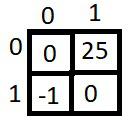
**Floyd Warshall**

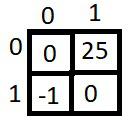
The problem is to find the shortest distances between every pair of vertices in a given **edge-weighted directed** graph. The graph is represented as an adjacency matrix of size **n\*n**. **Matrix[i][j]** denotes the weight of the edge from **i to j.**If **Matrix[i][j]=-1,**it means there is no edge from **i to j.**  
**Do it in-place.**

**Example 1:**

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



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

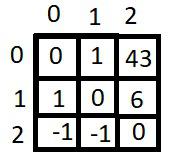


**Explanation:** The shortest distance between

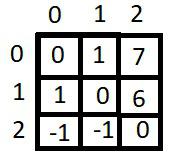
every pair is already given(if it exists).

**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 2 from 0 as 0->1->2

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 **modifies** the distances **for every pair in-place.**

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

**Constraints:**  
1 <= n <= 100  
-1 <= matrix[ i ][ j ] <= 1000