Primer Proyecto de Programación Moogle!

Lianny Revé Valdivieso

Facultad de Matemática y Computación de La Universidad de La Habana

25 de Julio de 2023

Moogle!

Moogle! es una aplicación *totalmente original* cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos.
Es una aplicación web, desarrollada con tecnología .NET Core 7.0, específicamente usando Blazor como *framework* web para la interfaz gráfica, y en el lenguaje C#. 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.

Sobre la búsqueda

Necesitas tener instaldo algún navegador y tener a dotnet disponible, o equivalente a esto tener .NET CORE 7.0 instalado, para ejecutarlo te debes ubicar en la carpeta del proyecto y ejecutar en la terminal de Linux:

```
make dev
```

Si estás en Windows, debes poder hacer lo mismo desde la terminal del WSL (Windows Subsystem for Linux), el comando ultimate para ejecutar la aplicación es (desde la carpeta raíz del proyecto):

```
dotnet watch run --project MoogleServer
```

Sobre la búsqueda

Para realizar una búsqueda, solo se necesita introducir una frase en el cuadro de búsqueda y hacer click sobre el botón buscar o bien, presionar la tecla Enter.
Implementamos la búsqueda de la manera más inteligente posible, de forma tal que el usuario no necesita limitarse a los documentos donde aparece exactamante la frase introducida por el usuario.
Aquí van algunos requisitos que debe cumplir esta búsqueda:

- En primer lugar, el usuario puede buscar no solo una palabra sino en general una frase cualquiera.
- Si no aparecen todas las palabras de la frase en un documento, pero al menos aparecen algunas, este documento también será devuelto, pero con una relevancia menor mientras menos palabras aparezcan.

- El orden en que aparezcan en el documento los términos de la consulta en general no importa, ni siquiera que aparezcan en lugares totalmente diferentes del documento.
- De la misma forma, si un documento tiene más términos de la consulta que otro, en general tiene una mayor relevancia (a menos que sean términos menos relevantes).
- Algunas palabras excesivamente comunes como las preposiciones, conjunciones, etc., serán ignoradas por completo ya que aparecerán en la inmensa mayoría de los documentos. Este requisito es funcional dentro de un conjunto de archivos en un mismo idioma, ya que está programado para que se haga de forma automática, o sea, no hay una lista cableada de palabras a ignorar, sino que se computan de los documentos.

Flujo de Funcionamiento

Al iniciar el servidor, este crea una instancia de Searcher el cual carga los Documentos y los procesa individualmente para obtener los datos relevantes sobre ellos, tales como su nombre, ruta y frecuencia de sus términos, estos luego son utilizados para crear un diccionario que contiene todos los términos del corpus textual (los documentos), con esto se calcula por cada término de cada documento su frecuencia inversa o IDF y se utiliza esta métrica para normalizar la frecuencia del término en el documento (TF) y asi definir la relevancia de cada uno en el documento y en el corpus, estos valores son almacenados para utilizarse en las búsquedas.

Cuando un usuario realiza una consulta mediante la interfaz gráfica esta pasa al servidor donde se procesan los datos introducidos por el usuario tales como: separar y procesar los terminos y calcular a cada uno de ellos su relevancia de la misma manera que se efectuó con cada documento del corpus. Tras desarrollarlo se procede a reducir el espacio de búsqueda mediante el cálculo de la relevancia o score de cada documento respecto al query, esto se cubre utilizndo la distancia coseno entre vectores la cual es una medida de similitud entre vectores, luego de calcular todos los scores , se organiza la lista de documentos según su score de mayor a menor y se devuelven al usuario, si hubo alguna palabra que no se pudo encontrar no será devuelto ningún resultado y el usuario debrá realizar una nueva búsqueda.

Sobre la Ingeniería de Software

Para implementar el algoritmo de búqueda hemos creado e implementado varias clases fundamentales. Cada clase es una abstracción de los componentes del motor de búsqueda.

- `DataFile` es una clase creada como representación de lo que es un documento.
- `DataFolder` por su parte, representa una carpeta o contenedor de documentos (o DataFiles), con sus propiedades y métodos proprios.
- `Query` es la clase que representa y procesa la consulta realizada por el usuario.
- `Tools` Contiene herramientas (métodos) para procesar texto.

- `Engine` es una clase base que se encarga de manejar los objetos de tipo `DataFile`, `DataFolder` y `Query` para la obtención de objetos de tipo `SearItem` y posteriormente de tipo `SearchResult` que son los devueltos al usuario.
- `SearchItem` representa objetos que son posibles documentos que coinciden al menos parcialmente con la consulta en `Query`.
- `SearchResult` representa un objeto que contiene los resultados de la búsqueda realizada por el usuario.
- `Moogle` contiene el método `Moogle.Query`. Este método devuelve un objeto de tipo `SearchResult`.