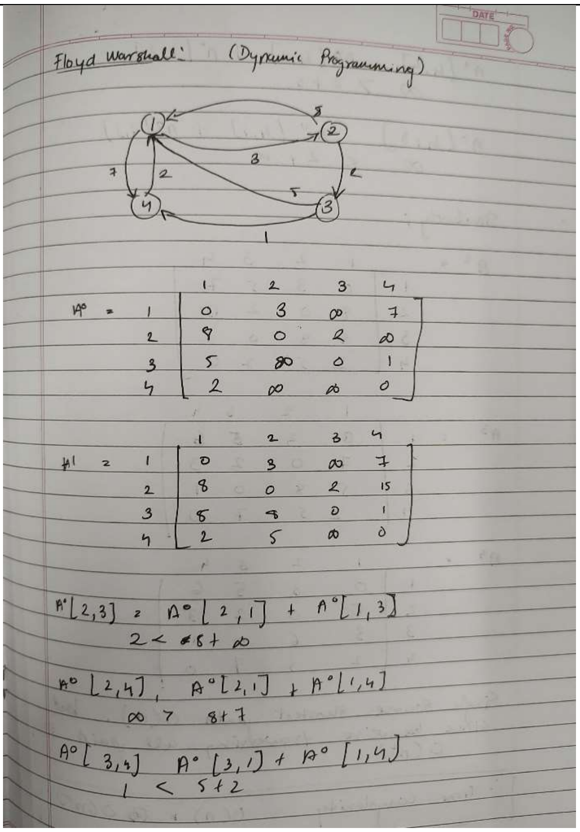
Bharatiya Vidya Bhavan's

SARDAR PATEL INSTITUTE OF TECHNOLOGY

(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Master of Computer Application

Experiment	6&7
Aim	To implement All Pair Shortest Path (Floydd Warshal Algorithm)
Objective	1) Learn All pair shortest path 2) Implement All pair shortest path 3) Derive the Time complexity Floydd Warshall algorithm
Name	Vivek Tiwari
UCID	2023510059
Class	FY MCA
Batch	C
Date of Submission	27-03-2024

Algorithm and Explanation of the technique used	**Algorithm**: For a graph with N vertices: Step 1: Initialize the shortest paths between any 2 vertices with Infinity. Step 2: Find all pair shortest paths that use 0 intermediate vertices, then find the shortest paths that use 1 intermediate vertex and so on.. until using all N vertices as intermediate nodes. Step 3: Minimize the shortest paths between any 2 pairs in the previous operation. Step 4: For any 2 vertices (i,j) , one should actually minimize the distances between this pair using the first K nodes, so the shortest path will be: min(dist[i][k]+dist[k][j],dist[i][j]). dist[i][k] represents the shortest path that only uses the first K vertices, dist[k][j] represents the shortest path between the pair k,j. As the shortest path will be a concatenation of the shortest path from i to k, then from k to


1
2
3
4
8
3
7
2
2
5
1

Floyd Warshall: (Dynamic Programming)

$$A^0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & \infty \\ 3 & 5 & \infty & 0 & 1 \\ 4 & 2 & \infty & \infty & 0 \end{array}$$

$$A^1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & \infty & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 8 & 8 & 0 & 1 \\ 4 & 2 & 5 & \infty & 0 \end{array}$$

$A^0[2,3] = A^0[2,1] + A^0[1,3]$

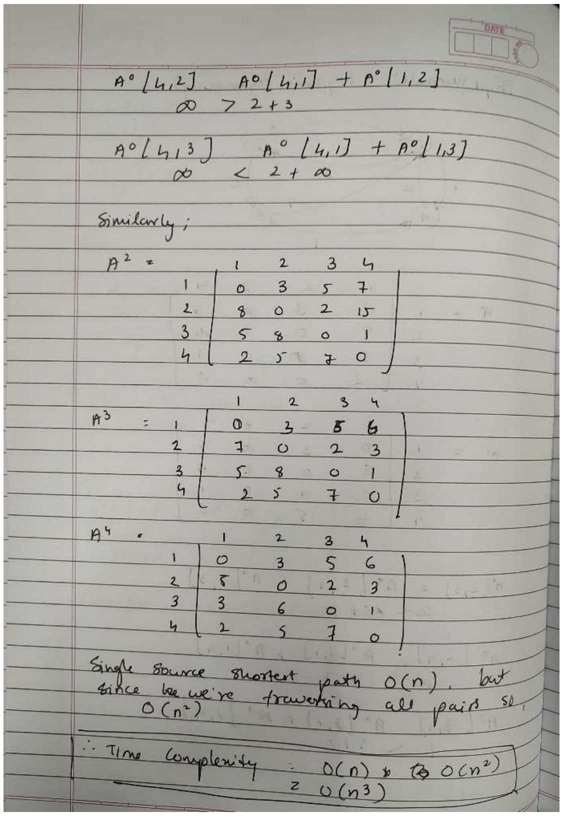
$2 < \cancel{8}8 + \infty$

$A^0[2,4], \quad A^0[2,1] + A^0[1,4]$

$\infty > 8+7$

$A^0[3,4] \quad A^0[3,1] + A^0[1,4]$

$1 < 5+2$



$A^0[4,2] \quad A^0[4,1] + A^0[1,2]$

$\infty > 2+3$

$A^0[4,3] \quad A^0[4,1] + A^0[1,3]$

$\infty < 2 + \infty$

Similarly;

$$A^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 7 \\ 2 & 8 & 0 & 2 & 15 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

$$A^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 7 & 0 & 2 & 3 \\ 3 & 5 & 8 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

$$A^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 5 & 6 \\ 2 & 5 & 0 & 2 & 3 \\ 3 & 3 & 6 & 0 & 1 \\ 4 & 2 & 5 & 7 & 0 \end{array}$$

Single source shortest path $O(n)$, but since ~~we~~ we're traversing all pairs so, $O(n^2)$

$\therefore$ Time Complexity $= O(n) * \cancel{3}\, O(n^2)$

$= O(n^3)$

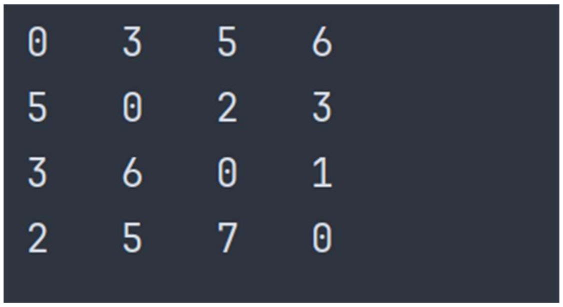
Program(Code)

```
public class floydWarshall {
    private static final int INF = Integer.MAX_VALUE;

    public static void main(String[] args) {
        int[][] graph = {
                {0, 3, INF, 7},
                {8, 0, 2, INF},
                {5, INF, 0, 1},
                {2, INF, INF, 0}
        };
        int[][] shortestDistances = floydWarshall(graph);
        for (int i = 0; i < shortestDistances.length; i++)
{
            for (int j = 0; j <
shortestDistances[i].length;
                 j++) {
                if (shortestDistances[i][j] == INF)
                    System.out.print("INF\t");
                else

System.out.print(shortestDistances[i][j]
                            + "\t");
            }
            System.out.println();
        }
    }

    public static int[][] floydWarshall(int[][] graph) {
        int V = graph.length;
        int[][] dist = new int[V][V];
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                dist[i][j] = graph[i][j];
            }
        }
        for (int k = 0; k < V; k++) {
            for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                    if (dist[i][k] != INF && dist[k][j] !=
                            INF &&
                            dist[i][k] + dist[k][j] <
                                    dist[i][j]) {
                        dist[i][j] = dist[i][k] +
                                dist[k][j];
                    }
                }
            }
        }
        return dist;
    }
}
```

Output	 0 3 5 6 5 0 2 3 3 6 0 1 2 5 7 0
Justification of the complexity calculated	The time complexity analysis for the Floyd-Warshall algorithm: 1. Initializing the distances matrix takes O(V^2) time. This is because we need to iterate over all elements of the VxV matrix to copy the values from the input graph to the distances matrix. 2. The algorithm consists of three nested loops: •The outermost loop runs V times, where V is the number of vertices in the graph. This loop iterates through all vertices and acts as the intermediate vertex through which shortest paths are checked. • Inside the outer loop, there are two nested loops that iterate over all pairs of vertices (i and j). Since there are V vertices, each of these nested loops runs V times. Therefore, the nested loops together result in O(V^2) iterations. • Inside the nested loops, each iteration involves comparing and possibly updating the distance values. This comparison and update operation takes constant time (O(1)) because it involves simple arithmetic operations. Therefore, the overall time complexity of the algorithm is dominated by the nested loops, resulting in O(n^3).
Conclusion	Application of Floyd- Warshall algorithm: Shortest Path Problem: The primary application of the Floyd-Warshall algorithm is in finding the shortest paths between all pairs of vertices in a weighted graph. This is useful in transportation networks, routing algorithms, and network optimization. Network Routing: In computer networks, the Floyd-Warshall algorithm can be used to compute the shortest paths between all pairs of nodes in a network. It helps in determining the most efficient routes for data packets to travel through the network. Traffic Management: In urban planning and transportation engineering, the Floyd-Warshall algorithm can be applied to model traffic flow and optimize traffic signal timings to minimize congestion and travel times. Airline Route Planning: Airlines can use the Floyd-Warshall algorithm to analyze flight routes and determine the shortest paths between airports. This helps in optimizing flight schedules and minimizing travel distances for passengers.