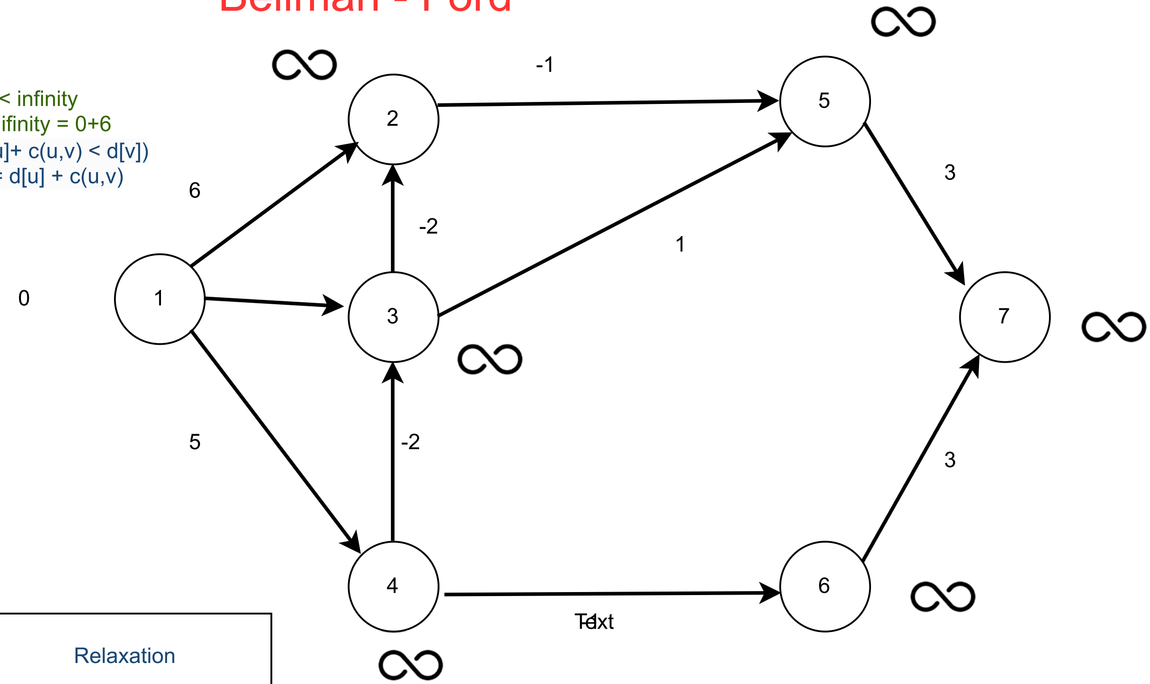
Single Source Shortest Path
Bellman - Ford

0+6 < infinity
so inifinity = 0+6
if(d[u]+ c(u,v) < d[v])
d[v]= d[u] + c(u,v)

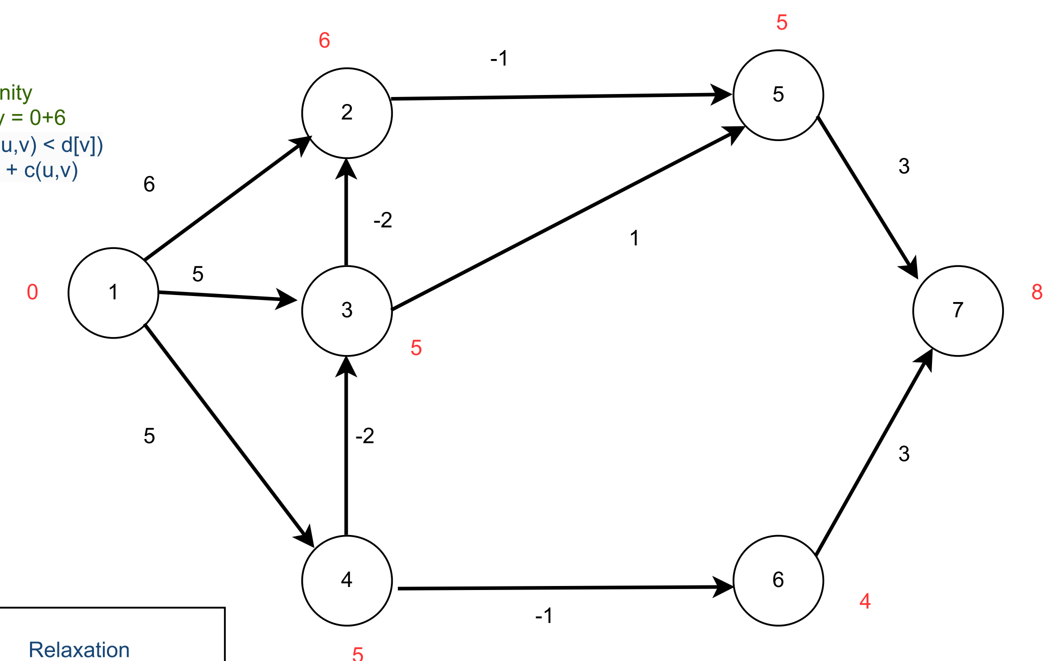


Relaxation

if(d[u]+ c(u,v) < d[v])
d[v]= d[u] + c(u,v)

Edge list = (1,2), (1,3),(1,4), (2,5), (3,2), (3,5), (4,3),(4,6), (5,7), (6,7)

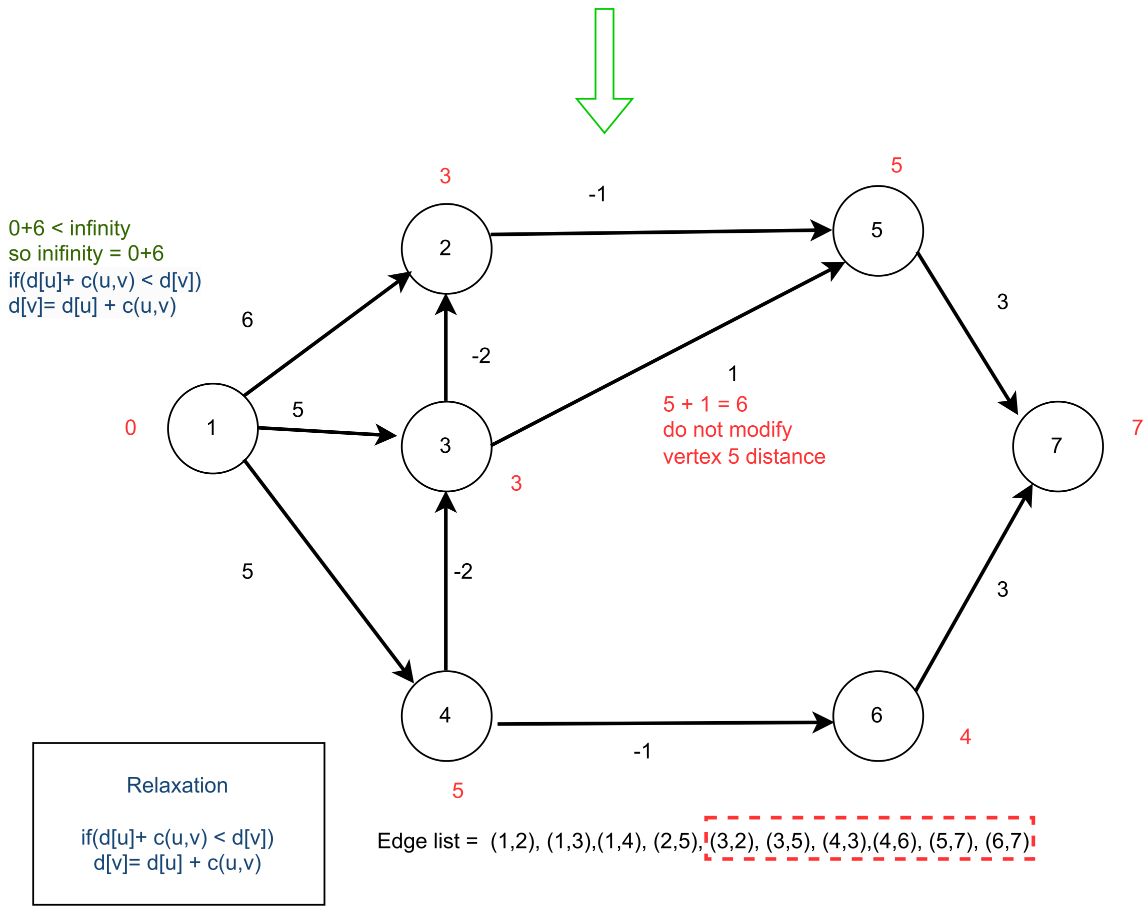
0+6 < infinity
so inifinity = 0+6
if(d[u]+ c(u,v) < d[v])
d[v]= d[u] + c(u,v)

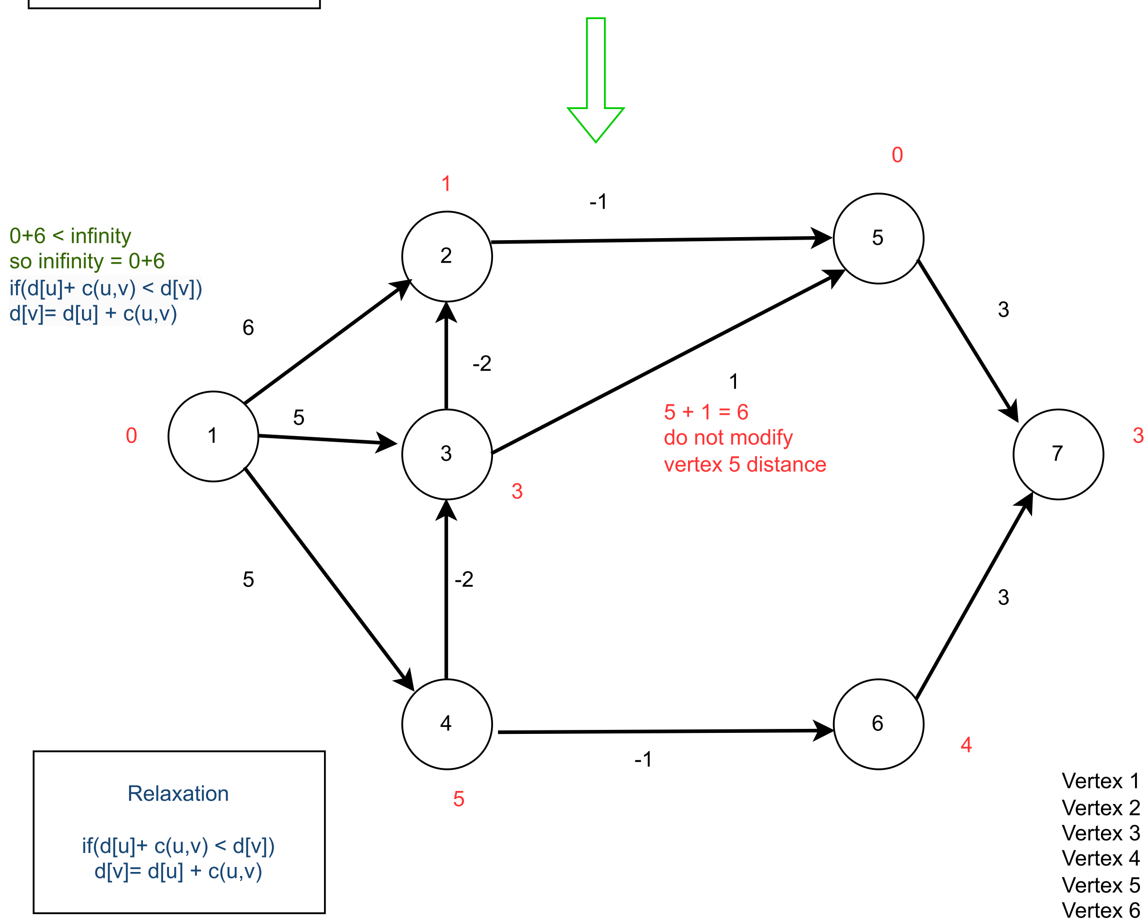


Relaxation

if(d[u]+ c(u,v) < d[v])
d[v]= d[u] + c(u,v)

Edge list = $\boxed{\text{(1,2), (1,3),(1,4), (2,5),}}$(3,2), (3,5), (4,3),(4,6), (5,7), (6,7)





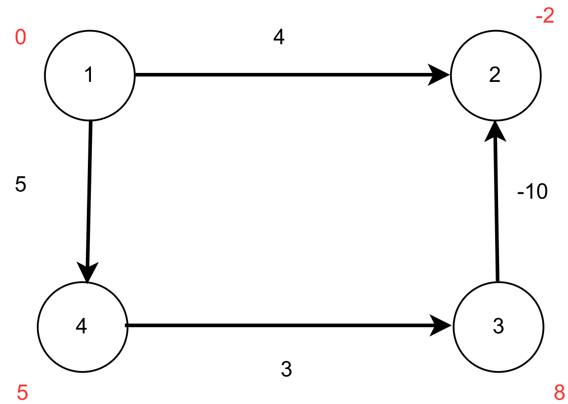
Vertex 1 = 0
Vertex 2 = 1
Vertex 3 = 3
Vertex 4 = 5
Vertex 5 = 0
Vertex 6 = 4
Vertex 7 = 3

Edge list = (1,2), (1,3),(1,4), (2,5), (3,2), (3,5), (4,3),(4,6), (5,7), (6,7)

Algorithm Efficiency -

--> The runtime for the Bellman-For shortest path algorithm is 0(VE)
--> The outer loop (the main iterations) executes V-1 times
--> The algorithm visits each vertex and follows the subset of edges to adjacent vertices, following a total of E edges across all loop executions.

Drawback of Bellman-Ford



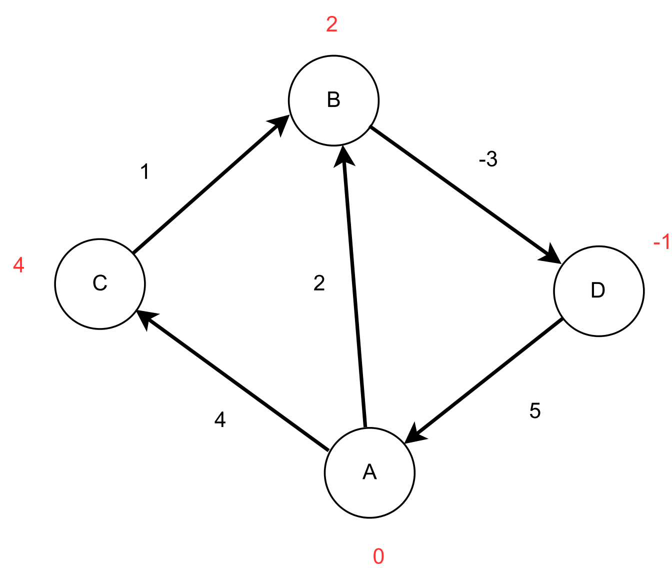
Edge list = (3,2),(4,3), (1,4), (1,2)

How many times should I relax the edges?
n = 4-1 = 3 times

1 = 0
2 = (-2)
3 = 8
4 = 5

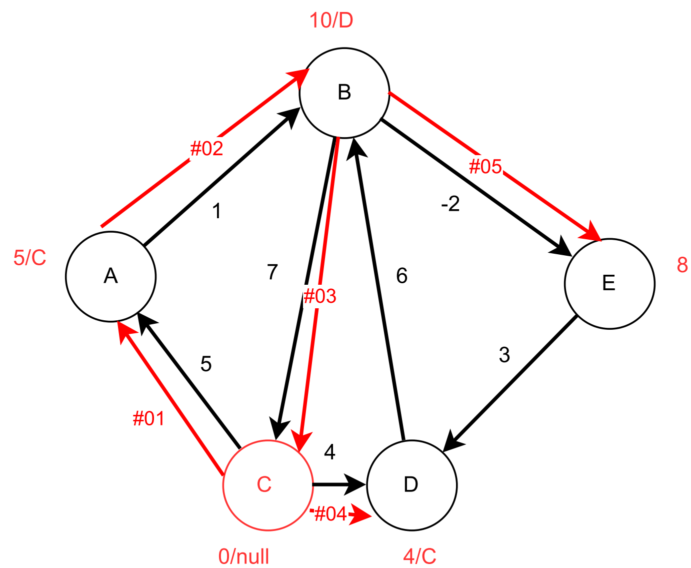
Practice

Start Vertex = B
Each Loop in the algorithm visits vertices in the graph in the following order: A, B, C, D



Practice

Start Vertex = C
Each Loop in the algorithm visits vertices in the graph in the following order: A, B, C, D, E

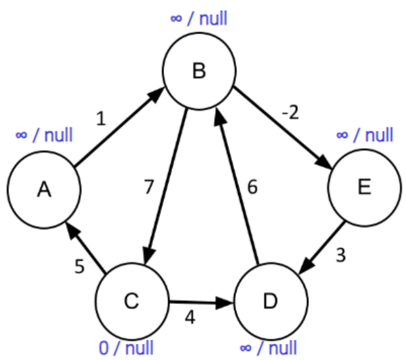


12.10.1: Bellman-Ford's shortest path.

c3zqy7

Jump to level 1

The Bellman-Ford algorithm is run on the following graph. The start vertex is C. Assume each loop of the algorithm visits vertices in the graph in the following order: A, B, C, D, E.



What are A's values after the first iteration? 5 / C Enter inf for ∞.

What are B's values after the first iteration? 6 / C

What are D's values after the first iteration? 4 / B

What are E's values after the first iteration? 8 / B

1 2 3 4

Check Next

✕ Each incorrect answer is highlighted.
Expected:
A: 5 / C
B: 10 / D
D: 4 / C
E: inf / null

A is visited first: B is adjacent, but A's current distance is ∞, so a path through A will not change B. So B remains ∞ / null.

B is visited next: No updates occur as B's current distance is ∞.

C is visited next: A is updated to 0 + 5 = 5 / C, and D is updated to 0 + 4 = 4 / C.

D is visited next: B is updated to 4 + 6 = 10 / D.

E is visited last: No updates occur as E's current distance is ∞.