Proyecto Moogle

Informe Escrito

Rodrigo Mederos González

C-111

En este resumido informe se realiza una breve descripción de la arquitectura básica del mismo, flujo de datos, las funcionalidades implementadas respecto a los operadores existentes en el programa, modelación empleada para representar los documentos y consultas, la función para organizar en una jerarquía de importancia los resultados de la búsqueda y así como el funcionamiento de los algoritmos de mayor complejidad que aparecen en la implementación.

1. Arquitectura básica del proyecto y flujo de datos durante la ejecución de la búsqueda

El proyecto, ubicado en la carpeta Moogle, tiene su corazón en la carpeta Moogle Engine, donde radica el 99% del trabajo realizado. Para el desarrollo de este trabajo backend, en Moogle Engine coexisten 5 archivos de CSharp (.cs), de los cuales tres fueron entregados por defecto por el proveedor de esta estructura inicial del proyecto en la descarga de Github, y su código fue relativamente adaptado en una medida ínfima para cumplir el objetivo del proyecto. Me refiero a Moogle.cs, SearchItem.cs y SearchResult.cs.

Son añadidos durante la fase de desarrollo dos pilares fundamentales de la búsqueda:

* Preprocessing.cs: consta de tres clases. La clase MyDirectory es la más sencilla de todo el proyecto; en su cuerpo de código solo aparecen las tareas de guardar la dirección de la carpeta donde está contenida la base de datos donde trabajará el resto del programa y la dirección de cada uno de ellos. La clase Document por otra parte se encarga de procesar y almacenar la información suficiente y necesaria de cada documento para la meta de la aplicación como es el caso de la frecuencia de términos (Term Frequency o TF) y parecido la clase AllDocuments para el caso de la frecuencia inversa de los términos(Inverse Frequency o IDF), que juntos permiten aplicar el modelo de búsqueda TF-IDF en que se basa este trabajo, el cual se explica más adelante y con más detalles
* Query.cs: con más de 500 líneas de código, este archivo lleva gran parte de la algoritmia compleja que hace realidad los deseos del usuario para con este software. Tomando el trabajo hecho por las clases anteriormente mencionada y con una única clase que lleva el mismo nombre, aquí están implementadas funciones de normalización o tokenización también de la búsqueda pedida por el usuario, TF-IDF a la misma, comparación de vectores, procesamiento de operadores, localización de un snippet válido y dinámico y la creación de sugerencias en caso de resultados escasos o insuficientes a lo requerido por el usuario.

El flujo de información es sencillo: la clase MyDirectory recibe la dirección de la base de datos, la convierte en las direcciones específicas de cada documento, la cual es utilizada por AllDocuments que a su vez usa para cada .txt la clase Document para así procesar la información a nivel general y particular. Luego la clase Query toma la entrada de la búsqueda (query) del documento Moogle.cs y utilizando toda la información preprocesada elabora una lista con los mejores resultados teniendo en cuenta los operadores, que hayan sido usados por el usuario e implementado para tener un impacto en el resultado final de la búsqueda, el snippet y la sugerencia (Fig. 1.1). Finalmente SearchItem.cs y SearchResult.cs se encargan de almacenar y mostrar los resultados de la búsqueda

SearchResult.cs

SearchItem.cs

Query.cs

Moogle.cs

query

MyDirectory.cs

Document.cs

AllDocuments.cs

Content

Fig. 1.1

1. Las funcionalidades a destacar implementadas en el proyecto es en el caso de los operadores y el snippet es el uso de la biblioteca de C# System.Text.RegularExpressions, con la cual la búsqueda se hace más fácil e incluso abre la oportunidad de construir expresiones más largas y complicadas como con el snippet donde se combina en la función CreateSnippets de la clase Query parte del texto donde aparecen dos palabras de la query que están muy alejadas en el documento original
2. El modelo usado empleado para la consulta es el modelo vectorial basado en la recuperación de información, por TF-IDF, abreviatura del término frecuencia–frecuencia inversa del documento (en inglés), y es una estadística numérica que pretende reflejar qué tan importante es una palabra para un documento en una colección. Su valor aumenta proporcionalmente al número de veces que aparece una palabra en el documento y se compensa con el número de documentos en el conjunto que contienen la palabra, lo que ayuda a ajustar el hecho de que algunas palabras aparecen con más frecuencia en general.

La estructura de datos principalmente usada para representar tanto el TF en los documentos, el IDF y formar los vectores que serían comparados son los diccionarios por su ventaja al no permitir palabras repetidas y de trabajo llave-valor.

La función ranking se apoya en la fórmula de similitud de vectores como aparece en la Fig. 1.2 siendo dj el vector formado por el valor del TF-IDF de cada palabra de la query en el documento j-ésimo y q el vector del TF-IDF de la query. Luego estos son ordenados en una lista y en caso de no aparecer resultados se elabora una sugerencia al usuario de una búsqueda probablemente más acertada a lo solicitado.



Fig. 1.2

1. El algoritmo de mayor nivel de complejidad utilizado en el proyecto es sin lugar a dudas el Edit Distance o  Levenshtein Distance. El objetivo de dicha idea es saber cuál es la mínima cantidad de operaciones que deben ser realizadas para convertir un string “x” a otro “y” solo pudiendo añadir, eliminar o cambiar caracteres. Este es un algoritmo de Programación Dinámica (Dynamic Programming o DP) en el cual se analiza sobre una matriz colocando las palabras a los lados como en la Fig. 1.3, y en cada estado donde comparamos la letra en la posición i con la de la posición j se toma la decisión de cuál operación es más “barata” entre cambiar la letra, quitarla completamente o cambiarla por otra, para convertir poco a poco la palabra inicial en la otra, y su parte de DP salta a relucir cuando como se guarda en la posición de la matriz por la que ya se pasó la opción con menor coste operacional, no hace falta volver a calcular dicho coste desde cero, sino quedarnos con el mínimo de los tres estados anteriores.

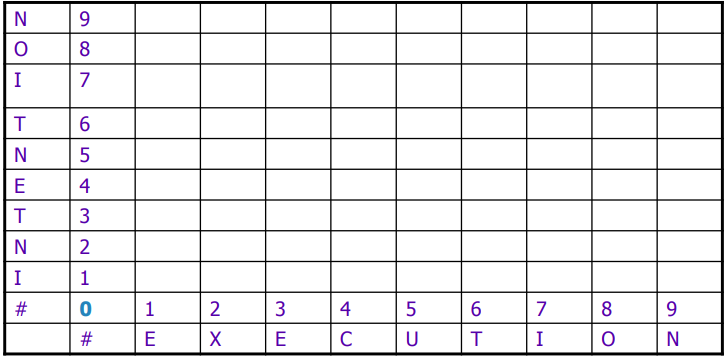
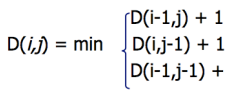




Fig. 1.3