



Bellman-Ford Algorithm for Shortest Path Problems

Sailesh Kumar (ur9289)

Daipayan Hati (ur9614)

Overview of Bellman-Ford Algorithm



- Definition: A graph-based algorithm for finding the shortest path from a single source to all other vertices in a weighted graph.
- Key Features:
 - Works on graphs with negative weight edges.
 - Can detect negative weight cycles.
- Applications:
 - Network routing protocols.
 - Solving linear programming problems.

Limitation of Dijkstra's Algorithm



Dijkstra is not suitable when the graph consists of negative edges.

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

Bellman-Ford Algorithm Steps

1. Initialize distances:

- Set the source vertex distance to 0 and all other distances to infinity.

2. Relax all edges $|V|-1$ times:

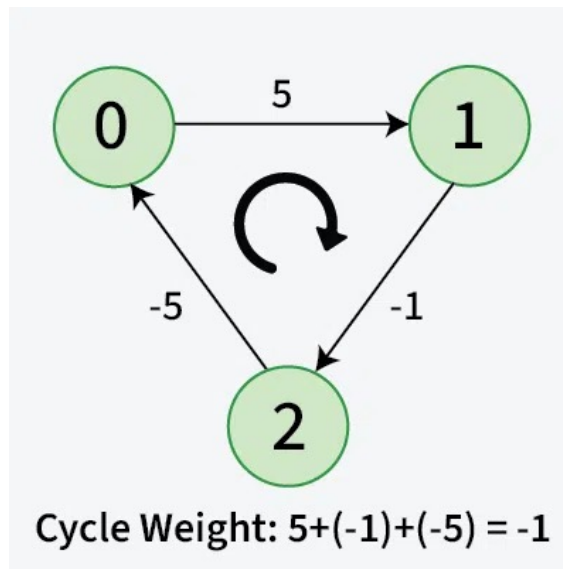
- For each edge (u, v) with weight w , update $\text{distance}[v] = \min(\text{distance}[v], \text{distance}[u] + w)$.

3. Check for negative weight cycles:

- Repeat edge relaxation once more; if any distance changes, the graph contains a negative cycle.


What is a negative cycle?

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

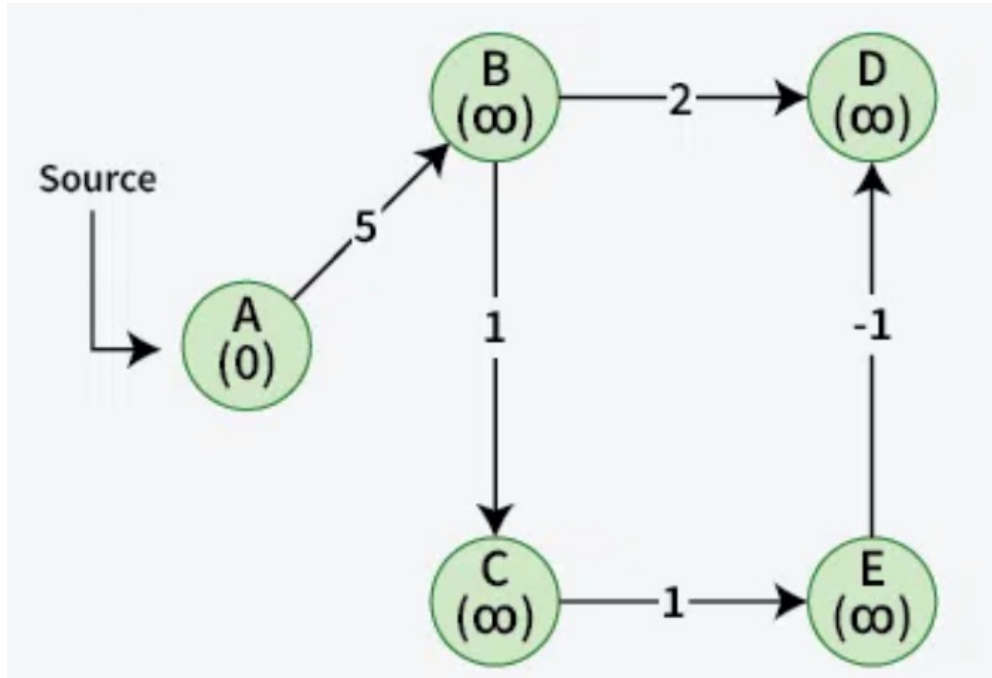


In the presence of negative weight cycle in the graph, the shortest path doesn't exist because with each traversal of the cycle shortest path keeps decreasing.

Why Relaxing Edges $(V - 1)$ times gives us Single Source Shortest Path?

- 
- A shortest path between two vertices can have at most $(V - 1)$ edges.
 - It is not possible to have a simple path with more than $(V - 1)$ edges (otherwise it would form a cycle).
 - Therefore, repeating the relaxation process $(V - 1)$ times ensures that all possible paths between source and any other node have been covered.

Example Graph

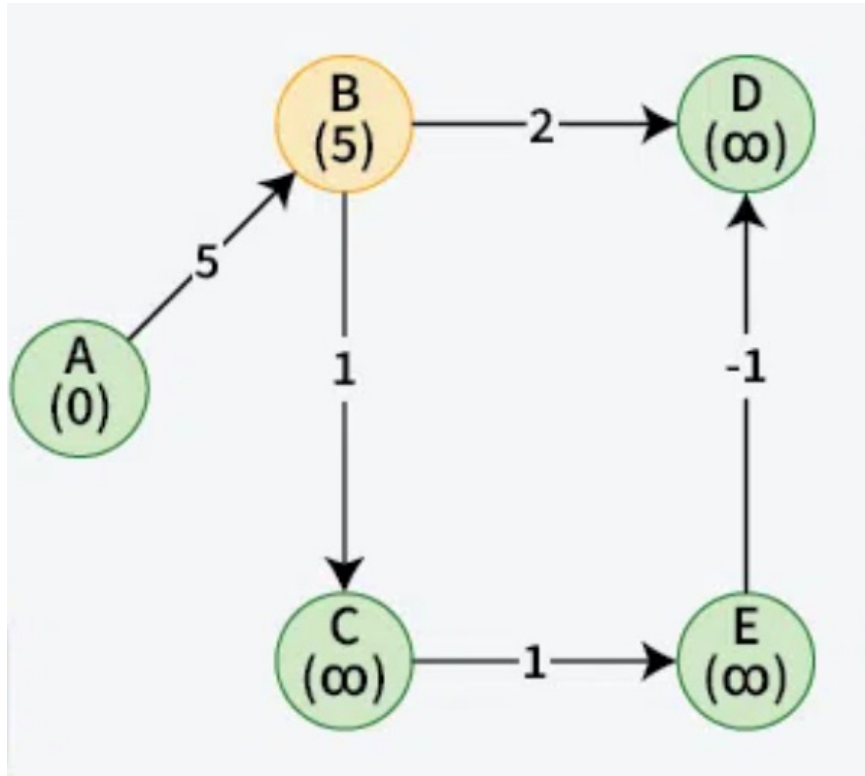


A	B	C	D	E
0	∞	∞	∞	∞

Distance Array

Initialize the distance array

1st Relaxation of edges



$\text{Dist}[A] + 5 < \text{Dist}[B]$

$0 + 5 < \infty$

Change distance of B

if $\text{dist}[u] \neq \infty$ && $\text{dist}[u] + \text{wt}(u,v) < \text{dist}[v]$
then, $\text{dist}[v] = \text{dist}[u] + \text{wt}(u,v)$;

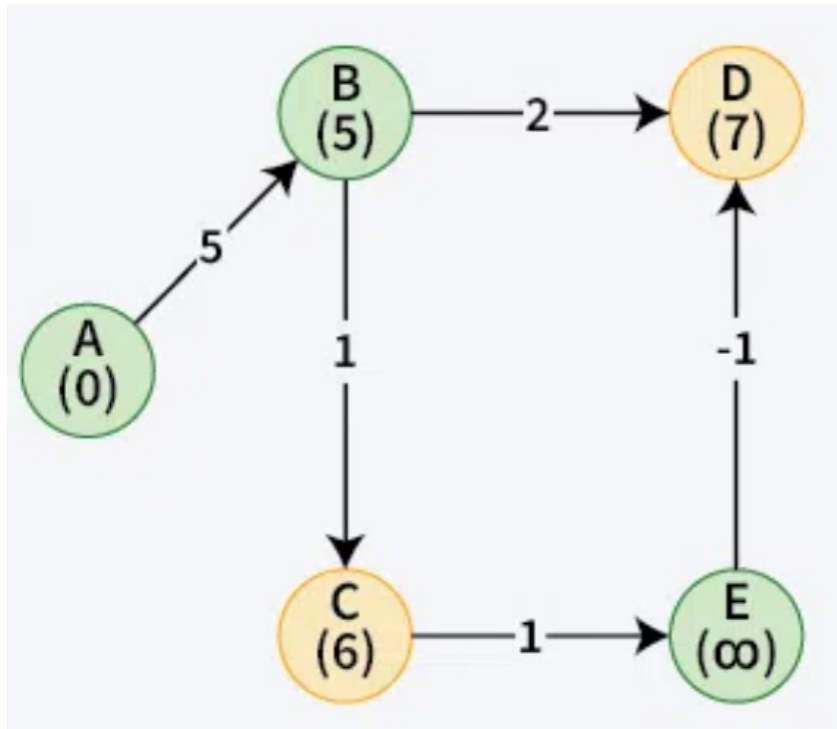
Distance Array

A	B	C	D	E
0	∞	∞	∞	∞



A	B	C	D	E
0	5	∞	∞	∞

2nd Relaxation of edges



if $\text{dist}[u] \neq \infty$ && $\text{dist}[u] + \text{wt}(u,v) < \text{dist}[v]$
then, $\text{dist}[v] = \text{dist}[u] + \text{wt}(u,v)$;

Distance Array

A	B	C	D	E
0	5	∞	∞	∞

↓

A	B	C	D	E
0	5	6	7	∞

$$\text{Dist}[B] + 2 < \text{Dist}[D]$$

$$5 + 2 < \infty$$

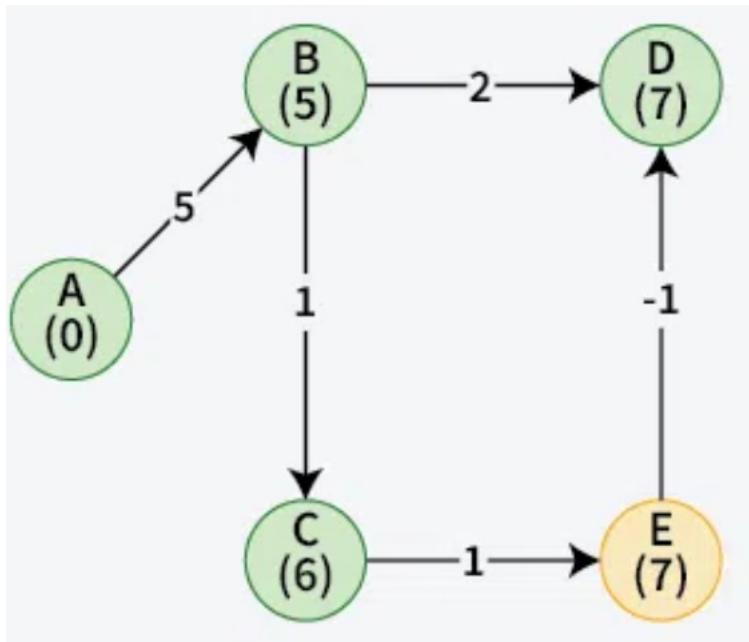
Change distance of D

$$\text{Dist}[B] + 1 < \text{Dist}[C]$$

$$5 + 1 < \infty$$

Change distance of C

3rd Relaxation of edges



$\text{Dist}[C] + 1 < \text{Dist}[E]$
 $6 + 1 < \infty$
Change distance of E

if $\text{dist}[u] \neq \infty$ && $\text{dist}[u] + \text{wt}(u,v) < \text{dist}[v]$
then, $\text{dist}[v] = \text{dist}[u] + \text{wt}(u,v)$;

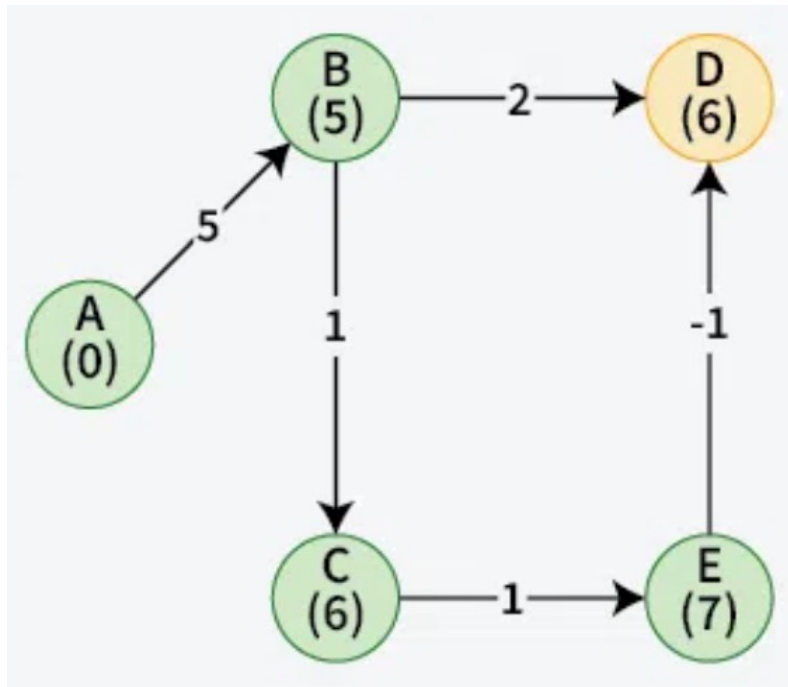
Distance Array

A	B	C	D	E
0	5	6	7	∞

↓

A	B	C	D	E
0	5	6	7	7

4th Relaxation of edges



if $\text{dist}[u] \neq \infty \ \&\& \ \text{dist}[u] + \text{wt}(u,v) < \text{dist}[v]$
then, $\text{dist}[v] = \text{dist}[u] + \text{wt}(u,v)$;

$\text{Dist}[E] + (-1) < \text{Dist}[D]$
 $7 + (-1) < 7$
Change distance of D

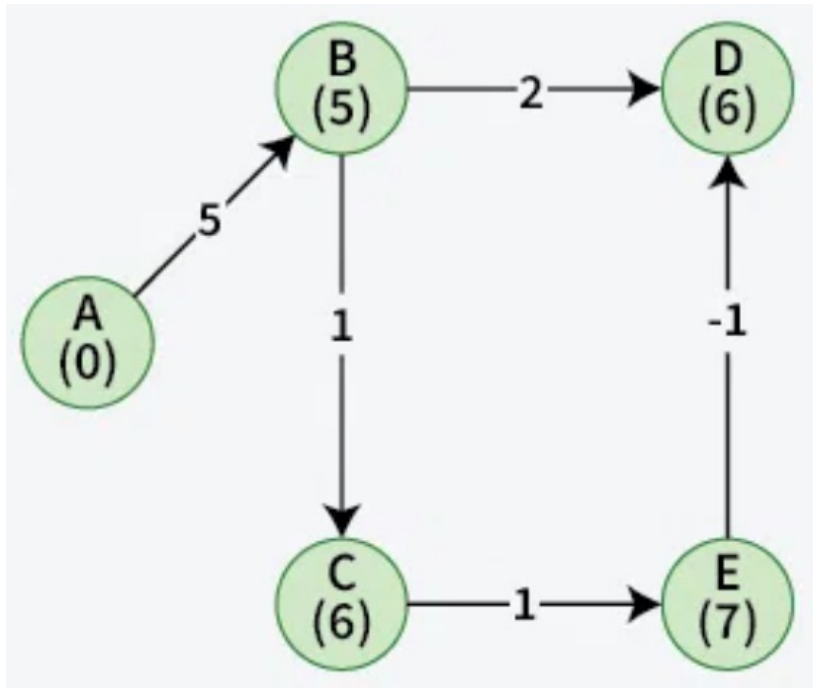
Distance Array

A	B	C	D	E
0	5	6	7	7



A	B	C	D	E
0	5	6	6	7

5th Relaxation to find a negative weight cycle



if $\text{dist}[u] \neq \infty$ && $\text{dist}[u] + \text{wt}(u,v) < \text{dist}[v]$
then, $\text{dist}[v] = \text{dist}[u] + \text{wt}(u,v)$;

A	B	C	D	E
0	5	6	6	7

Distance Array

Since no shorter distance for any node is found, the graph does not contain a negative weight cycle.

Dynamic programming table

- This table demonstrates how the Bellman-Ford Algorithm dynamically builds solutions to smaller subproblems (individual edge relaxations) and progressively combines them to solve the overall shortest path problem for all vertices

	A	B	C	D	E
Initial	0	∞	∞	∞	∞
1st Relaxation	0	5	∞	∞	∞
2 nd Relaxation	0	5	6	7	∞
3 rd Relaxation	0	5	6	7	7
4 th Relaxation	0	5	6	6	7
5 th Relaxation (Negative Cycle Check)	0	5	6	6	7

Code

```
class bellman {
    static int[] bellmanFord(int V, int[][] edges, int src) {

        // Initially distance from source to all other vertices
        // is not known(Infinite).
        int[] dist = new int[V];
        Arrays.fill(dist, (int)1e8);
        dist[src] = 0;

        // Relaxation of all the edges V times, not (V - 1) as we
        // need one additional relaxation to detect negative cycle
        for (int i = 0; i < V; i++) {
            for (int[] edge : edges) {
                int u = edge[0];
                int v = edge[1];
                int wt = edge[2];
                if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {

                    // If this is the Vth relaxation, then there is
                    // a negative cycle
                    if (i == V - 1)
                        return new int[]{-1};

                    // Update shortest distance to node v
                    dist[v] = dist[u] + wt;
                }
            }
        }
        return dist;
    }
}
```

```
public static void main(String[] args) {
    int V = 5;
    int[][] edges = new int[][] {
        {1, 3, 2},
        {4, 3, -1},
        {2, 4, 1},
        {1, 2, 1},
        {0, 1, 5}
    };

    int src = 0;
    int[] ans = bellmanFord(V, edges, src);
    for (int dist : ans)
        System.out.print(dist + " ");
}
```

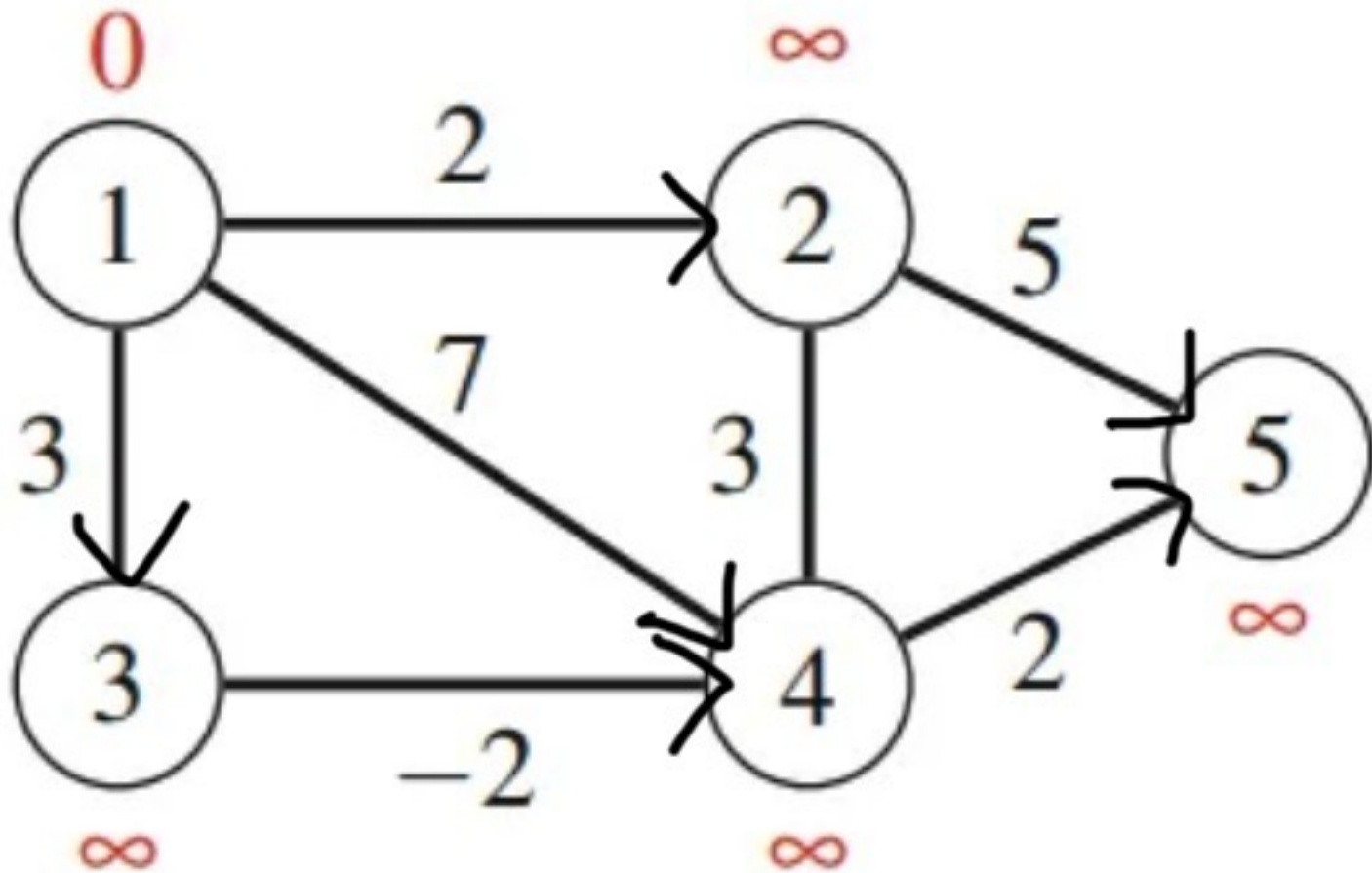
Output:

```
[Running] cd "/Users/sailesh/repos/Advanced
0 5 6 6 7
[Done] exited with code=0 in 0.395 seconds
```

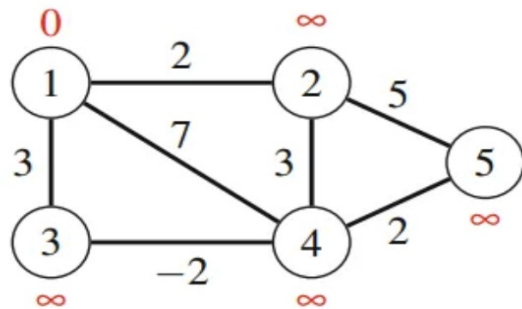
Comparison with Dijkstra's Algorithm

Feature	Bellman-Ford	Dijkstra's
Handles Negative Weights?	Yes	No
Time Complexity	$O(V * E)$	$O((V + E) * \log V)^*$
Algorithm Type	Dynamic Programming	Greedy
Negative Cycle Detection	Yes	No

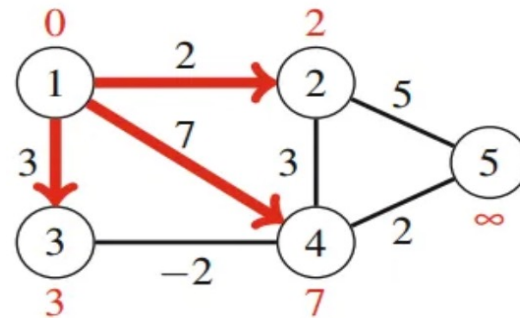
Example Problem



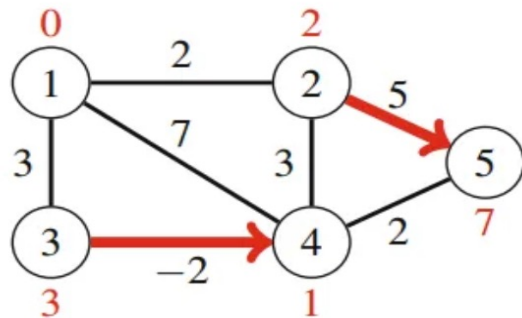
Solution to Example Problem



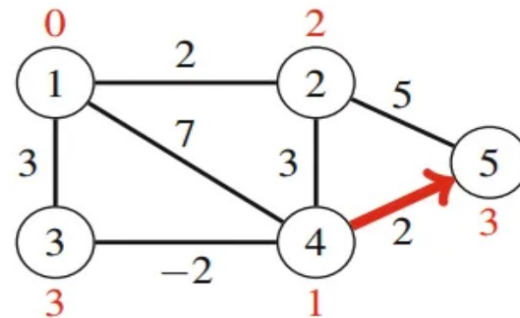
step 1



step 2



step 3



step 4

Vertex	1	2	3	4	5
Shortest Distance	0	2	3	1	3

Time Complexity

Worst Case: $O(V \cdot E)$

- The worst-case time complexity of Bellman-Ford algorithm is also $O(V \cdot E)$.
- This scenario occurs when the algorithm needs to iterate through all vertices and edges for $V - 1$ passes, followed by one more pass for detecting negative weight cycles.
- It's important to note that the worst case includes the presence of negative weight cycles, which can cause the algorithm to run indefinitely if not handled properly.



Thank you!
