Moogle!

Base de Datos(Clase DataBase)

El presente informe tiene el objetivo de presentar y explicar las características de esta versión del buscador Moogle!

El correcto funcionamiento de esta aplicación depende del contenido presente en la carpeta Content, por supuesto, este puede ser modificado a gusto del usuario (para mayor precisión a lo hora de los resultados se recomienda que todo el contenido este en un mismo idioma, en el caso contrario puede que palabras que en las anteriores condiciones eran insignificantes adquieran significación por pertenecer al idioma de la búsqueda) con los archivos que considere pertinentes que pertenezcan a la categoría de .txt, en cualquier otro caso el archivo no será procesado en la base de datos y no contará a la hora de la búsqueda.

La aplicación obtiene la dirección actual de la carpeta general y a continuación busca dentro de Content todos los .txt(s) presentes y almacena sus direcciones para su posterior análisis (estos archivos pueden estar dentro de subcarpetas, sueltos o los dos; depende totalmente del gusto del usuario).

Para ello se utilizan métodos pertenecientes a la librería IO como a la hora de obtener la ruta actual:

```
string ruta =
Directory.GetParent(Path.Combine(Directory.GetCurrentDirectory()))!.FullName!;
```

- GetCurrentDirectory() obtiene la dirección del archivo .cs desde donde se está ejecutando
- Combine se utiliza para adaptar la dirección al sistema en el que se esté ejecutando
- GetParent(). FullName! obtiene la ruta completa de la carpeta que contiene la dirección obtenida por GetCurrentDirectory

Esta librería se utiliza también en otras instancias que no mencionaremos.

Con el conjunto de direcciones de los .txt obtenidos se procede a la normalización (llamaremos normalizar a la eliminación de tildes, mayúsculas y cualquier carácter que no sea una letra o un número, comprendido entre tres métodos NormalizeExpresion, NormalizeString y Document) de dichos textos para generalizar lo más posible los resultados, es decir evitar que detalles como la falta o inclusión de tildes y el uso o no de mayúsculas reduzcan los resultados relevantes.

Para ello se utilizan métodos internos estándar como .ToLower y se añade el uso de la librería TextRegularExpressions con el método Regex.

A continuación se analizan estos ,se crea el vocabulario de nuestra base de datos (conjunto de todas las palabras pertenecientes a esta, obviamente sin repeticiones de las mismas, se desarrolla en el método Uniquewords en el cual a la vez se obtienen elementos necesarios para los siguientes pasos del procesamiento) y se comienza la vectorización de los mismos (este proceso se realiza sobre la base de la técnica de Term Frequency-Inverse Document Frequency(TF-IDF) y sus algoritmos correspondientes). Este es una medida numérica que representa la relevancia de un

documento en un conjunto de ellos, el TF aumenta con la frecuencia de una palabra en el documento mientras el IDF aumenta con la disminución de la cantidad de documentos que contienen la palabra por lo que se compensan entre sí y resulta muy efectivo a la hora de eliminar la influencia de palabras muy comunes en la base de datos (como es el caso de preposiciones, conjunciones, en general las conocidas como *stop-words).*

En el caso del TF se calcula a través de la siguiente fórmula:

$$TF = \frac{frecuenciadelterminoeneldocumento}{longituddeldocumento}$$

Mientras el IDF se basa en la siguiente:

$$IDF = log10\left(\frac{cantidaddedocumentos}{cantidadedocumentosquecontienenlapalabra}\right)$$

Se multiplican ambos valores para cada palabra y se obtiene un vector que representa el contenido del texto:

$$TF\text{–}IDF = TF \times IDF$$

De este cálculo se encargan los métodos BruteFrecuency(TF), InverseFrecuency(IDF) y TF_IDF respectivamente.

El conjunto o matriz formado por cada uno de estos vectores es nuestra base de datos, sobre los que se va a procesar los resultados de cada una de las búsquedas ingresadas.

Se considera vector en este sentido a una tabla donde cada espacio representa una palabra del vocabulario y el valor su importancia (Todos los vectores presentan una misma longitud, la ausencia de la palabra en el documento se representa por 0 , igualmente a raíz del cálculo de IDF una palabra que aparece en todos los documentos es representada también por 0)

Términos/Documentos	Doc1	Doc2
word1	0,24	0,45
word2	0	0,6

…

Esta base solo se calcula una vez por sesión, mediante la creación de una variable que llama al constructor de una clase DataBase dónde se indican y ejecutan los instrucciones de creación de la misma(explicadas anteriormente), lo que conlleva tanto a beneficios como desventajas:

- Beneficios : No es necesario volver a procesar esta por cada búsqueda lo que mejora la rapidez de respuesta de la aplicación (solo es al abrir la aplicación que esta se calcula).
- Desventajas: No se actualiza automáticamente, por lo que si el usuario realiza alguna adición, eliminación o modificación al contenido este no se contará hasta que la aplicación se reinicie.

No existe límite en la cantidad de documentos o su longitud en la base de datos pero estos aspectos influirán directamente en la velocidad de procesamiento

Búsqueda(Clase Auxiliaries)

Después de creada y cargada la base de datos, se espera a que el usuario ingrese una búsqueda y se procesa esta a continuación.

Se normaliza esta y se comienza el proceso de vectorización de la misma (como si fuera uno de los documentos)(se utiliza los mismos pasos anteriores adaptados) , a la misma vez empieza a calcularse la sugerencia correspondiente pues en caso de existir el vector original de la búsqueda podría verse alterado.

Haciendo un inciso en la sugerencia, esta es calculada mediante un método que se apoya en el algoritmo de la “Distancia de Levenshtein” (también llamado distancia de edición, se creo un método auxiliar de mismo nombre utilizado en el método principal Suggestion), este se puede definir como el número mínimo de operaciones necesarias para transformar una palabra en otra indicada, por lo cual es muy útil para definir cuándo una palabra es similar a otra si se establece un umbral de similitud (máxima distancia con la que consideraremos que son semejantes las palabras) en el caso del Moogle! Actual se estableció el umbral en “3”(este no posee un valor predeterminado en general por lo cual depende enteramente de la situación y se consideró que 3 otorgaba suficiente libertad de opciones a las sugerencias manteniendo un grado adecuado de similitud).

El calculo de la sugerencia de una búsqueda parte del reconocimiento de los términos ajenos al vocabulario existente, si todos los términos pertenecen a este el resultado será *“No hay sugerencias disponibles para esta búsqueda”*; de lo contrario se pasará a analizar posibles alternativas a dichas palabras.

Este análisis es individual por cada palabra así que es posible tener resultados válidos para una de las palabras y para la otra no. Este resultado se adquiere al calcular la distancia de edición de la palabra en cuestión con cada uno de los términos del vocabulario, se almacena la menor de ellas y se compara con el umbral; si cumple con la condición se tiene una sugerencia válida que se enseñará al usuario de lo contrario no se contará para esta.

Es posible que exista más de un término que presente la distancia mínima con la palabra analizada, en ese caso se tomará en cuenta la que se encuentre primero

La sugerencia se devuelve en el mismo formato que la búsqueda es decir si esta poseía palabras del vocabulario y ajenas se devolverá con esa misma estructura y orden, simplemente se sustituirán las ajenas con las similares encontradas (si son más de una y una de ellas no tenía resultados válidos esta se mantendrá como la original).

Si existen palabras ajenas y ninguno posee resultados válidos en el vocabulario existente el resultado es *“No hay sugerencias disponibles para esta búsqueda”*

Se avisa al usuario de que la sugerencia devuelta no siempre implica mayor criterio a la hora de los resultados pues se puede dar el caso de que el resultado sea una palabra común que no otorgue relevancia alguna.

Los resultados válidos de la búsqueda afectarán al vector de la misma pues se añadirá frecuencia a las nuevas palabras(Todo el proceso de confirmación del vector búsqueda queda a cargo del método WordsofSearch)

Volviendo al procesamiento de la búsqueda una vez se tiene el vector de la misma(con los cambios pertinentes de la sugerencia) se procede a la aplicación del operador (*)(del cual se hablará más adelante junto a los otros de su clase) y luego a la comparación de este con el de cada uno de los documentos de la base de datos mediante un algoritmo llamado “similitud de cosenos”(presente igualmente como algoritmo auxiliar CosineSimilarity), la cual es una medida para calcular la similitud entre dos vectores (está se mueve entre 0 y 1, a mayor resultado mayor similitud) como nuestros vectores están compuestos por palabras está indica que tan parecidos son los dos textos.

La fórmula para este algoritmo es la siguiente:

$$similaridad = \frac{productopunto}{magnituddelabúsqueda \times magnituddeldocumento}$$

- Productopunto es la multiplicación del vector de la búsqueda por el vector del documento que se está analizando
- Magnitud del vector(también llamado norma, representa la longitud del vector en el espacio) se calcula a través de la siguiente fórmula:

$$magnitud = \sqrt{(x_1^2 + x_2^2 + \cdots + x_n^2)}$$

- X es cada uno de los valores del vector

A este valor de similitud es lo que denominamos como Score del documento y es el criterio por el cual se ordenarán y añadirán los documentos a los resultados finales(otros criterios para añadir resultados dependen de los operadores (¡) y (^) y se abordaran más adelante).

Los resultados finales van dados en un conjunto de objetos del tipo SearchItem que almacenan Score, Title(título) y Snippet.

Una vez se comprueba que un documento posee un Score superior a 0 se procede a buscar su título correspondiente (en nuestro concepto de matriz a cada documento le corresponde una columna de esta con un orden correspondiente al cual se leen sus direcciones al principio de la creación de la base de datos, se utiliza este hecho para asociar columna con dirección y extraer el título correspondiente al .txt en el método Index) y se calcula el Snippet, en este caso el fragmento del texto más relevante respecto a la búsqueda analizada (la longitud de este es generalmente de alrededor de 50 palabras a menos que el documento analizado posee una longitud menor a esto en cuyo caso será de aproximadamente la mitad de dicha longitud).

La relevancia del fragmento se define a través de la cantidad de coincidencias con las palabras relevantes de la búsqueda que se pueden encontrar en él (estas coincidencias pueden ser de una misma palabra o de una variedad de ellas). El cálculo del Snippet parte de una cuenta inicial de coincidencias en el primer fragmento de la longitud establecida y a partir de ahí se van analizando los índices siguientes, es decir se analiza el que está a continuación del índice final y del índice inicial para cubrir todos los posibles fragmentos del documento.

Siempre va a existir algún Snippet para el documento pues para que este calificara para los resultados debía contener por lo menos una vez una de las palabras de la búsqueda, en ese caso justamente ese será el fragmento devuelto.

Todos los procesos explicados del Snippet se realizan en un método de igual nombre.

En el caso de no encontrarse ningún documento que cumpla las condiciones para ser añadido a los resultados se añadirá y devolverá un solo SearchItem:

Ejemplo

Búsqueda: Xcydtini

Resultados:

- ***No se encontraron textos coincidentes***

Por favor realice una búsqueda diferente …

Una vez llegado a este punto la aplicación está casi lista para finalizar su procesamiento, a partir de aquí solo resta juntar todos los resultados y devolverlos al usuario ordenados por orden de relevancia (esto se logra a través del valor del Score y un método de ordenamiento basado en el algoritmo de inserción(Order)) . El método RelevantDocuments se encarga de todos estos pasos .A continuación se muestran dos ejemplos del funcionamiento de la misma, uno inválido y otro válido

1. *Búsqueda: catt and harts*
   Sugerencia : ¿ Quisiste decir cast and parts?
   Resultados:

- ***The Strange Case Of Dr. Jekyll And Mr. Hyde***
  … mountain bandit (…) pursuit jekyll (…) to cast (…) to pamper to cast …
- ***The …***

2. *Búsqueda: granadinas*
   Sugerencia: ¿ Quisiste decir No hay sugerencias disponibles para esta búsqueda?
   Resultados:

- ***No se encontraron textos coincidentes***
  Por favor realice una búsqueda diferente

Este es el funcionamiento básico de la aplicación y su comportamiento, mas también están disponibles algunos operadores que podrían ayudar al usuario a obtener resultados más específicos a sus deseos.

Operadores(Clase Auxiliaries)

- *Potenciación(*) (otorga mayor importancia a la palabra a la que antecede)*

Este operador es múltiple es muchos sentidos, puede ser utilizado por supuesto en más de una palabra de la búsqueda y pueden ser más de uno por palabra

Ejemplo:

****Alice and the *cat*

Este operador modifica los valores del vector de la búsqueda por lo que se implementa durante la confección del mismo, utiliza un método que a través del método Regex identifica las palabras que poseen asteriscos, se analiza cuántos tiene cada una de ellas y se multiplica el valor de su posición correspondiente en el vector por 2 por cada asteriscos

Apoyándonos en el ejemplo anterior el valor se transformaría:

Valor(Alice)×6, … , Valor(And), … ,Valor(the), … ,Valor(cat)×2

A la hora de aplicar este operador también se tiene en cuenta los cambios ocasionados por la sugerencia, es decir si una de las palabras no existiera en el vocabulario pero existe una sugerencia válida se calculará el operador según la sugerencia

***Alic and the cat*

Valor(Alice)×4 …

Este operador está contenido en el método WordswithAsterisk.

- *Exclusión(!)(indica que el término no debe aparecer en los documentos del resultado)*

Este operador sigue el mismo patrón del anterior y utiliza el método Regex para identificar la palabra(s) que no deben aparecer, a continuación se procede a analizar su presencia en el documento por cada palabra que presente el símbolo en la búsqueda. Este operador afecta a los resultados por lo cual solo si implementa una vez calculado el Score y comprobado su validez, devuelve un bool y si uno de los términos este presente en el documento su valor es automáticamente falso y se dejan de analizar el resto de términos en el caso de existir más

Ejemplo:

Documento: Alice in Wonderland
Búsqueda: Alice visit !France and the !queen
Operador !: france(true) queen(false)
Resultado final: false

Al igual que en el operador Potenciación este tiene en cuenta los cambios que conlleva la sugerencia y funciona según esta si es necesario. La denominación del método es equivalente a su función (Exclusion).

- *Inclusión (^)(indica que el término necesariamente debe aparecer en los documentos del resultado)*

Su funcionamiento es análogo al del operador Exclusión, simplemente en vez de verificar que los términos no aparezcan verifica que estos estén presentes en el documento analizado, al igual que el caso anterior si el documento incumple al menos

una de las condiciones (en este caso la presencia de un término(s) específico) su valor es automático falso y no es válido como resultado.

Igualmente tiene en cuenta la presencia de la sugerencia y es funcional según esta. Su método se denomina Inclusion.

Ejemplo:

Documento: A Study in Scarlet

Búsqueda: ^Sherlock in ^Wonderland and Dr. ^Watson

Operador ^: sherlock(true) wonderland(false)

Resultado final: false

- *Cercanía (~)(otorga mayor relevancia al documento en dependencia de cuan cercanas están un o más pares de palabras indicadas)*

Este operador se encuentra dividido en dos métodos(CloseSearch y Close), el primero de ellos revisa la presencia del símbolo en la búsqueda (a través de un método Regex y un patrón especificado) y en caso de existir extrae los pares de palabras

En casos como: *Alice ~ cat ~ queen.* Se extraen dos pares (Alice y cat) y (cat y queen)

Es fundamental para su correcto funcionamiento que se deje espacio entre las palabras y el símbolo

Ejemplo: *Alice~cat* (no funcionará)

Casos como:

1- *~ Alice cat*
2- *Alice cat ~*

Se tratarán como si no existiera los símbolos

3- *Alice ~ ~ ~ cat*

Se tratará como si fuera un solo símbolo

Al igual que los anteriores operadores este tiene en cuenta los cambios que conlleva la existencia de una sugerencia válida.

Una vez establecidos los pares se procede en el segundo método a calcular la menor distancia entre todos los pares existentes, estas se suman y se calculan como denominador de una fracción con numerador constante, este resultado se suma al Score del documento analizado.

- *General (Condiciones de combinaciones de los operadores)*

1- *En una búsqueda de pueden utilizar cuántos operadores se deseen(por palabra)*
Ejemplo:

*Búsqueda: ***Alice ~ cat and the ^queen of !England*

2- *Si se utilizan dos o más operadores en una misma palabra solo se aplica el más interno a la palabra*
Ejemplo:
Búsqueda: !^Alice (devolverá los documentos donde aparece la palabra alice)
^!Alice (devolverá los documentos donde no aparece la palabra)
Lo mismo ocurre con el operador (*), no aplica al operador cercanía por su necesidad de estar separado.

- *Clases Algebraicas (Matrix y Vector)*

En estas clases se almacenan un conjunto de métodos de operaciones realizables con matrices y vectores que pueden ser de utilidad en el tipo de cálculos que se ejecutan en esta aplicación. A continuación se nombrarán y explicarán brevemente cada uno de ellos:

Clase Matrix

1- Sum(suma de matrices)

Este método comprueba en primer que las matrices pasadas como parámetros sean válidas, en este concepto se consideran válidas para la suma si las matrices son del mismo tamaño, para ello se utiliza un método privado que comprueba justamente esto. Después de esta comprobación se procede a ejecutar la suma:

$$A + B = a_{ij} + b_{ij} = c_{ij} = C$$

A,B pertenecen al conjunto de las matrices; i,j índices de fila y columna; C matriz resultado

La matriz resultado es la matriz donde en cada posición tiene la suma de las posiciones correspondientes de las matrices sumadas

2- ScalarxMatrix(multiplicación de una matriz por un escalar)

El método recibe una matriz y un escalar y procede a multiplicar cada posición de la matriz por el escalar

$$dA = da_{ij}$$

d es un escalar, A pertenece al conjunto de las matrices e i,j índices de fila y columna

3- MatrixxMatrix(multiplicación de dos matrices)

Este método en primer lugar comprueba que se pueda realizar la operación, para ello hace uso de un método auxiliar que comprueba que las matriz A tenga la misma cantidad de columnas que filas la matriz B, también analiza el caso de que los parámetros fueran ingresados en el orden equivocado y las organiza (ya que esta multiplicación no es conmutativa)

$$\Sigma a_{ik} \times b_{kj}$$

Dónde i,j son índices de fila, columna y k es el índice común de las columnas de la primera matriz y las filas de la segunda

El resultado es una matriz de tamaño (cantidad de filas de la primera matriz, cantidad de columnas de la segunda) donde cada fila de la inicial se multiplica por las columna de la segunda para obtener los nuevos valores.

4- MatrixVector(multiplicación de una matriz por un vector)

Al igual que en los anteriores métodos primero se comprueba el cumplimiento de los requisitos para poder realizar la operación (a través de un método privado), en este caso que el vector tenga una longitud igual a la cantidad de columnas de la matriz,(se trata de un vector columna).

$$\Sigma x_i \times c_i$$

Donde ci son cada una de las filas de la matriz y xi los valores del vector

El resultado es un nuevo vector de longitud de la cantidad de filas de la matriz donde los valores son el resultado de multiplicar cada fila por el vector

Los métodos están definidos para recibir valores double para permitir mayor libertad de precisión al usuario, se define matriz como un array bidimensional (a[,])

Clase Vector

1- Sum (suma de dos vectores)

Los procedimientos son semejantes a los implementados en Sum de matrices

$$C = a_i + b_i$$

2- ScalarxVector (multiplicación de un escalar por un vector)

Igualmente se procede igual al método ScalarxMatrix pero tratando con un vector

$$dA = da_i$$

3- VectorxVector (multiplicación de vectores)

Este método comprueba en primer lugar si los vectores son multiplicables entre sí, es decir si poseen una misma longitud, y procede a multiplicarlos

$$e = \Sigma a_i \times b_i$$

El resultado es el valor resultante de sumar el producto de cada una de las posiciones correspondientes a ambos vectores

4- Magnitude(Calcula la magnitud o norma de un vector)

Método que calcula el valor que representa la extensión de un valor en el espacio. La fórmula que utiliza este ya fue explicada anteriormente cuando se habló del IDF.

En esta clase también se utilizan valores double y se representan los vectores como arrays(a[])