Algoritmos y Estructuras de Datos II
Segundo parcial – 1/12/2007

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 ambos ejercicios. Un parcial se considera promocionado si está aprobado y su puntaje es 70 o superior.

Ej. 1. Diseño (50 puntos)

El hospital del Sindicato de Ascensoristas decidió imponer nuevas reglas de funcionamiento para ordenar su caótica organización. La nueva organización se basa en dos pilares: los médicos de cabecera y un esquema de prioridades para autorizar tratamientos.

El esquema funciona de la siguiente manera: Cuando un afiliado tiene alguna necesidad, debe recurrir a su médico de cabecera. Este, luego de una revisación general, le receta una lista de tratamientos, que el paciente debe realizar en estricto orden.

A su vez, cada cierto intervalo de tiempo, la oficina de administración escoge un afiliado con tratamientos pendientes y autoriza la realización del primer tratamiento de la lista. Esta elección se hace eligiendo siempre aquel paciente cuyo primer tratamiento tenga la mayor prioridad, de acuerdo con criterios sanitarios. Si existe más de un afiliado en esa situación, la oficina puede elegir cualquiera de ellos. Nótese que un mismo tratamiento siempre tiene la misma prioridad.

Además, un afiliado no puede volver a su médico de cabecera hasta no cumplir con la prescripción en forma completa.

Este mecanismo fue modelado con el siguiente TAD:

TAD HOSPITAL

géneros hospital

generadores

crear : conj(afiliado) × dicc(tratamiento × prioridad) $\longrightarrow$ hospital

recetar : hospital h × afiliado a × cola(tratamiento) c $\longrightarrow$ hospital
$(a \in \text{afiliados}(h) \wedge_{\text{L}} \text{vacía?}(\text{pendientes}(h, a))) \wedge \text{todosValidos}(c, \text{tratamientos}(h))$

autorizar hospital h $\longrightarrow$ hospital
$(\exists a : \text{afiliado})(a \in \text{afiliados}(h) \wedge_{\text{L}} \neg\text{vacía?}(\text{pendientes}(h, a)))$

observadores básicos

afiliados	: hospital	$\longrightarrow$	conj(afiliado)	
tratamientos	: hospital	$\longrightarrow$	conj(tratamiento)	
prioridad	: hospital h × tratamiento t	$\longrightarrow$	prioridad	$t \in \text{tratamientos}(h)$
pendientes	: hospital h × afiliado a	$\longrightarrow$	cola(tratamiento)	$a \in \text{afiliados}(h)$

otras operaciones

proximoAfiliado : hospital h $\longrightarrow$ afiliado $(\exists a : \text{afiliado})(a \in \text{afiliados}(h) \wedge_{\text{L}} \neg\text{vacía?}(\text{pendientes}(h, a)))$

axiomas .

...

pendientes(crear(c,d),a) $\equiv$ vacía

pendientes(recetar(h, a, c), a') $\equiv$ **if** $a \neq a'$ **then** pendientes(h, a') **else** c **fi**

pendientes(autorizar(h),a) $\equiv$ **if** $a =$ proximoAfiliado(h) **then** desencolar(pendientes(h, a)) **else** pendientes(h, a) **fi**

proximoAfiliado(h) = p $\equiv$ $p \in$ afiliados(h) $\wedge_{\text{L}}$ ($\neg$vacía?(pendientes(h, p)) $\wedge$ ($\forall a$: afiliado)
($a \in$ afiliados(h)) $\Rightarrow_{\text{L}}$ vacía?(pendientes(h, a)) $\vee_{\text{L}}$
prioridad(próximo(pendientes(h, a))) $\leq$ prioridad(próximo(pendientes(h, p)))))

Fin TAD

TAD Afiliado **ES** String
TAD Tratamiento **ES** String
TAD Prioridad **ES** Nat

Se quiere ahora diseñar el sistema propuesto, teniendo en cuenta que las operación **recetar** y **autorizar** deben realizarse en $O(\log a)$, donde a es la cantidad de afiliados. Se puede suponer que tanto los nombres de los afiliados como los de los tratamientos tienen una longitud acotada (no más de 20 letras). También puede suponer que todas las estructuras de soporte cuentan con iteradores para recorrer sus elementos en tiempo lineal.

Se pide:

i) Describir la estructura a utilizar, documentando claramente cómo la estructura resuelve el problema y cómo cumple con los requerimientos de eficiencia. El diseño debe incluir sólo la estructura de nivel superior. De ser necesario para justificar los órdenes de complejidad, describa las estructuras soporte. Si alguna de las estructuras utilizadas requiere que sus elementos posean alguna función especial (por ejemplo, comparación o función de hash) deberá escribirla.
ii) Escribir una versión en lenguaje imperativo del algoritmo **autorizar**, indicando la complejidad de cada uno de los pasos.

Ej. 2. Ordenamiento (25 puntos)

Se tiene un arreglo de secuencias no vacías de números naturales, cada una de los cuales contiene a todos los números dentro de un intervalo determinado. Se quiere generar un arreglo de naturales que contenga a todos los elementos de todas las secuencias, sin repetidos, ordenados de menor a mayor (ver figura).

Proponga un algoritmo que resuelva el problema en $O(k \log k + n)$, donde k es la cantidad de secuencias y n la cantidad total de elementos (es decir, la suma de las longitudes de todas las secuencias). Indique y justifique claramente la complejidad temporal de cada paso. Puede utilizar (sin reescribir) cualquiera de los algoritmos vistos en clase y las estructuras auxiliares que considere necesarias.

entrada	[[8 5 7 6] [11 10 12] [4 3 5 6]]
salida	[3 4 5 6 7 8 10 11 12]

Figura 1: Entrada y resultado esperado para el algoritmo de ordenamiento.

Ej. 3. Eliminación de la recursión (25 puntos)

Dada una secuencia de secuencias de naturales, la función **achatar** produce una secuencia de naturales formada por la concatenación de todas ellas (ver ejemplo).

achatar : secu(secu(nat)) $\longrightarrow$ secu(nat)
achatar(s) $\equiv$ **if** vacía?(s) **then** <> **else** prim(s) & achatar(fin(s)) **fi**

s = [[1 2 3],[5 1 4], [], [1]]
achatar(s) = [1 2 3 5 1 4 1]

Figura 2: Resultado de la función achatar

Se pide:

a) Indique qué tipo de recursión tiene la función achatar.
b) Aplicar la técnica de inmersión+plegado/desplegado para obtener una función que pueda ser transformada algorítmicamente a una forma iterativa. Indique claramente 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 achatar.