Introducción

¡En este proyecto desarrollaremos el Sistema de Recuperación de Información (SRI) Moogle!, aplicación totalmente original cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos. Un sistema de recuperación de información (SRI) es el responsable de encontrar información relevante la cual está presente en un registro documental que debe ser almacenado, representado, analizado (manipulado) y mantenido. Su estructura se compone de elementos básicos, como debe ser: los documentos en sí, la representación de los mismos, el tratamiento de las consultas sobre estos documentos y el tratamiento de los resultados.

La Recuperación de Información

El objetivo de la recuperación de información es, dada una necesidad de información y un conjunto de documentos, ordenar los documentos de más a menos relevantes para esa necesidad y presentarlos al usuario. El problema principal en este sistema es obtener representaciones homogéneas de documentos y consultas, y procesar convenientemente esas representaciones para obtener la salida.

En el proceso de recuperación de información se suelen distinguir las siguientes

etapas:

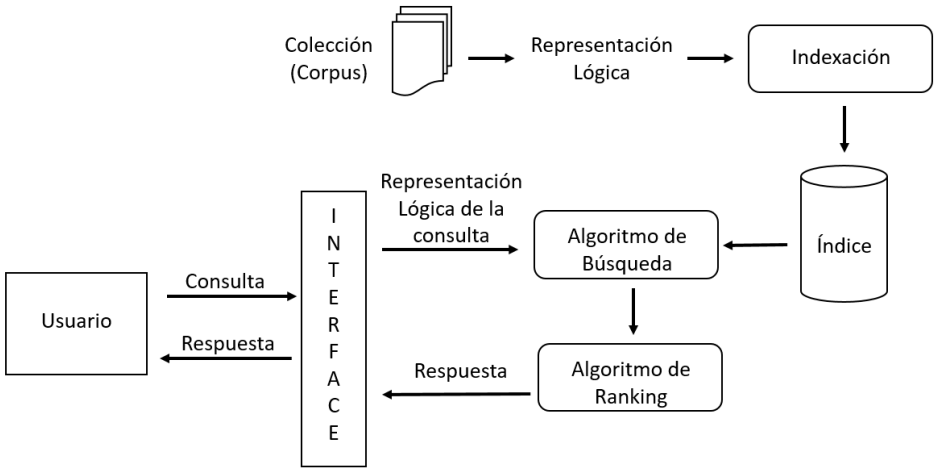

fig 1: Proceso de recuperación de información

Descripción del proyecto

- La búsqueda en nuestro proyecto nos permite no limitarnos a la búsqueda de una palabra sino una frase en general cualquiera, dándonos exactamente donde aparece está en los documentos.

- Si de todas las frases de la consulta no aparecen en un documento, pero al menos algunas aparecen este documento es devuelto, pero con un menos peso mientras menos palabras aparezcan.

- El orden en que aparezcan en el documento los términos de la consulta en general no importa, ni siquiera que aparezcan en lugares totalmente diferentes del documento.

Operadores

El proyecto operadores para mejorar la búsqueda del usuario:

 Exclusión, identificado con un ! delante de una palabra, indica que la palabra no debe aparecer en ningún documento devuelto.

 Inclusión, identificado con un ^ delante de una palabra, indica que la palabra debe aparecer en todos los documentos devueltos.

 Mayor Relevancia, identificado por varios * delante de una palabra, indica que la palabra es más relevante que las demás palabras de la consulta tantas veces como * tenga delante de ella.

 Cercanía identificado con un ~ entre las palabras indica que los documentos en los que aparecen estas dos palabras más cerca tendrán mayor peso.

Sugerencia

Para brindar una mayor exactitud en la búsqueda, el proyecto cuenta con un corrector de palabras, el cual se encarga de dar una sugerencia al usuario en caso de que la búsqueda no coincida con los datos almacenados.

Ideas extras

Con el fin de que nuestra aplicación sea mucho mejor, implementamos en la búsqueda que si las palabras exactas no aparecen, pero aparecen palabras derivadas de la misma raíz, también devuelve esos documentos, pero con un peso menor. También, si aparecen palabras relacionadas, aunque no tengan la misma raíz, devolvemos esos documentos, pero con menor peso que si apareciera la palabra exacta o una de la misma raíz.

Resultados de la Búsqueda

Ya calculados los pesos de los documentos, se le muestra al usuario mismos ordenados de mayor a menor peso, con el título, el peso y el fragmento donde se encontraron las palabras buscadas por este.

Algoritmos de Búsqueda. El Modelo Vectorial

El modelo vectorial fue planteado por Gerard Salton en 1968 y originalmente se implementó en un SRI llamado SMART. Aunque el modelo vectorial tiene más de 30 años, actualmente es usado debido a su buen rendimiento en la recuperación de información (RI). Este modelo es una mejora del sistema del modelo booleano, ya q reconoce que los pesos binarios son muy limitados y no proporcionan ningún valor intermedio.

El modelo de recuperación vectorial propone un emparejamiento parcial a diferencia del modelo booleano, asignando pesos no binarios a los términos de las preguntas del usuario y de los documentos. Estos pesos de los documentos se utilizan para calcular el grado de similitud entre cada documento de la colección y la consulta realizada por el usuario. Los documentos recuperados son ordenados de manera descendente en base al grado de similitud. La idea básica de este modelo consiste en la construcción de una matriz que contiene el vocabulario de la colección de referencia y los documentos existentes, donde las filas corresponden a los documentos y las columnas a las palabras contenidas en estos. En la inserción de un término y un documento se almacena un valor de importancia q significa la frecuencia de aparición de ese término en el documento.

Método Tf-Idf:

$$f_{i,j} = \frac{frec_{i,j}}{\max frec_j}$$

$$idf_i = \log \frac{N}{n_i}$$

Para obtener el peso de los términos de una consulta Salton y Buckley e el año 1987 sugirieron la fórmula:

$$w_{i,q} = \left[0.5 + \frac{0.5 frec_{i,q}}{\max frec_q}\right] \times \log \frac{N}{n_i}$$

A manera de resumen utilizando el TF-IDF (frecuencia de término - frecuencia inversa de documento), el cual determina la relevancia de una palabra asociada a un documento en una determinada colección, sumado a la Similitud del Coseno, método mediante el cual se asigna un peso a cada documento y se establece un ranking de resultados para el usuario.

Para ejecutar el programa:

Este proyecto está desarrollado para la versión objetivo de .NET Core 6.0. Para ejecutarlo debe ir a la ruta en la que está ubicado el proyecto y ejecutar en la terminal de Linux:

make dev

Si está en Windows, debe poder hacer lo mismo desde la terminal del WSL (Windows Subsystem for Linux), en caso contrario puede ejecutar:

dotnet watch run --project MoogleServer

Implementación MoogleEngine

Estructura de la biblioteca de clases MoogleEngine y Matrices donde se guarda las operaciones con matrices necesarias para el proyecto.

Clases Ayudantes:

Estas clases brindan funcionalidades al proyecto, encapsulando procesos y evitando repetir código en cada una de las clases principales

-Clase Útiles:

Esta clase brinda a otras clases propiedades y métodos que son utilizados en el proyecto. En ella se define el camino de los archivos de los documentos, métodos básicos del procesamiento de los textos y de cálculo, tipos de datos, entre otros

- Clase Lematizador:

Esta clase es la encargada de sacar las raíces de las palabras para buscar los lemas. Es una variación del algoritmo de Porter para el español, publicada en jesusutrera.com/articles/article01.html. La misma se ha modificado para

adaptarla a nuestro programa. El algoritmo de Porter [Porter, 1980], se basa en una serie de reglas condición / acción. Tiene la finalidad de obtener la raíz de una palabra (lexema) y consta de una serie de pasos, en cada uno de los cuales se aplican sucesivamente una serie de reglas (reglas de paso) que van suprimiendo sufijos o prefijos de la palabra hasta que nos quedamos sólo con el lexema final. Nuestra clase consta con el método Lematizar, el que recibe como entrada la palabra de la que se quiere obtener la raíz y devuelve como salida la raíz de la palabra. Es utilizada por las clases Documento y Consulta.

- Clase Listado Palabras Vacías:

Esta clase busca las palabras vacías en los documentos creando los valores de la propiedad Palabras Vacías de la clase Útiles. Brinda el método CreaListado() que devuelve como salida el listado de las palabras vacías que aparecen en los

documentos de la colección. Debido a que en la consulta pueden aparecer palabras vacías como lo artículos, las preposiciones, las conjunciones, etc., que no necesariamente aparecen

todas en los documentos, comenzamos el proceso adicionando a nuestra lista los artículos, las preposiciones, las conjunciones, los pronombres y los adverbios del español.

Clases principales

- Clase Documento:

Esta clase es la encargada de gestionar todo lo relativo a los documentos sobre los que se realizaran las consultas. Al llamar a su constructor se le pasa el camino de uno de los documentos contenidos en la colección de la carpeta Content, el cual es cargado a la memoria y procesado. Al procesar el documento se eliminan las palabras vacías, se extraen y normalizan los términos y se crean los lemas de estos. Almacena el texto del documento, los términos que aparecen en el mismo y los lemas de los mismos. Brinda métodos para obtener una lista de las posiciones donde aparece un término en el documento, la cantidad de veces que aparece, si existe un término en el documento y la distancia mínima en el documento entre dos términos. También tiene métodos para obtener la cantidad de veces que aparece un lema y si existe un lema en el documento. El método ExtraerOracion extrae del texto del documento una oración con la mayor cantidad de palabras posibles de una lista que le pasamos (términos que aparecen en la consulta) y que se utiliza para buscar el fragmento del texto que se devuelve junto con los resultados de la búsqueda.

-Clase Consulta o Query:

Esta clase es la encargada de gestionar todo lo relativo a las consultas. Al llamar a su constructor se le pasa la consulta, de la que se eliminan las palabras vacías, se normalizan los términos y se procesan los operadores. Almacena el texto de la consulta y los términos que aparecen en la misma. Brinda métodos para obtener los términos que aparecen en la consulta y la cantidad de veces que aparece cada uno, los términos obligatorios (operador ^), los términos prohibidos (operador !), los términos importantes (operador *) y los términos cercanos (operador ~).

- Clase Motor:

Realiza el proceso de la búsqueda de la consulta en los documentos de la colección y calcula los pesos de cada uno.

Al llamar a su constructor se crea la lista de documentos, los cuales se procesan y quedan listos para las consultas. También se crea la lista de palabras vacías. Este método es invocado una vez cuando se inicia la aplicación y deja ordenada toda la información para cuando se realicen las consultas. El método Realizar Consulta es el utilizado para realizar una consulta al grupo de documentos de la colección. Se le pasa como parámetro la consulta que se va a realizar y retorna un arreglo de SearchItem con los resultados de la consulta realizada. En la propiedad Sugerencia de esta clase se devuelve una sugerencia al usuario en caso de que la búsqueda no coincida con los datos almacenados. Al realizar el usuario una consulta, el proceso realizado es el siguiente:

1. Del total de documentos de la colección, buscamos los documentos que tengan al menos un término de la consulta. Si no tienen ningún término, el peso del documento será 0, por lo que no nos interesan ya que no aportan ninguna información.

2. Los documentos obtenidos en el paso anterior son filtrados para eliminar del grupo aquellos que tengan términos prohibidos o no tengan términos obligatorios (según los operadores ! o ^ que aparezcan en la consulta).

3. Se crea el vector TF de la consulta.

4. Se crea la matriz con los vectores TF de los documentos.

5. Realizamos la revisión de la matriz TF de los documentos antes de pasarla al cálculo de los pesos.

6. Se realiza en cálculo de los pesos de los documentos utilizando el coseno del ángulo formado por el vector de la consulta y el vector de cada uno de los documentos.

7. Se ordenan los documentos de mayor a menor de acuerdo a los pesos de cada uno.

8. Se procede a crear el arreglo de SearchItem, incluyendo fragmento del texto que se devuelve junto con los resultados de la búsqueda.

9. Se devuelven los resultados.

En el paso 5 se realiza la revisión de la matriz TF de los documentos antes de pasarla al cálculo de los pesos, debido a que si hay alguna columna en cero significa que el término no se encontró en ningún documento y esto haría una

división por 0 que invalidaría el cálculo. La revisión se realiza en cuatro pasos que son los siguientes:

1: buscamos las columnas que están en cero y si hay alguna buscamos si la palabra está mal escrita y nos da el sistema una sugerencia. Para buscar la sugerencia buscamos en todas las palabras de los documentos aquella con un

porciento de similitud mayor con respecto a nuestra palabra. Si hay una sugerencia, se devuelve una sugerencia al usuario y calculamos la frecuencia del nuevo término en los documentos, actualizando la columna de la matriz TF.

2: buscamos las columnas que están en cero y si hay alguna buscamos el lema de la palabra. Calculamos la frecuencia del lema en los documentos, actualizando la columna de la matriz TF. La cantidad de veces que aparece el lema la multiplicamos por 2, para que sea menor el peso del lema que el del término original.

3: buscamos las columnas que están en cero y si hay alguna la

eliminamos del vector TF de la consulta y de la matriz de vectores TF de los

documentos.

En el paso 1, para buscar la sugerencia buscamos en todas las palabras de los documentos aquella con un porciento de similitud mayor con respecto a nuestra palabra. Este porciento de similitud lo calculamos utilizando la distancia de edición entre dos palabras según el algoritmo de Damerau-Levenshtein. Se le llama distancia de edición al número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Se entiende por operación, bien una inserción, eliminación, sustitución o transposición de dos caracteres.

El código de este método lo tomamos de https://www.csharpstar.com/csharp□string-distance-algorithm