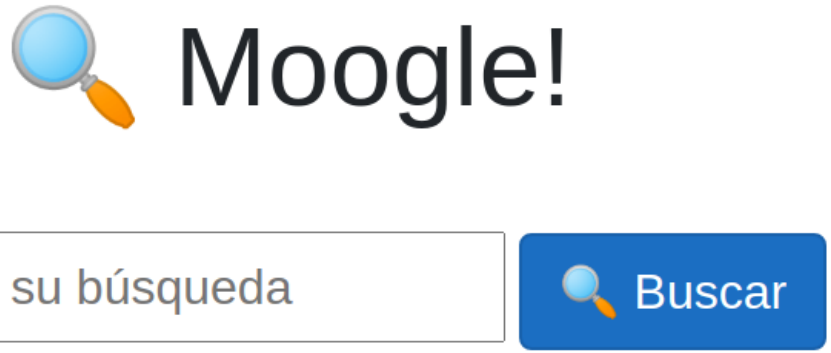

Raciel Pupo Santos

C-111

Objetivo del programa:

- Mostrar un conjunto de archivos con sus titulos y un fragmento de cada uno. Todo esto oraganizado de tal manera q en la primera posicion se va a encontrar el archivo q contiene mas veces la busqueda hecha por el usuario. Tambien si la busqueda realizada no se encuentra en ningun archivo, el programa le brindara una sugerencia al usuario con las palabras mas semejantes a la busqeda hecha por el usuario.

A continuacion se explicara paso a paso como el programa realiza su objetivo.

- Primeramente Para poder procesar la base de datos se tiene q introducir la ruta de la carpeta q contiene los documentos donde se indica.

Modificar en la clase SearchEngine y clase Moogle.

```
public static string directoryPath = //ruta de la carpeta con los documentos;
```

- Luego en la clase SearchEngine la cual representa al buscador se procesa la base de datos de documentos junto con la sugerencia y devuelve el conjunto de elementos q contiene los titulos, los fragmento y los scores q representan la importancia de cada elemento y por los cual se puede organizar de mayor a menor. En la siguinte imagen se puede ver al metodo de esta clase q es el encargado de devolver dicho conjunto.

```
    // Método para realizar una búsqueda y retornar una lista con los resultados de la búsqueda
public IList<SearchItem> Search(string query)
{
    // Eliminar los signos diacritics del query
    query = Document.RemoveDiacritics(query);

    // Calcular TF-IDF score para cada documento y su palabra con el query
    var numDocuments = _documents.Count;
    var queryWords = query.Split();
    var scores = new Dictionary<Document, double>();
    foreach (var document in _documents)
    {
        var score = 0.0;
        foreach (var word in queryWords)
        {
            score += document.CalculateTfIdf(word, _documentFrequencies, numDocuments);
        }
        scores[document] = score;
    }

    // Almacenar los documentos por score y generar el  resultado de la búsqueda
    var results = new List<SearchItem>();
    foreach (var kvp in scores.OrderByDescending(kvp => kvp.Value))
    {
        if (kvp.Value != 0)
        {
        var document = kvp.Key;
        var score = kvp.Value;
        var snippet = document.GenerateSnippet(query);
        results.Add(new SearchItem(document.Title, snippet, (float)score));
        }

    }
    return results;
}
```

- En esta clase tambien se devuelve la sugerencia q se le va a mostrar al usuario en caso de q la busqueda q haya realizado no aparezca en ningun documento de la base de datos

```
// Método para sugerir la mejor sugerencia basada en los resultados de la búsqueda
public static (string, string []) Suggest(List<string> quer1, string query, List<string> uniqueWords)
{
    //array para guardar los cálculos de levenshtein
    double[] cl = new double[uniqueWords.Count];
    //array para guardar las palabras que tengan mayor semejanza
    string[] sugerencias = new string[quer1.Count];
    //array para guardar las sugerencias con los operadores
    string[] sug_op = new string[sugerencias.Length];

    double distancia;
    for (int i = 0; i < quer1.Count; i++)
    {
        for (int j = 0; j < uniqueWords.Count; j++)
        {
            //comparando la palabra del query con la lista de todas las palabras de los documentos
            distancia = LevenshteinDistance.Compute(quer1[i], uniqueWords[j]);
            //almacenar el el array cl todas las distancias calculadas
            cl[j] = distancia;
        }
        //guardar las palabras que tengan menor distancia de levenshtein
        sugerencias[i] = uniqueWords[MenorDist(cl)];
    }
```

Clase Document:

- En esta clase devuelve el score(relevancia) de una palabra en los documentos y desde esta clase es donde se genera el fragmento de texto q se va a mostrar en la interfaz grafica. Este fragmento como se puede apreciar en la imagen va a contener a la palabra com mayor importancia de la busqueda hecha por el usuario.

```
30
31      // Método para generar un snippet de los documentos q contienen el query
32      public string GenerateSnippet(string query)
33      {
34          var regex = new Regex($"\\b({query})\\b", RegexOptions.IgnoreCase);
35          var match = regex.Match(Text);
36          if (match.Success)
37          {
38              var start = Math.Max(0, match.Index - 50);
39              var end = Math.Min(Text.Length, match.Index + 50);
40              return Text.Substring(start, end - start);
41          }
42          else
43          {
44              return Text.Substring(0, Math.Min(Text.Length, 100));
45          }
46      }
47
```