4. 14 To implement Floyd’s Algorithm to compute shortest paths between all pairs of cities, display the distance matrix before and after applying the algorithm, and identify the shortest path or city as required.

**AIM**

To implement Floyd’s Algorithm to compute shortest paths between all pairs of cities, display the distance matrix before and after applying the algorithm, and identify the shortest path or city as required.

**ALGORITHM**

**1**.**Start**

2.Input number of cities n and edges (u, v, w).

3.Initialize a distance matrix dist[n][n]:

* dist[i][i] = 0
* If edge exists, dist[u][v] = w and dist[v][u] = w (if undirected).
* Else, dist[i][j] = ∞.

4.Display initial matrix.

5.Apply Floyd’s Algorithm:

for k in range(n):

for i in range(n):

for j in range(n):

if dist[i][k] + dist[k][j] < dist[i][j]:

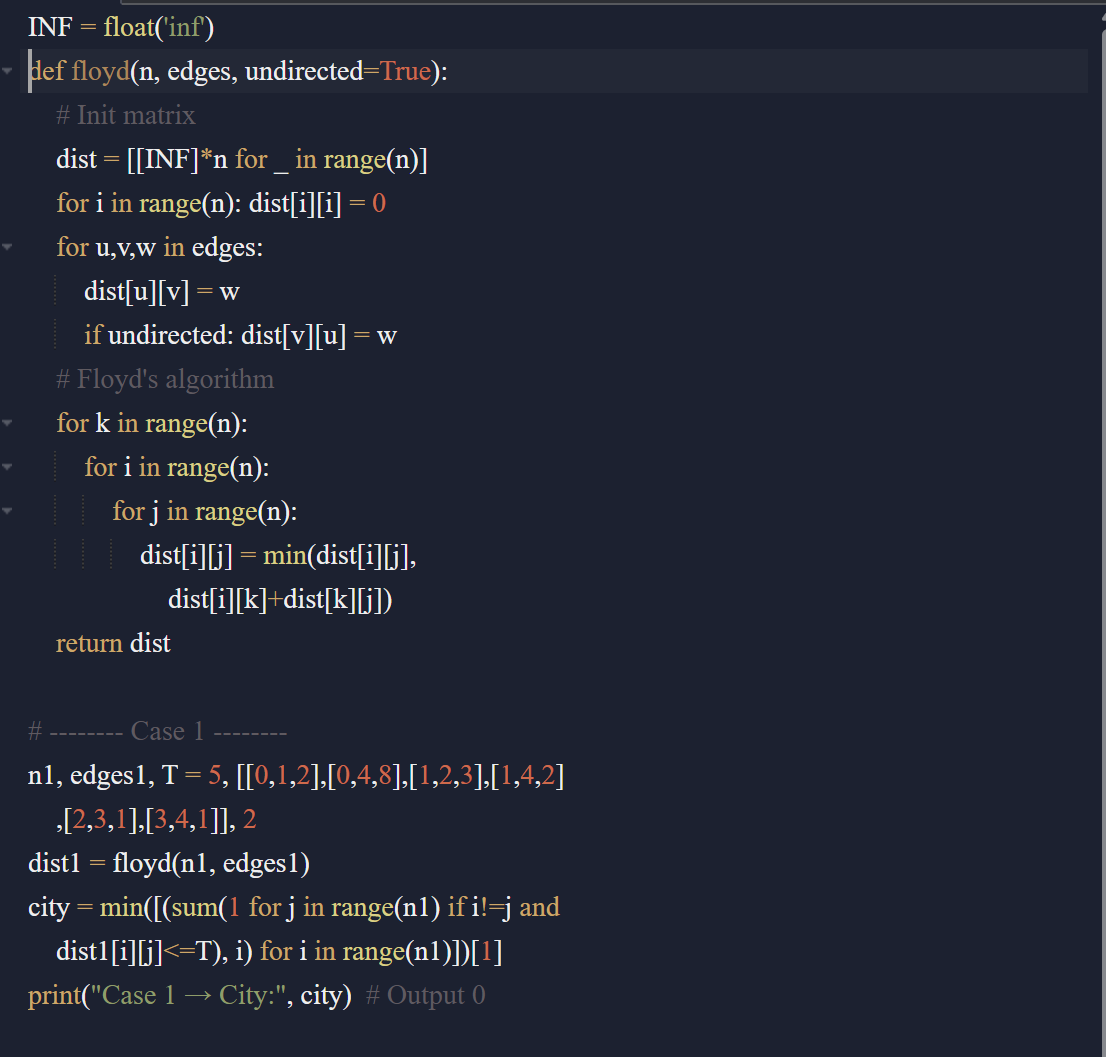
dist[i][j] = dist[i][k] + dist[k][j]

6.Display updated matrix.

7.For threshold problem: Count neighbors with distance ≤ threshold, select city with smallest count (tie → smallest index).

8.For test case: Output shortest path from source to destination.

**PROGRAM**



Input:

Distance Matrix BEFORE Floyd’s Algorithm:

[0, 2, inf, inf, 8]

[2, 0, 3, inf, 2]

[inf, 3, 0, 1, inf]

[inf, inf, 1, 0, 1]

[8, 2, inf, 1, 0]

Distance Matrix AFTER Floyd’s Algorithm:

[0, 2, 5, 6, 4]

[2, 0, 3, 4, 2]

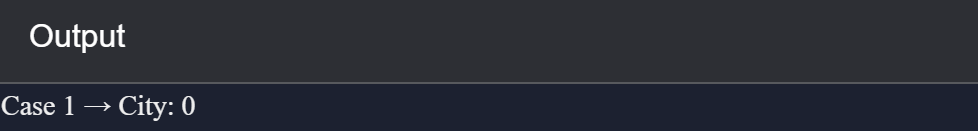
[5, 3, 0, 1, 2]

[6, 4, 1, 0, 1]

[4, 2, 2, 1, 0]

City with fewest neighbors within threshold 2: 0

Output:



**RESULT:**

* In **Case 1**, the city with the fewest neighbors within threshold = **City 0**.
* In **Case 2**, the shortest path distance from **City C → A** is **7**.

**PERFORMANCE ANALYSIS:**

* **Time Complexity: O(n³)**
* **Space Complexity: O(n²)**