

ANÁLISIS LISTA GRUMOS

Una vez que se ha procesado el/los documento/s y, por tanto, se han procesado también la lista de usuarios y la lista de conexiones, es momento de determinar cuáles son los grupos de la red social.

Para crear los grupos serán necesario el objeto Análisis, un ArrayList en el que se guardarán los usuarios procesados y un ArrayList que almacenará los usuarios que ya han sido procesados.

Cuando se ha finalizado la ejecución de la función pedir datos, se ejecuta la función crearGrupos(), en la cual se recorre el ArrayList del listado de usuarios, se crea un nuevo ArrayList que será donde se almacenen los usuarios pertenecientes al grupo (en cada iteración se creará un nuevo ArrayList), se llamará a la función uber_amigos() y se almacenará la lista de los usuarios del grupo en el ArrayList de grupos.

A continuación, una captura de la función crearGrupos():

```
private static void crearGrupos() {
    ArrayList usuariosGrumo; // Listado de los usuarios que pertenecen a un grupo

    elAnalisis.tIListaGrumos = hora();

    for (Object usuario : elAnalisis.listadoUsuarios) {
        usuariosGrumo = new ArrayList();
        uber_amigos((int) usuario, elAnalisis.listadoConexiones, usuariosGrumo);
        elAnalisis.grupos.add(usuariosGrumo);
    }
    quitarVacios(elAnalisis.grupos);

    elAnalisis.tFListaGrumos = hora();
}
```

La función uber_amigos() recibe los siguientes parámetros:

- El usuario inicial, de tipo int, que será el usuario del que se quieran saber sus conexiones.
- El listado de conexiones existentes en el archivo (procesado anteriormente).
- El listado de los usuarios que pertenecen a un grupo.

Lo primero que hace la función uber_amigos() es comprobar si el usuario ya ha sido procesado con anterioridad o no. Si no ha sido procesado, continúa. Si sí que ha sido procesado, termina la ejecución de la función, volviendo a la función crearGrupos() y seleccionando el siguiente usuario.

Después de comprobar si se ha procesado o no, si no ha sido procesado entonces añade el usuario al ArrayList de usuarios procesados. Luego, recorre la lista de conexiones en busca de ese usuario (ya sea siendo el primer usuario de la conexión o el segundo). En cualquiera de los dos casos, agregará tanto al primer usuario como al otro usuario de la conexión al grupo (si no está ya en la

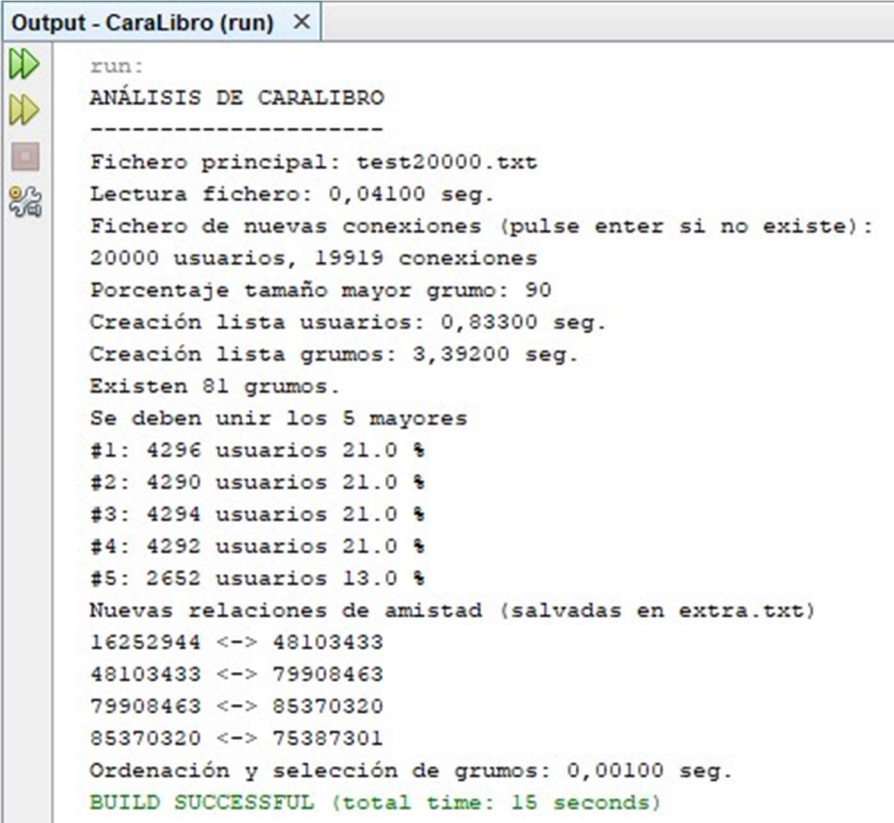
lista, realizando una comprobación para cada uno) y llamará de forma recursiva a la función `uber_amigos()`, pero esta vez el usuario principal será el que no estaba en el grupo.

Cuando finaliza toda la ejecución de la función `crearGrupos()`, es decir, se ha recorrido toda la lista de usuarios de la red, se ejecuta la función `quitarVacios()`, que recibe por parámetro un `ArrayList` y elimina las posiciones del mismo que estén vacías. En este caso, el `ArrayList` que se pasa por parámetro es el de los grupos.

Funciona con un bucle `do...while()`, donde dentro recorre toda la lista en busca de que haya una sublista vacía. Si la encuentra, elimina dicha posición y continúa.

A continuación, la función `uber_amigos()` y la salida por pantalla tras el proceso (utilizando el archivo de pruebas `test20000.txt`):

```
private static void uber_amigos(int usuarioInicial, ArrayList<Conexion> conexiones, ArrayList grupo) {
    if (!usuariosProcesados.contains(usuarioInicial)) { // Si ese usuario no se ha procesado todavía
        usuariosProcesados.add(usuarioInicial); // Agrego el usuario a usuariosProcesados
        for (Conexion laConexion : conexiones) { // Recorro la lista de conexiones
            if (laConexion.usuario1 == usuarioInicial) { // El usuario1 de la conexión es el usuarioInicial que estábamos procesando
                if (!grupo.contains(laConexion.usuario2)) { // Si el grupo todavía no contiene al otro usuario (usuario2 de la conexión), lo agrega
                    grupo.add(laConexion.usuario2);
                    uber_amigos(laConexion.usuario2, conexiones, grupo); // Se vuelve a llamar de forma recursiva
                }
            } else if (laConexion.usuario2 == usuarioInicial) { // El usuario2 de la conexión es el usuarioInicial que estábamos procesando
                if (!grupo.contains(laConexion.usuario1)) { // Si el grupo todavía no contiene al otro usuario (usuario1 de la conexión), lo agrega
                    grupo.add(laConexion.usuario1);
                    uber_amigos(laConexion.usuario1, conexiones, grupo); // Se vuelve a llamar de forma recursiva
                }
            }
        }
    }
}
```



```
Output - CaraLibro (run) X
run:
ANÁLISIS DE CARALIBRO
-----
Fichero principal: test20000.txt
Lectura fichero: 0,04100 seg.
Fichero de nuevas conexiones (pulse enter si no existe):
20000 usuarios, 19919 conexiones
Porcentaje tamaño mayor grupo: 90
Creación lista usuarios: 0,83300 seg.
Creación lista grupos: 3,39200 seg.
Existen 81 grupos.
Se deben unir los 5 mayores
#1: 4296 usuarios 21.0 %
#2: 4290 usuarios 21.0 %
#3: 4294 usuarios 21.0 %
#4: 4292 usuarios 21.0 %
#5: 2652 usuarios 13.0 %
Nuevas relaciones de amistad (salvadas en extra.txt)
16252944 <-> 48103433
48103433 <-> 79908463
79908463 <-> 85370320
85370320 <-> 75387301
Ordenación y selección de grupos: 0,00100 seg.
BUILD SUCCESSFUL (total time: 15 seconds)
```

Como se puede observar, el programa procesa correctamente los grupos (sabe cuántos grupos hay, en este caso 81).

En el siguiente documento se analizará cómo se ordenan los grupos, cómo se agregan nuevas conexiones extra y resultados.