

Segundo Parcial Técnicas de Documentación y Validación

Se está implementando un simulador para la gestión de vehículos eléctricos. Por ahora se dispone de:

Una clase *Arco* que define objetos con las propiedades:

- Nodo i
- Nodo j
- float w

indica la existencia de un camino entre un nodo i y un nodo j con un costo de energía w para ir de i a j. Siempre el nodo i es siempre menor al nodo j.

Una clase *Nodo* que define objetos con las propiedades:

- int id
- float g

Define que el pasar por el nodo id implica una ganancia de energía g

Por ejemplo:

```
nodo_i = new Nodo(1,100)
```

```
nodo_j = new Nodo (5,25)
```

```
arco_i_j = new Arco();
```

```
arco_i_j.setExtremos(nodo_i,nodo_j,50)
```

Transitar el camino nodo_i -> nodo_j tiene un resultante de energía de 75

Una clase *Camino* almacena una colección de *Nodos* y *Arcos* y provee funcionalidades para obtener:

- El camino con mayor energía resultante (`Nodo [] getEcoPath(int nodo_i, int nodo_j)`)
- El camino con mayor energía consumida (`Nodo [] getWorstPath(int nodo_i, int nodo_j)`)
- Energía remanente al final de un camino (`float getEnergy(Nodo [])`)

Implemente en JUnit:

- 1) Un test que compruebe que los extremos de un objeto arco deben respetar que el identificador del nodo inicial (i) es menor al identificador del nodo final (j), es decir $i < j$ y que ante esa situación inesperada se dispara la excepción `"CaminoInvertidoException"`
- 2) Un test que ante un único camino entre un nodo i y un nodo j, el camino propuesto por `getEcoPath` es idéntico al propuesto por `getWorstPath`

Implemente en TestNG

- 1) Un generador de caminos entre diferentes *Nodos*
- 2) Un test que verifique el cálculo de energía remanente (`getEnergy(Nodo [])`)