Algoritmos y Estructuras de Datos II
Segundo parcial – 22/11/2008

Aclaraciones

- El parcial **NO** es a libro abierto.
- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Incluya el número de orden asignado (léalo cuando circule la planilla), apellido y nombre en cada hoja.
- Al entregar el parcial complete los datos faltantes en la planilla.
- Cada ejercicio se calificará con B, R ó M. Una B no significa que el ejercicio está "perfecto", sino que cumple con los requisitos necesarios para aprobar. En los parciales promocionados se asignará una nota numérica más precisa a cada ejercicio.
- Para aprobar el parcial debe obtenerse B en el ejercicio 1 y en alguno de los ejercicios 2 y 3. Un parcial se considera promocionado si está aprobado y su puntaje es 70 o superior.

Ej. 1. Diseño

Considere el siguiente TAD que describe la estructura de un texto:

TAD TEXTO

géneros txt

observadores básicos

cantPalabras : txt $\longrightarrow$ nat

enPos : txt t $\times$ nat n $\longrightarrow$ palabra — $1 \leq$ n $\leq$ cantPalabras(t)

generadores

enBlanco : $\longrightarrow$ txt

agPalabra : txt $\times$ palabra $\longrightarrow$ txt

otras operaciones

posiciones : txt $\times$ palabra $\longrightarrow$ conj(nat)

cambiarPalabra : txt $\times$ palabra $\times$ palabra $\longrightarrow$ txt

subtexto : txt t $\times$ nat *inicio* $\times$ nat fin $\longrightarrow$ txt — 1$\leq$ inicio$\leq$ fin $\leq$ cantPalabras(t)

masRepetida : txt $\longrightarrow$ conj(palabra) — cantPalabras(t) > 0

axiomas

cantPalabras(enBlanco) $\equiv$ 0

cantPalabras(agPalabra(t, p)) $\equiv$ $1 +$ cantPalabras(t)

enPos(agPalabra$(t, p), n$) $\equiv$ **if** cantPalabras$(t) + 1 = n$ **then** p **else** enPos(t, n) **fi**

posiciones(enBlanco, p') $\equiv$ $\emptyset$

posiciones(agPalabra$(t, p), p'$) $\equiv$ **if** $p = p'$ **then** Ag(cantPalabras$(t) + 1$, posiciones(t, p'))) **else** posiciones(t, p') **fi**

cambiarPalabra(enBlanco, p, p') $\equiv$ enBlanco

cambiarPalabra(agPalabra$(t, p''), p, p'$) $\equiv$ agPalabra(cambiarPalabra(t, p, p'), **if** $p = p''$ **then** p' **else** p'') **fi**)

subtexto(t, i, f) $\equiv$ **if** $i = f$ **then** agPalabra(enPos(t, i), enBlanco) **else** agPalabra(enPos(t, f), *subtexto*$(t, i, f - 1)$) **fi**

masRepetida(t) $\equiv$ $c \iff (\forall p : palabra)(p \in c \iff \neg\exists(q : palabra)(q \in c \wedge \#\text{posiciones}(t, p) < \#\text{posiciones}(t, q)))$

Fin TAD

Se desea diseñar el módulo correspondiente al TAD texto. En particular, asumimos que trabajaremos sólo con textos en español, y por lo tanto podemos dar una cota para la longitud de la palabra más larga que puede aparecer en el texto.

Se requiere que las operaciones que se listan a continuación cumplan con la complejidad temporal indicada:

- *subtexto(in inicio:nat, in fin:nat, in t:txt)* $\rightarrow$ *txt*

  Devuelve el texto correspondiente al fragmento de t que comienza en la posición *inicio* y finaliza en la posición fin.

$O(fin - inicio)$ en el peor caso

- *cambiarPalabra(in anterior:palabra, in nueva:palabra, inout t:text)*

  Cambia todas las ocurrencias en el texto de la palabra anterior por la nueva.

  $O(k)$ en el peor caso, donde k es la cantidad de veces que se repite la palabra a cambiar.
- *palabrasMasRepetidas (in t:txt) → conj(palabras)* Devuelve el conjunto de palabras que más se repiten en el texto.

  $O(1)$ en el peor caso. Puede generar aliasing.

a) Describir la estructura a utilizar, documentando claramente cómo la misma resuelve el problema y cómo cumple con los requerimientos de eficiencia. El diseño debe incluir sólo la estructura de nivel superior. Para justificar los órdenes de complejidad, describa las estructuras soporte. **Importante**: si alguna de las estructuras utilizadas requiere que sus elementos posean una función especial (por ejemplo, comparación o función hash) deberá describirla.
b) Escribir una versión en lenguaje imperativo del algoritmo *cambiarPalabra*. Justifique la complejidad sobre el código.

Ej. 2. Eliminación de la recursión

Considere la siguiente función que calcula el binomio de Newton $\binom{n}{k}$.
$binomio : nat\ n \times nat\ k \ \rightarrow \ nat \qquad (k \leq n)$
$binomio(n, k) \equiv$ **if** $(n = k \vee k = 0)$ **then** 1 **else** $binomio(n-1, k-1) + binomio(n-1, k)$ **fi**

a) Indicar qué tipo de recursión tiene la función binomio.
b) Aplicar la técnica de inmersión+plegado/desplegado para obtener una función que pueda ser transformada algorítmicamente a una forma iterativa. Indique claramente el tipo de inmersión que utiliza, la signatura y la axiomatización completa de todas las funciones que introduzca.
c) A partir del resultado anterior genere un algoritmo imperativo con la versión iterativa de la función binomio.

Ej. 3. Sorting

Considere la siguiente estructura que contiene las notas de los alumnos de un curso:

planilla es **array** $[1..n]$ **de** *tupla* $\langle$ nombre: String $\times$ sexo: FM $\times$ puntaje: Nota $\rangle$

donde FM es $enum\{masc, fem\}$ y Nota es 0..10.

Se necesita ordenar la planilla de forma tal que todas las mujeres aparezcan al inicio de la tabla según un orden creciente de notas y todos los varones aparezcan al final de la tabla también ordenados de manera creciente respecto de su puntaje, como muestra en el siguiente ejemplo:

Entrada

Ana	*F*	10
Juan	*M*	6
Rita	*F*	6
Paula	*F*	7
Jose	*M*	7
Pedro	*M*	8

Salida

Rita	*F*	6
Paula	*F*	7
Ana	*F*	10
Juan	*M*	6
Jose	*M*	7
Pedro	*M*	8

a) Proponer un algoritmo de ordenamiento ordenaPlanilla(in out p: planilla) para resolver el problema descripto anteriormente y cuya complejidad temporal sea $O(n)$ en el peor caso, donde n es la cantidad de elementos del arreglo. Justificar el orden de complejidad del algoritmo propuesto. Puede utilizar (sin reescribir) cualquiera de los algoritmos vistos en clase.

b) Supóngase ahora que la planilla original ya se encuentra ordenada alfabéticamente por nombre. Puede asegurar que si existen dos o más mujeres con igual nota, entonces las mismas aparecerán en orden alfabético en la planilla ordenada? Justifique su respuesta.
c) ¿La cota $O(n)$ contradice el "lower bound" sobre la complejidad temporal en el peor caso de los algoritmos de ordenamiento? Explique su respuesta.