Facultad de Matemática y Computación

PROYECTO DE PROGRAMACIÓN I

Moogle!

Juan Miguel Maestre Rodriguez

C-121

Clase Normalize

Para normalizar los texto del proyecto se crea la clase Normalize en la cual se implementa el método llamado Normalizing, a este se le pasara un valor de tipo “string” llamado “text”. Dentro del método en la linea 9 a la variable “text” se le aplica el método ToLower el cual se le asigna a la misma variable(Esto es porque el lenguaje CSharp es sensible a mayúsculas y minúsculas “Case Sensitive” .Osea este lenguaje no interpreta de la misma forma “Perro” y “perro”).

```
9        text = text.ToLower();
```

Análogamente se utiliza la función <name>.Replace, en este caso text.Replace (“ á”, “ é ”, “ í ”, “ó” “ ú ”) por las mismas pero sin tilde por la misma razón anterior (Case Sensitive) .

```
10       text = text.Replace("á", "a");
11       text = text.Replace("é", "e");
12       text = text.Replace("í", "i");
13       text = text.Replace("ó", "o");
14       text = text.Replace("ú", "u");
```

Por ultimo se utiliza :

```
16       text = Regex.Replace(text, @"[^ña-z0-9]", " ");
```

Esta linea es para que todo lo que sea los caracteres que no están incluidos en ña-z0-9 sean sustituidos por un espacio en blanco. Para esta función primero tenemos que importar la librería System.Text.RegularExpressions;

```
using System.Text.RegularExpressions;
```

CLase TF-IDF

Para calcular el **TF-IDF** del proyecto se utiliza esta clase, pero primero que todo se tiene que saber que es TF-IDF(Term Frequency-Inverse Document Frecuency) frecuencia de término – frecuencia inversa de documento, es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección. Esta medida se utiliza a menudo como un factor de ponderación en la recuperación de información y la minería de texto

- **TF**: Cantidad de veces que se repite una palabra en el documento, entre la cantidad de palabras que contiene el documento.

- **IDF**: Cantidad de documentos y Cantidad de documentos en los que aparece una palabra.

El primero (**TF**) se calcula como la cantidad de veces que se repite una palabra en el documento, entre la cantidad de palabras que contiene el documento.

$$TF = np/nd.$$

Donde :

- **np** es la cantidad de veces que se repite una palabra en un documento
- **nd** es la cantidad de palabras que contiene el documento

La Frecuencia Inversa de Documento(**IDF**) se calcula como Logaritmo en cuya base tiene la cantidad de documentos en los que aparece una palabra en específico y en su argumento la cantidad total de documentos.

$$IDF = \log t/n$$

Donde:

- **n** es la cantidad de documentos en los que aparece la palabra.
- **t** es la cantidad total de documentos

DataBase

En esta clase se forma la base de daos de Moogle Project. Esta clase contiene el método llamado “Loading” al cual se le pasa una variable llamada “path” que es de tipo “string” a la cual se le asigna el texto “Content”

```
17      public static void Loading (string path = "Content")
```

En la línea 19 al array “directions” se le asigna el directorio de cada documento con la función Directory.GetFiles(Path.Join(“..” <variable>),””,SearchOption.AllDirectories);. A esta función se le pasó la variable path que tiene asignada la cadena de caracteres “Content”. Entonces esta encuentra la carpeta “Content” y después le asigna a cada posición del array la dirección de cada documento.

```
19      directions = Directory.GetFiles(Path.Join("..",  path),"",SearchOption.AllDirectories);
```

En las líneas 20, 21, 22 a los arrays declarados anteriormente “nameDocuments” , “text” , “allwords” se le asigna la misma longitud de “directions” con la funcionalidad <en este caso directions>.Lenght

```
20      nameDocuments = new string[directions.Length];
21      text = new string[directions.Length];
22      allwords = new string[directions.Length][];
```

En la línea 23 al array de diccionario que se había creado en con anterioridad le asignamos la misma longitud de “directions” respectivamente .

```
23      wordsDocuments = new Dictionary<string, float>[directions.Length];
```

En la línea 25 se creó un for en el cual se va iterando desde la primera posición del array “directions” por toda su longitud

```
25      for (int i = 0; i < directions.Length;i++)
```

En la linea 27 en cada iteración del for se le asigna al array nameDocuments en cada posición los nombres de cada documento guardado en cada posición del array “directions”. Esto es gracias a la función Path.GetFileNameWithoutExtension(<en este caso directions[i])

```
27          nameDocuments[i] = Path.GetFileNameWithoutExtension(directions[i]);
```

En la linea 28 se aplica la función File.ReadAllText(<en este caso directions[i]>). Este método abre los archivos correspondientes a los directorios guardados en cada

posición del array “directions”, lee todo el texto de cada archivo y lo devuelve como una cadena. Después se le aplica el método “Normalizing” que está en la clase “Normalize” esto para normalizar el texto(En el sección de la clase Normalize se explica detalladamente cual es el objetivo de la misma) y después esto se guarda en cada posición del array text declarado al principio de la clase.

```
28          text[i] = Normalize.Normalizing ( File.ReadAllText (directions[i]));
```

En la línea 29 a cada posición del array “text” que contiene los textos de los documentos se normalizan: Primero con la función StringSplitOptions.RemoveEmptyEntries se eliminan los espacios en los textos y después con el método .Split se separa cada palabra por espacios en blanco y esto se le asigna al array allwords declarado en el principio.

```
29          allwords [i]= text[i].Split(" ",StringSplitOptions.RemoveEmptyEntries);
```

La línea 30 tiene como objetivo que al ejecutar el programa no presente errores y diga qu está vacío.

```
30          wordsDocuments[i] = new Dictionary<string, float>();
```

De la línea 32 hasta la 39 se crea un foreach donde se va iterando por cada palabra de cada documento contenida en cada posición del array allwords. En la línea 34 se crea una condición donde si la palabra que está en los diccionarios de cada posición del array wordsDocuments está repetida entonces se suma 1 a un contador. Por cada repetición Esto se hace con el objetivo de saber cuantas veces se repite cada palabra en un documento. En caso que la palabra se encuentre solo una vez en el documento se le asigna 1 .

```
32          foreach (string word in allwords[i]) {
33
34              if (wordsDocuments[i].Keys.Contains(word)) {
35                  wordsDocuments[i][word] += 1;
36              } else {
37                  wordsDocuments[i].Add(word, 1);
38              }
39          }
```

De la línea 41 hasta la 44 se crea otro foreach donde se va iterando por cada posición del array allwords y entonces vamos a llamar al método “TF” contenido en la clase TF-IDF y le vamos a pasar cada diccionario de cada posición del array “allwords” que contiene cada palabra del documento con la cantidad de veces que se repite en el mismo y también le vamos a pasar el valor de la cantidad de palabras que contiene el diccionario(Sé conoce que es un poco complejo de entender pero básicamente esto lo que hace es substituir el valor de repetición que contenía cada palabra del diccionario por su TF(En la clase se TF-IDF se explica qué es esto)).

```
41          foreach ( var key in wordsDocuments[i].Keys)
42          {
43              wordsDocuments [i][key] = TF_IDF.TF(wordsDocuments[i] [key] , allwords[i].Length);
44          }
```

Clase Snippet

Para introducir el Snippet en el proyecto se crea la clase publica Snippeten la cual se implementa el método llamado ShowWords a este se le pasará un valor de tipo "string" llamado "text". Dentro del método se crea una variable llamada "result" donde: Si la longitud del texto tiene mas de 100 caracteres entonces en la variable se guardara un fragmento de hasta 100 caracteres, si no tiene una cantidad de caracteres superior a lo indicado anteriormente entonces se guardara el texto en su totalidad. Para poder guardar el fragmento de texto se utiliza el método [<palabra>.Substring(<inicio> , <final>)] .

```
1
2   namespace MoogleEngine;
3
    1 reference
4   public class Snippet
5   {
        1 reference
6       public static string ShowWrods(string text){
7           string result = "";
8
9           if (text.Length > 100) {
10              result = text.Substring(0, 100);
11          } else {
12              result = text;
13          }
14
15
16          return result;
17      }
18
19  }
```

Clase Score

Esta clase tiene como objetivo evaluar el nivel de importancia que tiene un documento según la búsqueda realizada. Para ello se crea un método público llamado "Ranking" se le pasará dos arrays uno de tipo string "modify" y otro de tipo float "IDF".

```
5       public static float [] Ranking (string [] modify , float [] IDF)
```

En la línea 7 se crea un array de tipo float "ranking" al cual se le asigna el mismo tamaño del array "directions" creado en la clase DataBase. Para ello se utiliza (DataBase.directions.Lenght).

```
7    float [] ranking = new float [DataBase.directions.Length];
```

En la línea 8 se crea una variable de tipo float llamada “tfxidf” a la que se le asigna temporalmente valor 0.

```
8        float tfxidf = 0;
```

En la línea 9 se crea un for en el que se se va iterando desde la posición 0 hasta el final del array de diccionarios “wordsDocuments” .

```
9        for (int i = 0; i < DataBase.wordsDocuments.Length; i++)
```

En la línea 11 se crea una variable entera “j” y se le asigna valor 0.

```
11            int j = 0;
```

De la línea 12 a la 24 se crea un foreach en el que vamos a iterar por cada “word” que contiene el array “modify” anteriormente declarado. Después se crea una condición donde cada diccionario del array de diccionarios “wordsDocuments” creado en la clase DataBase,si contiene la “word” de “modify” entonces se le agrega su nivel de importancia y coincidencia con “modify”(Esto es lo que se llama TF- IDF), ese valor se le asigna a la variable “tfxidf”. En caso de que no que no contenga la(s) palabra(s) se le resta valor. En la linea 27 se le asigna a cada posición de ranking el valor de “tfxidf” del documento en la posición que le corresponde. Y por ultimo en la línea 28 se vuelve a igualar a 0 la variable “tfxidf” para que su valor vuelva a reiniciarse y no hayan errores.

```
12        foreach (var word in modify )
13        {
14            if(DataBase.wordsDocuments[i].ContainsKey(word))
15            {
16                tfxidf = tfxidf + DataBase.wordsDocuments[i][word] * IDF[j];
17                j++;
18
19            }
20            else {
21                tfxidf /= 2;
22                j++;
23            }
24
25            }
26
27        ranking [i] = tfxidf;
28        tfxidf = 0 ;
29    }
```

Clase Moogle

Esta es la clase principal de Moogle Project la cual se encarga de correr el programa. En las líneas 6 y 7 se crean dos diccionarios con los cuales se trabajará a continuación:

Posteriormente se crea el método ¨Query" el cual mostrará los resultados de los ingresado por el usuario en el Programa, a este se le pasa una variable de tipo string¨llamada "query¨ .

```
12      string[] modify = query.Split(" ",StringSplitOptions.RemoveEmptyEntries);
```

En la línea 14 y 15 creamos dos arrays del mismo tamaño que ¨modify¨, el primero "count" va a ser un contador donde vamos a guardar la cantidad de documentos en los que aparece cada palabra de la query (Ej: Harry Potter... En la cantidad de documentos que se encuentra "Harry", ese valor se guardará en la posición (0) de "count"). En el segundo "IDF" se almacenará los IDF de cada palabra de la query en su posición correspondiente

```
14      int[] count = new int [modify.Length];
15      float [] IDF = new float [modify.Length];
```

De la línea 17 a la 25 se crean dos for, el primero itera desde la primera hasta la última posición de "modify" y el segundo igualmente pero de "text" contenido en la clase DataBase. Este bloque de código tiene como objetivo que : si la palabra de la "query" está contenida en un documento se le suma 1 al contador("count" array que se había declarado anteriormente, importante recordar que este tiene el mismo tamaño que "modify" por lo tanto almacenará la cantidad de repeticiones en la misma posición respectivamente) .

```
17      for (int i = 0; i < modify.Length; i++)
18      {
19          for (int j = 0; j < DataBase.text.Length; j++)
20          {
21              if (DataBase.allwords[j].Contains(modify[i]))
22              {
23                  count [i] += 1;
24              }
25          }
```

De la línea 28 a la 31 se crea otro for el cual va a iterar desde la primera hasta la última posición de "modify". Este bloque de código tiene como objetivo calcular los IDF de la "query". Para ello se llama al método "IDF" de la clase "TF-IDF" donde la cantidad de documentos será directions. Lenght y la cantidad en los cuales aparece la palabra es el contador(count).

```
28      for (int i = 0; i < modify.Length; i++)
29      {
30          IDF [i] = TF_IDF.IDF(DataBase.directions.Length, count [i]);
31      }
```

En la línea 33 se crea un array de tipo tipo float llamado ranking, esta linea tiene como objetivo guardar "Score"(el nivel de concordancia de un documento con la

query) de cada documento. Para ello se llama al método “Ranking” de la clase “Score” y se le pasan los arrays “modify” e “IDF”.

```
33          float[] ranking = Score.Ranking(modify, IDF);
```

En la línea 35 se inicializa el diccionario para que no presente un error al ejecutar el proyecto.

```
35          orderby = new Dictionary<string,string >();
```

De la línea 36 a la 40 se crea un for en el cual se va iterando sobre el array ranking con el objetivo de llenar los diccionarios ”order” y ”orderby”. En la llave del diccionario ”order” guardamos el nombre de cada documento correspondiente al valor que contiene en el ranking y en el diccionario ”orderby” guardamos el diccionario ”order” y en la llave se guarda el cuerpo del documento correspondiente.

```
36      for (int i = 0; i < ranking.Length; i++)
37      {
38          order [ranking[i]] = DataBase.nameDocuments[i];
39          orderby.Add(order[ranking[i]], DataBase.text[i]);
40      }
```

Las líneas 42 y 43 tienen como objetivo ordenar el ranking de los documentos, el primer método lo que hace ordenar los escores de menos a mayor y el segundo invertir el orden, para que sea de mayor a menor, esto es para que cuando se haga la búsqueda muestre los documentos que mayor coincidencia tienen con la query se muestren de primeros .

```
42          Array.Sort(ranking);
43          Array.Reverse(ranking);
```

El bloque de código siguiente tiene como objetivo que: Si el mayor “score”(importancia de un documento con la query) es 0 entonces devolverá un mensaje.

```
46      if (ranking[0] == 0){
47
48          SearchItem[] item = new SearchItem[1] {
49          new SearchItem("Estas inflando", "organizate palomo", 0.9f)
50          };
51          return new SearchResult(item, query);
52      }
```

En caso que el “score” sea distinto de 0 entonces se ejecuta el for el cual en la variable “snippet” se guardará un fragmento de texto del documento que el usuario solicitó. Para ello se llama la Clase “Snippet” específicamente al método “ShowWords” y se le

pasa el cuerpo de los documentos que es el diccionario “orderby”. Posteriormente a la lista “item” le agregamos el nombre de los textos, el snippet y su score.

```
59      for (int i = 0; i < 5; i++)
60      {
61          if (ranking[i] != 0)
62
63          {
64              string snippet = Snippet.ShowWrods(orderby[order[ranking[i]]]);
65              items.Add(new SearchItem(order[ranking[i]], snippet, ranking[i]));
66          }
67      }
```

Finalmente en la linea 69 se devuelven los resultados

```
69          return new SearchResult(items.ToArray(), query);
```