Algoritmos y Estructuras de Datos II
Diseño avanzado – 20 de Octubre de 2017

Ej. 1. Diseño 1

Estamos comenzando la exploración intensiva del planeta Marte utilizando unidades *Mars Rover*. El área de exploración se ha discretizado en coordenadas enteras y en cualquier momento pueden desplegarse unidades *rover*, aunque siempre en la celda (0,0) de la discretización. Los rovers desplegados pueden moverse en cualquier momento en cualquiera de sus cuatro direcciones (norte, sur, este u oeste), siempre que sea hacia una celda válida, y estos movimientos se registran con fines estadísticos. En cualquier momento puede también desactivarse definitivamente cualquiera de las unidades activas. Dado un *itinerario* (i.e., una secuencia de movimientos desde la celda (0,0)), la consulta más requerida es la de saber cuántos rovers (activos o inactivos) realizaron exactamente este itinerario. Si bien no interesa conocer el recorrido hecho por un rover inactivo, notar que este dato es importante para la consulta por itinerario.

TAD ROVER es NAT, TAD DIRECCION es {N, S, E, O}, TAD CELDA es tupla<NAT,NAT>

TAD EXPLORACIÓN

observadores básicos

activos	:	exploracion	$\longrightarrow$	conj(rover)	
inactivos	:	exploracion	$\longrightarrow$	conj(rover)	
recorridoActual	:	exploracion e × rover r	$\longrightarrow$	secu(direccion)	{r ∈ activos(e)}
vecesRecorrido	:	exploracion e × secu(direccion) i	$\longrightarrow$	nat	

generadores

iniciar	:		$\longrightarrow$	exploracion	
desplegar	:	exploracion e × rover r	$\longrightarrow$	exploracion	{$r \notin$ rovers(e)}
mover	:	exploracion e × rover r × direccion d	$\longrightarrow$	exploracion	{$r \in$ activos(e) ∧ movimientoValido(e,r,d)}
desactivar	:	exploracion e × rover r	$\longrightarrow$	exploracion	{$r \in$ activos(e)}

otras operaciones

rovers	:	exploracion e	$\longrightarrow$	conj(rover)	
ubicación	:	exploracion e × rover r	$\longrightarrow$	celda	{$r \in$ activos(e)}

axiomas ∀e: exploracion, ∀r: rover, ∀d: direccion

rovers(e) ≡ activos(e) ∪ inactivos(e)

...

Fin TAD

Se debe realizar un diseño que cumpla con los siguientes órdenes de complejidad temporal en el peor caso, siendo r la cantidad de rovers activos en el sistema:

- Desplegar y desactivar un rover en $O(\log r)$.
- Mover un rover en $O(\log r)$.
- Saber dónde está un rover dado en $O(\log r)$.
- Saber cuántos rovers realizaron el itinerario T en $O(long(T))$.
- Obtener el conjunto de rovers activos y el de inactivos, cada operación en $O(1)$.

Se pide:

1. Escriba la estructura de representación del módulo "Exploración" explicando detalladamente qué información se guarda en cada parte de la misma y las relaciones entre las partes. Describa también las estructuras de datos subyacentes.
2. Escriba el algoritmo para mover un rover y justifique el cumplimiento de los órdenes solicitados. Para cada una de las demás funciones, descríbalas en castellano, justificando por qué se cumple el orden pedido con la estructura elegida.

Ej. 2. Diseño 2

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 $O(\log(P))$
- 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.

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