Algoritmos y Estructuras de Datos II
Segundo parcial – 20/11/2010

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 monitorear el control de tráfico de una ciudad. Cada calle de la ciudad tiene un conjunto de semáforos asociados. Cada semáforo pertenece a una única calle, y puede estar en verde o en rojo. Además una calle puede intersecarse con otras calles. Una calle está liberada para circular si y sólo si todos sus semáforos asociados están en verde y todos los semáforos de todas las calles que la intersecan están en rojo. Los semáforos se identifican con un número natural y las calles con un string que representa su nombre. El estado de un semáforo es un tipo enumerado con los valores {rojo, verde}.

El siguiente TAD es una especificación para este problema.

TAD Ciudad

géneros ciudad

exporta ciudad, generadores, observadores, *liberada*

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

observadores básicos

calles	: ciudad	$\longrightarrow$	conj(calles)	
intersecciones	: ciudad c × calle l	$\longrightarrow$	conj(calles)	$\{l \in \text{calles}(c)\}$
semaforos	: ciudad c × calle l	$\longrightarrow$	conj(semaforo)	$\{l \in \text{calles}(c)\}$
estado	: ciudad c × semaforo s	$\longrightarrow$	color	$\{(\exists l)(l \in \text{calles}(c) \land_L s \in \text{semaforos}(c,l))\}$

generadores

crear	: conj(calle)	$\longrightarrow$	ciudad	
intersecar	: ciudad c × calle $l1$ × calle $l2$	$\longrightarrow$	ciudad	$\{l1 \neq l2 \land \{l1, l2\} \subseteq \text{calles}(c) \land_L l1 \notin \text{intersecciones}(c,\ l2)\}$
agSemaforo	: ciudad c × calle l × semaforo s	$\longrightarrow$	ciudad	$\{\neg(\exists l)(l \in \text{calles}(c) \land_L s \in \text{semaforos}(c,l))\}$
cambiarLuz	: ciudad c × semaforo s	$\longrightarrow$	ciudad	$\{(\exists l)(l \in \text{calles}(c) \land_L s \in \text{semaforos}(c,l))\}$

otras operaciones

liberada	: ciudad c × calle l	$\longrightarrow$	bool	$\{l \in \text{calles}(c)\}$
todosEn	: ciudad c × conj(semaforo) ss × color col	$\longrightarrow$	bool	$\{(\exists l : \text{calle})(l \in \text{calles}(c) \land_L ss \subseteq \text{semaforos}(c,\ l))\}$
todasBloqueadas	: ciudad c × conj(calle) cl	$\longrightarrow$	bool	$\{cl \subseteq \text{calles}(c)\}$

axiomas

calles(crear(c)) ≡ c

calles(intersecar(c, l1, l2)) ≡ calles(c)

calles(agSemaforo(c, l, s)) ≡ calles(c)

calles(cambiarLuz(c, s)) $\equiv$ calles(c)
intersecciones(crear(c), l) $\equiv$ $\emptyset$
intersecciones(intersecar(c, l1, l2), l) $\equiv$ **if** ((l = l1) $\vee$ (l = l2)) **then**
  Ag(**if** (l = l1) **then** l2 **else** l1 **fi**, intersecciones(c, l))
else
  intersecciones(c, l)
fi
intersecciones(agSemaforo(c, l1, s), l) $\equiv$ intersecciones(c, l)
intersecciones(cambiarLuz(c, s), l) $\equiv$ intersecciones(c, l)
semaforos(crear(c), l) $\equiv$ $\emptyset$
semaforos(intersecar(c, l1, l2), l) $\equiv$ semaforos(c, l)
semaforos(agSemaforo(c, l1, s), l) $\equiv$ **if** (l1=l) **then** Ag(s, semaforos(c, l)) **else** semaforos(c, l) **fi**
semaforos(cambiarLuz(c, s), l) $\equiv$ semaforos(c, l)
estado(intersecar(c, l1, l2), s) $\equiv$ estado(c, s)
estado(agSemaforo(c, l, s1), s2) $\equiv$ **if** (s1=s2) **then** verde **else** estado(c, s2) **fi**
estado(cambiarLuz(c, s1), s2) $\equiv$ **if** (s1=s2) **then**
  if (estado(c, s1) = verde) **then** rojo **else** verde **fi**
else
  estado(c, s2)
fi
liberada(c, l) $\equiv$ todosEn(c, semaforos(c,l), verde) $\wedge$ todasBloqueadas(c, intersecciones(c, l))
todosEn(c, ss, col) $\equiv$ $\emptyset$?(ss) $\vee_{\text{L}}$ ((estado(c, dameUno(ss)) = col) $\wedge$ todosEn(c, sinUno(ss), col))
todasBloqueadas(c, cl) $\equiv$ $\emptyset$?(cl) $\vee_{\text{L}}$ (todosEn(c, semaforos(c, dameUno(cl)), rojo) $\wedge$ todasBloqueadas(c, sinUno(cl)))

Fin TAD

Se desea diseñar el sistema propuesto, teniendo en cuenta que la operación *cambiarLuz(c,s)* y *estado(c,s)* deben realizarse en $O(log(S))$ y *liberada(c,l)* en $O(tam(l) + \#intersecciones(c, l))$, donde S es la cantidad total de semáforos de la ciudad y $tam(l)$ devuelve la cantidad de caracteres del string l. 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 *cambiarLuz* y *liberada*, indicando la complejidad de cada uno de los pasos.

Ej. 2. Sorting

Un *pico* es un arreglo $[a_1, \ldots, a_n]$ de naturales donde todos sus elementos son distintos y existe un i tal que $1 < i < n$ y los subarreglos $[a_1, \ldots, a_i]$ y $[a_i, \ldots, a_n]$ son el primero creciente y el segundo decreciente, o bien, el primero decreciente y el segundo creciente. El primer caso se lo denomina *pico positivo* y el segundo *pico negativo*. El *representante* de un pico p (notación $rep(p)$) es, en el caso de un pico positivo, su elemento máximo y, en el caso de un pico negativo, su elemento mínimo.

Dado un arreglo de picos $A = [p_1, \ldots, p_n]$, dar un algoritmo que devuelva un arreglo B de tuplas de naturales en donde cada tupla se corresponde con uno y sólo un pico de A. Si una tupla t_i se corresponde con un pico $p_j = [p_{j1}, \ldots, p_{jk}]$, entonces $t_i = \langle p_{j1}, rep(p_j), p_{jk}\rangle$. Las tuplas de B deben estar ordenadas en forma creciente en función de los representantes de sus picos asociados. Por ejemplo:

$$\begin{array}{ll} \textit{entrada} & [[1,3,\mathbf{5},2],[3,\mathbf{7},1],[5,3,\mathbf{0},2],[9,\mathbf{8},10],[8,\mathbf{10},7]] \\ \textit{salida} & [\langle 5,\mathbf{0},2\rangle,\langle 1,\mathbf{5},2\rangle,\langle 3,\mathbf{7},1\rangle,\langle 9,\mathbf{8},10\rangle,\langle 8,\mathbf{10},7\rangle] \end{array}$$

Dado un arreglo de picos A, el algoritmo debe resolver el problema en $O\big(n.log(n)+\sum_{1\leq i\leq n} log(long(A[i]))\big)$, donde n es la longitud de A. Se pueden utilizar (sin reescribir) cualquiera de los algoritmos vistos en clase. Dar el pseudocódigo *à la* C++, indicando la complejidad del algoritmo propuesto y justificando la respuesta.

Ej. 3. Divide & Conquer

Se cuenta con un tablero de $n \times n$ casilleros en donde algunos de sus casilleros contienen puntos y otros están vacíos. Se sabe de antemano que la cantidad total de puntos de cualquier tablero es siempre inferior o igual a una constante k. Se quiere encontrar las coordenadas $\langle x, y\rangle$ de cada uno de los puntos del tablero (las coordenadas de un punto son las coordenadas de la casilla en la que se encuentra), y para eso se cuenta con una función *cantPuntos*. Esta función indica la cantidad de puntos que se encuentran dentro de una región rectangular del tablero, identificada por la posición de sus vértices:

$$\text{cantPuntos}(\mathit{in}\ t\text{: tablero, } \mathit{in}\ x1, y1, x2, y2\text{: nat}) \rightarrow \text{nat} \qquad (x1 \leq x2 \leq \text{tamaño}(t), y1 \leq y2 \leq \text{tamaño}(t))$$

a) Escribir un algoritmo *à la* C++ que utilice la técnica de divide & conquer para encontrar las coordenadas de todos los puntos del tablero cuando $k = 2$. Se pueden escribir funciones auxiliares, pero se deberá escribir una función que resuelva el problema con el siguiente tipo:

$$\text{puntos}(\mathit{in}\ t\text{: tablero}) \rightarrow [\langle x : \text{nat},\ y\text{: nat}\rangle]$$

El arreglo de tuplas resultante debe contener una tupla por cada punto del tablero, con las coordenadas correspondientes. Considerando que la función *cantPuntos* tiene complejidad $O(1)$, el algoritmo dado debe poseer una complejidad estrictamente menor a $O(\text{tamaño}(t)^2)$.

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