Prácticas de SAR

Sistemas de Almacenamiento y Recuperación de información

Práctica 5: Whoosh

¿Qué es Whoosh?

Whoosh

- Whoosh es una librería de clases y funciones para indexar texto y luego buscar en el índice. Le permite desarrollar motores de búsqueda personalizados para tu contenido. Por ejemplo, si tienes un site de blogs, puedes usar whoosh para agregar una función de búsqueda de entradas en los blog.
- Como *Lucene*, Whoosh no es un motor de búsqueda, es una librería para crear motores de búsqueda.
- La indexación del texto, el nivel de información almacenada para cada término en cada campo, el análisis de las consultas de búsqueda, los tipos de consultas permitidas, los algoritmos de ranking, etc. son todos personalizables, reemplazables y extensibles.

Whoosh

- Whoosh es rápido, pero utiliza sólo Python puro, por lo que se ejecutará en cualquier lugar donde tengas Python, sin necesidad de un compilador.
- Whoosh utiliza, por defecto, Okapi BM25F, pero la función de ranking se puede personalizar.
- Whoosh crea índices bastante pequeños en comparación con muchas otras librerías.
- Todo el texto indexado en Whoosh debe ser unicode.
- Whoosh permite almacenar objetos Python con los documentos indexados.
- **Documentación de Whoosh**

¿Cómo funciona Whoosh?

Whoosh. Ejemplo de Indexación

```
from whoosh.index import create_in
from whoosh.fields import ID, TEXT

schema = Schema(title=TEXT(stored=True), path=ID(stored=True), content=TEXT)
ix = create_in("directorio_del_índice", schema)

writer = ix.writer()
writer.add_document(title="First document", path="/a",
                        content="Este es el texto del primer documento")
writer.add_document(title="Second document", path="/b",
                        content="Este es el texto de nuestro segundo documento")
```

.
.
.

```
writer.commit()
```

Whoosh. Ejemplo de Indexación

1. Se crea el esquema del índice. El esquema la lista de campos que contiene el índice. De cada campo se debe indicar la siguiente información:
   - el nombre del campo.
   - el tipo del campo: ID, STORED, KEYWORD, TEXT, NUMERIC, BOOLEAN, DATETIME, NGRAM, NGRAMWORDS.
   - si es almacena o no junto al índice.
   - el analizador que se debe utilizar (lo veremos luego).

```
schema = Schema(title=TEXT(stored=True), path=ID(stored=True), content=TEXT)
```

2. Se crea el índice in disco. Se le debe indicar un directorio para guardarlo. Opcionalmente se puede indicar un nombre (por defecto 'MAIN').

```
ix = create_in("directorio_del_índice", schema)
```

Whoosh. Ejemplo de Indexación

3. Se crea un escritor (writer) y se van añadiendo los documentos. No es necesario dar valor a todos los campos de cada documento.

```
writer = ix.writer()
writer.add_document(title="First document", path="/a",
content="Este es el texto del primer documento")
```

4. Se confirman los cambios.

```
writer.commit()
```

Whoosh. Ejemplo de Búsqueda

```
from whoosh.index import open_dir
from whoosh.qparser import QueryParser

ix = open_dir("directorio_del_índice")
# con with no es necesario cerrar el buscador
with ix.searcher() as searcher:
        parser = QueryParser("content", ix.schema)
        query = parser.parse("el_texto_de_la_consulta")
        results = searcher.search(query)
        print (results[0])
```

Whoosh. Ejemplo de Búsqueda

1. Se abre el índice a partir del directorio. Se puede indicar también el nombre del índice (si le hemos cambiado en nombre por defecto)

```
ix = open_dir("directorio_del_índice")
```

2. Se crea un búscador (searcher). Una vez terminadas las búsquedas se debe cerrar el searcher.

```
searcher = ix.searcher()
    ...
searcher.close()
```

Whoosh. Ejemplo de Búsqueda

3. El buscador necesita un objeto de tipo Query para conseguirlo se debe parsear la consulta.

```
parser = QueryParser("content", ix.schema)
query = parser.parse("el_texto_de_la_consulta")
```

4. Opcionalmente podemos hacer un parseo manual.

```
from whoosh.query import *
myquery = And([Term("content", "termino1"), Term("content", "termino2")])
```

5. Se lanza la búsqueda y se obtienen los resultados.

```
results = searcher.search(query)
results[0]
```

Ejercicios

Ejercicio 1

1. Modifica el programa "SAR_p5_Indexer.py" para indexa los ficheros correspondientes a las noticias de enero.
2. Utiliza varios analizadores distintos y comprueba como afecta eso al tamaño del índice. Justifica por qué ocurre esto.
3. Modifica el programa "SAR_p5_Searcher.py" para realiza las siguientes búsquedas sobre el índice:
   - Busca los documentos que contengan el texto valencia (31 documentos)
   - Busca valencia con la restricción de que no aparezca el término Salenko (13 documentos).
   - Busca documentos con el texto futbol (31 documentos).
   - Busca los términos Los Angeles y Aeroflot (3 documentos).

Ejercicio 2

Cada documento de enero contiene todas las noticias de un día. Los resultados de las búsquedas son poco útiles.

Modifica el indexador y el buscador para que indexen cada noticia como un documento virtual distinto. De esta manare las búsquedas serán más precisas.

a) Antes de indexar un documento divídelo por noticias e indexa por separado cada noticia como un documento.

b) Añade un identificador global a cada documentos (como un nuevo campo).

c) Añade un campo que identifique a cada noticia dentro del documento real.

d) Modifica el programa "SAR_p5_Searcher.py" para devolver además del nombre del fichero, el identificador global del documento y la posición que la noticia encontrada ocupa dentro del fichero "real"; los campos añadidos en el apartado b) y c).

e) Modifica el programa "SAR_p5_Searcher.py" para que si el resultado de la búsqueda son menos de 4 ficheros, además de la información anterior devuelva todo el contenido de la noticia. El cuerpo de la noticia NO debe almacenarse en el índice.

f) Realiza las mismas búsquedas que en el apartado d) del Ejercicio 1. Escribe las consultas realizadas y el número de resultados obtenidos.

Cosas útiles

Tokenización de documentos

- En whoosh un analizador es una función que recibe una cadena y devuelve una lista de tokens. Básicamente, un analizador es un tokenizador y una serie de filtros.
- Por defecto whoosh como tokenización divide por espacios y como filtro sólo pasa a minúsculas.
- Whoosh incorpora diferentes tokenizadores y filtros.

```
from whoosh.analysis import LowercaseFilter, RegexTokenizer

tokenizer = RegexTokenizer()
lfilter = LowercaseFilter()

tokens = tokenizer("These ARE the   **things I want!")
for token in lfilter(tokens):
    print (token.text)
```

Tokenización de documentos

- Al crear un esquema se puede indicar el analizador que debe utilizar whoosh:

```
my_analyzer = RegexTokenizer() | LowercaseFilter() | StopFilter()

schema = Schema(content=TEXT(analyzer=my_analyzer))
```

Más información sobre analizadores en Whoosh