184.Write a Program to implement Floyd's Algorithm to calculate the shortest paths between all pairs of routers. Simulate a change where the link between Router B and Router D fails. Update the distance matrix accordingly. Display the shortest path from Router A to Router F before and after the link failure.

Input as above

Output : Router A to Router F = 5

PROGRAM:

```
def floyd_warshall(n, edges):

    # Initialize the distance matrix with infinity

    inf = float('inf')

    dist = [[inf] * n for _ in range(n)]


    # Distance from a router 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


    # 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]

    return dist

def simulate_link_failure(dist, u, v):
    # Set the distance to infinity to simulate link failure
    inf = float('inf')
    dist[u][v] = inf
    dist[v][u] = inf

    # Reapply Floyd-Warshall Algorithm
    n = len(dist)
    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]

def print_distance_matrix(dist):
    for row in dist:
        print(row)

def find_shortest_path(dist, src, dest):
    return dist[src][dest]
```

```
# Example usage

n = 6

edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1], [3, 4, 6], [4, 5, 2], [2, 5, 5]]

link_failure = (1, 3)


# Initial distances

dist = floyd_warshall(n, edges)


print("Distance matrix before link failure:")

print_distance_matrix(dist)


# Shortest path from A to F before link failure

print(f"\nShortest path from A to F before link failure:
{find_shortest_path(dist, 0, 5)}")


# Simulate link failure

simulate_link_failure(dist, *link_failure)


print("\nDistance matrix after link failure:")

print_distance_matrix(dist)


# Shortest path from A to F after link failure

print(f"\nShortest path from A to F after link failure:
{find_shortest_path(dist, 0, 5)}")
```

OUTPUT:

```
Distance matrix before link failure:
[0, 3, 4, 5, 11, 9]
[3, 0, 1, 2, 8, 6]
[4, 1, 0, 1, 7, 5]
[5, 2, 1, 0, 6, 6]
[11, 8, 7, 6, 0, 2]
[9, 6, 5, 6, 2, 0]

Shortest path from A to F before link failure: 9

Distance matrix after link failure:
[0, 3, 4, 5, 11, 9]
[3, 0, 1, 2, 8, 6]
[4, 1, 0, 1, 7, 5]
[5, 2, 1, 0, 6, 6]
[11, 8, 7, 6, 0, 2]
[9, 6, 5, 6, 2, 0]

Shortest path from A to F after link failure: 9

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

TIME COMPLEXITY:O(N^3)