DijkstraAlgorithm.java

```
package pt.ipp.isep.dei.Domain;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class DijkstraAlgorithm {
    public static int[] dijkstra(int[][] graph, int src) {
        int n = graph.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[src] = 0;

        for (int i = 0; i < n - 1; i++) {
            int u = -1;
            int minDist = Integer.MAX_VALUE;
            for (int j = 0; j < n; j++) {
                if (!visited[j] && dist[j] < minDist) {
                    minDist = dist[j];
                    u = j;
                }
            }
            if (u == -1) break;
            visited[u] = true;
            for (int v = 0; v < n; v++) {
                 if (graph[u][v] > 0 && !visited[v] && dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] <
dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }
        return dist;
    }

    public static List<List<Integer>> dijkstraWithPaths(int[][] graph, int src) {
        int n = graph.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        int[] prev = new int[n];
        Arrays.fill(dist, Integer.MAX_VALUE);
        Arrays.fill(prev, -1);
        dist[src] = 0;
```

```
        for (int i = 0; i < n - 1; i++) {
            int u = -1;
            int minDist = Integer.MAX_VALUE;
            for (int j = 0; j < n; j++) {
                if (!visited[j] && dist[j] < minDist) {
                    minDist = dist[j];
                    u = j;
                }
            }
            if (u == -1) break;
            visited[u] = true;
            for (int v = 0; v < n; v++) {
                 if (graph[u][v] > 0 && !visited[v] && dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] <
dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                    prev[v] = u;
                }
            }
        }
        return reconstructPaths(prev, src, n);
    }

    private static List<List<Integer>> reconstructPaths(int[] prev, int src, int n) {
        List<List<Integer>> paths = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            List<Integer> path = new ArrayList<>();
            if (i != src && prev[i] != -1) {
                for (int at = i; at != -1; at = prev[at]) {
                    path.add(0, at);
                }
            }
            paths.add(path);
        }
        return paths;
    }
}
```