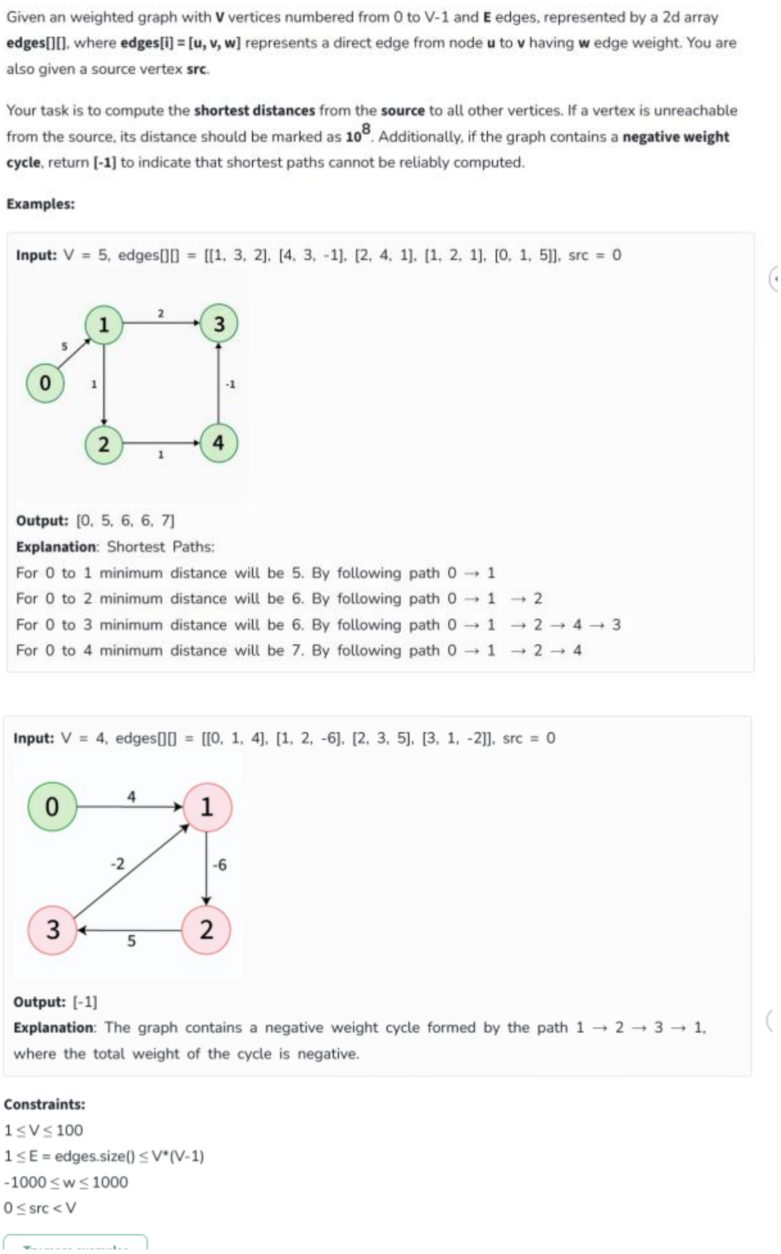
# Bellman - Ford :-



Given an weighted graph with **V** vertices numbered from 0 to V-1 and **E** edges, represented by a 2d array **edges[][]**, where **edges[i] = [u, v, w]** represents a direct edge from node **u** to **v** having **w** edge weight. You are also given a source vertex **src**.

Your task is to compute the **shortest distances** from the **source** to all other vertices. If a vertex is unreachable from the source, its distance should be marked as **$10^8$**. Additionally, if the graph contains a **negative weight cycle**, return **[-1]** to indicate that shortest paths cannot be reliably computed.

**Examples:**

**Input:** V = 5, edges[][] = [[1, 3, 2], [4, 3, -1], [2, 4, 1], [1, 2, 1], [0, 1, 5]], src = 0

**Output:** [0, 5, 6, 6, 7]
**Explanation**: Shortest Paths:
For 0 to 1 minimum distance will be 5. By following path 0 → 1
For 0 to 2 minimum distance will be 6. By following path 0 → 1 → 2
For 0 to 3 minimum distance will be 6. By following path 0 → 1 → 2 → 4 → 3
For 0 to 4 minimum distance will be 7. By following path 0 → 1 → 2 → 4

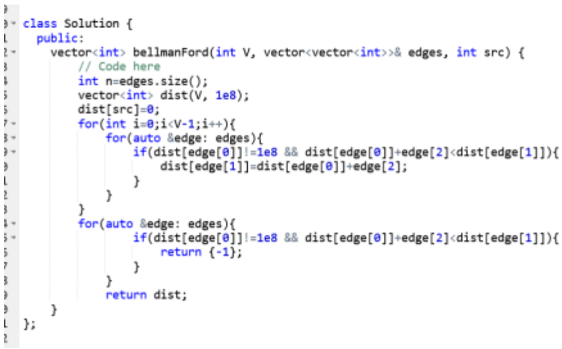
**Input:** V = 4, edges[][] = [[0, 1, 4], [1, 2, -6], [2, 3, 5], [3, 1, -2]], src = 0

**Output:** [-1]
**Explanation**: The graph contains a negative weight cycle formed by the path 1 → 2 → 3 → 1, where the total weight of the cycle is negative.

**Constraints:**
1 ≤ V ≤ 100
1 ≤ E = edges.size() ≤ V*(V-1)
-1000 ≤ w ≤ 1000
0 ≤ src < V

Concept :-

- Relax all the edges V-1 times.
- Relax one more to check for negative cycle.



```
class Solution {
  public:
    vector<int> bellmanFord(int V, vector<vector<int>>& edges, int src) {
        // Code here
        int n=edges.size();
        vector<int> dist(V, 1e8);
        dist[src]=0;
        for(int i=0;i<V-1;i++){
            for(auto &edge: edges){
                if(dist[edge[0]]!=1e8 && dist[edge[0]]+edge[2]<dist[edge[1]]){
                    dist[edge[1]]=dist[edge[0]]+edge[2];
                }
            }
        }
        for(auto &edge: edges){
                if(dist[edge[0]]!=1e8 && dist[edge[0]]+edge[2]<dist[edge[1]]){
                    return {-1};
                }
            }
            return dist;
    }
};
```

T.C . O(V.E)

S.C. O(V)

```
class Solution{
public:
    vector<int> BellmanFord(int V, vector<vector<int>>& edges, int src){
            int n= edges.size();
            vector<int> dist( V, 1e8);
            dist[src]= 0;
            for (int i=0; i<V-1; i++){
                    for( auto & edge : edges){
            if(dist(edge[0]] != 1e8 && dist[edge[0]] + edge[2]< dist[edge[1]])
            { dist[edge(2)] = dist[edge [0]] + edge (2); } } }

            for( auto & edge: edges){
            if (dist[edge[0]] != 1e8 && dist[edge[0]] + edge(2] < dist(edge[1]))
                { return {-1}; }

                    }};
```