Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

Recuperación de Información

Primera Tarea Programada

Profesor
Diego Villalba

Estudiante / Carnet
Ivannia Alvarado / B10273
Oscar Castro / B11616
Jenny Vásquez / B17016

Octubre, 2015

Tabla de Contenidos

1. Introducción

Los motores de búsqueda actualmente forman parte indispensable de nuestras vidas y de ahí que su estudio en un curso de Ciencias de la Computación sea un tema de suma importancia. Es así como este proyecto se trata de crear un motor de búsqueda básico que permita alcanzar el objetivo de conocer más a fondo cómo funciona este tipo de aplicación.

El enunciado del proyecto quedó bastante a libertad de cada quien en cuanto a la elección de herramientas de trabajo, por lo que en este caso se utilizó Scrapy como araña para la recolección de documentos de la web, Stanford Tokenizer para la tokenización y el parseo de los mismos y Java (NETBEANS) para el código de la configuración y de la aplicación.

2. Análisis del Problema

Se decidió abordar el problema de una forma modular, lo que implica la división del mismo en distintas partes que se comunican y complementan, pero actúan de forma autónoma. De esta forma, se tienen cuatro partes identificadas en la aplicación:

1. *Obtención de la colección:* esta es la parte que se encarga de conseguir la colección de documentos mediante la araña, además de organizarlos en un directorio y hacer la debida conversión al formato soportado por la aplicación.

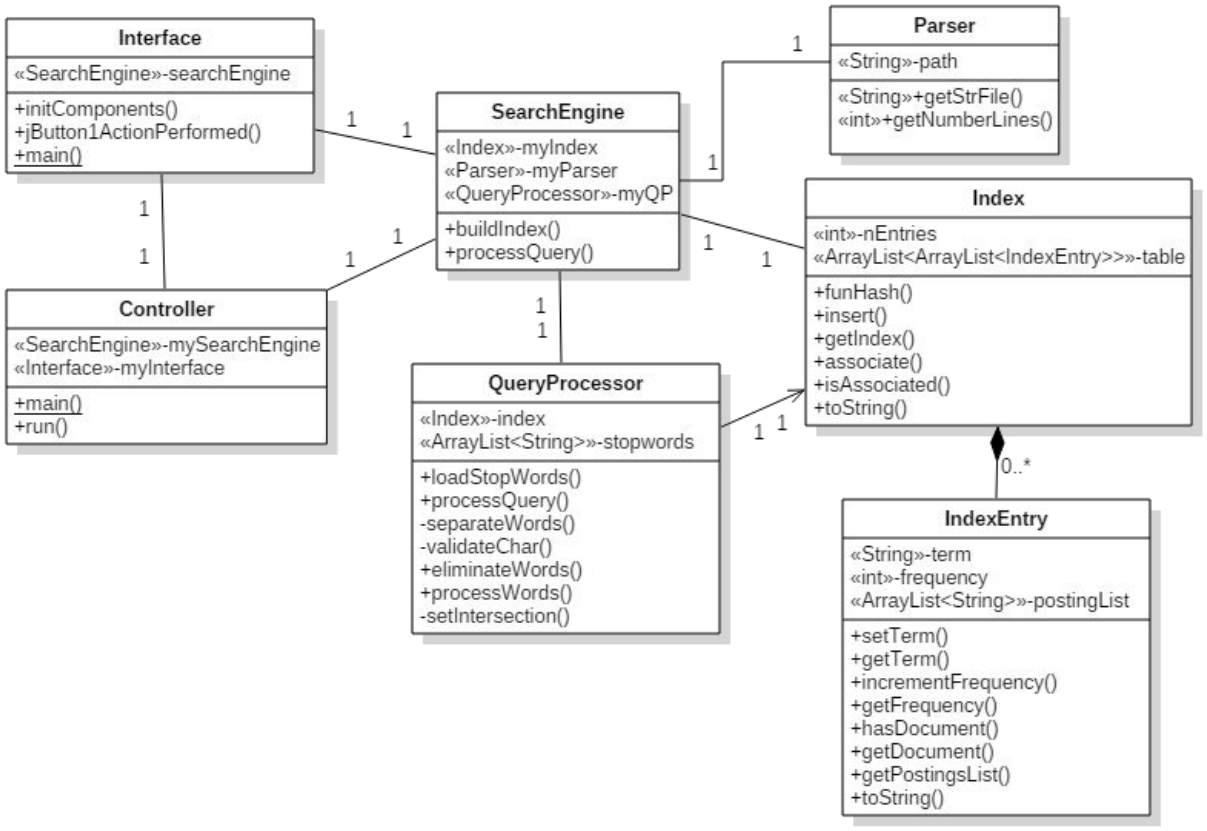
2. *Procesamiento de archivos, obtención de tokens y obtención de la lista de postings:* esta es la parte que se encarga de tomar el directorio con la información de las páginas obtener los tokens de cada documento y procesarlos hasta obtener la lista de postings. Este será el que se guarde en un archivo en disco, pues sirve de base para construir el índice de búsqueda en la parte siguiente.

3. *Construcción del Índice de búsqueda:* en esta parte se toma el archivo guardado en disco con la lista de postings en formato .txt y se construye el índice de búsqueda en la aplicación. El algoritmo utilizado para la construcción del índice es BSBI y la estructura de datos es una Tabla Hash (Tabla de Dispersión).

4. *Procesamiento de consultas:* en esta parte, se hace el procesamiento de la consulta ingresada por el usuario y se busca mediante el índice de los archivos que satisfagan la misma. También esta parte es la encargada de procesar la respuesta y mostrarla al usuario.

3. Diseño

La siguiente imagen muestra el diseño de clases de la aplicación y la relación entre las mismas.



4. Experimentos

Araña

Experimento 1: Consiste en realizar una araña con Nutch y una con Scrapy para comparar las diferencias de utilizar cada una.

Pre-procesamiento lingüístico

Experimento 2: Consiste en comparar la tokenización y normalización de las páginas sin quitar las *stopwords* y luego quitándolas.

Desarrollo y construcción de índices

Experimento 3: Consiste en comparar la utilización de dos versiones de función Hash distinta para la construcción del índice invertido. La primera función Hash se basa en sumar los códigos *ASCII* de los caracteres de un Token y aplicar posteriormente a dicha suma, el módulo de la cantidad de entradas del índice.

La segunda función hash consiste en dividir el token en partes de 4 bytes e interpretar cada una de esas partes como un valor *long int*. Los valores de todas las partes son sumados y posteriormente, mapeados con módulo de la cantidad de entradas del índice.

Procesamiento de consultas

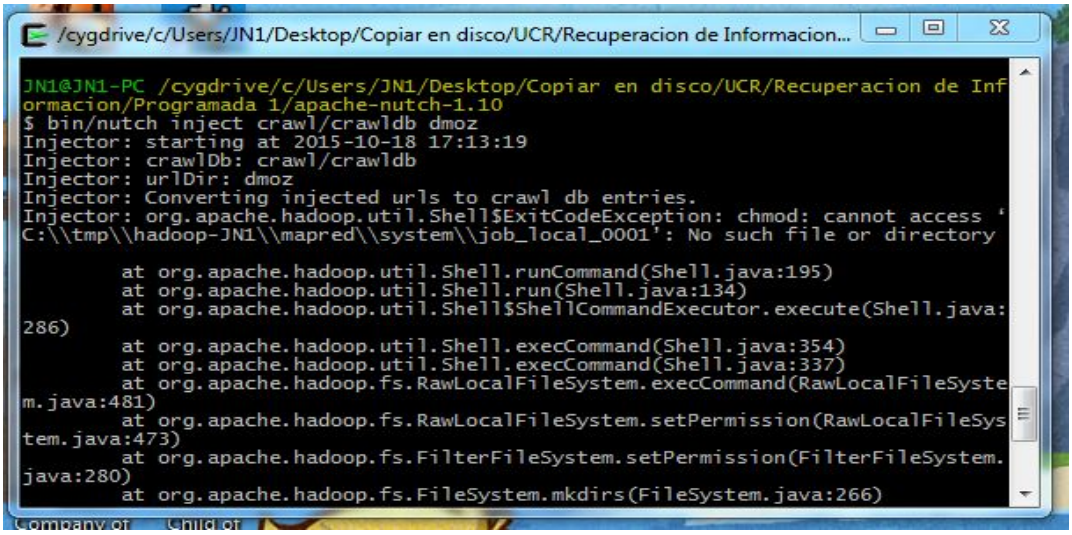
Experimento 4: Consiste en comparar los resultados generados al ingresar una consulta y la misma pero agregándole *stopwords* y algunos términos que no estén dentro del índice. El punto es demostrar que el *boolean retrieval* está funcionando correctamente y también se toma en cuenta el descarte de *stopwords*.

5. Resultados de los experimentos

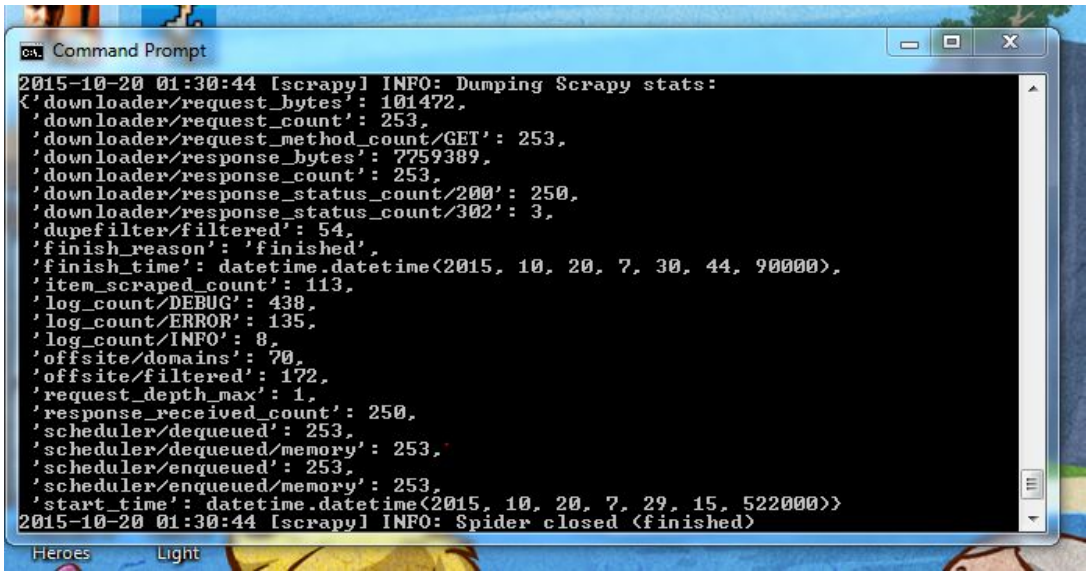
Araña

Experimento 1: *Araña con Nutch Vs Araña con Scrapy.*

No se pudo realizar la araña con Nutch ya que, aún siguiendo los pasos del tutorial de ellos. Se presentan demasiados errores, los cuales no se pueden resolver.



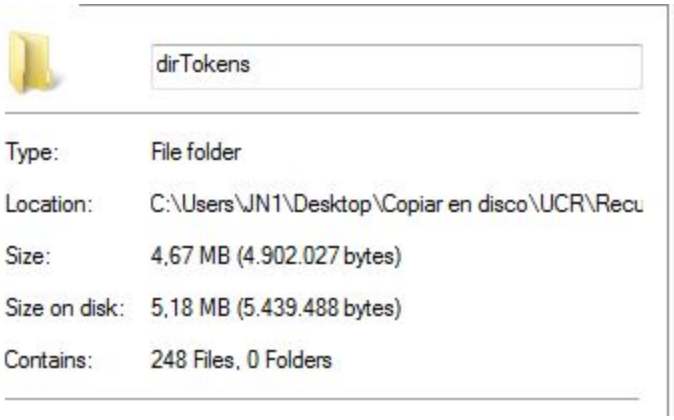
La araña se creó correctamente en Scrapy y se consiguieron 248 páginas.



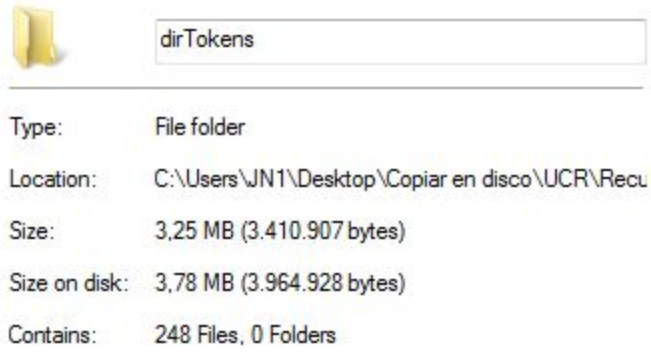
Pre-procesamiento lingüístico

Experimento 2: *Tokenización y normalización con stopwords vs sin stopwords.*

Después de usar el Stanford Tokenizer para tokenizar la información de las páginas y normalizar (aún con las *stopwords* entre los tokens) lo que devuelve el tokenizador se obtuvo un directorio de tokens que pesa unos 4.67 MB.



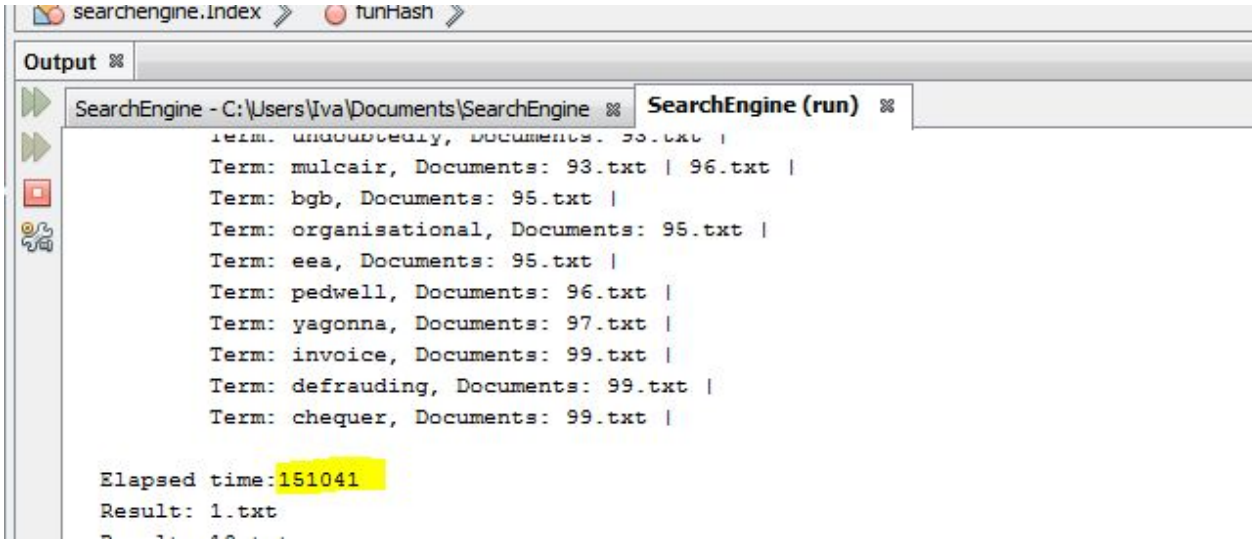
Después de usar el *Stanford Tokenizer* para tokenizar la información de las páginas y normalizar (quitando las *stopwords* de los tokens) lo que devuelve el tokenizador, se obtuvo un directorio de tokens que pesa cerca de 3.25 MB.



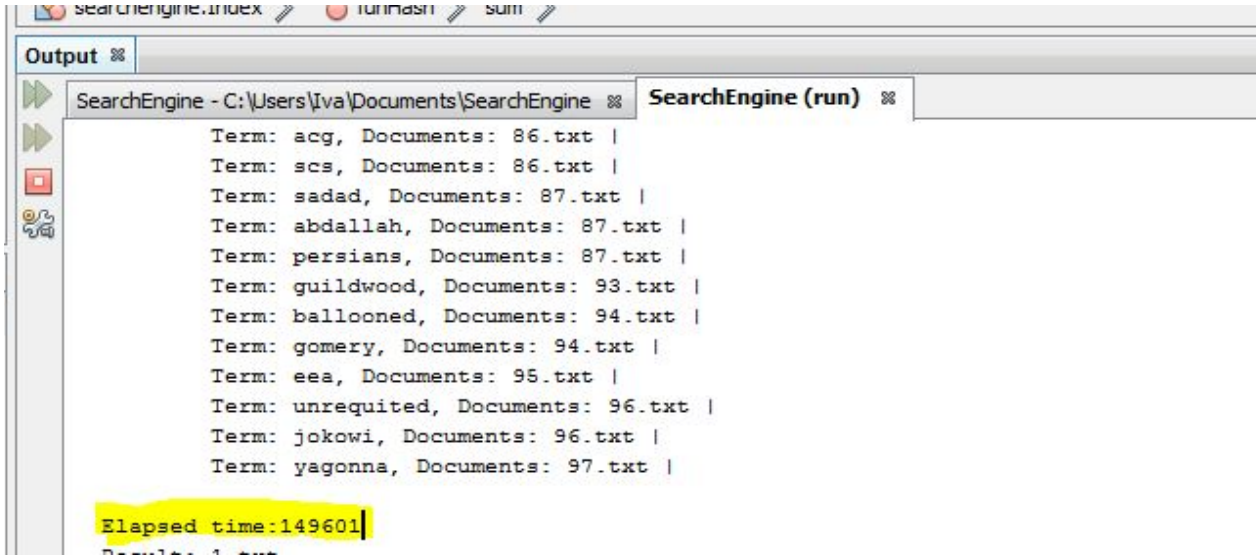
Desarrollo y construcción de índices

Experimento 3: *Hash de suma de códigos ASCII Vs hash de interpretación de 4 bytes.*

Ambas funciones se probaron utilizando el archivo postings.txt como entrada para la tabla hash e inicializando con 150 índices la tabla hash. De modo que con la primera función, la construcción del índice tomó un tiempo de duración aproximado de: 151041ms.

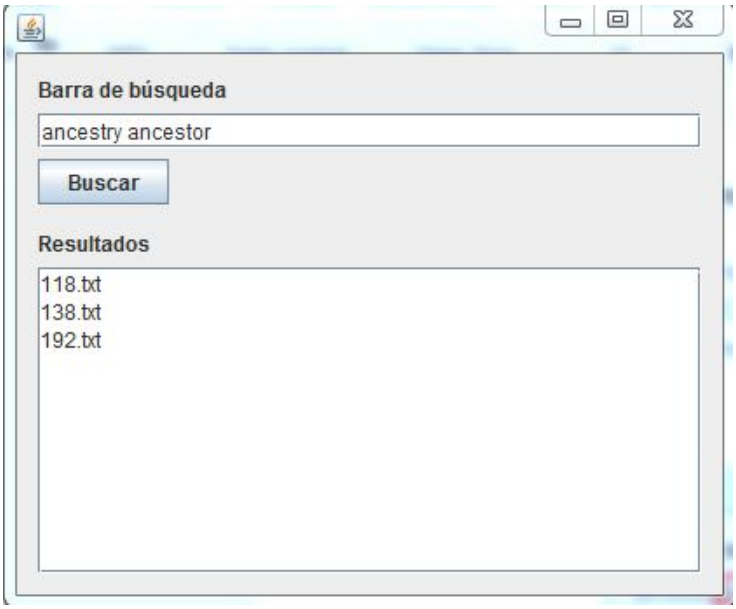


Mientras que con la segunda función, el tiempo fue de: 149601ms.

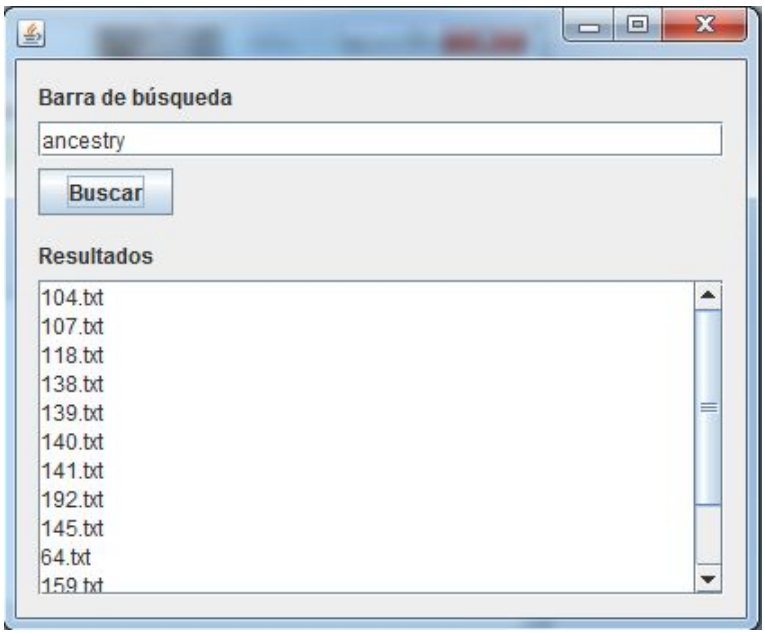


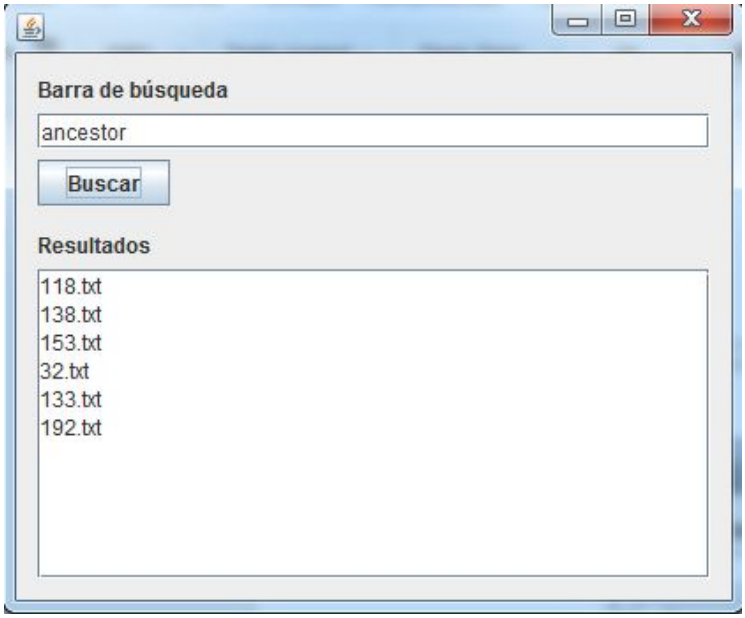
Procesamiento de consultas

Experimento 4: La consulta original a probar es *ancestry ancestor*, esta tiene como resultado el siguiente. Correspondiente a la intersección entre las listas de postings generadas para cada uno de los términos en el índice.

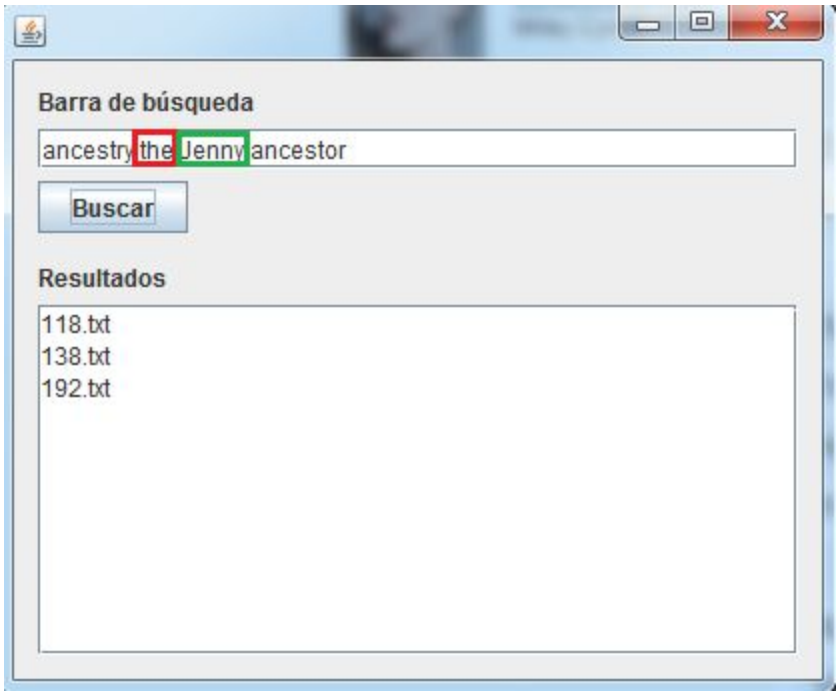


A continuación se muestra cada uno de los términos por separado para mostrar que la intersección mencionada funciona correctamente.





La consulta modificada corresponde a *ancestry the Jenny ancestor*, esta tiene como resultado el siguiente. Correspondiente a la intersección entre las listas de *postings* generadas para cada uno de los términos en el índice. A diferencia de la anterior, *the* es un *stopword* por lo tanto debe ser ignorada y *Jenny* no se encuentra en el índice por ello, no se toma en cuenta a la hora de calcular la intersección entre las listas de postings de términos resultantes.



6. Documentación de proceso

Fase 1: Se crea una araña en Scrapy que devuelve todo el código *html* de la página que encontró y guarda esto en archivos con formato: ***numeroarchivo.txt***, en el directorio ***paginas***. Luego éstas páginas pasan por el proyecto ***arreglarPaginas*** que quita las etiquetas *html* del archivo para queden en formato de texto plano y posteriormente puedan ser usados por la aplicación.

Fase 2: Luego de arreglar las páginas se corre el PTBTokenizer de la Stanford University para obtener los tokens de los archivos de las páginas (los archivos con los tokens de las páginas se guardan en el directorio ***dirTokens***) y pasan por el proyecto ***NormalizarTokens***, para quitar caracteres especiales, números o *stopwords* que no se van a indexar.

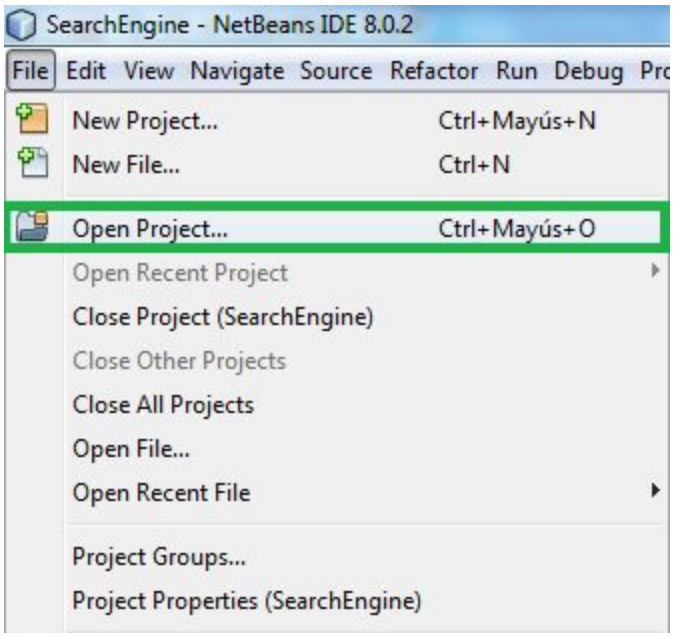
Una vez que se ha obtenido el directorio ***dirTokens***, el cual contiene todos los *tokens* de los documentos obtenidos por la araña, en formato txt, y organizado de forma que cada archivo hace referencia a los tokens del archivo original obtenido por la araña, se procede a trabajar en la generación del diccionario y el índice de búsqueda.

Lo primero que se hace es, con base en ***dirTokens***, construir un archivo llamado ***postings.txt***, el cual se guarda en disco, pues es la base para construir el índice cada vez que se inicie la aplicación del motor de búsqueda.

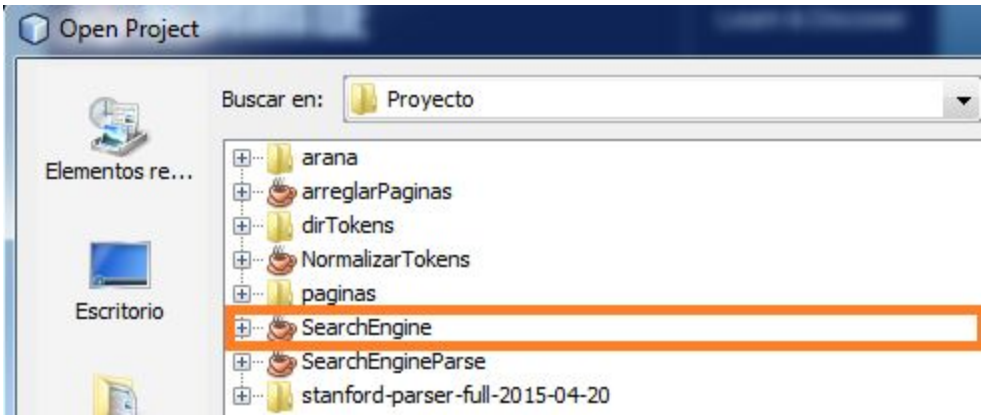
Es importante destacar que todos los pasos anteriores a la creación del archivo *postings.txt* solo se van a ejecutar para efectos de configuración del ambiente y no forman parte de la aplicación del motor de búsqueda en sí. Es decir, no se va a dar todo el procesamiento de archivos en cada corrida del programa.

Seguidamente, una vez que tenemos lista esta configuración y que ejecutamos el programa, éste automáticamente irá a buscar el archivo *postings.txt* para construir el índice invertido de la aplicación y cargarlo en memoria. Una vez con esto listo, el programa está preparado para recibir y responder consultas.

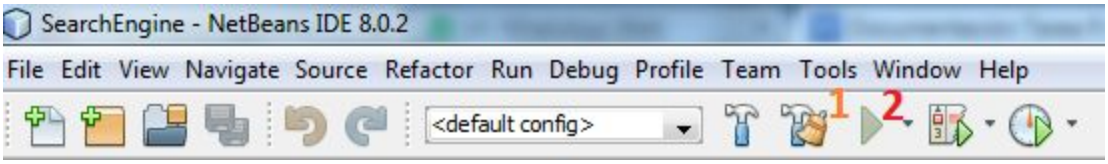
Fase 3: lo primero a realizar es abrir y ejecutar la aplicación desde el IDE. Para ello como mencionamos anteriormente, se necesitará de NETBEANS. Dentro de la aplicación procedemos a cargar la aplicación.



Luego de seleccionar la opción de abrir un proyecto, elegimos de la carpeta del proyecto entregado, el que se muestra a continuación.

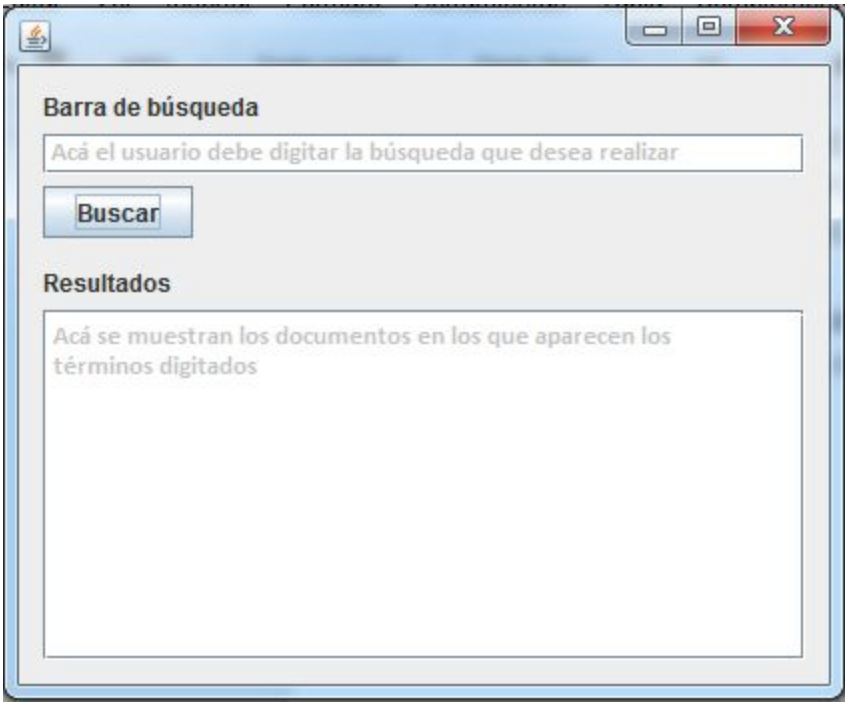


En esta parte se maneja la aplicación que será la encargada de procesar todo lo anteriormente creado y de obtener la consulta de parte del usuario para posteriormente ofrecerle una respuesta.



La imagen anterior apunta con el número 1 el ícono con el que puede ser compilado el proyecto, y con el número 2 el ícono que permite correr el proyecto. Al ejecutar el proyecto, como se mencionó anteriormente, se creará el índice generado por los términos y sus listas de *postings*. Esto tardará unos cuantos minutos, así que se solicita paciencia.

Luego de que cargue el índice, se mostrará la ventana donde el usuario podrá ingresar cuantas consultas desee, ésta se muestra a continuación.

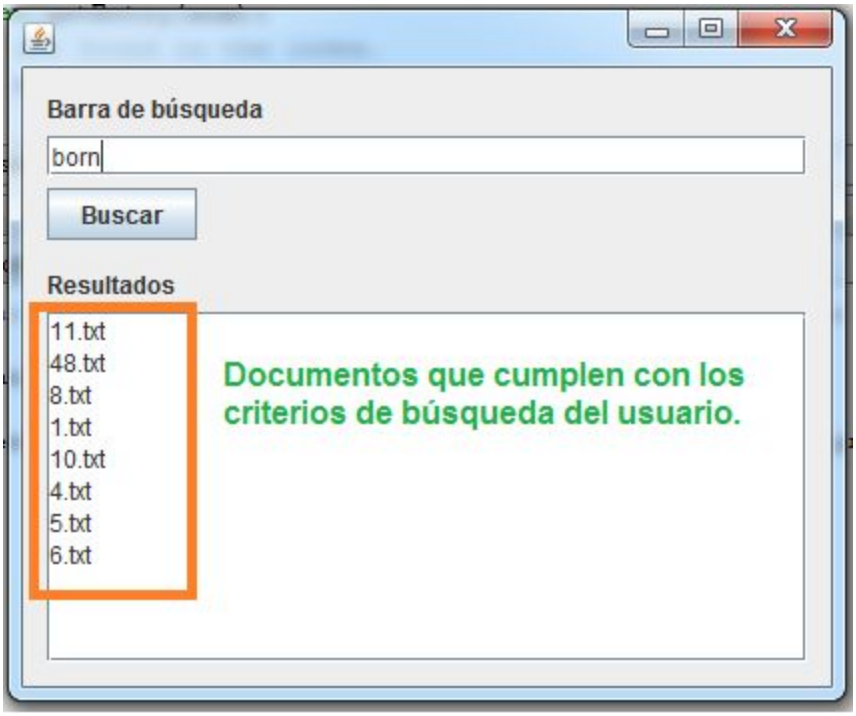


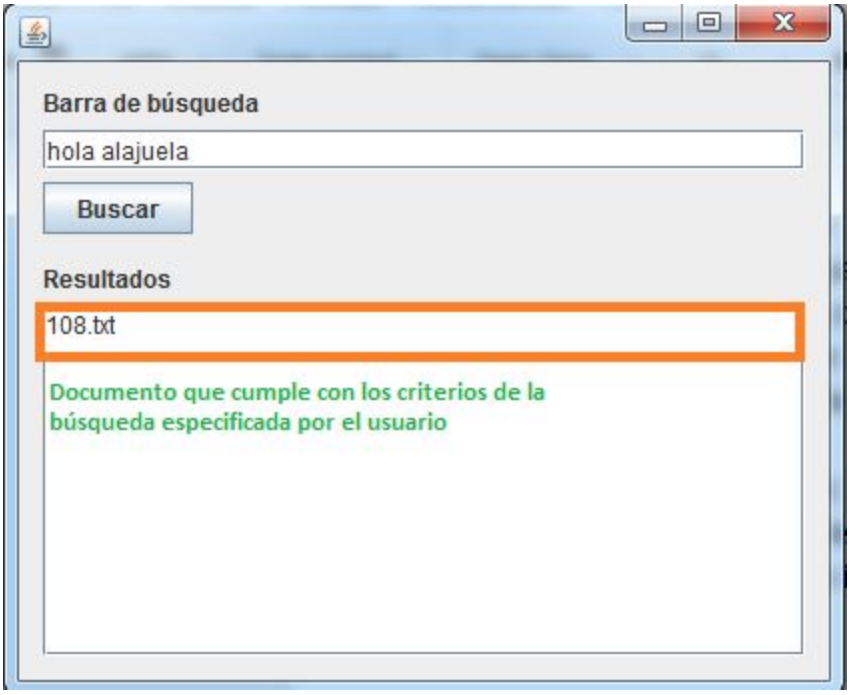
Luego de que se digita la búsqueda deseada, se procede a hacer clic sobre el botón de *Buscar*, al hacerlo el programa procede a hacer un análisis sobre la consulta. Lo primero a realizarse es a la eliminación de caracteres que no sean alfanuméricos de la consulta y a identificar las palabras que la componen. De estas palabras, se procede a eliminar las que corresponden a *stopwords*. Luego de que se limpia la consulta se procede a calcular las listas de *postings* para cada uno de los términos resultantes. El criterio para procesar la búsqueda resulta simple, se

buscan todos los documentos en que están cada una de las palabras que componen la consulta, sin tomar en cuenta aquellas que se consideran stopwords y aquellas que no se encuentran dentro del índice; es decir, las que no se encuentran en ningún documento del dominio de archivos que se procesaron para crear el índice.

Es claro, que lo que se realiza es una intersección de los conjuntos de documentos en que aparecen las palabras que componen la búsqueda; tomando en cuenta que existen conjuntos vacíos para las que no se encuentran en el índice, por esto debió tomarse un manejo especial, para que no se produjeran resultados vacíos cuando alguno de los términos ni siquiera estaba en el índice.

Luego de dar clic sobre el botón *Buscar*, y del procesamiento explicado con anterioridad se procederá a mostrar la lista de archivos resultantes a la lista. Las imágenes posteriores nos lo ilustran.




Barra de búsqueda
hola alajuela
Buscar
Resultados
108.txt
Documento que cumple con los criterios de la
búsqueda especificada por el usuario

7. Pasos para replicar el proceso

1. Instalar *python* y *pywin* (la arquitectura de pywin debe ser la misma que la de python).
2. Agregar *python* y *python\Scripts* a la variable de entorno ***path.***
3. En la consola de comandos, correr el comando ***pip install Scrapy.***
4. En el directorio ***arana***, correr el archivo ***correrArana.bat*** para que corra la araña que se realizó en Scrappy.
5. Correr el ejecutable del proyecto ***arreglarPaginas.***
6. Correr el archivo ***tokenizar.bat***
7. Correr el ejecutable del proyecto ***NormalizarTokens.***
8. Correr el ejecutable del proyecto ***SearchEngineParse*** (dura aproximadamente 30min).
9. Correr el ejecutable del proyecto ***SearchEngine*** (verificar antes que se ejecutó bien el paso anterior y se generó el archivo postings.txt, además, tomar en cuenta que el programa dura arrancando aproximadamente 10min).
10. Probar el buscador.

8. Problemas sin resolver

Esta aplicación, por su naturaleza, requiere de gran cantidad de operaciones que involucran procesamiento de archivos guardados en disco, lo cual, debido a las limitaciones que se poseen en cuanto a infraestructura y paralelismo, vuelve el proceso de ejecución del programa una tarea sumamente lenta. Es por esta razón que la creación del archivo base para el índice dura aproximadamente 30min, mientras que la carga del índice con los datos de dicho archivo dura aproximadamente 10min.

Como sabemos que el trabajo debe ser revisado y que posiblemente, quien realiza la revisión no posee tanto tiempo libre, decidimos recortar un poco la cantidad de archivos que se van a procesar, con el fin de poder simular el mismo proceso a una mayor velocidad. Es por esto que, de un directorio original con 248 archivos, vamos a tomar solamente 50 para la prueba de la aplicación. También se adjuntará el directorio original con todos los archivos, por si se desea hacer la prueba completa.

Además, cabe destacar que se buscaron todas las formas posibles de mejorar la eficiencia del código, pero por tratarse de algo tan pesado, hecho en Java y con un solo hilo de ejecución, fue imposible mejorar la velocidad, a un tiempo decente. Lo cual creemos que es algo justificable, pues para un motor de búsqueda real, se utilizan muchos más recursos.