Program:

```
 public class DijkstraAlgorithm {

          public void dijkstraAlgorithm(int[][] graph, int source) {
          // number of nodes
          int nodes = graph.length;
          boolean[] visited_vertex = new boolean[nodes];
          int[] dist = new int[nodes];
          for (int i = 0; i < nodes; i++) {
            visited_vertex[i] = false;
            dist[i] = Integer.MAX_VALUE;
          }

          // Distance of self loop is zero
          dist[source] = 0;
          for (int i = 0; i < nodes; i++) {

            // Updating the distance between neighboring vertex and source
vertex
            int u = find_min_distance(dist, visited_vertex);
            visited_vertex[u] = true;

            // Updating the distances of all the neighboring vertices
            for (int v = 0; v < nodes; v++) {
              if (!visited_vertex[v] && graph[u][v] != 0 && (dist[u] +
graph[u][v] < dist[v])) {
                dist[v] = dist[u] + graph[u][v];
              }
            }
          }
          for (int i = 0; i < dist.length; i++) {
            System.out.println(String.format("Distance from Vertex %s to
Vertex %s is %s", source, i, dist[i]));
          }

        }

        // defining the method to find the minimum distance
        private static int find_min_distance(int[] dist, boolean[]
visited_vertex) {
          int minimum_distance = Integer.MAX_VALUE;
          int minimum_distance_vertex = -1;
          for (int i = 0; i < dist.length; i++) {
            if (!visited_vertex[i] && dist[i] < minimum_distance) {
              minimum_distance = dist[i];
              minimum_distance_vertex = i;
            }
        }
        return minimum_distance_vertex;
      }

        public static void main(String[] args) {
        // declaring the nodes of the graphs
        int graph[][] = new int[][] {
          { 0, 1, 1, 2, 0, 0, 0 },
          { 0, 0, 2, 0, 0, 3, 0 },
          { 1, 2, 0, 1, 3, 0, 0 },
          { 2, 0, 1, 0, 2, 0, 1 },
          { 0, 0, 3, 0, 0, 2, 0 },
          { 0, 3, 0, 0, 2, 0, 1 },
          { 0, 2, 0, 1, 0, 1, 0 }
        };

        DijkstraAlgorithm Test = new DijkstraAlgorithm();

        Test.dijkstraAlgorithm(graph, 0);
      }
    }

    /*OUTPUT:
    Distance from Vertex 0 to Vertex 0 is 0
Distance from Vertex 0 to Vertex 1 is 1
Distance from Vertex 0 to Vertex 2 is 1
Distance from Vertex 0 to Vertex 3 is 2
Distance from Vertex 0 to Vertex 4 is 4
Distance from Vertex 0 to Vertex 5 is 4
Distance from Vertex 0 to Vertex 6 is 3

     */
```

Pgm6 a &b : Implement Warshall and Floyd's algorithm

Warshall's and Floyd's Algorithm

Warshall's Algorithm is used to compute the **transitive closure** of a directed graph, whereas **Floyd's Algorithm (Floyd-Warshall Algorithm)** is used to find the **shortest paths between all pairs of vertices** in a weighted graph.

[illegible] pairs of vertices → $O(V^3)$.

Program:

6

```
import java.lang.*;

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

    void floydWarshall(int dist[][])
    {

        int i, j, k;

        /* Add all vertices one by one  to the set of intermediate vertices.
          ---> Before start of an iteration,
               we have shortest
               distances between all pairs
               of vertices such that
               the shortest distances consider
               only the vertices in
```

```
            set {0, 1, 2, .. k-1} as
            intermediate vertices.
      ----> After the end of an iteration,
             vertex no. k is added
             to the set of intermediate
             vertices and the set
             becomes {0, 1, 2, .. 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 vertex k is on the shortest path
                // from i to j, then update the value of
                // dist[i][j]
                if (dist[i][k] + dist[k][j]
                    < dist[i][j])
                    dist[i][j]
                        = dist[i][k] + dist[k][j];
            }
        }
    }

    // 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);
          }
      }

      /* ----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  */
```

```
public class LCMCalculator {
    private static int gcd(int a, int b) {
        if (b == 0)
            return a;
        return gcd(b, a % b);
    }

    // Function to compute LCM of two numbers
    private static int lcm(int a, int b) {
        return (a * b) / gcd(a, b);
    }
```

```
    // Function to compute LCM of an array using transfer and conquer
    public static int lcmArray(int[] arr) {
        int result = arr[0]; // Start with the first element
        for (int i = 1; i < arr.length; i++) {
            // Transform: Combine each element with the result to form the
next state
            result = lcm(result, arr[i]);
        }
        return result;
    }

    public static void main(String[] args) {
        int[] numbers = {12, 15, 20, 25};
        int result = lcmArray(numbers);
        System.out.println("LCM of the array is: " + result);
    }

}

/* OUTPUT
 LCM of the array is: 300
 */
```