Proyecto de Programacion:

Moogle es un buscador que nos permitira buscar frases dentro de un conjunto de documentos. Moogle trabaja de manera inteligente y eficiente haciendo uso de muchos algoritmos y estructuras de datos que nos permiten optimizar el proceso. Moogle ademas cuenta con un sistema de operadores que haran la busqueda mas especifica y permitira un mayor control sobre el contenido a buscar.

Ahora analizaremos como funciona el algoritmo de busqueda y lo dividiremos en varias partes:

I) Lectura:

Primero que todo debemos leer la informacion de los documentos, para esto haremos uso de la **clase TextPreprocessing**, esta clase contiene todo tipo de funciones para leer informacion de documentos, extraccion de palabras, parsing, normalizacion de datos, obtencion de nombres de archivos en un directorio, buscar patrones en un texto, etc.

II) Indexacion:

Una vez leida la informacion a buscar debemos procesarla, para hacer esto haremos uso de las **clases Dictionary**, **WordInfo** y **VectorSpace**. En esas 3 clases se almacenara toda la informacion precalculada de los documentos.

En la **clase Dictionary** se almacenara el conjunto de todas las palabras distintas, sus

funciones basicas son insertar una nueva palabra y buscar el conjunto de palabras mas similares a una palabra dada. Para insertar una nueva palabra lo que haremos es generar algunas subpalabras de ella y almacenarlas y para buscar las mas similares a una dada lo que haremos es generar subpalabras de la misma y quedarnos con las mejores haciendo uso de la funcion de similitud **Score()** que se encuentra en esa clase. **Score()** consiste en una variacion del Edit Distance, pero con otros cambios que le permitiran priorizar prefijos y tener encuenta las posiciones relativas de los caracteres que matcheen.

En la **clase WordInfo** se almacenara todo lo relacionado con los documentos como la cantidad total de ellos, los documentos con mayor TF con respecto a una palabra y para cada palabra la cantidad de documentos que la contienen. Basicamente aqui podremos procesar el TF-IDF para cada palabra lo cual es la frecuencia de esa palabra en el documento multiplicada por el logaritmo de la cantidad total de documentos divido por la cantidad de documentos que contienen esa palabra.

En la **clase VectorSpace** se almacenara el conjunto de vectores de todos los documentos, esta clase nos servira para buscar rapidamente los documentos mas similares a una query dada. Para construir los vectores haremos uso de la **clase Vector**, la cual nos permite crear un vector dado un conjunto de palabras, este vector estara hecho por los pares (dimension, valor) en la cual la dimension es una palabra y el valor es su TF-IDF en ese documento. La **clase Vector** posee una funcion llamada **CosineSimilarity()** que nos permitira hallar el coseno entre dos vectores el cual usaremos como metodo de comparacion para obtener los mejores documentos con respecto a la query.

Cuando indexamos la informacion de los documentos y la almacenamos en la **clase VectorSpace**, tambien tenemos que hacer algo similar en cada documento. El objetivo de esto es optimizar las busquedas dentro de un mismo documento, separando cada documentos en partes o pedazos y tratarlos como documentos independientes para realizar el mismo algoritmo como si fuesen documentos.

III) Procesamiento de la query:

Cuando tenemos una query debemos analizar dos cosas importante:

1- Corregir posibles errores y crear nuevas sugerencias

2- Analizar los operadores de busqueda

Para corregir posibles errores y crear nuevas sugerencias lo que haremos es hacer uso de las **clases Dictionary**, **Request** y **SelectBestQuery**.

En la **clase Dictionary** lo que haremos es para cada palabra de la query buscaremos las mas similares haciendo uso de la funcion **FindWord()**

En la **clase Request** lo que haremos es hacer un request a un sitio web que nos proporcionara sinonimos de las palabras que queramos

En la **clase SelectBestQuery** lo que haremos es combinar la informacion recolectada en las **clases Dictionary** y **Request** para luego generar las mejores sugerencias posibles, tenga en cuenta que a cada palabra se le asigna un peso de similitud por lo tanto esta clase lo que hara es generar las mejores sugerencias tratando de optimizar los mejores pesos de palabras y crear las sugerencias mas similares a la query.

Para analizar los operadores de busqueda haremos uso de la **clase Parsing** la cual analizara la query y nos ayudara a obtener toda la informacion de los operadores tal como las palabras que deben y no deben existir, sus importancias y los pares de cercania.

IV) Busqueda:

Una vez obtenida las posibles mejores sugerencias de la query vamos a tratar de buscar los mejores documentos que tengan relacion con ellas, para esto haremos uso de la funcion **FindSimilarDocVector()** de la **clase VectorSpace** para buscar los mejores documentos, luego de eso haremos los mismo para cada documento y haremos de nuevo el uso de la funcion **FindSimilarDocVector()** para buscar la mejor parte o pedazo de cada documento que guarde mas relacion con la query. Luego de esto al conseguir las partes de los documentos mas relevantes lo que haremos es buscar un patron en esos textos que contenga algunas palabras de la query, para esto hacemos uso de comparaciones entre vectores algo similar a lo que hace la **clase VectorSpace**. Luego de hallar esto retornamos los mejores resultados y el algoritmo queda concluido