ESTRUCTURAS DE DATOS

Curso 2025/26

PRÁCTICA 8

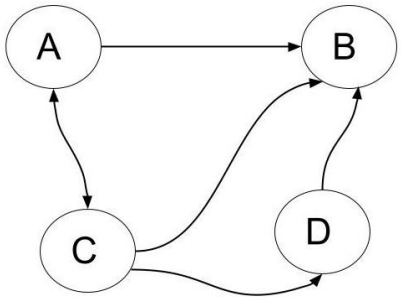
Grafos

Instrucciones

- *Se debe completar en una sesión.*
- *Práctica individual.*
- *Lee el enunciado completo antes de comenzar.*
- *Se habilitará una tarea para que entregues el código desarrollado.*
- *La práctica será APTA si se superan todos los test de validación proporcionados.*

En esta práctica se va a utilizar un grafo para representar los individuos que forman parte de una red social y las relaciones entre ellos. La red social que se va a representar es estilo Instagram o X, donde una persona puede seguir a otra pero esta relación no es recíproca, por ello se va a utilizar un grafo dirigido. El grafo se representará como un mapa **`Map<String, Set<String> > graph`**, donde para cada nodo se guarda su conjunto de arcos. Por ejemplo, para el grafo de la figura, el mapa tendrá las siguientes claves y valores:

Clave	valores
"A"	{"B", "C"}
"B"	{ }
"C"	{"B", "D", "A"}
"D"	{"B"}



Observar que aunque el nodo "B" no tenga arcos de salida, también forma parte de las claves.

Se proporcionan los ficheros **`MiRedSocial.java`**, donde se implementarán los métodos que se piden a continuación, y **`MiRedSocialJUnit.java`** con los test unitarios. Además, se proporcionan los ficheros **`miredsocial.txt`** y **`miredsocialNueva.ref`** con los datos de una pequeña red social para hacer pruebas. Estos dos últimos ficheros no deben estar dentro de la carpeta **`src`** de la estructura del proyecto, sino en la raíz del mismo.

Ejercicio 1

Implementa el método **`addDirectedEdge`** de la clase **`MiRedSocial`** que añade un nuevo arco dirigido a la red, desde el nodo **`source`** al nodo **`target`**. El método devuelve **`true`** si el arco se

inserta y **false** en caso contrario, teniendo en cuenta que no se permiten arcos repetidos y que no se permiten arcos entre un nodo y si mismo (es decir, no se permiten arcos de A a A). En caso de que **source** o **target** sean **null**, el método lanzará la excepción **NullPointerException**.

El método debe añadir los nodos al grafo en caso de que no existan.

```
public boolean addDirectedEdge (String source, String target);
```

Ejercicio 2

Implementa el método **containsNode** de la clase **MiRedSocial** que dado un nombre **name**, devuelva **true** si pertenece a la red y **false** en caso contrario. Si name es **null**, el método lanzará la excepción **NullPointerException**.

```
public boolean containsNode (String name);
```

Ejercicio 3

Implementa el método **size** de la clase **MiRedSocial** que devuelva un entero con el tamaño (número de nodos) de la red.

```
public int size ( );
```

Ejercicio 4

Implementa el método **getMembers** en la clase **MiRedSocial** que devuelva una copia del conjunto de miembros (nodos) de la misma.

```
public Set<String> getMembers( );
```

Ejercicio 5

Implementa el método **getFollowers** en la clase **MiRedSocial** que dado un nombre **name**, devuelva todos sus seguidores.

```
public Set<String> getFollowers(name);
```

Por ejemplo, para la red de la figura, **getFollowers("B")** devolverá **{"A","B","C"}**, o **getFollowers("A")** devolverá **{"C"}**. Si **name** no pertenece a la red devolverá un conjunto vacío y si **name** es **null** lanzará la excepción **NullPointerException**.

Ejercicio 6

Implementa el método **getFollowed** en la clase **MiRedSocial** que dado un nombre **name**, devuelva una copia de todos los miembros a los que sigue.

```
public Set<String> getFollowed(name);
```

Por ejemplo, para la red de la figura, **`getFollowed("B")`** devolverá **`{ }`**, o **`getFollowed("A")`** devolverá **`{"C", "B"}`**. Si **`name`** no pertenece a la red devolverá un conjunto vacío y si **`name`** es **`null`** lanzará la excepción **`NullPointerException`**.

Ejercicio 7

Implementa el método **`commonFollowers`** en la clase **`MiRedSocial`** que dado dos nombres **`name1`** y **`name2`** devuelva los seguidores comunes a ambos. Si **`name1`** o **`name2`** son **`null`** lanzará la excepción **`NullPointerException`**.

```
public Set<String> commonFollowers(String name1, String name2);
```

Por ejemplo, para la red de la figura, **`commonFollowers("B", "D")`** devolverá **`{"C"}`**, o **`commonFollowers("A", "C")`** devolverá **`{ }`**.

Ejercicio 8

Implementa el método **`suggestions`** en la clase **`MiRedSocial`** que dado un nombre **`name`** devuelva un conjunto con sugerencias de amistad. Para un **`name`** dado, se entiende como sugerencias de amistad, aquellos miembros de la red a los que NO sigue pero son seguidos por alguno de sus amigos. Por ejemplo, **`suggestions("A")`** devolverá **`{"D"}`** porque **`"B"`** sigue a **`"D"`** y **`"A"`** sigue a **`"B"`**" pero no sigue a **`"D"`**. El resultado no debe incluir el propio nodo. Si **`name`** no pertenece a la red, devolverá **`null`**. Y si **`name`** es **`null`** lanzará la excepción **`NullPointerException`**.

```
public Set<String> suggestions(String name);
```

Ejercicio 9

Implementa el método **`mostInfluencer`** en la clase **`MiRedSocial`** que devolverá el nombre del nodo que más seguidores tiene en la red. Por ejemplo, en el caso de la figura devolverá el nodo "B".

```
public String mostInfluencer( );
```

Este método no tiene test propio y se comprueba en el último test.

Ejercicio 10

La *hipótesis de los seis grados de separación* intenta probar que cualquier persona en la Tierra puede estar conectado a cualquier otra persona del planeta a través de una cadena de conocidos que no contiene más de cinco intermediarios. Vamos a calcular los grados de separación (o la distancia que separa) a los individuos que forman parte de nuestra red social.

Implementa en la clase **`MiRedSocial`** el método **`distanceToAll`** que dado un nodo inicial **`name`** devuelva un mapa con la distancia más corta a cada uno del resto de miembros (nodos) de la red. Si **`name`** es **`null`** lanzará la excepción **`NullPointerException.`**

```
public Map<String, Integer> distanceToAll(String name);
```