MEDIUM

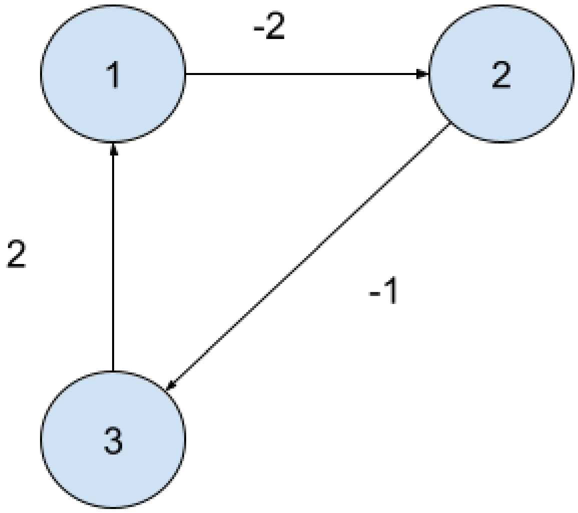
Bellman Ford Algorithm

Intuition

It's used to find the minimum distance from source to all nodes but it is also having a functionality to detect negative weight cycle as well.

It is applicable on Directed Graphs only unlike DAG in case of Dijkstra Algorithm.

Eg.



In this case if we use Dijkstra,

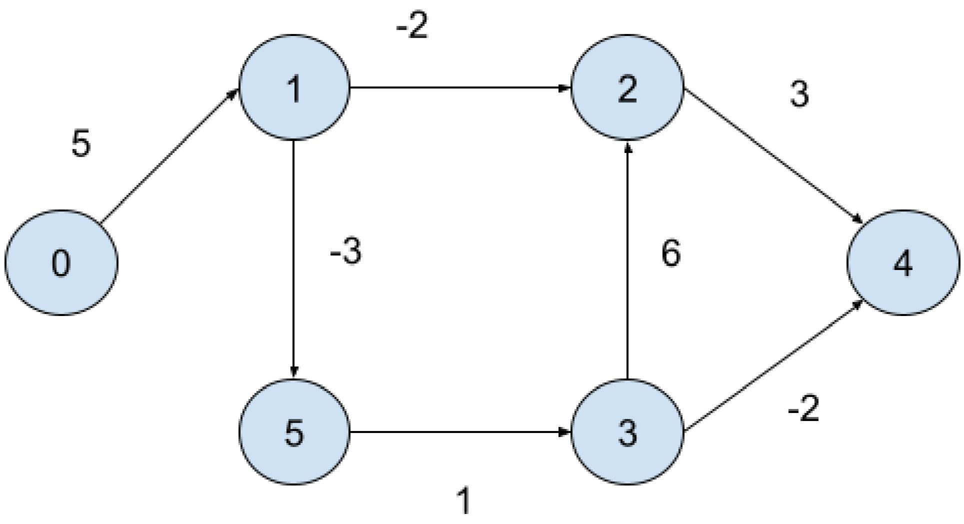
Then for the first time we will get :

Path Weight = -2+-1+2 = -1

But we can traverse again and this time we will get a smaller path weight and thus we will run into an infinite loop using the Dijkstra's Algorithm

Bellman Ford is an updated version of the Original Dijkstra Algorithm that can be used to detect the negative weight cycles in a graph.

Eg.



We need to have all edges ,they can be in any order

(u,v,wt)
(3,2,6)
(5,3,1)
(0,1,5)
(1,5,-3)
(1,2,-2)
(3,4,-2)
(2,4,3)

Bellman ford states that we need to Relax every edge ie. for N-1 times sequentially

Relax : If we have a distance array then

```
        if(distance[u]+wt<distance[v])
                {
                        Distance[v] = distance[u] + wt;
                }
```

This is known as the relaxation of edges

Initial Configuration

Distance :

0	inf	inf	inf	inf	inf

I == 0

0	5	3	inf	6	2

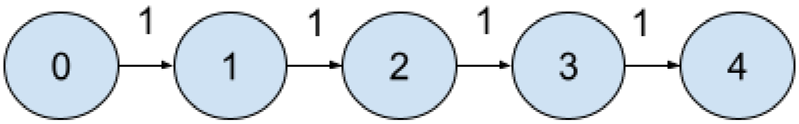
I == 1

0	5	3	3	1	2

.
.
.
i==5

Why n-1 iterations ?

- Edges can be in any order
- Imagine we have edges in particular order for the given graph :



(3,4,1)
(2,3,1)
(1,2,1)
(0,1,1)

Initial

0	inf	inf	inf	inf

0

0	1	inf	inf	inf

1

0	1	2	inf	inf

2

0	1	2	3	inf

4

0	1	2	3	4

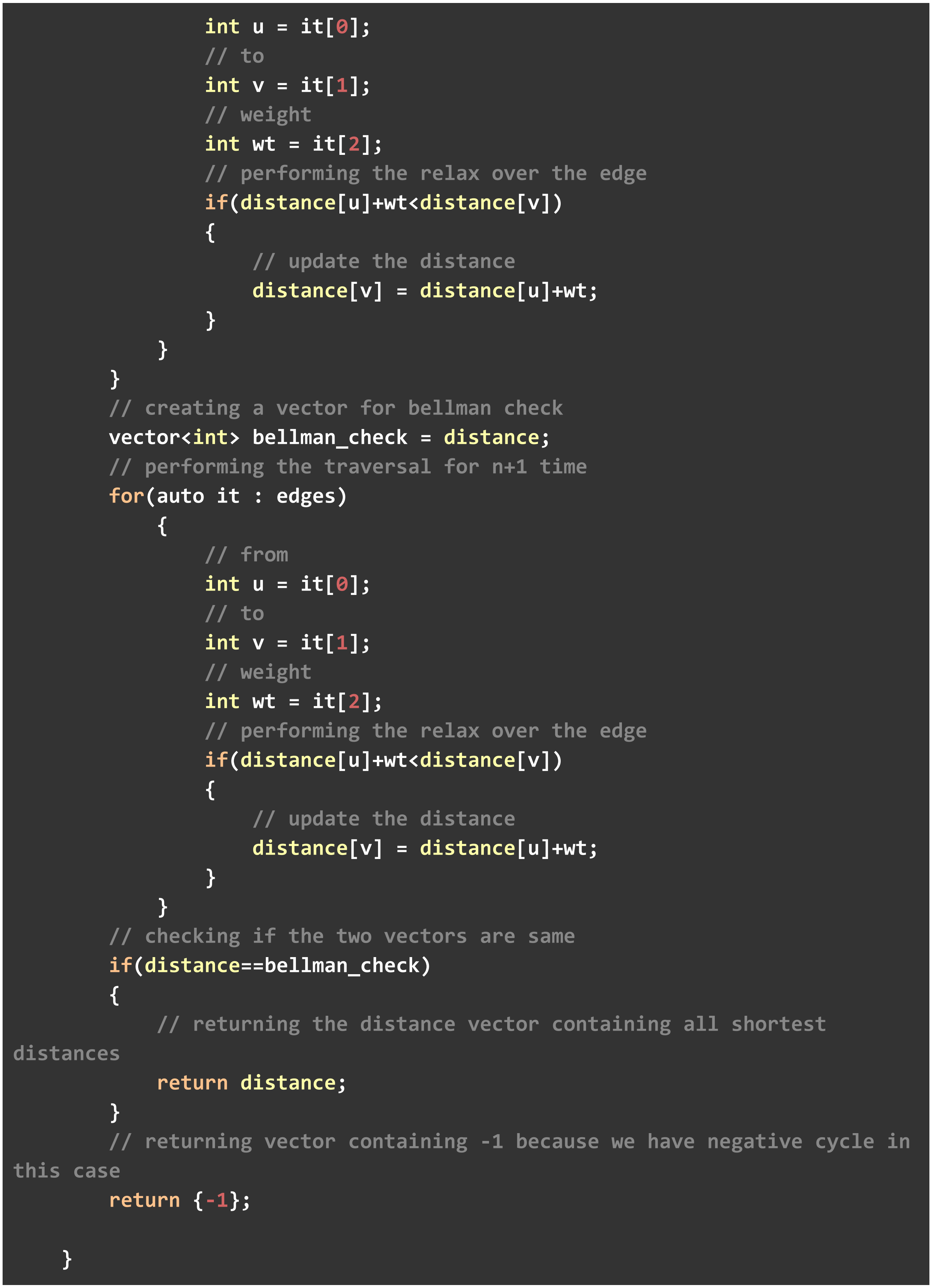
If there is a negative cycle if we do a nth iteration the values will still reduce and if the distance vector gets updated then we will have a negative weight cycle in our graph.

Approach

- Create a distance vector containing 1e8 initially for all nodes
- Mark the source to have distance 0
- Traverse for n times ie. 0-n-1 :
    - Traverse for all edges and perform relax ie:
        - Check if distance[from]+weight < distance[to] :
            - Update the distance distance[to] = distance[from]+weight
- Store the distance vector into another vector ie. bellman check vector
- Traverse for all edges and perform relax ie :
    - Check if distance[from]+weight < distance[to] :
        - Update the distance distance[to] = distance[from]+weight
- Check if distance and bellman check vectors are same :
    - Return distance vector because it has shortest distances
- Return vector containing -1 as in this case we will have negative weight cycle

Function Code

```
vector<int> bellman_ford(int n, vector<vector<int>>& edges, int source) {
        // Creating a distance vector which is initially infinite for all
        vector<int> distance(n,100000000);
        // Setting the source node distance to 0
        distance[source] = 0;
        // traversing for n times
        for(int i=0;i<n;i++)
        {
            // traversing for all edges and applying relax
            for(auto it : edges)
            {
                // from
```



```
                int u = it[0];
                // to
                int v = it[1];
                // weight
                int wt = it[2];
                // performing the relax over the edge
                if(distance[u]+wt<distance[v])
                {
                    // update the distance
                    distance[v] = distance[u]+wt;
                }
            }
        }
        // creating a vector for bellman check
        vector<int> bellman_check = distance;
        // performing the traversal for n+1 time
        for(auto it : edges)
            {
                // from
                int u = it[0];
                // to
                int v = it[1];
                // weight
                int wt = it[2];
                // performing the relax over the edge
                if(distance[u]+wt<distance[v])
                {
                    // update the distance
                    distance[v] = distance[u]+wt;
                }
            }
        // checking if the two vectors are same
        if(distance==bellman_check)
        {
            // returning the distance vector containing all shortest
distances
            return distance;
        }
        // returning vector containing -1 because we have negative cycle in
this case
        return {-1};

    }
```

Time Complexity

O(N*E)