2ª práctica de laboratorio de SIW
"Creación de un *crawler* básico"

Motivación

Un *crawler* es un programa que explora la Web de forma automática y recursiva, habitualmente con el propósito de alimentar el índice de un buscador web pero también para la creación de un *mirror* (una réplica de un sitio web dado). Con una política de visitas adecuada el uso de *crawlers* permite que las colecciones de documentos de los buscadores web estén actualizadas.

Los *crawlers* tienen un comportamiento muy diferente al de un usuario explorando un sitio web por lo que tienden a consumir más recursos del sitio que están "atacando". En consecuencia, los distintos sitios web suelen hacer uso de un fichero `robots.txt` que indica qué *crawlers* pueden visitar qué páginas y bajo qué condiciones.

Puedes aprender más sobre *crawlers* en los libros *"Mining the Web: Discovering Knowledge from Hypertext Data"* y *"Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data"* disponibles en el *Perusall* de la asignatura. También puedes consultar el artículo correspondiente en Wikipedia.

En el caso de que en algún momento futuro necesites usar un *crawler* puedes recurrir a alguna de las siguientes herramientas: *Wget* o *Apache Nutch*, p.ej. (hay más, por supuesto). También puedes usar *Scrapy* desde Python.

Descripción del ejercicio y del entregable

El objetivo del ejercicio es crear un script en Python que implemente un *crawler* básico para la descarga de archivos HTML. Dicho script deberá permitir al menos la configuración mediante el paso de 3 parámetros:

1. El nombre del fichero que contiene las URLs semilla.
2. El máximo de archivos a descargar.
3. El tiempo (en segundos) que debe esperar entre dos peticiones GET.

El archivo de URLs será un archivo de texto que tendrá una URL http o https por línea y que contendrá al menos una URL. Así, un archivo válido podría contener únicamente lo siguiente:

```
http://www.ingenieriainformatica.uniovi.es
```

Pero también sería un archivo válido otro con el siguiente contenido:

```
https://en.wikipedia.org/wiki/Dromaeosauroides
https://en.wikipedia.org/wiki/Theropoda
https://en.wikipedia.org/wiki/Dinosaur
https://en.wikipedia.org/wiki/Early_Cretaceous
```

El script sólo debe procesar y almacenar archivos HTML por lo que debería determinarse el tipo de contenido examinando las cabeceras de respuesta HTTP, en concreto `Content-type`.

El script procesará de manera recursiva el contenido de cada archivo HTML para buscar nuevos enlaces. A tal fin se recomienda utilizar la biblioteca Beautiful Soup que permite extraer con facilidad todos los elementos de un tipo: utiliza `find_all` para encontrar todos los elementos de tipo `a`, es decir, enlaces.

El script **no realizará descargas en paralelo** y deberá esperar obligatoriamente el número de segundos especificado por línea de órdenes entre una y otra petición.

Una vez se hayan descargado el máximo de documentos indicado por parámetro el script finalizará.

¡Atención! La exploración de las semillas puede hacerse primero en anchura o primero en profundidad. Puede implementarse cualquiera de ellas o ambas y que sea una opción configurable. En caso de implementar ambas se considerará como un *bonus* en la evaluación.

De manera opcional el script puede localizar el archivo `robots.txt` para los sitios que explore y actuar según sus indicaciones. Dicha funcionalidad también se considerará como un *bonus*. Para ello se puede utilizar la biblioteca `robotparser`.

En el campus virtual se entregará un archivo comprimido que contendrá el script así como algunos archivos de semillas indicando, además, para cada uno de ellos qué listado de archivos debería descargar el programa con una configuración dada. **No es necesario incluir los archivos descargados.**

Pseudocódigo

```
max_downloads = 10

def crawl(url, seconds):
    if max_downloads == 0:
        return
    html = download(url)                        # requests
    max_downloads -= 1
    sleep(seconds)                              # time
    links = parse_and_find_all_links(html)      # BeautifulSoup
    for l in links:
        l = normalize_link(url, l)
        crawl(l, seconds)

def normalize_link(url, link):
    if link.startswith("/") or link.startswith("#"):
        return join(url, link)                  # urlparse
    return link

crawl("http://example.com", 10)
```

Sobre Python

Las prácticas desde las 2ª hasta la 7ª (ambas inclusive) se deben desarrollar en Python. En caso de que seas "pythonista" utiliza el entorno de desarrollo que uses habitualmente. Si no fuera el caso deberías instalar Python 3 y un IDE (p.ej. PyScripter, PyCharm o incluso Visual Studio Code). Si tuvieras algún problema con la instalación del entorno en tu ordenador concierta una tutoría lo antes posible con dani@uniovi.es.

Recuerda que para poder utilizar algunas de las bibliotecas y módulos necesarios tendrás que instalarlos en tu entorno. P.ej., ejecutando desde una ventana de línea de órdenes `pip install beautifulsoup4` o `pip install –user beautifulsoup4` si lo estás instalando en un ordenador compartido (caso de los PCs de la Escuela).

Recuerda también que no basta con instalar la biblioteca sino que también tienes que importar el módulo correspondiente en tu script para poder usarlo.

Para refrescar la memoria puede ser útil https://www.pythoncheatsheet.org/ así como visitar la página de *Python for Beginners*.

Para cualquier duda sobre el ejercicio contacta con dani@uniovi.es lo antes posible.

Bibliotecas y módulos a utilizar en este ejercicio

- Requests: HTTP for Humans™
- time — Time access and conversions
- Beautiful Soup
- urllib.parse — Parse URLs into components
- urllib.robotparser — Parser for robots.txt