Alumno: Carlos Antonio Bresó Sotto

Grupo: C113

Aquí, el proyecto.

Una vez abierta la página de Moogle, este buscador selecciona en la carpeta content los archivos .txt que mejor respondan a la búsqueda insertada. Todo el trabajo esta principalmente en Moogle.cs y Tools.cs, clases donde están todos los métodos que usa el buscador.

Requisitos para que funcione:

Para empezar la carpeta “Content” debe de tener al menos un archivo .txt, puse algunos libros allí, pero puedes cambiarlos por cualquier otro cuerpo de búsqueda.

No se debe borrar el archivo “Sinonimos.json” que esta dentro de MoogleEngine. (si desea ver resultados mas cercanos al query puede dejar en comentario la línea 30 de Program.cs y la búsqueda trabajara sin tener en cuenta sinónimos )

Funcionamiento de búsqueda:

Antes de cargar la pagina del buscador, el programa realiza 4 acciones:

Guarda los nombres de los txt en un string [].

Lee y guarda los archivos en un string[].

Lee y guarda todas las palabras en un diccionario, donde a cada palabra le corresponde su frecuencia de aparición en todos los txt.

Se monta un diccionario de sinónimos, donde cada palabra esta relacionada con un conjunto de sinónimos.

Librerías:

Nota: Ciertas de las librerías usadas solo se prestan para resolver una muy pequeña parte del programa.

System

System.Diagnostics

System.Collections

System.Text.RegularExpressions

System.Linq

System.Text.Json

En Moogle.cs hay principalmente 3 metodos:

Query

Search

Search\_Suggestion

Cada uno conceptualmente diferenciado. Y estos se apoyan en los métodos que están en la clase Tools.cs, creada solo por comodidad.

Query:

Una vez insertado el query a buscar, se limpia el mismo de signos de puntuación que estorban y se lleva a minúsculas (siempre que se piensa en comparar strings, se lleva todo a minúsculas). Este query limpio se deja en un string [] y se concatena con otro string [] , donde están los sinónimos de las palabras del query. Y todas estas palabras son las que se pasan como parámetro al método de búsqueda Search().

Search:

Este método deberá devolver un SearchItem [] donde estarán los documentos donde aparecen las palabras a buscar organizados por su relevancia. Para ello primero se calcula el Idf de cada palabra y se guardan estos valores en un int[]. Se calcula el Tf de cada palabra en los textos previamente leidos, el score de cada objeto será el producto Tf\*Idf, el snippet se escoge a partir de la primera aparición de la palabra en el texto ( 50 espacios a la izquierda y 50 a la derecha, se tiene en cuenta si la palabra esta en el mismo inicio o final del texto), el Title correspondiente esta en el string [] de nombres de los documentos.

Ahora solo queda ordenar. Pasa que si varias palabras aparecen en un mismo texto se crean varios SearchItem para un mismo documento, entonces al final de la búsqueda se unifican todos los SearchItem con el mismo titulo sumando su score y manteniendo el snippet del SearchItem con mayor score.

Luego se ordena la lista según el score y se transforma a SearchItem[] y se devuelve este como resultado.

Operadores:

Los operadores están todos dentro de este método y modifican la búsqueda según cual se use.

Se juega mucho con que las palabras a buscar vienen en un string[] y están libres de todos los signos de puntuación excepto los propios operadores, de manera que estos están al inicio de cada palabra o separados de al misma mediante un único espacio en blanco.

Ejemplos de uso:

(Para: \*, ¡, ^)

pizza !piña

\*\*\*amor Julieta

Watson ^Holmes

(Para: ~ ) Algoritmo ~ Euclides

Solo se deben de usar así y en los casos de \*, ! y \* estos van sin separarce de la palabra asociada porque son trarados como parte de la misma, luego se detecta que la palabra lleva un operador (varios para el caso de \*) y se efectúan (Los operadores !, \*, ^ son dificiles de explicar sin código delante)

Operador de Cercanía:

Este entra en acción justo cuando terminamos de llenar la lista de SearchItems. Primero se identifican las palabras afectadas, luego se llama al método del operador con las palabras y la lista como parámetros y el método multiplicara (por un valor mayor que 1) el score de los documentos que contengan ambas palabras según la distancia que tengan en el documento ( si cada palabra aparece varias veces solo se tomara en cuenta las apariciones más cercanas)

Search\_Suggestion:

Siempre se devuelve una sugerencia sin importar si la búsqueda es enteramente válida. La sugerencia devuelve un string donde en están en mismo orden las palabras más “parecidas” a la frase del query insertado.

La manera de hallar una palabra mas parecida a otra es mediante la distancia de Levenshtein(buscar en google) . Entonces, gracias a que tenemos un diccionario con todas las palabras que existen en el cuerpo y sabemos las del query, solo queda comparar y nos quedamos con la que tenga menor distancia. Así sugerimos una frase compuesta por las palabras con la menor distancia Levenshtein.