UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

TAREA 1: GENERACIÓN DEL GRAFO

Sergio González Velázquez
Andrés Gutiérrez Cepeda
David Carneros Prado

Asignatura: Sistemas Inteligentes

Grupo de Trabajo: BC1

Titulación: Grado en Ingeniería Informática

Fecha: 5 de octubre de 2018

➢ 1. Objetivo de la tarea

El objetivo de la tarea 1 consiste en implementar una clase Grafo que contenga:

-Un **constructor** que reciba como parámetro el fichero graphml.

-Un método ***perteneceNodo*** que tenga como parámetro el id de un nodo y devuelva si dicho nodo pertenece o no al grafo.

-Un método ***posiciónNodo*** que tenga como parámetro el id de un nodo y devuelva longitud y latitud de dicho nodo.

-Un método ***adyacentesNodo*** que tenga como parámetro el id de un nodo y devuelva una lista de las aristas adyacentes a dicho nodo.

➢ 2. Herramientas utilizadas

▪ 2.1 Lenguaje de programación.

Hemos decidido que para la realización de las prácticas de Sistemas Inteligentes utilizaremos el lenguaje de programación **Python 3**. La principal motivación de esta decisión es que nos gustaría aprender a programar en este lenguaje, pues creemos que es bastante versátil y puede utilizarse para el desarrollo cualquier tipo de tarea. Por tanto, consideramos que es muy recomendable aprender a programar en Python y nos puede resultar útil en el futuro.

Además, otra de las razones que nos conducen a decantarnos por Python es que, al tratarse de un lenguaje de código de abierto, podemos encontrar una gran cantidad de publicaciones que nos ayudarán a la hora de desarrollar nuestro trabajo.

▪ 2.2 Paquete de gestión de grafos

Concretando un poco más, utilizaremos como paquete de gestión de grafos **NetworkX.** Se trata de una biblioteca de Python utilizada para crear y manipular estructuras de redes complejas. En nuestro caso, utilizaremos las funciones que proporciona para leer un grafo en formato *graphml* y su posterior manipulación. Para ello, hemos consultado la documentación de esta biblioteca:

https://networkx.github.io/documentation/stable/

Es necesario descargar e instalar el módulo de la librería NetworkX para poder hacer uso de sus funciones. Aunque existen diferentes formas de instalar módulos externos en Python, nosotros hemos utilizado ***pip***.

En Linux, que es el sistema operativo que usaremos para el desarrollo de nuestra práctica, basta con escribir la siguiente secuencia de comandos:

```
sudo apt-get install python3-pip

sudo apt-get update

sudo pip3 install setuptools

sudo pip3 install networkx
```

- **2.3 Repositorio de código privado.**

Utilizaremos un repositorio de código privado en **github** que nos permitirá de tener un control de las versiones de nuestra práctica además de ofrecernos la posibilidad de trabajar de forma paralela a todos los miembros del grupo de trabajo.

https://github.com/DavidCarneros/BC1-01.git

➢ 3. Implementación

- **3.1 Constructor**

En el constructor de la clase Grafo simplemente utilizamos la función ***read_graphml*** de NetworkX que permite leer un grafo en formato GraphML desde una ruta dada (*path*) y devolver un objeto de tipo *NetworkX graph.*

```
def __init__(self,path): #Constructor
    self.__graph = read_graphml(path)
```

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.readwrite.graphml.read_graphml.html

- **3.2 Método *perteneceNodo***

El propósito de este método es determinar si un nodo pasado como parámetro pertenece o no al grafo. Para ello, primeramente, hacemos uso de la función ***l*ist** de NetworkX que retorna una lista con todos los nodos del grafo. A continuación, recorremos dicha lista hasta encontrar el id del nodo pasado como parámetro. En caso de terminar este recorrido y no haber encontrado el nodo, el método devuelve False.

```
def perteneceNodo(self, idNodo):
    listaNodos=list(self.__graph.nodes)
    for i in listaNodos:
        if idNodo==i:
            return True
    return False
```

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.Graph.nodes.html

- **3.3 Método *posicionNodo***

El método *posiciónNodo* se utiliza para obtener la posición (longitud,latitud) de un nodo pasado como parámetro.

En primer lugar, lo que hacemos es comprobar si dicho nodo pertenece o no al grafo utilizando el método ***pertenceNodo*** implementado anteriormente. En caso de que el nodo no pertenezca al grafo se retorna *'Error'* y finaliza la ejecución del método.

Si por el contrario el nodo si pertenece al grafo, utilizamos la función **get_node_attributes** de NetworkX que devuelve un diccionario con un determinado atributo (indicado como parámetro en la función) de cada uno de los nodos del grafo. Es por eso, que nosotros obtenemos dos diccionarios, uno con todas las latitudes y otro con todas las longitudes. Finalmente, devolvemos únicamente la longitud y latitud del nodo indicado.

```
def posicionNodo(self, idNodo):
    if  self.perteneceNodo(idNodo) == True:
        attrX=get_node_attributes(self.__graph,"x")
        attrY=get_node_attributes(self.__graph,"y")

        return [attrX.get(idNodo),attrY.get(idNodo)]
    else:
        return "Error"
```

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.classes.function.get_node_attributes.html

- **3.4 Método *adyacentesNodo***

Una vez que se ha comprobado que el nodo pasado como parámetro existe, utilizamos la función edges de la biblioteca networkx que devuelve una vista (EdgeView) de las aristas adyacentes a dicho nodo. Como el segundo parámetro pasado a la función *edges* lo ponemos a True, conseguimos un diccionario con todos los atributos de cada una de las aristas adyacentes al nodo dado.

La variable local 'aristas' contiene dicha vista. Cada elemento de ella es una lista que contiene el nodo origen en la posición el nodo destino en la posición 1 y un diccionario con las propiedades de dicha arista en la posición 2.

Recorremos la vista 'aristas' introduciendo en 'lista' los atributos nodo origen (i0), nodo destino (i1), nombre y longitud de las aristas (que se encuentran en un diccionario con las propiedades de esa arista (i2)).

```
def adyacentesNodo(self, idNodo):
    lista = []
    if self.perteneceNodo(idNodo) == True:
        aristas=self.__graph.edges(idNodo,True)
        for i in aristas:
            lista.append((i[0],i[1],i[2].get("name"),i[2].get("length")))
        return lista
    else:
        return "Error"
```

https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.Graph.edges.html