187.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 floyds_algorithm(graph):
    n = len(graph)
    distance = graph

    for k in range(n):
        for i in range(n):
            for j in range(n):
                distance[i][j] = min(distance[i][j], distance[i][k] +
distance[k][j])

    return distance

def simulate_link_failure(graph, node1, node2):
    graph[node1][node2] = float('inf')
    graph[node2][node1] = float('inf')
    return graph

# Define the graph representing the network topology
graph = [
    [0, 3, 8, float('inf'), -4],
    [float('inf'), 0, float('inf'), 1, 7],
    [float('inf'), 4, 0, float('inf'), float('inf')],
    [2, float('inf'), -5, 0, float('inf')],
    [float('inf'), float('inf'), float('inf'), 6, 0]
]

# Applying Floyd's Algorithm to calculate shortest paths
distance_matrix = floyds_algorithm(graph)

# Simulating link failure between Router B and Router D
failed_graph = simulate_link_failure(distance_matrix, 1, 3)

# Displaying the shortest path from Router A to Router F before
link failure
```

```
print("Router A to Router F (Before link failure) =", distance_matrix[0][4])

# Displaying the shortest path from Router A to Router F after link failure
print("Router A to Router F (After link failure) =", floyds_algorithm(failed_graph)[0][4])
```

OUTPUT:

```
Router A to Router F (Before link failure) = -4
Router A to Router F (After link failure) = -4


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

TIME COMPELXITY:O(N^3)