183.Implement Floyd's Algorithm to find the shortest path between all pairs of cities. Display the distance matrix before and after applying the algorithm. Identify and print the shortest path

Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4

PROGRAM:

```
def floyd_warshall(n, edges, distanceThreshold):
    # Initialize the distance matrix with infinity
    inf = float('inf')
    dist = [[inf] * n for _ in range(n)]

    # Distance from a node to itself is 0
    for i in range(n):
        dist[i][i] = 0

    # Fill initial distances based on edges
    for u, v, w in edges:
        dist[u][v] = w
        dist[v][u] = w  # Assuming undirected graph; remove if directed

    print("Initial distance matrix:")
    for row in dist:
        print(row)

    # Floyd-Warshall Algorithm
    for k in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][j] > dist[i][k] + dist[k][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    print("\nDistance matrix after applying Floyd-Warshall algorithm:")
    for row in dist:
        print(row)

    # Print shortest paths within the distance threshold
    print("\nShortest paths within the distance threshold:")
    for i in range(n):
        for j in range(i + 1, n):
            if dist[i][j] <= distanceThreshold:
                print(f"Shortest path from {i} to {j} is {dist[i][j]}")

# Example usage
n = 4
edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1]]
distanceThreshold = 4

floyd_warshall(n, edges, distanceThreshold)
```

OUTPUT:

```
Initial distance matrix:
[0, 3, inf, inf]
[3, 0, 1, 4]
[inf, 1, 0, 1]
[inf, 4, 1, 0]

Distance matrix after applying Floyd-Warshall algorithm:
[0, 3, 4, 5]
[3, 0, 1, 2]
[4, 1, 0, 1]
[5, 2, 1, 0]

Shortest paths within the distance threshold:
Shortest path from 0 to 1 is 3
Shortest path from 0 to 2 is 4
Shortest path from 1 to 2 is 1
Shortest path from 1 to 3 is 2
Shortest path from 2 to 3 is 1

=== Code Execution Successful ===
```

TIME COMPLEXITY:O(N^3)