Título: Entrega ejercicio 5 TP4 **Materia:** Programación 3

Alumno: Santiago Rampoldi

Introducción:

/*En esta solución utilizo y considero inicializados:
Un grafo no dirigido representado mediante una lista de adyacencias.

mejorRuta[ ] un arreglo que guarda la mejor distancia a la ciudad y el padre de la misma.
dist[ ] arreglo con las distancias hacia el puerto actual, no necesariamente las mejores.
U[ ] arreglo con booleanos para controlar las ciudades ya visitadas, inicializado en 0.
Estos arreglos están indexados por los vértices del grafo.

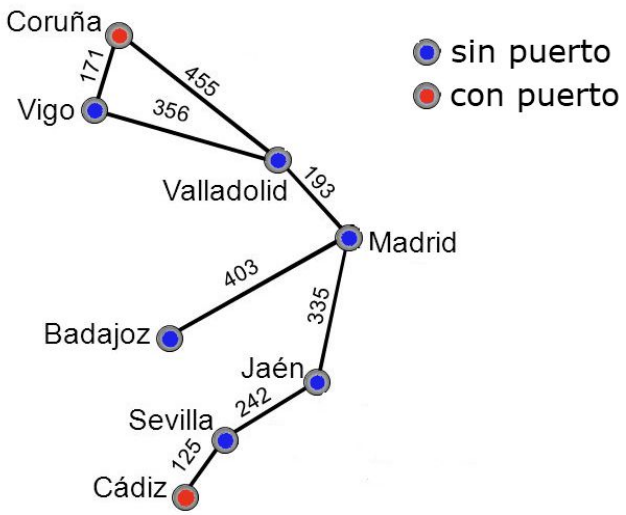
puertos[ ] un arreglo con todas las ciudades que tienen puerto.
También considero implementados los métodos para añadir y para encontrar la ciudad indicada en cada arreglo.
*/

Pseudocódigo:

```
public arreglo[Ruta[ ]] Dijkstra(ciudades[Ciudad])   {

constante infinito = ;   //un número muy grande, depende el lenguaje

        for each (Ciudad c en ciudades[ ])   {

                if c.hasPuerto()                    //Devuelve true si la ciudad posee un puerto
                        puertos.add(c);

        mejorRuta[c] = (Infinito, vacío);

}

        for each (ciudad p en puertos[ ]) { //Empieza el algoritmo Dijkstra, que se repite
                                            //una vez por cada ciudad con puerto para
                                            //asegurarse de encontrar la mejor ruta.

        dist[p] = (0, p); //La ciudad con puerto tiene distancia 0 y es su propio padre.



        while (!solucion(U[ ]))   {        //Verifica que todas las ciudades hayan sido visitadas
```

```
        u = seleccionar(U[ ], dist[ ])    //Selecciona la ciudad no visitada con menos distancia
                                          //Si la ciudad aún no fue visitada, la distancia es infinito

        U[u] = 1;                         //La marco como visitada

        for each (ciudad v (vecino de u)) {

        if (factible(u,v))                //Compara la distancia guardada anterior de la ciudad
                                          //con la que ofrece la nueva ruta
                dist[v] = (dist[u] + w(u, v), u);

                }
        }
        mejorRuta[ ] = comparar(mejorRuta[ ], dist[ ]); //Compara las mejores rutas guardadas
                                                        //con las nuevas calculadas y se queda
}
i = 0;

for each (ciudad ciu en ciudades[ ])  {
        retorno[i] = getRuta(mejorRuta[ ], ciu);
        i++;
        }
}

return retorno[];

}


public Ruta[ ] getRuta(mejorRuta[ ], ciudad i)  {

        salida.add(i);

        while (!i.hasPuerto()) {
         i = mejorRuta[padre]; //i se vuelve el padre guardado en mejorRuta[]
         salida.add(i);
        }

   return salida[ ];
}
```

Seguimiento de ejemplo:



puertos[ ] (Cadiz, Coruña)
ciudades[ ] (Cádiz, Sevilla, Jaén, Madrid, Badajoz, Valladolid, Vigo, Coruña)

Para cada puerto:
En este seguimiento arrancamos en Cádiz.
En cada iteración voy a ir marcando con fondo gris la casilla modificada en ese paso.

Primera iteración:

	Cádiz	**Sevilla**	**Jaén**	**Madrid**	**Badajoz**	**Valladolid**	**Vigo**	**Coruña**
U	0	0	0	0	0	0	0	0
dist[]	(0, Cádiz)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)

En esta tabla, que representa el arreglo *dist[ ]*, se muestra el inicio del algoritmo donde la única distancia que se conoce es la de Cádiz.
Se actualiza su distancia a 0 y es su mismo padre.

Como los vecinos de Cádiz tienen la distancia en infinito, la primera vez que se ejecute *factible()* va a devolver true y se va a guardar la nueva distancia.

Segunda iteración:

La primer selección es Cádiz, se la marca como visitada y se actualizan las distancias y padres de sus vecinos.

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	0	0	0	0	0	0	0
dist[]	(0, Cádiz)	(125, Cádiz)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)

Tercera iteración:

Se repite el mismo proceso para cada ciudad, siempre actualizando la distancia y padres correspondientes.

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	0	0	0	0	0	0
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)

En este caso la distancia hasta Jaén es la distancia de esa ruta ya recorrida (125 a través de Sevilla) sumado a la distancia entre Sevilla y Jaén (242).

Cuarta iteración:

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	1	0	0	0	0	0
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(702, Jaén)	(inf, vacío)	(inf, vacío)	(inf, vacío)	(inf, vacío)

Quinta iteración:

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	1	1	0	0	0	0
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(702, Jaén)	(1105, Madrid)	(895, Madrid)	(inf, vacío)	(inf, vacío)

Sexta iteración:

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	1	1	0	1	0	0
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(702, Jaén)	(1105, Madrid)	(895, Madrid)	(1251, Valladolid)	(1350, Valladolid)

Esta es la primera vez que el método *seleccionar()* tiene 2 ciudades no visitadas con distancia distinta de infinito, en este caso elige Valladolid que tiene menor distancia.

Séptima iteración:

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	1	1	1	1	1	0
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(702, Jaén)	(1105, Madrid)	(895, Madrid)	(1251, Valladolid)	(1350, Valladolid)

Octava iteración:

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	1	1	1	1	1	1
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(702, Jaén)	(1105, Madrid)	(895, Madrid)	(1251, Valladolid)	(1350, Valladolid)

Ya recorrió todas las ciudades, y no encontró mejores rutas.

Terminó el primer ciclo de Dijkstra.

Guardo las distancias obtenidas en el arreglo *mejorRuta[ ]*.

Recorro todo de vuelta arrancando desde el puerto de Coruña y genero una tabla similar que va ser *dist[ ]*(Reemplazo los valores anteriores por infinito).

La tabla resultante sería esta:

	Cádiz	Sevilla	Jaén	Madrid	Badajoz	Valladolid	Vigo	Coruña
U	1	1	1	1	1	1	1	1
dist[]	(1350, Sevilla)	(1125, Jaén)	(983, Madrid)	(648, Valladolid)	(1051, Madrid)	(455, Coruña)	(171, Coruña)	(0, Coruña)

Terminó el segundo ciclo de Dijkstra.

Como se puede observar varias de las distancias calculadas en el segundo ciclo son mejores que las del primero, así que comparo los dos arreglos posición por posición y elijo la mejor alternativa en cada caso, actualizando la distancia y el padre si es necesario.

Dándome como resultado **esta tabla:**

	Cádiz	**Sevilla**	**Jaén**	**Madrid**	**Badajoz**	**Valladolid**	**Vigo**	**Coruña**
Ciclo elegido	1	1	1	2	2	2	2	2
dist[]	(0, Cádiz)	(125, Cádiz)	(367, Sevilla)	(648, Valladolid)	(1051, Madrid)	(455, Coruña)	(171, Coruña)	(0, Coruña)

Este proceso puede repetirse tantas veces como ciudades tenga dentro de *puertos[ ]*, lo cual lleva siempre a la mejor opción.
Ahora que ya tengo la mejor solución para cada ciudad armo las rutas con el método *getRuta()* y luego las llevo al arreglo *retorno[ ]* que es que tiene la solución final.

En este seguimiento voy a armar **la ruta para llegar desde Badajoz al puerto más cercano**.

salida[ ] = vacío;

i = Badajoz;

Añado i a *salida[ ]*;

salida [Badajoz];

Entro en el while:

Primera iteración:

i = Madrid;

salida [Badajoz, Madrid];

Segunda iteración:

i = Valladolid;

salida [Badajoz, Madrid, Valladolid];

Tercera iteración:

i = Coruña;

salida [Badajoz, Madrid, Valladolid, Coruña];

Coruña tiene puerto así que **salgo del while.**

Devuelvo *salida[Badajoz, Madrid, Valladolid, Coruña]* que es la ruta armada.

El arreglo final retorno quedaría así:

Pos	0	1
	Badajoz	etc...
	Madrid	etc...
	Valladolid	etc...
	Coruña	etc...