Proyecto Final CCOM 3034: Estructuras Fundamentales
Por: Rafael Esparra Rivera 801 08 1826
Roxana González Burgos 801 10 2806
Fecha: 12 de diciembre de 2012

I. Introducción

El propósito de este proyecto es crear un sistema de búsqueda simple. Luego de procesar el contenido de un conjunto de documentos, de formato .txt, de un directorio, se le permite al usuario hacer una búsqueda de una frase en los archivos de textos. Luego se le devuelve al usuario una lista de tres archivos con la frecuencia mas alta.

II. Contribuciones de cada miembro del grupo

Rafael Esparra se encargo de los algoritmos de creación de la Tabla de frecuencia y de búsqueda en este. Roxana González trabajo en la implementación de el algoritmo de búsqueda para mas de una palabra y en el procesamiento de los datos removiendo las palabras vacías(en ingles “stop words”) y los signos de puntuación.

III. Resumen del Diseño

El cliente es llamado client.cpp, el mismo, al ejecutarse, automáticamente baja los archivos de texto que están en un directorio previamente destinado por el cliente. Una vez procesados los datos, se le permite al usuario realizar búsquedas de frases y son desplegados aquellos tres archivos con mayor relevancia a la búsqueda.

Clase SearchEngine:
La clase SearchEngine se ocupa de realizar las búsquedas dado un conjunto de archivos de textos guardados en un directorio, la misma se ocupa de procesar la data y organizarla.

Métodos:
La clase debe estar sincronizada con el cliente, con la clase ParsedFile.h y con el archivo de texto “stop-words.txt”, pues necesita de que el directorio sea definido y obtener los datos de los archivos para que se pueda realizar una búsqueda. Este proceso se realiza utilizando las funciones descritas adelante.

1. FetchFile(): Función que dado un directorio, guarda toda la data obtenida de los archivos de textos guardados en ese directorio, en mapas y vectores para facilitar la búsqueda de la frase de la cual el usuario quiera hacer la búsqueda.

2. StopWords(): Función que abre un archivo adjunto llamado “stop-words.txt” y guarda en un vector todas las palabras que se denominan palabras vacías. La misma será útil a la hora de procesar la data para excluir las mismas en futuros procesos.

3. SearchKey(): Función que recibe una frase del usuario y busca en la data previamente procesada en un mapa y despliega en que archivos la frase tiene mas relevancia.

4. sortResult(): Función que ayuda a organizar el mapa de searchkey().

5. getDir(): Sunción que recibe el directorio están los archivos de texto y ejecuta FetchFiles().

IV. Algoritmos principales del Programa

El algoritmo utilizado para la implementación del search se divide en dos partes principales. Primero se crea una estructura que guarde las palabras contenidas en todos los archivos y la cantidad de veces que aparece en cada uno. Este consiste en un mapa que tiene por índices las palabras que aparecen en los archivos. En cada uno de estos espacios identificados con cada palabra se almacena un mapa que tiene como índice el numero del archivo donde aparece esta palabra y como valor la cantidad de veces que aparece en este. Como segunda parte del algoritmo se encuentra la forma de búsqueda, que consiste en acumular la cantidad de veces que aparecen las palabras buscadas en un archivo y guardar estos resultados en una vector de resultados. Luego que se obtiene este vector de resultados se ordena descendentemente para luego desplegar los primeros tres resultados lo cuales tienen la mayor frecuencia de las palabras.

V. Prueba del Software y resultados

Prueba #1	Prueba #2	Prueba #3
Search: love 1: The-Anatomy-of-Love.txt 9 2: Until-Next-Time.txt 9 3: Salaam-E-Ishq.txt 8	Search: music guitar 1: The-Children-of-Hip-Hop.txt 12 2: Stuart-Mossman.txt 9 3: Reggae-Music.txt 8	Search: library school student 1: They-Shall-Have-Music.txt 10 2: Teen-Idol.txt 10 3: Service-Safari.txt 10

Luego de realizadas las pruebas a la versión final del programa, logramos obtener resultados totalmente satisfactorios