Algoritmos y Estructuras de Datos II
Segundo parcial – 4 de Noviembre de 2015

Aclaraciones

- El parcial es **a libro abierto**.
- Cada ejercicio debe entregarse **en hojas separadas**.
- Incluir en cada hoja el número de orden asignado, número de hoja, apellido y nombre.
- Al entregar el parcial, completar el resto de las columnas en la planilla.
- Cada ejercicio se calificará con **P**romocionado, **A**probado, **R**egular, **I**nsuficiente o **N**o Entregó. Notar que **I** es distinto de **N**.
- El parcial completo está aprobado si el primer ejercicio tiene al menos **A**, y entre los ejercicios 2 y 3 hay al menos una **A** y a lo sumo una **I**. Para más detalles, ver "Información sobre la cursada" en el sitio Web.

Ej. 1. Diseño

Para combatir el flagelo de los *tuppers* abandonados en la heladera, una empresa está diseñando el sistema de control del robot *Tupperminator*, encargado de la gestión y limpieza de los *tuppers* almacenados.

Los dueños de la heladera periódicamente introducen porciones de comida guardadas en *tuppers*. Cada *tupper* está etiquetado con el nombre del alimento, su peso en gramos y su fecha de vencimiento desde comenzado el *Tupperminator*.

Además de almacenar comida, los usuarios pueden pedirle a *Tupperminator* que les devuelva algún tupper con el alimento previamente introducido, mencionando el nombre del alimento, mediante la operación `Comer`. También, pueden decirle al robot "voy a tener suerte", a lo que *Tupperminator* les devolverá alguno de los alimentos más próximos a vencerse. En caso de haber varios alimentos igualmente viejos, el robot priorizará a los más pesados.

Finalmente, la principal tarea de *Tupperminator* será descartar la comida vencida al comienzo del día. Para evitar malos olores, el robot directamente *desintegra* los tuppers vencidos.

TAD TUPPER ES TUPLA(NOMBRE:STRING, PESO:NAT, VENCIMIENTO:NAT)

TAD TUPPERMINATOR

generadores

nuevo	:		⟶	Tupperminator
agregarTupper	:	Tupperminator × tupper	⟶	Tupperminator
nuevoDia	:	Tupperminator	⟶	Tupperminator

observadores básicos

tuppers	:	Tupperminator	⟶	MConj(tupper)
diaActual	:	Tupperminator r	⟶	Nat
dias	:	Tupperminator r × Tupper t	⟶	Nat $\{t \in tuppers(r)\}$

otras operaciones

voyATenerSuerte		Tupperminator r	⟶	tupper $\{\text{cantElems(tuppers(r))} > 0\}$
comer	:	Tupperminator r × string c	⟶	Tupperminator $\{(\exists t : tupper) t \in tuppers(r) \land t.nombre = c\}$

Fin TAD

Diseñe del TAD TUPPERMINATOR debe respetar los siguientes requerimientos de complejidad temporal en **peor caso**:

- COMER(r, n) debe resolverse en $O(|n| + \log P_{max} + \log D)$.
- AGREGARTUPPER(r, t) debe resolverse en $O(|t.nombre| + \log P_{max} + \log D)$. Puede consumir hasta $O(\text{dias}(r, t))$ extra solamente si $\text{dias}(r, n) > F$.
- VOYATENERSUERTE(r) debe resolverse en $O(|n_{max}| + \log D + \log P_{max})$
- NUEVODIA(r) debe resolverse en $O(\log D + P_{max}|n_{max}|)$ solamente si se vencerá algún alimento, $O(1)$ en caso contrario.

donde r es el Tupperminator en cuestión, D es la cantidad de días distintos en los que vence algún alimento, $dias(r, t)$ es la cantidad de días que le quedan a t para vencerse, $|n_{max}|$ es el nombre más largo de alimento almacenado actualmente, P_{max} es la cantidad máxima de alimentos que vencen en un mismo día, y F es la cantidad máxima de días que tuvo un alimento hasta vencerse, de los que estuvieron alguna vez en la heladera.

Se pide:

(a) Diseñar una estructura para representar el Tupperminator. Indicar en castellano el invariante de representación de la estructura propuesta, explicando para qué sirve cada parte, o utilizando nombres autoexplicativos.

(b) Implementar los algoritmos correspondientes a las operaciones *comer* y *nuevoDia*, respetando las complejidades temporales estipuladas. Explicar el resto en castellano.

(c) Justificar claramente cómo y por qué los algoritmos, la estructura y los tipos soporte permiten satisfacer los requerimientos pedidos. No es necesario diseñar los módulos soporte, **pero sí describirlos, justificando por qué pueden (y cómo logran)** exportar los órdenes de complejidad que su diseño supone.

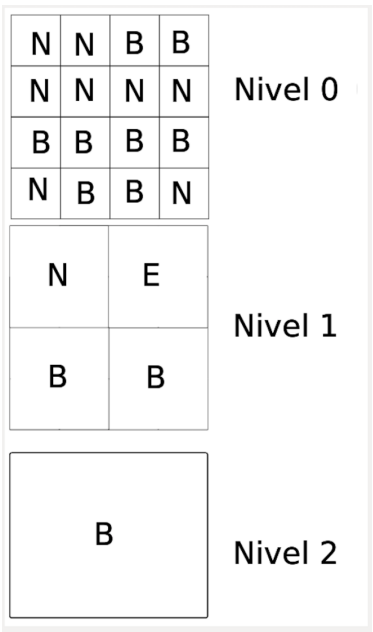
Ej. 2. Ordenamiento

Un arreglo de n naturales se dice que es *piola* si a lo sumo contiene $\log n$ valores **distintos** mayores que $n \log n$. Notar que en un arreglo *piola* podría haber varias posiciones con el mismo valor.

a) Dar un algoritmo que ordene arreglos piola en tiempo **estrictamente menor** que $O(n \log n)$ en el peor caso.
b) Justificar detalladamente que el algoritmo cumple con la complejidad pedida y que ordena correctamente el arreglo de entrada.

Ej. 3. Dividir y conquistar

El famoso y archiconocido juego **3mb0l3** se juega, como todos saben, sobre un tablero cuadrado de lado n con $n = 2^k$ para algún k natural. El tablero está dividido en $n \times n$ casillas. Participan dos jugadores, cada uno tiene su identificador **B** o **N**. En algunas casillas del tablero, un jugador colocó su identificador. Puede haber casillas vacías y no hay más de un identificador por casilla. Una vez finalizado el juego, para decidir el ganador se consideran $k+1$ niveles distintos, numerados de 0 a k. El nivel i se forma dividiendo el cuadrado original en $2^{k-i} \times 2^{k-i}$ casillas. Así, el nivel k consta de una sola casilla, el $k-1$ de un tablero de 2×2 y así siguiendo. Se puede ver en la figura cómo quedan los distintos niveles. Para cada casilla de cada nivel (salvo el 0), se toma como dueño de la misma al jugador que más casillas controla del nivel anterior teniendo en cuenta solamente las que están contenidas en la casilla grande. Se coloca el identificador del dueño en cada casilla, y en caso de empate se coloca una **E**. Para cada nivel, se toma como ganador del nivel al que más casillas controla de ese nivel. Acá nuevamente puede haber empate. Por último se dice que el ganador del juego es aquel que haya ganado mayor cantidad de niveles. Realizar un algoritmo que dado un tablero, con los dueños de cada casillero en el nivel 0, devuelva si gana **B** o **N** todo el juego, o **E** si hay empate.



Se pide dar un algoritmo que utilice la técnica de Dividir y Conquitar y además se debe calcular y justificar su complejidad temporal.

En el ejemplo de la figura, se tiene un tablero con $n = 2^2$, por lo que hay niveles de 0 a 2. Por ejemplo, en el nivel 1 la casilla inferior izquierda es controlada por **B**, pues entre las casillas correspondientes en el nivel 0 hay tres **B** y una **N**. El nivel 0 es empate pues hay igual cantidad de casillas controladas por cada jugador, el nivel 1 lo gana **B** pues hay dos **B** y sólo una **N** y el nivel 2 lo gana **B**. Como **B** ganó dos niveles y **N** ninguno, el algoritmo debe dar **B** como respuesta.