Ejercicio 3 [3 puntos].

El cacheado de ficheros en servidores web reduce el coste de tener que cargar dicho fichero de disco cada vez que se realiza una petición. Una caché MRU (*Most Recently Used*) permite establecer estrategias acerca de qué ficheros (objetos `MyFile`) permanecen en cache, controlando así su tamaño. En concreto, cuando se hace la petición al servidor web de un fichero concreto puede ocurrir que:

- No se encuentra en la cache
  - Si la caché no está llena, el fichero se lee de disco y se guarda en la caché. A continuación se devuelve el fichero consultado (objeto `MyFile`).
  - Si la caché está llena, el fichero cacheado que lleve más tiempo sin ser accedido se elimina. A continuación, el fichero accedido se lee de disco y se guarda en la caché. Por último se devuelve el fichero consultado (objeto `MyFile`).
- Sí se encuentra en la caché. Entonces se recupera y se devuelve el fichero consultado (objeto `MyFile`).

El comportamiento de una cache MRU queda descrito por la clase `MRUCache` que se presenta al final del enunciado. Esta clase contiene un método `getFile()` que dado el nombre de un fichero (`fname`) permite recuperarlo (bien de cache o bien de disco) según los criterios descritos anteriormente. Para poder recuperar el fichero de disco, la clase `MRUCache` contiene un método `readFileFromDisk()` que dado el nombre del fichero es capaz de leer su contenido y almacenarlo en un objeto de la clase `MyFile`. Por último, es posible mostrar el estado de la cache (nombre de los ficheros cacheados en un instante concreto) llamando al método `printMRU()`.

Se pide:

1) Seleccionar la/s estructura/s de datos necesarias para implementar la cache MRU.

2) Añadir los atributos necesarios para que la clase `MRUCache` pueda hacer uso de las estructuras de datos seleccionadas en el punto anterior.

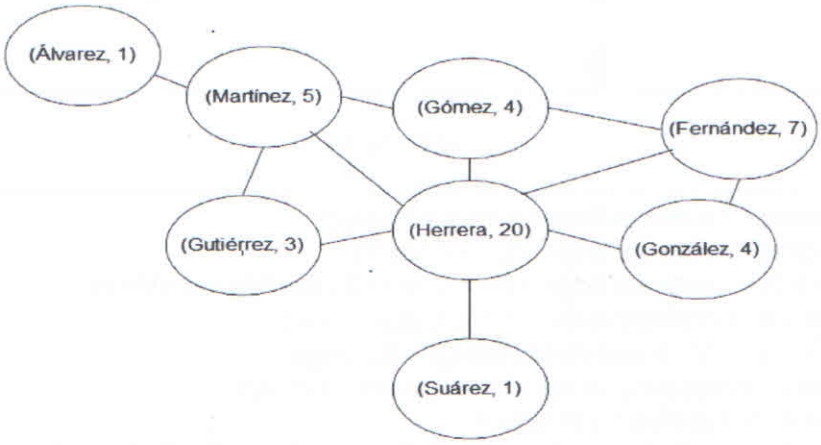
3) Implementar los métodos `getFile()` y `printMRU()`.

NOTA: se requiere que las consultas de cache tengan lugar en el menor tiempo posible, por lo que se tendrá en cuenta la complejidad algorítmica de la solución dada.

Ejercicio 3 [2,5 puntos]

Se desea usar el TAD Grafo para representar las colaboraciones entre un conjunto de autores o investigadores pertenecientes a una universidad y en un año concreto. Este grafo de colaboraciones almacena en los nodos el nombre de cada autor junto con el número de sus trabajos publicados en un año. Por otro lado, dos autores se conectan por una arista si han publicado, al menos, un trabajo juntos. Vamos a considerar en este ejercicio que dicho grafo ya está construido y también que el grafo es siempre conexo.

Un posible ejemplo de grafo de colaboraciones puede ser:



Se pide desarrollar métodos en Java para implementar las siguientes operaciones sobre el grafo descrito anteriormente:

(a) **[1,25 puntos]** Obtener un listado ordenado de todos los autores por el número de publicaciones de cada autor. Dicho método tiene la sigiente cabecera:

```
Collection <Autor> getRanking (Graph<Autor,Integer> grafo);
```

(b) **[1,25 puntos]** Utilizando el algoritmo de búsqueda en anchura en un grafo, obtener la colección de autores que están a distancia de *k* arcos de un autor dado en el grafo de colaboraciones. Dicho método tiene la siguiente cabecera:

```
Collection <Autor> getDistanciaK
        (Graph<Autor,integer> grafo, Vextex<Autor>, int distancia);
```