Bellman-Ford in C++

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
        /*  Function to implement Bellman Ford
        *   edges: vector of vectors which represents the
graph
        *   S: source vertex to start traversing graph with
        *   V: number of vertices
        */
        vector<int> bellman_ford(int V,
vector<vector<int>>& edges, int S) {
                vector<int> dist(V, 1e8);
                dist[S] = 0;
                for (int i = 0; i < V - 1; i++) {
                        for (auto it : edges) {
                                int u = it[0];
                                int v = it[1];
                                int wt = it[2];
                                if (dist[u] != 1e8 &&
dist[u] + wt < dist[v]) {
                                        dist[v] = dist[u] +
wt;
                                }
                        }
                }
                // Nth relaxation to check negative cycle
                for (auto it : edges) {
                        int u = it[0];
                        int v = it[1];
                        int wt = it[2];
                        if (dist[u] != 1e8 && dist[u] + wt
< dist[v]) {
                                return { -1};
                        }
                }


                return dist;
        }
};


int main() {

        int V = 6;
        vector<vector<int>> edges(7, vector<int>(3));
        edges[0] = {3, 2, 6};
        edges[1] = {5, 3, 1};
        edges[2] = {0, 1, 5};
        edges[3] = {1, 5, -3};
        edges[4] = {1, 2, -2};
        edges[5] = {3, 4, -2};
        edges[6] = {2, 4, 3};

        int S = 0;
        Solution obj;
        vector<int> dist = obj.bellman_ford(V, edges, S);
        for (auto d : dist) {
                cout << d << " ";
```

Initialization

Vertex	dist
0	0
1	∞
2	∞
3	∞
4	∞
5	∞

📊 After each iteration of relaxation (V-1 = 5 times):

We'll update dist[] step by step, showing changes caused by each edge.

🔁 Iteration 1:

Process edges:

1. 0→1 (5) → dist[1] = 5
2. 1→2 (-2) → dist[2] = 3
3. 1→5 (-3) → dist[5] = 2
4. 5→3 (1) → dist[3] = 3
5. 3→4 (-2) → dist[4] = 1
6. 2→4 (3) → already dist[4] = 1 so not updated
7. Other edges don't apply yet.

Result:

dist = [0, 5, 3, 3, 1, 2]

🔁 Iteration 2 to 5:

Now that distances are optimal and no further relaxation improves any values, **no changes happen**.

✅ Final dist[] after Bellman-Ford

Vertex	Final dist
0	0
1	5
2	3
3	3
4	1
5	2

} cout << endl; return 0; }	✅ **Correct Output:** 0 5 3 3 1 2
Output:- 0 5 3 3 1 2	