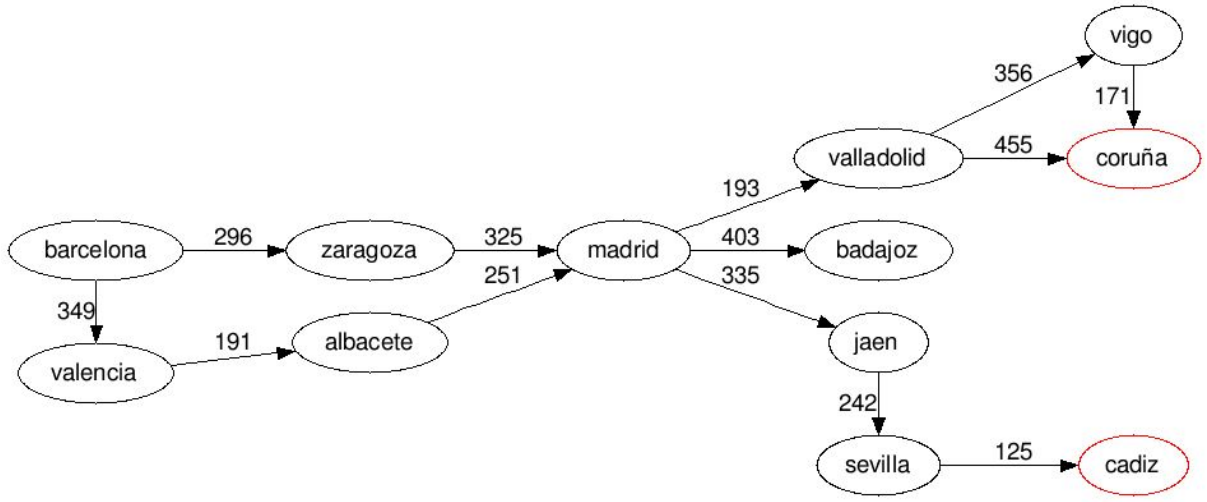
Ejercicio número 5 - Trabajo Práctico 4

Desde un cierto conjunto grande de ciudades del interior de una provincia, se desean transportar cereales hasta alguno de los 3 puertos pertenecientes al litoral de la provincia. Se pretende efectuar el transporte total con mínimo costo sabiendo que el flete es más caro cuanto más distancia tiene que recorrer. Dé un algoritmo que resuelva este problema, devolviendo para cada ciudad el camino que debería recorrer hacia el puerto de menor costo.

Grafo de ejemplo:



Los marcados en rojo son los puertos.

Pseudocódigo

```

Class Camino {
    conjunto<String nombre_ciudad, Integer distancia> ciudades;
    conjunto<String nombre_ciudad> puertos;
    //Constructor
    Camino();
    //Constructor que crea un camino vacio, this.ciudades = null;
    Camino(String);
    Boolean terminaEnPuerto();
}

Class ConjuntoCaminos {
    conjunto<Camino> caminos;

    Boolean esVacio();
    //Retorna el primer elemento del conjunto
    Camino getPrimerElemento();
    //Agrega los elementos ordenandolos por distancia total,
    //agregando al principio los de menor distancia total.
    Void agregarOrdenado(Camino c);
}
```

```

/*
 * Recibe el grafo de ciudades completo.
 * Recorre todas las ciudades y por cada una:
 * - Genera los caminos totales hacia los demás nodos
 * - Borra la ciudad actual del grafo
 * - Obtiene los caminos posibles a ser solucíon (terminan en un puerto)
 * - Selecciona el mejor y lo agrega a la solución
 * Devuelve un conjunto con la ciudad y su camino_minimo a un puerto para cada ciudad
 */
function voraz(Grafo g){
    conjunto<Camino, String> solucion;
    //inicialmente solución está vacía
    for (todos las ciudades de g : String ciudad_actual){
        //para cada elemento del grafo, señalado como ciudad_actual
        Camino camino_minimo = new Camino();
        ConjuntoCaminos caminos_totales = new ConjuntoCaminos();
        ConjuntoCaminos caminos_minimos = new ConjuntoCaminos();
        //inicializo variables
        ConjuntoCaminos caminos_totales = seleccionar(grafo g, ciudad_actual);
        g.borrar(ciudad_actual);
        ConjuntoCaminos caminos_minimos = factible(caminos_totales);
        solucion.agregar(Solucion(caminos_minimos), ciudad_actual);
    }
    return solucion;
}
```

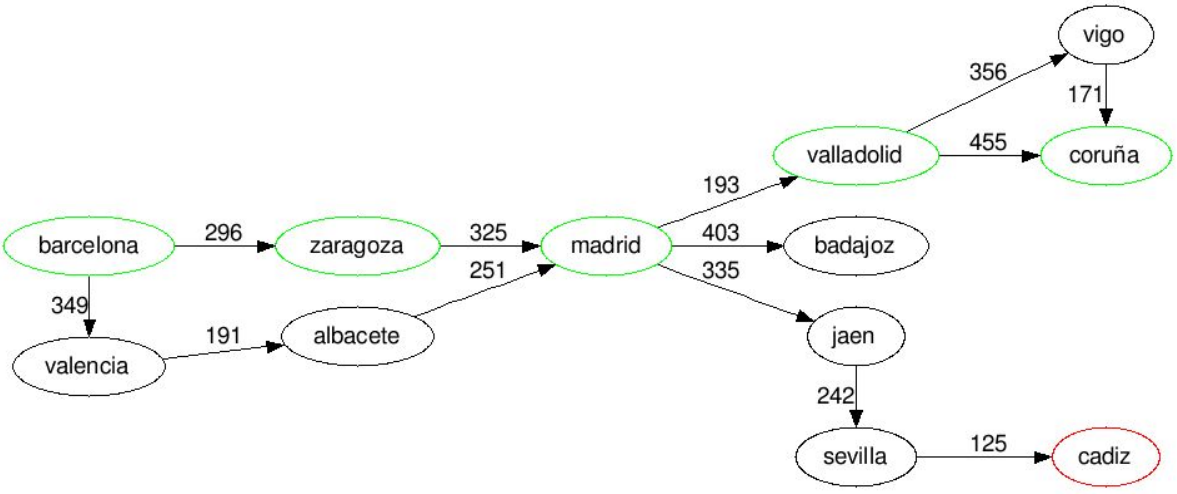
```
/*
 * Recibe el grafo completo y una ciudad para utilizar como origen.
 * A partir de la misma, utilizando el algoritmo de Dijkstra obtengo todos
 * los caminos posibles a las demás ciudades, retornandolos en una variable
 * de tipo ConjuntoCaminos.
 * @function seleccionar
 */
function seleccionar(Grafo g, Ciudad ciudad_actual) {
    return Dijkstra(g, ciudad_actual);
    //supongo que la funcion que llama a Dikstra devuelve todos los caminos
    //posibles desde la ciudad_actual hacia todos los demás elementos del grafo
}

/*
 * Recibe todos los caminos desde una ciudad hacia las demas.
 * Recorre todos y agrega a la solución (ordenados por peso)
 * aquellos en que el último nodo es un puerto.
 * @function factible
 */
function factible(ConjuntoCaminos caminos_totales) {
    ConjuntoCaminos caminos_solucion = new ConjuntoCaminos();
    for (todos los elementos de caminos_totales : Camino camino_actual) {
        if (camino_actual.terminaEnPuerto()) {
            caminos_solucion.agregarOrdenado(camino_actual);
        }
    }
    return caminos_solucion;
}

```

```
74
75  /*
76   * Recibe los caminos_minimos hacia cada puerto desde una ciudad;
77   * Si no está vacío, devuelve el primero, ya que como están ordenados
78   * el primero va a ser el más corto.
79   * Si está vacío, crea y retorna un nuevo camino "vacio" para agregar a la solución
80   * @function agregarSolucion
81   */
82  function agregarSolucion(ConjuntoCaminos caminos_minimos) {
83      if (!camino_minimos.esVacio()) {
84          return caminos_minimos.getPrimerElemento();
85      }
86      Camino camino_vacio = new Camino("vacio");
87      return camino_vacio;
88  }
89
```

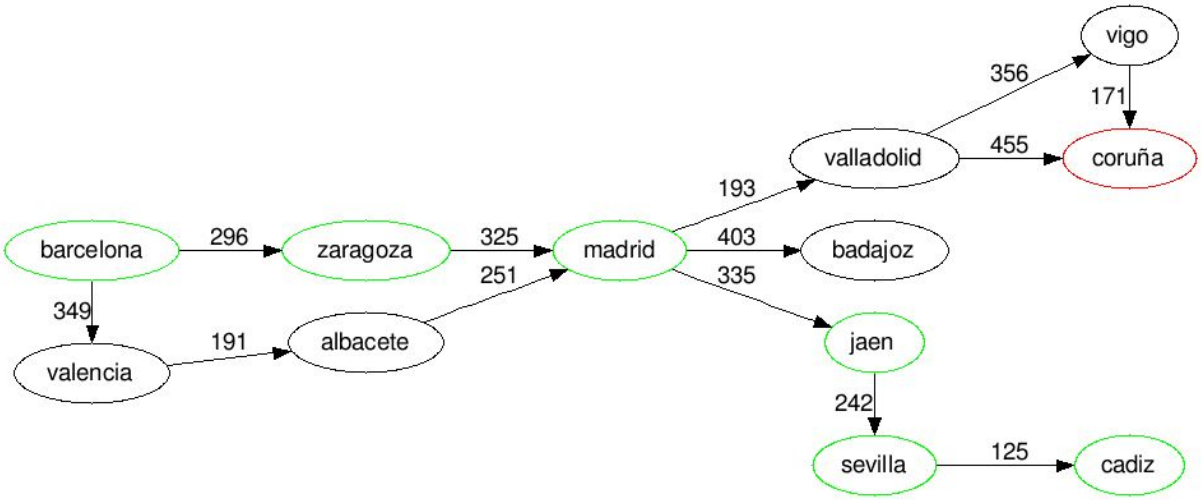
**conjunto es un tipo de datos a definir.*

Seguimiento visual:

Por ejemplo, tomando la primer ciudad “Barcelona” dentro de la función ***voraz***.
Genera todos los caminos posibles a todas las demás ciudades con la función ***seleccionar***.
factible Selecciona sólo los que terminan en puerto, para este ejemplo serían:
Mejor camino a Coruña



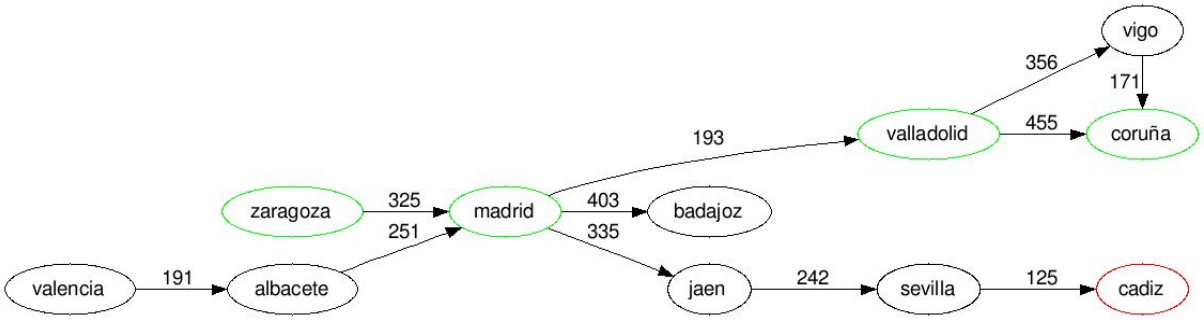
Mejor camino a Cadiz



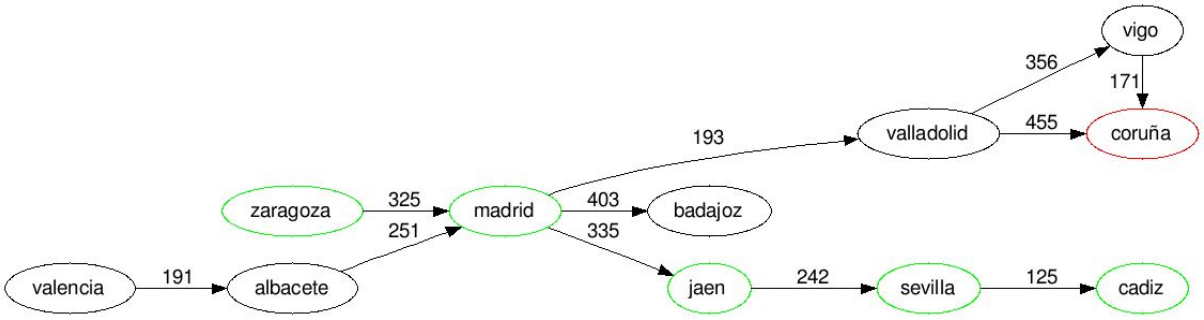
solución compara ambos caminos por distancia:
El que va a Coruña mide 1269
El que va a Cadiz mide 1323
Entonces se queda con Coruña y en el conjunto de soluciones agrega:

```
{
        "Barcelona",
         Camino {
                [Zaragoza, 296],
                [Madrid, 325],
                [Valladolid, 193],
                [Coruña, 455]
        }
}
```

Siguiente iteración
Tomando la ciudad "Zaragoza" dentro de la función ***voraz***.
Genera todos los caminos posibles a todas las demás ciudades con la función ***seleccionar***.
factible Selecciona sólo los que terminan en puerto, para este ejemplo serían:
Mejor camino a Coruña



Mejor camino a Cadiz



solución compara ambos caminos por distancia:
El que va a Coruña mide 973
El que va a Cadiz mide 1047

Entonces se queda con Cadiz y en el conjunto de soluciones agrega:

```
{
        “Zaragoza”,
         Camino {
                [Madrid, 325],
                [Jaen, 335],
                [Sevilla, 242],
                [Cadiz, 125]
        }
}
```

La solución hasta este momento sería:

```
{
        “Barcelona”,
         Camino {
                [Zaragoza, 296],
                [Madrid, 325],
                [Valladolid, 193],
                [Coruña, 455]
        }
},
{
        “Zaragoza”,
         Camino {
                [Madrid, 325],
                [Jaen, 335],
                [Sevilla, 242],
                [Cadiz, 125]
        }
}
```

Esto se repite para todas las ciudades del grafo.