Estructuras de Datos y Algoritmos
Práctica I – Curso 2025/26

La Red Social

Versión 28/09/2025

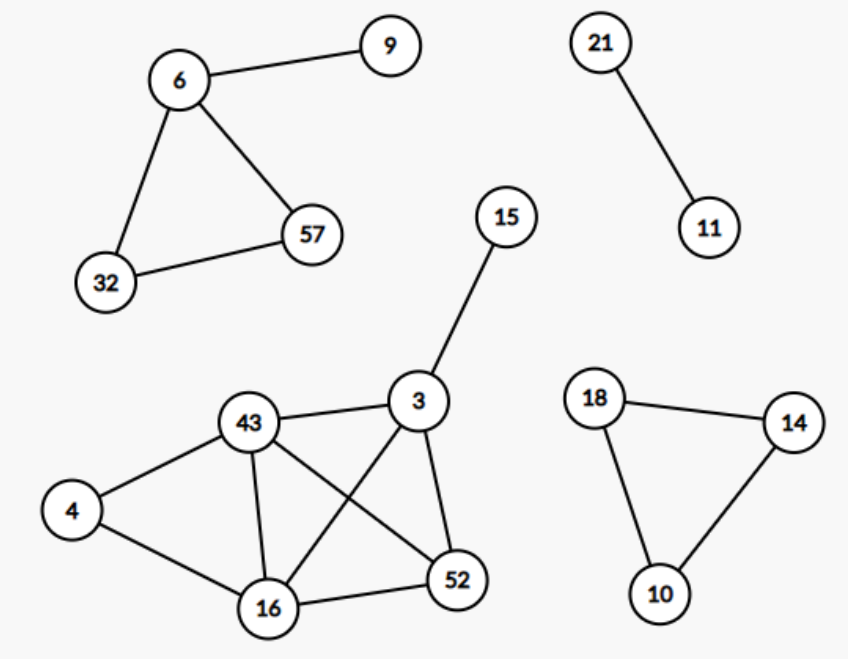
1. Introducción

Enhorabuena!! El CEO de la nueva red social **Y** te ha contratado para un trabajo muy importante que puede suponer tu salto a la fama. Esta red social, especializada en la importante tarea de divulgación de memes virales, ha tenido un crecimiento explosivo en su número de usuarios pero es necesario que se consolide para que empiece a proporcionar beneficios económicos.

La red está organizada en base a "conexiones de amistad" entre pares de usuarios. Estas conexiones son bidireccionales: Si A es amigo de B automáticamente B es amigo de A y ambos han tenido que autorizar la conexión. Cuando un usuario publica un meme, este es enviado inmediatamente a todos sus amigos, y cada uno de ellos puede decidir si se reenvía a su vez a sus amigos, y así sucesivamente. El sistema controla que si el mismo meme es recibido por distintos caminos de la red solo se muestre una vez.

Los usuarios de **Y** no tienen sentimientos de lealtad hacia la red, su único motivo para permanecer en ella es que les proporcione su ración diaria de memes graciosos. Si perciben que las otras redes sociales a las que están apuntados les proporcionan más memes graciosos que **Y** entonces la abandonarán.

Se denomina **grumo** a un conjunto de usuarios que son mutuamente accesibles en la red. En el siguiente gráfico se muestra una red de 15 usuarios donde existen 4 grumos (los números son los identificadores de los usuarios y cada línea indica una conexión de amistad):



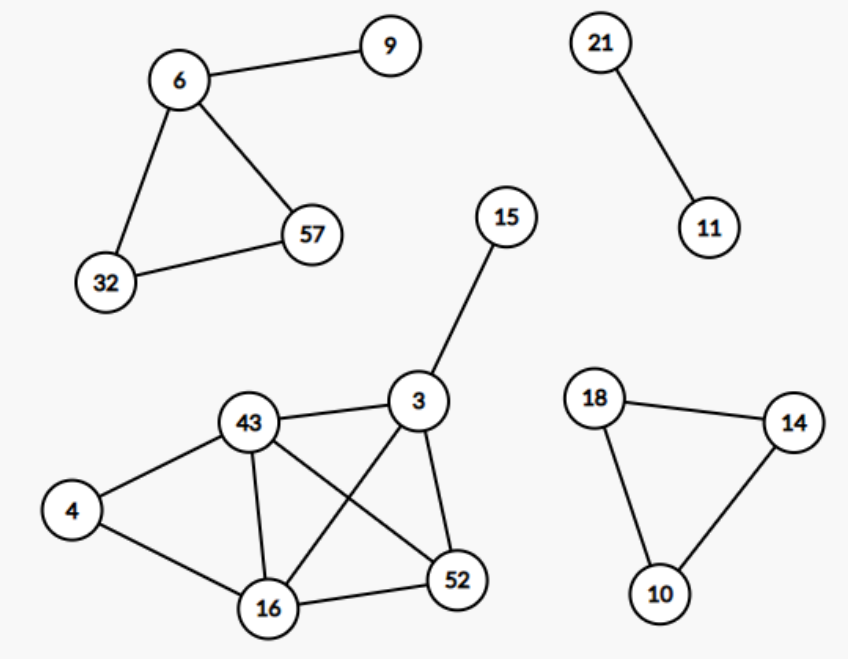
Si un meme es gracioso la mayoría de los usuarios que lo reciban lo reenviarán y por lo tanto es casi seguro que llegará a todos los usuarios del grumo donde se originó. Sin embargo, **los usuarios que no pertenezcan al grumo no lo recibirán**.

Por ese motivo es muy importante para la supervivencia de **Y** que su red tenga una conectividad muy alta, es decir que la gran mayoría de sus usuarios pertenezcan a un mismo grumo, con lo cual está garantizado que cualquier meme gracioso se distribuya a prácticamente todos los usuarios de la red.

Desafortunadamente los usuarios de esta red no suelen establecer muchas conexiones de amistad entre ellos y en general el grumo de mayor tamaño no suele contener más del 20% de los usuarios. Para resolver este problema se les ha ocurrido la idea de **pagar a usuarios concretos para que se hagan amigos**, conectando de esa forma los grumos de mayor tamaño hasta que se consiga tener un grumo que contenga a más de un determinado porcentaje de usuarios.

Si tomamos como ejemplo el gráfico de la página anterior, podemos apreciar que existen cuatro grumos de tamaños 6, 4, 3 y 2 usuarios, en orden descendente. Si, por ejemplo, el objetivo fuera conseguir que en un único grumo estuviesen más del 85% de los usuarios,

podemos ver que uniendo los 3 grumos más grandes tendríamos el (6+4+3)/15 = 87% de los usuarios en él. Una forma de conseguirlo sería pagar a los usuarios **32** y **43** para que se hicieran amigos y hacer lo mismo con los usuarios **3** y **18**:



Por supuesto existen otras muchas posibilidades, para unir dos grumos basta con se hagan amigos cualquier usuario del primero con cualquier usuario del segundo. Es importante que se pague a la menor cantidad posible de personas, por eso se deben unir los grumos mayores, y es fácil observar que para unir n grumos basta con crear $n - 1$ nuevas relaciones de amistad.

Tu objetivo es el crear una aplicación a la que se proporcionará el estado actual de la red (el listado con las conexiones de amistad existentes) y el porcentaje mínimo de usuarios que queremos que estén en el grumo de mayor tamaño. La aplicación analizará la red detectando los grumos existentes y su tamaño, y si detecta que el grumo de mayor tamaño no contiene un determinado porcentaje mínimo de usuarios, entonces presentará una propuesta de relaciones de amistad "por dinero" con las que se podría conseguir ese objetivo.

2. Descripción Técnica

La información sobre la red se proporciona como un fichero de texto que contiene una línea por cada conexión de amistad, con el siguiente formato: Cada línea consta de dos enteros separados por un espacio en blanco y esos enteros son los identificadores de un par de usuarios que tienen una conexión de amistad entre sí.

Propiedades del fichero:

- Solo existe una línea para cada conexión de amistad, si el usuario 32 está conectado con el 45, entonces aparecerá o bien la línea "32 45" o bien la línea "45 32", pero no ambas.
- Las líneas de conexiones no están ordenadas.
- Los identificadores de usuarios pueden ser cualquier entero, no son correlativos.

Ejemplo: En el recuadro del margen derecho se muestra el contenido de un fichero que describe la red que aparece en el gráfico de la primera página (15 usuarios, 17 conexiones).

```
43 52
52 16
43 16
43 4
4 16
43 3
15 3
57 6
6 32
57 32
6 9
11 21
18 10
14 18
10 14
3 52
16 3
```

ejemplo.txt

2.1. Estructuras de Datos

En esta primera práctica todas las estructuras que se van a utilizar son listas secuenciales (clase **ArrayList** de Java). En ellas se van a poder usar las siguientes operaciones predefinidas:

- Acceso basado en índice (método predefinido **get**)
- Inserción al final (métodos predefinidos **add** o **addAll**)
- Búsqueda secuencial (método predefinido **contains**)
- Ordenación (método predefinido **sort**)

Para representar las conexiones se sugiere crear una clase sencilla con dos atributos enteros en Java (los dos enteros son los identificadores de los usuarios con conexión de amistad).

Las estructuras necesarias son:

- Una lista de conexiones (**red**), que se obtiene del fichero.
- Una lista con los identificadores de los usuarios (**usr**), que se debe obtener procesando la lista de conexiones. Es una lista de enteros, sin ordenar.
- Una lista de grumos (**grus**). Obtenerla es el objetivo principal de la aplicación. Es una lista de listas de enteros (cada elemento es la lista de usuarios que pertenece al grumo)
- Una lista con los usuarios que ya han sido procesados como pertenecientes a un grumo (**asig**). Esta lista sirve de ayuda en el proceso de detección de grumos, para evitar procesar más de una vez a cada usuario. Es una lista de enteros.

2.2. Algoritmo

El algoritmo para resolver el problema en esta primera práctica **está prefijado**. Si tenéis una solución mejor o alguna optimización reservadla para la segunda práctica.

La parte central del algoritmo consiste en obtener todos los usuarios de un grumo. Se va a resolver mediante el algoritmo clásico de **búsqueda en profundidad** en grafos. Se implementará mediante una función, **uber_amigos**, que tiene como parámetros a un usuario inicial, la lista red y una lista de usuarios (**grumo**) que representa el resultado de la función y que se va a ir ampliando con todos los usuarios que pertenezcan al mismo grumo. El proceso es muy sencillo: Se recorre la lista red buscando conexiones a las que pertenezca el usuario inicial. Para cada una de ellas el otro usuario es un amigo directo suyo. Si ese otro usuario todavía no pertenece a **grumo**, se le añade y la función se llama **recursivamente** a sí misma con ese otro usuario en el papel del usuario inicial (para así añadir a los amigos de ese amigo, *ad nauseum*, al grumo).

El algoritmo general consta de las siguientes etapas:

1. **Lectura del fichero** (o los ficheros) de datos: El programa pedirá el nombre del fichero con los datos de conexiones, que tendrá el formato indicado al principio del apartado. Con los datos de este fichero se creará la estructura **red**. A continuación pedirá el nombre de un fichero con conexiones extra, si se pulsa [enter] sin introducir un nombre no hará nada, si se introduce un nombre leerá el fichero, que contiene conexiones extra con el mismo formato que el fichero anterior ~~pero ahora sin las dos primeras líneas con los valores de n y m.~~ El objetivo de este segundo fichero es poder comprobar fácilmente si la solución del problema obtenida en una ejecución anterior del programa es o no correcta.
2. **Creación de la lista de usuarios**: El objetivo es crear la lista **usr** con los identificadores de los usuarios. Para ello se recorren todas las conexiones almacenadas en red y si alguno de los dos usuarios de cada conexión no está todavía en **usr** se les añade.
3. **Creación de la lista de grumos**: Se van a recorrer todos los usuarios (lista **usr**), y para cada uno, si no está en la lista **asig**, se añadirá a la lista de grumos (**grus**) el grumo al que pertenece (obtenido mediante la función **uber_amigos**). Es fundamental el no repetir trabajo ya realizado, por ello va a existir una lista de usuarios (**asig**) que almacene todos los usuarios para los que ya se ha detectado el grumo al que pertenecen. Para ello, cada vez que obtengamos un nuevo grumo (y lo añadamos a **grus**), incluiremos a todos sus usuarios en **asig**.
4. **Ordenación y selección de los grumos**: Se ordenará la lista de grumos (**grus**) en orden descendente según su tamaño (número de usuarios de cada grumo). Mediante un bucle se irán sumando el número de usuarios de cada grumo (debido a la ordenación vamos procesando los grumos del mayor hacia el menor) hasta que consigamos tener un porcentaje superior al requerido (el programa pedirá que se introduzca ese porcentaje justo después de pedir el nombre de los ficheros). Si se necesita unir más de un grumo entonces mostramos las nuevas conexiones de amistad necesarias (se escoge unir el segundo usuario de un grumo con el primer usuario del grumo siguiente).
5. **Salvar la lista de nuevas relaciones**: Si en la etapa anterior se ha detectado que se necesita unir más de un grumo, la lista de conexiones de amistad necesarias se salvará automáticamente en un fichero con nombre **extra.txt** y con el formato estándar ~~(líneas de conexiones solamente, no deben aparecer las 2 líneas iniciales con n y m)~~.

En el recuadro de la página siguiente se muestra un ejemplo del aspecto que debe tener una ejecución del programa (en color azul los datos introducidos por el usuario):

```
ANALISIS DE Y
-------------
Fichero principal: test10000.txt
Fichero de nuevas conexiones (pulse enter si no existe):
Porcentaje mayor tamano grumo: 23
Lectura fichero: 0,03522
Creacion lista usuarios: 0,16389
Creacion lista grumos: 0,92940
Ordenacion y seleccion de grumos: 0,00077

10000 usuarios, 29027 conexiones
Existen 100 grumos.

Se deben unir los 5 mayores
#1: 641 usuarios (6,41%)
#2: 484 usuarios (4,84%)
#3: 450 usuarios (4,50%)
#4: 444 usuarios (4,44%)
#5: 370 usuarios (3,70%)
Nuevas relaciones de amistad (salvadas en extra.txt)
14152263 29482837
66968937 39015977
7820301 22854829
32029721 63448340
```

```
14152263 29482837
66968937 39015977
7820301 22854829
32029721 63448340
```

Fichero extra.txt

Los tiempos que nos interesan

Información que escribe por pantalla el método **informe** (ver Anexo)

Se puede apreciar que se mide el tiempo empleado en las etapas 1, 2, 3 y 4. En esta ejecución se ha creado el fichero **extra.txt** con el contenido mostrado en el recuadro de la derecha.

Para comprobar si el resultado es correcto se ejecuta otra vez el programa, esta vez pidiendo que se añadan las conexiones salvadas en **extra.txt**:

```
ANALISIS DE Y
-------------
Fichero principal: test10000.txt
Fichero de nuevas conexiones (pulse enter si no existe): extra.txt
Porcentaje mayor tamano grumo: 23
Lectura fichero: 0,03325
Creacion lista usuarios: 0,16791
Creacion lista grumos: 0,94975
Ordenacion y seleccion de grumos: 0,00077
10000 usuarios, 29031 conexiones
Existen 96 grumos.
El mayor grumo contiene 2389 usuarios (23,89%)
No son necesarias nuevas relaciones de amistad
```

3. Objetivos

Además de crear la aplicación, el **objetivo principal** de la práctica será el evaluar su eficiencia respecto al tiempo, realizando una serie de medidas del tiempo empleado para distintos tamaños del fichero de entrada.

Para ello en el Campus Virtual se proporcionarán varios ficheros de tamaños distintos y, como se puede apreciar en el apartado anterior, la aplicación debe mostrar el tiempo empleado en las distintas etapas .

Solamente nos interesan las etapas **2, 3 y 4** y la suma de sus tiempos (que llamaremos **total**). En esta primera práctica la etapa 4 suele tardar un tiempo despreciable y no es necesario dedicar mucho tiempo a su análisis. Respecto al tamaño de la entrada se puede usar o bien el número de conexiones ($\boldsymbol{m}$) o bien el número de usuarios ($\boldsymbol{n}$), ya que ambos valores son proporcionales ~~prácticamente iguales~~ en el tipo de redes que vamos a utilizar.

El análisis consistirá en:

- **Obtener las medidas**: Tiempos promedio de las etapas 2, 3, 4 y el total para distintos tamaños ($\boldsymbol{m}$ o $\boldsymbol{n}$) de las redes.
- Cargar las medidas en una hoja de cálculo o programa similar para poder **representar gráficamente las medidas** y obtener una **fórmula** que **estime su tipo de crecimiento**.
- No es obligatorio, pero puede ser conveniente Llevar a cabo un **análisis teórico** de la complejidad de las etapas 2, 3 y 4 para tener una idea de lo que debería obtenerse. Si se quiere realizar este análisis se puede suponer los siguientes órdenes para las operaciones sobre listas:
  - Acceso por índice: $O(1)$
  - Inserción al final: $O(1)$ amortizado
  - Búsqueda secuencial: $O(n)$
  - Ordenación: $O(n \log n)$
- Con los resultados anteriores, estar en condiciones de poder **realizar una estimación** (extrapolación) del tiempo **total** que tardará la aplicación para un tamaño dado (como referencia se va a suponer que **Y** tiene actualmente 10.000.000 de usuarios y que su objetivo es llegar a 100.000.000 de usuarios el próximo año)

4. Documentación para presentar: Análisis de Eficiencia

Debéis presentar (mediante una tarea que se abrirá en el Campus Virtual) un documento PDF donde indiquéis los resultados del análisis. No es necesario que os extendáis demasiado (con 1 o 2 páginas es suficiente) ni que torturéis a ChatGPT para incluir un montón de párrafos de calidad literaria: Solo tenéis que incluir vuestros nombres y grupo de prácticas, y **para cada una** de las etapas 2, 3, 4 y el total indicar lo siguiente:

- Fórmula de la eficiencia en función del tamaño de la red ($\boldsymbol{m}$ o $\boldsymbol{n}$, lo que halláis elegido), expresada con el término de mayor crecimiento indicado de forma exacta y el resto de términos en notación asintótica.
- La(s) tabla(s) de las medidas experimentales: Para cada tamaño el tiempo promedio, el número de repeticiones y cualquier otra métrica que hayáis utilizado.
- La(s) gráfica(s) de los valores de la(s) tabla(s) con la función de ajuste.
- La extrapolación global del tiempo total para $n = 10.000.000$ y $n = 100.000.000$

Nota: No es necesario usar una tabla o gráfica distinta para cada etapa, si queréis podéis agrupar toda la información en una única tabla y en un único gráfico (si se ven claramente los resultados, claro).

Recomendaciones para el análisis:

- Escoger un número suficiente de valores de tamaños de entrada para que el ajuste sea razonable.
- Los tamaños de entrada grandes son mucho más relevantes que los pequeños a la hora de escoger la función de ajuste.
- El número de repeticiones para calcular el promedio se escogerá observando la variabilidad de las medidas.
- El número de repeticiones puede ser distinto para cada tamaño de la entrada. Tened en cuenta que para tamaños grandes el tiempo que tarda el algoritmo en evaluarse puede ser muy grande, en este caso es razonable usar menos repeticiones.
- Para la representación gráfica y obtención de estimaciones se aconseja usar programas tipo hoja de cálculo o similares. Aunque es posible usar scripts en R o programas avanzados como Statistica, Derive o Mathematica la tarea es suficientemente sencilla para que no sean necesarios y suelen dar problemas de sobreajuste a la hora de inferir las funciones que indican la complejidad.

5. Defensa y evaluación de la Práctica

En la defensa se pedirá una modificación sencilla de vuestro código y que realicéis rápidamente una recogida de datos (tiempo promedio) para obtener una estimación de la eficiencia del algoritmo modificado (por supuesto con tamaños adecuados para que todo se pueda hacer en unos minutos).

La evaluación de la práctica se divide en dos etapas:

1. Presentación electrónica del documento PDF con el análisis y de los ficheros ***.java** que contienen el código del programa creado para obtener las medidas. Se habilitará en el Campus Virtual de la asignatura una tarea de subida de ficheros cuya fecha límite será el **domingo 2 de noviembre a las 23:59.** Al principio de todos los ficheros debe aparecer un comentario con el nombre de quienes la han desarrollado.

2. Evaluación **presencial**, en laboratorio, ante el profesor. Se realizará en el lugar, día y hora correspondiente al horario de prácticas del subgrupo al que pertenezca durante la semana del 3 al 6 de noviembre.

Es en la defensa de la práctica donde se produce la evaluación, la presentación electrónica es simplemente un requisito previo para garantizar la equidad entre subgrupos y la comprobación preliminar de autoría.

Nota: El no poder realizar la modificación y análisis requerido puede dar lugar (según las circunstancias) a que se considere la práctica como no presentada por no haber podido acreditar su autoría.

Anexo: Información Adicional

1. En el Campus Virtual se ha depositado un programa (`GeneradorCasosPrueba.java`) para que podáis generar ficheros de cualquier tamaño. Los ejemplos del enunciado se han creado con el fichero de tamaño 10.000 de este programa, y podéis usarlo para comprobar que vuestro programa funcione correctamente.

2. Cuando estéis seguros de que vuestro programa es correcto entonces debéis crear una versión adaptada a la tarea de **obtener medidas** para distintos tamaños de la entrada: Se sugiere que **incorporéis** la función **`generaCaso`** del generador de casos de prueba en vuestro programa para que podáis omitir el proceso de salvar y leer los ficheros de entrada (recordad que no vamos a usar el tiempo de lectura del fichero en el análisis)[1].

3. El porcentaje de tamaño del mayor grumo no tiene casi influencia en el tiempo del algoritmo y podéis suponer que siempre vale 90%.

4. Por la forma en que se generan los casos de prueba el número de grumos es aproximadamente $\sqrt{n}$.

5. No es obligatorio pero si muy aconsejable el utilizar orientación a objetos en vuestro programa. En concreto se aconseja que uséis la clase **`Conexión`** del generador de casos de prueba para representar las conexiones de amistad y que definái s una clase que represente la red social, y que implemente esta interfaz:

```
public interface IRedSocial {
    public int numUsuarios();
    public int numConexiones();
    public int numGrumos();
    public void leeFichero(String nomfich) throws IOException;
    public void setRed(List<Conexion> red);
    public void creaUsuarios();
    public void creaGrumos();
    public void ordenaSelecciona(double pmin);
    public void salvaNuevasRel(String nomfich) throws IOException;
    public void informe();
}
```

   Los métodos **`numUsuarios`**, **`numConexiones`** y **`numGrumos`** son *getters* de los valores n, m y del número de grumos. El método **`leeFichero`** implementa la etapa 1 (y se sustituye por **`setRed`** cuando obtengáis la lista de conexiones por la función `generaCaso`). Los métodos **`creaUsuarios`**, **`creaGrumos`**, **`ordenaSelecciona`** y **`salvaNuevasRel`** implementan las etapas 2, 3, 4 y 5. Por último el método **`informe`** escribe por pantalla la información que se muestra en los ejemplos de ejecución del apartado 2.

6. Al medir el tiempo es esperable que las medidas tengan más variabilidad que si hubiéramos medido la operaciones (de hecho es posible que alguna de las etapas tenga **mucha** variabilidad). No presupongáis que el término de mayor crecimiento tiene un exponente entero ($n, n^2, n^3, n^4, \ldots$), dadlo con el exponente que mejor se ajuste a vuestros datos y si detectáis mucha variabilidad podéis indicar adicionalmente un rango (por ejemplo indicar como orden $3.45 \cdot 10^{-7}\, n^{3.4} + O(n^3)$ y decir que el exponente puede estar en el rango $[3.2 - 3.7]$)

[1] No debéis inicializar el generador de números aleatorios a un valor concreto, ya que ahora os interesa obtener listas de conexiones diferentes para un mismo tamaño. El motivo de inicializar a un valor concreto en `GeneradorCasosPrueba.java` es el de poder comprobar la corrección de vuestro programa (al obtenerse siempre los mismos datos para un determinado tamaño de la entrada).