```cpp
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

// SL.No.: - 31
// Admission No.: - 21JE0269
// Name: - Chotaliya Zeel Vijaybhai

const int INF = 100;  // Using 100 as infinity as per the assignment
const int NODES = 7;

class BellmanFordAlgorithm {
private:
    vector<vector<int>> cost_matrix;
    vector<vector<int>> distance_matrix;

public:
    BellmanFordAlgorithm() : cost_matrix(NODES, vector<int>(NODES)),
                             distance_matrix(NODES, vector<int>(NODES)) {}

    void input() {
        for (int i = 0; i < NODES; ++i) {
            for (int j = i; j < NODES; ++j) {
                cout << "Enter cost (max:100) of the node " << j << " from " << i << endl;
                cin >> cost_matrix[i][j];
                cost_matrix[j][i] = cost_matrix[i][j];  // Symmetric matrix
            }
        }
    }

    void bellmanFord(int root) {
        vector<int>& distances = distance_matrix[root];  // 'd' matrix as per assignment

        // Initialize distances
        for (int i = 0; i < NODES; ++i) {
            distances[i] = INF;
        }
        distances[root] = 0;

        // Relax all edges |V|-1 times
        for (int i = 1; i < NODES; ++i) {
            // For each edge (u,v)
            for (int u = 0; u < NODES; ++u) {
                for (int v = 0; v < NODES; ++v) {
                    if (cost_matrix[u][v] != INF && distances[u] != INF &&
                        distances[u] + cost_matrix[u][v] < distances[v]) {
                        distances[v] = distances[u] + cost_matrix[u][v];
                    }
                }
            }
        }

        // Check for negative weight cycles
        for (int u = 0; u < NODES; ++u) {
            for (int v = 0; v < NODES; ++v) {
                if (cost_matrix[u][v] != INF && distances[u] != INF &&
                    distances[u] + cost_matrix[u][v] < distances[v]) {
                    cout << "Graph contains negative weight cycle!" << endl;
                    return;
                }
            }
        }
    }

    void costupdate() {
        for (int root = 0; root < NODES; ++root) {
            bellmanFord(root);
        }
    }

    void display() {
```

```
        cout << "The input cost matrix:" << endl;
        for (const auto& row : cost_matrix) {
            for (int cost : row) {
                cout << setw(3) << cost << " ";  // 'setw()' function is used to specified
white space
            }
            cout << endl;
        }

        cout << "\nThe updated cost matrix:" << endl;
        for (const auto& row : distance_matrix) {
            for (int dist : row) {
                cout << setw(3) << dist << " ";  // 'setw()' function is used to specified
white space
            }
            cout << endl;
        }
    }
};

int main() {
    BellmanFordAlgorithm bellmanFord;
    bellmanFord.input();
    bellmanFord.costupdate();
    bellmanFord.display();
    return 0;
    }
```