



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 (Floyd Warshal Algorithm)
Objective	1) Learn All pair shortest path 2) Implement All pair shortest path 3) Derive the Time complexity Floyd Warshall algorithm
Name	Durgesh Dilip Mandge
UCID	2023510032
Class	FY MCA
Batch	B
Date of Submission	22/04/24
Algorithm and Explanation of the technique used	<p>Algorithm:</p> <p>For a graph with N vertices:</p> <p>Step 1: Initialize the shortest paths between any 2 vertices with Infinity.</p> <p>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.</p> <p>Step 3: Minimize the shortest paths between any 2 pairs in the previous operation.</p> <p>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: $\min(\text{dist}[i][k] + \text{dist}[k][j], \text{dist}[i][j])$.</p> <p>$\text{dist}[i][k]$ represents the shortest path that only uses the first K vertices, $\text{dist}[k][j]$ 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 j.</p>

Program Code	<pre>import java.util.Arrays; public class FloydWarshall { static final int INF = Integer.MAX_VALUE; public static void floydWarshall(int[][] graph) { int V = graph.length; int[][] dist = new int[V][V]; // Initialize the distance array for (int i = 0; i < V; i++) {</pre>
---------------------	--

```
        dist[i] = Arrays.copyOf(graph[i], V);
    }

    // Floyd Warshall algorithm
    for (int k = 0; k < V; k++) {
        System.out.println("Intermediate dist array after pass " + (k + 1)
+ ":"); printSolution(dist);
        System.out.println();

        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                // Check if vertex k is an intermediate vertex that reduces
the distance 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]; }
            }
        }
    }

    // Print final shortest distances between every pair of vertices
    System.out.println("Final shortest distances between every pair of
vertices:");
    printSolution(dist);
}

// Utility function to print the solution
public static void printSolution(int[][] dist)
{ int V = dist.length; for (int i = 0; i < V;
++i) { for (int j = 0; j < V; ++j) { if (dist[i][j]
== INF) {
        System.out.print("INF\t");
    } else {
        System.out.print(dist[i][j] + "\t");
    }
    }
    System.out.println();
}
}

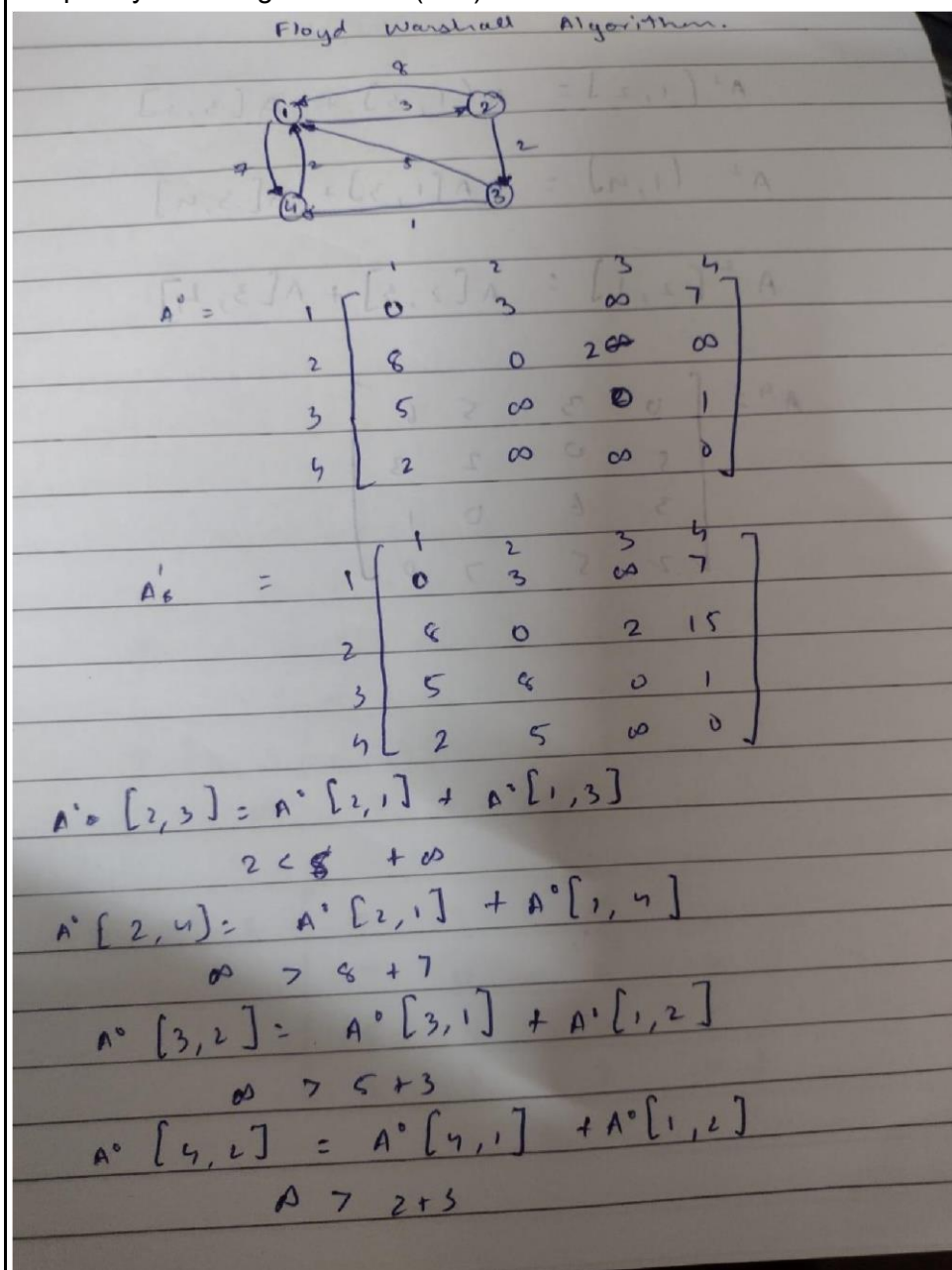
public static void main(String[] args) {
    int[][] graph = {
        {0, 3, INF, 7},
        {8, 0, 2, INF},
        {5, INF, 0, 1},
        {2, INF, INF, 0}
    };
}
```

	<pre>// Call the Floyd Warshall method floydWarshall(graph); } }</pre>
Output	<pre>Intermediate dist array after pass 1: 0 3 INF 7 8 0 2 INF 5 INF 0 1 2 INF INF 0 Intermediate dist array after pass 2: 0 3 INF 7 8 0 2 15 5 8 0 1 2 5 INF 0 Intermediate dist array after pass 3: 0 3 5 7 8 0 2 15 5 8 0 1 2 5 7 0 Intermediate dist array after pass 4: 0 3 5 6 7 0 2 3 5 8 0 1 2 5 7 0 Final shortest distances between every pair of vertices: 0 3 5 6 5 0 2 3 3 6 0 1 2 5 7 0</pre>

Justification of the complexity calculated

The time complexity of the Floyd-Warshall algorithm is $O(V^3)$, where V is the number of vertices in the graph.

1. Initialization: Initializing the 'dist' array with the graph values takes $O(V^2)$ time, where V is the number of vertices.
2. Main Loop: The main loop runs for V iterations ($k = 0$ to $V-1$), and in each iteration, it examines all pairs of vertices (i, j) . For each pair, it checks if there exists a shorter path passing through vertex k . This operation takes $O(1)$ time for each pair, resulting in a time complexity of $O(V^2)$ per iteration.
3. Overall Complexity: As the main loop runs V times, the total time complexity of the algorithm is $O(V^3)$.



$$A^0[4,3] = A^0[4,1] + A^0[1,3]$$

$$\infty < 2 + \infty$$

Similarly,

$$A^2 = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{matrix}$$

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

$$5 < 8 + 8.$$

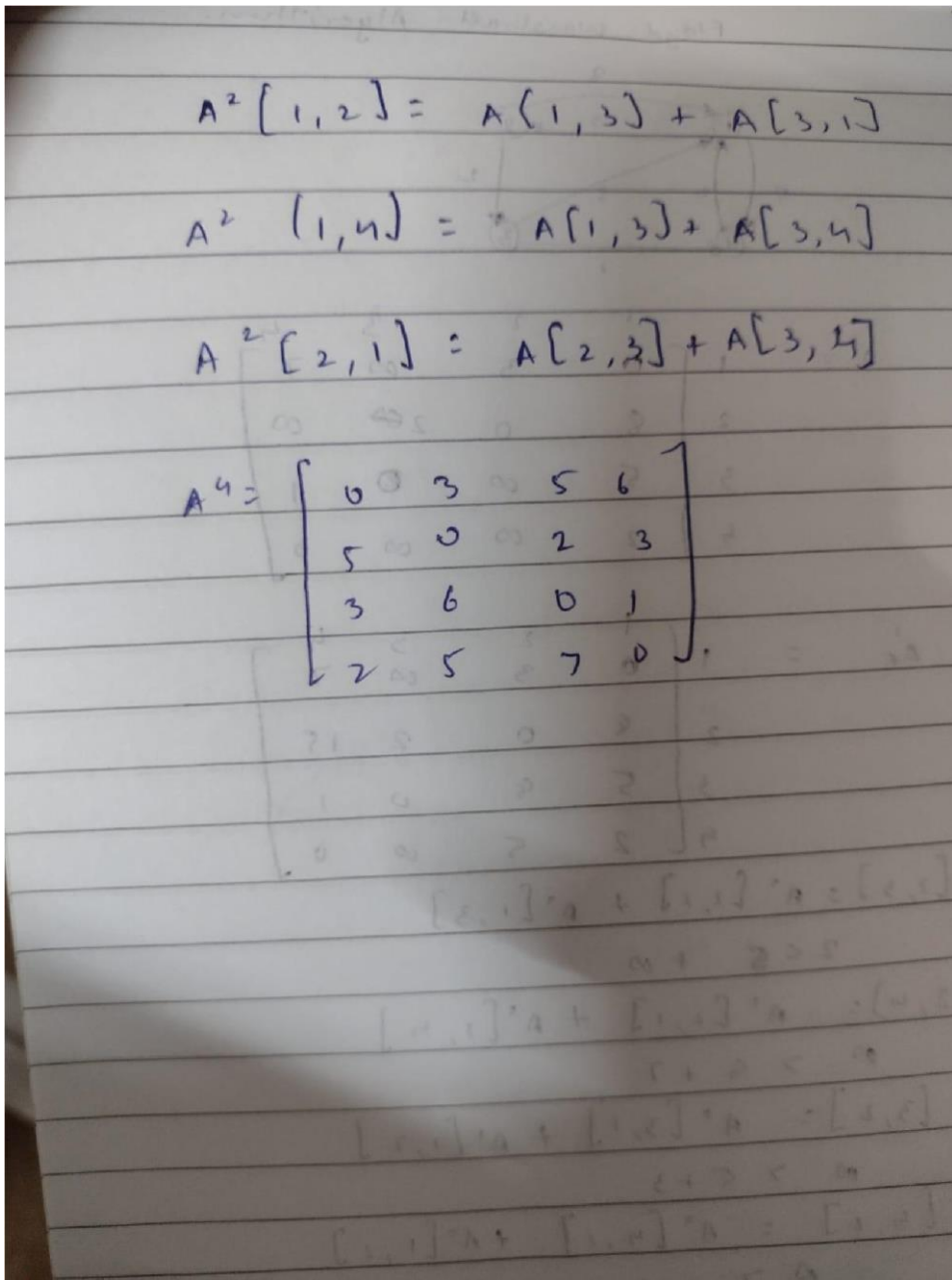
$$A^1[3,4] = A^1[3,2] + A^1[2,4]$$

$$1 < 8 + 5$$

$$A^1[4,3] = A^1[4,2] + A^1[4,3]$$

$$\infty > 8 + 7$$

$$A^3 = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$$

	 <p>$A^2[1,2] = A[1,3] + A[3,2]$$A^2[1,4] = A[1,3] + A[3,4]$$A^2[2,1] = A[2,3] + A[3,1]$$A^4 = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix}$</p>
Conclusion	<p>Despite its cubic time complexity, Floyd-Warshall is often preferred for dense graphs or when the number of vertices is relatively small because it computes the shortest paths between all pairs of vertices efficiently.</p> <p>Here are some applications:</p> <ul style="list-style-type: none">• Transportation Planning: Transportation companies, such as logistics firms or public transportation agencies, utilize the Floyd-Warshall algorithm to optimize their route planning.• Network Routing: Floyd-Warshall algorithm is used in network routing protocols to compute the shortest paths between all pairs of nodes in a network.

