Moogle!

Definición general del Proyecto:

Moogle! es una aplicación cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos. La funcionalidad principal del proyecto, así como los requerimientos del mismo, se informan en el Readme.md que se encuentra con el proyecto.

Arquitectura del sistema y descripción de procesos:

```
class Utils:
```

Esta clase contiene 4 métodos:

1. **`EnumerateFiles`**: como su nombre indica emplea la función Directory.EnumerateFiles(string) que devuelve una colección enumerable de nombres completos (con sus rutas de acceso) para los archivos en el directorio especificado por la ruta de acceso absoluta o relativa al directorio que se va a buscar. Esta cadena no distingue entre mayúsculas y minúsculas. Dicha colección es convertida en array.
2. **`LoadDocuments`**: emplea la función Directory.GetCurrentDirectory() que devuelve una cadena que contiene la ruta de acceso absoluta del directorio de trabajo actual y no termina con una barra diagonal inversa (\), se le pasa la ruta de nuestro contenedor de archivos. Este método retorna un llamado al método Engine de la clase VectorialModel.
3. **`TokenizeWord`**: recibe una palabra y la normaliza utilizando Char.IsLetterOrDigit que indica si un carácter Unicode pertenece a alguna categoría de letras o dígitos decimales.
4. **`Sort`**: método para ordenar los objetos de tipo SearchItem, que representan documentos coincidentes con la consulta, según su score final disponiéndolos de forma descendente.

```
class DataFile:
```

Cuenta con las siguientes propiedades:

- Name: nombre del documento.
- Root: ruta del contenedor de documentos.
- tfs: diccionario cuyas llaves son las palabras de un determinado documento, asociadas a su TF (buscar en el Readme del proyecto).
- NumWord: total de palabras de un documento en específico.

En el constructor de la clase primeramente se guarda la ruta, se inicializa el diccionario, seguido se extrae el nombre del documento. Luego se lee el documento empleando el método Stream Reader. Añado a un array cada palabra del documento utilizando el método String.Split, hecho esto se llama al método TokenizeWord de la clase Utils para normalizar cada palabra.

Hallo el TF según la fórmula especificada en el Readme, calculando la frecuencia de cada palabra en el documento y luego dividiendo entre la cantidad de palabras del documento (NumWord), este valor será añadido al Diccionario tfs.

```
class DataFolder:
```

- Files: array del tipo DataFile, es decir, que contiene las instancias de los documentos y las propiedades de estos.
- idfs: Diccionario con los IDF (buscar este término en el Readme del proyecto) de cada palabra con respecto a la totalidad de documentos del contenedor.
- norma: propiedad que permite acceder al vector normalizado de un documento.

Se inicializa el diccionario en el constructor, luego se obtiene la ruta de cada documento llamando al método EnumerateFiles de Utils. Hallo el IDF de cada palabra calculando la frecuencia de la misma en la totalidad de documentos y luego el logaritmo natural del total de documentos dividido entre la frecuencia de la palabra.

Posteriormente procedo con la norma de cada documento, accedo a las palabras y sus tfs en dicho documento, para calcular la relevancia que sería el producto TF-IDF (buscar en el Readme del proyecto) y finalmente la raíz cuadrada de la sumatoria de las relevancias de las palabras de ese documento, cada una al cuadrado.

Además cuenta con dos métodos:

- **TFS**: devuelve una matriz con los tfs dado un array de palabras que serán las de la consulta, cada fila corresponde a un documento y cada columna a una de estas palabras.

- **IDFS**: devuelve un array de idfs dado un array de palabras que serán en nuestra funcionalidad, las palabras de la consulta.

`class Engine` **en VectorialModel:**

Primeramente, se declara e inicializa la propiedad Folder de tipo DataFolder.

Implementa varios métodos:

- **TokenizeQuery**: procede con el mismo tratamiento que la normalización de las palabras de un documento, pero en este caso para el arreglo de palabras de la consulta.
- **QueryTf**: dadas las palabras de la consulta y el diccionario asociado al operador de prioridad de la clase OperatorsQuery, se calcula el tf de cada palabra que por conveniencia adopta la forma del inverso del total de palabras de la consulta y además aumento el numerador tantas veces como prioridad tenga la palabra.
- **Build_Documents_Vectors**: dada la matriz con los tfs de la consulta y el arreglo con los idfs, este método devuelve una matriz con la relevancia de las palabras de la consulta (TF-IDF buscar en el Readme).
- **Query_Norm**: devuelve la norma del vector consulta.
- **Vectorial_Model**: dados el vector documento y el vector de la consulta se implementa la fórmula para la similitud del coseno (Buscar en el Readme) el numerador es la sumatoria del producto escalar de los vectores y el denominador es el producto de las normas. De esta forma se obtiene el score de cada documento.
- **Snippet**: devuelve dado un documento un fragmento de este. Será el resultado que arroje la búsqueda.

- **Query:** método del tipo SearchResult que pone en marcha todo el proceso de búsqueda para arrojar los resultados que esperados. Llamada a los métodos: TokenizeQuery, WordsYes, WordsNot, WordsPriority, IDFS, TFS, Build_Documents_Vectors, QueryTf y VectorialModel respectivamente. Finalmente se multiplican los scores por el valor devuelto por NeedWords y RemoveWords, si es mayor que 0 busco su snippet. Ordeno con Sort los scores y doy los resultados según similitud con la consulta.

```
class OperatorsQuery:
```

- **WordsNot:** devuelve una lista con las palabras que según la aparición del operador ! no deben ser devueltos documentos que la contengas.
- **RemoveWords:** retorna 0 si el documento contiene la palabra para luego en la clase Engine multiplicar el score de ese documento por 0 y que no sea devuelto como similar.
- **WordsYes:** devuelve una lista con las palabras que según la aparición del operador ^ deben ser devueltos documentos que la contengas.
- **NeedWords:** retorna 0 si el documento no contiene la palabra para luego multiplicar el score de ese documento por 0 y que no sea devuelto como similar.
- **WordsPriority:** calcula la cantidad de * que le dan prioridad a una palabra en la consulta y guarda este valor en un diccionario que luego es devuelto asociando las palabras con prioridad a la cantidad de * que tenían.