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.**
Note : Modify the distances **for every pair in-place.**

Examples :

Input: matrix = [[0, 25],[-1, 0]]

	0	1
0	0	25
1	-1	0

Output: [[0, 25],[-1, 0]]

	0	1
0	0	25
1	-1	0

Explanation: The shortest distance between every pair is already given(if it exists).

Input: matrix = [[0, 1, 43],[1, 0, 6],[-1, -1, 0]]

	0	1	2
0	0	1	43
1	1	0	6
2	-1	-1	0

Output: [[0, 1, 7],[1, 0, 6],[-1, -1, 0]]

	0	1	2
0	0	1	7
1	1	0	6
2	-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.

Expected Time Complexity: $O(n^3)$
Expected Space Complexity: O(1)

Constraints:

1 <= n <= 100

-1 <= matrix[i][j] <= 1000

Try more examples

```
class Solution:
    def shortest_distance(self, matrix):
        n = len(matrix)

        # Convert -1 to infinity for the purposes of Floyd-
Warshall algorithm
        dist = [[float('inf')] * n for _ in range(n)]

        # Initialize distances from the adjacency matrix
        for i in range(n):
            for j in range(n):
                if i == j:
                    dist[i][j] = 0
                elif matrix[i][j] != -1:
                    dist[i][j] = matrix[i][j]

        # Floyd-Warshall Algorithm
        for k in range(n):
            for i in range(n):
                for j in range(n):
                    if dist[i][k] != float('inf') and
dist[k][j] != float('inf'):
                        dist[i][j] = min(dist[i][j],
dist[i][k] + dist[k][j])

        for i in range(n):
            for j in range(n):
                if dist[i][j] == float('inf'):
                    dist[i][j] = -1
        for i in range(n):
            for j in range(n):
                matrix[i][j] = dist[i][j]
```