Exercício 1:

Nesta função é criado um grafo Dijkstra com os vértices da Matriz de Adjacência usada no Sprint 3.

O(2 * n) + O(log n) + O(log n) = O(n)

```
BinaryOperator<Double> operator = (x, y) -> x + y; // somar distancias
Comparator<Double> portComparator = (o1, o2) -> o1 < o2 ? -1 : (o1 == o2 ? 0 : 1);

LinkedList<Location> portLinkedList = new LinkedList<>();
Double zero = 0.0;

ArrayList<Location> vertices = dijkstraGraph.vertices();
ArrayList<String> paths = new ArrayList<>();
Map<Integer, Integer> pathsMap = new HashMap<>();
for (Location fromLocation : vertices) {
    for (Location toLocation : vertices) {
        String path = dijkstraGraph.shortestPath(dijkstraGraph, fromLocation, toLoc
        if (path != null) {
            String[] splitPath = path.split( regex: ",");
            if (Double.parseDouble(splitPath[splitPath.length - 1]) != 0.0) {
                //System.out.println("From: " + fromLocation.getName() + " To: " +
                paths.add(path);
            }
        }
    }
}
```

Nesta parte da função chamamos o método shortestPath. Posteriormente percorremos o arraylist e incrementamos o número de vezes que aparece determinada key. Se no HashMap criado a key tiver o valor de X, se a key aparecer novamente ai incrementar esse mesmo X.

O(2 * n) + O(log n) + O(n) + O(2*n^2) = O(n)

```
ArrayList<Location> vertices = dijkstraGraph.vertices();
ArrayList<String> paths = new ArrayList<>();
Map<Integer, Integer> pathsMap = new HashMap<>();
for (Location fromLocation : vertices) {
    for (Location toLocation : vertices) {
        String path = dijkstraGraph.shortestPath(dijkstraGraph, fromLocation, toLocation, portComparator
        if (path != null) {
            String[] splitPath = path.split( regex: ",");
            if (Double.parseDouble(splitPath[splitPath.length - 1]) != 0.0) {
                //System.out.println("From: " + fromLocation.getName() + " To: " + toLocation.getName()
                paths.add(path);
            }
        }
    }
}

for (String path : paths) {
    String[] splitPath = path.split( regex: ",");
    for (int i = 0; i < splitPath.length - 1; i++) {
        int location = Integer.parseInt(splitPath[i]);
        //
        // System.out.println(location);
        if (pathsMap.containsKey(location)) {
            pathsMap.put(location, pathsMap.get(location) + 1);
        } else {
            pathsMap.put(location, 1);
        }
    }
}
```

Por último, ordenamos o HashMap de forma decrescente e retornamos o n números de shortestPaths que o utilizador pede.

O(2*n^2) = O(n)

```
List<Map.Entry<Integer, Integer>> list = new LinkedList<>(pathsMap.entrySet());

Collections.sort(list, (o1, o2) -> o2.getValue().compareTo(o1.getValue()));

Map<Integer, Integer> sortedPaths = new LinkedHashMap<~>();
int counter = 0;
for (Map.Entry<Integer, Integer> entry : list) {
    sortedPaths.put(entry.getKey(), entry.getValue());
    if (counter < n) { // o n de portos com mais shortest paths
        System.out.println(dijkstraGraph.getVertex(entry.getKey()).getName() + " " + entry.getValue());
    }
    counter++;
}
```