MOOGLE!

Moogle! es un buscador local de texto que utiliza el modelo vectorial TF-IDF en aras de obtener una mejora importante de eficacia y rapidez. Se basa en la idea de crear una base de datos con los documentos en los que se quiere trabajar, evaluar la similaridad entre el contenido de estos y la búsqueda realizada por el usuario, ordenar los documentos por este coeficiente, hacerle una sugerencia a este previendo posibles errores ortográficos o de otro tipo y finalmente mostrarle en pantalla los documentos coincidentes ordenados más una frase contenida en este donde puede encontrarse su búsqueda o algo relacionado con esta.

Funcionalidad:

- El programa cuenta con una clase Vocabulary, en la cual se encuentran implementados los métodos que dan forma a la base de datos. Primeramente el método ConvertToString() convertirá a cada documento en texto y el método ConvertToWorlds separará este en un array de palabras, eliminando signos de puntuación. Luego estos son utilizados por GenerateMatrix(), que genera una matriz(un array de diccionarios concretamente) en la cual cada elemento es un diccionario representante de un documento de texto. De tal forma cada elemento de esta matriz es representado como una palabra del respectivo documento y su TF.

-La clase Moogle contiene la parte funcional

del proyecto. Primero se convierte a palabras la query del usuario análogamente a como se hizo con los documentos. Luego se crea un vector que será un diccionario con los pares palabra-IDF. A través del método vectorQuery(), se determina un vector que contiene los pares palabra - (TF-IDF), para luego con el método documentsImportancy() obtener un nuevo y último diccionario con los pares índice de documento en la matriz - relevancia del documento,esta se halla mediante multiplicación escalar de cada vector documento(de la matriz, formado por pares palabra-(TF-IDF)) por el vector query, y luego se divide este resultado por el producto de las normas del vector documento y del vector query mencionados. Este es el coeficiente de similaridad que se asigna al score al instancias un objeto SearchItem(). Luego se ordenan estos objetos en el array SearchItem[] por es score mediante el método Sort(). (Entiéndase por norma de un vector, su concepto algebraico, que consiste en la raíz cuadrada de la suma de sus componentes y el score sería el coseno del ángulo entre dos vectores)

$$\mathrm{SimCos}(d_{(d)},q) = \frac{\sum_{n=1} \left(P_{(n,d)} \times P_{(n,q)} \right)}{\sqrt{\sum_{n=1} \left(P_{(n,d)} \right)^2 \times \sum_{n=1} \left(P_{(n,q)} \right)^2}}$$

-La sugerencia la fabrica el método WordsDistance de la clase Distance, que se encarga de comparar las palabras de la query con las de los documentos mediante la distancia de Levenshtein.
La distancia de Levenshtein, distancia de edición o distancia entre palabras es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, bien una inserción, eliminación o la sustitución de un carácter. Por ejemplo, la distancia de Levenshtein entre "casa" y "calle" es de 3 porque se necesitan al menos tres ediciones elementales para cambiar uno en el otro.

- El Snippet se obtiene mediante la clase de mismo nombre, con el método GetSnippet(). Su función es recorrer el documento y devolver la primera ocurrencia de la palabra más importante de la query en ese documento por el criterio TF.

Consideraciones:
-Este programa ha sido desarrollado para buscar en documentos en inglés, y no se asegura su correcto funcionamiento en otro idioma.
-La rapidez de este programa depende necesariamente del número de documentos y su tamaño. Es posible que su rendimiento se vea afectado para un número demasiado grande de documentos extensos. (Por lo que no espere que funcione para 100MB tan rápido como para 20MB).