Algoritmos y Estructuras de Datos II
Recuperatorio del segundo parcial – 17 de julio de 2015

Aclaraciones

- El parcial es **a libro abierto**.
- Entregar cada ejercicio **en hojas separadas**.
- Incluir en cada hoja el número de orden asignado, número de hoja, apellido y nombre.
- Al entregar el parcial, completar el resto de las columnas en la planilla.
- Cada ejercicio se calificará con **MB**, **B**, **R** o **M**, y podrá recuperarse independientemente de los demás. La cursada podrá aprobarse con hasta 1 (un) ejercicio **R** (regular) en cada parcial (post-recuperatorios), siempre y cuando ninguno de ellos sea un ejercicio 1. Para más detalles, ver "Información sobre la cursada" en el sitio Web.

Ej. 1. Diseño

Una importante empresa de software intenta insertarse en el mercado de los buscadores web. Para eso desean diseñar un motor de búsqueda, que inicialmente será muy rudimentario. En esta primera versión, el motor de búsqueda sólo podrá indexar documentos y realizar búsquedas por dos palabras clave.

Un documento no es más que una secuencia de palabras, y las palabras son cadenas de caracteres del alfabeto castellano (más los símbolos de puntuación usuales). En todo momento pueden indexarse nuevos documentos, que nunca se desindexan. Cada documento tiene, además, un nombre único que lo identifica que, eventualmente, podrá cambiarse. Los nombres de los documentos tienen una longitud no acotada. Lo mismo sucede con los documentos, que pueden tener cualquier cantidad de palabras, y estas palabras pueden ser de cualquier longitud.

Como en todo motor de búsqueda, el objetivo es encontrar documentos a partir de palabras clave. En esta versión inicial sólo se podrán hacer búsquedas del tipo $(palabra_1 \wedge palabra_2)$ que devolverán el conjunto de documentos que contienen **ambas** palabras (tanto $palabra_1$ como $palabra_2$).

Por ejemplo, un documento que consiste en la secuencia de palabras "¡Qué genio tiene el ingenuo de Eugenio!" será parte del conjunto de respuestas para las búsquedas (genio $\wedge$ ingenuo) y (el $\wedge$ Eugenio!). Por el contrario, este documento no estará el el conjunto de respuestas si la búsqueda es (Eugenio! $\wedge$ ornitorrinco).

TAD DOCUMENTO **ES** SECU(PALABRA)
TAD NOMBRE **ES** STRING
TAD PALABRA **ES** STRING

TAD BUSCADOR

generadores

iniciar	:	$\longrightarrow$ buscador		
indexarDocumento	:	buscador b $\times$ nombre n $\times$ secu(palabra) d	$\longrightarrow$ buscador	$\{n \notin \text{documentos}(b)\}$
cambiarNombre	:	buscador b $\times$ nombre n_1 $\times$ nombre n_2	$\longrightarrow$ buscador	$\{n_1 \in \text{documentos}(b) \wedge n_2 \notin \text{documentos}(b)\}$

observadores básicos

documentos	:	buscador	$\longrightarrow$ conj(nombre)	
texto	:	buscador b $\times$ nombre n	$\longrightarrow$ secu(palabra)	$\{n \in \text{documentos}(b)\}$

otras operaciones

buscar	:	buscador b $\times$ palabra p_1 $\times$ palabra p_2	$\longrightarrow$ conj(nombre)	

axiomas ...

Fin TAD

Diseñe el TAD BUSCADOR respetando los siguientes requerimientos de complejidad temporal en **peor caso**:

- INDEXARDOCUMENTO(b, n, d) $O\big(\log(n) + \log(d) \times \max_p(d) \times \text{docs}(b) \times \max_n(b)\big)$
- BUSCAR(b, p_1, p_2) $O\big(\log(p_1) + \log(p_2) + \max_n(b) \times \text{docs}(b)\big)$
- TEXTO(b, n) $O\big(\log(n)\big)$

donde $\max_p(d)$ es la longitud de la palabra más larga en d; $\max_n(b)$ es la longitud del nombre del documento más largo en b; y docs(b) es la cantidad de documentos indexados por b hasta el momento.

a) Muestre la estructura de representación propuesta. Explique para qué sirve cada parte de la estructura, o utilice nombres autoexplicativos.

b) Escriba el pseudocódigo del algoritmo BUSCAR(**in** b : estr, **in** p_1 : palabra, **in** p_2 : palabra).

Justifique claramente cómo y por qué los algoritmos, la estructura y los tipos soporte permiten satisfacer los requerimientos pedidos. No es necesario diseñar los módulos soporte, **pero sí describirlos, justificando por qué pueden (y cómo logran)** exportar los órdenes de complejidad que su diseño supone.

Ej. 2. Ordenamiento

a) FIGFA (Federación Intergaláctica de Fútbol Asociado) maneja la información de miles de millones de equipos que participan en los distintos torneos de fútbol de todo el universo. Para cada equipo se calcula un puntaje que sirve para elaborar el ránking universal de equipos, que puede consultarse en su página web. FIGFA es muy desorganizada y no tiene optimizada esta información, es decir que arma el ránking desde cero cada vez que alguien lo consulta. Por otra parte, los registros históricos de la página web muestran que, en prácticamente todos los casos, a los visitantes sólo les interesa conocer los primeros 100 equipos del ránking. Sugiera a FIGFA dos algoritmos de *sorting* que considere apropiados para la generación del ránking, y uno que considere inapropiado. Justifique por qué en cada caso.

b1) La Real Academia Española mantiene un compendio completo del vocabulario español. Este compendio es simplemente la lista ordenada de todas las palabras del lenguaje. Al fin de cada año se enriquece el lenguaje con varias palabras nuevas. Esto se hace agregando al compendio del año anterior un conjunto desordenado de nuevas palabras. Obviamente, para obtener el nuevo compendio anual es necesario reordenarlo. Se estima que cada año se agrega una cantidad de palabras equivalente al 5 % del tamaño del compendio del año anterior. Sugiera qué estrategias de ordenamiento son más apropiadas en esta situación. Justifique.

b2) El hecho de que se agregue un 5 % de palabras nuevas, ¿es relevante? ¿Cambiaría su respuesta (y cómo) si cada año se agregase un 50 % de palabras nuevas? ¿Y si fuese un 100 %, es decir, se duplicara el vocabulario cada año?

Ej. 3. Dividir y conquistar

Dado un árbol binario de números enteros, se desea calcular la máxima suma de los nodos pertenecientes a un camino entre dos nodos cualesquiera del árbol. Un camino entre dos nodos n_1 y n_2 está formado por todos los nodos que hay que atravesar en el árbol para llegar desde n_1 hasta n_2, incluyéndolos a ambos. Un camino entre un nodo y sí mismo está formado únicamente por ese nodo. El algoritmo debe recorrer **como máximo una vez** cada nodo del árbol, que no necesariamente es completo. Se considerará incorrecto a un algoritmo que no cumpla con esta condición.

Figura 1: Ejemplo de un camino de máxima suma en un posible ab(int). Resultado correcto: 50.

Se pide dar un algoritmo MÁXIMASUMACAMINO(a : ab(int)) → int que resuelva el problema utilizando la técnica de *Dividir y Conquistar*, calculando y justificando claramente su complejidad. El algoritmo debe tener una complejidad temporal de peor caso igual o mejor que $O(n)$ siendo n la cantidad de nodos del árbol.