Proyecto de Programación I. Moogle!

Facultad de Matemática y Computación.
Universidad de La Habana. Curso 2022.

Nombre: Marian S. Álvarez Suri.
Grupo: 121.

El Proyecto Moogle! consiste en un sistema de recuperación de información centrado en la búsqueda de cadenas de texto dentro de un conjunto de archivos de texto plano.
La idea central del proyecto ha sido facilitar al usuario a la hora de realizar la búsqueda. Es por ello que cuenta con varias implementaciones que permiten ser más específicos en lo que se desea encontrar, mostrando una amplia gama de resultados y de alternativas.
La interfaz gráfica ha sido programada en su mayoría con HTML y CSS, mientras que el código de la búsqueda fue implementado en C#.
Para ejecutar el proyecto una vez descargado el código, solo debemos abrir la carpeta que lo contiene en un terminal y ejecutar el comando:

make dev

Haciendo un recorrido por el código del proyecto, encontramos las clases **Start**, **SearchQuery**, **Results**, **Moogle**, **SearchItem** y **SearchResult**.

Una vez se ejecuta el proyecto, inmediatamente se procede a llamar a la clase **Start.** Dicha clase es la encargada de almacenar todos los archivos de texto plano (.txt) que se encuentran ubicados en la carpeta Content (y en todas sus subcarpetas) en una Lista que, a su vez, tendrá un listado de las palabras que contenga cada documento, o sea, una Lista de Listas.
Este proceso se realiza a través de un método de la clase Start llamado LoadFiles, en el cual se ejecutan las siguientes funciones:

- TxtReader: Devuelve una lista de documentos que contiene, a su vez, un listado de cada palabra que aparece en un documento. Para lograr esto, se invocan los métodos GetFilesNames y WordsSeparator. GetFilesNames devuelve las rutas de todos los documentos contenidos en la carpeta Content, mientras que WordsSeparator se encarga de normalizar los caracteres que aparecen en el documento, evitando la aparición de signos de puntuación y cambiando las vocales con tildes por vocales sin acentos (esto lo realiza en una llamada a la función Normalize), y posteriormente separa las palabras de los archivos y las almacena en una lista de palabras.

- TFIDF Calculator: Su objetivo es calcular el TFIDF de todos los documentos. (Puede encontrar información sobre este algoritmo en *http://www.tfidf.com/)* Al finalizar su ejecución, tendremos los valores del TF almacenados en una lista de diccionarios y el IDF en un diccionario independientes.

- TFIDF: Es un método sencillo que devuelve una lista de diccionarios con el valor del TFIDF de cada documento.

- DocumentsMagnitude: A partir del empleo del modelo vectorial, este método calcula la Magnitud de un documento y lo devuelve como un array de valores de tipo double porque será necesario emplearlo más adelante, y es un cálculo que solo necesita ser realizado una única vez ya que no depende de la búsqueda que se realice. (Una breve explicación sobre el

modelo vectorial en https://www.sciencedirect.com/topics/computer-science/vector-space-models).

Con esta información almacenada, se proceden a ejecutar las instrucciones de la clase principal **Moogle.**
Una vez se introduce la búsqueda, la clase Moogle ejecuta una instrucción que llama al método WordsSeparator de la clase Start, pues se quiere separar las palabras de la query y almacenarlas también en una lista de palabras. Seguidamente, se ejecutan una serie de funciones contenidas en la clase **SearchQuery**:

- OperatorsUsed: Es una función que devuelve toda la información necesaria para tener los operadores que se emplearon en la búsqueda.

- TFIDF: Es un método nombrado de la misma forma que el que se encuentra en la clase Start, pero que esta vez solo recibe una lista de palabras, por lo que los parámetros no son los mismos, aunque la esencia de su objetivo sea el mismo. Se encarga de calcular el TFIDF de la query.

- QueryMagnitude: Calcula la magnitud de la query.

- CosineSimilarity: Emplea el modelo vectorial para calcular la similitud de cosenos de los documentos y lo devuelve como un array de double.

- Operators: Usando los datos que habían sido obtenidos del método OperatosUsed, se modifican los valores de la similitud de cosenos para lograr una búsqueda óptima. Esta función invoca a un submétodo llamado Proximity_Operator que se encarga de hacer las operaciones necesarias para el operador de cercanía si es que este fue usado. Esto fue diseñado así para lograr una mayor legibilidad del código ya que el operador de cercanía realiza funciones bastante diferentes del resto de operadores.

En este punto se tendrán identificados los documentos más relevantes a la búsqueda, donde entra en juego la clase **Results** para obtener un fragmento del documento que contiene parte de la búsqueda efectuada. Desde la clase principal Moogle se llaman los siguientes método de Results:

- GetHighestArrayPositions: Es el método que devuelve la posición que ocupan en el array de similitud de cosenos los documentos más relevantes. Si no se encontró ningún documento similar a la búsqueda, retorna -1.

A partir de este punto se divide el flujo del programa en dos variantes: si se encuentran resultados y si no se encuentran.
Veamos qué métodos se ejecutan si fueron encontrados:

- MostImportantDocs: Devuelve la ruta de los documentos más importantes ordenados.

- GetSnippet: Devuelve un fragmento del texto original que contenga parte del query.

Si no se encuentran resultados hay dos alternativas:

La primera consiste en buscar dentro de las palabras que contienen los documentos, sinónimos de la búsqueda ya realizada y comenzar todo el proceso de búsqueda nuevamente habiendo añadido todos los sinónimos al query.

La segunda alternativa es aplicar un algoritmo de distancia entre palabras para encontrar aquellas que estén contenidas en los documentos que más se asemejen a la búsqueda original. Por tanto, se llama al método Suggestions de la clase SearchQuery que realiza esta función auxiliándose de HammilDistance. Esto le otorga la opción al usuario de realizar una nueva búsqueda que sí tendrá resultados garantizados.