libreta

August 18, 2015

1 Búsqueda y ranqueo de páginas web

Este documento presenta una revisión y reimplementación del buscador de páginas web desarrollado en el capítulo 4 del *libro Programming Collective Intelligence*.

1.1 Diseño del buscador

1.2 Implementación

1.2.1 Módulos

Se utilizarán algunon módulos de Python que facilitará la implementación del buscador, en particular con el almacenamiento de la información (bases de datos) y la recolección e interpretación de las páginas web (descarga y parseo de documentos HTML).

```
In [1]: import urllib2
        from bs4 import*
        from urlparse import urljoin
        from sqlite3 import dbapi2 as sqlite
        import re
```

1.2.2 Información de prueba

Se diseñó el sitio web http://eduardoacye.github.io para realizar las pruebas del buscador implementado y se utilizan las *stop words* en español del sitio https://code.google.com/p/stop-words/

```
In [2]: test_urls = ["http://eduardoacye.github.io"]
        test_db   = "searchindex.db"
        ignore_words = set([line.strip() for line in
                            open("stop-words/stop-words_spanish_1_es.txt", "r")]
                           +
                           [line.strip() for line in
                            open("stop-words/stop-words_spanish_2_es.txt", "r")])
```

1.2.3 Funciones para el indexado

get_page

- `url` es una cadena de caracteres.

Regresa el contenido y la codificación de la página asociada a la URL

```
In [3]: def get_page(url):
            try:
                resource = urllib2.urlopen(url)
            except:
```

```
            raise Exception("Could not open %s" % url)
        encoding = resource.headers["content-type"].split("charset=")[-1]
        content = unicode(resource.read(), encoding)
        return content, encoding
```

parse_page

- `content` es una cadena unicode cuyo contenido son los caracteres que conforman a una página en HTML.

Regresa un objeto BeautifulSoup que representa el HTML parseado.

```
In [4]: def parse_page(content):
            html_tree = BeautifulSoup(content)
            return html_tree
```

has_href

- `link` es un objeto `BeautifulSoup.Tag`.

Regresa un booleano determinando si el enlace contiene un atributo href.

```
In [5]: def has_href(link):
            return "href" in link.attrs
```

link_url

- `base_url` es una cadena de caracteres que representa una URL.
- `link` es un objeto BeautifulSoup.Tag.

Regresa una cadena de caracteres que representa la URL del enlace.

```
In [6]: def link_url(base_url, link):
            url = urljoin(base_url, link["href"])
            if url.find("'") != -1:
                raise Exception("Malformed URL %s" % url)
            url = url.split("#")[0]
            return url
```

is_http

- `url` es una cadena de caracteres que representa una URL.

Regresa un booleano determinando si el recurso web que indica la URL es una página web.

```
In [7]: def is_http(url):
            return url[0:4] == "http"
```

db_connect

- `db_name` es una cadena de caracteres que representa el nombre de la base de una base de datos.

Regresa una conexión a la base de datos especificada por el nombre.

```
In [8]: def db_connect(db_name):
            return sqlite.connect(db_name)
```

db_close

- `connection` es un objeto obtenido con una llamada a `db_connect`.

Cierra la conexión con la base de datos.

```
In [9]: def db_close(connection):
            connection.close()
```

db_commit

- `connection` es un objeto obtenido con una llamada a `db_connect`.

Guarda en la base de datos los cambios realizados.

```
In [10]: def db_commit(connection):
             connection.commit()
```

db_create_tables

- `connection` es un objeto obtenido con una llamada a `db_connect`.

Crea las tablas de la base de datos correspondientes a la funcionalidad del crawler.

```
In [11]: def db_create_tables(connection):
             connection.execute("create table urllist(url)")
             connection.execute("create table wordlist(word)")
             connection.execute("create table wordlocation(urlid, wordid, location)")
             connection.execute("create table link(fromid integer, toid integer)")
             connection.execute("create table linkwords(wordid, linkid)")
             connection.execute("create index wordidx on wordlist(word)")
             connection.execute("create index urlidx on urllist(url)")
             connection.execute("create index wordurlidx on wordlocation(wordid)")
             connection.execute("create index urltoidx on link(toid)")
             connection.execute("create index urlfromidx on link(fromid)")
             db_commit(connection)
```

db_get_table y db_get_tables **Para db_get_table:**

- `connection` es un objeto obtenido con una llamada a `db_connect`. `-table` es una cadena de caracteres que representa el nombre de la tabla en la base de datos.

Regresa una lista con el contenido de la tabla especificada.
Para db_get_tables:

- `connection` es un objeto obtenido con una llamada a `db_connect`.

Regresa una tupla con listas con el contenido de cada tabla en la base de datos.

```
In [12]: def db_get_table(connection, table):
             table = connection.execute("select * from %s" % table)
             return table.fetchall()

         def db_get_tables(connection):
             return (db_get_table(connection, "urllist"),
                     db_get_table(connection, "wordlist"),
                     db_get_table(connection, "wordlocation"),
                     db_get_table(connection, "link"),
                     db_get_table(connection, "linkwords"))
```

is_indexed

- `connection` es un objeto obtenido con una llamada a `db_connect`.
- `url` es una cadena de caracteres que representa una URL.

Regresa un booleano que determina si la URL ya fué inspeccionada por el crawler.

```
In [13]: def is_indexed(connection, url):
             table = connection.execute("select rowid from urllist where url='%s'" % url)
             result = table.fetchone()
             if result is not None:
                 url_id = connection.execute("select * from wordlocation where urlid = %d"
                                             % result[0]).fetchone()
                 if url_id is not None:
                     return True
             return False
```

strip_html_tags

- `html` es un objeto `BeautifulSoup`.

Regresa una cadena unicode con el contenido del HTML sin las etiquetas.

```
In [14]: def strip_html_tags(html):
             html_inside_tag = html.string
             if html_inside_tag is None:
                 resulting_text = ""
                 for tag in html.contents:
                     sub_text = strip_html_tags(tag)
                     resulting_text += sub_text + "\n"
                 return resulting_text
             else:
                 return html_inside_tag.strip()
```

separate_words

- `text` es una cadena unicode que representa algún texto.

Regresa una lista con las palabras del texto.

```
In [15]: def separate_words(text):
             splitter = re.compile(ur"\W*", re.UNICODE)
             splitted = splitter.split(text)
             return [s.lower() for s in splitted if s != ""]
```

select_entry_id

- `connection` es un obteto obtenido con una llamada a `db_connect`.
- `table` es una cadena de caracteres que representa el nombre de una tabla de la base de datos.
- `column` es una cadena de caracteres que representa el nombre de una columna de la tabla especificada de la base de datos.
- `value`. es un valor que puede estar almacenado en la columna especificada de la tabla especificada de la base de datos.

Regresa `None` si no se encontró una entrada con el valor en la columna de la tabla, de lo contrario se regresa el id de la entrada.

```
In [16]: def select_entry_id(connection, table, column, value):
             table = connection.execute("select rowid from %s where %s = '%s'"
                                        % (table, column, value))
             result = table.fetchone()
             if result is None:
                 return result
             else:
                 return result[0]
```

insert_entry

- `connection` es un obteto obtenido con una llamada a `db_connect`.
- `table` es una cadena de caracteres que representa el nombre de una tabla de la base de datos.
- `column` es una cadena de caracteres que representa el nombre de una columna de la tabla especificada de la base de datos.
- `value` es un valor que puede estar almacenado en la columna especificada de la tabla especificada de la base de datos.

Crea una nueva entrada en la tabla especificada con el valor dado en la columna especificada.

```
In [17]: def insert_entry(connection, table, column, value):
             table = connection.execute("insert into %s (%s) values ('%s')"
                                        % (table, column, value))
             return table.lastrowid
```

index_page_words y index_page **Para index_page_words:**

- `connection` es un objeto obtenido con una llamada a `db_connect`.
- `url_id` es un id de la base de datos que está asociada a una página web.
- `words` es una lista de palabras.

Relaciona las palabras con la URL en la base de datos.
Para index_page:

- `connection` es un objeto obtenido con una llamada a `db_connect`.
- `url` es una cadena de caracteres que representa una URL.
- `html` es un objeto `BeautifulSoup`.

Procesa el documento de HTML para relacionar las palabras en el documento con la URL.

```
In [18]: def index_page_words(connection, url_id, words):
             print "      INDEXING WORDS FROM URL ID %s" % url_id
             for i in range(len(words)):
                 word = words[i]
                 if word in ignore_words:
                     continue
                 word_id = select_entry_id(connection, "wordlist", "word", word)
                 if word_id is None:
                     print "        %s" % word
                     insert_entry(connection, "wordlist", "word", word)
                     word_id = select_entry_id(connection, "wordlist", "word", word)
                 connection.execute("insert into wordlocation(urlid, wordid, location) values (%d,%d,%d
                                    % (url_id, word_id, i))

         def index_page(connection, url, html):
```

```
            if is_indexed(connection, url):
                return
            print "    INDEXING %s" % url
            text = strip_html_tags(html)
            words = separate_words(text)

            url_id = select_entry_id(connection, "urllist", "url", url)
            if url_id is None:
                insert_entry(connection, "urllist", "url", url)
                url_id = select_entry_id(connection, "urllist", "url", url)
            index_page_words(connection, url_id, words)
            print "      INDEXED! %s" % url
```

index_link_words y index_link **Para index_link_words:**

- `connection` es un objeto obtenido con una llamada a `db_connect`.
- `link_id` es un id de la base de datos que está asociada a enlace.
- `words` es una lista de palabras.

Relaciona las palabras con el enlace en la base de datos.
Para index_link:

- `connection` es un objeto obtenido con una llamada a `db_connect`.
- `from_url` es una cadena de caracteres que representa una URL.
- `to_url` es una cadena de caracteres que representa una URL.
- `link` es un objeto `BeautifulSoup.Tag`.

Procesa la etiqueta de HTML ... para relacionar las palabras en el enlace con las URLs involucradas.

```
In [19]: def index_link_words(connection, link_id, words):
             for word in words:
                 if word in ignore_words:
                     continue
                 word_id = select_entry_id(connection, "wordlist", "word", word)
                 if word_id is None:
                     print "        PALABRA %s" % word
                     insert_entry(connection, "wordlist", "word", word)
                     word_id = select_entry_id(connection, "wordlist", "word", word)
                 connection.execute("insert into linkwords(wordid, linkid) values (%d,%d)"
                                    % (word_id, link_id))

         def index_link(connection, from_url, to_url, link):
             text = strip_html_tags(link)
             words = separate_words(text)

             from_url_id = select_entry_id(connection, "urllist", "url", from_url)
             if from_url_id is None:
                 insert_entry(connection, "urllist", "url", from_url)
                 from_url_id = select_entry_id(connection, "urllist", "url", from_url)
             to_url_id = select_entry_id(connection, "urllist", "url", to_url)
             if to_url_id is None:
                 insert_entry(connection, "urllist", "url", to_url)
                 to_url_id = select_entry_id(connection, "urllist", "url", to_url)

             if from_url_id == to_url_id:
```

```
            return
        table = connection.execute("insert into link (fromid, toid) values (%d,%d)"
                                   % (from_url_id, to_url_id))
        link_id = table.lastrowid
        index_link_words(connection, link_id, words)
```

crawl

- `urls` es una lista de cadenas de caracteres que representan URLs.
- `connection` es un objeto obtenido con una llamada `db_connect`.
- `depth` es un entero positivo que representa la profundidad a la que se procesarán las páginas (similar a la profundidad de una búsqueda en grafos).

Navega por páginas web a partir de la lista de URLs dada, procesando palabras y enlaces.

```
In [20]: def crawl(urls, connection, depth=2):
             print "CRAWL"
             for i in range(depth):
                 print "  DEPTH %d" % i
                 new_urls = set()
                 print "  URLs %s" % urls
                 for url in urls:
                     content, encoding = get_page(url)
                     html = parse_page(content)
                     print "    VISITED %s" % url
                     index_page(connection, url, html)

                     links = html.select("a")
                     for link in links:
                         if has_href(link):
                             ref_url = link_url(url, link)
                             if is_http(ref_url) and not is_indexed(connection, ref_url):
                                 new_urls.add(ref_url)
                             index_link(connection, url, ref_url, link)
                 urls = list(new_urls)
```

```
In [ ]:
```