Proyecto de Programación I

Jabel Resendiz Aguirre

October 14, 2023

1 Introducción

Como parte del I proyecto de la carrera de programación de la carrera Ciencias de la Computación ,que es una especie de buscador dentro de una base de dato de textos en formato ".txt" y esta es la explicación de mi proyecto Moogle!.Aunque las especificaciones iniciales están en archivo readme.md es válido resaltar que la carpeta MoogleServer contiene otra llamada Content,que será la que almacenara los textos y el vocabulario general de estos a su vez será principalmente para dominio del idioma español,y en otra medida de idiomas derivados del latín y anglosajones.

2 MoogleEngine

MoogleEngine es el nombre de la biblioteca que en diferentes clases se encarga de la implementación de la parte no gráfica del proyecto y de la lógica de los algoritmos y la matemática detrás de la búsqueda.Todas las clases de la misma están implementadas en csharp.Este proceso se desarrolla en dos fases imprescindibles,una primera encargada de leer los documentos,trabajar sobre sus respectivos vocabularios y encapsular toda la información posible para trabajar con la futura cadena,esto se ejecuta antes de que la app cargue la pagina para facilitar en medida una dinámica búsqueda.Esto es gracias a que se llama la clase Principal en el archivo Program.cs de MoogleServer. Para una mejor explicación del proyecto veamos un análisis de las clases de la de menor complejidad a la de mayor.

2.1 CreatedVocabulary.cs

Esta clase es la encargada de manipular los vocabularios de cada documento específico implementando.Algunas de las implementaciones que realiza son:

- Localizar la ruta de los documentos,almacenar sus títulos y leerlos.
- Crear el vocabulario de cada documento y uno más global escaneando los documentos para sustituir tildes,evitar signos no alfanuméricos y no almacenar palabras repetidas.Para encapsular esta información y mucha más específica de cada documento se ha creado una clase File y considerar los documentos como objetos de la misma.

2.2 File.cs

Esta clase como hemos explicado es la encargada de encapsular en un objeto toda la información relativa a cada texto tales como:

- Indice del documento
- Título del documento
- Un diccionario donde a cada palabra del vocabulario se le hace corresponder la cantidad de veces que aparece.
- El cuerpo del texto
- El mismo texto pero normalizado(definimos normalizar en la próxima clase)
- Un string snippet que es un fragmento del texto
- Una variable Score que será el peso de cada documento con respecto a la entrada
- Un diccionario con el respectivo peso de cada palabra con el query

2.3 Normalize.cs

Es la clase encargada de a través de varios métodos normalizar las cadenas de texto para su posterior trabajo,digamos tales como sustituir tildes y otros acentos de otros idiomas,eliminar signos no alfanuméricos, suprimir saltos de líneas,espacios en blancos para tener una alineación dentro de los textos que manipulamos.Además hay implementada un método para trabajar con los títulos que se van a imprimir en la página.

2.4 AplicatedMath.cs

Esta clase realiza trabajosmatemáticos tales como calcular tf*idf, normalizar vectores,y una que calcula similitud.He aquí donde empieza el primer de los principales análisis. El principal algoritmo de búsqueda que efectúo es el cálculo de tfidf.En el enfoque Frecuencia de Término(TF) lascoordenadas del vector documento d_j es representado como una función de conteo de términos con la longitud del documento.

$$f(x) = \begin{cases} 0 & \text{si la palabra no aparece} \\ (R)/(l) & \text{en otro caso} \end{cases}$$

donde

1. R es el numero de veces que se repite la palabra en el documento
2. l es la longitud del documento

La idea básica de la frecuencia inversa IDF es calcular en proporción a los documentos donde aparece el término t_j con respecto al número total de documentos.

$$IDF(t_j) = \log_{10}\left(\frac{N}{n_j - 0.5}\right)$$

1. N es el número de documento de la colección

2. n_j es la cantidad de documentos donde aparece el término t_j.(-0.5 es usado para evitar tener

$$log_{10}(1) = 0$$

);

Luego el peso de la palabra i dentro del documento j será la multiplicación del TF de la palabra dentro del documento y el IDF del valor de la palabra dentro del corpus de documentos.

EL método de normalizar vectores es para darle a los vectores una longitud de 1. A través de la formulas

$$\sqrt{\sum_{i=1}^{n} X_i^2}$$

donde X_i son los valores del vector. El método de similitud es para calcular un score para el documento en función del producto de ambos vectores normalizados,y aquellos vectores que den mayor valores es porque su similitud es mayor con la cadena inicial.

$$s(A, B) = \frac{\sum_i (A_i \cdot B_i)}{(\sqrt{\sum_i A_i^2}) \cdot (\sqrt{\sum_i B_i^2}) + 1}$$

siendo A y B dos vectores siendo A_i y B_i los componentes de ambos vectores.

Ademas cuenta con los metodos de Suggest ,Search y LevinshteinDistance encargados de dar una sugerencia a través de algoritmos de búsqueda de complejidad O(log2 N) y de recursion. Mejor explicados en la clase Principal y dentro del código

2.5 QueryWork.cs

Este método es para trabajar con el query(cadena inicial)

- Process Query es el método de crear el vocabulario en función de las palabras que pertenezcan al vocabulario global de los documentos.

- VectorQuery es el método que crea el vector query como un diccionario donde a cada palabra del vocabulario antes sacado su valor TF-IDF de acuerdo:TF(la frecuencia de la palabra dentro del query)y el IDF(como la cantidad de documentos en el que aparece la palabra),este último ya lo teníamos calculado. Este vector es normalizado seguidamente.

- Operator es un método público que se encarga de realizar operaciones de acuerdo a la presencia de operadores que no son mas que signos tales como:

    1. ! Operadores que advierte de que ningún documento con la palabra que le sucede debe ser devuelto.A toda palabra con ese operador se le sustituirá dentro del vector query su valor TF-IDF por el valor -1.
    2. $\wedge$ el operador opuesto al anterior,solo deben ser devueltos documentos que tengan la palabra que le sucede al signo.A toda palabra con ese operador se le sumará 100 a su TF-IDF

3. $*$ es operador prioridad que determina que la palabra que le sucede al operador aumenta en relevancia tanta como signos tenga,al TF-IDF se le agregara un valor equivalente a la cantidad de * por 1000.
4. $\sim$ este es un operador cercanía y a diferencia de los anteriores no es valor de una palabra sino del documento,o sea que se almacene dentro de un array de dobles donde en la posición i estará el valor correspondiente al incremento final del documento i

Para este operador se busca dentro del texto del documento actual la menor distancia(distancia es la cantidad de palabras que separan dos palabras dadas dentro de un texto)entre las palabras separadas por y el valor incremento del documento i será el valor de dividir 0.5/ (esa distancia menor).

- el método Snippet que cuenta dentro de un rango de 100 palabras donde más veces aparezca más palabras del query y lo guarda en el campo snippet de la clase file asociado por supuesto a cada documento
- Un método que ayuda al usuario a localizar las palabras que quería buscar dentro del snippet entre emogis

2.6 Principal.cs

Es el método encargado de unificar todos esos métodos y encaminar el trabajo para almacenar en un array los objetos SearchItem organizados por su score

- En su constructor se ejecutan los procesos relacionados con la creación de la matriz de documentos con su TF-IDF calculado y encapsulada esperando a trabajar con el query de ahí que se implemente en una primera fase antes de que el usuario pueda visualizar la página.Donde el método Calculate TF-IDF realizarél trabajo de asignarle a cada palabra un TF-IDF
- Una segunda fase que si dependerá del query y que se implementará una vez que el usuario de al botón buscar.Se crea el vocabulario del query;se calcula la similitud con cada uno de los vectores de la matriz de documentos pero no sin antes tener en cuenta los valores relativos a cada operador.
- Un método de Sugerencia para query donde las palabras no coinciden todas con palabras del vocabulario

1. método de la sugerencia para cadenas donde no hay una coincidencia fuerte dentro de los textos de la base de dato Explicación:
2. Busca en el vocabulario de palabras del corpus si la palabra esta: en caso de estarlo pues agregar la palabra al texto de la sugerencia. Si no llama al metodo Suggest de la clase AplicatedMath
3. En Suggest se encarga de buscar el rango de palabras cuya longitud oscile entre uno hasta 2 caracteres
4. Esta busqueda la hace a traves del metodo Search con algoritmo de Busqueda Binaria

5. LevinshteinDistance es el metodo que mide las palabras dentro de ese rango y devuelve la diferencia de caracteres entre cada palabra del rango y la actual tomada
6. Suggest termina viendo la palabra con menor valor de entre estas y guardando para la sugerencia

Luego que están escogidos los documentos que pasan esa prueba de operadores se les calcula la similitud con el vector query y se suma al valor de la similitud de ese documento,el valor incremento realtivo al símbolo de cercanía $\sim$ y la cantidad de palabras del vocabulario del query que coinciden con el vocabulario del documento.Se organizan los objetos de acuerdo a ese score total y se guardan en un array como objetos de SearchItem para luego la clase Moogle llame a la otra SearchResult con ese array de objetos SearchItem y los imprima en la página.

3 Conclusiones

Espero que os guste y quede a la altura de lo que buscaban,aunque pienso modificarlo más porque me ha inspirado bastante.Gracias.

Contents