# *Proyecto de programación*

**Nombre:**Olivia Ibañez Mustelier **Grupo:**C111

# Moogle!

Motor de búsqueda que nos permite dado una query insertada por el usuario realizar una búsqueda entre una serie de archivos txt ubicados en la carpeta **contens**. Esta nos devuelve desde el documento con mayor frecuencia hasta el que menor frecuencia tiene respectivamente, también cuenta con una serie de operadores que se encargan de filtrar la búsqueda. El programa se encuentra implementado en un total de seis clases más dos clases llamados **SearchItem** y **SearchResult** que ya habían sido implementadas.

## Clases:

1. **Query**
2. **Textos**
3. **Sugerencias**
4. **Operadores**
5. **TF_IDF**
6. **Moogle**

# 1 Query

## 1.1 Mi_Query

Para empezar la busqueda primero se debe saber que se esta buscando, que desea hallar el usuario, a traves de lo que escriba en a query, para despues saber cuales textos se deben dovolver.
Clase que se encargará de hacer todas las operaciones necesarias con la query incertada por el usuario.

## Métodos:

1. **Repetición_de_palabras:** Una vez incertada la query puede darse el caso de que al usuario le haya sido necesario repetir más de una vez algunas o todas las palabras. Por eso se implementa este método que conge una sola vez cada una de ellas y guarda cuantas veces se repiten, de esta manera cuando se requiera hacer algún trabajo con la query solo se procesran

una vez, haciendo ḿas eficiente el trabajo, además nos dará una idea de cuales son las más importantes según su número de repeticiones.

2. **Dame_una_palabra:** Con el objetivo de mejorar la búsqueda se encuentran los operadores, que no son más que caracteres especiales que se ponen delante o entre palabras de la query para que el usuario pueda especificar mejor lo que desea(se explicarán en detalle más adelante), por esta razón es necesario obtener las palabras que están detrás de los operadores en caso de que hayan y para esto es que se utiliza este método.
3. **Dame_una_palabra2:** Con un objetivo parecido al método anterior, este se encarga en caso de ser necesario de obtener la palabra que se encuentra delante del operador.
4. **Operadores:** Este método tiene la tarea de buscar en la query la presencia de algún operador y en caso de encontralo, apoyándose de los dos métodos anteriores, extrae la o las palabras a la que modifiqua y la cantidad de operadores que lo hacen en caso de ser necesario,luego las guarda en listas separadas según el tipo de operador.

# 2 Textos

Con el objetivo de trabajar de manera general con los textos, se crea la clase **Normalización** , y luego para encargarns de los de contens en específico la clase **Procesamiento_txt**.

## 2.1 Normalizacion

### Métodos:

1. **Normalizar:** Para trabajar más cómodamente con cualquier texto se utiliza este método que se encarga de quitar las tildel y ponerlo todo a minúsculas.
2. **Separar:** Como el nombre lo indica este método separa las palabras del texto que se le pase quitando los signos de puntuación o algun símbolo que no represente nada para la búsqueda.
3. **Hazlo_todo:** Se encarga de llamar a los dos métodos anteriores para que realicen sus acciones y normalicen el texto.

## 2.2 Procesamiento_txt

### Métodos:

1. **Leer:** Lee y guarda los textos que se encuentran en la carpeta **contens** del proyecto.
2. **CreatyDictionary:** Con un objetivo similar al método **Rpeticion_de_palabras** de la query, se crea este para guardar los textos en una lista de diccionarios, donde cada diccionario es un texto con las palabras que presenta y al lado la cantidad de veces que se repiten, para así a la hora de trabajar con ellos no hacer más de una vez el mismo proceso.
3. **Procesar:** Llama al método **Hazlo_todo** de la clase anterior y normaliza cada uno de los textos.
4. **Recorte_de_textos:** Se le conoce como **Snipet** y se utiliza para una vez encontrado los textos como resultado de la búsqueda, especificar que parte de ellos es la que se va a mostrar

en pantalla para el usuario, donde se encuentren alguna o algunas de las palabras de mayor relevancia de la query.

# 3 Sugerencias

## Métodos:

1. **Levenshtein:** Método que devuelve la menor distancia entre dos palabras, la de laquery y las del texto, apoyándose del metodo textbfMenor.
2. **Menor:** Consiste en determinar el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, bien una inserción, eliminación o la sustitución de un carácter, cada una le suma uno a la distancia.
3. **Palabra_semejante:** Devueleve una lista de palabras que se obtienen de llamar al método Levenshtein el cual da el resultado de cuan cerca esta una palabra de otra las de los textos a la de la query que se esta analizando, guardando el resultado y la palabra del texto que le corresponda. En caso se ser menor a la que se tenía hasta el momento, se actualiza al resultado y la palabra del texto. Quedándonos al final con las palabras que más se parescan a cada una de las de la query, devolviendolas como una lista.
4. **Sugerir:** Con las lista de palabras semejantes a las palabras de la query y la relevancia de estas últimas en los textos(**Tf_IDF**) se decide que se le puede sugerir al usuario. En caso de tener poca relevancia(artículos, preposiciones, conjunciones, etc...) se devuelve la misma palabra.

# 4 Operadores

En esta clase se trabaja con las listas que se habian creado anteriormente de las palabras que son modificadas por algun operador.

## Significado de cada operador:

- Un símbolo '!' delante de una palabra, indica que esa palabra **no debe aparecer** en ningún documento que sea devuelto.
- Un símbolo '^' delante de una palabra, indica que esa palabra **tiene que aparecer** en cualquier documento que sea devuelto.
- Un símbolo '~' entre dos o más términos indica que esos términos deben **aparecer cerca**, o sea, que mientras más cercanos estén en el documento mayor será la relevancia. Por ejemplo, para la búsqueda **"algoritmos ~ ordenación"**, mientras más cerca están las palabras **"algoritmo"** y **"ordenación"**, más alto debe ser el **score** de ese documento.
- Cualquier cantidad de símbolos '*' delante de un término indican que ese término es más importante, por lo que su influencia en el **score** debe ser mayor que la tendría normalmente (este efecto será acumulativo por cada '*').

### Métodos:

1. **Mascara:** Utilizando las lista de las palabras que son modificadas por los operadores '!' y '^' y me dice si están o no en los textos. Así a la hora de devolver el resultado de la búsqueda se sabrá cuales son los textos que se pueden mostrar.
2. **Multiplicar:** Multiplica la cantaidad de operadores '*' que presenta la palabra delante, en caso de haber alguna, por 25 para aumentar su importancia en la query.
3. **Distancia:** Calcula la distancia que hay de una palabra a otra (la cantidad de palabras o espacios que las separan).Se empieza por cualquiera de las dos y se ve que distancia tiene con la que me queda, en caso de encontrarse otra vez con la misma, el conteo comienza desde cero, de lo contrario si se encuentra con la que se busca, guardo la distancia e inicio el contador hasta encontrar otra vez la palabra inicial y así sucesivamente quedándome siempre con el menor de los valores que me halla dado hasta el momento.
4. **Menor_distancia:** Llamando al metodo anterior calcula la distancia entre las palaras de la query modificadas por el operador

# 5 TF_IDF

### Métodos:

1. **Resultado:** Este método se utiliza para calcular la relevancia de cada texto según la presencia en estos de las palabras de la query. Este procedimiento se calcula mediante el TF, que no es más que la frecuencia con la que se repite cada palabra de la query en cad texto, la cual ya esta calculada desde que se convirtieron los textos en diccionarios donde el valor de las palabras es su frecuencia. Para evitar que adquieran importancia palabras que en realidad no lo son, como preposiciones, conjunciones, artículos, etc..... se calcula el IDF que no es mas que ver en cuantos textos esta la palabra, ya que mientras más aparesca menos relevante será. Luego se multiplica TF*IDF y este es el valor del texto. Además de este cálculo también se aumenta el valor del texto si se encuentra una palabra modificada por el operador de importancia, o el de cercanía en caso de encontrarse, causando este último que mientras más cerca estén las palabras más relevante será el texto.
2. **Ordenar:** Se ordenan los textos segun el valor que tenga cada uno hallado por el metodo anterior.

# 6 Moogle

Esta clase le da orden y coerencia a cada una de las aciones y método a utilzar para la búsqueda, especificando el orden en que se llama cada uno.
Primero se hace todo el procesamiento de la query, normalizándola y viendo cuántas veces se repite cada palabra, luego se leen los textos e igual se normalizan y se guardan en una lista de diccionarios. Después se guardan las palabras semejantes a las de la query, que son las que se devolverán en las sugerencias.
Una vez hecho todo esto se pasa a crear las listas donde se guardarán las palabras que son modificadas por los operadores, por separado según el tipo, para luego guardarse en caso de que haya alguno en la query.

Por último pero no menos importante se calcula TF_IDF para saber el score de los textos y devolver los que mayor resultado obtengan.