4.12 FLOYD’S ALGORITHM FOR ALL-PAIRS SHORTEST PATH

Question:

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.

AIM

To implement Floyd’s Algorithm in C to compute the shortest paths between all pairs of cities and display the distance matrix before and after applying the algorithm.

ALGORITHM

1. Input the number of cities n and the distance matrix dist[n][n].
2. Initialize a path matrix path[n][n] to track intermediate nodes.
3. For each intermediate node k, update dist[i][j] as:
    - dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
4. Update path[i][j] if a shorter path is found via k.
5. After the algorithm, print the updated distance matrix.
6. Use the path matrix to reconstruct and print the shortest path between each pair.

PROGRAM

a)

```python
def floyd_warshall(n, edges):
    INF = float('inf')
    dist = [[INF] * n for _ in range(n)]

    for i in range(n):
        dist[i][i] = 0

    for u, v, w in edges:
        dist[u - 1][v - 1] = w

    print("\n Distance Matrix Before Floyd's Algorithm:")
    for row in dist:
        print(row)

    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]

    print("\n Distance Matrix After Floyd's Algorithm:")
    for row in dist:
        print(row)

    print(f"\n Shortest path from City 1 to City 3: {dist[0][2]}")


n = int(input("Enter number of cities: "))
m = int(input("Enter number of edges: "))
edges = []
print("Enter each edge as: from to weight (e.g., 1 2 3)")
for _ in range(m):
    u, v, w = map(int, input("Edge: ").split())
    edges.append([u, v, w])

floyd_warshall(n, edges)
```

Input:

Enter number of cities: 4
Enter number of edges: 8
Enter each edge as: from to weight (e.g., 1 2 3)

Edge: 1 2 3
Edge:1 3 8
Edge: 1 4 -4
Edge: 2 4 1
Edge: 2 3 4
Edge: 3 1 2
Edge: 4 3 -5
Edge: 4 2 6

Output:

```
Enter number of cities: 4
Enter number of edges: 8
Enter each edge as: from to weight (e.g., 1 2 3)
Edge: 1 2 3
Edge: 1 3 8
Edge: 1 4 -4
Edge: 2 4 1
Edge: 2 3 4
Edge: 3 1 2
Edge: 4 3 -5
Edge: 4 2 6

 Distance Matrix Before Floyd's Algorithm:
[0, 3, 8, -4]
[inf, 0, 4, 1]
[2, inf, 0, inf]
[inf, 6, -5, 0]

 Distance Matrix After Floyd's Algorithm:
[-7, -4, -9, -11]
[-2, 0, -4, -6]
[-5, -2, -7, -9]
[-10, -7, -12, -14]

 Shortest path from City 1 to City 3: -9
>>> |
```

b)

```python
def floyd_warshall(n, edges):
    INF = float('inf')
    dist = [[INF] * n for _ in range(n)]

    for i in range(n):
        dist[i][i] = 0

    for u, v, w in edges:
        dist[u][v] = w
        dist[v][u] = w

    print("\nDistance Matrix Before Floyd's Algorithm:")
    for row in dist:
        print(row)

    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]

    print("\nDistance Matrix After Floyd's Algorithm:")
    for row in dist:
        print(row)

    print(f"\nShortest path from Router A to Router F: {dist[0][5]}")


n = int(input("Enter number of routers: "))
m = int(input("Enter number of links: "))
edges = []
print("Enter each link as: from to cost (e.g., 0 1 5 for A-B)")
for _ in range(m):
    u, v, w = map(int, input("Link: ").split())
    edges.append([u, v, w])

floyd_warshall(n, edges)
```

Input

Enter number of routers: 6
Enter number of links: 8
Enter each link as: from to cost (e.g., 0 1 5 for A-B)

Edge: 0 1 1
Edge: 0 2 5
Edge: 1 2 2
Edge: 1 3 1
Edge: 2 4 3
Edge: 3 4 1
Edge: 3 5 6
Edge: 4 5 2

Output

```
Enter number of routers: 6
Enter number of links: 8
Enter each link as: from to cost (e.g., 0 1 5 for A-B)
Link: 0 1 1
Link: 0 2 5
Link: 1 2 2
Link: 1 3 1
Link: 2 4 3
Link: 3 4 1
Link: 3 5 6
Link: 4 5 2

Distance Matrix Before Floyd's Algorithm:
[0, 1, 5, inf, inf, inf]
[1, 0, 2, 1, inf, inf]
[5, 2, 0, inf, 3, inf]
[inf, 1, inf, 0, 1, 6]
[inf, inf, 3, 1, 0, 2]
[inf, inf, inf, 6, 2, 0]

Distance Matrix After Floyd's Algorithm:
[0, 1, 3, 2, 3, 5]
[1, 0, 2, 1, 2, 4]
[3, 2, 0, 3, 3, 5]
[2, 1, 3, 0, 1, 3]
[3, 2, 3, 1, 0, 2]
[5, 4, 5, 3, 2, 0]

Shortest path from Router A to Router F: 5
>>>
```

RESULT:

Thus search program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

· **Time Complexity:** $O(n^3)$

· **Space Complexity:** $O(n^2)$