Algoritmos y Estructuras de Datos II
Segundo parcial – 13 de junio de 2014

Aclaraciones

- El parcial es **a libro abierto**.
- Entregar cada ejercicio **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 **MB**, **B**, **R** o **M**, y podrá recuperarse independientemente de los demás. La cursada podrá aprobarse con hasta 1 (un) ejercicio **R** (regular) en cada parcial (post-recuperatorios), siempre y cuando ninguno de ellos sea un ejercicio 1. Para más detalles, ver "Información sobre la cursada" en el sitio Web.

Ej. 1. Diseño

El sindicato de piratas de Mêlée Island™ desea un sistema para controlar las peleas que ocurren en la isla. En ella conviven varios piratas, que liberan sus tensiones bebiendo grog y peleándose entre ellos. El grog viene en distintas variedades, tales como *Grog Clásico*, *Grog XD*, etc. Cada variedad de grog tiene una graduación alcohólica distinta. Como todo el mundo sabe, la habilidad con la espada es secundaria en las peleas. Lo que más importa es cuántos insultos sabe cada pirata. Los insultos tienen distinta potencia: es clarísimo que decir "Peleas como un granjero" tiene menor potencia que insultar diciendo "Ordeñaré tu sangre hasta la última gota".

En una pelea, gana el pirata que sume la mayor potencia entre todos los insultos que conoce. En caso de empate, gana el pirata que esté más borracho, es decir aquel cuyo último grog bebido haya sido más fuerte. Si aun así hubiese empate, ninguno de ellos gana. Siempre que un pirata no gana, aprende algún nuevo insulto tomado de aquellos que no conoce (si es que no conoce todos ya)[1]. Los piratas recién llegados no conocen ningún insulto, y se sabe que el último grog que bebieron antes de llegar a Mêlée es el *Grog Free*, que tiene 0 % de alcohol.

TAD INSULTO **ES** TUPLA(STRING, NAT)

TAD GROG **ES** TUPLA(STRING, NAT)

TAD PIRATA **ES** NAT

TAD MÊLÉE

generadores

empezar	: conj(pirata) × conj(insultos)	⟶	melee	
agregarPirata	: melee m × pirata p	⟶	melee	$\{p \notin \text{piratas}(m)\}$
agregarInsulto	: melee m × insulto i	⟶	melee	$\{i \notin \text{insultos}(m)\}$
beberGrog	: melee m × pirata p × grog	⟶	melee	$\{p \in \text{piratas}(m)\}$
pelear	: melee m × pirata p_1 × pirata p_2	⟶	melee	$\{p_1 \neq p_2 \wedge \{p_1, p_2\} \subseteq \text{piratas}(m)\}$

observadores básicos

piratas	: melee	⟶	conj(pirata)	
insultos	: melee	⟶	conj(insulto)	
insultosQueConoce	: melee m × pirata p	⟶	conj(insulto)	$\{p \in \text{piratas}(m)\}$
últimoGrogQueBebió	: melee m × pirata p	⟶	grog	$\{p \in \text{piratas}(m)\}$
peleas	: melee	⟶	dicc(pirata, dicc(pirata, secu(bool)))	

Fin TAD

Diseñe el TAD MÊLÉE respetando los siguientes requerimientos de complejidad temporal en **peor caso**:

- INSULTOS(m) debe resolverse en $O(1)$
- PIRATAS(m) debe resolverse en $O(1)$
- ÚLTIMOGROGQUEBEBIÓ(m, p) debe resolverse en $O(\log(P))$
- INSULTOSQUECONOCE(m, p) debe resolverse en $O(\log(P))$

[1] HINT: el dameUno de conjunto debería ser determinístico.

- PELEAR(m, p_1, p_2) debe resolverse en $O(\log(P))$
- AGREGARINSULTO(m, i) debe resolverse en $O(P \times Ins)$)

donde P es la cantidad de piratas en m e Ins es la cantidad de insultos en m. Puede suponer que la longitud de cualquier insulto estará acotada por 200 caracteres.

a) Muestre la estructura de representación propuesta. Explique para qué sirve cada parte de la estructura, o utilice nombres autoexplicativos.

b) Escriba el pseudocódigo del algoritmo PELEAR(**inout** m : estr, **in** p_1 : pirata, **in** p_2 : pirata).

Justifique 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

a) Guybrush Threepwood tiene muchos problemas de ordenamiento que resolver, y necesita averiguar si el algoritmo COUNTINGSORT que vimos en clase es o no estable. Explique por qué {sí, no} lo es, y cómo se podría modificarlo para que {no, sí} lo sea. Puede utilizar pseudocódigo y/o castellano, pero justifique adecuadamente o LeChuck le hará caminar por la tabla.

b) "Hablando de estabilidad", gruñó LeChuck, "¿por qué este maldito algoritmo no es estable?" Lo que LeChuck estaba haciendo era utilizar un módulo COLADEPRIORIDAD (diseñado sobre heap minimalista), encolando primero los n elementos a ordenar y finalmente llamando n veces a la operación desencolar(). Explíquele al malvado jefe pirata por qué este algoritmo no es estable. Justifique claramente su respuesta en base al invariante de representación del módulo COLADEPRIORIDAD.

Ej. 3. Dividir y conquistar

Un primo emprendedor de Stan está planeando construir una serie de lujosos *resorts* en la costa sur de Mêlée Island™. Esto escandalizó a la Gobernadora Elaine Marley, quien exigió saber qué partes del paisaje de la isla iban a quedar tapadas por las obras. Para ello nos encomendó la tarea de calcular, en base a los datos de los edificios a construir, el perfil de alturas resultante.

Las dimensiones de cada edificio están dadas por una terna de NATs ($x\_inicial$, $ancho$, $altura$). Todos los edificios se construyen al nivel del mar. Para poder predecir las alturas del perfil resultante, Elaine desea obtener una lista de pares ($x\_inicial$, $altura$) ordenados por su coordenada x. No debe haber pares redundantes: de un par al siguiente debe haber siempre un cambio en la altura.

Ejemplo: Para los edificios $A, B, C, D = \langle(0, 2, 7), (1, 1, 10), (1, 5, 6), (4, 3, 10)\rangle$ tenemos el perfil de alturas $\langle(0, 7), (1, 10), (2, 6), (4, 10), (7, 0)\rangle$, tal como muestra la figura.

Figura 1: *Izquierda*: Edificios, *Derecha*: Perfil de alturas

Se pide calcular mediante la técnica de *Divide and Conquer* la lista de pares que describe el perfil de alturas para un arreglo de N edificios. Dada la urgencia queremos conocer el perfil de alturas en tiempo $O(N \log(N))$.