Informe del primer proyecto de programación

Michell Viu Ramirez

Julio, 2023

Contents

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'* 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.

2 Clases

El proyecto modificado cuenta con tres clases diferentes:

2.1 Clase MétodosAdicionales

Es una clase estática que consta de dos métodos:

- **ArrayQuery** que recibe como parémetro un string query y devuelve un array de tipo string, su función en el proyecto es normalizar el string query así como convertirlo en un array de tipo string y eliminar términos que no son relevantes para la búsqueda como preposiciones, conjunciones y artículos ya que no nos interesa que en nuestro resultado de búsqueda aparezca un documento que solo contenga alguno de estos elementos (**OJO**:solo se han eliminado estas palabras para el idioma español e inglés).
- **SubString** que recibe como parámetros un array de tipo string, un int "inicio", un int "fin"y un string "termino", y retorna un string. Su función es crear el "snippet" del "término" buscado en caso de que aparezca en el documento, ya que su funcionamiento se basa en convertir desde la posicion "inicio" del array hasta la posicion "fin" del array en un string que será la porción donde aparece dicho término en el documento(el "inicio" y el "fin" se introduce en dependencia del tamaño que se quiera tener dicho snippet,más adelante se explica más detallado su calculo).

2.2 Clase Matriz

Consta de una propiedad de tipo double[,], nueve métodos que cada uno de ellos realiza una operación entre matrices y/o vectores. En su constructor recibe un matriz de tipo double (double[,]).

2.2.1 Métodos de la clase Matriz

- **SumaMatriz** recibe como parámetros dos matrices de tipo "Matriz", y devuelve una "Matriz" ,su función es relizar la suma entre dos matrices. En caso de que las matrices no cumplan las condiciones para poder realizar dicha operación se lanza una excepción advirtiendolo.
- **RestaMatriz** recibe como parámetros dos matrices de tipo "Matriz", y devuelve una "Matriz" ,su función es relizar la resta entre dos matrices. En caso de que las matrices no cumplan las condiciones para poder realizar dicha operación se lanza una excepción advirtiendolo.
- **MultiplicaMatriz** recibe como parámetros dos matrices de tipo "Matriz", y devuelve una "Matriz",su función es relizar la multiplicación entre dos matrices. En caso de que las matrices no cumplan las condiciones para poder realizar dicha operación se lanza una excepción advirtiendolo.
- **EscalarMatriz** recibe como parámetros una "Matriz" y un int "escalar" y devuelve una "Matriz", su función es multiplicar dicho escalar por la Matriz.
- **VectorMatriz** recibe como parámetros un array de tipo double y una "Matriz" y devuelve un array de tipo double, su función es multiplicar un vector por una matriz.
- **MultiplicaVectores** recibe como parámetros dos arrays de tipo double y devuelve un array de tipo double, su función es multiplicar dos vectores.
- **SumaVectores** recibe como parámetros dos arrays de tipo double y devuelve un array de tipo double, su función es sumar dos vectores.
- **RestaVectores** recibe como parámetros dos arrays de tipo double y devuelve un array de tipo double, su función es restar dos vectores.
- **MultiplicaEscalarVector** recibe como parámetros un array de tipo double y un int "escalar" y devuelve un array de tipo double, su función es multiplicar un escalar por un vector.

2.3 Clase Documentos

Esta clase consta de seis propiedades y de nueve métodos, la propiedad 'nombreDocumento' de tipo string que represeta el nombre del "Documento",la propiedad 'score' de tipo float que representa la relevancia de ese "Documento" para una búsqueda dada, la propiedad 'cantidadDePalabras' de tipo int que representa la cantidad de palabras del "Documento", la propiedad 'contenido' de tipo string[ ] que representa cada palabra del "Documento" y la propiedad 'snippet' de tipo string que representa una porción del "Documento" para una búsqueda dada y la propiedad 'palabrasTF' de tipo Dictionary(string,int) que representa la cantidad de veces que aparece una palabra en un "Documento".

En su constructor se recibe como parámetros un string ‘nombreDoc‘, un int ‘longitud‘ y un string[ ] ‘contenido‘, las propiedades nombreDocumento, cantidadDePalabras y contenido reciben en el constructor los valores de nombreDoc, longitud y contenido respectivamente,score se inicializa en 0, snippet se inicialeza en ‘””‘ y palabrasTF en un nuevo Dictionary(string,int). La propiedad ‘nombreDocumento‘ consta de un método ‘get‘ llamado ‘getNombre‘, la propiedad ‘score‘ consta de un método ‘get‘ y un método ‘set‘ llamados ‘getScore‘ y ‘setScore‘ respectivamente, la propiedad ‘cantidadDePalabras‘ consta de un método ‘get‘ llamado ‘getCantidadDePalabras‘, la propiedad ‘contenido‘ consta de un método ‘get‘ llamado ‘getContenido‘, la propiedad ‘snippet‘ consta de un método ‘get‘ llamado ‘getSnippet‘ y un método llamado ‘AddSnippet‘, y la propiedad ‘palabrasTF‘ consta de un método ‘get‘ llamado ‘getDictionary‘. Cada uno de estos métodos se utiliza tanto para obtener el valor de dicha propiedad o modificarlo en dependencia si posee el método o no. El método ‘AddSnippet‘ recibe un string ‘snippet‘ como parámetro y simplemente lo modifica por tanto devuelve void, para ello utilizamos una condicional if preguntando si ‘this.snippet‘== ”” de ser así significa que es el primero snippet agregado y por tanto ‘this.snippet‘ = ‘snippet‘ de lo contrario significa que ya existe otro ‘snippet‘ de otro término buscado y por tanto se lo agregamos con tres puntos suspensivos para diferenciar cada uno. Ej:

```
”””
if (this.snippet == ””)
this.snippet = snippet;
else
this.snippet += ”...” + snippet;
”””
```

3 Ejecución y funcionamiento del método Query

Lo primero que se hace en este método es crear una variable *‘archivos‘* de tipo string[ ] que se le asigna el valor que se obtiene del método *‘GetFiles‘* de la clase *‘Directory‘* que carga los archivos .txt que se encuentran en una ruta dada Ej:

```
”’
string[] archivos = Directory.GetFiles(fullPath, ”*.txt”);
”’
```

Posteriormente se crea una variable llamada ‘terminosBuscados‘ de tipo string[ ] que va a representar el valor del método *‘ArrayQuery‘* de la clase ‘MetodosAdicionales‘ recibiendo como parámentro a la ‘query‘(parámentro del método ‘Query‘), una variable llamada ‘resultadoDoc‘ que va a representar una ‘Lista‘ que va a contener objetos de tipo ‘Documentos‘ y una variable llamada ‘palabrasIDF‘ que va a contener un Dictionary(string,int) que recibe como key el término buscado y como value en cuántos documentos aparece ese término.

Luego se entra en un triple for, el primero(el más externo, utilizamos la variable int i = 0, aumentando en 1 a medida que avance el ciclo y se cambie

para el siguiente .txt) va a recorrer el array ‘archivos‘, inicialmente en este ciclo lo primero que se realiza es convertir el ‘archivo‘ en cuestión en una variable de tipo string llamada ‘contenido‘ usando el método ‘ReadAllText‘ de la clase ‘File‘ que recibe como parámeto el archivo que se esta analizando en ese momento. Luego se normaliza el string ‘contenido‘ convirtiendo todo el texto en minúsculas, eliminando carácteres especiales, remplazando las vocales con tilde por vocales sin tilde y quitando los saltos de línea. Ya con el contenido normalizado utilizamos el método ‘Split‘ para convertir el string ‘contenido‘ en un array de tipo string llamado ‘alltext‘, además se crea una variable llamada ‘doc‘ de tipo ‘Documentos‘ que recibe como parametros la variable ‘archivos[i]‘, la variable ‘alltext.Length‘ y la variable ‘alltext‘
Ej:
”’
var doc = new Documentos(archivos[i], alltext.Length, alltext);
”’
Además se crea una variable ‘flag‘ de tipo bool inicializada en ‘false‘ que posteriormente nos servirá para saber si el archivo contiene a ‘terminosBuscados‘.
Ahora entramos en el segundo ciclo for que va a recorrer el array ‘terminosBuscados‘(utilizamos la variable int j=0, aumentando en 1 a medida que avance el ciclo y se cambie para el siguiente término buscado), dentro de este ciclo creamos una variable de tipo string llamada ‘terminoBuscado‘ que tomará el valor de ‘terminosBuscados[j]‘,una variable de tipo int llamada ‘cont‘ inicializada en 0 que más adelante nos servirá como contador para saber cuantas veces aparece ‘terminoBuscado‘ en ‘alltext‘,además se crea una variable de tipo int llamada ‘aux‘ inicializada en -1 que nos servirá como bandera posteriormente en el calculo del ‘snippet‘.
Entramos en nuestro tercer y último ciclo for que va a recorrer el array ‘alltext‘(utilizamos la variable int k = 0, aumentando en 1 a medida que avance el ciclo), primeramente aplicamos una condicional if que va a recibir como parámetro ‘alltext[k].Equals(terminoBuscado)‘ ya que el método ‘Equals‘ devuelve un tipo bool en dependencia si ‘terminoBuscado‘ coincide exactamente con ‘alltex[k]‘, luego si eso devuelve true quiere decir que ese documento contiene a ‘terminoBuscado‘ y por tanto aumentamos en 1 a ‘cont‘ y dentro de la misma condicional procedemos a calcular el ‘snippet‘ de ‘terminoBuscado‘ en ese archivo (**OJO**: calculamos el ‘snippet‘ de cada ‘terminoBuscado‘ dejando tres puntos suspensivos entre cada snippet), para calcular el ‘snippet‘ utilizamos una condicional if para saber si ‘aux‘== -1, evidentemente entraremos a la condicional porque aux inicializa en dicho valor pero no importa porque ya estamos seguros de que ‘terminoBuscado‘ aparece en ‘alltext‘, inmediatamente cambiamos el valor de ‘aux‘ y le asignamos el valor ‘k‘ para que calcule el ‘snippet‘ de cada termino una sola vez y en su primera aparicion, para el calculo del ‘snippet‘ utilizaremos el método explicado anteriormente ‘SubString‘ de la clase ‘MetodosAuxiliares‘ para ello necesitamos cuatro parámetros, el primero será el array de string de donde se quiere extrar el ‘snippet‘ que en este caso es ‘alltext‘, el segundo será el int inicio que en este caso va a estar representado por el método ‘Max‘ de la clase ‘Math‘ que recibirá como parámetros 0 y

k-4(*queremos quedarnos para el ‘snippet‘ con 4 palabras antes y 4 palabras despues del ‘terminoBuscado‘*) utilizamos este método para asegurarnos que si el ‘terminoBuscado‘ se encuentra en las primeras posiciones no se vaya de rango ya que si es asi comenzará en la posición 0, análogamente para el tercer parámetro necesitamos un int ‘fin‘ que estará representado por el método ‘Min‘ de la clase ‘Math‘ que va a recibir como parámetros ‘doc.getContenido().Lenght()-1‘ y k+4, igual que para el parámetro anterior utilizamos este método para evitar irnos de rango, y nuestro cuarto parámetro será ‘terminoBuscado‘. Ej:
”’
aux = k;
int snippetStart = Math.Max(0, k - 4);
int snippetEnd = Math.Min(doc.getContenido().Length - 1, k + 4);
string snippet = MetodosAdicionales.SubString(alltext, snippetStart, snippetEnd, terminoBuscado);
”’
Luego guardamos dicho ‘snippet‘ en el documento en cuestión(‘doc‘) para el ‘terminoBuscado‘ con el método ‘AddSnippet‘ de la clase ‘Documentos‘.

El siguiente paso es actualizar el diccionario del objeto ‘doc‘ de tipo ‘Documentos‘ para ello utilizamos una condicional preguntando si ‘count‘ es mayor que 0 de esta manera estaremos seguros de que ‘terminoBuscado‘ aparece al menos una vez en dicho documento, si entramos dentro de la condicional ‘flag‘ será true ya que ese documento contiene al menos a un ‘terminoBuscado‘, posteriormente actualizamos el diccionario de ‘doc‘ con el método ‘getDictionary‘ de la clase ‘Documentos‘ y el método ‘Add‘ recibiendo como parámetros ‘terminoBuscado‘ y ‘count‘.

Ahora nos quedaría actualizar la variable ‘palabrasIDF‘ de tipo Dictionary ya que ese ‘terminoBuscado‘ aparece al menos en un documento. Para ello creamos una nueva variable de tipo int llamada ‘cantidaddedocumentos‘ que representara la cantidad de documentos en que aparece ‘terminoBuscado‘ para ello utilizamos una condicional, dentro de ella utilizamos el método ‘TryGetValue‘ de la clase ‘Dictionary‘, si entramos en la condicional aumentamos en 1 la variable ‘cantidaddedocumentos‘ y acutalizamos ‘palabrasIDF‘ asignandole a ‘terminoBuscado‘(key) el valor de ‘cantidaddedocumentos‘, en caso contrario sabremos que es el primer documento donde aparece ‘terminoBuscado‘ por tanto actualizamos ‘palabrasIDF‘ asignandole a ‘terminoBuscado‘(key) el valor 1.

Luego de terminar el segundo ciclo realizamos una condicional if preguntando por ‘flag‘ en caso de ser true sabremos que al menos un ‘terminoBuscado‘ aparece en ‘doc‘ por tanto lo agregamos a nuestra lista ‘resultadoDoc‘ con el método ‘Add‘ de la clase ‘List‘.

Por último nos quedaría recorrer la lista ‘resultadoDoc‘ y hacer el calculo del ‘score‘ de cada documento y luego organizarlos en orden descendiente para tener en la primera posición al documento que más ‘score‘ posea para esa búsqueda,

para ello utilizaremos la fórmula de TF(frecuencia del término)*IDF(frecuencia inversa), el TF se calcula dividiendo la cantidad de veces que aparece un ‘terminoBuscado‘ entre la cantidad de palabras de ese documento y el IDF se calcula mediante el logaritmo en base 2 de la cantidad de archivos entre la cantidad de archivos en que aparece dicho ‘terminoBuscado‘, luego calculamos la sumatoria del ‘TF*IDF‘ de cada uno de los ‘terminosBuscados‘ y ese valor representará el ‘score‘ del documento, para esto creamos dos variables de tipo int una se va a llamar ‘tf‘ y la otra ‘idf‘ y utilizamos el método ‘ForEach‘ de la clase ‘List‘ para recorrer la lista ‘resultadoDoc‘, luego creamos una variable ‘score‘ de tipo float y entramos en un ciclo for para recorrer cada uno de los ‘terminosBuscados‘ y hacer el calculo del ‘score‘, luego con el método ‘getDictionary‘ de la clase ‘Documentos‘ y el método ‘TryGetValue‘ de la clase ‘Dictionary‘ le asignamos valor al ‘tf‘ de terminosBuscados[i], igualmente con el diccionario ‘palabrasIDF‘ y el método ‘TryGetValue‘ le asignamos valor al ‘idf‘de terminosBuscados[i], luego usando una condicional if y preguntando si ‘tf‘ es mayor que cero realizamos el calculo del ‘score‘ con los términos ‘tf‘ e ‘idf‘ y se lo sumamos a la variable ‘score‘, en caso de no entrar en la condicional simplemente le sumamos 0 al ‘score‘ porque ese término no aparece en ese documente, luego de calculado el ‘score‘ total del documento los multiplicamos por la cantidad de palabras de la query que contiene ese documento, dandole más relevancia a los documentos que más cantidad de palabras de la query contienen, luego se lo introducimos al documento con el método ‘setScore‘ de la clase ‘Documentos‘.
Ej:
”’
int idf;
int tf;
resultadoDoc.ForEach($d \Rightarrow$

float score = 0;
int palabras;
for (int i = 0;$i\langle terminosBuscados.Length$; i++)

d.getDictionay().TryGetValue(terminosBuscados[i], out tf);
palabrasIDF.TryGetValue(terminosBuscados[i], out idf);
if $(tf\rangle 0)$

palabras++;
score += ((float)tf / d.getCantidadDePalabras()) * (float)Math.Log2((float)archivos.Length / idf);

else score += 0;

score=score+palabras;
d.setScore(score);
);

Posteriormente organizo la lista ‘resultadoDoc‘ con el método ‘Sort‘ de la clase ‘List‘ en dependencia del tamaño del ‘score‘.
Seguidamente creo un array de tipo ‘SearchItem‘ llamado ‘items‘ que tendrá como longitud el tamaño de la lista ‘resultadoDoc‘, luego mediante un ciclo for que recorre ‘resultadoDoc‘ le asigno a ‘items‘ los valores de ‘nombre‘, ‘snippet‘ y ‘score‘ mediante los métodos ‘getNombre‘, ‘getSnippet‘ y ‘getScore‘ respectivamente todos de la clase ‘Documentos‘.
Por último el método ‘Query‘ retorna nuevo objeto de tipo ‘SearchResult‘ que recibe como parámetros el array ‘items‘ y el string ‘query‘.

4 Observaciones

- Mediante su ejecución se le pueden añadir nuevos documentos a la carpeta Content.
- Se puede realizar la búsqueda de varios términos.
- El proyecto está probado con una base de datos de 100 MB de archivos .txt
- El score del documento representa la ‘RELEVANCIA‘ de ese documento para la búsqueda, por tanto pueden aparecer documentos que contienen menor cantidad de términos buscados (no es así generalmente) que otros que contengan más, esto de debe a la frecuencia con que aparezcan los elementos de la búsqueda en cada documento y la cantidad de palabras que contiene el documento.

****Espero que este informe haya sido útil y que el proyecto haya alcanzado las espectativas.****