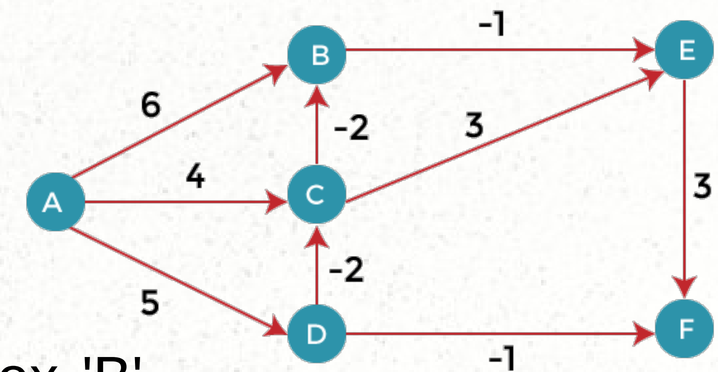
Therefore, the distance of vertex D is 5.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)


6
∞
∞
-1
B
E
6
-2
3
0
∞4
4
A
C
3
-2
5
D
F
-1
∞5
∞

Bellman Ford Algorithm





Consider the edge (B, E). Denote vertex 'B' as 'u' and vertex 'E' as 'v'. Now use the relaxing formula:

$d(u) = 6$

$d(v) = \infty$

$c(u , v) = -1$

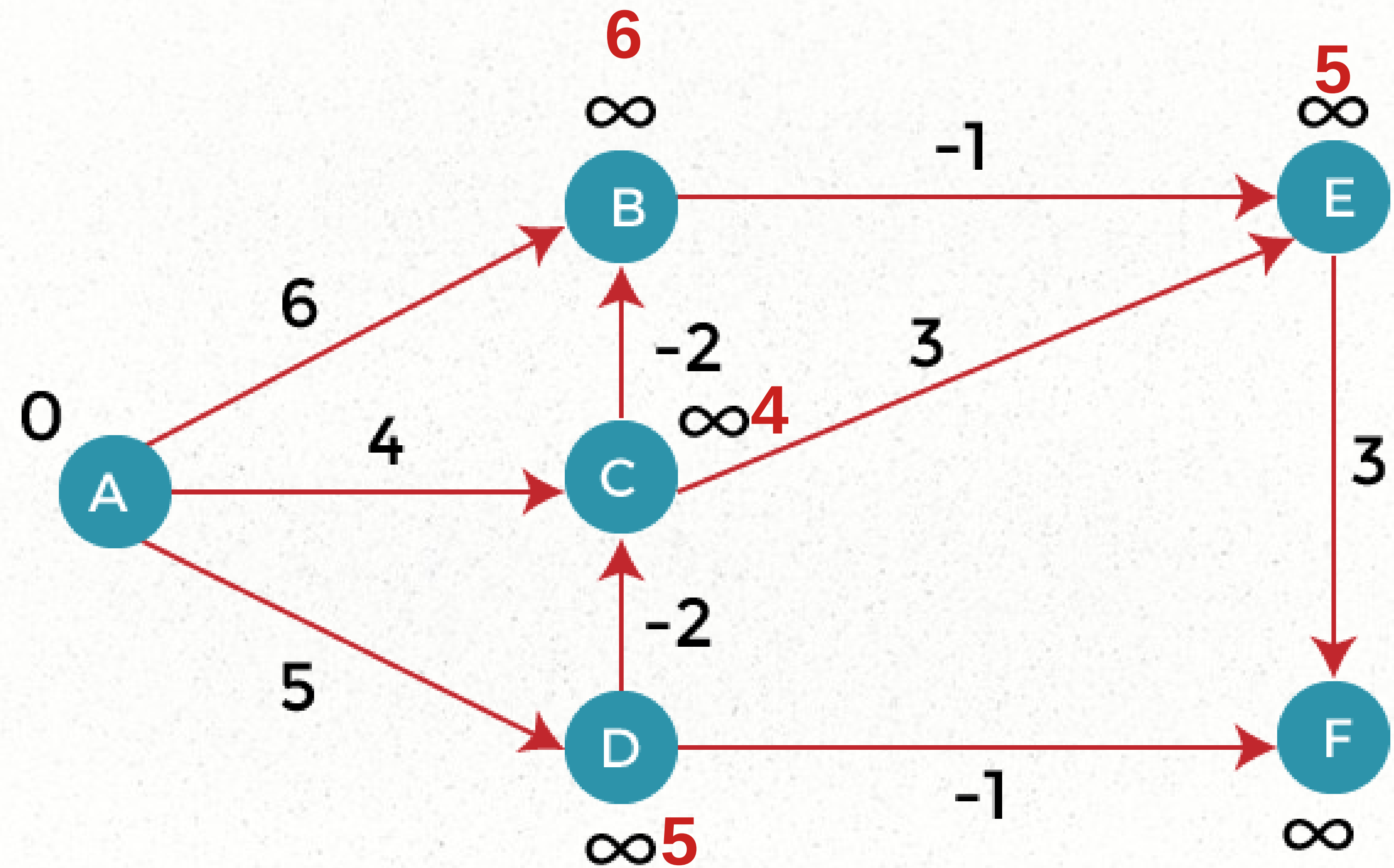
Since (6 - 1) is less than ∞, so update

$d(v) = d(u) + c(u , v)$

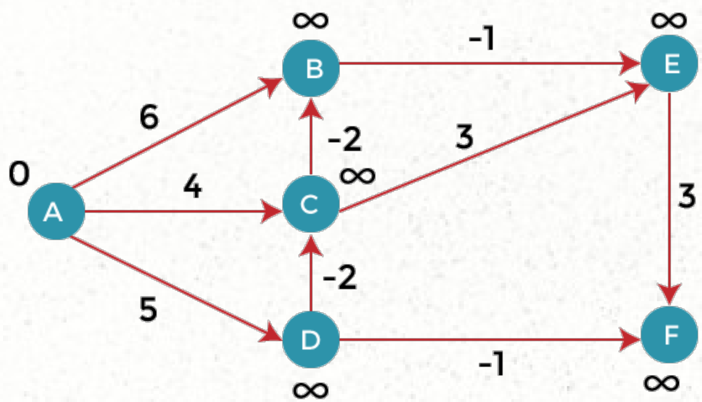
$d(v) = 6 - 1 = 5$

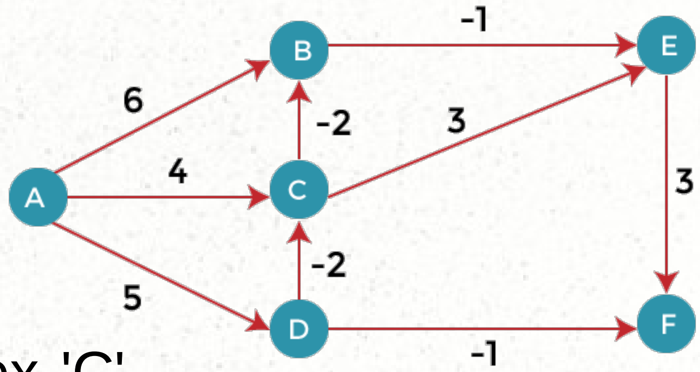
Therefore, the distance of vertex E is 5.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)


6
∞
5
∞
-1
B
E
6
-2
3
0
4
∞4
A
C
3
-2
5
D
F
-1
∞5
∞

Bellman Ford Algorithm





Consider the edge (C, E). Denote vertex 'C' as 'u' and vertex 'E' as 'v'. Now use the relaxing formula:

d(u) = 4

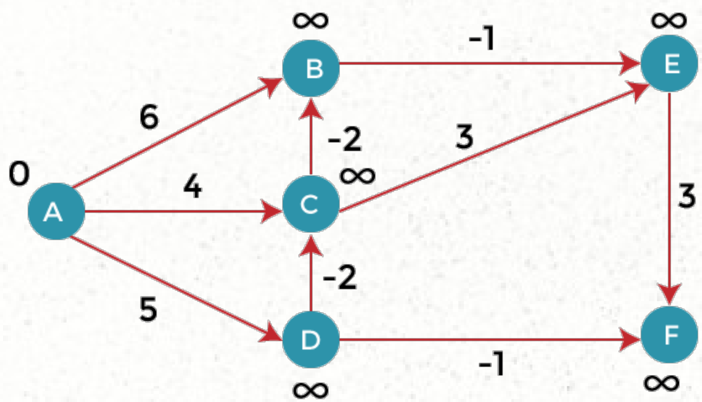
d(v) = 5

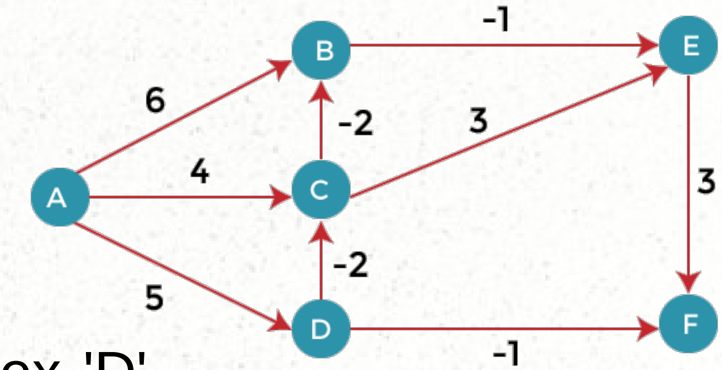
c(u , v) = 3

Since (4 + 3) is greater than 5, so there will be no updation. The value at vertex E is 5.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)

Bellman Ford Algorithm





Consider the edge (D, C). Denote vertex 'D' as 'u' and vertex 'C' as 'v'. Now use the relaxing formula:

d(u) = 5

d(v) = 4

c(u , v) = -2

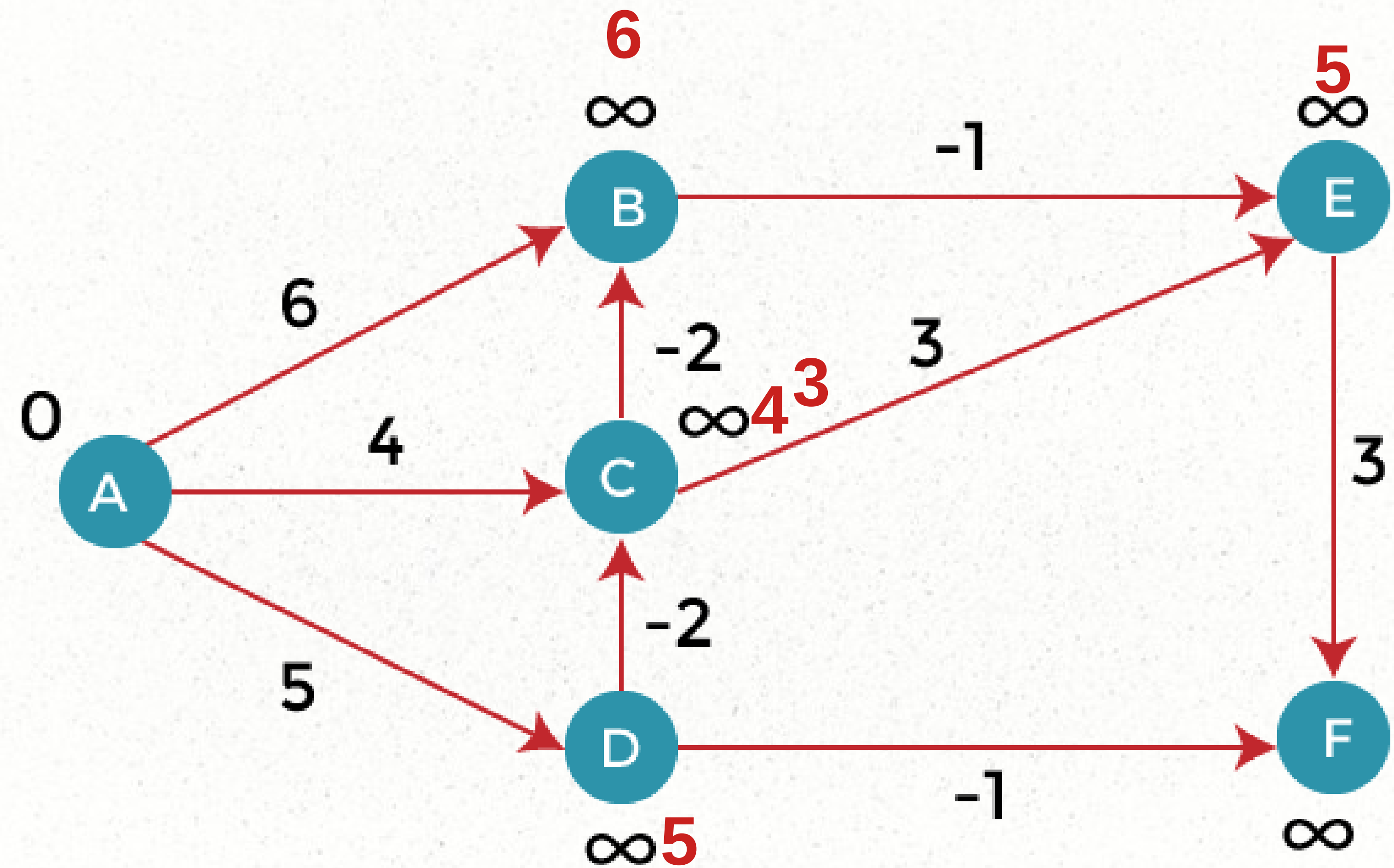
Since (5 -2) is less than 4, so update

d(v) = d(u) + c(u , v)

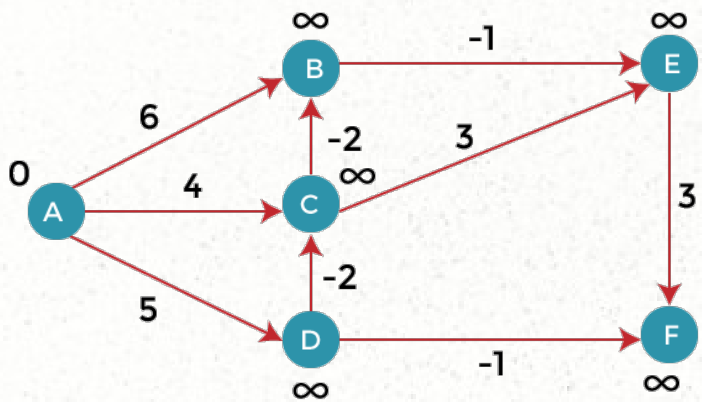
d(v) = 5 - 2 = 3

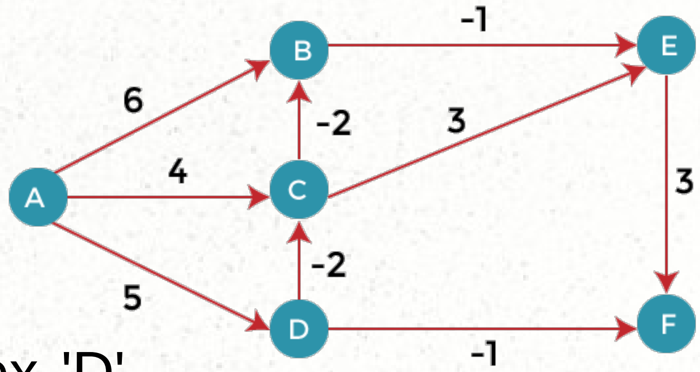
Therefore, the distance of vertex C is 3.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)


6
∞
5
∞
-1
B
E
6
-2
3
0
∞43
4
A
C
3
3
-2
5
D
F
-1
∞5
∞

Bellman Ford Algorithm





Consider the edge (D, F). Denote vertex 'D' as 'u' and vertex 'F' as 'v'. Now use the relaxing formula:

d(u) = 5

d(v) = ∞

c(u , v) = -1

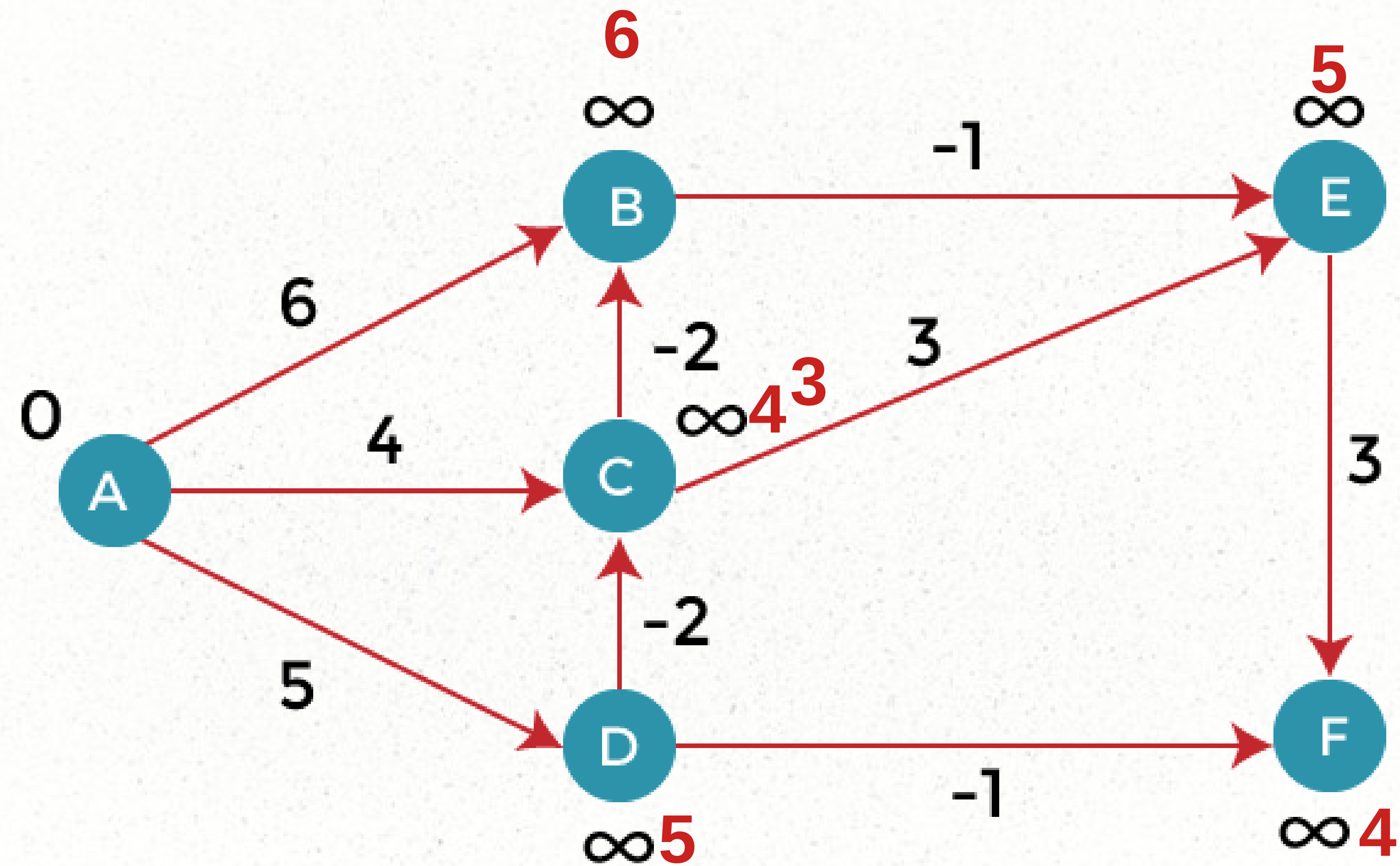
Since (5 -1) is less than ∞, so update

d(v) = d(u) + c(u , v)

d(v) = 5 - 1 = 4

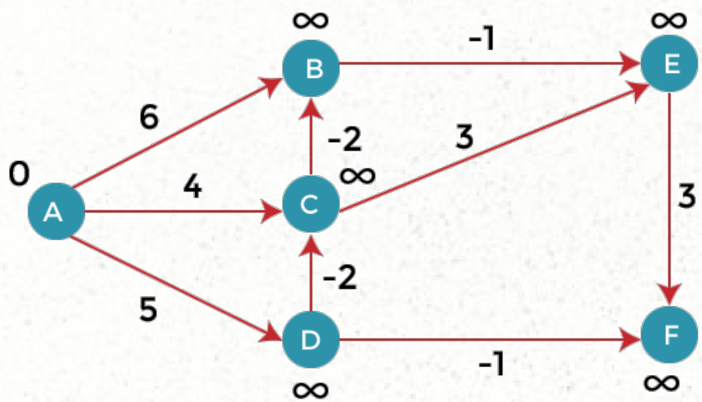
Therefore, the distance of vertex F is 4.

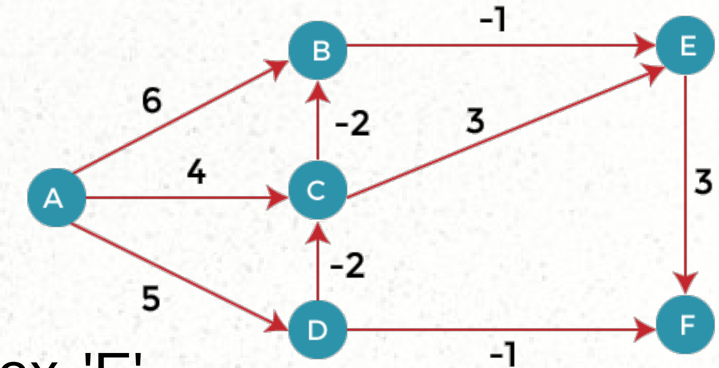
(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)



6
∞
5
∞
-1
B
E
6
-2
3
0
∞43
4
3
A
C
-2
5
D
F
-1
∞5
∞4

Bellman Ford Algorithm





Consider the edge (E, F). Denote vertex 'E' as 'u' and vertex 'F' as 'v'. Now use the relaxing formula:

$d(u) = 5$

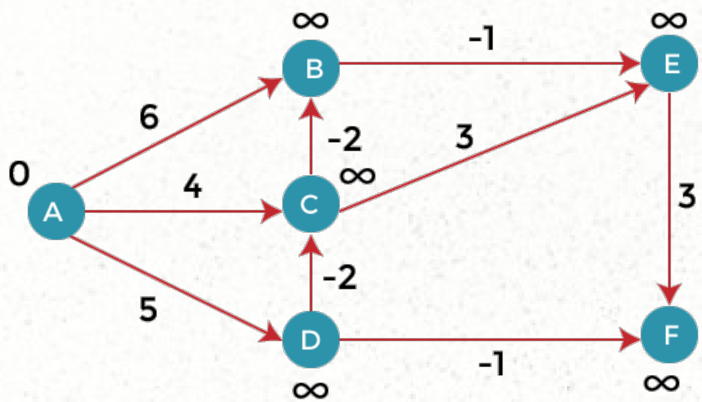
$d(v) = \infty$

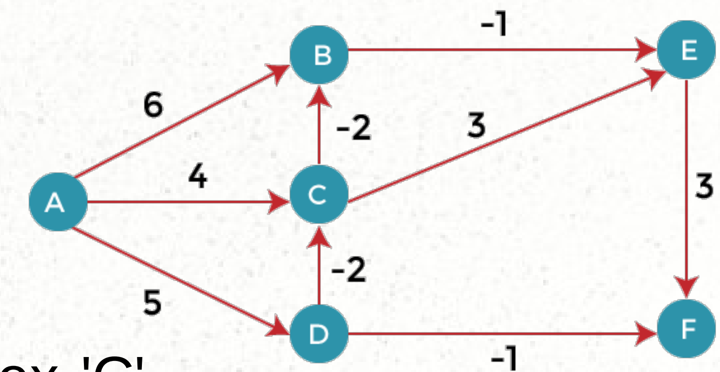
$c(u , v) = 3$

Since (5 + 3) is greater than 4, so there would be no updation on the distance value of vertex F.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)

Bellman Ford Algorithm





Consider the edge (C, B). Denote vertex 'C' as 'u' and vertex 'B' as 'v'. Now use the relaxing formula:

d(u) = 3

d(v) = 6

c(u , v) = -2

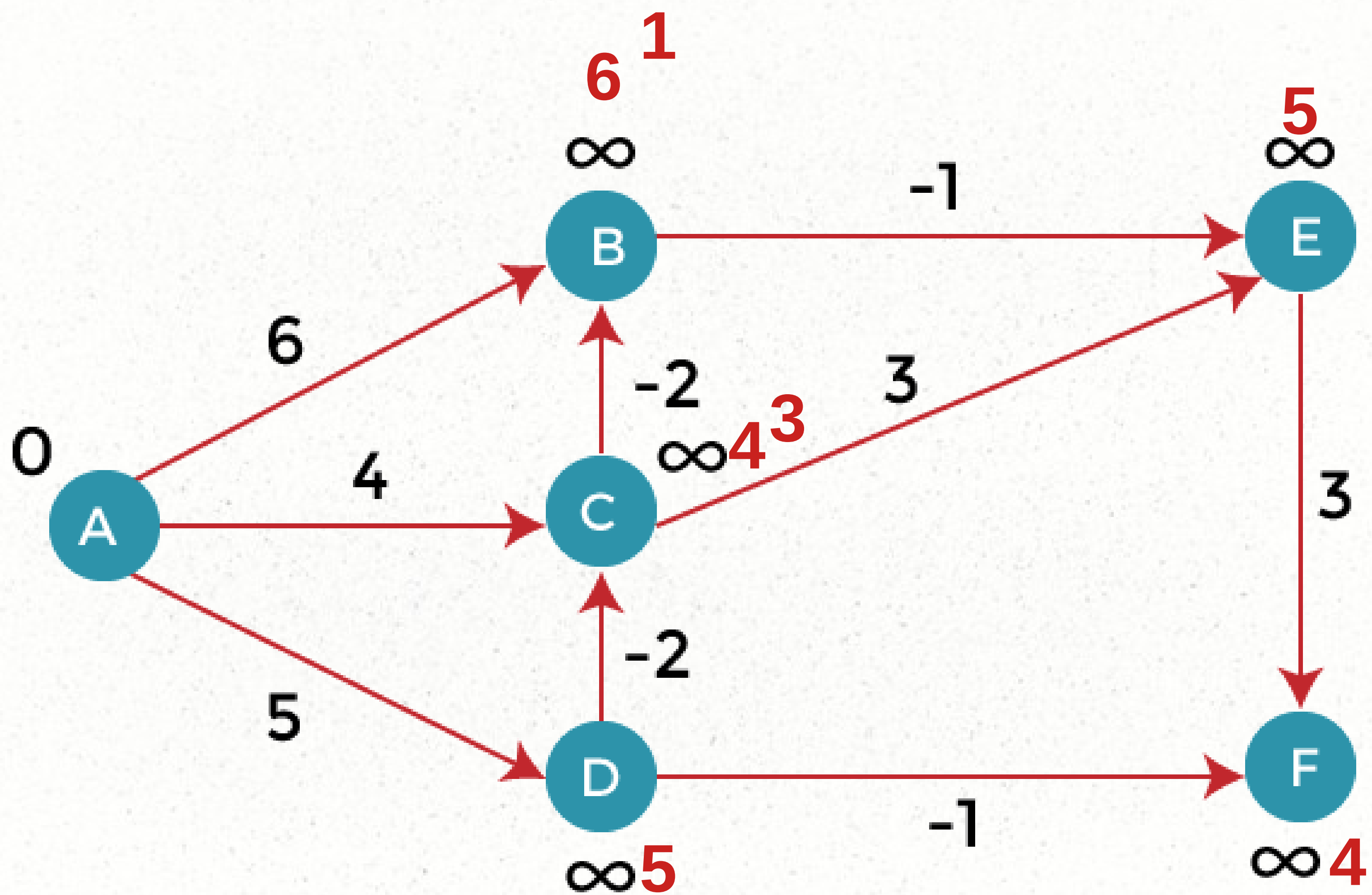
Since (3 - 2) is less than 6, so update

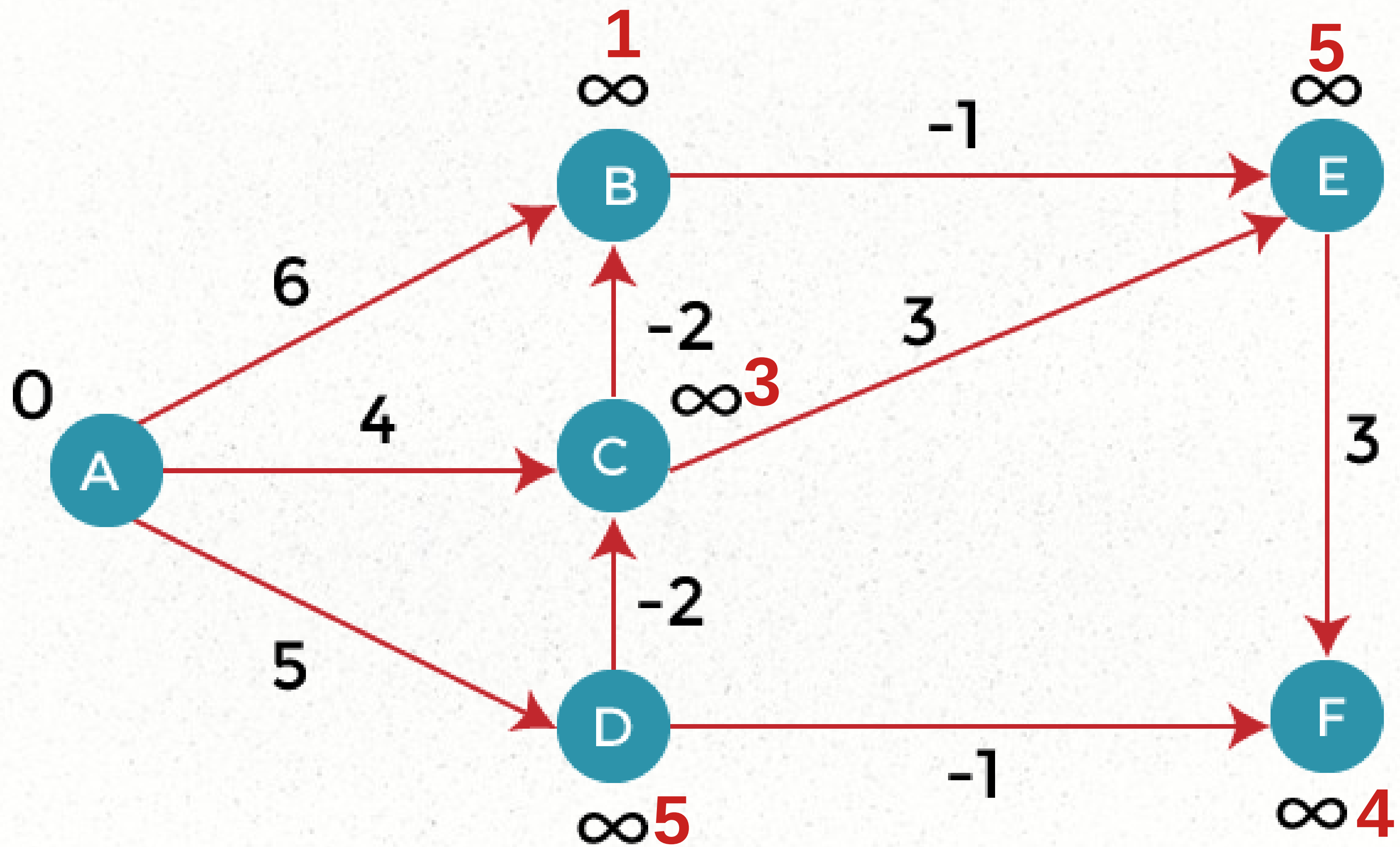
d(v) = d(u) + c(u , v)

d(v) = 3 - 2 = 1

Therefore, the distance of vertex B is 1.

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)


A
B
C
D
E
F
0
6
4
5
-2
-2
-1
3
3
-1
∞
6
1
∞
5
∞43
∞5
∞4

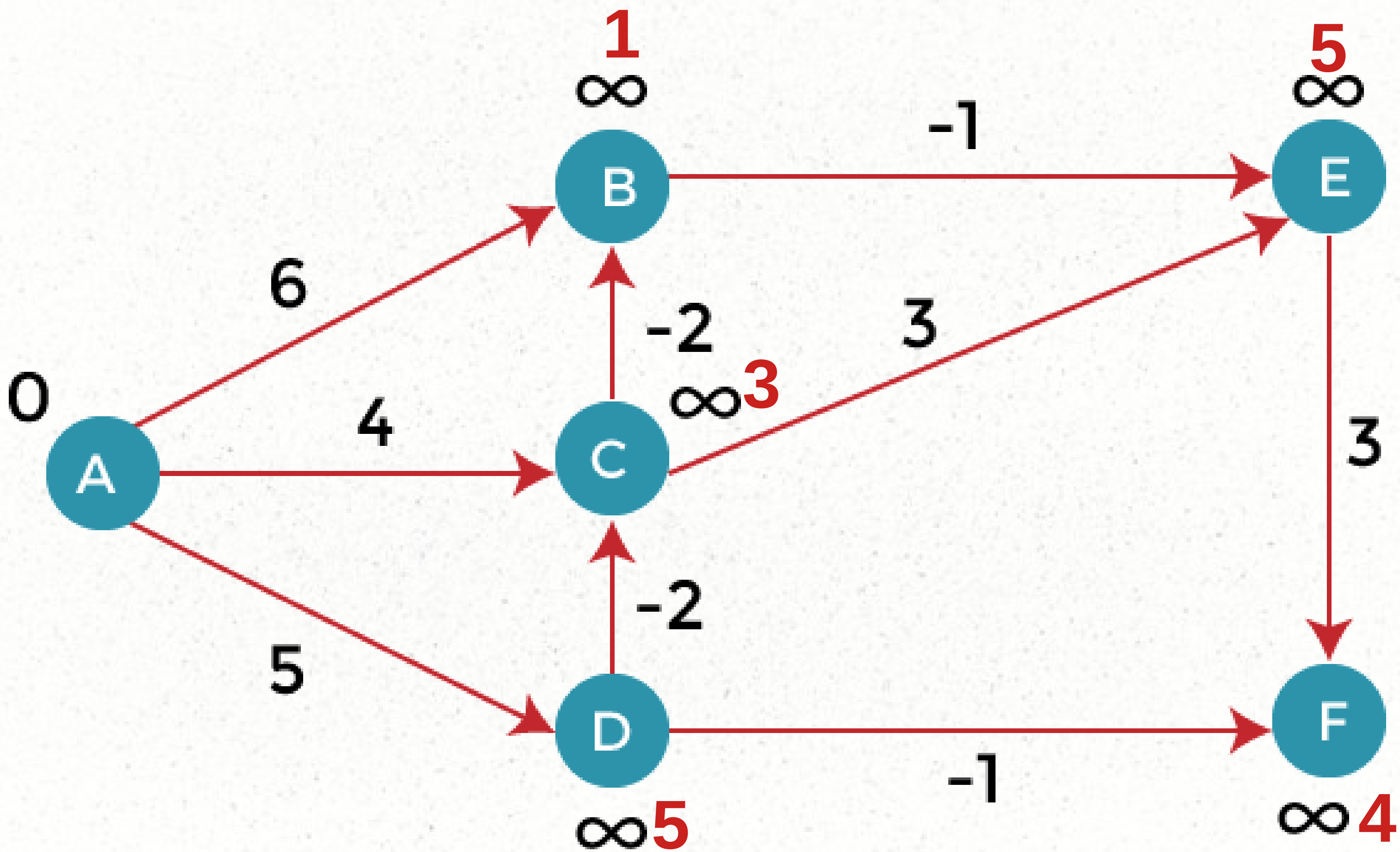


1
∞
5
∞
-1
B
E
6
-2
3
0
∞3
4
A
C
3
-2
5
D
F
-1
∞5
∞4

END OF ITERATION 1

Iteration 2

(A, B)
(A, C)
(A, D)
(B, E)
(C, E)
(D, C)
(D, F)
(E, F)
(C, B)



Bellman Ford Algorithm

- The first edge is (A, B). Since (0 + 6) is greater than 1 so there would be no updation in the vertex B.
- The next edge is (A, C). Since (0 + 4) is greater than 3 so there would be no updation in the vertex C.
- The next edge is (A, D). Since (0 + 5) equals to 5 so there would be no updation in the vertex D.

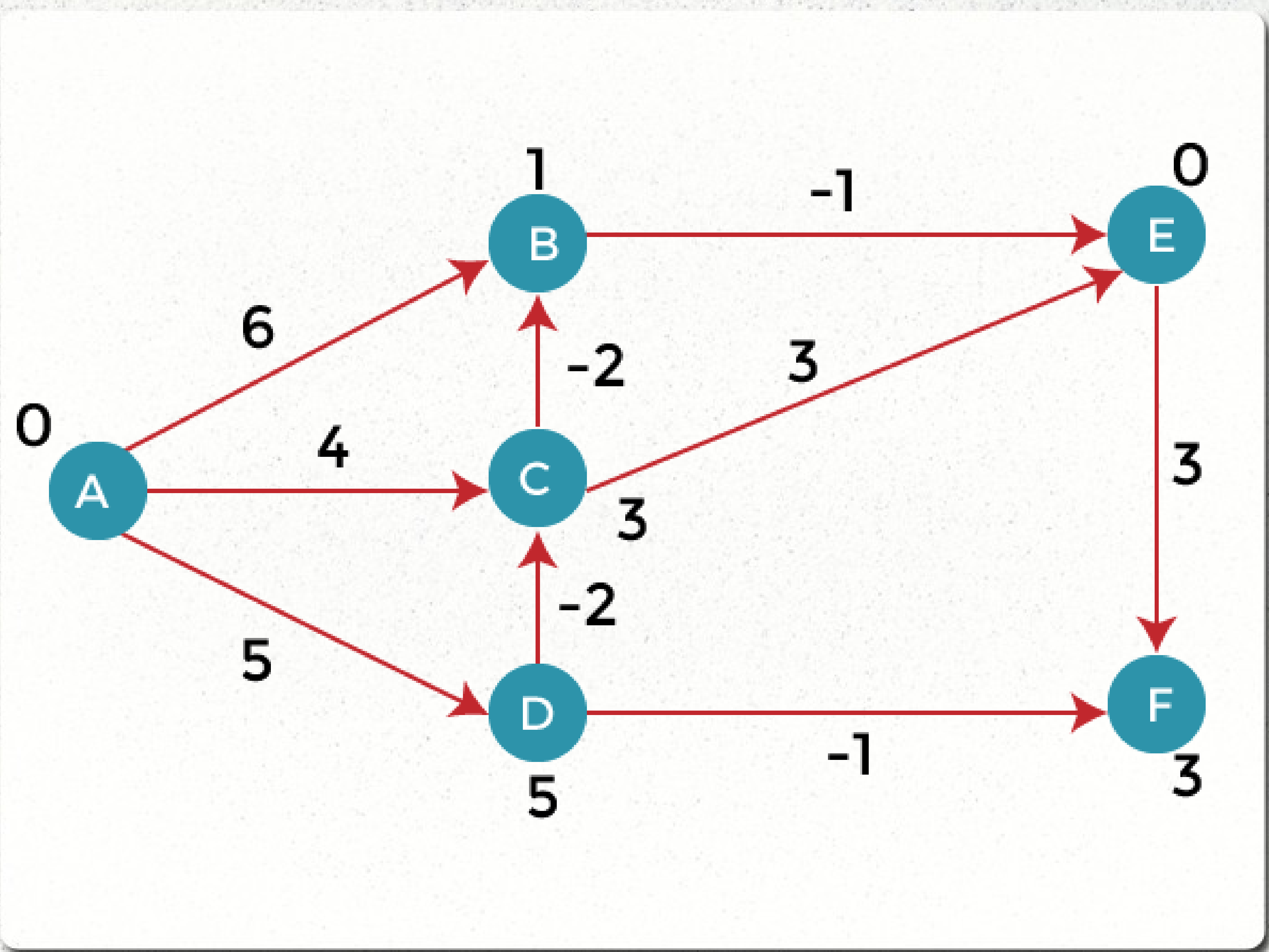
Bellman Ford Algorithm

- The next edge is (B, E). Since (1 - 1) equals to 0 which is less than 5 so update:
- d(v) = d(u) + c(u, v)
- d(E) = d(B) +c(B , E)
- = 1 - 1 = 0
- The next edge is (C, E). Since (3 + 3) equals to 6 which is greater than 5 so there would be no updation in the vertex E.
- The next edge is (D, C). Since (5 - 2) equals to 3 so there would be no updation in the vertex C.

Bellman Ford Algorithm

- The next edge is (D, F). Since (5 - 1) equals to 4 so there would be no updation in the vertex F.
- The next edge is (E, F). Since (5 + 3) equals to 8 which is greater than 4 so there would be no updation in the vertex F.
- The next edge is (C, B). Since (3 - 2) equals to 1` so there would be no updation in the vertex B.

End of Iteration 2



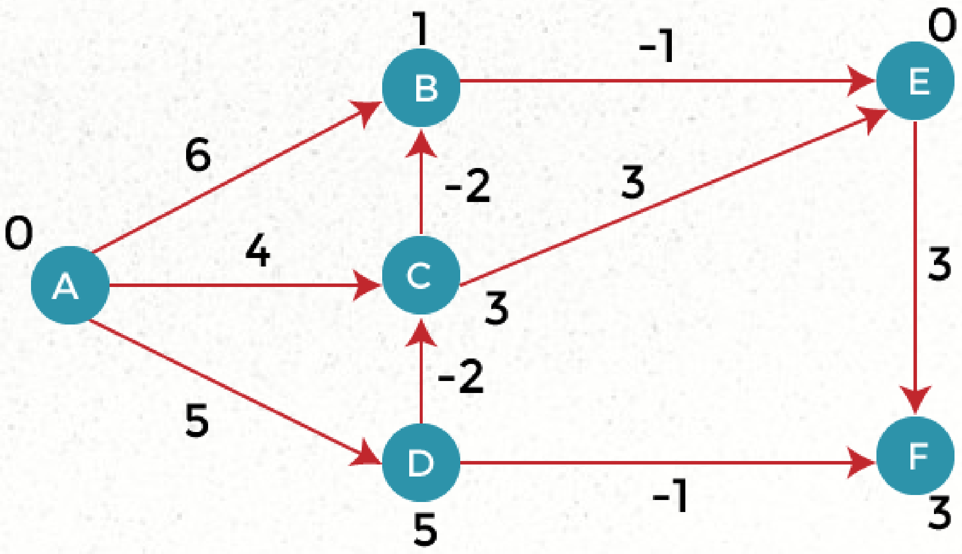
0
A
6
4
5
1
B
-2
C
3
-2
D
5
-1
3
E
0
3
F
-1
3

Bellman Ford Algorithm

- Third Iteration:
- We will perform the same steps as we did in the previous iterations. We will observe that there will be no updation in the distance of vertices.

Bellman Ford Algorithm

- The following are the distances of vertices:
- A: 0
- B: 1
- C: 3
- D: 5
- E: 0
- F: 3



```
function BellmanFord(list vertices, list edges, vertex source) is

    // This implementation takes in a graph, represented as
    // lists of vertices (represented as integers [0..n-1]) and edges,
    // and fills two arrays (distance and predecessor) holding
    // the shortest path from the source to each vertex

    distance := list of size n
    predecessor := list of size n

    // Step 1: initialize graph
    for each vertex v in vertices do

        distance[v] := inf             // Initialize the distance to all vertices to
infinity
        predecessor[v] := null         // And having a null predecessor

    distance[source] := 0              // The distance from the source to itself is, of
course, zero

    // Step 2: relax edges repeatedly

    repeat |V|−1 times:
         for each edge (u, v) with weight w in edges do
             if distance[u] + w < distance[v] then
                 distance[v] := distance[u] + w
                 predecessor[v] := u

    // Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges do
        if distance[u] + w < distance[v] then
            error "Graph contains a negative-weight cycle"

    return distance, predecessor
```

```
function BellmanFord(list vertices, list edges, vertex source) is

    // This implementation takes in a graph, represented as
    // lists of vertices (represented as integers [0..n-1]) and edges,
    // and fills two arrays (distance and predecessor) holding
    // the shortest path from the source to each vertex

    distance := list of size n
    predecessor := list of size n

    // Step 1: initialize graph
    for each vertex v in vertices do

        distance[v] := inf             // Initialize the distance to all vertices to
infinity
        predecessor[v] := null         // And having a null predecessor

    distance[source] := 0              // The distance from the source to itself is, of
course, zero

    // Step 2: relax edges repeatedly

    repeat |V|−1 times:
         for each edge (u, v) with weight w in edges do
             if distance[u] + w < distance[v] then
                 distance[v] := distance[u] + w
                 predecessor[v] := u

    // Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges do
        if distance[u] + w < distance[v] then
            error "Graph contains a negative-weight cycle"

    return distance, predecessor
```

Negative Cycle Detection

Bellman Ford Algorithm

- Time Complexity
- The time complexity of Bellman ford algorithm would be O(VE)
- Using Max Heap, Dijkstra’s complexity is O(ElogV), otherwise it is $O(V^2)$

Thank you

Any Question

_??????????_