Motor de Búsqueda de palabras

Un *motor de búsqueda* es un sistema que construye un índice a partir de texto y responde a consultas utilizando este índice. Consiste en utilizar palabras para indexar una colección de texto con el objetivo de acelerar la tarea de búsqueda.
Diccionario: el conjunto de todas las palabras que aparecen en el/los texto/s con su posición, contiene palabras repetidas.
Ocurrencias (postings): listas conteniendo la información necesaria para cada palabra del vocabulario (documentos donde aparece, posición)

Diccionario

Se tienen N documentos, donde se leerán todas las palabras de cada uno de ellos. Los documentos deben ser archivos de texto leídos como un binario (char)), sobre un vector que contendrá cada palabra, id de documento al que pertenece, y posición en dicho documento. Una vez finalizada la lectura de los documentos, se procede a generar el diccionario.
El diccionario es un archivo binario de registros de tipo término:

Estructura del diccionario (tabla de términos en un archivo):

```
typedef struct {
  char palabra[20];
  int idDOC;
  int pos; //incrementa palabra por palabra, y no letra por letra
}termino;
```

Motor de búsqueda

El Motor se realiza utilizando una estructura de datos adecuada,donde para cada término se debe tener la lista de posiciones donde aparece en el/los documentos.
Se tendrá una estructura de datos de tipo Árbol binario de búsqueda, construido mediante un orden lexicográfico por cada palabra. Dentro de cada nodo del árbol estará la palabra clave y una lista de las ocurrencias de dicha palabra en los documentos.

Estructura del motor: *Árbol binario de búsqueda en orden lexicográfico, compuesto de listas:*

```
typedef struct nodoA
{
   char palabra[20];
   int frecuencia; //representa la cantidad de nodos de la lista
   nodoT* ocurrencias; //ordenada por idDOC, luego por pos
   struct nodoA* der;
   struct nodoA* izq;
}nodoA;
```

Lista de ocurrencias: Ordenada por idDOC, y por posición por cada idDOC.

```
typedef struct nodoT
{
    int idDOC;
    int pos;
    struct nodoT* sig;
}nodoT;
```

Una vez construido el árbol con información, se procede a analizar los tipos de búsqueda de términos que se pueden realizar, veamos algunas definiciones de consultas:

Búsqueda única: La búsqueda más simple es utilizando una única palabra.

Frases Textuales: títulos de obras, procesos o hechos históricos, etc. (p.e. “Don segundo Sombra”,”El modelo TCP”, “Platero y yo”), nombres compuestos (p.e. “San José de Costa Rica””) No basta si los términos están en un documento. Hay que saber además sus posiciones relativas en mismo

Consulta de Términos Parecidos: Espacio Métrico (Léxico, Distancia de Edición)

- ***Distancia de Edición de Levenshtein***: Número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra, se usa ampliamente
    - *ej.1*. d(casado,cazado)= 1 C a s a d o C a z a d o
    - *ej.2*. d(Casa, Calle)= 3 C a s a C a l l e

  Es una generalización de la distancia de Hamming (Aplicada por bits) para códigos autocorrectores.
  Explicación algoritmo : > Aquí
  Codigo: **>** Aqui

Objetivos

El programa a realizar debe ser capaz de leer documentos y abstraer las palabras para volcarlas en el diccionario, incluir nuevos términos añadidos al diccionario en el motor de búsqueda y realizar las operaciones de usuario siguientes:

OPERACIONES DEL USUARIO

1) Buscar todas las apariciones de un término en algún documento (operación or).
2) Buscar todas las apariciones de un término en un documento y otro (operacion and).
3) Buscar la aparición de más de un término en el mismo documento.
4) Buscar una frase completa (las palabras deben estar contiguas en alguno de los documentos).
5) Buscar la palabra de más frecuencia que aparece en alguno de los docs.
6) Utilizar la distancia de levenshtein en el ingreso de una palabra y sugerir similares a partir de una distancia <= 3.

Reglas del TP

1. El código debe estar correctamente modularizado por cada estructura de datos.
2. Las variables como las funciones deben poseer nombres coherentes a su objetivo.
3. Los tipos de datos definidos por el usuario deben respetar la estructura mencionada.
4. Al momento de la entrega, el programa no puede poseer errores que impidan su ejecución. (esto se puede reveer)
5. Se debe elaborar un informe que explique el funcionamiento de los módulos y las funciones principales. (conciso, como un glosario)
6. La exposición es grupal y todos los integrantes del grupo deben participar, la nota es un promedio entre la nota individual del TP y la grupal.

Condiciones	Puntos	Obtenido
• Correcta modularización • Utilización de librerías • Nombres significativos en funciones y variables utilizadas • Compilación del proyecto sin errores	15	
• Desarrollo de las funciones necesarias para la creación del diccionario, incluye: ○ Leer los	20	

archivos de textos y pasarlos al arreglo de términos completando los campos de la estructura. ○ Crear un archivo binario donde se guarde el diccionario completo. ○ Funciones auxiliares para la muestra del diccionario.		
● Creación de la estructura compuesta solicitada y la carga de la misma con los términos del diccionario de manera correcta	15	
● Correcta resolución de las operaciones de usuario, puntos 1, 2 y 3	25	
● Correcta resolución de las operaciones de usuario, puntos 4, 5 y 6	25	

Tabla de puntuación:

Puntaje	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Nota	1	2	3	4	5	6	7	8	9	10

UTN FRMDP - Laboratorio de Computación 2 - TP Final
Comisiones 5 y 6

Condición	Desaprobado	Aprobado