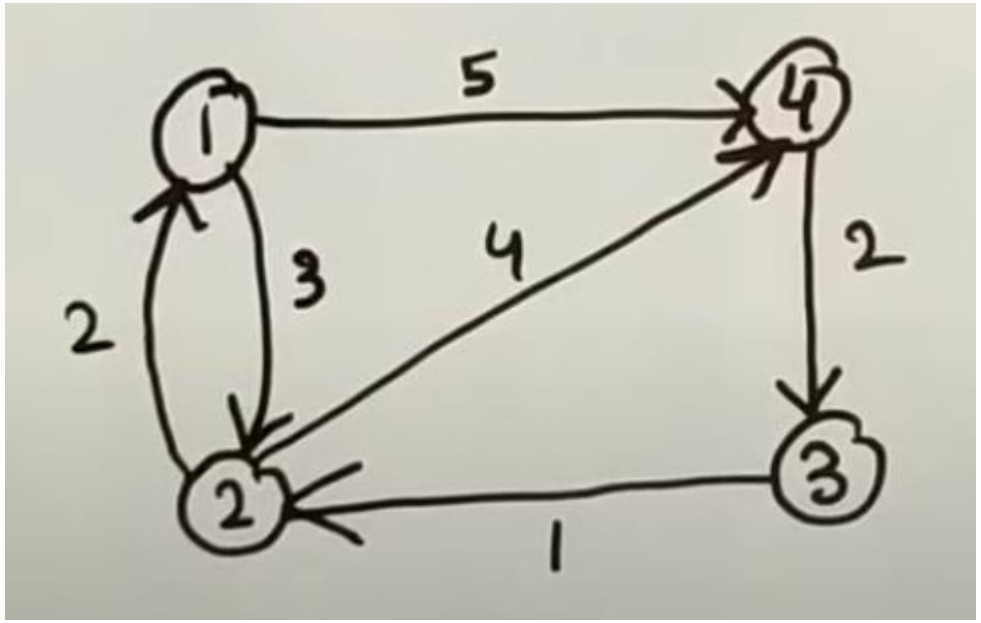
Floyd Warshall Algorithm (An all pair shortest path algorithm)

By

Dr. GC Jana

Floyd Warshall Algorithm

- Find the shortest path between every two vertex's using Floyd Warshall Algorithm



$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Floyd Warshall Algorithm

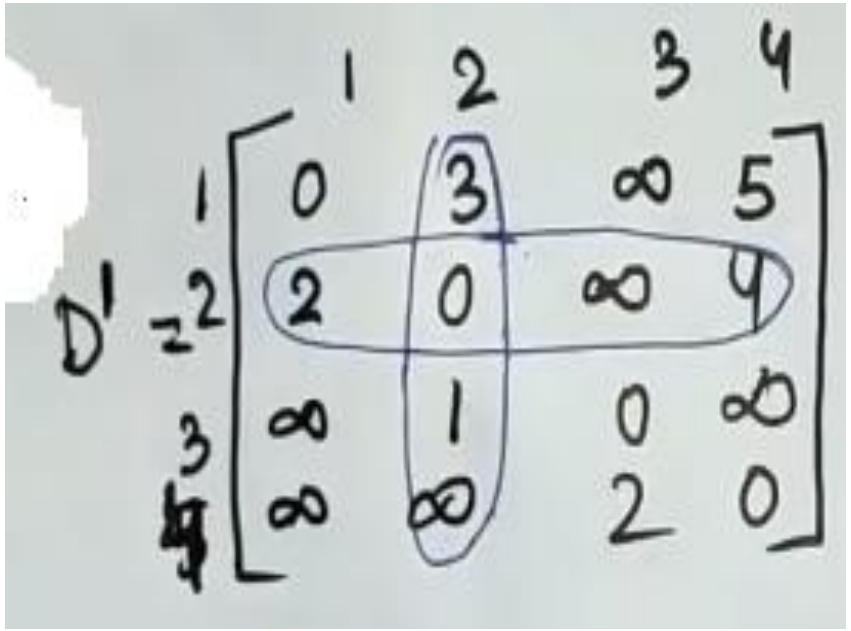
$$D^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

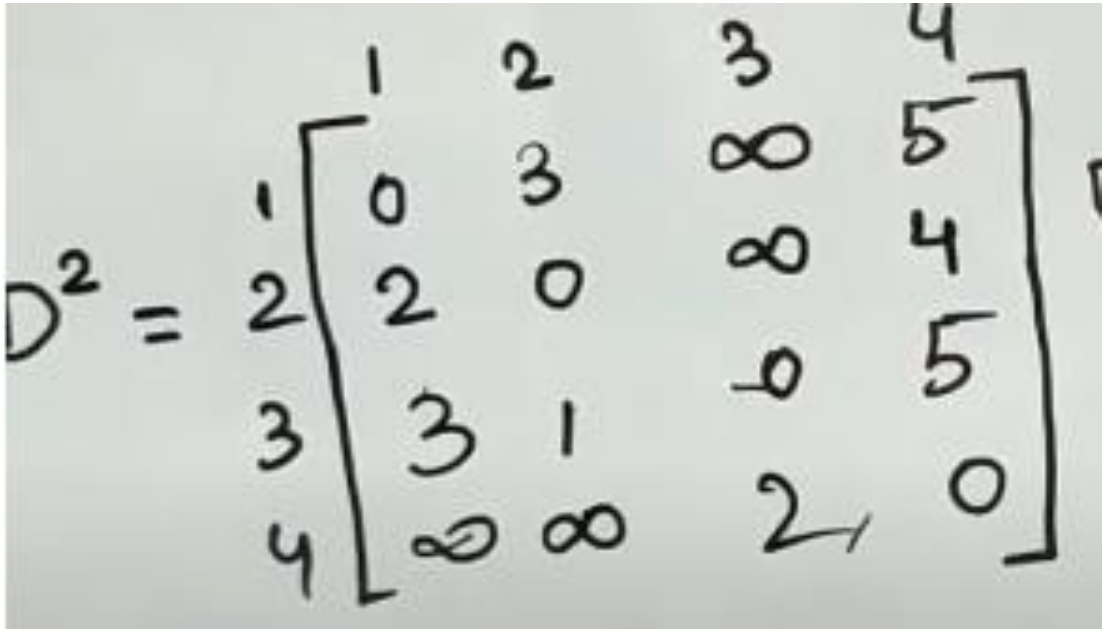
$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array}$$

$$D^1(2,3) = \text{Min}\left[D^0(2,3),\ D^0[2,1] + D^0[1,3]\right]$$

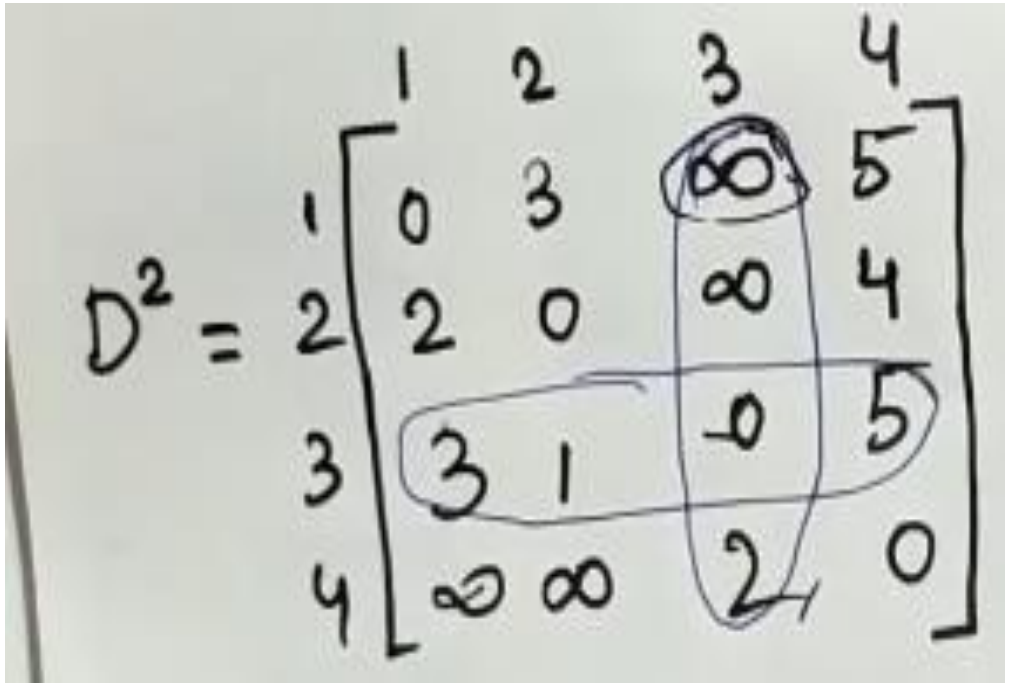
$$D^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & \infty & 4 \\ \infty & 1 & 0 & \infty \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

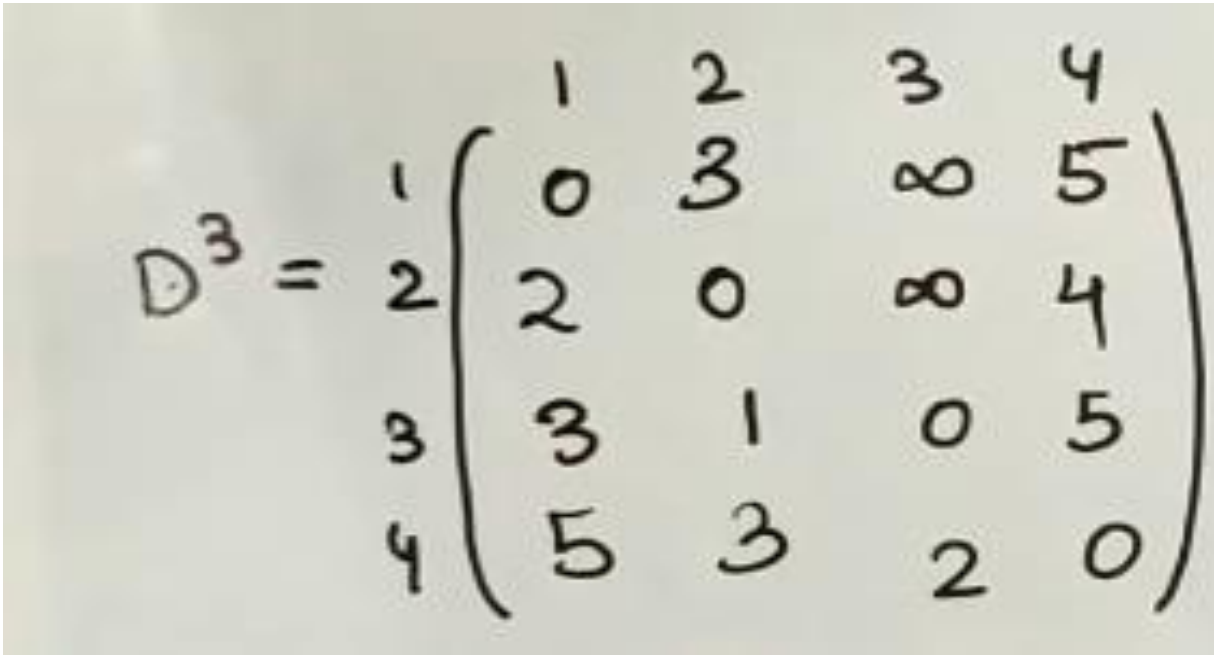
Floyd Warshall Algorithm





Floyd Warshall Algorithm





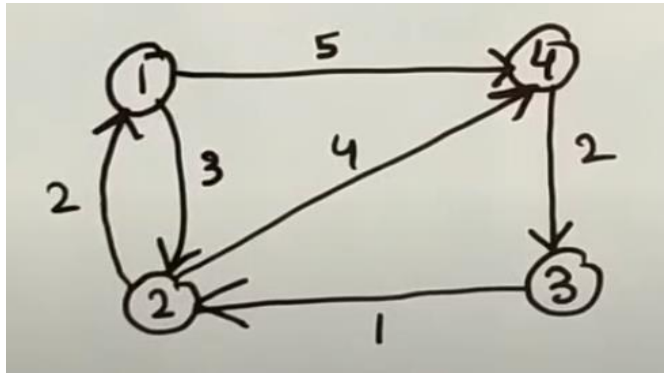
Floyd Warshall Algorithm

$D^3 =$

	1	2	3	4
1	0	3	∞	5
2	2	0	∞	4
3	3	1	0	5
4	5	3	2	0

$D^4 =$

	1	2	3	4
1	0	3	7	5
2	2	0	6	4
3	3	1	0	5
4	5	3	2	0



Floyd Warshall Algorithm Algorithm:

•Initialize the solution matrix same as the input graph matrix as a first step.

•Then update the solution matrix by considering all vertices as an intermediate vertex.

•The idea is to pick all vertices one by one and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.

•When we pick vertex number **k** as an intermediate vertex, we already have considered vertices **{0, 1, 2, .. k-1}** as intermediate vertices.

•For every pair **(i, j)** of the source and destination vertices respectively, there are two possible cases.

- **k** is not an intermediate vertex in shortest path from **i** to **j**. We keep the value of **dist[i][j]** as it is.
- **k** is an intermediate vertex in shortest path from **i** to **j**. We update the value of **dist[i][j]** as **dist[i][k] + dist[k][j],** if **dist[i][j] > dist[i][k] + dist[k][j]**

Pseudo-Code of Floyd Warshall Algorithm :

```
For k = 0 to n – 1
For i = 0 to n – 1
For j = 0 to n – 1
Distance[i, j] = min(Distance[i, j], Distance[i, k] + Distance[k, j])
```

where i = source Node, j = Destination Node, k = Intermediate Node

Floyd Warshall Algorithm-Java Code

https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

```
import java.io.*;
import java.lang.*;
import java.util.*;

class AllPairShortestPath {
    final static int INF = 99999, V
= 4;

    void floydWarshall(int
dist[][])
    {

        int i, j, k;
```

```
for (k = 0; k < V; k++) {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++) {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++) {

    if (dist[i][k] + dist[k][j]
                            < dist[i][j])
                            dist[i][j]
                                = dist[i][k] + dist[k][j];
                    }
                }
            }
```

Floyd Warshall Algorithm-Java Code

```
// Print the shortest distance matrix
    printSolution(dist);
}

void printSolution(int dist[][])
{
    System.out.println(
        "The following matrix shows the shortest "
        + "distances between every pair of vertices");
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            if (dist[i][j] == INF)
                System.out.print("INF ");
            else
                System.out.print(dist[i][j] + "   ");
        }
        System.out.println();
    }
}
```

```
// Driver's code
    public static void main(String[] args)
    {
        /* Let us create the following weighted graph
           10
        (0)------->(3)
        |        /|\
        5 |         |
        |         | 1
        \|/        |
        (1)------->(2)
           3         */
        int graph[][] = { { 0, 5, INF, 10 },
                          { INF, 0, 3, INF },
                          { INF, INF, 0, 1 },
                          { INF, INF, INF, 0 } };
        AllPairShortestPath a = new AllPairShortestPath();

        // Function call
        a.floydWarshall(graph);
    }
}
```

Floyd Warshall Algorithm-Output

Output

The following matrix shows the shortest distances between every pair of vertices

0 5 8 9
INF 0 3 4
INF INF 0 1
INF INF INF 0

Complexity Analysis of Floyd Warshall Algorithm:

- **Time Complexity: $O(V^3)$,** where V is the number of vertices in the graph and we run three nested loops each of size V
- **Auxiliary Space: $O(V^2)$,** to create a 2-D matrix in order to store the shortest distance for each pair of nodes.