Algoritmos y Estructuras de Datos II
Segundo recuperatorio – 13/12/2010

Aclaraciones

- El parcial es a **libro abierto**.
- Numerar las hojas entregadas. Completar en la primera hoja la cantidad total de hojas entregadas.
- Incluir el número de orden asignado, apellido y nombre en cada hoja.
- Al entregar el parcial completar los datos faltantes en la planilla.
- Cada ejercicio se calificará con P, A, R o M.
- Para aprobar el parcial se deberá obtener al menos una A en el primer ejercicio y en los ejercicios 2 y 3 se deberá obtener al menos una A y una R. Para promocionar, todos los ejercicios deberán ser calificados con P (P no significa perfecto)

Ej. 1. Diseño

Se desea diseñar un sistema para mantener una base de datos de artículos científicos. Cada artículo es identificado unívocamente mediante un string **no acotado**. El sistema almacena también las citas de cada artículo ingresado y permite consultar, dado un artículo, el conjunto de artículos citados por él. Todas las citas son siempre a artículos almacenados dentro del mismo sistema. El sistema también permite saber si dos artículos se citan mutuamente y cuántas son sus citas en común. Observar que el sistema especificado no almacena al artículo en sí, sino sólo su identificador y sus referencias.

El siguiente TAD es una especificación para este problema.

TAD Biblioteca

géneros biblioteca

exporta biblioteca, generadores, observadores, otras operaciones

igualdad observacional
$(\forall b, b' : \text{biblioteca})\ (b =_{\text{obs}} b' \iff (\cdots))$

observadores básicos

articulos : biblioteca $\longrightarrow$ conj(articulo)

citas : biblioteca b × articulo a $\longrightarrow$ conj(articulo) $\{a \in \text{articulos}(b)\}$

generadores

crear : $\longrightarrow$ biblioteca

agArticulo : biblioteca b × articulo a $\longrightarrow$ biblioteca $\{a \notin \text{articulos}(b)\}$

agCita : biblioteca b × articulo a_1 × articulo a_2 $\longrightarrow$ biblioteca $\{\{a_1, a_2\} \subseteq \text{articulos}(b) \wedge_{\text{L}} a_2 \notin \text{citas}(b, a_1)\}$

otras operaciones

seCitanMutuamente : biblioteca b × articulo a_1 × articulo a_2 $\longrightarrow$ bool $\{\{a_1, a_2\} \subseteq \text{articulos}(b)\}$

#citasEnComun : biblioteca b × articulo a_1 × articulo a_2 $\longrightarrow$ nat $\{\{a_1, a_2\} \subseteq \text{articulos}(b)\}$

axiomas

articulos(crear()) $\equiv$ $\emptyset$

articulos(agArticulo(b, a)) $\equiv$ Ag(a, articulos(b))

articulos(agCita(b, a1, a2)) $\equiv$ articulos(b)

citas(agArticulo(b, a1), a2) $\equiv$ **if** a1 = a2 **then** $\emptyset$ **else** citas(b, a2) **fi**

citas(agCita(b, a1, a2), a3) $\equiv$ **if** a3 = a1 **then** Ag(a2, citas(b, a3)) **else** citas(b, a3) **fi**

seCitanMutuamente(b, a1, a2) $\equiv$ a2 $\in$ citas(b, a1) $\wedge$ a1 $\in$ citas(b, a2)

#citasEnComun(b, a1, a2) $\equiv$ #(citas(b, a1) $\cap$ citas(b, a2))

Fin TAD

ARTICULO es STRING

Se desea diseñar el sistema propuesto teniendo en cuenta que la operación *seCitanMutuamente(b, a1, a2)* debe realizarse en $O(tam(a1) + tam(a2))$, donde $tam(a)$ devuelve la cantidad de caracteres del identificador del artículo a. Por otro lado, la operación *#citasEnComun(b, a1, a2)* debe tener una complejidad de $O(tam(a1) + tam(a2) + \#citas(b, a1) + \#citas(b, a2))$. Por último, la operación *AgCita(b, a1, a2)* debe tener una complejidad de $O(tam(a1) + tam(a2) + \#citas(b, a1))$.

Considerar que la comparación de dos números naturales tiene complejidad $O(1)$. Este **no es el caso** de la comparación entre dos strings de longitud arbitraria, cuya operación tiene una complejidad de $O(n)$, donde n es el tamaño del string más corto de los dos. Se pide:

a) 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.

b) Escribir un pseudocódigo *à la* C++ del algoritmo para *agCita* y *#citasEnComun*, indicando la complejidad de cada uno de los pasos.

Ej. 2. Sorting

Sea A un arreglo $[a_1, \ldots, a_n]$ de naturales y sea *Ord(A)* el arreglo A ordenado ascendentemente. Dada una constante k, decimos que A está *k-casi ordenado* si para cada posición i de A, la posición del elemento $Ord(A)[i]$ no está a más de k posiciones de distancia de i. Formalmente, si el tamaño de A es n:

$$\text{KCasiOrdenadoAsc(A)} \equiv (\forall i)(1 \leq i \leq n \rightarrow (\exists j)(|j - i| \leq k \land 1 \leq j \leq n \land_{\text{L}} \text{Ord}(A)[i] = A[j]))$$

Dado un arreglo A que cumple KCasiOrdenadoAsc(A) se pide escribir un algoritmo que devuelva el arreglo A ordenado ascendentemente. El algoritmo propuesto debe tener complejidad $O(n)$, donde n es el tamaño de A. Observar que dada la definición de complejidad temporal, se puede suponer $k \ll n$. Dar el pseudocódigo *à la* C++, indicando la complejidad del algoritmo y justificando la respuesta. Se pueden utilizar (sin reescribir) cualquiera de los algoritmos vistos en clase.

Ej. 3. Divide & Conquer

Dado un arreglo $A = [a_1, \ldots, a_n]$ de elementos de tipo α, un *elemento mayoritario* de A es cualquier elemento que ocurra en estrictamente más de $\lfloor n/2 \rfloor$ posiciones. Por ejemplo, si $n = 6$ ó $n = 7$, cualquier elemento mayoritario va a ocurrir en al menos 4 posiciones del arreglo.

El tipo α sólo cuenta con la función igualdad. Esta función puede ser invocada usando el símbolo usual, es decir, dos elementos a_1 y a_2 de tipo α pueden compararse con la expresión $a_1 = a_2$. El tipo α **no** provee ninguna función de orden, con lo cual no es posible establecer si un elemento es mayor o menor que otro.

a) Escribir un algoritmo *à la* C++ para encontrar un elemento mayoritario de un arreglo de α (o para determinar que no hay elementos mayoritarios). Se pueden escribir funciones auxiliares, pero se deberá escribir una función que resuelva el problema con el siguiente tipo:

$$\text{hayMayoritario}(\textit{in } A\text{: array de } \alpha, \textit{ out Elem}\text{: } \alpha) \rightarrow \text{bool}$$

Si A tiene un elemento mayoritario, el valor del parámetro de salida *Elem* debe ser dicho elemento y el valor booleano de retorno debe ser *true*. En caso contrario, el valor de retorno deber ser $false$ y *Elem* puede tener cualquier valor. Considerando que la operación de igualdad entre elementos de tipo α tiene $O(1)$, el algoritmo debe poseer una complejidad $O(n.log(n))$, donde n es el tamaño de A.

b) Calcular y justificar la complejidad del algoritmo propuesto. Para simplificar el cálculo, se puede suponer que n es potencia de dos.