L01_UO295180
Desarrollo de un crawler

Mario Orviz Viesca - UO295180

1. Introducción

La práctica propuesta para este ejercicio es el desarrollo de un crawler. La tecnología utilizada será `python` con la ayuda de las siguientes librerías:

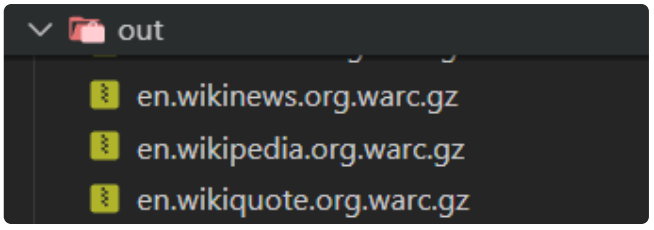
- `requests`: Utilizada para realizar peticiones
- `warcio`: Utilizada para almacenar los documentos obtenidos
- `bs4`: Utilizada para obtener todos los enlaces de un documento html
- `urllib`: Utilizada para trabajar con urls, y también su módulo `RobotFileParser` para leer y comprobar los archivos `robots.txt` de los sitios web

Para este trabajo, he optado por implementar dos algoritmos de búsqueda diferentes, un algoritmo de búsqueda en profundidad y otro de búsqueda en anchura.
La idea principal consiste en, partiendo de un archivo de urls (semillas), hacer una petición a cada una de ellas, obteniendo todas las urls que residen en ellas, y repetir el proceso con estas, de acorde a el algoritmo escogido.

2. Conceptos generales

El proyecto se basa en un único archivo `crawler.py` que implementa la lógica del crawler. Aparte de eso, consta de un archivo `seeds.txt` que contiene las urls "semilla" que sirven como punto de partida para que el crawler comience la búsqueda.

El programa exporta los resultados en formato `.warc.gz` dentro del directorio `/out`. Estos archivos tendrán como nombre el dominio de la página web, guardando en cada archivo todos los resultados obtenidos dentro de ese dominio. Antes de guardar un archivo en formato `warc`, el algoritmo comprueba que esa página no esté indexada ya.



Ejemplo de archivos generados

2.1. Parametrización

El programa se puede configurar con 8 diferentes variables que residen en el archivo `.py` del programa:

```
OUT_DIR = "out"
SEEDS = "seeds.txt"
DELAY = 2
MAX_DOWNLOADS = 50
MAX_DEPTH = 5
USER_AGENT = 'Mozilla/5.0'
CHECK_ROBOTS = False
SAVE_HTML = False
RANDOMIZE_BREATH = False
```

- `OUT_DIR` contiene el directorio de salida donde se almacenan los resultados obtenidos
- `SEEDS` contiene el archivo donde residen las urls que alimentan al crawler
- `DELAY` indica el tiempo (en segundos) que el programa espera entre descargas
- `MAX_DOWNLOADS` indica el número máximo de descargas que realizará el programa
- `MAX_DEPTH` (solo para el algoritmo de búsqueda en profundidad) indica la profundidad máxima a la que llegará el algoritmo
- `USER_AGENT` indica el nombre de agente de usuario que utilizará el programa a la hora de realizar peticiones
- `CHEK_ROBOTS` es un valor booleano que indica si el crawler debe comprobar antes si tiene permitido descargar dicha url comprobando el archivo `robots.txt`
- `SAVE_HTML` es un valor booleano que indica si el programa debe guardar archivos `.html` directamente
- `RANDOMIZE_BREATH` es un valor booleano que indica si se debe randomizar la búsqueda en anchura o no (explicado más adelante)

2.2. Ejecución

El scrip está pensado para ejecutarse desde la línea de comandos. Con la parametrización anterior, se aportan unos valores por defecto para la ejecución del programa, aunque estos valores pueden ser configurados mediante los siguientes argumentos desde la línea de comandos:

- `--mode` indica el tipo de algoritmo que se desea usar. Puede ser `depth` o `breath`
- `--max-downloads` indica el número máximo de descargas que realizará el programa
- `--delay` indica el tiempo que esperará el programa entre cada descarga
- `--max-depth` en el caso de que la búsqueda sea en profundidad, indica la profundidad máxima a la que llegará el algoritmo
- `--check-robots` si se indica, el programa comprobará los archivos `robots.txt` (por defecto no lo hace)

- `--save-html` si se indica, el programa guardará ficheros `.html`. Desactivado por defecto
- `--randomize-breath` si se indica, el programa randomizará los links explorados por la búsqueda en anchura una vez consumidas las semillas. Desactivado por defecto
- `--seeds` indica el path relativo del fichero de semillas

Ejemplo de ejecución:

```
python crawler.py --mode depth --max-downloads 100 --delay 1 --max-
depth 5 --seeds seeds.txt --check-robots --save-html --randomize-breath
```

3. Funcionalidad

En esta sección se describen las funciones principales que utiliza el crawler para realizar su labor.

3.3. Obtención de links

Esta es la función principal del crawler, utilizada por ambos algoritmos para descargar y obtener los links de un sitio web. Su funcionamiento es muy sencillo y parecido al del pseudocódigo proporcionado para la realización del ejercicio.
Esta función recibe como parámetro una url la cual será a la que haga la petición y extraiga sus links. Como valor de retorno devolverá siempre una lista que contenga los links obtenidos y un valor boolean que indique si ha tenido éxito o no (Si no ha tenido éxito, la lista que devolverá estará vacía). Su funcionamiento es el siguiente
En primer lugar, si el valor `CHECK_ROBOTS` es `True`, utilizará la función `obtainRobotsPermission` para leer el archivo `robots.txt` y comprobar si el crawler tiene permiso para analizar esta página. A continuación, usando el módulo `requests` hará una petición `get` a la url indicada y comprobará que el valor del `Content-Type` devuelto es `text/html`. A continuación llamará a la función `save_html` para registrar el sitio web y parará el proceso la cantidad de tiempo estipulada en la variable `delay`.
Una vez hecho esto, utilizando la herramienta `BeautifulSoup`, obtendrá todos los elementos `a` dentro del texto del html, obtendrá el valor `href` de estos elementos y los normalizará usando la función `normalizeLink` para evitar enlaces relativos a la página web.
Una vez normalizados, los links se añaden a una lista de links que será devuelta al finalizar la ejecución.

3.4. Obtención de permisos en `robots.txt`

La función `obtainRobotsPermission` será la encargada de leer los archivos `robots.txt` (si es que existen) e indicar si podemos o no analizar un determinado dominio. Hace uso del módulo `RobotFileParser` y de `urlparse`. La dirección de el archivo `robots.txt` se obtiene con la siguiente línea:

```
robots_url = f"{parsed.scheme}://{parsed.netloc}/robots.txt"
```

Luego, sobre una instancia de `RobotFileParser`, se hace un `rp.set_url(robots_url)`, para a continuación hacer una llamada a `rp.read()`, y para comprobar si podemos acceder a la url dentro de dicho dominio utiliza:

```
rp.can_fetch(USER_AGENT, url)
```

3.5. Normalizar links

Esta función es muy sencilla. Recibe la url original y el link, y comprueba si al parsear el link con `urllib`, encuentra `netloc` o no. Si no lo encuentra, simplemente hace un

`urljoin` de la url original con el link ya que se trata de un enlace relativo.

3.6. Guardar registros

Para esta funcionalidad se utilizan dos funciones diferentes: `save_html` y `already_in_warc`. La segunda se utiliza para evitar guardar duplicados en los archivos `warc`. La primera simplemente lo que hace es crear un archivo `warc` con el `netloc` le la url que va a guardar si no existe, y si existe primero comprueba que no haya guardado con anterioridad esta url y luego la añade al archivo existente

4. Algoritmos de búsqueda

4.7. Búsqueda en profundidad

El algoritmo de búsqueda en profundidad se trata de un algoritmo recursivo que se lanza desde la función `depth_first_crawl` y se realiza en la función recursiva `crawl_depth`.
En la primera, se obtienen las url semilla y se lanza por orden a la función `crawl_depth`, que será la encargada de llevar a cabo toda la ejecución del algoritmo.
Esta segunda función, el primer paso realizar una llamada a `obtain_links` con la url con la que fue invocada, y realiza una llamada recursiva para todas las urls devueltas por ella.

4.8. Búsqueda en anchura

El algoritmo de búsqueda en anchura se implementa en la función `breath_first_crawl`.
Cuenta con una lista llamada `frontier` donde se almacenan las urls que todavía no se han explorado y con una lista explored con las urls que se han explorado.
A continuación, la función entra en un bucle que se repite siempre que frontier contenga elementos y que el número de descargas no supere al el número de descargas máximas que se ha especificado. En cada iteración, el programa extrae un elemento de `frontier` y realiza una llamada a la función `obtain_links`.
Si la función `obtain_links` devuelve `True`, significa que un elemento ha sido descargado, por lo que se aumenta el número de elementos descargados y se añaden los links devueltos por la función al final de `frontier` en orden.
En el caso de que el valor de `RANDOMIZE_BREATH` fuese `True`, una vez consumidas las semillas, se embarajaría la lista `frontier` para randomizar el siguiente link que se escoja (esto produciría una búsqueda aleatoria, por lo que no podríamos prever los resultados que obtendremos de ella)

5. Evaluación

Para realizar la evaluación de este algoritmo, he desarrollado una función que guarda automáticamente el resultado de la ejecución de cada uno de estos algoritmos en un archivo de texto.
El nombre de estos archivos es la fecha en la que se generaron, permitiendo saber en que momento se realizó cada ejecución, y se guardan dentro de un sistema de directorios dentro del directorio `/metrics`. La forma de este sistema de directorios es la siguiente:

```
/metrics/{nombre_archivo_semillas}/{tipo}/{configuración}/{fecha}.txt
```

En el apartado configuración, el nombre de las carpetas varían en función de el tipo de búsqueda.
En el caso de una búsqueda en anchura (`breath`), el nombre de la carpeta configuración se corresponderá con la cantidad de archivos descargados.
En el otro caso, búsqueda en profundidad (`depth`), el nombre de la carpeta configuración refleja la cantidad de archivos descargados y la profundidad de la búsqueda en este formato: `/{descargas}_{profundidad}`.

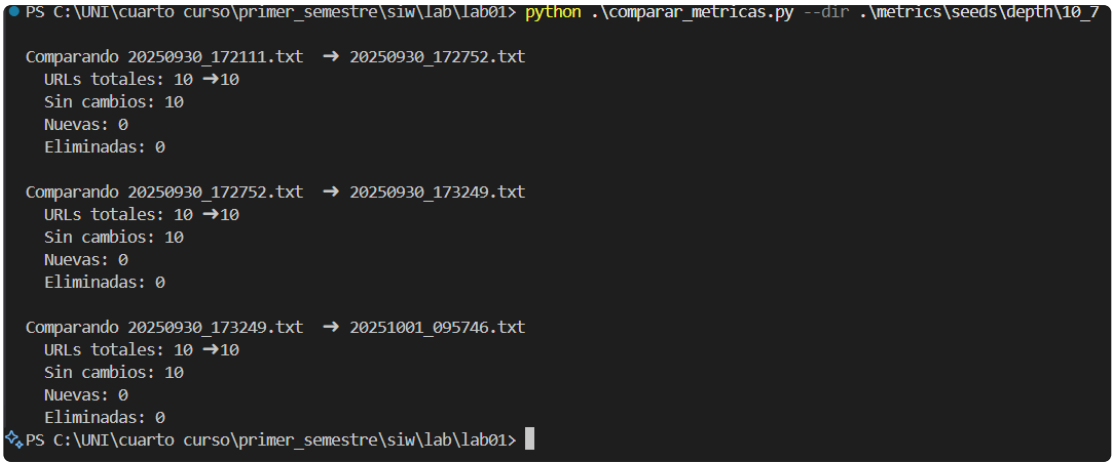
5.9. Comprobación de los resultados

Puede llegar a ser tedioso comparar dos archivos de texto buscando diferencias entre los links, sobre todo cuando la cifra empieza a crecer, por lo que incluyo un programa llamado `comparar_metricas.py` que compara los archivos de texto dentro de un

directorio en busca de diferencias entre las urls. Este script se ejecuta de la siguiente forma:

```
python comparar_metricas.py --dir /directorio
```

Un ejemplo de ejecución de este script sería el siguiente:



En caso de que una ejecución mostrara nuevas urls, estas saldrían reflejadas aquí

6. Entrega

En la entrega se incluyen:

- El archivo `crawler.py`
- El archivo `comparar_metricas.py`
- Dos archivos de semillas: `seeds.txt` y `seeds_2.txt`
- El directorio `/metrics` con al menos 1 ejecución del crawler para diferentes algoritmos con diferentes configuraciones. Aquí se recogen todas las URLS que deberían obtenerse del crawler en un principio (siempre que el `RANDOMIZE_BREADTH` sea `False` al menos)