Estudiante: Laura Muñoz Santos **Grupo:** C121

Informe del 1er proyecto de programación (Moogle)

Introducción

Con este informe escrito se pretende plantear las cuestiones más relevantes de la implementación de este Primer Proyecto de Programación. Moogle, así denominado, es un buscador, que tiene como objetivo explorar dentro de un conjunto de documentos, y así poder ofrecerle al usuario la palabra o documento que está buscando... si se encuentra. A continuación se explicará la arquitectura del proyecto, así como sus métodos más interesantes.

Clases

Este proyecto consta de varias clases como: LeerDocumentos, TFIDF, SimilitudCoseno, Snippet, Levenshtein, Operadores y Moogle.

Normalización de documentos

LeerDocumentos es la clase encargada de leer los documentos que se encuentran en la carpeta content. Esta clase posee dos métodos: LeerDocs y LeerPalabras. LeerDocs es el método encargado de guardar en un array la dirección donde se encuentras los archivos así como almacenar en otro array los nombres de los documentos como tal (sin la dirección delante). Con el método LeerPalabras, utilizando el array que contiene los documentos y el método StreamReader, podemos guardar en una lista todas las palabras que contiene cada documento, para poder posteriormente trabajar con ella. Para este método se investigó como normalizar las palabras (convertirlas a mínusculas, quitar las comas, puntos, espacios, etc), y se utilizarón varios métodos valiosísimos como Char.IsLetterOrDigit y .ToLower. Finalmente se alamacenó todas las palabras en una lista. También se trabajó con diccionarios, en este caso para guardar todas las palabras por documentos, y que tuvieran de Value la cantidad de repeticiones por cada documentos.

TF-IDF

Una de las clases más importantes e interesantes, es la clase llamada TFIDF (Frecuencia de término - Frecuencia inversa de documento). Con el TFIDF podemos saber cuan relevante es una palabra en un conjunto de documentos, y se utiliza mucho en sistemas de recuperación de información, entre ellos los propios motores de búsqueda de internet. Por una parte está el TF (a mayor frecuencia de un término en un documento, mayor importancia tendrá), por otra parte el IDF (a mayor frecuencia en los documentos, menor importancia del término). Por ejemplo el IDF no tiene en cuenta palabras muy usadas en muchos documentos como los artículos, conjunciones, preposiciones, por lo tanto su tfidf es muy pequeño, sin embargo el tfidf de las palabras más raras es muy alto.

La clase TFIDF posee varios métodos, entre ellos se encuentran: tfidf, Query y TFIDF_Query. El primero se encarga de calcular el TFIDF de todas las palabras de los documentos, y los guarda en un diccionario donde el Key es la palabra por documento y el Value es su valor de TFIDF. Para calcular este valor utilicé una fórmula:

TF = número de veces que aparece un término en un documento/número total de palabras

IDF = log10 (número total de documentos/cantidad de documentos que contienen a esa palabra)

TFIDF = TF * IDF

El método Query lo que hace es normalizar el query, como mismo se hizo con las palabras de los documentos, y una vez que estén normalizados, los guardamos en una lista. El método TFIDF_Query hace el mismo proceso del método tfidf, pero con la query, pero en este caso nos devuelve un array de doubles con los TFIDF de cada palabra de la query por documento.

Similitud del Coseno (Score)

Otra clase bastante interesante es la llamada SimilitudCoseno, que consta de dos métodos: Matriz y SimCos. Con la implementación de esta clase ya el proyecto busca, quizá no de manera muy eficiente pero llegado a este punto ya puede buscar términos. Primeramente se encuentra el método Matriz (que ayuda a calcular el score o similitud del coseno). Este método coloca en una matriz de n * m, n (cant documentos), m (cant palabras de la query), el TFIDF de cada palabra de la query que se encuentra en el documento, si la palabra de la query no aparece en el documento, entonces esa posición en la matriz es igual a 0. El segundo método utiliza lo que se conoce como Similitud del Coseno. Este es un método para saber cuán similares son dos textos entre sí. Este método devuelve un array de doubles, de longitud igual a la cantidad de documentos. La similitud del coseno consiste en comparar vectores, y si son similares devolverá un valor muy cercano a 1 (o 1), pues quiere decir que sus vectores se superponen, y por lo tanto el ángulo donde el coseno es 1, es 0 grados. Todo lo contrario pasa cuando los vectores no se asemejan en nada, sus vectores serán ortogonales entre sí y devolverá un valor muy cerca a 0 (o 0). Ya yendo a el algorítmo en sí, y luego de haber indagado bastante sobre este tema, para calcular la similitud del coseno entre el TFIDF de la query y la matriz anteriormente hallada, debemos aplicar el producto matricial entre las mismas. Para esto utilicé la fórmula:

simCos = v1 * v2 / |v1|* |v2|

v1 es el valor del TFIDF de una palabra de la query y ***v2*** el valor del TFIDF de la palabra de la query en el documento (valor de esa posición en la matriz).

Snippet

Esta clase tiene dos métodos: BuscarSnippet y Snippet. La clase tiene como objetivo devolver un pedazo de texto, a partir de las diferentes palabras introducidas en el query.

Para esto deberá recorrer todos los documentos y cuando se encuentre la palabra que tenga mayor relevancia del documento, entonces devolverá un pedazo de texto de cantidad máxima igual a 31 aproximadamente, que contenga a esa palabra.

Operadores

Los operadores se crearon para hacer la búsqueda mucho más fácil, por ejemplo, cuando el usuario quiere un término en específico, cuando no quiere que aparezca ese término, entre otras muchas funcionalidades. Implementé diferentes operadores como:

El operador de Anulación (!) que hace que no aparezca el documento que contiene la palabra con el operador ! delante. Este método recorre todos los documentos y si se encuentra la palabra introducida en la query convierte el score del documento en 0.

El operador de Relevancia (*) que hace que se priorice el documento, y mientras más asteriscos tenga la palabra delante, más importancia va a tener. Este método recorre todos los documentos y si se encuentra la palabra introducida en la query multiplica el score del documento varias veces y le suma 1.

También está el operador (^), que significa que esa palabra (documento en donde se encuentre) se tiene que devolver siempre. Este método recorre todos los documentos y si no se encuentra la palabra en el documento analizado entonces convierte el score en 0.

Por último se encuentra el operador de cercanía (~), operador que se tiene que encontrar entre dos palabras, para de esta forma, que devuelva el documento donde estás dos palabras estén más cerca. Dicho operador se implementó de la siguiente manera:

- Primero se almacenó en una lista todas las palabras de la query, incluyendo operadores.

- Luego se indentifica la posición (posiciones) de la lista donde se encuentra el operador (operadores)de cercanía (si se encuentra) y las dos palabras a analizar su distancia.

- Se almacena en un array todas las palabras que contengan al operador, para luego recorrer posición por posición del array, y alamcenar en dos variables la primera y la segunda palabra a comparar. Se analiza a su vez si ellas estan en los documentos. Si están ambas en un documento, entonces se halla la distancia entre los indices donde se encuentran dichas palabras. Asimismo, se compara esta distancia con otras que puedan existir en otros txt, para quedarnos con la más pequeña, y con el indice del documento al que le pertenece dicha distancia.

- Por último, al tener la posición del documento que contiene la menor distancia entre esas palabras, solamente se aumenta el score de ese txt, en este caso se le sumó 2.

Distancia de Levenshtein

Es un algorítmo que calcula la distancia entre las palabras, para ver que nivel de semejanza tiene una con respecto a otra. Su nombre es en honor a Vladimir Levenshtein, científico ruso, quién se encargó de descubrir el útil funcionamiento de este método de sugerencia. En

este proyecto este algorítmo tiene mucha utilidad, pues es empleado con el objetivo de que si el usuario introduce una palabra parecida a una que se encuentra en los textos, Moogle sea capaz de sugerirle el texto que mayor semejanza tenga. La distancia de Levenshtein lo que hace es devolver un número, que mientras menor sea, significa que menor cantidad de cambios (eliminación, inserción, sustitución) hay entre una palabra y otra, por lo tanto esa palabra que menor cantidad de cambios posea, es la palabra sugerida. La clase Levenshtein consta de tres métodos: SimilitudPalabras, MenorDist, Calculo. El método Calculo, es el que devuelve un número (distancia). El método SimilitudPalabras es el método que trabaja con la sugerencia y con los operadores también, en el caso de los operadores lo que hace el método es hallar la sugerencia de la palabra, pero si tiene operadores, guardar las sugerencias con sus respectivos operadores.

Este algorítmo consiste en una matriz de largo igual a la cantidad de letras que tiene la primera palabra (palabra introducida en la query), y ancho (palabra a comparar, palabra del documento). En cada posición de la matriz va un numero que varía en dependencia de que letras (valores) están en ese i, y en ese j. Por ejemplo si hasta ese momento, todas las letras coinciden, entonces el valor que le corresponde a cada casilla de la matriz es 0, pues no hay diferencia entre una cadena y otra. Sin embargo si en un determinado momento hay dos letras que no coinciden, el valor sería distinto de 0, pues ya existiría una diferencia entre una cadena y otra. Mientras más diferentes sean las cadenas entre si, esos números tienden a crecer. Para terminar, la distancia de Levenshtein sería el valor que se ubica en la última posición de la matriz, que no es más que la diferencia que hay entre una palabra y la otra. Si ese valor es 0 significa que las palabras son iguales, si es un número cercano a 0, signfica que se parecen teniendo alguna que otra variación, pero si el número está distanciado del 0, podría significar una diferencia importante entre las cadenas.

Llamada a las clases

Para finalizar se encuentra la clase Moogle, donde llamo a todas las demás clases y métodos, para poder utilizarlos en la ejecución del programa. En esta clase se encuentra el algorítmo que ordena los documentos, desde el más relevante hasta el menos relevante, también trabaja con la similitud del coseno, y aquellos documentos en los que sea un valor igual a 0, los ignora.