Esta es mi versión del Moogle!, un buscador de texto al que se le agregan documentos .txt antes de ejecutarlo y que tiene las particularidades siguientes:

El proyecto retorna los títulos, con la especificación de que son .txt, de los documentos dentro de los que aparezca al menos una parte del query que se busque.

El score no aparece al devolver los textos, sin embargo es el encargado del orden de relevancia en el que se presentan los documentos, que se muestran organizados descendentemente de acuerdo a su score.

Está hecho para que analice textos en español, pues en la búsqueda del snippet evita que las palabras de poca relevancia como artículos o preposiciones sean buscadas, si esto no es de importancia para el usuario puede emplear textos en ingles, solo que un solo idioma a la vez.

El snippet que se retorna es la primera linea en la que aparece la palabra (o frase, pues esta si aparece completa es prioritaria), sin embargo dentro de la línea devuelta puede ocurrir que si buscas una palabra como "casa" que es la raíz de muchas palabras, la coincidencia del snippet (no para determinar el score), pueda ser una palabra como "casada". Además cuando la query cuenta con palabras comunes (como artículos, conjunciones o preposiciones) y estas no fueron completamente eliminadas por el idf al no aparecer en todos los documentos, te devuelve el título de aquellos documentos en los que aparece pero el snippet es la primera línea del documento; esto cambia si la búsqueda es únicamente esta palabra (como por ejemplo solo buscar "el" y que no se cancele por su idf) en este caso si devuelve la primera línea en la que aparezca dicho artículo.

También cuenta con la implementación de operadores los cuales se presentan como chars delante de la palabra cuya implicación en la búsqueda queremos modificar, se ha de tener en cuenta que cada operador puede ser presentado por una sola de las palabras para que funcione correctamente, estos operadores son:

- * : este es el operador de relevancia, la palabra que lo incluya obtiene un mayor tf al momento de calcular el tf×idf del query, mientras más veces esté repetido este char delante de la palabra más relevancia tendrá.

- ^ : este operador define que los documentos devueltos tienen que tener la palabra a la que anteceden, de manera tal que aunque el documento tenga relevancia por presentar otras palabras de la búsqueda, si no tiene esa en específico no debe ser devuelto por el buscador.

- ! : este operador es el contrario del anterior, pues si la palabra a la que antecede se encuentra en el texto así sea una sola vez este debe ser

descartado completamente.

A continuación se hace un desglose de las clases creadas y los métodos implementados dentro de las misma con una pequeña explicación de su función y proceso:

Class Obtein

Constructores:

- string[ ] documentos
- Dictionary<string, int[ ]> vocabulary
- Dictionary<string, string[ ]> words
- int[ ] wordsCount

Métodos:

Vocabulary (Dictionary<string, int[ ]> vocabulary)

Devuelve un diccionario donde cada key es una string que representa una palabra única, asociada a un array de int[ ] con tamaño igual a la cantidad de documentos rellenado con la frecuencia que le corresponde a la palabra en cada documento.

NormalizeText (string[ ] words)

A partir de un string (ya sea una sola palabra o un texto completo) pone todas las palabras en minúscula, elimina las tildes y los signos ; después crea un array de string en el que las palabras están separadas y sin espacios.

CosineSimilarity (double a)

Es una fórmula exacta sacada de internet. Recibe dos array double[ ] que funcionan como vectores para hallar la similitud de cosenos entre ellos y que devuelva un número que sería el score entre un documento y la query introducida.

Words (Dictionary<string, string[ ]> words)

Devuelve un relación entre la dirección de un *.txt* y el texto que contiene ya normalizado y dividido en palabras. Tiene tantas key como documentos se analizan y las palabras de cada string[ ] no coinciden.

WordsCount (int[ ] wordsCount)

Devuelve un array con las cantidad de palabras que conforman los textos de cada documento.

ComunWords (string[ ] palabrasComunes)

Contiene los artículos, preposiciones y conjunciones y adverbios para eliminarlas del array introducido, a la vez lo normaliza y divide en palabras sueltas. Solo se utiliza en el snippet para que en la búsqueda de la sección del texto en la que aparece el query no se traten de buscar palabras sin relevancia.

FindLineNumber (int a)

Lee el texto de la dirección que se le introduce, lo lleva a minúscula y devuelve el numero de la linea en la que se encuentra el string que recibe, en caso de no hallarse devuelve -1.

GetLine (string line)

Lee el texto de la dirección introducida y devuelve todo el texto que aparece correspondiente a la linea del número dado.

Oper (string x)

Devuelve del string introducido una sola palabra (la primera en caso de ser más de una) que inicie con el char que se busca.

Class TextAnalize

Constructores:

- double[ ] idf
- double[,] tf_idf
- Obtein a

Métodos

TF (double[,] tf)

Devuelve un array doble con la misma cantidad de columnas que palabras en el vocabulario y número de filas igual a la cantidad de documentos a analizar.

Como el diccionario del vocabulario ya contiene al frecuencia en la que aparecen las palabras únicas, lo que hace este método es pasar por cada key y extrae el array de valores de frecuencia para aplicar la formula de tf (frecuencia de la palabra en el documento entre cantidad de palabras del documento), este último valor obtenido del int[ ] wordsCount.

IDF (double[ ] idf)

Devuelve un array de tamaño igual a la cantidad de palabras únicas.

Se utiliza el diccionario del vocabulario para definir, pasando por cada key, en cuantos de los valores la frecuencia es distinta de cero y por tanto en cuantos documentos aparece la palabra, el total de elementos del int[ ] indica cuantos documentos están siendo analizados.

Con estos valores se utiliza la formula matemática estándar para calcular el idf, y una vez obtenido el resultado este se coloca en el lugar correspondiente del array según la posición de la key con la que se trabajó.

TF_ IDF (double[,] tf_idf)

Devuelve una array doble del mismo tamaño del tf.

El cálculo que se realiza es que multiplica cada columna del array de tf por el valor del indice correspondiente del idf.

QueryVec (double[ ] queryVector)

Devuelve un array del tamaño del vocabulario en el que las únicas casillas con valor distinto de cero son las correspondientes a palabras coincidentes con el query, si ninguna coincide el valor del array será todo cero y al calcular el score no se devolverá ningún documento.

Se utiliza el vocabulario para hallar las coincidencias lo que indica en que parte del array se deberá colocar el valor de tf×idf. Se revisa cuantas veces esta la palabra en el query y con esa información se calcula el tf, luego se multiplica con el valor del array de idf previamente establecido, y el resultado se coloca en el lugar ya definido dentro del array queryVector.

Se analiza si alguna de las palabras del query presenta * con el método *Obtein.Oper*; en caso de que el valor devuelto por este método no sea nulo, se

lleva la cuenta de cuantas veces se repite ese char en la palabra, ese resultado se multiplica por 2 y se suma a la cantidad de veces que se repite la palabra en el query para aumentar se tf y su relevancia en la búsqueda.

Score (Dictionary<string, double> score)

Devuelve un diccionario con el tamaño de la cantidad de documentos analizados siempre que su score sea mayor que cero, en el que la key es la ruta del documento y el value el valor de score que le corresponde dependiendo de la query introducida.

Para calcular el score se crea un nuevo array del tamaño del vocabulario que se llena con cada fila del array doble de tf_ idf a medida que se cambia de documento, este se utiliza para con el queryVector realizar la similitud de cosenos que s el número que se busca.

Antes de devolver el diccionario se llama al método *UnimportantText* que contiene los documentos que deben ser eliminados de la devolución en caso que se hayan introducido operadores en la búsqueda; si esta lista no es nula se itera a través del diccionario de score y comprobando con las key del mismo y los elementos de la lista remove para quitar las entradas correspondientes.

Snippet (string line)

Devuelve la primera linea del texto en la que se encuentra una coincidencia con el query, en caso de obtener ninguna devuelve la primera linea del texto.

Para buscar el query primero lo busca como frase completa y si no arroja ningún resultado, busca palabra por palabra, sin contar los artículos, preposiciones y conjunciones que ya fueron eliminados al convertir el query de string a string[ ].

Utiliza los métodos *Optein.FindLineNumber* para saber en que linea se encuentra y *Obtein.GetLine* para que devuelva el texto solicitado en un string.

Query (SerchItem[ ] (title, snippet, score))

Utiliza el método *SerchItem[ ]* preestablecido, ahora con tamaño igual a la cantidad de documentos analizados. El title es el nombre del documento, ya separado de las indicaciones de la dirección para encontrarlo, mantiene el tipo de documento que es: en este caso *.txt* . El snippet es obtenido a través del método *Snippet*. Se asocian estos valores con una iteración del Dictionary<string, double> score para asociar en cada caso el titulo, la sección en la que aparece el query en el documento y el score que le corresponde.

UnimportantText (List<string> remove)

Devuelve una lista con las direcciones de todos los documentos que deben ser descartados en la devolución de la búsqueda dependiendo de los operadores introducidos en el query,

Analiza el query en busca de palabras antecedidas por los char ^ y ! , en caso de alguna de estas no ser nulas se procede a normalizar las palabras para eliminar los símbolos y poder buscarlas dentro de los textos analizados. En caso que la palabra que contenía el char ^ no se encuentre en el texto se añade la dirección del documento a la lista. En el caso del char ! si la palabra se encuentra en el texto y su dirección no ha sido ya añadida a la lista se incluye entre los elementos de remove.

Class Moogle

Constructor:

- TextAnalize textAnalize

Modificación del método:

SearchResult Query

Devuelve el título y el snippet asociado.

Dentro de este método se llama al método *TextAnalize.Query* y se organiza por descendentes dependiendo del score.

Class Matriz

Constructor:

- int[,] matrix

Métodos:

MatrixAdd (int[,] a)

Para dos matrices del mismo tamaño suma los elementos de posiciones iguales [i,j] y los coloca en esa misma posición del nuevo array doble.

$$A, B \in M_{m\times n}(K), C = A + B \Leftrightarrow c_{ij} = a_{ij} + b_{ij} \forall i, j.$$

MatrixMult (int[,] b)

Para dos matrices MxN y NxP, se crea un nueva matriz de dimención MxP. El cálculo se realiza de la siguiente manera:

$$A \in M_{m\times n}(K), B \in M_{n\times p}(K), C = AB \Leftrightarrow c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \forall i, j.$$

MatrixVec (int[ ] c)

Se multiplica algebraicamente, moviendo los índices de posición según corresponde, una matriz con un vector, siempre que la matriz tenga tantas columnas como valores posee el vector. Devuelve un array o vector de tamaño igual a la cantidad de filas que tiene la matriz.

$$A \in M_{m\times n}(K), B \in M_{n\times 1}(K), C = AB \Leftrightarrow c_{i} = \sum_{j=1}^{n} a_{ij}b_{j} \forall i, j.$$

MatrixEsc (int[,] d)

Se obtiene una nueva matriz del mismo tamaño de la matriz introducida, lo que cambia en que cada valor de la matriz original es multiplicado por un número x. La operación algebraica seria la siguiente.

$$Sean : A \in M_{m\times n}(K), \alpha \in K \Rightarrow \alpha A = (\alpha a_{ij}) \forall i, j.$$

Class Vec

Constructor:

- int[ ] vector

Métodos:

VecMult (int e)

Se multiplica algebraicamente dos vectores (dos arrays) y se obtiene un número. Para calcular se realiza $e = \sum a_k b_k$ siendo k el tamaño de los vectores.

VecEsc (int[ ] f)

Se multiplica un vector con un escalar y se obtiene un nuevo vector en el cual los valores del vector original han sido multiplicados por un numero x.

EJEMPLOS

Aquí se presentan algunas búsquedas de prueba realizada para comprobar la funcionalidad y eficiencia del proyecto:

Query: harry lupin

- 3-harry-potter-y-el-prisionero-de-azkaban.txt

  ... Harry Potter ...

- 5-harry-potter-y-la-orden-del-fenix.txt

  ... Harry Potter ...

- 1-harry-potter-y-la-piedra-filosofal.txt

  ... Harry Potter ...

- 2-harry-potter-y-la-camara-secreta.txt

  ... Harry Potter ...

- 4-harry-potter-y-el-caliz-de-fuego.txt

  ... Harry Potter ...

- 6-harry-potter-y-el-principe-mestizo.txt

  ... Harry Potter Y El Príncipe ...

Query: harry !lupin

- 1-harry-potter-y-la-piedra-filosofal.txt

  ... Harry Potter ...

- 2-harry-potter-y-la-camara-secreta.txt

  ... Harry Potter ...

Query: harry ^lupin

- 3-harry-potter-y-el-prisionero-de-azkaban.txt

  ... Harry Potter ...

- 5-harry-potter-y-la-orden-del-fenix.txt

  ... Harry Potter ...

- 4-harry-potter-y-el-caliz-de-fuego.txt

  ... Harry Potter ...

- 6-harry-potter-y-el-principe-mestizo.txt

  ... Harry Potter Y El Príncipe

Query: Julio Verne

(las palabras pueden ser puestas con mayúscula o minúscula y se obtiene el mismo resultado)

- Julio Verne - Aventura de tres Rusos y tres Ingleses.txt

  ... Julio Verne ...

- 10. EL REMEDIO ANEXIONISTA.txt

  ... EL REMEDIO ANEXIONISTA. PATRIA, 2 de julio de 1892 ...

- Julio Verne - Ante la bandera.txt

  ... Del 5 al 25 de Julio.- Han transcurrido dos semanas ...

En el último caso la palabra que coincide es únicamente "julio", pues "Verne" no aparece referenciado en el documento. Como se puede observar a la hora de devolver los documentos se le da mayor importancia (por lo que sale primero) a aquel en el que aparece la frase completa para después buscar la coincidencia de al menos una parte del query

Query: 10

- 02. DISCURSOS CONMEMORATIVOS 10 DE OCTUBRE.txt

  ... DISCURSOS CONMEMORATIVOS AL 10 DE OCTUBRE (FRAGMENTOS). JOSÉ MARTÍ. ...

- 01. LECTURA EN STECK HALL.txt

  ... 10 ...

- Julio Verne - Aventura de tres Rusos y tres Ingleses.txt

  ... 10 ...

- Julio Verne - Ante la bandera.txt

  ... 10 ...