Proyecto de Programación I. Moogle!
Facultad de Matemática y Computación
Universidad de La Habana . Curso 2022

Nombre: Dennis Daniel González Durán *Grupo*: C121

En este proyecto realizaremos una aplicación cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos utilizando la tecnología .NET Core 6.0 y el lenguaje de programación C#.
La búsqueda se basa en la utilización de la Frecuencia de Término – Frecuencia Inversa del Documento (TF-IDF). Como forma de comparación principal se utiliza el método de la similitud de cosenos, el cual permite conocer qué tan parecidos son dos documentos a través del ángulo que existe entre ellos luego de ser expresados en forma vectorial.

Además de las clases definidas en la orientación del trabajo, el proyecto usa la clase InfoDocs, dedicada al manejo de la información que se puede obtener a partir de los documentos (TF, IDF y palabras cercanas entre sí). También utiliza la clase MakeSearch donde se realiza todo lo referente a la obtención de la representacion vectorial del query y los documentos, así como el cálculo del ángulo entre vectores.
Se procede de la siguiente forma:

Durante la ejecución del servidor se inicia el trabajo en la clase InfoDocs:

Método GetPath: obtendrá la ruta de donde se almacenan los documentos.
Método GetFiles: guarda la dirección de cada uno de los documentos en un array.

Con el trabajo de estos 2 métodos tendremos un array con la dirección de cada documento.

Método Manejartexto: Separa un string en un array de palabras con el mismo formato de escritura (minúscula y sin tildes).
Método ReadFiles: reconoce el texto de cada documento a paritr del array de direcciones y luego de separar las palabras con el método ManejarTexto almacena cada conjunto de palabras en una lista.

Con la ayuda de un ciclo y estos 2 métodos tendremos una colección llamada ListadeListas que almacenará las palabras de todos los documentos.
Luego se utiliza el **Método CreateD** que nos permitirá tener un diccionario con cada palabra leida hasta el momento y su TF e IDF en cada documento.
Por último, se ejecutará el **Método NearWords**, que almacenará en una colección las palabras cercanas a cada palabra previamente leida.

La ejecución del programa desde este momento se centra en las clases Moogle y Makesearch:

Método NormalizeQuery: Recorre el query y con el método **Manejartexto** crea una colección con las palabras del query.

Método AnalizeQuery: Rellena una lista con las palabras de la query modificadas por algún operador.
Método DataVector: Crea un array con el TF IDF de las palabras del query, que representará el query en forma de vector.
Método ImportantOperator: Modifica el array anterior según las palabras modificadas por el operador de importancia.

Con la ayuda de estos métodos habremos almacenado las palabras del query, sus modificaciones por los operadores empleados, así como el TF IDF de estas para realizar la búsqueda.

Método MagnitudeQuery: Calcula la Magnitud del query expresado como vector.
Método Sumapunto: Calcula el producto vectorial entre el vector query y un vector documento.
Método MagnitudeDoc: Calcula la Magnitud de un documento expresado como vector.
Método Similarity: Calcula el coseno del ángulo entre el vector del query y el vector de un documento.

Utilizando estos métodos y un ciclo podremos almacenar en un diccionario llamado *Match*, qué tan parecido es cada documento con nuestro query.
Luego este diccionario será modificado por el método **Operators**, encargado de adaptar la búsqueda a los operadores introducidos.

Método MostImportantWord: recorre el query e identifica la palabra con mas importancia en este.
Método ImportanceAsignation: Asigna a cada documento la palabra del query sobre la cual estará basado su snippet.
Método Snippet: Crea un string con palabras de un documento a partir de una palabra central perteneciente a la query.
Método H_Distance: si una palabra no aparece en ningún documento, identifica una palabra cercana a esta para sugerirla en la búsqueda.

Mediante un ciclo, los métodos antes mencionados permiten dar el toque final a la búsqueda, añadiendo a cada resultado un snippet, así como una sugerencia sobre qué buscar en caso de errores de escritura.

Bibliografía:
https://es.wikipedia.org/wiki/Tf-idf
https://stuka2013.wordpress.com/2013/01/30/algoritmo-distancia-hamming-levenshtein-en-c/
https://es.m.wikipedia.org/wiki/Similitud_coseno
Miguel Katrib. “Empezar a programar. Un enfoque multiparadigma con C#”.