Informe escrito de Proyecto de Programación I: MOOGLE!

Eveliz Espinaco Milián

Grupo C112

Primer año de Lic. en Ciencia de la Computación
Facultad de Matemática y Computación
Universidad de La Habana
Curso 2023-2024

Índice

1. Introducción

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 6.0, específicamente usando Blazor como *framework* web para la interfaz gráfica, y en el lenguaje C#.
La aplicación está dividida en dos componentes fundamentales: MoogleServer, un servidor web que renderiza la interfaz gráfica y sirve los resultados; y MoogleEngine, una biblioteca de clases donde está implementada la lógica del algoritmo de búsqueda. Moogle! está formado por un total de 6 clases: Moogle, SearchItem, SearchResult, StaticMatrix, QueryVectors y OperationsVectors.
En el presente informe se explicará lo más detalladamente posible la lógica del algoritmo de búsqueda utilizado, así como el flujo de ejecución de la aplicación a través de la explicación de la clase Moogle, la cual fue modificada y las clases StaticMatrix, QueryVectors y OperationsVectors, de mi autoría.

2. Moogle

La clase Moogle es el motor impulsor del Moogle!. A través del método Query se encarga de dirigir el flujo de ejecución del código, el cual consiste en hacer la búsqueda a través de la multiplicación de una matriz por un vector. Para ello es necesario crear tanto la matriz como el vector, luego determinar la multiplicación y por último establecer el snippet del documento de mayor score que se va a devolver (todo este proceso se explicará más detalladamente a lo largo del informe).
Moogle, primeramente, por medio del método OpenDataBase ordena abrir los documentos de la base de datos en un array de tipo string el cual se envía como parámetro al método ToDoMatrix de la clase StaticMatrix encargado de construir la matriz, este proceso ocurrirá una única vez, en la primera ejecución de la aplicación. Luego pasa como parámetro el string query introducido por el usuario a la clase QueryVectors el cual devuelve una dupla de vectores (un vector principal y un vector sugerencia). Estos son remitidos al método Multiplication de la clase OperationsVectors, el cual devuelve un vector que contiene el score de cada documento respecto a la búsqueda, de aquí se selecciona el mayor y se le pasa como parámetro al método Snippet.

Campos de la clase:

- Files: es un array con la ruta de cada uno de los documentos de la base de datos.
- Content: es un array con la información de cada uno de los documentos de la base de datos.
- LengthDataBase: total de palabras de la base de datos.
- LengthDoc: total de palabras de cada documento.
- TotalDoc: Total de documentos que dispone la base de datos.
- TextSplitter: Es la matriz en que se convertirá la base de datos.
- Synonymous: es un diccionario que dispone el programa con sinónimos.

- executionsProgram: Es un contador de ejecuciones del programa. Solo si es la primera ejecución se cargará la base de datos, evita así que se repita este proceso con cada ejecución.

3. StaticMatrix

Esta clase es la encargada de crear la matriz donde estará toda la información disponible para dar respuesta a la búsqueda.

Métodos que la conforman:

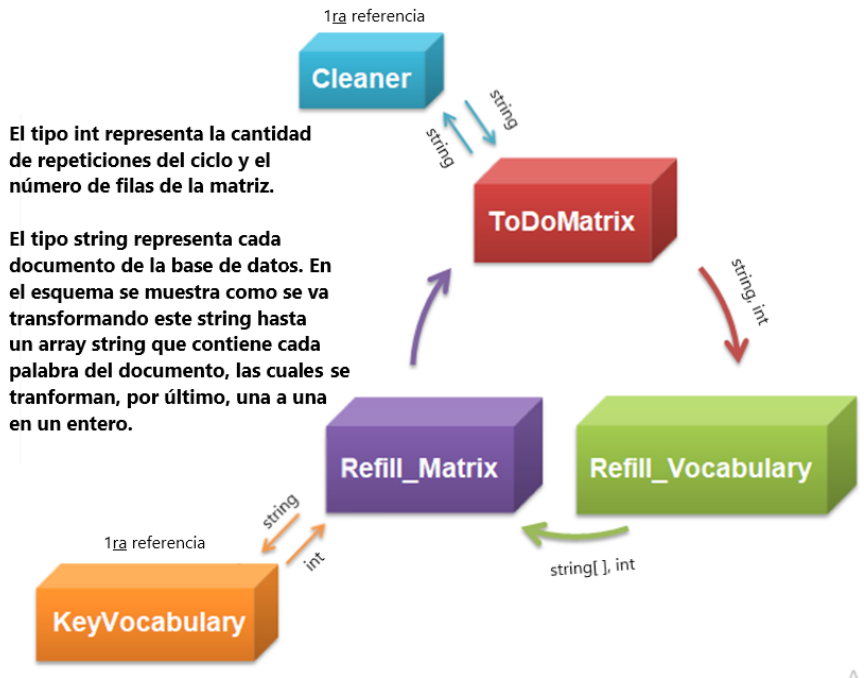
- ToDoMatrix: recibe el array content y se da la tarea de recorrerlo y mandar a limpiar el contenido de cada uno de sus documentos de tildes, mayúsculas y signos de puntuación y de formar con ellos el vocabulario del que dispondrá el programa.
- Cleaner: es el que elimina las tildes, las mayúsculas y signos de puntuación.
- RefillVocabulary: recibe los documentos uno a uno ya normalizados, los divide en palabras y guarda cada una en una posición de un array, recorre este y archiva cada palabra en un diccionario(Vocabulary), haciendo función de llaves a las cuales les corresponde un número único(Key).
- RefillMatrix: recibe también un documento a la vez. Almacena el documento n en la fila n de la matriz y cada una de sus columnas representa una palabra. Cuando el flujo del programa llega a este método pasando como parámetro un documento, este anteriormente fue procesado por RefillVocabulary, por lo que su función es revelar con cada palabra el número que le corresponde en el diccionario Vocabulary, con ayuda del método KeyVocabulary. Este número representará la columna en la matriz, en esa posición se guardará la cantidad de veces que aparece la palabra entre la cantidad total de palabras del documento n.
- KeyVocabulary: recibe una palabra y devuelve el número que le corresponde en el diccionario Vocabulary.

Campos de clase:

- Key. Solo es utilizada por el método RefillVocabulary; consiste en un contador de palabras, es una variable global debido a la necesidad de que el conteo no se reinicie mientras se recorren los documentos.
- Vocabulary: contine todas las palabras que dispone el programa para la búsqueda. Es utilizado además de por el método RefillVocabulary, que lo rellena, y otros que a partir de él crean una matriz (RefillMatrix) o vectores como se verá más adelante.

¿CÓMO FUNCIONA?

Una vez abierto cada documento de la base de datos que se dispone en el array content, este es enviado al método ToDoMatrix de la clase StaticMatrix, este va recorriendo el array e interactuando en cada recorrido con los demás métodos de la clase como se muestra en el esquema:



Este proceso se realiza con cada documento, o sea con un for que dirige el método ToDoMatrix. Una vez terminado todo el recorrido quedará como resultado una matriz (textSplitter) donde se buscará las respuestas de las búsquedas.

4. QueryVectors

Como toda la información de nuestro programa se ha convertido en una matriz para poder interactuar con ella necesitamos un ente de la misma especie, vectores. La funcionalidad de esta clase es precisamente la creación de estos vectores, específicamente 2: el primero, un fiel seguidor de cada palabra del usuario y el segundo adiciona sinónimos disponibles de los vocablos introducidos.

Métodos que la conforman:

- QueryVector: Recibe la búsqueda del usuario y se encarga de crear ambos vectores. Para ello ordena limpiar la frase (con el método Cleaner de la clase StaticMatrix, visto anteriormente), dividirla y hacerle corresponder a cada palabra una posición de un array (a través del método KeyVocabulary de la clase StaticMatrix, visto anteriormente), la cual rellena con una fórmula:

  $$(SearchRequests) * log(total\_de\_documentos/(TdocQuery)$$

  esta ecuación se entenderá mejor más adelante, y solicita al método AddSynonyms. Como objetivo principal del programa es complacer en todo lo posible las necesidades del usuario, se crean facilidades que permiten una comunicación más específica, las cuales se analizan en este método con ayuda de SearchRequests.

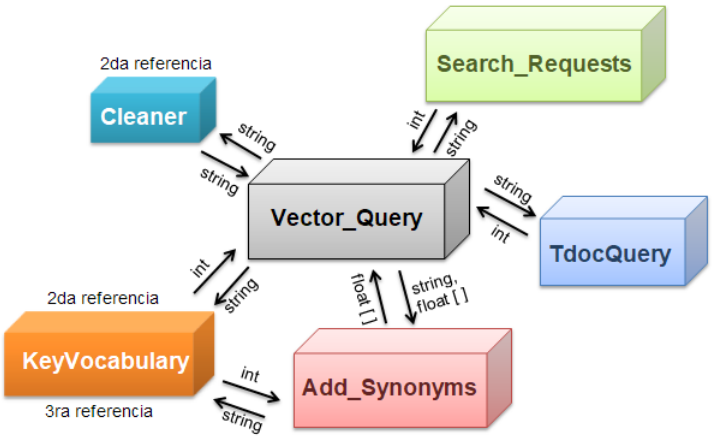
- SearchRequests: determina el valor de las palabras de la búsqueda para ello analiza si el usuario implementó alguna petición como la posibilidad de utilizar en la búsqueda caracteres especiales como: ! (le quita importancia a la palabra), $\wedge$ (la palabra tiene que aparecer) y $*$ (le da importancia a la palabra, se pueden utilizar tantos asteriscos como se desee, la correspondencia de la cantidad de asteriscos y la importancia de la palabra es una función exponencial de la forma y = 2x, donde la x son la cantidad de asteriscos e y la importancia de la palabra).

- TdocQuery: establece la cantidad de documentos que contienen la búsqueda para la fórmula utilizada por VectorQuery.

- AddSynonyms: Busca en un diccionario de sinónimos del que dispone el programa (Synonyms) equivalentes a las palabras que implementó el usuario y los incorpora a lo que será un segundo vector.

¿CÓMO FUNCIONA?

Una vez que el usuario introduce una frase de búsqueda esta se envía al método VectorQuery que interactúa con otros métodos como se muestra en el esquema:



En este punto del flujo del programa ya tenemos establecido tanto la matriz y los vectores de búsqueda, ahora, ¿cómo se obtendrá el hallazgo?

5. OperationsVectors

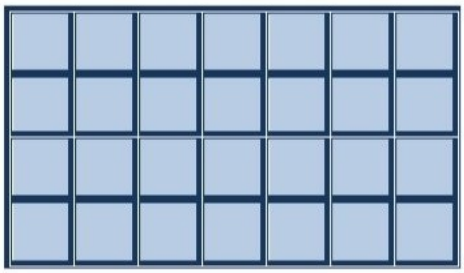
Métodos que la conforman:

- Multiplication: Realiza el cálculo de la fórmula TF-IDF a través de la multiplicación de una matriz. $\mathbf{M}_{mxn}$ (la base de datos) por una matriz $\mathbf{M}_{nx1}$ (los vectores de búsqueda).

Este proceso se hace con ambos vectores. El array es devuelto al método Query de la clase Moogle donde se determina el documento de mayor score fiel a la búsqueda y el segundo será el de mayor score de los documentos restantes incluyendo los del vector sugerencia. Con esta información se completa title, snippet y score, respuesta de la búsqueda.

M_{mxn} Donde m es la cantidad de documentos y n representa la frecuencia de la palabra **(a/b)** donde **a** en **la cantidad de repeticiones de la palabra en el documento** y **b la cantidad total de palabras que tiene el documento**

M_{nx1} Está formada por ceros en su mayoría solo en la posición que le corresponde a las palabras del query valdrá **(SearchRequests) * log(c/d)** donde **c** es **el total de documentos de la base de datos** y **d la cantidad de documentos que contienen la palabra**



$$M_{mxn} * M_{nx1} = M_{mx1}$$

(SearchRequests) * (a/b) * log(c / d)

Si la palabra no tiene ninguna relevancia SearchRequests = 1

Cada posición n va a valer el score del documento n