PROYECTO SAR

Buscador web

Luis José Ferrer Estellés
Diana Bachynska
David Oltra Sanz
Ricardo Díaz Peris

WIKIPEDIA
La enciclopedia libre

Índice

Contribución al proyecto de cada miembro

Luis José Ferrer Estellés:

-index_dir
-index_file
-make_permuterm
-make_stemming
-show_stats

Diana Bachynska:

-and_posting
-or_posting
-minus_posting
-make_permuterm
-make_stemming
-reverse_posting

Ricardo Díaz Peris:

-parse_wikipedia_textual_content
-solve_query
-get_field
-get_posting
-get_stemming
-get_permuterm
-solve_and_show

David Oltra Sanz:

-start_crawling
-solve_query
-get_field
-get_posting
-get_stemming
-get_permuterm
-solve_and_show

Funcionalidades extra

Multifield

Para la implementación de multifield hemos modificado en primer lugar el método index_file, en el caso de tener que aplicar multifield se crea un diccionario dentro del diccionario self.index con cada atributo de self.fields y dentro de cada uno se realizará tal cual estaba en la opción básica exceptuando el atributo url que no se tokenizara. Para luego comprobar si en la búsqueda hace falta buscar con multifield se especifica posteriormente en este documento en *get_posting*

Stemming

Para la implementación de stemming hemos completado los métodos make stemming y get stemming.

make_stemming

En primer lugar, en el make stemming hemos diferenciado entre sí hay multifield o no, si hay multifield creamos diccionarios dentro del diccionario como hemos realizado en el self.index.A continuación, dentro de cada diccionario si hay multifield o directamente en self.sindex si no hay multifield creamos una posting list con todos los términos(sin repetir término) en los que aparece el stem de cada token de self.index.

get_stemming

Para conseguir las posting lists de un elemento con stemming, lo primero que debemos hacer es calcular el stem del término que queremos conseguir las posting lists, después comprobamos si se ha pasado un field como argumento, en ese caso haríamos una búsqueda multifield, sino normal. La búsqueda consiste en primero conseguir los términos que corresponden a ese stem (guardadas en el diccionario de stemming) y la función para luego conseguir sacar las posting lists. Una vez conseguidos ambos basta con recorrer los términos (si hay alguno) y conseguir las posting lists de estos términos y juntarlas todas sin repetir.

Permuterm

Para la implementación de permuterm hemos completado los métodos make permuterm y get permuterm.

make_permuterm

En primer lugar, en el make permuterm si es multifield creamos un índice permuterm para cada campo, iteramos sobre los tokens de cada campo,creamos todos los posibles permuterms de un token,si el índice permuterm no tiene el permuterm, la creamos,si el token no está en el índice permuterm, lo añadimos. Si no hay

multifield, realizamos el mismo procedimiento pero solo para el campo "all” por lo que no es necesario crear un diccionario de diccionarios.

get_permuterm

Para conseguir una posting list de un término con comodín por permuterm ya no es tan sencillo como en el stemming, primero comprobamos si el término tiene ‘*’ (comodín de cualquier longitud) o ‘?’ (comodín de longitud 1) para luego tenerlo en cuenta. Una vez visto volvemos a comprobar si hay algún comodín (por si acaso) y construimos su permuterm cambiando el orden de cada lado del comodín incluyendo un ‘$’ en medio para indicar el fin de la palabra. Una vez construido el permuterm comprobamos si se ha pasado algún campo para tener en cuenta si hay que hacer búsqueda multifield o no, una vez comprobado cogemos todas las claves del diccionario de permuterm. Clave por clave vamos comprobando si empieza por nuestro permuterm y si cumple con la longitud indicada al principio del párrafo, cualquier clave que cumpla ambas y su término no esté en la lista de términos será añadido. Si se ha añadido algún término se recorren todos y se cogen sus posting lists.

Descripción y justificación de las decisiones de implementación realizadas

parse_wikipedia_textual_content

Si hace match con la expresión regular de título y resumen, ejecuta el resto del método y lo primero que hace es guardar la url pasada como argumento, el título y el resumen. A partir de aquí se sigue el mismo algoritmo, calculas los matches que haga con la expresión regular de las secciones y miras cuando empieza cada sección con el span, te guardas cada comienzo en una lista y esa es la forma de dividir el texto de las secciones, con la expresión regular correspondiente consigues separar para guardar en el diccionario lo necesario, luego con las subsecciones haces lo mismo. Al final se te queda el diccionario con la url, el título, el resumen, la lista de diccionarios de secciones y la lista de subsecciones para cada sección.

start_crawling

Mientras haya elementos en la cola y no se sobrepase el límite, se extrae de la cola, implementada con un heap, la siguiente url. Si es válida y no excedemos el límite de profundidad, se obtiene el contenido de página de la wikipedia a la que se accede con la url y los enlaces a otras páginas que se obtienen de ésta. Como los enlaces son relativos, tenemos que convertirlos en enlaces completos, por lo que hemos diseñado el método asegurar_url_absoluta que completa el enlace. Estos enlaces si son válidos se insertan en la cola.

El contenido se estructura en diccionarios con el método parse_wikipedia_textual_content y se añade al documento. Si se introduce una batch_size entonces los documentos se guardarán en archivos según la cantidad que hayamos introducido, y si no se introduce entonces todos los documentos se guardan en el mismo archivo.

index_dir

En primer lugar, si está activado el uso de stemming llamamos a make_stemming para hacer el índice self.sindex. A su vez, si está activado el uso de permuterm llamamos a make_permuterm para rellenar self.ptindex

index_file

En primer lugar, comprobamos si self.docs esta vacío , si lo esta lo inicializamos con id=1 y si no está vacío le ponemos como id la cantidad de documentos más 1. A continuación, si hay multifield creamos un índice para cada sección del artículo si no existe ya. Ya dentro del for, después de llamar a (j = self.parse_article(line))

comprobamos si el articulo ya esta indexado para no indexarlo de nuevo, seguidamente, sacamos el id para la clave artículo y guardamos en su valor una tupla de docid y la posición del artículo en el fichero. Por último, es multifield iteramos sobre field para ver qué campos tenemos que tokenizar, dentro de los que hay que tokenizar iteramos sobre los tokens y los guardamos en el índice con una posting list de los artículos en los que aparece, sino si no es multifield, tokenizamos el texto de j[all] iteramos sobre los tokens y los guardamos en el índice con una posting list de los artículos en los que aparece.

show_stats

En este método, si es multifield mostramos el número de tokens en cada campo, si es multifield y está activado el uso de stemming mostramos el número de stems en cada campo sino está activado el multifield y si el uso de stemming mostramos el número de stems de self.sindex, si es multifield y permuterm mostramos el número de permute rms en cada campo y sino está multifield y si permuterm si está activado el uso de permuterm mostramos el número de permuterms de self.ptindex.

solve_query

Lo primero que haces es comprobar si tiene *, ? o : para utilizar una expresión regular distinta a la que utilizaríamos normalmente para poder separar en términos con sus fields y sus comodines. Después comienza el método recursivo, una vez tokenizas si es de distancia 1 devuelves la posting list con el método ***get_posting***, si es de más de uno suponemos que el penúltimo término es un operador, descubres qué operador es y vuelves a llamar al solve_query teniendo en cuenta el tipo de operador para utilizar los métodos correspondientes.

get_field

Para la realización de este método, primero separamos el token del campo,si no hay campo, devolvemos el token y el campo por defecto,si hay campo, devolvemos el token y el campo.

get_posting

Primero miras si tiene algún field, después, si no lo tiene pero sí que existe un diccionario multifield, igualas el field al que es por defecto, después llamas al método correspondiente para conseguir la posting list

reverse_posting

Coges todos los artículos y utilizas el minus para restarles los que pasas como parámetro

and_posting

Calcula la intersección (AND) de dos listas de postings (`p1` y `p2`). La función empieza comprobando si alguna de las listas está vacía, si es así, devuelve una lista vacía inmediatamente, ya que la intersección con una lista vacía es siempre vacía.

Luego, utiliza dos índices (`i1` y `i2`) para recorrer ambas listas de manera sincronizada. En cada iteración, compara los elementos actuales de `p1` y `p2`. Si son iguales, añade el elemento a la lista de resultados y avanza ambos índices. Si no son iguales, avanza solo el índice del menor elemento. Este proceso continúa hasta que uno de los índices alcance el final de su lista.

or_posting

Calcula la unión (OR) de dos listas de postings (`p1` y `p2`). Similar al método `and_posting`, comienza verificando si alguna de las listas está vacía, devolviendo la lista no vacía si es el caso. Luego, usa dos índices para recorrer ambas listas de manera sincronizada. Si los elementos actuales de `p1` y `p2` son iguales, añade el elemento a la lista de resultados y avanza ambos índices. Si son diferentes, añade el menor elemento a la lista de resultados y avanza el índice correspondiente. Una vez que se ha recorrido una de las listas por completo, añade los elementos restantes de la otra lista a la lista de resultados.

minus_posting

Calcula la diferencia (EXCEPT) de dos listas de postings (`p1` y `p2`), devolviendo los elementos que están en `p1` pero no en `p2`. Si los elementos actuales de `p1` y `p2` son iguales, avanza ambos índices sin añadir nada a la lista de resultados. Si los elementos son diferentes, añade el elemento actual de `p1` a la lista de resultados y avanza el índice de `p1`. Este proceso se repite hasta que se ha recorrido una de las listas por completo. Finalmente, añade cualquier elemento restante de `p1` a la lista de resultados.

Solve_and_show

Para la realización de este método,para cada artículo en la posting list obtenemos el docid y la línea, obtenemos el documento, abrimos el documento, parseamos el artículo y mostramos el título y la url.

Método de coordinación

En cuanto a la coordinación a la hora de realizar el proyecto, nosotros nos hemos decantado por dividirnos en dos grupos de dos, uno más encargado de la indexación y otro más encargado de la recuperación. A su vez, nos hemos repartido los métodos a realizar y si alguno se quedaba más atascado con algún método recibia ayuda de otro integrante del grupo.