Algoritmos y Estructuras de Datos II
Recuperatorio del segundo parcial – 12/07/2008

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

Se desea diseñar un sistema para administrar las notas de los alumnos de un curso. El diseño deberá respetar la siguiente especificación.

TAD SISTNOTAS

géneros sn

observadores básicos

alumnos : sn $\longrightarrow$ conj(alu)
evaluaciones : sn $\longrightarrow$ conj(instEval)
notas : sn s $\times$ alu a $\longrightarrow$ dicc(instEval,nota) $\quad a \in \text{alumnos}(s)$

generadores

nuevo : $\longrightarrow$ sn
agAlumno : sn s $\times$ alu a $\longrightarrow$ sn $\quad a \notin \text{alumnos}(s))$
pasarPlanilla : sn s $\times$ instEval e $\times$ dicc(alu $\times$ nota) p $\longrightarrow$ sn
$\quad e \notin \text{evaluaciones}(s) \wedge (\forall a : \text{alu})(a \in \text{claves}(p) \Rightarrow (a \in \text{alumnos}(s)))$

otras operaciones

cantNotas : sn s $\times$ alu a $\longrightarrow$ nat $\quad a \in \text{alumnos}(s)$
alumnosConNota : sn s $\times$ instEval e $\times$ nota $\longrightarrow$ conj(alu) $\quad e \in \text{evaluaciones}(s)$

axiomas

$\text{alumnos}(\text{nuevo}) \equiv \emptyset$
$\text{alumnos}(\text{agAlumnos}(s, a)) \equiv \text{Ag}(a, \text{alumnos}(s))$
$\text{alumnos}(\text{pasarPlanilla}(s, e, p)) \equiv \text{alumnos}(s)$
$\text{evaluaciones}(\text{nuevo}) \equiv \emptyset$
$\text{evaluaciones}(\text{agAlumnos}(s, a)) \equiv \text{evaluaciones}(s)$
$\text{evaluaciones}(\text{pasarPlanilla}(s, e, p)) \equiv \text{Ag}(e, \text{evaluaciones}(s))$
$\text{notas}(\text{agAlumnos}(s, a), a') \equiv$ **if** $a = a'$ **then** $\emptyset$ **else** $\text{notas}(s, a')$
$\text{notas}(\text{pasarPlanilla}(s, e, p), a') \equiv$ **if** $def?(a', p)$ **then** $\text{definir}(e, \text{obtener}(a', p), \text{notas}(s, a'))$
else $\text{notas}(s, a')$ **fi**
$\text{cantNotas}(s) \equiv \#\text{claves}(\text{notas}(s))$
$\text{alumnosConNota}(\text{pasarPlanilla}(s, e, p), e', v) \equiv$ **if** $e = e'$ **then** $\text{selec}(\text{claves}(p), p, v)$
else $\text{alumnosConNota}(s, e', n)$ **fi**

Fin TAD

donde
selec: conj(clave) c$\times$ dicc(clave,sign) d$\times$ sign $\rightarrow$ conj(clave) $\quad c \subseteq \text{claves}(d)$
$\text{selec}(c, d, v) \equiv$ **if** $\emptyset?(c)$ **then** $\emptyset$ **else**
if $\text{obtener}(\text{dameUno}(c), d) = v$ **then** $\text{Ag}(\text{dameUno}(c), \text{selec}(\text{sinUno}(c), d, v))$
else $\text{selec}(\text{sinUno}(c), d, v)$ **fi**
fi

Notar que el sistema deberá mantener las notas obtenidas por los alumnos en las distintas instancias de evaluación. Los alumnos se identifican por el número de libreta universitaria. El mismo está conformado por

cinco dígitos, cuyos dos últimos describen el año de ingreso (por ejemplo, 999/99). Las instancias de evaluación son identificadas por su nombre, que es string de hasta 30 caracteres de longitud (por ejemplo, "*primerparcial*", "*trabajodelaboratorio*", etc.). Las notas serán un valor entero comprendido entre 0 y 10.

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

- *pasarPlanilla(inout s:Sistema, in i:InstEval, in p:dicc(alu,Nota))*

  Agrega una nueva instancia de evaluación i y define la nota obtenida en esa instancia de evaluación para cada alumno contenido en la planilla p. El parámetro p no puede ser modificado por esta operación.

  $O(m)$ en caso promedio y $O(n * m)$ en el peor caso, donde n es la cantidad de alumnos en el sistema y m es la cantidad de alumnos en la planilla p.
- *cantNotas(in s:Sistema, in a:alu)* $\rightarrow$ nat

  Devuelve la cantidad de instancias de evaluación para las cuales el alumno a tiene nota definida.

  $O(1)$ en caso promedio y $O(n)$ en el peor caso, donde n es la cantidad de alumnos en el sistema.
- *alumConNota(in s:Sistema, in e:instEval, in v:Nota)* $\rightarrow$ conj(alu)

  Devuelve el conjunto de alumnos que obtuvieron la nota v en la instancia de evaluación e.

  $O(1)$ 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 *pasarPlanilla* utilizando iteradores. Justifique la complejidad sobre el código.

Ej. 2. Sorting

Sea A un arreglo de n elementos, A es un *pico* si y sólo si sus elementos están ordenados de la siguiente manera: A puede ser partido en dos secuencias consecutivas, la primera de las cuales es creciente y la segunda decreciente. Por ejemplo, el siguiente arreglo es un pico.

1	2	2	4	6	6	6	9	8	8	5	4	4	2

a) Proponga un algoritmo para ordenar en modo **decreciente** los elementos de un array con estructura de pico. El algoritmo propuesto debe tener complejidad temporal $O(n)$ en el peor caso. Justifique la complejidad del algoritmo propuesto.
b) Diremos que un array es *casi-pico* si el mismo puede obtenerse a partir de un array pico sólo intercambiando algunos pares de elementos adyacentes pero sin mover de lugar el valor máximo. Por ejemplo, el siguiente arreglo es casi-pico ya que puede obtenerse a partir del array pico mostrado en el punto a) intercambiando las posiciones 2 y 3, 9 y 10, y 12 y 13.

1	2	**4**	**2**	6	6	6	9	8	**5**	**8**	4	**2**	**4**

Desarrolle un algoritmo de complejidad temporal $O(n)$ en el peor caso para ordenar arrays casi-pico. Justifique la complejidad del algoritmo propuesto.

Ej. 3. Generalización de funciones

Considere la siguiente función que dada una secuencia produce otra capicúa:

$capicualizar : secu(\alpha) \rightarrow secu(\alpha)$

$capicualizar(s) \equiv$ **if** $vacia?(s)$ **then** $<>$ **else** $prim(s) \bullet capicualizar(fin(s)) \circ prim(s)$

a) Indicar qué tipo de recursión tiene la función capicualizar.
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) Dar un algoritmo imperativo a partir de la versión iterativa de la función capicualizar obtenida en el paso anterior.