# Bellman Ford Algorithm

## what is Bellman Ford Algorithm ?

***Bellman ford algorithm*** is a single-source shortest path algorithm. ***The bellman-Ford algorithm*** helps to find the shortest distance from the source node to all the other nodes in the graph. But, we have already learned ***Dijkstra's algorithm*** to fulfill the same purpose. Now, the question is ***how this algorithm is different from Dijkstra's algorithm***.

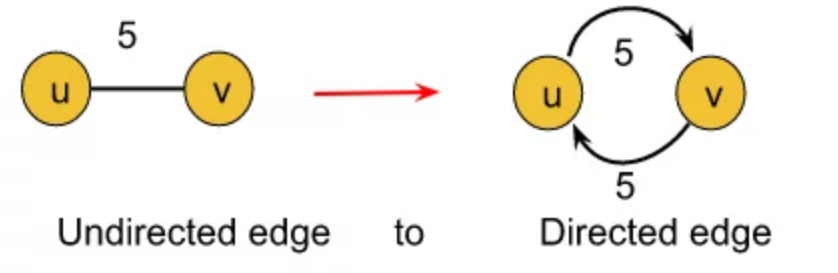
While learning Dijkstra's algorithm, we came across the following two situations, where Dijkstra's algorithm failed:

- ***If the graph contains negative edges.***
- ***If the graph has a negative cycle (In this case Dijkstra's algorithm fails to minimize the distance, keeps on running, and goes into an infinite loop. As a result it gives TLE error).***

***Negative Cycle:*** *A cycle is called a negative cycle if the sum of all its weights becomes negative. The following illustration is an example of a negative cycle:*


***Bellman-Ford's algorithm*** successfully solves these problems. The ***Bellman-Ford algorithm*** can handle graphs with negative edges and negative cycles, which Dijkstra's algorithm cannot.

💡 ***Bellman-Ford's algorithm*** is only applicable on ***directed graphs. In order to apply this algorithm to an undirected graph, we just need to convert the undirected edges into directed edges like the following:***




## Drawbacks of Bellman ford algorithm:

- The bellman ford algorithm does not produce a correct answer if the sum of the edges of a cycle is negative.

## Rule of this algorithm:

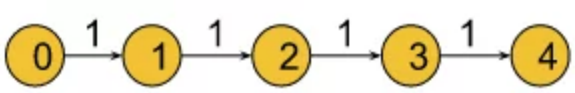
```
We will go on relaxing all the edges (n - 1) times sequentially,
where n = number of vertices.
```

## Relaxing means:

```
// distance is our distance array.
If (distance[u] + edge weight(u -> v) < distance[v])
{
      distance[v] = distance[u] + edge weight(u -> v);
}
```

## Why we are relaxing the edges only n-1 times ?

Let's try to first understand this using an example:




| Checking in each iteration | Given order of the edges: u | v | wt |
|---|---|---|---|
| dist[3] + 1 < dist[4] | 3 | 4 | 1 |
| dist[2] + 1 < dist[3] | 2 | 3 | 1 |
| dist[1] + 1 < dist[2] | 1 | 2 | 1 |
| dist[0] + 1 < dist[1] | 0 | 1 | 1 |

In the above graph, the algorithm will minimize the distance of the ith node in the ith iteration like dist[1] will be updated in the 1st iteration, dist[2] will be updated in the 2nd iteration, and so on. So we will need a total of 4 iterations(i.e. N-1 iterations) to minimize all the distances as dist[0] is already set to 0.

## How to detect a negative cycle in the graph ?

- We know if we keep on rotating inside a negative cycle, the path weight will be decreased in every iteration. But according to our intuition, we should have minimized all the distances within N-1 iterations(that means, after N-1 iterations no relaxation of edges is possible).
- In order to check the existence of a negative cycle, we will relax the edges one more time after the completion of N-1 iterations. And if in that Nth iteration, it is found that further relaxation of any edge is possible, we can conclude that the graph has a negative cycle. Thus, the Bellman-Ford algorithm detects negative cycles.