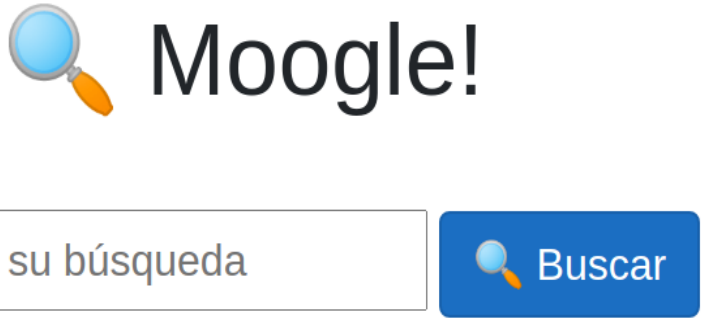
MATCOM
Facultad de Matemática y Computación
Universidad de La Habana

Proyecto de Programación I

Moogle!

Autor: Carlos Daniel Largacha Leal

Moogle! es una aplicación original cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos.



Interfaz gráfica de Moogle!

Ejecución del Moogle!:

-Copiar a la carpeta “Content” los documentos.

-Asegurarse de que los documentos sean de extensión “.txt”.

-Ejecutar el programa.

¿Qué hace el programa?

Al introducir una cadena de texto en el cuadro de búsqueda y hacer clic en el botón buscar Moogle! buscará la cadena de texto introducida en el conjunto de documentos dado, teniendo en cuenta la relación de cada palabra en cada uno de los documentos.

Se mostrarán los resultados más relevantes para su búsqueda y en cada uno se mostrará una sección del texto y el nombre del documento en el que se encontrará la cadena de texto introducida.

Ante una consulta con un error de escritura, el programa sugerirá una cadena de texto similar a la introducida que se encuentre en los documentos. Para usarla copie la sugerencia en el cuadro de diálogo y volver a realizar la búsqueda.

El código está agrupado bajo el ***namespace MoogleEngne*** y consta de 7 clases:

-Moogle: clase principal del proyecto.

-Ordenar: clase encargada de ordenar los resultados de búsqueda.

-Searchitem: información acerca de los documentos que responden a la búsqueda.

-Searchresult: representa el resultado de la búsqueda

-Suggestion: provee una sugerencia a partir de la consulta del usuario

-TF_IDF: convierte el conjunto de documentos en una matriz de vectores

-Utilidades: multiplica una lista de números decimales (donde se encuentra el tf_idf de la consulta) y se multiplica por una lista de matrices.

¿Cómo funciona el programa?

Comenzando con la clase (TFIDF), esta se encarga de crear un diccionario que relaciona las diferentes palabras de todos los archivos de texto con su orden de aparición. Luego, se genera una lista de matrices, donde cada posición "n" de la lista corresponde a la posición o valor de la palabra "n" en el diccionario. De esta manera, las matrices tienen un tamaño igual a la cantidad de archivos de texto. El objetivo de esta lista es organizar el tf-idf de cada palabra en cada texto.

También se tiene una lista (vector_idf) en la que se almacenan los valores idf de cada palabra y una lista de cadenas (títulos) donde se guardan los títulos de cada archivo de texto.

¿Qué es tf-idf?

tf-idf (*Term frequency – Inverse document frequency*), es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección. El valor tf-idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite manejar el hecho de que algunas palabras son generalmente más comunes que otras.

El ***tf-idf*** es el producto de dos medidas:

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$$

tf(t,d) representa la frecuencia de una palabra t en un documento d, es decir la cantidad de veces que una palabra se repite en un documento.

Idf(t,D) es frecuencia inversa de documento, es una medida de si la palabra t es común o no, en la colección de documentos. Se obtiene dividiendo el número total de documentos por el número de documentos que contienen el término, y se toma el logaritmo de ese cociente:

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

En la clase Utilidades, se encuentran dos métodos: uno que multiplica una lista de números decimales (donde se encuentra el tf-idf de la consulta) por la lista de matrices, y otro que realiza una multiplicación elemento a elemento entre una matriz y una lista de igual dimensión.

En la clase Moogle, la consulta se convierte en una matriz de cadenas separadas por palabras y luego se convierte en un vector calculando su **tf**. Este vector se multiplica término a término con el vector que contiene los valores **idf** de los textos, y con el vector resultante se calcula el puntaje de cada texto, multiplicándolo por la matriz que contiene los **tf-idf** de todos los textos. Se busca el fragmento relevante revisando cada texto relevante y buscando la palabra más relevante de ese texto para la búsqueda, y se devuelve una cadena de 20 palabras después de esa palabra.

Finalmente, se crea el objeto SearchResult y se ordena según su puntaje con la función Sort de la clase Ordenar, y se devuelve este objeto.

La clase Suggestion sirve para en caso de que alguna de las palabras de la consulta no esté presentes en ninguno de los documentos, se sugerirá una palabra que si esté presente en estos. Esto se hace empleando el algoritmo de Levenshtein: el cual permite calcular la cantidad de transformaciones mínimas a la que debe ser sometida una palabra para transformarse en otra. Estas transformaciones son:

Inserción: agregar un carácter.

Eliminación: quitar un carácter.

Sustitución: cambiar un carácter por otro.