Bellman-Ford algorithm:

1. This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
2. This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.  
   **a)** Do following for each edge u-v

If dist[v] > dist[u] + weight of edge uv, then update dist[v]

dist[v] = dist[u] + weight of edge uv

1. This step reports if there is a negative weight cycle in graph. Do following for each edge u-v

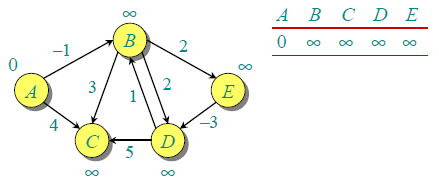
If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”

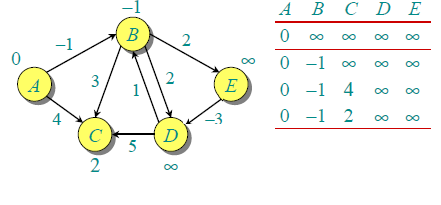
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn’t contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle.

Example:

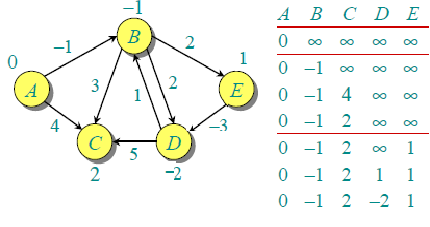
**Example**  
Let us understand the algorithm with following example graph. The images are taken from [this](http://www.cs.arizona.edu/classes/cs445/spring07/ShortestPath2.prn.pdf)source.

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so all edges must be processed 4 times.

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/bellman2.png)

Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.  
[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/After1stIteration.png)

The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).

[](http://d1gjlxt8vb0knt.cloudfront.net/wp-content/uploads/seconditeration2.png)

The second iteration guarantees to give all shortest paths which are at most 2 edges long. The algorithm processes all edges 2 more times. The distances are minimized after the second iteration, so third and fourth iterations don’t update the distances.

Complexity :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Negative edges? | Negative  cycle | Time complexity |
| Dijkstra | Single source | NO | NO | O(V lg V + E) |
| Bellman-Ford | Single source | YES | YES | O ( | V | | E | ) |

Chart of experimental values:

|  |  |  |
| --- | --- | --- |
| No of Nodes | Dijkstra’s (sec) | Bellman ford’s (sec) |
| 1000 | 0.02354 | 0.02355 |
| 5000 | 0.5337 | 0.5539 |
| 10000 | 2.1728 | 2.3027 |
| 15000 | 5.04 | 5.71 |
| 20000 | 9.1839 | 9.5033 |
| 30000 | 20.73 | 23.2003 |
| 50000 | 55.3941 | 75.4112 |