Informe de Moogle!

José Miguel Leyva De la Cruz

1 Clase DataServer

Esta clase tiene como principal objetivo crear un objeto de tipo **BBDD**, llamado **BaseDatos**, e inicializar la constante **Delimitors**, siendo esta un string que contiene todos los símbolos de puntuación del idioma español.

El objeto **BaseDatos** al ser de tipo **BBDD**, se encarga de crear la Base de Datos y el Universo de Palabras sobre el cual trabajará el motor de búsqueda. Debido a lo costoso que es realizar semejante tarea, este objeto es creado antes de que la aplicación comience a funcionar, de esta manera se tendrá la base de datos lista para realizar todas las operaciones necesarias.

2 Clase BBDD para crear la Base de Datos y el Universo de Palabras

```
13 references
public Dictionary<string, Dictionary<string, double>> Larousse; //Almacenar la relacion palabra documento
4 references
public Dictionary<string, Dictionary<string, double>> TF_IDF; //Almacenar todas las fecuencias
10 references
public Dictionary<string, InfoDocument> Infos;
4 references
public int totalfichers;
2 references
private char[] delimitors;
```

Figure 1: Campos de la clase **BBDD**

La clase BBDD posee varios campos. Entre ellos se aprecian las tres estructuras más importantes de la aplicación: **Larousse, TF_IDF** e **Infos**. Su objetivo fundamental consiste crear un objeto en el cual, utilizando las estructuras antes mencionadas, almacena todo el universo de palabras que utilizará el programa y les establece una relación con los documentos en los que se realizará la búsqueda. Además, se encarga de la **Vectorización** de los documentos. Todo este proceso será explicado a continuación.

2.1 Larousse

Esta estrucutura de datos es un diccionario encargado de almacenar en sus **keys** las palabras que se utilizarán para las operaciones de búsqueda, y en sus **values** son otro diccionario que guarda como **key**, los documentos en los que aparece dicha palabra y como **value**, la cantidad de veces que esta se repite en el texto del documento.

2.2 TF_IDF

Muy parecida a **Larousse**, su única diferencia es que en vez de almacenar la cantidad de veces que una palabra aparezca en un documento, guarda el valor de su **TF_IDF** con respecto al documento en que se encuentra.

2.3 Infos

Este último diccionario, tiene como **Keys**, las rutas de los documentos y como **values**, objetos del tipo **InfoDocument**, se aqbordará sobre ellos más adelante.

2.4 Vectorización

Para realizar todas las operaciones de búsqueda y los diferentes cálculos que se abordarán en breve, la herramienta **Moogle!** utiliza un **Modelo Vectorial**. Para ello necesita expresar los documentos en forma de vectores. La vectorización empleada consiste en que cada coordenada del vector es la cantidad de veces que una palabra se repite en un documento. Ejemplo: Suponga que existe un documento que posee 20 palabras, una de estas palabras se repite 10 veces, otra 7 y la otra 3, por tanto ese documento expresado en forma de vector sería $(10, 7, 3)$. Aquí se aprecia la aplicación del diccionario **Larousse**, de esta manera se resume que los **values** del mismo son los documentos ya vectorizados.

2.5 Método AddFile

Este es el único método que posee la clase **BBDD**, recibe como parámetro un string (ficher) y tiene como funcionalidad, "llenar" el diccionario **Larousse**. Lo hace de la siguiente forma:

El constructor de la clase BBDD crea dentro de su ámbito un array de string el cual contiene las rutas de todos los documentos, luego mediante el uso de un ciclo **for**, llama al método **AddFile** en cada iteración del mismo, pasándole como argumento la dirección de cada documento. Una vez dentro del método, crea un string utilizando el método ReadAllText de clase File y utiliza además el método ToLower para leer todo el contenido del documento y normalizarlo a letras minúsculas. Hecho esto crea un array de strings mediante el método Split, utilizando el campo **Delimitors** de la clase **DataServer**. Ahora, itera por cada palabra utilizando un bucle **Foreach** y hace diferentes preguntas: Si la palabra no está en **Larousse**, pues crea un diccionario nuevo cuyo par de <**Key,Value**> será el documento actual, y el número 1, finalmente agrega la palabra y este nuevo diccionario al **Larousse**; si la palabra ya estaba guardada pero el documento es nuevo, le añade a dicha palabra el nuevo documento; si la palabra ya estaba en **Larusse** y continua en el mismo documento, entonces aumentar en 1, la cantidad de veces que dicha palabra aparece. Ya vectorizados los documentos, se puede proceder al cálculo de la "Norma" de cada uno de ellos. La norma de un vector se calcula de la siguiente forma:

$$\sqrt{\sum_{i=1}^{n}(x_i)^2}$$

Siendo x_i la componente i-ésima del vector.

Una vez finalizado el proceso de Crear la Base de Datos y haber vectorizado los documentos. El constructor de la clase **BBDD** todo lo que le resta por hacer es inicializar **TF_IDF**, mediante la clase **Frecuency**.

3 Clase InfoDocument

Esta clase, crea objetos de tipo **InfoDocument**, los cuales se utilizan para guardar las distintas propiedades de los documentos:

1. norma: Una vez vectorizado el documento, este valor almacena la norma del vector.
2. index: El índice del documento dentro del Diccionario **Infos**, de la clase **BBDD**.
3. cuerpo: El contenido del documento.

```
1   namespace MoogleEngine;
    5 references
2   public class InfoDocument
3   {
4       //Guardar la informacion de todos los documentos
        1 reference
5       public InfoDocument(double norma, int index, string cuerpo)
6       {
7           this.norma = norma;
8           this.index = index;
9           this.cuerpo = cuerpo;
10      }
11
        16 references
12      public int index { get; set; }
        2 references
13      public double norma { get; set; }
        5 references
14      public string cuerpo { get; set; }
15  }
```

Figure 2: Clase **InfoDocument**

4 Clase Frecuency, Cálculo del TF_IDF

4.1 Método TF_IDF

Este método calcula el **TF_IDF**, simplemente aplicando la fórmula, para la parte **TF** (Term Frecuency o Frecuencia de Términos), dividie la cantidad de veces que una palabra aparece en un documento entre el total de términos que este

posee. La parte **IDF** (Inverse Term Frecuency ó Frecuencia Inversa) consiste en calcular el $\log_{10}$ del, total de documentos entre la cantidad de documentos en los que aparece el término en cuestión. Finalmente para obtener el **TF_IDF** se multiplican ambas partes.

$$TF.IDF = \frac{t_i}{d_j} * \log \frac{N}{N_i}$$

Siendo t_i, la cantidad de veces que aparece el i-ésimo término en el documento, d_j la cantidad de términos que posee el documento, N el total de documentos de la Base de Datos y N_i la cantidad de documentos que contienen dicho término.

4.2 Método VectorWord

Este método recibe como parámetros un diccionario, **vector**, en el cual estarán todos los documentos que contienen el término actual, y la cantidad de veces que se repite dicho término; otro diccionario que contiene los documemtos y objetos de tipo **InfoDocument** y una variable que representa el total de ficheros. La función de este método es iterar por todos los elementos de **vector**, convertir todo el contenido de cada elemento en un **array** de **string** mediante el método **Split**, y luego llamar al método **TF_IDF** explicado anteriormente.

4.3 Método GetTF_IDF

Este método, recibe en sus parámetros el diccionario **Larousse** y el diccionario **Infos** ambos pertenecientes a la clase **BBDD**. Su funcionalidad consiste en iterar por cada una de las palabras almacenadas en el **Larousse**. Con cada iteración se llama al método **VectoWord** y los resultados que este va obteniendo se guardan en un nuevo diccionario cuyas **keys** son palabras de **Larousse**, y sus **values** son otro diccionario que tiene como pares los documentos en los que aparece la palabra y el **TF_IDF** de dicho archivo con respecto a la palabra.

Luego de todos estos procesos estarán creadas las tres estructuras fundamentales de la aplicación. Se puede notar que este procedimiento, es bastante largo y costoso para la pc, por tanto se realiza ante de arrancar el buscador; de esta forma, ya el programa tendrá todo su universo de palabras cargado, el **TF** de cada palabra con respecto a cada documento y el **IDF** de cada documento con respecto a cada palabra.

Figure 3: **Crear la estructura el objeto BaseDatos antes de correr la aplicación**

5 Clase Moogle

5.1 Método Scores

Este método como su nombre lo indica devuelve un array de double con los **Score** de cada documento con respecto a la búsqueda introducida por el usuario, dicho array es creado totalmente nuevo dentro del método y tiene como tamaño, la cantidad de documentos que haya en la base de datos. Recibe como parámetro una lista de palabras. Luego itera por cada una de las palabras de la lista, verifica si pertenecen al Universo de palabra creado y alamacenado en **Larousse**, en caso de que no: pues la "ignora", en caso contrario entonces: Indexa en la palabra en el Diccionario **TF_IDF**, objeto BaseDeDatos creado anteriormente, e itera por cada documento que esta palabra tenga asociado (Recordar que estos diccionarios almacenaban el documento en el que aparece la palabra y el valor del TF_IDF de dicho documento con respecto a la palabra). Utiliza el tipo de dato **var** para crear una variable "info", en la cual se guardará el objeto **InfoDocument**, que le corresponde al documento sobre el cual se esté iterando. Finalmente en el array de nombre "score", se almacena en la posición de info.Index la multiplicación de la cantidad de veces que la palabra aparece en la lista pasada como parámetro por el **TF_IDF** del documento.

Lo explicado anteriormente fue el producto escalar entre vectores, un vector **Query** y un vector **Document**, ambos vectorizados de la forma en que se explicó anteriormente. Luego de tener el producto escalar de ambos vectores, este número se divide entre el producto de las normas de cada vector. Haciendo esto se obtiene el valor del coseno del ángulo que forman estos dos vectores, o sea todo este proceso fue aplicado para calcular el **Coseno de Similitud entre dos documentos con forma de vectores**, ya que mientras mayor sea el coseno, más pequeño será el ángulo, es decir los vectores estarán más "cerca". En otras palabras, es el documento que más se "parece" a la búsqueda introducida, por tanto es el más relevante.

$$\frac{V_1 * V_2}{|V_1| * |V_2|} = \cos \prec (V_1, V_2)$$

5.2 Método SnippetPlus

Este método utiliza 4 enteros: start (para identificar el indice en donde comenzará el snippet), end (el indice del final del snippet), cant (cantidad de veces que una palabra de la query aparezca en el snippet que se analiza) y temp (para ir comparando). El snippet, no es más que un fragmento del documento, pero no cualquiera, es el pedazo que mayor relevancia tenga con respecto a la búsqueda del usuario, es decir una vez obtenido el documento más relevante, se extrae de el mismo el fragmento más relevante. Este procedimiento consiste en iterar por las primeras 30 palabras del texto del documento, en caso de que posea menor cantidad que 30 palabras se devuelve el mismo documento, luego si existe una ocurrencia de alguna palabra de la query, se aumenta el número de la variable **cant**. Al finalizar este ciclo entonces comienza otro, que ocurrirá mientras la cantidad de palabras del documento menos start sea mayor o igual que

30. Entonces todo lo que hace es “quitar” la primera palabra y “agregar” una al final. Entonces se pregunta si: La palabra quitada está en la búsqueda del usuario, entonces disminuye “cant”, si la palabra agregada está en la búsqueda entonces aumenta en 1 la variable cant, luego se compara con “temp” y si es mayor entonces se actualizan los índices (start y end) y se le asigna a temp el valor de cant. Finalmente se tienen los índices de donde se encuentra el snippet de mayor importancia, se recorre el documento en el fragmetno que cae dentro de estos índices y se concatena cada palabra en un string, dicho string es el que se retorna al final del método.

5.3 Método Evaluate

Recibe entre sus parámetros el array de **scores** creado en el método anterior y devuelve un array de objetos de tipo **SearchItem**. Este método itera por cada uno de los documentos de **Infos**, llama al método **Snippetplus** con cada documento, luego utilizando expresiones Lambda obtiene el título de cada documento y finalmente crea un objeto **SearchItem** con este título, el snippet y el score que esté guardado en la posición que señala la propiedad “index” del objeto **InfoDocument** en el que se está iterando actualmente.