**Floyd Warshall Algorithm**

**Medium**Accuracy: 44.25% Submissions: 17200 Points: 4

The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph. The Graph is represented as adjancency matrix, and the matrix denotes the weight of the edegs (if it exists) else -1. **Do it in-place.**

**Example 1:**

**Input:** matrix = {{0,25},{-1,0}}

**Output:** {{0,25},{-1,0}}

**Explanation:** The shortest distance between

every pair is already given(if it exists).

**Example 2:**

**Input:** matrix = {{0,1,43},{1,0,6},{-1,-1,0}}

**Output:** {{0,1,7},{1,0,6},{-1,-1,0}}

**Explanation:** We can reach 3 from 1 as 1->2->3

and the cost will be 1+6=7 which is less than

43.

**Your Task:**  
You don't need to read, return or print anything. Your task is to complete the function **shortest\_distance()**which takes the matrix as input parameter and modify the distances for every pair in-place.

**Expected Time Complexity:**O(n3)  
**Expected Space Compelxity:**O(1)

**Constraints:**  
1 <= n <= 100

class Solution {

  public:

    void shortest\_distance(vector<vector<int>> &matrix){

        // Code here

        int V=matrix.size();

        for (int i=0; i<V; i++) {

            for (int j=0; j<V; j++) {

                if (matrix[i][j]==-1) matrix[i][j]=INT\_MAX;

            }

        }

        for (int k=0; k<V; k++) {

            for (int i=0; i<V; i++) {

                for (int j=0; j<V; j++) {

                    if (matrix[i][k]==INT\_MAX or matrix[k][j]==INT\_MAX) continue;

                    matrix[i][j]=min(matrix[i][j], matrix[i][k]+matrix[k][j]);

                }

            }

        }

        for (int i=0; i<V; i++) {

            for (int j=0; j<V; j++) {

                if (matrix[i][j]==INT\_MAX) matrix[i][j]=-1;

            }

        }

    }

};