Bellman Ford Algorithm (Simple Implementation)

Given a graph and a source vertex *src *in graph, find shortest paths from *src *to all vertices in the given graph. The graph may contain negative weight edges.
We have discussed Dijkstra's algorithm for this problem. Dijkstra's algorithm is a Greedy algorithm and time complexity is O(V+E LogV) (with the use of Fibonacci heap). *Dijkstra doesn't work for Graphs with negative weight edges, Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is O(VE), which is more than Dijkstra.*

1) This step initializes distances from the source to all vertices as infinite and distance to the 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

3) 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 the shortest distances if the graph doesn't contain a 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
How does this work? Like other Dynamic Programming Problems, the algorithm calculates shortest paths in a bottom-up manner. It first calculates the shortest distances which have at-most one edge in the path. Then, it calculates the shortest paths with at-most 2 edges, and so on. After the i-th iteration of the outer loop, the shortest paths with at most i edges are calculated. There can be maximum |V| – 1 edges in any simple path, that is why the outer loop runs |v| – 1 times. The idea is, assuming that there is no negative weight cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most (i+1) edges ====> Very Important

```
import sys
class Solution:
    def isNegativeWeightCycle(self, n, edges):
        #Code here
        cost = [float("Inf")]*n
        cost[0] = 0
        for i in range(n-1):
```

```
            for info in edges:
                u = info[0]
                v = info[1]
                wt = info[2]
                if cost[u]!=float("Inf")and cost[u]+wt<cost[v]:
                    cost[v] = cost[u]+wt
        #Checks for negative cycles.If -ve cycle is present then BF can't
be used.
        for info in edges:
                u = info[0]
                v = info[1]
                wt = info[2]
                if cost[u]!=float("Inf") and cost[u]+wt<cost[v]:
                    return 1
        return 0
```