**Bellman-Ford Algorithm**

**Medium**Accuracy: 56.77% Submissions: 3895 Points: 4

Given a weighted, directed and connected graph of V vertices and E edges, Find the shortest distance of all the vertex's from the source vertex S.  
**Note:**The Graph doesn't contain any negative weight cycle.

**Example 1:**

**Input:**

A picture containing bubble chart

Description automatically generated

**S** = 0

**Output:**

0 9

**Explanation**:

Shortest distance of all nodes from

source is printed.

**Example 2:**

**Input:**

Diagram

Description automatically generated

**S** = 2

**Output:**

1 6 0

**Explanation**:

For nodes 2 to 0, we can follow the path-

2-0. This has a distance of 1.

For nodes 2 to 1, we cam follow the path-

2-0-1, which has a distance of 1+5 = 6,

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **bellman\_ford()**  which takes number of vertices Vandan E sized list of lists of three integers where the three integers are u,v, and w; denoting there's an edge from u to v, which has a weight of w as input parameters and returns a list of integers where the ith integer denotes the distance of ith node from source node. If some node isn't possible to visit, then it's distance should be 100000000(1e8).

**Expected Time Complexity:** O(V\*E).  
**Expected Auxiliary Space:** O(V).

**Constraints:**  
1 ≤ V ≤ 500  
1 ≤ E ≤ V\*(V-1)  
-1000 ≤ adj[i][j] ≤ 1000  
0 ≤ S < V

class Solution{

    public:

    /\*  Function to implement Dijkstra

    \*   adj: vector of vectors which represents the graph

    \*   S: source vertex to start traversing graph with

    \*   V: number of vertices

    \*/

    void bfs(vector< vector<pair<int, int>> > &graph, vector<int> &distance, int S) {

        queue<int> q;

        vector<bool> visited(graph.size(), false);

        q.push(S);

        while (!q.empty()) {

            int p=q.front();

            q.pop();

            for (auto x : graph[p]) {

                if (x.first!=S) distance[x.first]=min(distance[x.first], distance[p]+x.second);

                if (!visited[x.first]) {

                    q.push(x.first);

                    visited[x.first]=true;

                }

            }

        }

    }

    vector<int> bellman\_ford(int V, vector<vector<int>> adj, int S) {

        // Code here

        vector<int> distance(V, 100000000);

        distance[S]=0;

        vector< vector<pair<int, int>> > graph(V);

        for (int j=0; j<adj.size(); j++) {

            vector<int> v=adj[j];

            graph[v[0]].push\_back({v[1], v[2]});

        }

        for (int i=1; i<V/2; i++) {

            bfs(graph, distance, S);

        }

        return distance;

    }

};