Moogle! es un programa que hace busquedas en tiempo real sobre una base de datos de documentos.

En principio necesita una base de datos de archivos en formato .txt ubicados en la carpeta Content.

Cuenta ademas del archivo principal Moogle.cs con 5 archivos .cs necesarios (bueno 4 necesarios para funcionar) que son):
-AuxItems, una clase que guarda el texto y titulo de un documento.
-Methods, una clase que tiene todos los metodos que son invocados en el programa.
-SearchItem, una clase que guarda el titulo, snippet y score de cada documento al hacer la query
-SearchResult, que como su nombre indica devuelve el resultado de la busqueda.
-Matrix, una clase que ayuda en el trabajo con operaciones con matrices, pero que no fue necesario usar en el funcionamiento del proyecto.

Lo primero que hace el proyecto es hacer un arreglo de AuxItems donde en cada posicion hay un documento de la carpeta Content.
Luego duplica ese arreglo pero con los documentos normalizados, esto es, los mismos textos de los documentos, pero sin signos de puntuacion, mayusculas ni saltos de linea.

Ahora creo un diccionario y voy recorriendo los textos de mis documentos. Cada vex que encuentro una palabra "nueva"para mi diccionario la anado con un numero natural que significa que numero de palabra es.
Paralelo a esto voy creando una matriz de (palabras;documentos).
Al terminar esta parte del codigo me queda un diccionario en el que todas las palabras presentes se les asigna un numeroy una matriz de tamano [cant de palabras; cant de documentos]

Aqui empieza la parte algebraica, pues por cada documento voy calculando la importancia que tiene cada una de laspalabras que lo forman,a traves de un metodo llamado TF*IDF. En este punto ya tengo el IDF de cada palabra de mi universo ya calculada (ya que este no varia) y solo llamo al metodo privado TF que calcula el TF de cada una de las palabras de un documento y al multiplicarlas por su respectivo IDF me queda la importancia de esas palabras para ese documento. Luego le asigno ese valor a su posicion correspondiente en la matriz.
Al acabar me queda la matriz con el TF*IDF de cada palabra de mi universo de palabras en cada documento.

En este punto del programa es que valoro la query. Lo primero que hago es normalizarla.
Luego le calculo el TF*IDF para ver que tan relevante es para mi universo. En caso de que no tenga palabras en comun con mis documentos su relevancia va a ser obviamente 0, y a partir de ahi se incrementara en dependencia.

Una vez que tengo el TF*IDF de la query puedo aplicar una formula llamada "Similitud Coseno" que me dice el angulo de desviacion de un vector respecto a otro. Si ese coseno da 0 entonces ambos vectores apuntan al mismo lugar.
Aplicando esto al proyecto. Si trato los documentos como un conjunto de vectores y la query como otro la similitud de coseno me dira que tan parecidos son. Dicho y hecho: creo un metodo que calcula la similitud coseno entre dos vectores y lo llamo tantas veces como documentos tenga. Me queda un array de valores, los scores, a mayor score, mas parecido es ese documento a la query.

Ordeno con un Array de Tuplas los documentos por valor de score y luego creo un array de SearchItem del mismo tamano que la cantidad de documentos. En cada posicion guardo el titulo, texto sin normalizar y score del documento en orden de importancia descendente. Esto provoca que de un documento n en adelante los siguientes tengan score 0, y ocupen espacio por gusto, pero eso estoy intentando solucionarlo.

Al final SearchResult con mi array de SearchItem como argumento imprime en pantalla el Titulo y el Texto de los documentos ordenados de mas a menos por valor de score.