Bellman Ford Algorithm

This Algorithm is type from Dynamic programming because to try to found short path from source /all Node in graph and if graph contain negative weight that meaning some shortest path maybe not exist because FB is procedure used to find all shortest path in graph to order compare from source with all other nodes in graph so the algorithm not contain any negative cycle length and FB algorithm is rely on the condition is called relax operation and relax procedure takes two nodes as Parameter

Dijkstra not work with negative cycle length but FB is simpler than Dijkstra so time complexity of FB type from DP (O(VE)) more than Dijkstra type from Greedy algorithm (O(VLogV))

IF Distance from the source to First Node (A) + edge.length is less than Distance to the second node than the first node is denoted as the predecessor of the second node and the distance to the second node is recalculated (distance(A) + edge.length)

Detailed steps:

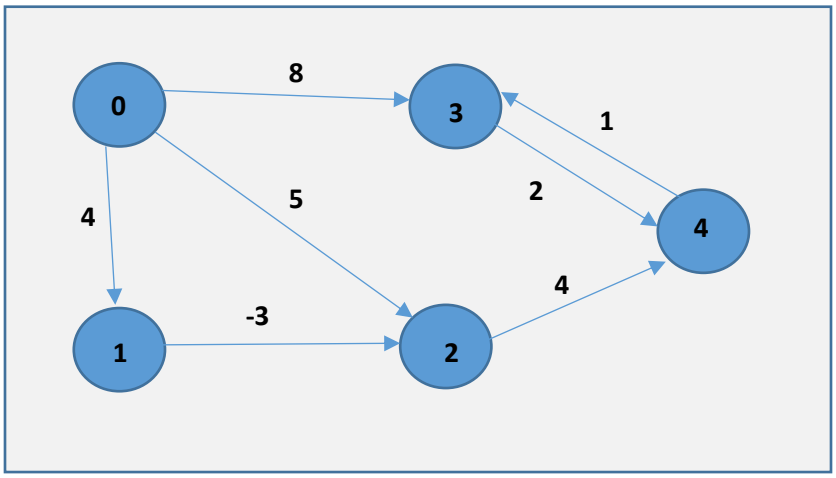
1- This step initializes distances from source to all vertices as infinite
2- Distance to source itself equal 0 and create array dist [ ] of size vertex with all values is infinite except the distance to source itself and must total number of vertices in the graph is for example 5 so all edges must be processed 4 times
3- This step calculates shortest distances
If distances [vertex] > distances [source] + weight of edge then update distances [source] distances [vertex] = distances [source] + weight of edge

4- This step reports if there is NCL in graph.
If distances [vertex] > distances [source] + weight of edge
Then error its graph contains negative weight cycle

The detailed of step 4 is step 3 if graph doesn’t contain NCL

Example:

The graph may contain negative weight edges and Dijkstra algorithm is greedy algorithm and time complexity is O(vlogv) so Dijkstra doesn't work for graph s with negative weight edges and Bellman works for each for such graph and it's also simpler than Dijkstra and suit well for distribution systems but time complexity of ford bellman is O(VE) which is more than Dijkstra



Random Order	
U	V
3	4
4	3
2	4
0	2
1	2
0	3
0	1

Vertex	Distance	Parent
0	0	-
1	∞	-
2	∞	-
3	∞	-
4	∞	-
5	∞	-

Complexity:

Space: O (V)
Time: O (V.E)
Short Path: O(V2.E)
Short Path in worst Case: O(V4.E)

Start v = 1:

3 → 4 = INF + 2 < ∞ - wrong ✗

0 → 2 = 0 + 5 < 2 - correct ✓ -> Then vertex = 2 dis = 5 parent = 0

4 → 3 = INF + 2 < ∞ - wrong ✗

0 → 3 = 0 + 8 < ∞ - correct ✓ -> Then vertex = 3 dis = 8 parent = 0

4 → 3 = 10 + 1 < 8 - wrong ✗

Start v = 2:

3 → 4 = 8 + 2 < ∞ - correct ✓ -> Then vertex = 4 dis = 10 parent = 3

0 → 2 = 0 + 5 < 5 - wrong ✗

2 → 4 = 5 + 4 < 10 - correct ✓ -> Then vertex = 4 dis = 9 parent = 2

0 → 3 = 0 + 8 < 8 - wrong ✗

1 → 2 = 4 - 3 < 5 - correct ✓ -> Then vertex = 2 dis = 1 parent = 1

0 → 1 = 0 + 4 < 4 - wrong ✗

4 → 3 = 9 +1 < 8 - wrong ✗

Start v = 3:

3 → 4 = 8 + 2 < 9 - wrong ✗

2 → 4 = 1 + 4 < 9 - correct ✓ -> Then vertex = 4 dis = 5 parent = 2

0 → 2 = 0 + 5 < 1 - wrong ✗

1 → 2 = 4 - 3 < 1 - wrong ✗

0 → 3 = 0 + 8 < 8 - wrong ✗

0 → 4 = 0 + 1 < 5 - correct ✓ -> Then vertex = 4 dis = 5 parent = 0

Start v = 4:

3 → 4 = 8 + 2 < 9 - wrong X

4 → 3 = 5 + 1 < 8 - correct √ -> Then vertex = 3 dis = 6 parent = 4

2 → 4 = 1 + 4 < 5 - wrong X

0 → 2 = 0 + 5 < 1 - wrong X

1 → 2 = 4 - 3 < 1 - wrong X

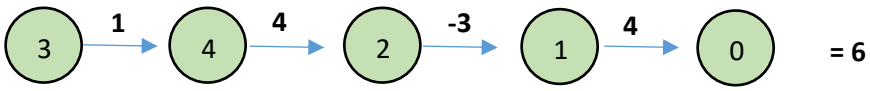
0 → 3 = 0 + 8 < 6 - wrong X

0 → 1 = 0 + 4 < 4 - wrong X

After Finished All Iteration (v)

Vertex	Distance	Parent
0	0	Null
1	4	0
2	1	1
3	6	4
4	5	2

For example if you need Go to 



= 6

Bellman Ford Code

```
Public void FB (int start, int next [ ][ ]) {
        For (int i = 1; i <= vertex; node++){
              distance [i] = max_value
        }
        distance [start] = 0;
        For (int i = 1; i <= vertex - 1; node++) {
            For (int j = 1; j <= vertex; j++) {
                For (int k = 1; k <= vertex; k ++) {
                    If (next [j][k] != max_value) {
                        If (distance [k] > distances [j] + next [j][k])
                            distance [k] = distances [j] + next [j][k];  }
                    }
                }
            }
        For (int j = 1; j <= vertex; j++) {
                For (int k = 1; k <= vertex; k++){
                    If (next [j][k] != max_value) {
                        If (distance [k] > distances [j] +next [j][k]
                            system.out.println("This Graph is NCL");
                        }
                    }
                }
```