Informe Escrito

Proyecto Moogle!

Por: Adrián Hernández Castellanos Grupo: 112

Estudiante de Ciencias de la Computación de la Universidad de La Habana

Temáticas a tratar:

- Estructura del proyecto
- Procesamiento del texto
- Transformación de la búsqueda (query)
- Trabajo con operadores
- Recomendaciones
- Emisión de las respuestas

Moogle! es un sistema de búsqueda que contiene una o un conjunto de palabras y devuelve los documentos relacionados a estas mediante una interfaz gráfica (sencilla).

La clave para el funcionamiento del sistema se encuentra en la optimización y la implementación de métodos específicos y clases que nos permitirán realizar la comparación de una palabra o conjunto de ellas contra una lista que contiene todos los vocablos del texto.

Estructura del proyecto:

El sistema se encuentra dividido en varios subsistemas, cada cual con una función en particular, de forma tal que concatenando las respuestas de estas funciones con las entradas de las siguientes se consigue transformar la información obtenida previamente, eliminar los datos innecesarios y transformar los datos importantes al tipo de dato correspondiente para dar una respuesta válida.

Procesamiento del texto:

Para esta sección nos vamos a dirigir al apartado TextProcessing, que se encuentra en la carpeta MoogleEngine dentro de la carpeta raíz del proyecto. Aquí se realizará el análisis del directorio donde se contienen los archivos, para posteriormente obtener la información necesaria (Título y texto de los documentos). Para ello se ha implementado un tipo de dato File que es el encargado de contener dicha información. Posteriormente se construye una lista de todos los archivos existentes dentro del directorio, y a partir del texto de cada uno de estos archivos se crea una lista de palabras, que contiene a todas las palabras que se encuentran en los archivos del directorio. Para la creación de esta lista de palabras se ha destinado un nuevo tipo de dato (Word) que posee al string el cual conforma la palabra, así como el conjunto de *score que posee la palabra en cada archivo de la lista de archivos.

*score: representa un valor específico para cada palabra dentro de un archivo, determinando la importancia que toma esta en un conjunto de documentos para cada uno en concreto. Para ello se implementa el cálculo Term Frecuency * Inverse Document Frequency.

Transformación de la búsqueda:

Teniendo ya una lista de archivos y otra de todas las palabras que aparecen en ellos con sus respectivos score (sin repetir ninguna de ellas), solo nos queda comparar cada una de las palabras de la búsqueda con las que aparecen en la lista de palabras. En el caso de que no aparezca ninguna, el buscador debe devolver una respuesta indicando este hecho, y una recomendación de palabras parecidas que sí aparecen en la lista (de esto se hablará más adelante).

Obtenidas ya las palabras que están en la lista a través de un método designado para eliminar signos de puntuación, con sus respectivos score, debemos analizar cada documento en los que aparece (eliminando así los que presentan score = 0 de la lista de posibles respuestas) y organizar los documentos por orden de prioridad a partir del score, asegurando que ninguno de ellos se repita dos veces. En el caso en que un mismo documento contenga dos o más palabras de la búsqueda, ambos score se suman y el documento pasa a contener el snippet de la palabra más significativa.

Para lograr este propósito diseñamos un nuevo tipo de dato (LINE) el cual constituye una línea de datos que contiene toda la información necesaria para devolver la respuesta a la búsqueda (palabra específica que se busca/documento en el que aparece/score de la palabra en el documento/ fragmento del texto en el que aparece la palabra en dicho documento).

La obtención de este fragmento de texto (snippet) se realiza analizando el correspondiente al documento y revisando la posición aproximada que toma la palabra objetivo en este. Teniendo en cuenta la cantidad de palabras del documento y la posición de la palabra relevante en este contexto, se determina si el snippet contiene a todo el texto del documento, a las primeras palabras del documento, a las últimas palabras del documento o a un texto específico del documento.

Trabajo con operadores:

Existen 4 operadores específicos que modifican la búsqueda, ellos son:

- ^ (indica que la palabra a la que precede debe encontrarse en cualquier documento mostrado, de forma tal que deben eliminarse de las líneas de respuesta los documentos que no la contengan)
- ! (indica que la palabra a la que precede no puede aparecer en ningún documento mostrado, por tanto deben eliminarse de las líneas de respuesta los documentos que la contengan)
- * (indica que la palabra a la que precede es más importante que las demás, por lo cual debe aumentar exponencialmente su score en cada documento que la contenga; además, mientras más operadores “*” posea, mayor es su importancia)
- ~ (este operador debe aparecer entre dos palabras; indica que mientras más cerca estén ambas palabras, mayor es la importancia del documento que las contiene)

Para el trabajo con operadores, se implementó la clase Operators, que contiene todos los métodos necesarios para crear listas de palabras y realizar la ejecución del proceso correspondiente en las respuestas. Para ello se crean 4 listas de palabras que contienen operadores, cada lista para un operador específico, y antes de ofrecer la respuesta, se modifica la lista de respuestas a partir de los requisitos de cada operador.

Recomendaciones:

En el caso en que una palabra no aparezca en la lista, debemos realizar una recomendación de alguna de ellas que se encuentre ya contenida en la lista. Para ello se ha implementado una clase (Recomendations) en la cual se encuentran métodos que nos permitirán recomendar la palabra más parecida a la que se quiere encontrar, para posteriormente enviarla a una lista de recomendaciones que se devolverá en el método de respuesta. Para lograr recomendar la palabra más parecida, se ha implementado el cálculo de la Distancia de Levenshtein, que representa la cantidad de modificaciones necesarias que debe hacerse en los caracteres de una palabra para que se convierta en otra. La palabra que, en comparación con la objetivo, tenga menor distancia, es la que menos modificaciones requiere para convertirse en ella, y por tanto la más parecida.

Emisión de respuestas:

Finalmente, teniendo las líneas de respuestas ya modificadas por los operadores y una lista de recomendaciones que ofrecer, solo queda convertir de líneas de respuestas a tipo SearchItem, ordenar por relevancia a partir del score de mayor a menor y devolver.