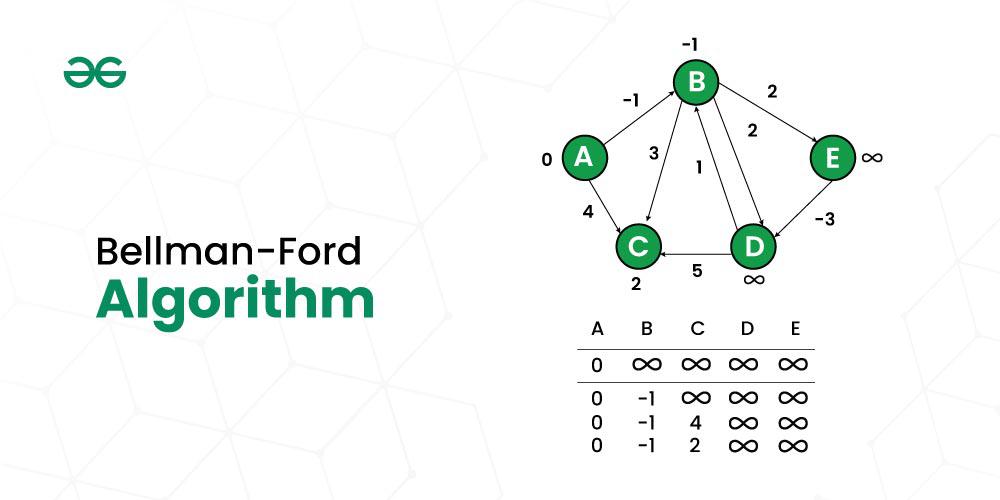
# **Bellman–Ford Algorithm**

Imagine you have a map with different cities connected by roads, each road having a certain distance. The **Bellman–Ford algorithm** is like a guide that helps you find the shortest path from one city to all other cities, even if some roads have negative lengths. It’s like a **GPS** for computers, useful for figuring out the quickest way to get from one point to another in a network. In this article, we’ll take a closer look at how this algorithm works and why it’s so handy in solving everyday problems.



**Table of Content**

* [Bellman-Ford Algorithm](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#bellmanford-algorithm)
* [The idea behind Bellman Ford Algorithm](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#the-idea-behind-bellman-ford-algorithm)
* [Principle of Relaxation of Edges for Bellman-Ford](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#principle-of-relaxation-of-edges-for-bellmanford)
* [Why Relaxing Edges N-1 times, gives us Single Source Shortest Path?](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#why-relaxing-edges-n1-times-gives-us-single-source-shortest-path)
* [Why Does the Reduction of Distance in the N’th Relaxation Indicates the Existence of a Negative Cycle?](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#why-does-the-reduction-of-distance-in-the-nth-relaxation-indicates-the-existence-of-a-negative-cycle)
* [Working of Bellman-Ford Algorithm to Detect the Negative cycle in the graph](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#working-of-bellmanford-algorithm-to-detect-the-negative-cycle-in-the-graph)
* [Algorithm to Find Negative Cycle in a Directed Weighted Graph Using Bellman-Ford](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#algorithm-to-find-negative-cycle-in-a-directed-weighted-graph-using-bellmanford)
* [Handling Disconnected Graphs in the Algorithm](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#handling-disconnected-graphs-in-the-algorithm)
* [Complexity Analysis of Bellman-Ford Algorithm](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#complexity-analysis-of-bellmanford-algorithm)
* [Bellman Ford’s Algorithm Applications](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#bellman-fords-algorithm-applications)
* [Drawback of Bellman Ford’s Algorithm](https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/#drawback-of-bellman-fords-algorithm)

## **Bellman-Ford Algorithm**

***Bellman-Ford*** *is a* ***single source*** *shortest path algorithm that determines the shortest path between a given source vertex and every other vertex in a graph. This algorithm can be used on both* ***weighted*** *and* ***unweighted*** *graphs.*

A **Bellman-Ford** algorithm is also guaranteed to find the shortest path in a graph, similar to [**Dijkstra’s algorithm**](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/). Although Bellman-Ford is slower than **Dijkstra’s algorithm**, it is capable of handling graphs with **negative edge weights**, which makes it more versatile. The shortest path cannot be found if there exists a **negative cycle** in the graph. If we continue to go around the negative cycle an infinite number of times, then the cost of the path will continue to decrease (even though the length of the path is increasing). As a result, **Bellman-Ford** is also capable of detecting **negative cycles**, which is an important feature.

## **The idea behind Bellman Ford Algorithm:**

*The Bellman-Ford algorithm’s primary principle is that it starts with a single source and calculates the distance to each node. The distance is initially unknown and assumed to be infinite, but as time goes on, the algorithm relaxes those paths by identifying a few shorter paths. Hence it is said that Bellman-Ford is based on “****Principle of Relaxation****“.*

## **Principle of Relaxation of Edges for Bellman-Ford:**

* It states that for the graph having **N** vertices, all the edges should be relaxed **N-1** times to compute the single source shortest path.
* In order to detect whether a negative cycle exists or not, relax all the edge one more time and if the shortest distance for any node reduces then we can say that a negative cycle exists. In short if we relax the edges **N** times, and there is any change in the shortest distance of any node between the **N-1th** and **Nth** relaxation than a negative cycle exists, otherwise not exist.

## **Why Relaxing Edges N-1 times, gives us Single Source Shortest Path?**

In the worst-case scenario, a shortest path between two vertices can have at most **N-1** edges, where **N** is the number of vertices. This is because a simple path in a graph with **N** vertices can have at most **N-1** edges, as it’s impossible to form a closed loop without revisiting a vertex.

By relaxing edges **N-1** times, the Bellman-Ford algorithm ensures that the distance estimates for all vertices have been updated to their optimal values, assuming the graph doesn’t contain any negative-weight cycles reachable from the source vertex. If a graph contains a negative-weight cycle reachable from the source vertex, the algorithm can detect it after **N-1** iterations, since the negative cycle disrupts the shortest path lengths.

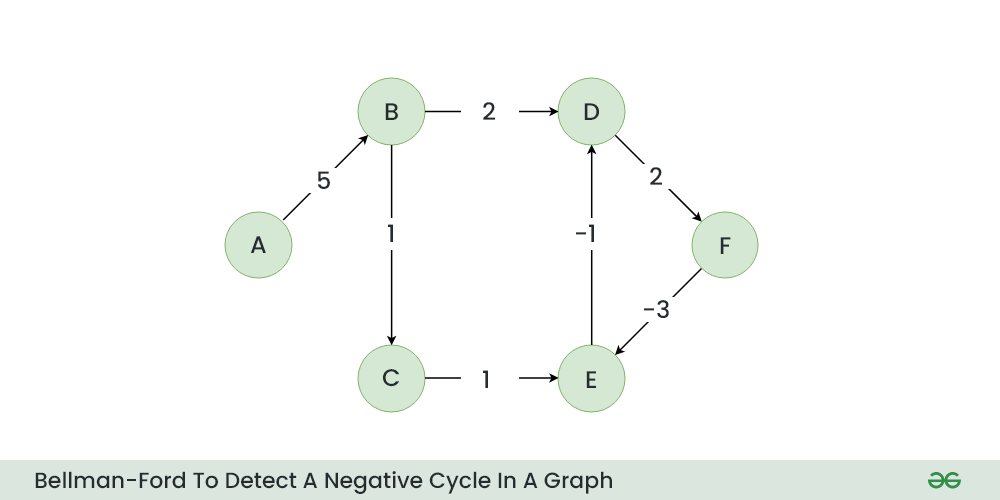
In summary, relaxing edges **N-1** times in the Bellman-Ford algorithm guarantees that the algorithm has explored all possible paths of length up to **N-1**, which is the maximum possible length of a shortest path in a graph with **N** vertices. This allows the algorithm to correctly calculate the shortest paths from the source vertex to all other vertices, given that there are no negative-weight cycles.

## **Why Does the Reduction of Distance in the N’th Relaxation Indicates the Existence of a Negative Cycle?**

As previously discussed, achieving the single source shortest paths to all other nodes takes **N-1** relaxations. If the N’th relaxation further reduces the shortest distance for any node, it implies that a certain edge with negative weight has been traversed once more. It is important to note that during the **N-1** relaxations, we presumed that each vertex is traversed only once. However, the reduction of distance during the N’th relaxation indicates revisiting a vertex.

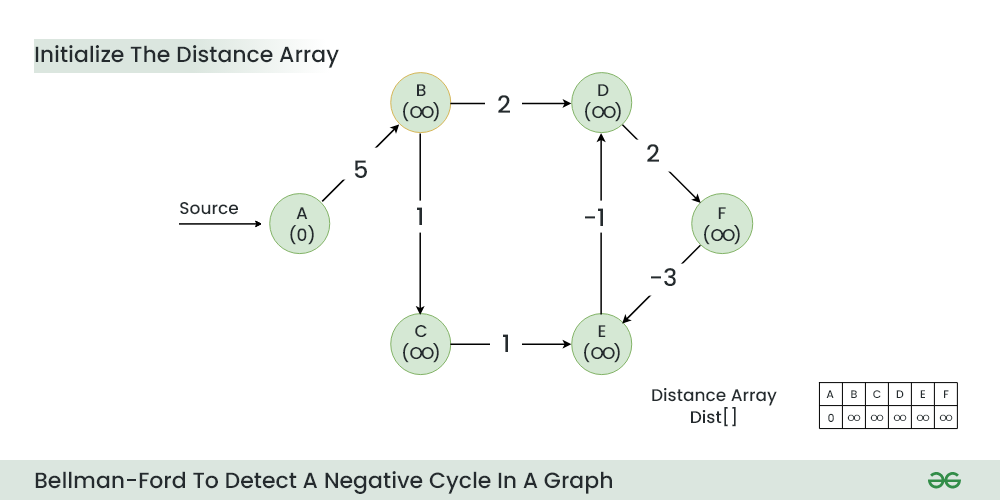
## **Working of Bellman-Ford Algorithm to Detect the Negative cycle in the graph:**

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

**

*Initial Graph*

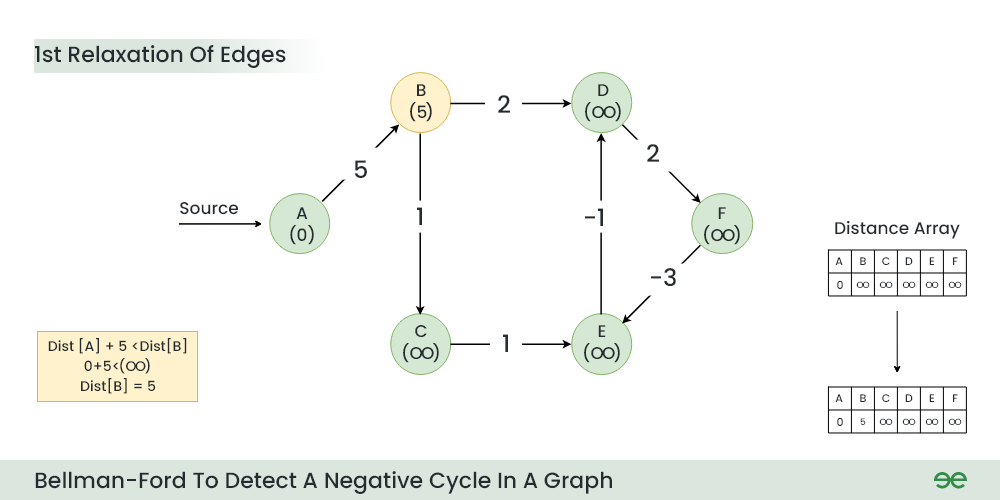
***Step 1:*** *Initialize a distance array Dist[] to store the shortest distance for each vertex from the source vertex. Initially distance of source will be 0 and Distance of other vertices will be INFINITY.*

**

*Initialize a distance array*

***Step 2:*** *Start relaxing the edges, during 1st Relaxation:*

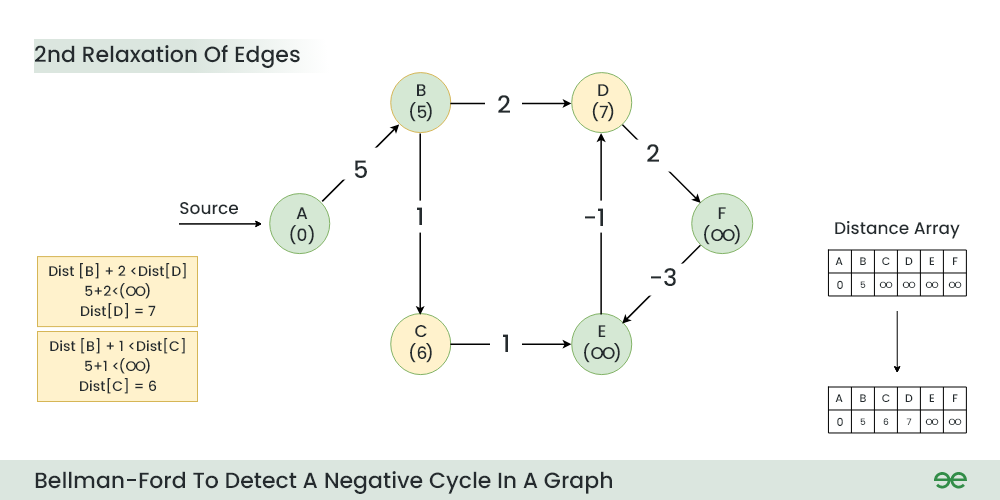
* *Current Distance of B > (Distance of A) + (Weight of A to B) i.e. Infinity > 0 + 5*
  + *Therefore, Dist[B] = 5*

**

*1st Relaxation*

***Step 3:*** *During 2nd Relaxation:*

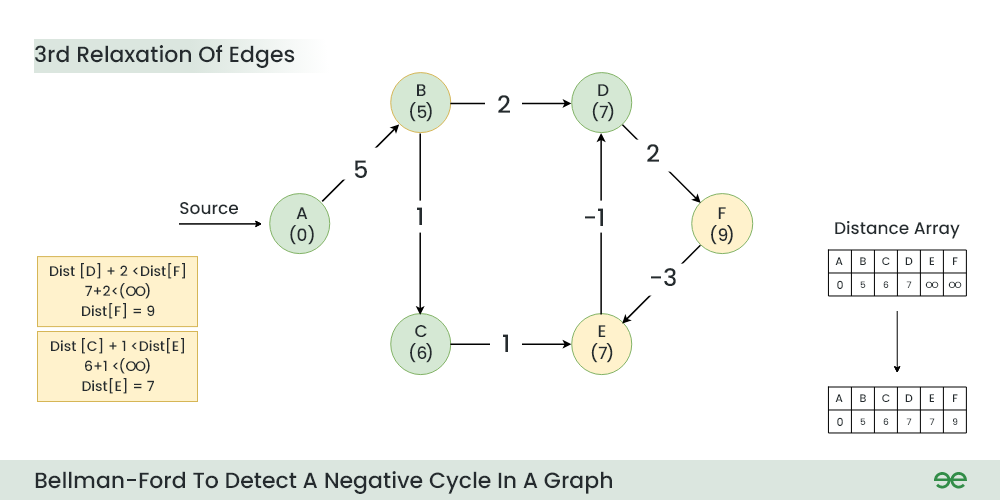
* *Current Distance of D > (Distance of B) + (Weight of B to D) i.e. Infinity > 5 + 2*
  + *Dist[D] = 7*
* *Current Distance of C > (Distance of B) + (Weight of B to C) i.e. Infinity > 5 + 1*
  + *Dist[C] = 6*

**

*2nd Relaxation*

***Step 4:*** *During 3rd Relaxation:*

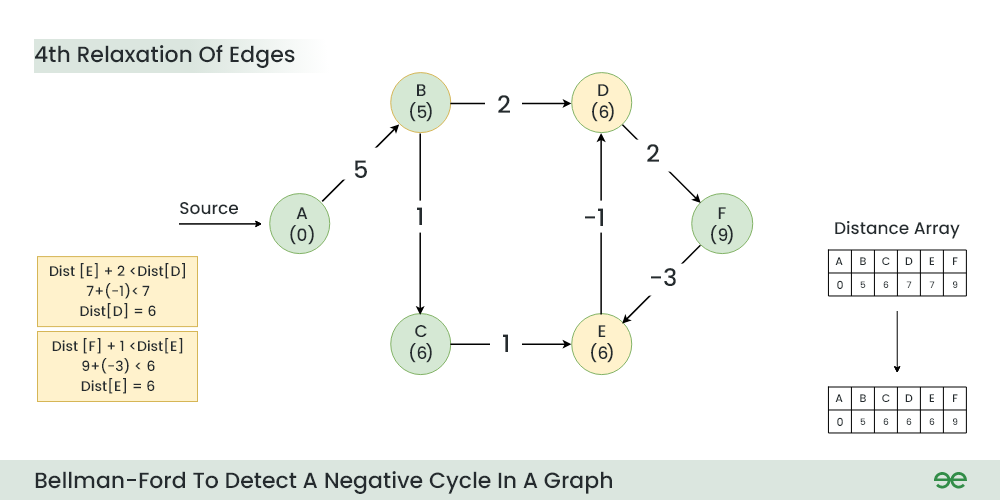
* *Current Distance of F > (Distance of D ) + (Weight of D to F) i.e. Infinity > 7 + 2*
  + *Dist[F] = 9*
* *Current Distance of E > (Distance of C ) + (Weight of C to E) i.e. Infinity > 6 + 1*
  + *Dist[E] = 7*

**

*3rd Relaxation*

***Step 5:*** *During 4th Relaxation:*

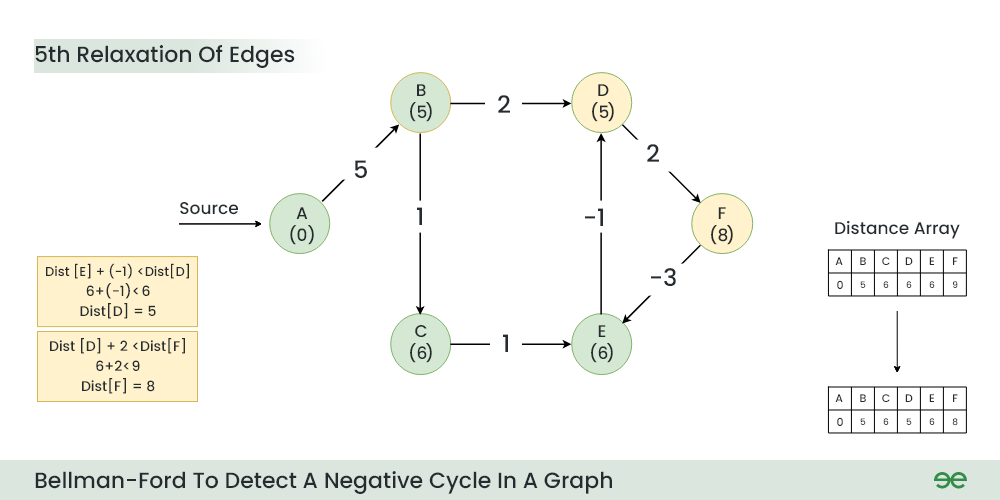
* *Current Distance of D > (Distance of E) + (Weight of E to D) i.e. 7 > 7 + (-1)*
  + *Dist[D] = 6*
* *Current Distance of E > (Distance of F ) + (Weight of F to E) i.e. 7 > 9 + (-3)*
  + *Dist[E] = 6*

**

*4th Relaxation*

***Step 6:*** *During 5th Relaxation:*

* *Current Distance of F > (Distance of D) + (Weight of D to F) i.e. 9 > 6 + 2*
  + *Dist[F] = 8*
* *Current Distance of D > (Distance of E ) + (Weight of E to D) i.e. 6 > 6 + (-1)*
  + *Dist[D] = 5*
* *Since the graph h 6 vertices, So during the 5th relaxation the shortest distance for all the vertices should have been calculated.*

**

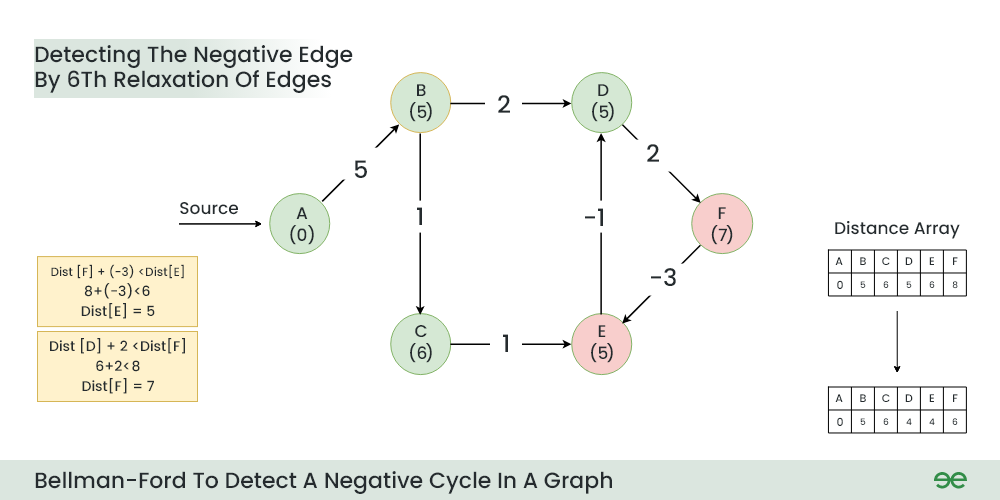
*5th Relaxation*

***Step 7:*** *Now the final relaxation i.e. the 6th relaxation should indicate the presence of negative cycle if there is any changes in the distance array of 5th relaxation.*

*During the 6th relaxation, following changes can be seen:*

* *Current Distance of E > (Distance of F) + (Weight of F to E) i.e. 6 > 8 + (-3)*
  + *Dist[E]=5*
* *Current Distance of F > (Distance of D ) + (Weight of D to F) i.e. 8 > 5 + 2*
  + *Dist[F]=7*

*Since, we observer changes in the Distance array Hence ,we can conclude the presence of a negative cycle in the graph.*

**

*6th Relaxation*

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

## **Algorithm to Find Negative Cycle in a Directed Weighted Graph Using Bellman-Ford:**

* Initialize distance array dist[] for each vertex ‘**v**‘ as **dist[v] = INFINITY**.
* Assume any vertex (let’s say ‘0’) as source and assign **dist = 0**.
* Relax all the **edges(u,v,weight) N-1** times as per the below condition:
  + **dist[v] = minimum(dist[v], distance[u] + weight)**
* Now, Relax all the edges one more time i.e. the **Nth** time and based on the below two cases we can detect the negative cycle:
  + Case 1 (Negative cycle exists): For any **edge(u, v, weight), if dist[u] + weight < dist[v]**
  + Case 2 (No Negative cycle) : case 1 fails for all the edges.

## **Handling Disconnected Graphs in the Algorithm:**

*The above algorithm and program might not work if the given graph is disconnected. It works when all vertices are reachable from source vertex* ***0****.  
To handle disconnected graphs, we can repeat the above algorithm for vertices having* ***distance = INFINITY,*** *or simply for the vertices that are not visited.*