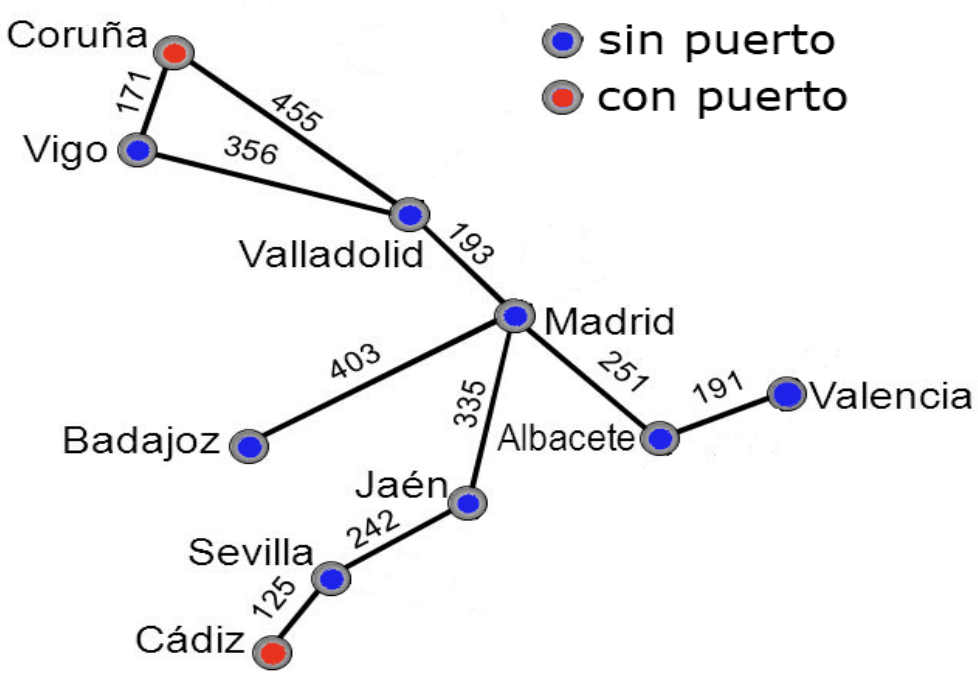
```
function closestPortDijkstra (Graph g, City source){

        //ATTRIBUTES
        List<City> cities;                      //vertices of the Graph
        List<City> ports;                       //cities (vertices) with ports
        HashMap<City, Distance> distances;      //Distance = (Integer value, City previous)
        City closestPort;                       //Will be the closest Port to the source City when found by the algorithm
        List<City> path;                        //Path of cities to the closest port

        //INITIALIZATION
        for each (City c in cities){
                distances.add(c);
                distances(c).value() = infinity;
                distances(c).previous() = null;
        }
        distances(source).value() = 0;
        distances(source).previous = source;

        //MAIN LOOP
        while (cities.hasUnvisited()){                  //Visited is a boolean attribute of City
                City u = cities.getMin(distances);      //returns an unvisited city with the min distance to the source given
                cities.markVisited(c);
                for each (neighbor v of u){
                        int aux = distances(u).value() + distanceBetween (u, v);        //comienzo Factible
                        if (aux < distances(v).value()){
                                distances(v).value() = aux;
                                distances(v).previous() = u;
                        }                                                               //fin Factible
                }
        }
        path = closestPortPath(distances, ports);
        return path;
}

function getMin(){                                      //Seleccionar
        City min = infinity;
        for each (City c in distances){
                if ((!c.visited() && (distances(min).value()){
                        min = c;
                }
        }
        return min;
}
```

```
function closestPortPath (List<Distance> distances, List<City> ports){
	List<City> result = null;
	int shortestDistance = infinity;
//looks in distances for each of the port cities and calculates the closest one, then builds the path in the result list
	for each (city p in ports){
		int aux = distances(p).value();
		if (aux < shortestDistance){
			result.reset();
			shortestDistance = aux;
			closestPort = p;
			result = buildPath(distances, p);
		}
	}
	return result;
}

function buildPath(List<Distance> distances, City c){
	//adds the port city to the path
	List<City>result = new List;
	while (distances(c).value() != 0){
		result.add(c);
		c = distances(c).previous();
	}
	result.add(c);
	return result;
}
```

Ejemplo con ciudad origen: Valencia.
Para simplificar se utilizará una versión de menor tamaño del grafo provisto.
cities = (Valencia, Albacete, Madrid, Jaén, Valladolid, Badajoz, Sevilla, Cádiz, Vigo, Coruña)
ports = (Cádiz, Coruña)



Inicialización:

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	false	false	false	false	false	false	false	false	false	false
Distance	(0, Valencia)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)

Iteración 1
Toma la ciudad con menor distancia que es dónde esta en el momento, Valencia, con distancia 0. LA marca como visitada y agrega la/s distancia/s (Valor, Ciudad Previa) de su/s vecino/s.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	false	false	false	false	false	false	false	false	false
Distance	(0, Valencia)	(191, Valencia	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)

Iteración 2
Vuelve a buscar la ciudad no visitada con menor distancia que es Albacete, la marca como visitada y define la distancia de los vecinos.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	false	false	false	false	false	false	false	false
Distance	(0, Valencia)	(191, Valencia)	(442, Albacete)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)	(∞, null)

Iteración 3
Repite la acción con Madrid.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	false	false	false	false	false	false	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(∞, null)	(∞, null)	(∞, null)	(∞, null)

Iteración 4
Repite con Badajoz.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	false	false	true	false	false	false	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(∞, null)	(∞, null)	(∞, null)	(∞, null)

Iteración 5
Repite con Jaén.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	true	false	true	false	false	false	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(1039, Jaén)	(∞, null)	(∞, null)	(∞, null)

Iteración 6
Repite con Valladolid.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	true	true	true	false	false	false	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(1039, Jaén)	(∞, null)	(1201, Valladolid)	(1300, Valladolid)

Iteración 7
Repite con Sevilla

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	true	true	true	true	false	false	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(1039, Jaén)	(1164, Sevilla)	(1201, Valladolid)	(1300, Valladolid)

Iteración 8
Repite con Cádiz.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	true	true	true	true	true	false	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(1039, Jaén)	(1164, Sevilla)	(1201, Valladolid)	(1300, Valladolid)

Iteración 9
Repite con Vigo

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	true	true	true	true	true	true	false
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(1039, Jaén)	(1164, Sevilla)	(1201, Valladolid)	(1300, Valladolid)

En este paso la distancia (valor) de Coruña es 1300, llegando desde Vigo, ese valor sería 1372. Como no es menor, se mantiene el valor establecido en la iteración 6.

Iteración 10

Repite con Coruña, la última ciudad no visitada.

	Valencia	Albacete	Madrid	Jaén	Valladolid	Badajoz	Sevilla	Cádiz	Vigo	Coruña
visited	true	true	true	true	true	true	true	true	true	true
Distance	(0, Valencia)	(191, Valencia)	(251, Albacete)	(797, Madrid)	(845, Madrid)	(635, Madrid)	(1039, Jaén)	(1164, Sevilla)	(1201, Valladolid)	(1300, Valladolid)

Una vez completada la lista de distancias se busca el puerto más cercano y su correspondiente distancia y camino.

```
function closestPortPath (List<Distance> distances, List<City> ports){
        List<City> result = null;
        int shortestDistance = infinity;
        for each (city p in ports){                          //Iteración 1, comienza con Cádiz
                int aux = distances(p).value();              //aux = 1164
                if (aux < shortestDistance){                 //true
                        result.reset();                      //vacía la lista resultado para cargar un camino nuevo
                        shortestDistance = aux;              //1164
                        closestPort = p;                     //Cádiz
                        result = buildPath(distances, p);    //entra a la función que arma el camino con Cádiz
                }
        }
        return result;
}

function buildPath(List<Distance> distances, City c){
        List<City>result = new List;
        while (distances(c).value() != 0){                   //true
                result.add(c);                               //agrega la ciudad a la lista resultado (al principio)
                c = distances(c).previous();                 //c = ciudad previa
        }
        result.add(c);                                       // se agrega la última ciudad (source) a la lista
        return result;
}
```

Iteración 1

```
while (distances(c).value() != 0){                   //true
        result.add(c);                               //agrega Cádiz a la lista resultado
        c = distances(c).previous();                 //c = Sevilla
}
```

Iteración 2

```
while (distances(c).value() != 0){                   //true
        result.add(c);                               //agrega Sevilla a la lista resultado
        c = distances(c).previous();                 //c = Jaén
}
```

Iteración 3

```
while (distances(c).value() != 0){                   //true
        result.add(c);                               //agrega Jaén a la lista resultado
        c = distances(c).previous();                 //c = Madrid
}
```

Iteración 4

```
while (distances(c).value() != 0){                    //true
        result.add(c);                                //agrega Madrid a la lista resultado
        c = distances(c).previous();                  //c = Albacete
}
```

Iteración 5

```
while (distances(c).value() != 0){                    //true
        result.add(c);                                //agrega Albacete a la lista resultado
        c = distances(c).previous();                  //c = Valencia
}
```

Iteración 6

```
while (distances(c).value() != 0){                    //false
        result.add(c);
        c = distances(c).previous();
}
```

Agrega Valencia a la lista que queda conformada:

(Valencia, Albacete, Madrid, Jaén, Sevilla, Cádiz)

Y el puerto closestPort = Cádiz

En la siguiente iteración de la función closestPortPath el valor shortestDistance esta establecido como 1164. Al entrar con Coruña, el valor de la distancia es de 1300. Como no es menor a 1164, el algoritmo termina y retorna la lista resultado de la iteración anterior.