***Jean Carlo García Wong C-121***

Con este informe quiero describir con lujo y detalles mi proyecto de programación del primer semestre del primer año de la asignatura de programación.

Primeramente, Moogle! es un buscador de texto mediante el cual el usuario puede realizar una búsqueda inteligente entre varios textos de contenidos diversos ya sea con fines investigativos o solo por curiosidad.

Al ejecutar mi programa el usuario debe ingresar su búsqueda entonces desde ahí cargo todos los archivos de textos que posee mi proyecto y posterior a esto analizo la consulta que ingreso el usuario para saber además si empleó algún operador de búsqueda y eliminar elementos que no presenten relevancia para a partir de eso saber cómo operar en los documentos. Para el análisis de la consulta del usuario hago uso de una función llamada SplitList la cual uso para fragmentarla haciendo un mejor análisis de esta. Después hago uso del método ComputeScore el cual se encarga de darle un valor a la búsqueda por cada término de la consulta en cada documento, es importante recalcar que este método funciona en conjunto con los operadores de búsqueda implementados; para ser más preciso primero hago uso de la función llamada numeroDocs que me da previamente en cuantos documentos se encuentra la consulta ingresada por el usuario, después se analiza la presencia de operadores en la consulta, luego se aplica la funcionalidad del operador, posteriormente se calcula el score de búsqueda con un algoritmo reconocido mundialmente como TF\*iDF para así terminar con el método ComputeSnippet el cual me da una cadena de texto de una longitud aproximada de 50 palabras mediante la cual el usuario puede observar si concuerda con lo que está buscando, pero esto no termina aquí, mi programa es capaz de buscar pedazos de frases que se asemejen a lo que el usuario pide, lo anterior lo ejecuto con la función DamareauLevenshtein la cual mediante una matriz de palabras es capaz de determinar qué tan semejantes son dos palabras en términos numéricos y la función Calculatequery que me retorna la palabra más parecida en el texto. Lo anteriormente dicho se ejecuta si lo que pide el usuario no es encontrado textualmente en los documentos y si existe además una búsqueda parecida a la cual el usuario pudo hacer referencia también.

Luego de haber calculado el score de cada documento haremos el procedimiento de almacenamiento de los documentos en una lista del objeto DocsFinal del cual hablo más adelante, con el método doRanking. Básicamente lo que hace este método es comprobar según los diccionarios usados para el cálculo del score revisar en que documentos se encontró los resultados esperados por el usuario y en cuales no, recalcar que solamente se añaden a la lista de DocsFinal solamente los documentos que tienen un valor de búsqueda. Para lograr lo anterior tuve que hacer uso de una clase DocsFinal la cual me permitiera almacenar el nombre del archivo, su snippet y su score, para posteriormente ordenar dichos documentos según su valor de score de mayor a menor utilizando para esto los métodos CompareTo y Equals en la clase DocsFinal y también usando interfaces IEquatable y IComparable. Por último, se crean los ítems que se devolverán en pantalla que posteriormente verá el usuario y se retornan en el método principal Query.

Para representar los documentos en mi proyecto vi más conveniente usar diccionarios por la facilidad que me generan a la hora de trabajar porque con ellos puedo saber sobre que búsqueda estoy operando gracias al Key-Value pair además me facilita el trabajo con la nueva clase creada DocsFinal la cual me permite almacenar todos los documentos cada uno con sus respectivos datos, como ya había explicado anteriormente.

Este proyecto ha sido de gran utilidad como aprendizaje, me ayudo a mejorar mi enfoque acerca de la programación y a saber la realidad de cómo es el día a día de un programador. Espero que haya sido de utilidad.