Prácticas de SAR

Sistemas de Almacenamiento y Recuperación de información

Práctica 6: Web Crawler

Web Crawler

Web Crawler

¿Qué es un Web Crawler?

- Un **web crawler** (también conocido como *indexador web*, *araña web*, *spider*, ...) es un programa que inspecciona las páginas web de forma metódica y automático.
- El uso más frecuentes de un web crawler es crear una copia de todas las páginas web visitadas para indexarlas y ser utilizadas posteriormente dentro de un motor de búsqueda.

Objetivo de la práctica

En esta práctica se propone la creación de un Web Crawler MUY simple.

Web Crawler. Dentro de un buscador

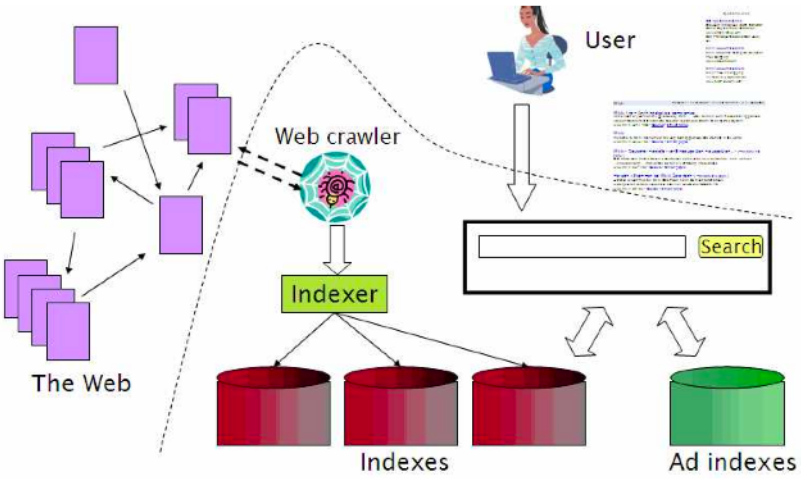

Figura 1: Web crawler dentro de un buscador web

Web Crawler. Esquema

Web Crawler. Esquema simplificado

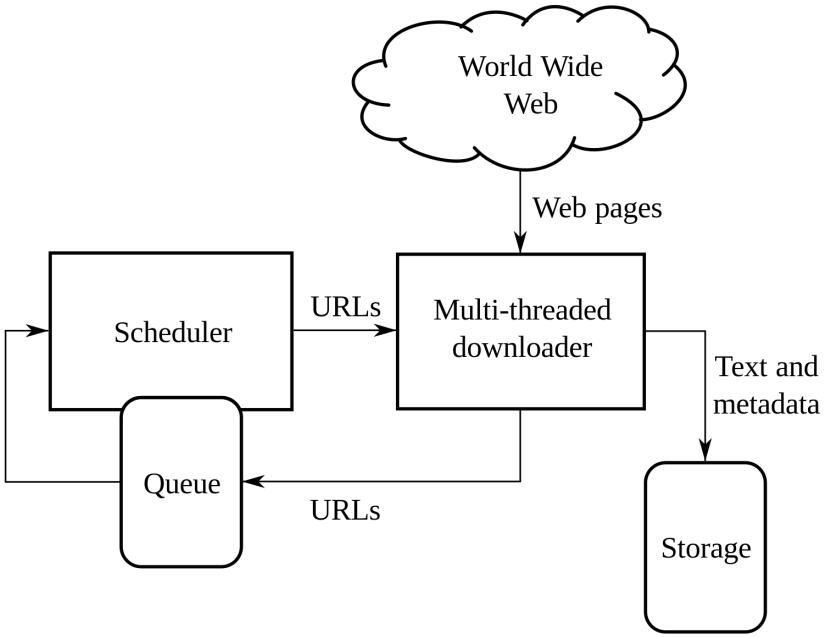

Figura 2: Esquema de un web crawler

```
inverted_index = ..
processed_urls = ..
pending_urls = [urls iniciales]
for iter in range(MAX):
    url = get_next_url(pending_urls)
    page = download_web(url)
    url_id = add_processed_url(processed_urls, url)
    add_to_index(inverted_index, url_id, extract_text(page))
    for new_url in extract_urls(page, url):
        if (new_url not in processed_urls) and (new_url not in pending_urls):
            add_pending_url(pending_urls, new_url)
```

Ejercicio

¿Qué debo hacer?

¿Qué debo hacer?

- Construir un web crawler básico. Para ello debes completar la plantilla **SAR_p6_web_crawler_plantilla**
- La plantilla incluye funciones completas y otras que debes completar
- Si lo consideras necesario puedes cambiar las cabeceras y los cuerpos de las funciones
- No se pueden añadir nuevas funciones

¿Qué funciona ya?

- El bucle principal del crawler
- Todo lo relativo a gestión de páginas web

¿Qué debo completar?

- Las funciones que trabajan con el índice invertido y el diccionario de documentos (urls)

Librerías utilizadas

```
import bs4
import colorama
from random import randrange
import re
import requests
from requests.exceptions import SSLError, ReadTimeout, ConnectTimeout, ConnectionError
import sys
from urllib.parse import urljoin, urlsplit, urlunsplit


OK = colorama.Fore.GREEN
ERROR = colorama.Fore.RED
BLUE = colorama.Fore.BLUE
BACKRED = colorama.Back.RED
RESET = colorama.Style.RESET_ALL

MAX_TOKEN_LEN = 15
CLEAN_RE = re.compile('\W+')
```

¿Qué funciona?

- Extracción del "site" de una url:

```
def get_site(url):
    return urlsplit(url).netloc
```

- Descarga de páginas web:

```
def download_web(url):
    print('Getting "%s" ... ' % url, end='')
    try:
        r = requests.get(url, timeout=1)
        print(OK + 'ok!' + RESET)
    except (SSLError, ReadTimeout, ConnectTimeout, ConnectionError) as err:
        print(ERROR + 'ERROR: %s' % err + RESET)
        return None
    return bs4.BeautifulSoup(r.text,  'lxml')#'html.parser')
```

- Extracción del texto de una página web:

```
def extract_text(content):
    return CLEAN_RE.sub(' ', content.text).lower()
```

¿Qué funciona?

- Extracción de los enlaces de una página web:

```
def extract_urls(contenido, baseurl):
    url_list = []
    for link in contenido.find_all('a'):
        newurl = link.get('href')
        if newurl is None:
            continue
        full_new_url = urljoin(baseurl, newurl.strip())
        surl = urlsplit(full_new_url)
        if surl.scheme in ['http', 'https']:
            ext = surl.path[surl.path.rfind('.'):].lower()
            if ext not in [".pdf", ".jpg"]:
                newurl = urlunsplit((surl.scheme, surl.netloc, surl.path, '', ''))
                url_list.append(newurl)
    return url_list
```

¿Qué funciona?

- Bucle principal:

```
if __name__ == "__main__":
    MAX = int(sys.argv[1]) if len(sys.argv) > 1 else 10
    inverted_index, processed_urls, pending_urls = {}, {}, []
    add_pending_url(pending_urls, "http://www.upv.es", processed_urls)
    for iter in range(MAX):
        url = get_next_url(pending_urls)
        print('(%d)' % iter, end=' ')
        page = download_web(url)
        if page is not None:
            url_id = add_processed_url(processed_urls, url)
            text = extract_text(page)
            add_to_index(inverted_index, url_id, text)
            url_list = extract_urls(page, url)
            for new_url in url_list:
                add_pending_url(pending_urls, new_url, processed_urls)
    info(inverted_index, processed_urls, pending_urls)
```

¿Qué funciona?

- Función para mostrar información (ii):

```
def info(index, processed, pending):
    print("\n====\nINFO\n====")
    # about de index
    print('Number of tokens:', len(index))
    print('Number of processed urls:', len(processed))
    if len(processed) != len(set(processed.values())):
        print (BACKRED + "ERROR: SOME URLS ARE DUPLICATED" + RESET)
    print('Number of pending urls:', len(pending))
    print('-' * 50)
    # searching words
    words = ["valencia", "upv", "google", "informática", "momento",
             "barcelona",  "proyecto"]
    for word in words:
        refs = get_posting(index, processed, word)
        if refs is None:
           print ("%s'%s'%s is not indexed" % (ERROR, word, RESET))
        else:
           print ("%s'%s'%s is in:" % (BLUE, word, RESET), ', '.join(sorted(refs)))
    print('-' * 50)
```

¿Qué funciona?

- Función para mostrar información (i):

```
    # about the sites
    l1 = sorted(set(get_site(url) for url in processed.values()))
    l2 = sorted(set(get_site(url) for url in pending_urls).difference(l1))
    max_len = max(len(s) for s in l1 + l2)
    l1 = ([s.ljust(max_len) for s in l1])
    l2 = ([s.ljust(max_len) for s in l2])
    print('Processed Sites (%d):' % len(l1))
    for i in range(int(len(l1)/4)+1):
        print('\t'+'\t'.join(l1[i*4:i*4+4]))
    print('-' * 50)
    print('Pending Sites (%d):' % len(l2))
    for i in range(int(len(l2)/4)+1):
        print('\t'+'\t'.join(l2[i*4:i*4+4]))
```

¿Qué debo completar?

```
def add_processed_url(url_dic, url):
    """Anyade url al diccionario de documentos

    Args:
        url_dic: el diccionario de documentos
        url: la url que se debe anyadir

    Returns:
        int: el docid de url dentro del diccionario
    """
    # COMPLETAR
    pass

def get_next_url(url_queue):
    """Extrae una url de la cola de urls url_queue y la devuelve

    Args:
        url_queue: una cola de urls

    Returns:
        text: una url de la cola
    """
    # COMPLETAR
    pass
```

¿Qué debo completar?

```
def add_pending_url(url_queue, url, url_dic):
    """Anyade url a la cola de urls si no esta todavia en ella o en el diccionario de documentos

        Args:
            url_queue: la cola de urls
            url: la url que se debe anyadir
            url_dic: el diccionario de documentos

        Returns:
            boolean: True si la url se ha anyadido. False si ya existia
    """
    # COMPLETAR
    pass
```

¿Qué debo completar?

```
def add_to_index(index, urlid, text):
    """Anyade el docid correscondiente a una url a las posting list de los terminos contenidos
    en text

        Args:
            index: el indice invertido
            urlid: el docid de la url
            text: el texto que se debe procesar

        Returns:
            int: numero de terminos procesador
    """
    # COMPLETAR
    pass
```

¿Qué debo completar?

```
def get_posting(index, dic, term):
    """Devuelve una lista con todas las urls donde aparece un termino
```

```
    Args:
        index: el indice invertido
        dic: el diccionario de documento, necesario para obtener las urls a partir de los
urlids
        term: el termino

    Returns:
        list: una lista con las urls donde aparece term, None si el termino no esta en el
indice invertido
"""
# COMPLETAR
pass
```