# Estructuras de Datos y Algoritmos

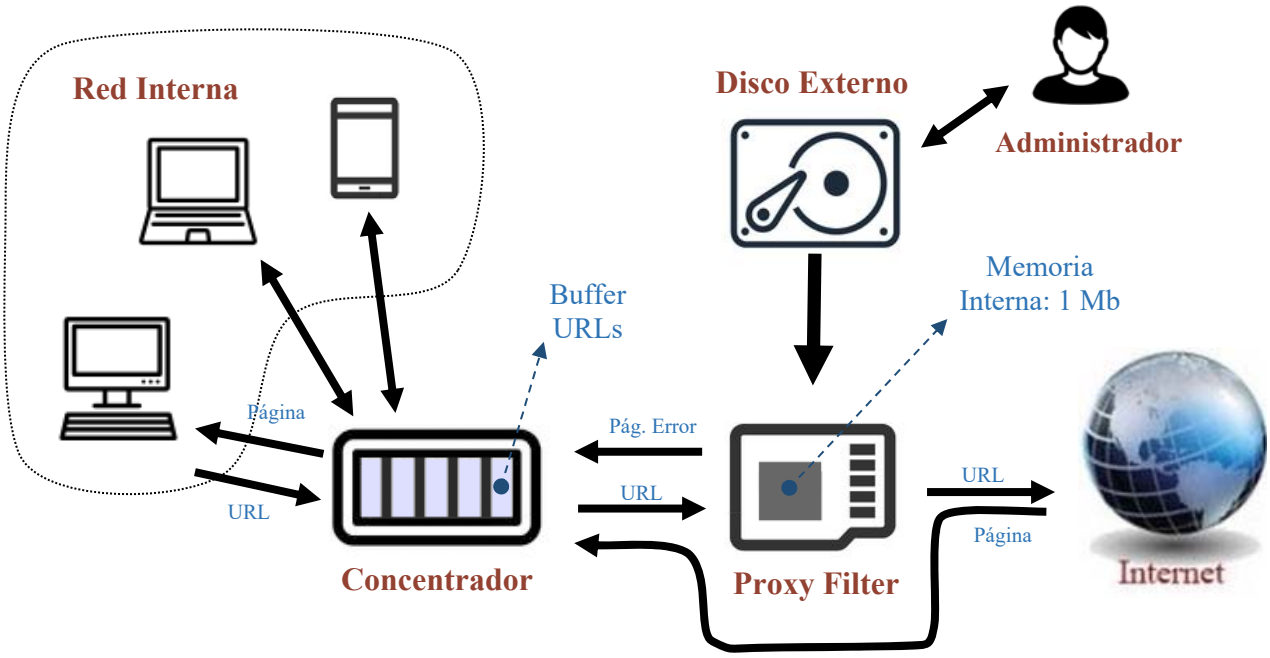
# Práctica I - Curso 2018/19

# Proxy Filter

## 1. Introducción

Una gran corporación, con miles de empleados, tiene organizada su red interna de forma que todo intento de acceso a cualquier página web realizada por alguno de sus ordenadores o dispositivos móviles pase primero por un **filtro** que consiste en comprobar si la dirección de la página está o no en una lista de **direcciones prohibidas** (porque se sabe que instalan virus, etc.). Si la dirección **está** en la lista entonces se devuelve un código de error HTTP "Acceso Denegado", y si **no está** se permite que la petición pase a Internet y se devuelva su resultado.

Este proceso de comprobación se realiza automáticamente por parte de un dispositivo denominado **Proxy Filter** de acuerdo al siguiente esquema:




Las direcciones de las páginas web (de ahora en adelante las denominaremos **URL**) a las que quieren acceder todos los equipos de la corporación llegan al concentrador, que las va almacenando en un buffer mientras el proxy está ocupado. El trabajo del proxy es, repetidamente, pedir una URL al concentrador y comprobar si está en el fichero de URL prohibidas: Si no lo está envía la URL a Internet (que devuelve la página directamente al concentrador, el cual la remite al equipo que la solicitó), pero si es una URL prohibida no envía la URL a Internet sino que envía al concentrador una página que muestra un aviso para que se la remita al equipo.

El proxy es una tarjeta con procesador **programable** y memoria interna. Como ésta memoria es muy limitada tiene una conexión a un disco duro externo, del que puede leer ficheros completos (modo "ráfaga") pero no escribir datos. El contenido del disco duro se actualiza periódicamente por el administrador añadiendo nuevas direcciones prohibidas.

## 2. Descripción del Problema

En el momento actual la red informática de la corporación está en crisis:

- El número de URL prohibidas es de unos 2.000.000, como el tamaño medio de las URLs es de unos 50 caracteres (codificación ASCII, es decir 1 byte por carácter) esto hace que el tamaño de un fichero u estructura de datos que contenga la lista de URLs prohibidas tenga que ser mayor que 96 Megabytes.
- El tamaño de la memoria interna del Proxy es de **1 Megabyte**. Esto implica que los datos sobre URLs prohibidas deben almacenarse en el disco duro externo. Pero el Proxy no puede leer datos del disco duro externo poco a poco, su única funcionalidad es **leer un fichero de golpe y cargar todo su contenido** en la memoria interna. La forma que se les ha ocurrido para resolver ese problema es el **dividir** el fichero original en unos 100 ficheros de menos de 1 Megabyte cada uno. Para comprobar si una URL es prohibida o no el Proxy va leyendo secuencialmente cada uno de los 100 ficheros (cada lectura sobrescribe los datos de la anterior) y busca la URL entre las que contiene el fichero.
- La lectura de un fichero completo en el disco duro externo tarda unos 20 milisegundos, independientemente del tamaño del fichero (latencia).
- Analizando la frecuencia con que se solicitan URL prohibidas se ha visto que es un suceso relativamente raro (1 de cada 1000 peticiones), lo que significa que la mayoría de las URLs requerirán que el Proxy tenga que leer secuencialmente unos 100 ficheros, lo que tarda más de 2 segundos.
- El número medio de URLs que llegan al Concentrador es de 25 por segundo.

Se puede apreciar que existe un problema grave ya que es necesario que el Proxy procese las URLs al menos a la misma velocidad con que llegan al Concentrador, en caso contrario su buffer se llenará y empezará a rechazar peticiones (colapso de la red). Actualmente el Proxy las procesa a 0.5 URLs/seg. y debería procesarlas 50 veces más rápido (25 URLs/seg.)

Cosas que **no sirven para resolver el problema**:

- Cambiar la configuración o elementos del sistema: No está permitido.
- Omitir el chequeo de URL prohibidas: Imposible, se tendría un acceso a una web maliciosa cada 40 segundos, algo imposible de afrontar por el sistema de seguridad.
- Comprimir (sin pérdidas) el fichero de URLs: Se ha comprobado que su contenido es casi aleatorio y ninguna técnica estándar reduciría su contenido apreciablemente.
- Comprimir (con pérdidas) el fichero de URLs: Se necesitaría comprimir cada URL a un valor de 16 bits, lo que daría lugar a falsos positivos (URLs correctas catalogadas como prohibidas) con una frecuencia apreciable (mayor que 1 de cada 1.000). **No se permite ningún método que pueda producir falsos positivos**[1].

La única posibilidad para resolver el problema es **reprogramar** el Proxy usando nuevos **algoritmos y/o estructuras de datos** que permitan reducir el número de accesos al disco duro externo por cada URL procesada.

[1] Por supuesto tampoco se permite ningún método que produzca falsos negativos (que una URL prohibida pase el filtro).

## 3. Descripción de la Práctica

Conceptos y terminología utilizada:

- ***N***: Es el número de URL prohibidas (2 millones en el ejemplo del apartado anterior y en el caso de prueba, puede ser distinto a éste valor en la prueba del programa).
- ***Mb:*** Megabyte, 1024·1024 = 1.048.576 bytes.
- **URL**: Son direcciones web representadas como cadenas de texto ASCII (1 byte por cada carácter). Su longitud varía entre 18 y 200 caracteres, la longitud media es aproximadamente 50 caracteres.
- **El fichero**: Se refiere al fichero que contiene la lista de las ***N*** URL prohibidas. Es un fichero de texto con una URL por línea. Se usa el estilo Unix: Un único carácter de fin de línea, LF (valor 10). Las URLs no están ordenadas.
- **Subfichero**: Se denomina así a los ficheros de tamaño menor que 1 Mb en que se divide el fichero para que el Proxy pueda trabajar con ellos (recordad que el Proxy sólo puede leer un fichero cargándolo completamente en memoria). Lo estándar es nombrar estos ficheros como 000.txt, 001.txt, 002.txt, etc. Si se necesita otra forma de nombrarlos debe consultar primero al profesor.

La práctica consiste en crear 3 aplicaciones:

- **GeneraFicheros**: El objetivo de ésta aplicación es dividir *el fichero* en *subficheros*. Es necesario pensar bien la forma en que se reparten las URLs entre los *subficheros*, ya que puede ser clave para conseguir mejorar la eficiencia del Proxy. También es posible almacenar no sólo URLs sino otra información que ayude al Proxy, o bien tener *subficheros* que almacenen URLs y otros que almacenen otra cosa.. De todas formas siempre el tamaño de cada *subfichero* debe ser menor que 1 Mb para que pueda leerlos el Proxy.

  Esta aplicación pide al usuario que se introduzca el nombre del *fichero* y a continuación genera los *subficheros* (en el mismo directorio del *fichero*).

- **ActualizaFicheros**: El objetivo de ésta aplicación es incluir una nueva URL prohibida en los *subficheros*. Pedirá al usuario que introduzca la nueva URL y modificará los *subficheros* de la forma adecuada. Se puede suponer que se ejecutará en el mismo directorio donde se encuentran los *subficheros*.

- **ProxySim**: Esta es la aplicación que simula al Proxy. La aplicación pedirá al usuario el nombre de un fichero de texto que contiene una secuencia de URLs (una por línea) para su verificación. La aplicación ejecutará un bucle en el que aplicará el algoritmo diseñado por vosotros para comprobar si cada URL pertenece al conjunto de URLs prohibidas o no, usando la información contenida en los *subficheros*. Si es una URL prohibida se escribirá por pantalla, en caso contrario no se hará nada. Al terminar de procesar todas las URLs escribirá en pantalla información estadística sobre las operaciones realizadas (ver final del apartado 4). **Nota importante**: Cada URL se debe procesar de forma **independiente**, no se puede usar el "truco" de leer un *subfichero*, comprobar para todas las URLs si alguna está en él, leer el siguiente *subfichero*, volver a comprobar todas las URLs, etc. **El proxy real lee una sola URL del controlador, no puede leer varias** (aunque eso sería ventajoso para su eficiencia, no es posible por motivos de diseño).

## 4. Detalles sobre la implementación de ProxySim

La lectura de un *subfichero* se debe efectuar **exactamente** con el siguiente código Java[2]:

```
static byte[] subfich; // Datos del subfichero (var. global)

static void leeSubfichero(String nomfich) throws IOException {
  File fich = new File(nomfich);
  int tam = (int) fich.length(); // Tamaño en bytes
  subfich = null;
  subfich = new byte[tam];
  try ( FileInputStream fis = new FileInputStream(fich)) {
    fis.read(subfich); // Lectura de todo el fichero
  }
}
```

Se puede apreciar que el contenido del *subfichero* se transfiere a un **array de bytes**. Si el *subfichero* contiene URLs y se quiere poder acceder fácilmente a la *i*-ésima URL almacenada (en formato String) se puede usar el código siguiente:

```
static int[] indURL; // Indices de las URLs en "subfich"

static void creaIndices() {
  // 1. Contar el numero de URLs
  int n = 0;
  for(int i = 0; i < subfich.length; i++) {
      if(subfich[i] == 10) { n++; }
  }
  // 2. Almacenar posición de separadores
  indURL = null;
  indURL = new int[n];
  int k = 0;
  for(int i = 0; i < subfich.length; i++) {
      if(subfich[i] == 10) { indURL[k++] = i; }
  }
}

static String accesoURL(int i) {
    int a = i == 0 ? 0 : indURL[i-1] + 1;
    int b = indURL[i] - 1;
    return new String(subfich, a, b-a+1, StandardCharsets.US_ASCII);
}
```

La función **creaIndices** debe ser llamada tras la ejecución de **leeSubfichero**. A partir de ese momento es posible obtener la URL *i*-ésima usando la función **accesoURL**.

[2] Los códigos mostrados usan variables globales, que habitualmente se consideran una mala práctica de programación. En este caso su uso se justifica por las limitaciones de memoria impuestas: Si en lugar de usar vars. globales las funciones devolvieran arrays, existirían puntos de ejecución con 2 copias "vivas" de arrays de gran tamaño (aunque en la siguiente instrucción un array reemplazaría al otro) que provocarían la activación del filtro (profiler) de uso de memoria.

Si el *subfichero* no contiene URLs sino otro tipo de datos, es posible acceder a ellos mediante la clase **DataInputStream**:

```
...
ByteArrayInputStream bais = new ByteArrayInputStream(subfich);
DataInputStream dis = new DataInputStream(bais);
...
```

A partir de aquí es posible usar la variable **dis** para leer datos de tipo simple (dis.readInt(), dis.readDouble(), .. etc.)

Para generar un fichero compatible con lo anterior se debe usar **DataOutputStream** y sus métodos writeInt(..), writeDouble(..), etc.

```
...
FileOutputStream fos = new FileOutputStream(nomfich);
DataOutputStream dos = new DataOutputStream(fos);
...
```

## Comprobación del límite de memoria

Las dos primeras aplicaciones pueden usar libremente cualquier cantidad de memoria, pero la aplicación **ProxySim** tiene limitada la cantidad de memoria que puede usar a un máximo de 1 Mb.

Por ejemplo, si vuestra aplicación usa el método de acceso a URLs indicado anteriormente y el array **indURL** ocupa 0.2 Mb, entonces vuestros *subficheros* no pueden ser mayores que 0.8 Mb.

Para poder controlar la cantidad de memoria que utiliza la aplicación se puede usar la siguiente función:

```
public static long memIni;
public static Runtime rt;

public static long usoMemoria() {
  rt.gc();
  return rt.totalMemory() - rt.freeMemory() - memIni;
}
```

Donde las variables globales **memIni** y **rt** deben inicializarse al comienzo de la ejecución del programa de la siguiente forma:

```
...
rt = Runtime.getRuntime();
rt.gc();
memIni = rt.totalMemory() - rt.freeMemory();
...
```

**Nota importante**: Al evaluar vuestras prácticas se utilizará una máquina virtual con el profiler activado para que detecte un uso de memoria superior a 1 Mb, por lo que es muy importante que previamente comprobéis mediante la función **usoMemoria** que no lo sobrepasáis (se consideraran no válidas las prácticas que no cumplan esa condición).

## Salida de ProxySim

Ademas de ir escribiendo en pantalla las URLs prohibidas que ha ido detectando, cuando termine el proceso la aplicación **ProxySim** debe escribir en pantalla la siguiente información:

- Número total de lecturas de subficheros realizadas (es decir, el número de llamadas a la función **leeSubfichero.**
- El ratio de lecturas de subficheros por URL (el valor anterior dividido por el número de URL comprobadas)
- El número de operaciones de acceso a array realizadas. Se debe definir un contador (de tipo **long**, ya que éste valor puede ser muy alto) que se incremente cada vez que se realice un acceso a elemento de cualquier array, *salvo aquellas relacionadas con la lectura de subficheros*.
- El resultado de una última llamada a **usoMemoria**

# 5. Restricciones para la primera práctica

En ésta primera práctica no deben utilizarse estructuras de datos o algoritmos **avanzados** (entendiendo como tales aquellos no contemplados en la asignatura), en principio las técnicas que debéis explorar son la ordenación de los datos, otros algoritmos sencillos o el uso de estructuras inspiradas en los temas 4 (árboles) o 5 (tablas de dispersión). Es completamente normal el **no conseguir resolver el problema planteado** en el apartado 2 (aumentar 50 veces la velocidad). De hecho es muy difícil conseguirlo con estos requisitos (pero no imposible, hay una idea muy sencilla que permite *casi* conseguirlo). En la segunda práctica se pedirá la implementación de una estructura de datos (no contemplada en la asignatura) con la que si será posible resolver completamente el problema.

# 6. Presentación y Evaluación de la práctica

La práctica está pensada para ser codificada en Java. Aunque también se pueden utilizar los lenguajes Python, C, Haskell o R para la implementación del programa, existen problemas específicos en el uso de esos otros lenguajes (en particular Python y R), por lo que en ese caso se aconseja consultar primero al profesor.

Se recomienda que la práctica se realice en parejas de dos personas, aunque en casos especiales se puede permitir la realización individual o en grupos de 3 personas (en este último caso la nota obtenida sufre una minoración)

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 las 23:59 del domingo 14 de octubre de 2018, un fichero comprimido que contenga el código fuente de las tres aplicaciones mencionadas en el enunciado. En el código fuente debe aparecer (como comentario) en las primeras líneas el nombre de quienes han realizado la práctica.
2. Presentarse a la sesión de evaluación que le corresponda según su grupo de laboratorio en la semana del 15 al 21 de octubre de 2018. En esta sesión se probarán las aplicaciones realizadas (con nuevos datos). Es posible que en esa sesión se pida la modificación del código de la práctica y la obtención de nuevos resultados.

En el caso de realización por parejas (la situación habitual), tan sólo es necesario que uno cualquiera de ellos realice la presentación electrónica. En la evaluación, sin embargo, si es necesaria la presencia de ambos y la evaluación puede ser distinta para cada uno de ellos.