Recuperación de la Información

Curso 2020/2021

Práctica 3: Crawler

Carmen del Mar Ruiz de Celis

Índice

1. Añadir Tika al proyecto

Lo primero que hay que hacer es descargar la biblioteca de Tika y añadirla al path del editor. Una vez hecho esto, la importo en la clase en la que voy a usarla.

2. Regex

Para el desarrollo del programa hago uso de varios Regex. Para ello, estudié la estructura de las URL de Wikipedia consultando artículos y el código HTML de sus páginas web.
Principalmente, las URL de Wikipedia que nos interesan (es decir, de artículos) se forman así:

- Utiliza el protocolo HTTPS.
- No utiliza el comúnmente conocido www como subdominio, sino el código del país, el cual determinará también el idioma en el que se mostrará la página. Por lo tanto, en nuestro caso se tratará de es.
- wikipedia como dominio, obviamente.
- .org como extensión.
- /wiki/ como recurso.
- A partir de aquí, corresponde con el nombre del artículo que estemos leyendo.

De aquí nacen los siguientes Regex:

2.1. Semilla

Éste se encargará de comprobar que la URL que hemos elegido para empezar a descargar se ha introducido correctamente, o en otras

palabras, que cumple con la estructura que se ha explicado anteriormente.

```
https:\/\/es\.wikipedia\.org\/wiki\/([A-Za-z0-9\_\-\.\(\)ñáéíóúüÑ
ÁÉÍÓÚÜ]+)
```

2.2. Limpiar los documentos

Algunos enlaces a otras secciones de wikipedia, cumplen con la estructura anterior, pero contienen en la parte variable una estructura del tipo Área:Artículo, como por ejemplo Ayuda:Contenidos o Usuario:Nombre. Luego, usaré este Regex para eliminarlos y extraer así lo que verdaderamente nos interesa.

```
href=\"\\/wiki\\/.*:.*\"
```

2.3. Corregir errores de codificación

Al descargar el código HTML, las vocales con acento, diéresis y la ñ se encuentran en hexadecimal, por lo que debemos usar un vector de Regex para hacer un replaceAll.

```
static String[] reemplazaroriginal = { "á", "é", "í", "ó", "ú",
"Á", "É", "Í", "Ó", "Ú", "ñ", "Ñ", "ü", "Ü" };

static String[] reemplazar = { "%C3%A1", "%C3%A9", "%C3%AD",
"%C3%B3", "%C3%BA", "%C3%81", "%C3%89", "%C3%8D","%C3%93",
"%C3%9A", "%C3%B1", "%C3%91", "%C3%BC", "%C3%9C" };
```

2.4. Extraer los enlaces a otros artículos

Por último, ya podemos extraer los enlaces a los documentos que nos interesan. En enlace está de forma relativa dentro de una etiqueta html, así que tendremos que usar una variante más corta del primer Regex.

```
href=\"(\\/wiki\\/([A-Za-z0-9\\_\\-\\.\\(\\)ñáéíóúüÑÁÉÍÓÚÜ]+))
```

3. Estructura

Tal y como hemos visto en clase, el Crawler se puede resumir en 3 partes o ciclos:

Aunque en mi caso y de manera más detallada se podría ampliar a:

Estos pasos se encapsulan en la función `núcleo()`, la cual es la que se encontrará dentro del bucle que iterará `i` veces.

3.1. Descarga

En primer lugar, se pide al usuario la url fija por la que se va a empezar a descargar. Una vez comprobado que es un enlace válido y que el artículo existe a través de la función `HttpResponse(URL url)`, se descarga mediante la función `HTMLCode(URL url)` para obtener el código

HTML y `Tika().parseToString(url)` para obtener el texto plano. El código HTML nos interesa para extraer los enlaces que pondremos a planificar y el texto plano será lo que almacenemos.

3.2. Almacenamiento

Para almacenar los documentos, extraigo el título del artículo haciendo uso de los grupos que incorporé en los Regex y crearé un documento con él, comprobando que dicho fichero no existe ya. Realmente, esta comprobación podría obviarse, ya que más adelante se comprueba que no se añadan enlaces visitados a la cola, o en otras palabras, que no se descarguen documentos ya descargados. Ambas comprobaciones hacen lo mismo, pero he dejado ambas porque la seguridad nunca está de más.
En este fichero guardaré el texto plano. Para ello hago uso de la función `escribirFichero(File file, String content)`.

Nota: el fichero requiere una ruta absoluta, por lo que si quieres probar el programa debes cambiarla, de lo contrario dará un error.

3.3. Filtros

Una vez tenemos el código html, hay que corregir los caracteres hexadecimales y eliminar los enlaces que no necesitamos que podrían colarse en nuestra lista de intereses con los Regex detallados anteriormente.

3.4. Planificación

Tras limpiar el código, podemos extraer los enlaces y añadirlos a la cola de descarga. Se comprueba que los enlaces no hayan sido

visitados ya y que no estén ya en la cola, para evitar duplicaciones. Tal y como he dicho anteriormente, podría dejarse esta condición o la comprobación de que no se descargue si ya existe un fichero con el mismo nombre, ya que su función es la misma; pero yo he dejado ambas.

Para la estructura de la cola he elegido un ArrayList, ya que:

- Puedo hacer uso de la función contains(Obj obj) para comprobar si un elemento ya está encolado para no repetir elementos. Esta condición podría obviarse, ya que con las condiciones de no descargar documentos si existen localmente no habría problemas. Además, esta función es algo costosa, teniendo en cuenta que, en este caso, la estructura es dinámica y las entradas crecen rápidamente en cada iteración; pero, de nuevo, he decidido dejarla como en los casos anteriores.
- Puedo acceder y eliminar cualquier elemento en coste O(1).

Por otra parte, para los enlaces visitados he utilizado un HashSet, ya que así puedo encontrar cualquier elemento en coste O(1) y comprobar rápidamente si un enlace se ha visitado o no.

Tras esto, se eliminaría el link actual de la cola, se añadiría a visitados y se repetiría el proceso con el siguiente enlace hasta que la cola se vacíe o se haya iterado el número de veces indicadas, en este caso 30000.