MoogleEngine

Noel Hernández Morffi, C113

Julio, 2023

Abstract

Motor sencillo de búsqueda en el que la lógica del ranking de relevancia es TF-IDF; las consultas tienen operadores de aparición; y los documentos sólo se cargan con la primera búsqueda de cada ejecución, por lo que es eficiente.

Contents

1 Ejecutando el Proyecto

Para ejecutar el proyecto:

1. En Moogle.cs, cambiar el valor de la variable 'ContentPath' por la ruta de los archivos txt.
2. Cargar el proyecto: Pararse en la carpeta del proyecto, abrir un terminal y usar el comando: 'dotnet build'.
3. Abrir la carpeta del proyecto con vscode (en la carpeta del proyecto, abrir un terminal y usar el comando: '. code') (o click derecho-⟩ abrir con-⟩ vscode).
4. Dentro de vscode, abrir un nuevo terminal, y ejecutar el proyecto con el comando: 'dotnet watch run –project MoogleServer'.
5. Cuando abra el navegador, usar el buscador.

2 Funcionalidades básicas

Funcionalidades básicas:

- El ranking de relevancia es TF-IDF.
- Las consultas tienen operadores de aparición:
  - Un símbolo '!' delante de una palabra indica que esa palabra no debe aparecer en ningún documento que sea devuelto.
  - Un símbolo '^' delante de una palabra indica que esa palabra sí debe aparecer en cualquier documento que sea devuelto.
- Los documentos solo se cargan con la primera búsqueda de cada ejecución, por lo que es eficiente.

3 Implementación

3.1 Abstracción de Datos

La clase 'Moogle' tiene variables estáticas (son estáticas porque la mayoría tiene valores constantes que no cambian durante la ejecución, y almacenan información importante que se comparte entre varias funciones de la clase):

- 'contentPath' de tipo string: aquí va la ruta de la carpeta de todos los documentos a buscar.
- 'filePaths' de tipo array de string: en cada posición se guarda la ruta de cada archivo de extensión .txt.
- 'cantTxts' de tipo int: cantidad de txts.
- 'primeraBusqueda' de tipo bool: indica si es la primera búsqueda (consulta) o no.
- 'd_TitleText' de tipo Dictionary$\langle string, string \rangle$: diccionario en el cual cada Llave será el título de cada documento; y cada Valor el texto de ese documento (para el snippet).
- 'd_TitlePalabraTf' de tipo Dictionary$\langle string, Dictionary\langle string, int \rangle \rangle$: diccionario en el cual cada Llave será el título de cada documento; y cada Valor un diccionario, en el cual cada Llave será cada palabra de ese documento, y cada Valor la frecuencia de esa palabra (tf) en ese documento.
- 'd_PalabraIdf' de tipo Dictionary$\langle string, double \rangle$: diccionario en el cual cada Llave será cada palabra de la consulta (query), y cada Valor la frecuencia inversa de esa palabra (idf).

3.2 Flujo de ejecución

En cada búsqueda(o llamada al método 'Moogle.Query'):

- Las palabras de la consulta (query) se guardan en un array de tipo string 'palabrasQuery', que además tiene 2 máscaras de tipo bool ('palabrasNoDebenAparecer' y 'palabrasSiDebenAparecer') para indicar cuáles palabras tienen operadores de aparición. El método 'OperadoresAparicion' se encarga de darle valores correctos a las 2 máscaras y limpiar (eliminar) los símbolos de operadores de cada palabra de 'palabrasQuery'.
- Se crea el array de SearchItem 'items', que luego será pasado como parámetro al objeto 'result' de tipo SearchResult, que devuelve el método 'Query'. Además se inicializa cada objeto de tipo SearchItem de 'items' (para permitir indexar en 'items'), llamando al método 'SearchResult.Inicializar'.
- Si es la primera búsqueda(consulta), se cargan los diccionarios 'd_TitleText' y 'd_TitlePalabraTf' con sus valores correctos, llamando a los métodos 'Cargar_d_TitleText' y 'Cargar_d_TitlePalabraTf' respectivamente. Si no es la primera búsqueda, entonces no se cargarán estos 2 diccionarios, pues ya estarían cargados.
- Se carga 'd_PalabraIdf' llamando al método 'Cargar_d_PalabraIdf'. Para cada palabra del query, su idf $= \log_2 \left(\frac{\text{cantTxts}+1}{\text{CantTxtsConPalabraX(palabra)}+1} \right)$.
- Se llama al método 'AsignarTitleSnippetScore', donde se asignan los valores de title, snippet y score a cada elemento de 'items', recorriendo cada elemento de 'items' a la par que cada elemento de 'd_TitlePalabraTf'. En este método se llama al método 'ScoreAndValidez', el cual además de retornar el valor correcto de score (multiplicando $tf \times idf$), aprovecha el recorrido de 'items', 'd_TitlePalabraTf' y 'palabrasQuery' para dar la validez de cada documento(elemento de 'items') según las palabras que deben o no aparecer. Si el doc. no es válido, title y snippet serán string vacíos, y score -1.
- Se crea el objeto de tipo SearchResult 'result' a devolver. Se ordena por score con el método 'OrdenaPorScore' de la clase SearchResult (bubble sort).

- Y antes de retornar el resultado, cambiamos 'primeraBusqueda' a false, pues ya se ha realizado la búsqueda(consulta).