Proyecto Moogle!

Estudiante: Claudia Hernández Pérez

Grupo: C-122

El proyecto consta de 6 clases funcionales y una adicional con contenido relacionado con el curso de Álgebra I sobre operaciones con matrices. A continuación se dispone a explicar dichas clases:

- *ProcessDocuments*

Esta clase se encarga de procesar los documentos de la base de datos antes de comenzar la búsqueda. Se llama al método en la línea anterior a ***app.Run()*** para realizar esta operación una sola vez antes de que arranque el programa y esté listo para ejecutarse (fig 1.1).

```
26  MoogleEngine.ProcessDocuments.LoadDocuments();
27  app.Run();
```

Fig. 1.1 Líneas de la clase Program de la carpeta MoogleServer

¿Pero a qué le llamamos “procesar los documentos”?

```
5       //Array con las direcciones de los documentos
        6 references
6       private static string[]? docs { get; set; }
7       // Diccionario con los nombres de los documentos y las palabras sin repetir por documentos con la cantidad de veces que aparecen
8       // en el documento
        3 references
9       public static Dictionary<string, Dictionary<string,int>>? documentWords { get; set; }
10      // Diccionario con los nombres de cada documento con las palabras del documento
        4 references
11      public static Dictionary<string, string[]>? allWords { get; set; }
12      // Diccionario con los nombres de cada documento con el texto normalizado del documento
        3 references
13      public static Dictionary<string, string>? files { get; set; }
14      // Diccionario con los nombres de cada documento con el texto original del documento
        3 references
15      public static Dictionary<string, string>? fileswhitoutnormalize { get; set; }
```

Fig. 1.2 Líneas de la clase ProcessDocuments de la carpeta MoogleEngine

Primeramente, se accede al contenido de la carpeta “Content” (que estará dentro de la carpeta moogle-main), independientemente de la ubicación en el dispositivo de la carpeta moogle-main, se guardan las direcciones de los documentos en el array *docs* y con esta información se acceden a los nombres de cada documento con la función predeterminada `Path.GetFileNameWithoutExtension(string path)` y al texto de cada documento con `File.ReadAllText(string path)`. Con el cuerpo del documento y con la función `string.Split(string separator)` se crea un array donde cada posición es una palabra. Luego

con estos datos guardados, en un ciclo `foreach`, se itera cada palabra por documento para determinar la cantidad de veces que aparece (fig. 1.2).

- *Documents*

Esta clase está dirigida al trabajo con los documentos, se "normalizan". Es decir, en caso de que las búsquedas se realicen en español, se eliminan las tildes para evitar errores ortográficos. Además, independientemente del idioma, se eliminan los signos de puntuación y cualquier otro símbolo ajeno al alfabeto (de la "a" hasta la "z", incluyendo la "ñ") y los números (fig. 1.3).

```
8       // Función para normalizar los textos y hacer mas cómoda la búsqueda evadiendo errores de ortografía y símbolos innecesarios
        2 references
9       public static string Normalize(string text)
10      {
11          text = text.ToLower();
12
13          text = text.Replace("á", "a");
14          text = text.Replace("é", "e");
15          text = text.Replace("í", "i");
16          text = text.Replace("ó", "o");
17          text = text.Replace("ú", "u");
18
19          text = Regex.Replace(text, @"[^a-zñ0-9]", " ");
20
21          return text;
22      }
```

Fig. 1.3 Líneas de la clase Documents de la carpeta MoogleEngine

Se encuentra además esta otra función para seleccionar el "snippet" (pedazo breve de un texto), que se explicará su funcionamiento (fig. 1.4).

```
24      public static string Snippet(string text, string textwhitoutnormalize, string word) {
25          //text texto normalizado
26          //textwhitoutnormalize texto sin normalizar
27          //word palabra con mayor IDF en el texto
28
29          string? snippet = null;
30
31          int index = text.IndexOf(word); // Índice de la primera ocurrencia de la secuencia de caracteres de la palabra
32          while (snippet == null) {
33              if ((index == 0
34                  || text.Substring(index - 1, 1) == " "
35                  )
36                  && (index + word.Length == text.Length
37                  || text.Substring(index + word.Length, 1) == " ")) // Verificar que sea exactamente la palabra
38                  {
39                  if (index < 150) { // Imprime los primeros caracteres hasta la primera ocurrencia de la palabra
40                      snippet = textwhitoutnormalize.Substring(0, index);
41                  } else if (index > 150) { // Imprime 150 caracteres antes de la primera ocurrencia de la palabra hasta esta
42                      snippet = textwhitoutnormalize.Substring(index - 150, 150);
43                  }
44                  // Concatena la otra mitad del snippet
45                  if (((text.Length - 1) - index) < 150) { // Imprime desde la palabra hasta el final del documento
46                      snippet += textwhitoutnormalize.Substring(index);
47                  } else if (((text.Length - 1) - index > 150)) { // Imprime desde la palabra y 150 caracteres después
48                      snippet += textwhitoutnormalize.Substring(index, 150);
49                  }
50              } else { // Si llega aquí es porque no era exactamente la palabra y se le indica que continúe hasta la próxima ocurrencia
51                  index = text.IndexOf(word, index + 1);
52              }
53          }
54
55          return snippet;
56      }
```

Fig. 1.4 Líneas de la clase Documents de la carpeta MoogleEngine

Con la función `string.IndexOf(string word)` se busca la ocurrencia de la secuencia de caracteres que se pase como parámetro, que en este caso es la palabra con mayor valor de IDF en el documento previamente calculado en la clase Moogle. Esta función trabajará en el texto normalizado para encontrar la palabra en cuestión, para ello se ponen condiciones: para evitar

que esté en el medio de una palabra diferente, se comprueba que el caracter anterior a la ocurrencia de caracteres sea un espacio en blanco o que el índice de la primera ocurrencia sea 0, que eso se traduce a que sea la primera palabra del texto y se verifica que el caracter después del índice más la cantidad de letras de la palabra menos 1 (porque si no se sumaría la primera letra de la palabra dos veces) sea un espacio en blanco o sea la última palabra del texto. Si se incumple una de las dos significará que estará como prefijo o sufijo de otra palabra.

- *Matrix*

El valor de ***"relevancia"*** de una palabra está dado por el cálculo de su ***TF*** (Term Frequency) por su ***IDF*** (Inverse Document Frequency), para ello se ha utilizado la fórmula:

$$\frac{nd}{Cd} \cdot \log\left(\frac{T}{N}\right)$$

Donde,
nd es la cantidad de ocurrencias de una palabra en un documento,
Cd es la cantidad total de palabras en el documento,
T es la cantidad total de documentos,
N es la cantidad de documentos en los que aparece la palabra

En esta clase se realiza el cálculo, en el momento de la búsqueda, de ***TF-IDF*** por cada palabra de la query que se entra como parámetro en la función `QueryTF_IDF`.

```
34          // Cantidad de documentos en los que aparece la palabra
35          float count = 0;
36          foreach (string document in documents.Keys) {
37              if (documents[document].ContainsKey(query)) {
38                  count += 1;
39                  Dictionary<string,int> word = new Dictionary<string, int>();
40                  word.Add(query, documents[document][query]);
41                  reduceDocuments.Add(document, word);
42              }
43          }
44
45          IDFValue = Matrix.IDF(allFiles, count);
46
47          foreach (string key in reduceDocuments.Keys) { // Iterar los documentos en los que aparece la palabra para calcular su TF-IDF
48              TF_IDF = new Dictionary<string,float>();
49              int total = words[key].Length; // Cantidad de palabras del documento
50              // Valor de TF la palabra en cada documento
51              float TFValue = TF(reduceDocuments[key][query], total);
52              TFValue *= IDFValue; // Calcular el TF-IDF
53              TF_IDF.Add(query, TFValue);
54
55              WordTF_IDF.Add(key, TF_IDF);
56          }
```

Fig. 1.5 Líneas de la clase Matrix de la carpeta MoogleEngine

En el primer foreach se iteran los documentos para conocer en cuántos aparece la palabra para calcular después el IDF. Luego, se iteran los documentos seleccionados en los que sí aparecía la palabra y se calcula el TF en cada uno. Finalmente se calcula la multiplicación, obteniendo el valor de "relevancia" de la palabra.

- *Moogle*

Esta es la clase principal del programa, donde comienza el proceso de búsqueda. Comienza en el momento en que se recibe la ***query***. Primeramente, se le hace el mismo "tratamiento" que a los documentos para que coincidan y exista uniformidad. Además, se implementó una forma de reconocer si se repiten palabras en la query para reducir el tiempo de la búsqueda, si existen repeticiones de palabras en la query, solo se calculará su valor de TF-IDF una vez (fig. 1.6).

```
18          // Normalización de la query para comenzar la búsqueda
19          query = Documents.Normalize(query);
20          // Convertir la query en un array con cada palabra
21          string[] arrQueryTemp = query.Split(" ", StringSplitOptions.RemoveEmptyEntries);
22          // Array con las palabras de la query sin repetir
23          string[] arrQuery = arrQueryTemp.Union(arrQueryTemp).ToArray();
24
25          // Diccionario que contiene las palabras de la query y cuantas veces se repiten
26          Dictionary<string,int> repeat = new Dictionary<string, int>();
27
28          // Relacionar cada palabra con la cantidad de veces que aparece en la query
29          foreach (string word in arrQueryTemp) {
30              if (repeat.ContainsKey(word)) {
31                  repeat[word] += 1;
32              } else {
33                  repeat.Add(word, 1);
34              }
35          }
```

Fig. 1.6 Líneas de la clase Moogle de la carpeta MoogleEngine

Luego de terminado esto, se iteran las palabras sin repetir de la query y se calculan su TF-IDF en los documentos en los que aparece, pero para conseguir el ***score*** por documento se necesita la suma de estos valores. Para hacerlos coincidir se crearon arrays por cada palabra de la query de forma tal que se establece una correspondencia entre el orden de los documentos con los índices. Este array queda de la siguiente manera, si la palabra está contenida en el documento que se está analizando se encontrará el valor de TF-IDF correspondiente a esa palabra en ese documento, de forma contraria el valor será 0.

```
71          // Cantidad que se le va a restar al score del documento por cada palabra de la query que no aparezca
72          int resta = 10 * arrQueryTemp.Length;
73          for (int i = 0; i < TF_IDFs.Count; i++) { // Itera por cada palabra de la query
74              for (int j = 0; j < TF_IDFs[i].Length; j++) { // Itera por cada documento
75                  if (TF_IDFs[i][j] != 0) { // Verificar si la palabra esta en el documento
76                      // Si está se suma el valor de TF-IDF multiplicado por la cantidad de veces que aparece en la query
77                      suma[j] += TF_IDFs[i][j] * repeat[arrQuery[i]];
78
79                      if (wordIDF[i] > max[j]) { // Verificar cuál es la palabra con mayor valor de IDF en el documento
80                          max[j] = wordIDF[i]; // Se actualiza el valor máximo
81                          indexquery[j] = i; // y la posición i de la palabra de la query con ese valor
82                      }
83                  } else {
84                      suma[j] -= resta;
85                  }
86              }
87          }
```

Fig. 1.7 Líneas de la clase Moogle de la carpeta MoogleEngine

Esta forma facilita la suma, sobre todo porque lo interesante es devolver prioritariamente los documentos que más palabras de la query contenga, así si hay un 0 en la posición del array será cómodo identificar que el documento no contiene una de las palabras. Si esto pasa, es conveniente modificar este score para que tenga un valor menor. Además, se aprovecha este ciclo para reconocer la palabra con mayor valor de IDF por documento para el snippet, como se había mencionado anteriormente (fig. 1.7).

Finalmente, se procede a llenar el array ítems de tipo SearchItem, que contendrá los títulos de los documentos y los snippets correspondientes, en orden de relevancia acorde a la query (fig. 1.8).

```
144             if (score.Length == 0) { // Si no hubo coincidencias
145                 Array.Resize(ref items, 1);
146                 items[0] = new SearchItem("Búsqueda no encontrada", "intenta algo más", 0);
147 
148             } else { // Llenar el array items con las coincidencias, su snippet y el score por documento
149                 int k = 0;
150                 for (int i = 0; i < score.Length; i++) {
151                     foreach (string title in scoreDocuments[score[i]]) {
152                         items[k] = new SearchItem(title, snippet[k], score[i]);
153                         k += 1;
154                     }
155                 }
156             }
```

Fig. 1.8 Líneas de la clase Moogle de la carpeta MoogleEngine

En el caso excepcional en que la búsqueda sea vacía, es decir, que no se escriba ninguna palabra se imprimirá esto en pantalla (fig. 1.9).

```
162         } else { // Si la búsqueda es vacía
163             SearchItem[] items = new SearchItem[1];
164             items[0] = new SearchItem("Búsqueda inválida", "intenta algo más", 0);
165             return new SearchResult(items, query);
166         }
```

Fig. 1.9 Líneas de la clase Moogle de la carpeta MoogleEngine

- *SearchItem y SearchResult*

Estas clases estaban implementadas por defecto y permiten el funcionamiento correcto del programa, las cuales no fueron modificadas.

- *Adicionales*

Adicionalmente, implementé que al devolver los resultados de la búsqueda imprimiera el tiempo que había tardado la búsqueda, así como los documentos que contienen al menos una palabra de la query. Además, de que se puede buscar dando click en el botón buscar o presionando la tecla “enter” (fig. 1.10).

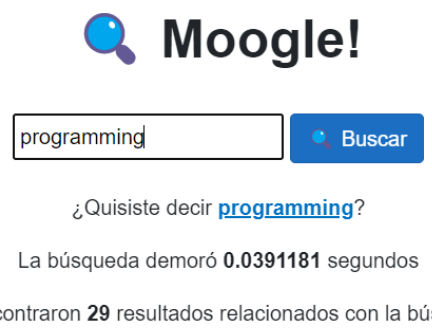

Fig. 1.10 Interfaz gráfica del Moogle!