INFORME TP FINAL

Diccionario:

int cantDatosArch(): Esta función calcula el tamaño de cada texto, y lo usamos para el arreglo de términos dinámico, es para la dimensión del array.

void inicIdDoc(): Esta función lo único que hace es abrir el archivo de "idDoc", y lo que hacemos es escribir en el archivo el número cero, para luego incrementar el id según la cantidad de textos que se inserten en el diccionario.

void agregaCaracterAPalabra(char*, char): Esta función inserta letra por letra mientras sea un carácter válido en un arreglo de char, además inserta el terminator a la palabra.

void leerArchivo(termino*, int*, char*): Función principal, recibe por parámetro el array de términos, los validos y el nombre del archivo de texto, abre el archivo de idDoc y el de texto, después usamos la función de la librería "string.h", memset lo que hace es inicializar el arreglo de char con ceros porque si no lo inicializamos la primer palabra la mostraba con caracteres raros, luego incrementamos el idDoc por cada texto ingresado, después leemos caracter por caracter del archivo de texto, y preguntamos si la letra es válida, si es válida llamamos a la función agregaCaracterAPalabra (Explicada anteriormente), y si no es válida es porque se leyó un carácter raro o un espacio, después pasamos la palabra, el id y la posición al arreglo de términos, incrementamos válidos y reinicializamos el arreglo de char nuevamente con ceros, luego para la última palabra verificamos que haya una palabra en el arreglo, ya que si no verificamos no se insertaria en el array de términos, si el último char del archivo es un punto la palabra se insertaria, si es un caracter invalido la palabra queda guardada en el arreglo y nunca se inserta, después cerramos los archivos.
Esto se va a repetir hasta que se lea el último carácter de los archivos de texto.

void escrituraDiccionario(termino*, int): Esta función recibe el arreglo de términos y los válidos, abre el archivo en "ab" y mientras haya términos en el array, lo que va a hacer es escribir cada estructura de términos en el archivo diccionario, va a hacer esto por cada texto insertado.

void leer(): Esta función simplemente es para corroborar que se haya escrito todo bien en el archivo de diccionario, lo que hace es mostrar todo su contenido.

void mostrarArregloTerminos (termino*, int): Esta funcion es para corroborar que se haya insertado todo bien en el arreglo de terminos, lo que hace es mostrar cada estructura nada mas.

void insertarTextosBase(): Esta función inserta los textos base, llama a todas las anteriores funciones para su correcto funcionamiento y que quede modularizado, inicializa el archivo de idDoc (función explicada anteriormente), luego crea el arreglo de terminos dinamico con la dimension que retorna la funcion cantDatosArch (Explicada anteriormente), luego llama a la función leerArchivo, además llama a la función escrituraDiccionario, se hace un free del arreglo de términos y se repite otra vez todo con el segundo texto.

Motor de Búsqueda:

void generarArbolBusqueda(nodoA ABBDiccionario):** ésta es la función principal del motor, ya que recibe el archivo de registros y guarda cada dato en una variable de tipo término. Luego se procede a generar un nodo de tipo ocurrencia que irá dentro de cada nodo del árbol binario. Luego de eso, se invoca a la función **buscarEInsertar**,

void buscarEInsertar (nodoA arbol, char* palabra, nodoT* ocurrencia):** La función obtiene los datos necesarios para generar el árbol, y dentro de cada nodo, crear una nueva lista con las ocurrencias de la palabra. Si la palabra a ingresar en el árbol ya existe, se agrega únicamente la ocurrencia en la sublista con la función **InsertarOcurrencia**. En caso de no encontrar la palabra ya en el árbol, determina si, alfabéticamente, la palabra a ingresar debe ir a la izquierda o la derecha, para luego, de manera recursiva, moverse por los datos hasta encontrar una hoja. Finalmente, cuando la función sea llamada por última vez, y el árbol sea igual a NULL, crea un nuevo nodo con la palabra, e inserta su primera ocurrencia en la lista correspondiente.

void insertarOcurrencia(nodoT ocurrencias, nodoT* nuevo):** Comprueba el lugar a donde debería ser ingresada la nueva ocurrencia de la palabra. Si el ID del documento es igual al que ya se encuentra, se comprueba si la posición es menor a la primera posición ya encontrada. Si es menor, se ubica en el primer lugar; si es mayor, se llama nuevamente a la función para que la recorra de manera recursiva, y así encontrar el lugar adecuado. Si el ID del documento es menor, se ingresa al inicio de la lista, y si es mayor, nuevamente se llama a la función hasta encontrar el lugar correcto. En caso de encontrar la lista vacía, se guarda la ocurrencia en el primer nodo.

Funciones de Usuario:

void buscarPalabraYMostrarOcurrencias(nodoA*, char*)*:* esta función recibe el árbol de términos y una palabra determinada que se desee buscar en el mismo. Si la palabra se encuentra, procede a llamar a la función *void mostrarOcurrencias (nodoT*)* enviando la lista de ocurrencias del nodoA* en el que se encuentra el término, para mostrar cada una con su respectivo IdDoc y pos.

void buscarPalabraEnDosDocumentosYMostrarOcurrencias (nodoA*, char*, int, int): esta función recibe el árbol de términos, una palabra a buscar y dos IdDoc. Su objetivo es verificar que un término determinado se encuentre en dos IdDoc pasados por parámetro, a través de la función *int buscarIdDocEnOcurrencias (nodoT*, int)*, que retorna true o false. Si esta última función retorna true, se procede a mostrar cada ocurrencia de la palabra que sea de alguno de los IdDoc. Si la palabra no hubiera sido encontrada, todo lo planteado anteriormente no se ejecuta.

void buscarPalabrasEnMismoDoc (nodoA*, char*, char*, int): esta función recibe el árbol de términos, verifica que las dos palabras pasadas por parámetro se encuentran en el mismo (a través de la función *int buscarPalabraEnArbol (nodoA*, char*)*) y si es así, verifica que por lo menos una de sus respectivas ocurrencias está en un idDoc determinado. Si lo anterior también se cumple, se procede a mostrar todas las ocurrencias de ambas palabras que sean de dicho documento.

void buscarFrase (nodoA*, char*): recibe el árbol de términos y una frase como string, que es leída a través de la función *leerFrase(char*)*. Luego, se genera un arreglo de strings con cada palabra de la frase llamando a *retornarArregloDeStrings (char*, int)*, del cual los validos son asignados por *retornarCantPalabras (char*)*. Con dicho arreglo, se procede a obtener la lista de ocurrencias de la primer palabra (con *retornarListaOcurrenciasDeUnaPalabra (nodoA*, char*)*), y de esta forma, se recorre cada ocurrencia de la lista y se comprueba si las demás palabras de la frase pueden estar contiguas (todo esto a través de la variable 'idDoc' de la ocurrencia y otra llamada 'pos' que incrementa a medida que se recorre el arreglo de strings, además de la función *buscarPalabraConIdDocYPos (nodoA*, char*, int, int))*. Finalmente, se cortará el recorrido de la lista si se encontró una contigüidad entre las palabras de la frase y se mostrará el idDoc y pos de la primera palabra de la misma, o de otra forma se informará lo contrario.

void mostrarTerminoMasFrecuente (nodoA*): recibe el árbol de términos y (si no está vacío) busca el nodo del término más frecuente (a través de *buscarTerminoMasFrecuente*

(nodoA)*) para mostrar su lista de ocurrencias. Esta última función compara la frecuencias de la raíz, el subárbol izquierdo y el derecho, y retorna el nodo mayor.

void sugerirSimilares (nodoA*, char*): recorre el árbol de términos y muestra las palabras cuya distancia de Levenshtein con la palabra pasada por parámetro es menor o igual a tres (a través de la función *Levenshtein(char* , char*)*).

void agregarTexto (nodoA**): recibe un árbol de términos por referencia y lee un texto escrito por el usuario, mientras filtra sus palabras y las agrega a un arreglo de estructura *termino* (a través de la función *llenarArregloDeTerminosEIncrementarIdDoc (termino*, int*)*, que también incrementa el idDoc que está en archivo), el cual se agregara al final del archivo *diccionario* e insertará de forma ordenada en el árbol con *recorrerArregloTerminosYAgregarAArbol(nodoA**, termino*, int)*.

void menu (nodoA**): recibe un árbol de términos por referencia, muestra las opciones con la función *mostrarOpciones()* y lee la opción que pida el usuario, llamando a la que desee a través un *switch(opcion).* Si el usuario desea seguir en el menú después de utilizar una función, se llama de forma recursiva a sí misma.