Proyecto de prácticas:
Recuperador de Noticias

Sistemas de Almacenamiento y Recuperación de Información
(1ª parte del proyecto coordinado SAR - ALT)

versión 1.4

Trabajo en grupo (grupos de 4 personas). Si el grupo de trabajo para el proyecto de prácticas es distinto al grupo para el proyecto de teoría, avisad al profesor de prácticas.

Objetivo

El presente proyecto de prácticas de SAR es la primera parte del proyecto conjunto entre las asignaturas de *Sistemas de Almacenamiento y Recuperación de Información* (semestre 3B) y *Algorítmica* (semestre 4A). Esto implica que el código que desarrolle cada grupo se utilizará, por el mismo grupo, como base para el proyecto de la asignatura de ALT en el próximo curso.
El objetivo del presente proyecto es la implementación en *python3* de un sistema de indexación y recuperación de noticias.
En PoliformaT/Lessons se proporcionan los tres ficheros siguientes con extensión `.py`:

- `SAR_Indexer.py`. Programa principal para la indexación de noticias. Crea un objeto de la clase `SAR_Project` para extraer las noticias de una colección de documentos alojados en un directorio local, las indexa y guarda en disco la instancia del objeto.
- `SAR_Searcher.py`. Programa principal para la recuperación de noticias. Carga de disco una instancia de la clase `SAR_Project` para responder a las consultas que se le realicen.
- `SAR_lib.py`. Este es el único fichero que se debe modificar para dotar a los dos anteriores de sus funcionalidades. El funcionamiento básico del proyecto (indexación y recuperación de noticias) se puede conseguir completando un número reducido de métodos de la clase `SAR_Project`. A partir de aquí, cada funcionalidad adicional que se quiera añadir supondrá completar uno o dos métodos adicionales y posiblemente modificar alguno de los ya implementados. También se pueden añadir nuevos métodos y/o atributos si se considera necesario.

Nota: Solamente hay que modificar el fichero `SAR_lib.py`, los otros dos (`SAR_Indexer.py` y `SAR_searcher.py` ni se tocan ni se entregan).

Entrega y Evaluación

En PoliformaT se abrirá una tarea para que el representante de cada grupo suba:

- El fichero `SAR_lib.py` con las funcionalidades implementadas.
- Una **memoria** del proyecto en formato pdf.

Se permite realizar nuevas entregas mientras la tarea de PoliformaT esté abierta (se considera la última). Tanto en el fichero fuente como en la memoria deberán identificarse TODAS las funcionalidades implementadas y TODOS los miembros del grupo.
La nota máxima del proyecto de prácticas será de **3 puntos**, esto incluye tanto el código como la memoria descriptiva.
La aplicación deberá contar con unas **funcionalidades mínimas** que se puntuarán en total (código y memoria) con un máximo de **1,5 puntos**. Adicionalmente, se podrán **ampliar las funcionalidades** para obtener mayor nota, hasta un máximo total (código de las ampliaciones y la parte de la memoria correspondiente a las ampliaciones) de **1,5 puntos adicionales**.

1. Funcionalidades básicas (hasta 1,5 puntos)

1.1. Indexador (`SAR_Indexer.py`)

La Figura **??** muestra el mensaje de ayuda del programa `SAR_Indexer.py`. Se ha incluido la biblioteca `argparse`, para facilitar el análisis de los argumentos de entrada del programa. Este análisis ya está incluido en el código que hay disponible en PoliformaT y, por tanto, no se debe modificar ni subir el fichero `SAR_Indexer.py`.

```
$ python SAR_Indexer.py --help
usage: SAR_Indexer.py [-h] [-S] [-P] [-M] [-O] newsdir index

Index a directory with news in json format.

positional arguments:
  newsdir           directory with the news.
  index             name of the file to save the project object.

optional arguments:
  -h, --help        show this help message and exit
  -S, --stem        compute stem index.
  -P, --permuterm   compute permuterm index.
  -M, --multifield  compute index for all the fields.
  -O, --positional  compute positional index.
```

Figura 1: Ayuda del `SAR_Indexer.py` (incluye las funcionalidades ampliadas).

Las funcionalidades básicas con las que debe contar el indexador son las siguientes:

- Requiere obligatoriamente dos argumentos de entrada:
  - `newsdir`, el directorio donde está la colección de noticias en formato `json`, y
  - `index`, el nombre del fichero donde se guardará el objeto `SAR_Project` con los índices creados.
- El programa crea un objeto de la clase `SAR_Project` e invoca a su método `index_dir`.

  ```
  indexer = SAR_Project()
  t0 = time.time()
  indexer.index_dir(newsdir, **vars(args))
  ```

  El método `index_dir`, junto con el resto de métodos a los que éste llame, deberá:

  - Procesar los documentos `.json` del directorio de manera recursiva y extraer las noticias.
  - Tokenizar cada noticia:
    - Eliminando símbolos no alfanuméricos (comillas, interrogantes, ...) y
    - extrayendo los términos (consideraremos separadores de términos los espacios, los saltos de línea y los tabuladores).

    No se deben distinguir mayúsculas y minúsculas en la indexación.
  - Asignar a cada documento un identificador único (docid) que será un entero secuencial.
  - Asignar a cada noticia un identificador único. Se debe saber a qué documento (fichero) pertenece cada noticia y qué posición relativa ocupa dentro de él.
  - Crear un índice invertido accesible por término. Cada entrada contendrá una lista con las noticias en las que aparece ese término.
- Finalmente, el programa principal guarda en disco el objeto `indexer` utilizando la biblioteca `pickle`. También muestra información sobre el proceso de indexado.

  ```
  t1 = time.time()
  with open(indexfile, 'wb') as fh:
      pickle.dump(indexer, fh)
  t2 = time.time()
  indexer.show_stats()
  print("Time indexing: %2.2fs." % (t1 - t0))
  print("Time saving: %2.2fs." % (t2 - t1))
  print()
  ```

A continuación se muestra un extracto del método `index_dir` ya incluido en la biblioteca.

```
def index_dir(self, root, **args):

    ...

    for dir, subdirs, files in os.walk(root):
        for filename in files:
            if filename.endswith('.json'):
                fullname = os.path.join(dir, filename)
                self.index_file(fullname)

    ##########################################
    ## COMPLETAR PARA FUNCIONALIDADES EXTRA ##
    ##########################################
```

Como se puede observar, el método realiza el recorrido recursivo por el directorio utilizando la función `walk` de la biblioteca `os`. Para conseguir la funcionalidad básica de indexado sólo hace falta completar el método `index_file`.

```
def index_file(self, filename):

    ...

    with open(filename) as fh:
        jlist = json.load(fh)

    #################
    ### COMPLETAR ###
    #################
```

La variable `jlist` es una lista con tantos elementos como noticias hay en el fichero. Cada noticia es un diccionario con los campos *title*, *date*, *keywords*, *article*, *summary*. En la versión básica sólo debe ser indexado el contenido de la clave *article*.

Por ejemplo, podemos iterar sobre las noticias de `jlist`, tomar el contenido de *article* e indexarlo de la forma siguiente:

```
# COMPLETAR: asignar identificador al fichero 'filename'
for new in jlist:
    # COMPLETAR: asignar identificador a la notica 'new'
    content = new['article']
    # COMPLETAR: indexar el contenido de 'content'
```

1.2. Recuperador de noticias (SAR_searcher.py)

La Figura **??** muestra el mensaje de ayuda del programa `SAR_Searcher.py`. Se ha incluido la biblioteca `argparse` para facilitar el análisis de los argumentos de entrada. Este análisis ya

```
$ python SAR_Searcher.py --help

usage: SAR_Searcher.py [-h] [-S] [-N | -C] [-A] [-R]
                       [-Q query | -L qlist | -T test]
                       index

Search the index.

positional arguments:
  index                 name of the file with the index object.

optional arguments:
  -h, --help            show this help message and exit
  -S, --stem            use stem index by default.
  -N, --snippet         show a snippet of the retrieved documents.
  -C, --count           show only the number of documents retrieved.
  -A, --all             show all the results. If not used, only the first 10
                        results are showed. Does not apply with -C and -T
                        options.
  -R, --rank            rank results. Does not apply with -C and -T options.
  -Q query, --query query
                        query.
  -L qlist, --list qlist
                        file with queries.
  -T test, --test test  file with queries and results, for testing.
```

Figura 2: Ayuda del `SAR_Searcher` (incluye funcionalidades ampliadas).

está incluido en el código que hay disponible en PoliformaT y, por tanto, no se debe modificar ni subir el fichero `SAR_Searcher.py`.
Las funcionalidades básicas del recuperador de noticias (*searcher*) son las siguientes:

- Sólo tiene un argumento obligatorio, `index`, que es el nombre del fichero que contiene el objeto de la clase `SAR_Project` guardado previamente con el programa `SAR_Indexer.py`.
- Si sólo se le pasa el argumento `index` (si no le pasamos argumentos opcionales), el programa entrará en un bucle de petición de consulta y devolución de las noticias relevantes. Además del modo interactivo, podemos pasarle consultas directamente desde la línea de comandos mediante argumentos opcionales de tres formas:
  - `-Q` permite pasar una consulta directamente en la llamada al programa. Por ejemplo, `python SAR_Searcher.py -Q 'messi AND valencia' indice_2015.bin`, resuelve la consulta `'messi AND valencia'` utilizando para ello el índice guardado en el fichero `indice_2015.bin`, muestra el resultado y finaliza.
  - `-L` para pasar una lista de consultas mediante un fichero. Por ejemplo, `python SAR_Searcher.py -L query_list.txt indice_2015.bin`, resuelve las consultas contenidas en el fichero `query_list.txt` utilizando el índice guardado en el fichero `indice_2015.bin`, muestra el resultado consulta por consulta y finaliza.

- `-T` se utiliza para pasar una lista de consultas resueltas para evaluar el recuperador. Esto permite determinar si el programa funciona correctamente. Cada línea del fichero de consultas debe tener una consulta y el número de noticias que se deben recuperar para esa consulta; los dos valores (consulta y cardinalidad de la respuesta) deben estar separados por un tabulador. Por ejemplo, `python SAR_Searcher.py -T query_test_list.txt indice_2015.bin` comenzará a comprobar si todas las consultas contenidas en `query_test_list.txt` obtienen el resultado esperado sobre el índice guardado en el fichero `indice_2015.bin`. Si todas las consultas devuelven el número de noticias esperado, se mostrará el mensaje `'Parece que todo ha ido bien, buen trabajo!'`, en caso contrario, se mostrará un mensaje de error resaltando la consulta con el resultado incorrecto.

- Se debe permitir utilizar las conectivas AND, OR y NOT en las consultas. El orden de evaluación y prioridad de las conectivas será de izquierda a derecha. Por ejemplo, la consulta `'term1 AND NOT term2 OR term3'` deberá devolver las noticias que contienen `term1` pero no `term2` más las que contienen `term3`.

  IMPORTANTE: para realizar la intersección (AND) y unión (OR) de *postings list* se deben implementar los algoritmos de *merge* vistos en teoría, en los métodos:

  - `and_posting(self, p1, p2)` y
  - `or_posting(self, p1, p2)`.

- La presentación de los resultados se realizará en función de los argumentos opcionales con los que se ejecute el programa `python SAR_Searcher.py`:

  - `-C`, muestra la consulta y el número de noticias devuelto por el buscador.
  - `-N`, muestra la consulta y, para cada noticia recuperada por el buscador:
    - Un número de orden para numerar las noticias recuperadas.
    - Una medida de similitud (*score*) entre la consulta y la noticia. Si no se implementa la funcionalidad de ranking, el *score* de todas las noticias será 0.
    - El identificador de la noticia (`newid`).
    - La fecha de la noticia.
    - El título de la noticia.
    - Las *keywords* de la noticia.
    - Un trozo de la noticia donde aparezcan las palabras de la consulta encontradas en la noticia (`snippet`).
  - Si no se indica ninguno de los argumentos anteriores se mostrará, en una línea por noticia, la misma información que en la opción `-N` pero sin incluir el *snippet*.

  En el apéndice de este documento se incluyen algunos ejemplos de resultados con diferentes opciones.

- Si se incluye el argumento `-A`, el recuperador mostrará todas las noticias recuperadas, en caso contrario (la opción por defecto) sólo se mostrarán las 100 primeras.

2. Funcionalidades ampliadas (hasta 1,5 puntos)

Para obtener la máxima puntuación, además de las funcionalidades básicas, se deberán implementar perfectamente al menos cinco de las siguientes funcionalidades extra.

Atención: Las funcionalidades extra sólo se puntuarán si las funcionalidad básicas funcionan correctamente.

2.1. Stemming

Permitir la realización de *stemming* de las noticias y las consultas. Se ha añadido un parámetro (`-S`, `--stem`) para activar esta funcionalidad tanto al crear los índices como al hacer las consultas.

Ya se ha incluido un *stemmer* en castellano; para obtener el *stem* de un *token* es suficiente con hacer `self.stemmer.stem(token)`. La clase `SAR_Project` tiene un atributo `self.sindex` para guardar el índice de *stems*.

Además de otros métodos, para conseguir esta funcionalidad se sugiere completar y utilizar los métodos:

- `make_stemming`, para crear el índice de *stems*.
- `get_stemming`, para obtener la postings list asociada al *stem* de un término.

2.2. Multifield

En el funcionamiento básico de `SAR_Indexer.py` sólo se indexa el contenido del campo `'article'` del diccionario de la noticia.

Con la ampliación *multifield*, se añadirán índices adicionales para el titular de la noticia, la categoría y la fecha. Una vez implementada la funcionalidad, los términos de las consultas podrán incluir uno de los prefijos `'title:'`, `'article:'`, `'summary:'`, `'keywords:'` y `'date:'` para indicar que ese término se debe buscar en el índice de los titulares, el cuerpo, el resumen, las *keywords* o la fecha, respectivamente. Si no se indica ningún campo, el término se buscará en el índice del cuerpo de la noticia (`'article'`).

Se debe permitir mezclar búsquedas sobre diferentes índices en la misma consulta.

Ejemplo de funcionamiento: la consulta `'title:messi AND valencia'` deberá recuperar las noticias donde aparezca `'messi'` en el titular y `'valencia'` en el cuerpo de la noticia.

Como recomendación de implementación se sugiere utilizar el índice principal, `self.index`, para guardar los índices de todos los campos. Esto se puede conseguir haciendo que sea un diccionario de diccionarios de forma que tenga un primer nivel para determinar el campo. Por ejemplo, `self.index['title']` se corresponda con el diccionario para almacenar los términos del campo `'title'`.

Debe tokenizarse el contenido de todos los campos a excepción del campo `'date'`, que contiene el valor de la fecha de la noticia y, por este motivo, no debe tokenizarse. En la plantilla del fichero `SAR_Indexer.py` se ha añadido una lista con el nombre de todos los campos junto con un booleano que determina si el campo debe ser tokenizado o no:

```
# lista de campos, el valor booleano indica si se debe tokenizar el campo
# NECESARIO PARA LA AMPLIACION MULTIFIELD
fields = [("title", True), ("date", False), ("keywords", True),
          ("article", True), ("summary", True)]
```

2.3. Búsquedas posicionales

Esta ampliación permite la búsqueda de varios términos consecutivos utilizando las dobles comillas. Esto hace necesario el uso de *postings list* posicionales.

Ejemplo de funcionamiento: buscar `'"fin de semana"'` con la funcionalidad implementada deberá devolver sólo las noticias en las que los tres términos aparecen de forma consecutiva, mientras que buscar `'fin de semana'` encontraría todas las noticias en las que aparecen los tres términos sin importar la posición.

Para conseguir este funcionamiento en el indexador, será necesario almacenar en el índice invertido, además de los documentos en los que aparece cada termino, en qué posiciones aparecen.

Para el recuperador, será necesario completar el método `get_positionals` para obtener la *postings list* de una secuencia consecutiva de términos.

2.4. Ranking

Esta funcionalidad permitirá devolver las noticias ordenadas en función de su relevancia, utilizando para ello una medida de similitud entre la noticia y la consulta. La puntuación de ordenación de los documentos debe aparecer en los resultados. El criterio de ordenación será la similitud coseno con pesado **tf*idf** vista en teoría.

Para facilitar la implementación de esta funcionalidad, se ha añadido el atributo `self.weight`. Además, será necesario completar el método `rank_result` para ordenar la lista de resultados.

2.5. Permuterm

Esta funcionalidad permitirá la búsqueda utilizando `'*'` y `'?'` como comodines (*wildcard queries*). Sólo se permite un comodín por palabra.

Ejemplo de funcionamiento: buscar `'s*dney'` encontraría todas las noticias que contengan algún término que comience por `'s'` y termine en `'dney'`.

Para lograr esta funcionalidad será necesario implementar índices *permuterm*. Sin embargo, no os compliquéis mucho con la estructura/eficiencia del índice: una lista ordenada por *permuterm* y una búsqueda dicotómica puede dar buenos resultados.

En esta ampliación se recomienda utilizar el atributo `self.ptindex` para guardar el índice *permuterm* y completar el método `get_permuterm` para obtener la *postings list* de un termino utilizando el índice *permuterm*.

2.6. Paréntesis

Con la funcionalidad básica, la aplicación debe permitir el uso de conectivas `'AND'` y `'OR'`, además del `'NOT'`, para hacer consultas complejas. En ese caso, la evaluación predeterminada será de izquierda a derecha.

Con esta ampliación se permitirá el uso de paréntesis, `'('` y `')'`, para modificar la prelación/precedencia de los operadores lógicos.

Ejemplo de funcionamiento: la consulta `'term1 AND NOT (term2 OR term3)'` deberá devolver las noticias que contienen `'term1'` pero no `'term2'` ni `'term3'`.

Para lograr esta funcionalidad será necesario ampliar el método `solve_query` para realizar algún tipo de análisis sintáctico (*parsing*) más elaborado de la consulta. Para facilitar el análisis se puede asumir que todas las consultas estarán bien construidas (es decir, no es necesario detectar errores sintácticos).

2.7. Funcionamiento conjunto de las funcionalidades extra

Se espera que todas las funcionalidad extra implementadas funcionen de forma conjunta en la misma consulta, teniendo en cuenta que:

- El uso de *stemming* no se aplica a las búsquedas posicionales. Con la opción de *stemming* activada, la consulta `'día AND "semana"'` realizará *stemming* en día pero no en semana.
- En las búsquedas posicionales no se podrán utilizar comodines. La consulta `'"fin de sem*a"'` no está permitida.

3. FAQ

A continuación se da respuesta a algunas preguntas frecuentes. Si surgen más dudas se irán añadiendo en revisiones sucesivas de este documento.

¿Qué se entrega?

El responsable del grupo deberá subir a la tarea de PoliformaT únicamente dos ficheros:

- `SAR_lib.py`, con las funcionalidades implementadas.
- `informe.pdf`, con el informe descriptivo del trabajo realizado. El informe debe incluir como mínimo:
  - Enumeración de las funcionalidades extra implementadas.
  - Descripción y justificación de las decisiones de implementación realizadas. En el caso de la ampliación *ranking* hay que detallar la distancia utilizada.
  - Descripción del método de coordinación utilizado por los miembros del grupo en la realización del proyecto.

- Descripción de la contribución al proyecto de cada miembro del grupo.
- Podéis añadir toda la información que consideréis importante, incluidas opiniones subjetivas de la marcha del proyecto y las dificultades afrontadas.

Se permiten los reenvíos mientras la tarea esté activa, se evaluará la última entrega.

¿Cómo será la evaluación del proyecto?

Para evaluar el proyecto se tendrán en cuenta tres factores:

1. El funcionamiento correcto tanto del programa básico como de las funcionalidades extras implementadas. Se utilizará la opción de test, argumento `-T` en el buscador, para comprobar que los resultados son los esperados. Para hacer esta evaluación se utilizarán los programas `SAR_Indexer.py` y `SAR_Searcher.py` originales y la última versión de la biblioteca `SAR_lib.py` subida por el responsable del grupo a la tarea de PoliformaT.

2. El código enviado. Se hará una evaluación de la eficiencia y corrección del código desarrollado por el grupo, penalizándose las implementación poco eficiente o excesivamente farragosas. Se recomienda añadir comentarios donde se considere oportuno (sin llegar al extremo de comentar las cosas demasiado obvias).

3. La memoria del proyecto.

4. Una prueba objetiva individual y por grupo.

¿Hay que procesar los documentos con alguna biblioteca?

Los documentos serán ficheros *json*. Cada fichero contendrá las noticias correspondientes a un día o una semana. La carga de los ficheros *json* ya está incluida en la plantilla de la biblioteca `SAR_lib.py`.

¿Las ampliaciones han de ser acumulativas?

Para obtener la máxima puntuación sí, las ampliaciones deben ser acumulativas, con las dos únicas excepciones comentadas en el apartado correspondiente.
Entre los ficheros de ayuda proporcionados hay ejemplos de combinaciones de funcionalidades (sobre todo para poder comprobar si se obtiene el resultado esperado).

¿Cómo se generan los snippets?

La descripción de *snippet* da libertad para generarlos de varias formas, siempre que esas formas tengan sentido. Algunas opciones (existen otras) serían:

- **Trabajar a nivel de palabras** (el cuerpo de la noticia como lista de palabras). Obtener el índice de la primera y última ocurrencia de cada término (puede haber términos que no aparezcan en el cuerpo de la noticia). Quedarse con las posiciones mínimo y máximo de los índices anteriores. Extraer un fragmento de texto entre las posiciones anteriores añadiendo un pequeño contexto a izquierda y derecha. Esta opción puede dar snippets muy largos si hay términos al inicio y al fin de la noticia.
- **Sacar un snippet de cada término** (su primera ocurrencia en el documento, para simplificar) poniendo dicho término con un contexto antes y después. Opcionalmente se pueden unir segmentos que se solapen. Esta opción es "ligeramente" más compleja la anterior.

No hace falta respetar ni las mayúsculas ni los saltos de línea del documento original. Se puede trabajar con los textos normalizados.
En cualquier caso, **en la memoria del proyecto se debe describir el método utilizado para calcular los snippets**.

¿Se pueden usar operadores de conjuntos (tipo `set` en python) en lugar de los algoritmos de unión e intersección de postings lists vistos en teoría?

NO, para unir e intersectar *postings lists* se deben utilizar los algoritmos vistos en teoría, el AND y el OR de dos *postings list*. No es por una cuestión de eficiencia, sino para practicar lo visto en la teoría de la asignatura.

¿Se puede tener un diccionario inverso para la versión de stemming?

Para la ampliación de *stemming* se sugieren estas 2 opciones:

- Tener un diccionario donde cada clave sea un *stem* y el valor asociado a esa clave sea la lista de términos con ese *stem*. Este diccionario se puede generar al final del indexado con el método `make_stemming`. Por ejemplo, la clave `'quij'` podría tener asociada como valor la lista `['quijada', 'quijote', 'quijotesco']`.

  La forma de usar este diccionario auxiliar es:

  - obtener el *stem* del término de la *query*.
  - la *postings list* del *stem* será la unión de las postings list de todos los términos con ese *stem*.

- Generar otro diccionario inverso donde los términos son *stems*. Esto ocupa más memoria que la opción anterior.

Si os habéis planteado alguna otra opción eficiente, podéis implementarla.
En cualquier caso, **en en la memoria del proyecto se debe describir el método de stemming útilizado**.

¿Qué pasa si el número de noticias no coincide con el de referencia?

En la mayoría de casos esto es debido a uno de estos dos factores:

- Una implementación errónea de los métodos de indexación o de recuperación de noticias.
- Una normalización y *tokenización* de las noticias errónea o distinta de la utilizada para el cálculo de la referencia.

¿Cómo combinar consultas posicionales y stopwords?

En este proyecto no se eliminan *stopwords*.

Además de la biblioteca `SAR_lib.py`, ¿se pueden añadir más ficheros que contengan funciones o clases ?

NO. Eso sí, podéis añadir a la clase `SAR_Project` de la biblioteca `SAR_lib.py` todas las variables o métodos que consideréis conveniente. Incluso se pueden añadir más clases o funciones pero siempre dentro de `SAR_lib.py`, sin modificar los programas de indexación y recuperación de noticias ni añadir más ficheros.

Si implementamos funcionalidades extra, ¿tienen que funcionar sólo si se pasan ciertos argumentos o tienen que funcionar siempre?

Para que se active el *stemming* se le debe indicar con el parámetro **-S** en el searcher. Las demás funcionalidades extra deben ser "implicitas".

Apéndices

A. Ejemplos de indexación

```
$ python SAR_Indexer.py corpora/2015 2015_index.bin

=======================================
Number of indexed days: 285
---------------------------------------
Number of indexed news: 803
---------------------------------------
TOKENS:
        # of tokens in 'article': 44684
---------------------------------------
Positional queries are NOT allowed.
=======================================
Time indexing: 0.68s.
Time saving: 0.09s.
```

```
$ python SAR_Indexer.py -S -P -M -O corpora/2015 2015_index_full.bin
=======================================
Number of indexed days: 285
---------------------------------------
Number of indexed news: 803
---------------------------------------
TOKENS:
        # of tokens in 'title': 3321
        # of tokens in 'date': 285
        # of tokens in 'keywords': 2266
        # of tokens in 'article': 44684
        # of tokens in 'summary': 6919
---------------------------------------
PERMUTERMS:
        # of permuterms in 'title': 26115
        # of permuterms in 'date': 3135
        # of permuterms in 'keywords': 18440
        # of permuterms in 'article': 407377
        # of permuterms in 'summary': 59173
---------------------------------------
STEMS:
        # of stems in 'title': 2642
        # of stems in 'date': 285
        # of stems in 'keywords': 1968
        # of stems in 'article': 23284
        # of stems in 'summary': 4738
---------------------------------------
Positional queries are allowed.
=======================================
Time indexing: 3.55s.
Time saving: 0.87s.
```

B. Ejemplos de recuperación

```
$ python SAR_Searcher.py -Q 'isla AND valencia AND pero' 2015_index.bin
========================================
Query: 'isla AND valencia AND pero'
Number of results: 2
#1       (0) (626) (2015-03-25) Diez motivos para querer vivir en Palma de
↪  Mallorca       (mallorca,palma,vivir,motivos)
#2       (0) (665) (2015-01-03) Las Fuerzas Armadas: un paso al frente y dos atrás en
↪  derechos sociales (frente,derechos,Fuerzas,Armadas,sociales)
========================================
```

```
$ python SAR_Searcher.py -N -Q 'isla AND valencia AND pero' 2015_index.bin
========================================
Query: 'isla AND valencia AND pero'
Number of results: 2
#1
Score: 0
626
Date: 2015-03-25
Title: Diez motivos para querer vivir en Palma de Mallorca
Keywords: mallorca,palma,vivir,motivos
exclusivas de la isla la cala de l oratori o ... claudia schiffer cuando tenía casa en
↪  la isla su fuerte ...
--------------------
#2
Score: 0
665
Date: 2015-01-03
Title: Las Fuerzas Armadas: un paso al frente y dos atrás en derechos sociales
Keywords: frente,derechos,Fuerzas,Armadas,sociales
mínima capacidad operativa pero sin cumplir con lo que dice ... los cuerpos comunes la
↪  diferencia es menor pero aún notable ...
========================================
```

```
$ python SAR_Searcher.py -C -L queries_2015_minimo.txt 2015_index.bin
de        803
isla        43
valencia        40
sanidad        34
cultura        66
videojuegos        3
videojuegos OR cultura        69
videojuegos OR NOT videojuegos        803
isla AND valencia        3
isla AND NOT valencia        40
NOT pero        292
isla AND NOT valencia AND NOT pero        8
NOT isla AND NOT valencia AND NOT pero        276
isla AND valencia AND pero        2
isla OR valencia        80
```

```
isla OR NOT valencia        766
NOT isla OR valencia        763
NOT isla OR NOT valencia        800
NOT isla OR NOT valencia AND pero        509
NOT isla OR NOT valencia AND NOT pero        291
NOT isla OR NOT valencia OR NOT pero        801
años        453
google        12
cultura        66
google OR cultura        76
años AND google        7
años AND cultura        43
```

```
$ python SAR_Searcher.py -S -T results_2015_minimo_stemming.txt 2015_index_full.bin
de        803
isla        43
valencia        40
sanidad        34
cultura        70
videojuegos        4
videojuegos OR cultura        74
videojuegos OR NOT videojuegos        803
isla AND valencia        3
isla AND NOT valencia        40
NOT pero        291
isla AND NOT valencia AND NOT pero        8
NOT isla AND NOT valencia AND NOT pero        275
isla AND valencia AND pero        2
isla OR valencia        80
isla OR NOT valencia        766
NOT isla OR valencia        763
NOT isla OR NOT valencia        800
NOT isla OR NOT valencia AND pero        510
NOT isla OR NOT valencia AND NOT pero        290
NOT isla OR NOT valencia OR NOT pero        801

Parece que todo ha ido bien, buen trabajo!
```