Proyecto de primer semestre de Programación de la Facultad de Matemática y Computación de la Universidad de La Habana.

Ernesto Castellanos Vázquez

C – 113

Flujo de Datos:

La ejecución del programa comienza cuando desde el método Main de la clase program de MoogleServer se realiza una llamada a al método mainLoader de la clase Loader
esta clase posee tres propiedades estáticas que son necesarias durante toda la ejecución del programa:

matrix: una representación de las palabras en cada documento como un array de diccionarios cada posición del array es un documento diferente, cada documento se representa como un diccionario de pares palabra como key y una lista de objetos de tipo Position como value (que es solo un objeto usado para almacenar la posición de cada palabra) esto se realiza de esta manera ya se debe acceder a los documentos de forma secuencial y a la lista de posiciones según la palabra que le corresponda.

documents: array que guarda la ruta y nombres de cada documento.

weight: guarda el peso de cada palabra en cada documento(cada posición del array representa un documento diferente como un diccionario de pares palabra-peso) , utilizado para calcular la similitud cosenica.

El método mainLoader comienza cargando en la propiedad documents la ruta completa y nombre de los documentos luego utilizando un ciclo recorre cada uno de ellos y mediante una instancia de la clase StreamReader guarda cada linea en una variable, entonces separa las palabras de esta linea y las guarda como una key en el diccionario respectivo de matrix y coloca la posición en la lista del value en caso de que la palabra ya exista simplemente inserta la nueva posición, luego en otro ciclo recorre cada diccionario(documento) de matrix y cada palabra de esos diccionarios, para calcular el valor TFIDF de esa palabra en ese documento y almacenarlo en la propiedad weight como par palabra-peso en el documento.

Luego de esto el flujo vuelve al programa principal el cual realiza al realizar una búsqueda envía esta frase a Moogle.Query, este debe separar cada palabra de la frase guardando su posición y crear una frase para enviar como la sugerencia, para esta frase mientras comprueba cada palabra, en caso de que no exista llama a Distance.minDist que le devuelve la palabra mas parecida a esta de todas las presentes en los documentos, el método también crea los objetos searchItem que almacenan los nombres de los documentos, extraidos de la propiedad documents de la clase loader , crea el snippet de la ultima palabra de la búsqueda presente en el documento llamando al método snippet de la clase score, guarda el valor del peso del documento en la consulta, multiplicando el valor devuelto por Score.score por el valor de los posibles operadores y ordena los resultados de la búsqueda según este peso eliminando los documentos con valor nulo, ademas de esto existen las siguientes clases.

La clase Operator que permite modificar el valor del score de cada documento según la existencia de operadores y posee los métodos siguientes:

operatorValue: este método recibe gran cantidad de argumentos que son necesarios para implementar los operadores, utiliza la instrucción switch para escoger que método llamar según el carácter que aparece a la izquierda de cada palabra, menos el de cercanía ya que este se comprueba si esta presente al final de la palabra, y se asume que si el usuario introdujo este operador es porque desea que el documento devuelto contenga ambas palabras es por eso que si una de las dos no esta presente se devuelve 0, para el operador de importancia solo devuelve la cantidad de asteriscos mas 1(ya que si solo aparece un asterisco retornaría 1 como valor de importancia).

close: para el operador de cercanía utiliza un algoritmo similar a la parte merge del merge-sort para encontrar según las posiciones de las palabras el momento en el que estas estaban mas cerca entre si y devolver esa distancia;

isNot: simplemente devuelve true si la palabra no se encuentre en el documento, sirve tanto para el operador de existencia como para el no existencia.

La clase Score se usa principalmente para obtener el valor de los documentos en una búsqueda, contiene los métodos:

IDF: devuelve el IDF de la palabra solicitada para lo cual recorre cada uno de los documentos y comprueba si la palabra esta presente y devuelve el logaritmo del cociente entre la cantidad total de documentos y la cantidad total de documentos que contienen el término.

TFIDF: se vale del método anterior para calcular el tf-idf multiplicando la cantidad de elementos de la lista de posiciones(la cantidad de posiciones indica la cantidad de ocurrencias de esa palabra en el documento) por el idf del documento.

M: método sobrecargado que entrega la magnitud(valor utilizado para el calculo de la similitud cosenica) elevando cada termino al cuadrado o multiplicándolo por el termino correspondiente y sumando todos estos valores a medida que se encuentran.

score: devuelve el valor del documento para esa búsqueda valiéndose de la similitud cosenica calculada con la ayuda del método M.

snippet: accede a la posición donde aparece la primera ocurrencia de la palabra, posición que se encuentra en la lista de posiciones de la propiedad estática matrix de la clase Loader y mediante StreamReader lee el documento y guarda esa linea entera y la devuelve al método que la llamó.

La clase Distance tiene como función devolver la palabra mas cercana a la que se le solicita de todos los documentos para esto tiene dos métodos:

lev: este calcula la similitud entre dos palabras con un proceso recursivo que permite implementar el algoritmo levenshtein.

minDist: este recibe la palabra a buscar y mediante una serie de ciclos anidados compara esa palabra con cada una de las almacenadas gracias al método anterior y devuelve la mas parecida.

La clase ResultComparer hereda de IComparer y tiene como única función servir de criterio de comparación para ordenar los objetos de tipo SearchItem que debe devolver el método Query.