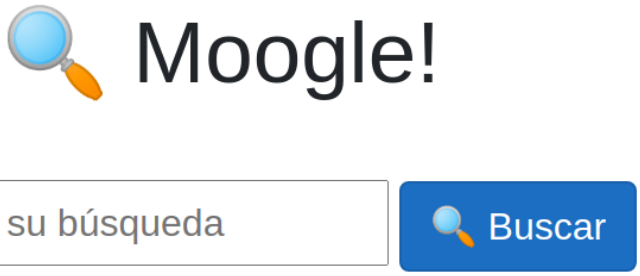
Moogle!

Luis Daniel Silva Martínez
C-111

21 de Julio, 2023



Contents

1 Introducción

¿Alguna vez ha deseado buscar entre todos sus archivos de texto una palabra o frase particular? ¿Ha deseado organizar sus documentos de texto según su relevancia respecto a una búsqueda? Pues usted debería probar Moogle!, el nuevo buscador de texto actualmente sensación entre decenas de estudiantes de la Facultad de Matemáticas y Computación de la Universidad de La Habana.

1.1 Qué es Moogle!?

Moogle! es una aplicación totalmente original desarrollada con tecnología .NET Core 6.0, específicamente usando Blazor como framework web para la interfaz gráfica, y en el lenguaje C#.

1.2 Para qué sirve?

Aplicando el Modelo Vectorial y la medida numérica TF-IDF, esta aplicación es capaz de leer archivos de texto en formato .txt ubicados en la carpeta Content y una búsqueda introducida por el usuario, y devolver los documentos de texto relevantes en relación a lo buscado, ordenados de más relevantes a menos.

1.3 Cómo usarlo?

Para ejecutar el proyecto en Linux se debe abrir la terminal de Linux y ejecutar el comando: `make dev`
En Windows, el comando para ejecutar la aplicación es (desde la carpeta raíz del proyecto): `dotnet watch run --project MoogleServer`
Una vez con el proyecto andando, basta con escribir la búsqueda que se desea realizar y pulsar el botón *Buscar* para obtener los resultados.

2 Funcionamiento

La aplicación está dividida en dos componentes fundamentales:

- `MoogleServer` es un servidor web que renderiza la interfaz gráfica y sirve los resultados.
- `MoogleEngine` es una biblioteca de clases donde está implementada la lógica del algoritmo de búsqueda.

2.1 Implementación

El programa hace uso de un modelo vectorial con el algoritmo TF-IDF (Term Frequency - Inverse Document Frecuency)(1). Se separan todos los términos del texto en cada documento de la carpeta Content y se calcula su valor TF-IDF para vectorizar los documentos. La entrada del usuario (query) es tratada como un documento más, que recibe el mismo

tratamiento anterior. Luego se compara el vector de la query con cada vector de los documentos procesados usando la similitud de coseno(2) para obtener los archivos con mayor semejanza a la búsqueda del usuario. Luego de obtener los resultados, en caso de recibir pocos, se calculará una sugerencia de query usando la distancia de Leveshtein, bastante útil en casos donde ocurren errores de ortografía o typos.

$$TFIDF_{(t,d)} = \frac{tf}{tw} \times (\log_2 \frac{1+N}{1+df}) \tag{1}$$

- t: término
- d: documento
- tf: frecuencia del término t en d
- tw: total de términos en el documento d
- df: cantidad de documentos que contienen a t
- N: total de documentos en el corpus

$$SimCos(\theta) = \frac{D \cdot Q}{||D|| \cdot ||Q||} \tag{2}$$

- D: vector del documento
- Q: vector de la query (búsqueda)

2.2 Flujo

El proyecto sigue un camino marcado:

1. Inicialización de la interfaz gráfica
2. Procesamiento de los documentos de la carpeta Content
3. Cálculo de los pesos TF-IDF de cada término en los documentos procesados
4. Recepción de la búsqueda (query) introducida por el usuario
5. Procesamiento de la query y cálculo del peso TF-IDF de cada término de la misma
6. Cálculo de la similitud y relevancia de cada documento respecto a la query
7. Obtención de un fragmento del documento (snippet) relevante respecto a la query
8. Cálculo de una sugerencia en caso de que la búsqueda genere pocos resultados
9. Obtención de los resultados y mostrarlos en la interfaz gráfica
10. Nueva búsqueda

2.3 Descripción del Código

La lógica del algoritmo de búsqueda está implementada en el `MoogleEngine` con diferentes clases.

2.3.1 Moogle.cs

La clase Moogle.cs es el puente entre todo lo que sucede por detrás en el proyecto y la interfaz gráfica y la que se encarga de mantener el flujo del programa.

2.3.2 DataBase.cs

DataBase.cs se encarga de la obtención de los documentos en la carpeta Content y su tokenización usando Regex.

2.3.3 TFIDF.cs

Como su nombre lo indica, esta clase calcula el peso TFIDF de cada término y los asocia juntos.

2.3.4 Query.cs

la clase Query.cs toma la entrada del usuario, la tokeniza y calcula el TFIDF de los términos.

2.3.5 Similarity.cs

Similarity.cs calcula el score de cada documento respecto a la query con la fórmula de similitud del coseno para obtener los documentos más relevante a la búsqueda. Además calcula el snippet que será mostrado junto al título del archivo de texto.

2.3.6 Suggestion.cs

Encargada de obtener una nueva query similar a la del usuario pero con terminos presentes en el corpus de documentos, usando el algoritmo de distancia de Leveshtein.

3 Especificaciones

El programa está cableado para encontrar la carpeta Content dentro de la carpeta que contiene a todo el proyecto. Para realizar búsquedas entre los documentos .txt de su elección, deberá copiarlos a la carpeta Content ubicada en el directorio principal del proyecto. Moogle! puede demorarse cargando la interfaz gráfica en algunas máquinas, y en su primera búsqueda tarda más de lo usual, cargando y procesando los documentos en Content. Una vez realizada la primera búsqueda y cargados los documentos, las siguientes búsquedas serán más rápidas.

4 Funciones extras

- Para introducir la query puede presionar Enter luego de escribir para que haga la búsqueda sin tocar el botón de Buscar.
- Para introducir la sugerencia dada por el programa puede presionarla directamente y llevara a cabo la búsqueda con esa sugerencia como query.