Estructuras de Datos y Algoritmos

Práctica I - Curso 2021/22

La Red Social

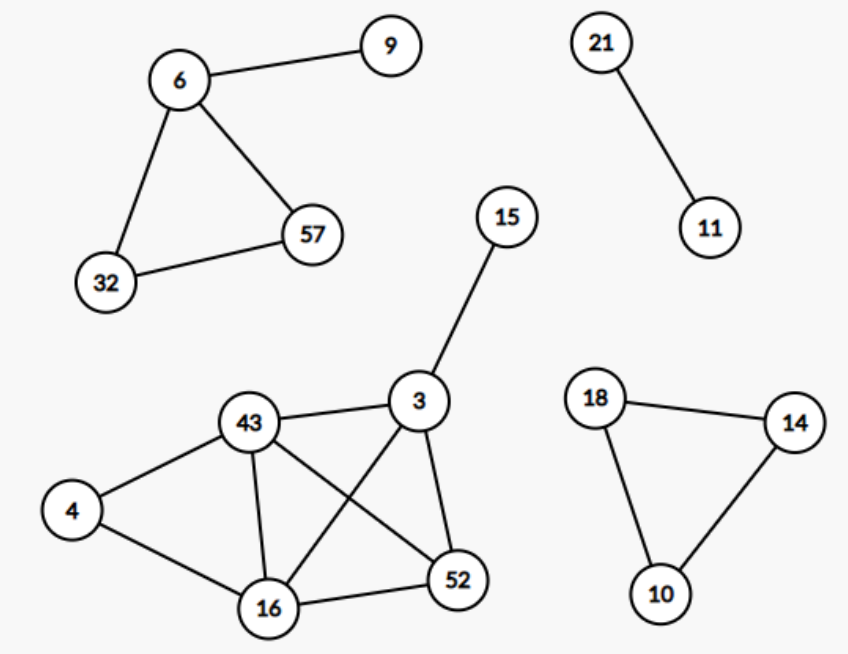
1. Introducción

Enhorabuena!! El CEO de la nueva red social CARALIBRO™ 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 usarios 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 CARALIBRO™ 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 CARALIBRO™ 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):

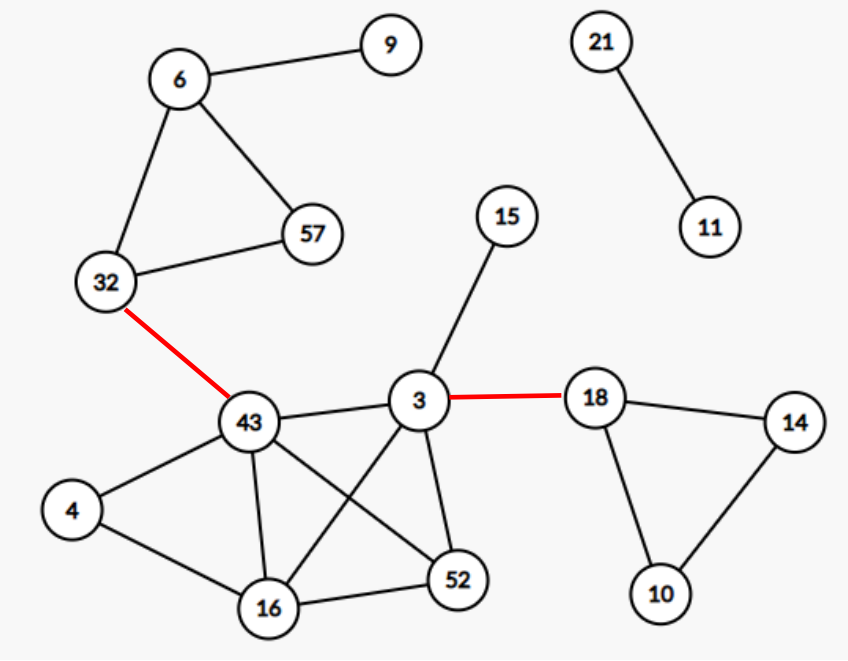


Si un meme es gracioso la mayoría de los usuarios que lo reciban lo reenviarán y por lo tanto es casi seguro que llegará a todos los usuarios del grumo donde se originó. Sin embargo, **los usuarios que no pertenezcan al grumo no lo recibirán**.

Por ese motivo es muy importante para la supervivencia de CARALIBRO™ 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 usarios 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 donde la primera línea contiene un entero que indica el número de usuarios (***n***), la segunda línea un entero que indica el número de conexiones de amistad (***m***) y despues le siguen *m* líneas cada una de ellas conteniendo dos enteros separados por un espacio en blanco. 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).

```
15
17
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

Estructuras de Datos

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

- Acceso basado en índice (**`get`** en Java, **`[]`** en Python)
- Inserción al final (**`add`** o **`addAll`** en Java, **`append`** en Python)
- Búsqueda secuencial (**`contains`** en Java, operador **`in`** o **`not in`** en Python)
- Ordenación (**`sort`** tanto en Java como en Python)

Para representar las conexiones se sugiere crear una clase sencilla con dos atributos enteros en Java y usar tuplas de dos enteros en Python (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.

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 pertene a **`grumo`**, se le añade y la función se llama **recursivamente** a si 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 usarios de cada conexión no están 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 iran 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 escoje unir el primer usuario de un grumo con el segundo 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):

```
ANÁLISIS DE CARALIBRO
---------------------
Fichero principal: test10000.txt
Lectura fichero: 0,02611 seg.
Fichero de nuevas conexiones (pulse enter si no existe):
10000 usuarios, 9951 conexiones
Porcentaje tamaño mayor grumo: 90
Creación lista usuarios: 0,14603 seg.
Creación lista grumos: 0,71146 seg.
Ordenación y selección de grumos: 0,00101 seg.
Existen 49 grumos.
Se deben unir los 5 mayores
#1: 2090 usuarios (20,90%)
#2: 2090 usuarios (20,90%)
#3: 2090 usuarios (20,90%)
#4: 2088 usuarios (20,88%)
#5: 1552 usuarios (15,52%)
Nuevas relaciones de amistad (salvadas en extra.txt)
29917185 <-> 85819403
32800774 <-> 84738093
95715370 <-> 60358727
30179397 <-> 39682073
```

```
29917185 85819403
32800774 84738093
95715370 60358727
30179397 39682073
```

Fichero extra.txt

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`**:

```
ANÁLISIS DE CARALIBRO
---------------------
Fichero principal: test10000.txt
Lectura fichero: 0,02718 seg.
Fichero de nuevas conexiones (pulse enter si no existe): extra.txt
10000 usuarios, 9955 conexiones
Porcentaje tamaño mayor grumo: 90
Creación lista usuarios: 0,14162 seg.
Creación lista grumos: 0,86799 seg.
Ordenación y selección de grumos: 0,00081 seg.
Existen 45 grumos.
El mayor grumo contiene 9910 usuarios (99,10%)
No son necesarias nuevas relaciones de amistad
```

3. Objetivos

Además de crear la aplicación, el otro objetivo 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 Aula 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[1].

Solamente nos interesan las etapas 2, 3 y 4. 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 (m) o bien el número de usuarios (n), ya que ambos valores son 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 y 4 para distintos tamaños (m o n) de las redes.
- Cargar las medidas en una hoja de cálculo o programa similar para poder representar gráficamente las medidas y obtener una formula que estime su tipo de crecimiento.
- Llevar a cabo un **análisis teórico** de la complejidad de las etapas 2, 3 y 4. Para 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 estimar el tiempo que tardará la aplicación para un tamaño dado (como referencia se va a suponer que CARALIBRO™ tiene actualmente 200.000.000 de usuarios y que su objetivo es llegar a 1.000.000.000 de usuarios el próximo año)

No se os va a pedir un documento con el análisis anterior, pero en la entrega electrónica de la práctica se va a tener que rellenar un cuestionario donde habrá preguntas referentes al análisis y es posible que el profesor os solicite en la defensa que le mostreis la hoja de cálculo con los datos y que justifiqueis algún resultado.

4. Presentación y Evaluación de la práctica

Las prácticas se deben realizar de forma **individual**. Se pueden codificar en Java o Python.

Para una correcta evaluación de la práctica el alumno deberá:

1. Presentar electrónicamente (por el Aula Virtual de la Escuela o por correo electrónico), antes de la fecha límite un fichero comprimido (zip) que contenga el código fuente de la aplicación descrita en el enunciado. En el código fuente debe aparecer (como comentario) en las primeras líneas el nombre del alumno que ha realizado la práctica.
2. Presentarse a la sesión de evaluación que le corresponda según su grupo de laboratorio en la semana establecida. En esta sesión se probará la aplicación, se pedirá una modificación sencilla, y se presentarán los resultados de su análisis de eficiencia.

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.

[1] En realidad las medidas se deberían obtener como número de operaciones, no como tiempo en segundos, pero se ha elegido el medir directamente el tiempo para simplificar el desarrollo de la práctica. Como aunque se usen los mismos datos el tiempo puede variar de una ejecución a otra por lo que se recomienda realizar **varias medidas** para el mismo fichero y calcular el **tiempo promedio** de ellas.