Algoritmos y Estructuras de Datos II
Recuperatorio del segundo parcial – 04/08/2007

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 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 (50 puntos)

La flota de Estartrec quiere un sistema eficiente para controlar a sus naves en la batalla. Como los sistemas de la flota no tienen mucha memoria, **es necesario no desperdiciarla**. Sin embargo, por requerimientos legales, ninguna nave puede darse de baja del sistema, aún cuando sea destruida.

Cada nave tiene una tripulación pero no siempre la misma. Interesa saber quiénes abordaban una determinada nave al momento en que fue destruida, lo que sucede cuando recibe su quinto disparo. Se espera que el ritmo de producción se mantenga de manera tal que las naves destruidas nunca superen el 10 % de la flota.

Las naves se identifican por su número de serie, que es secuencial. Los tripulantes por su nombre, alfanumérico acotado.

La operación `RegistrarDisparo()`, que toma una nave, debe tener $O(1)$ en el caso esperado, y hasta $O(n)$ en el peor caso, con n la cantidad total de naves. Como los disparos caen en ráfagas, se desea que registrar el segundo disparo consecutivo a la misma nave tenga siempre $O(1)$.

La operación `QuéNavesTripuló()`, que toma un tripulante, debe tener $O(1)$ en el peor caso. `TripularNave()`, que toma una nave y un tripulante (que podría ya pertenecer a la tripulación) debe tomar $O(n)$ en el peor caso. Sobre `DescenderDeNave()` no hay requisitos.

Además, debe proveerse la función `ConstruirNuevaNave()`.

(30 ptos.) a) Proponga una estructura que permita un diseño de todas las operaciones mencionadas satisfaciendo los órdenes de complejidad pedidos. Justifique todas aquellas decisiones que haya tomado (i.e., suposiciones sobre características de los datos que recibe, descripción breve de la estructura que permite implementar los módulos elegidos como parte de la estructura, etc.).

(15 ptos.) b) Proponga un algoritmo para la operación `ConstruirNuevaNave()` y justifique su orden de complejidad temporal en peor caso.

(5 ptos.) c) Describa cómo cambiarían los órdenes de complejidad temporal si los tripulantes pasaran a identificarse por un string acotado pero numérico.

Ej. 2. Técnicas algorítmicas (30 puntos)

Un *árbol rojo-negro*[1] es un árbol binario que cumple las siguientes propiedades:

1. Las hojas son negras.
2. Los nodos rojos sólo pueden tener hijos negros.
3. Todos los caminos desde la raíz hasta una hoja contienen el mismo número de nodos negros.

Dado un árbol binario y una función color() que toma un nodo del árbol y en $O(1)$ devuelve `negro` o `rojo`, se pide:

[1] Se utiliza para este ejercicio una versión levemente simplificada de la definición real de árboles rojo-negro.

(20 ptos.) a) Escribir un algoritmo que visite cada nodo a lo sumo una vez y que utilice la técnica de Divide & Conquer para determinar si el árbol es rojo-negro.

(10 ptos.) b) Marcar claramente las distintas etapas del algoritmo.

Ej. 3. Ordenamiento (20 puntos)

Proponga un algoritmo de ordenamiento para cada uno de los escenarios que se presentan a continuación. El algoritmo **deberá** valerse de las características particulares de cada escenario para funcionar más eficientemente (aunque eso no altere la complejidad). En todos los casos los arreglos son de números naturales y están numerados de la posición 1 a la n.

Determinar la complejidad temporal de los algoritmos propuestos.

(10 ptos.) a) El subarreglo compuesto por las posiciones pares está ordenado. Ejemplo: [23, 2, 1, 3, 9, 20].

(10 ptos.) b) Además del arreglo, se cuenta con una lista $L = [\langle c_1, f_1 \rangle, \langle c_2, f_2 \rangle, \ldots, \langle c_m, f_m \rangle]$ de m ($m << log(n)$) pares de posiciones que indican grandes fragmentos del arreglo que están compuestos por números continuos. Los fragmentos del arreglo indicados por los pares de L ($A[c_i \ldots f_i]$) podrían no estar ordenados, pero las posiciones en el arreglo de cada son definitivas. Ejemplo: [1, 3, . . . , 102, 101, . . . 5500, 2, 9000, 7340, 11000, . . . , 10000, . . .]. L = ($\langle 2, 5498 \rangle$, $\langle 5502, 6502 \rangle$).

Si utiliza algoritmos vistos en clase no es necesario escribirlos.