Algoritmos y Estructuras de Datos II
Recuperatorio del segundo parcial – 1º de Julio de 2016

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, o **I**nsuficiente.
- 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**. Para más detalles, ver "Información sobre la cursada" en el sitio Web.

Ej. 1. Diseño

Hace mucho tiempo, en una galaxia muy muy lejana, existía un mundo muy poco desarrollado habitado únicamente por simios bajo el control de un ser supremo. Estos simios todavía no eran muy avanzados evolutivamente y vivían arriba de los árboles, trasladándose mediante lianas. En este mundo algunos árboles estaban conectados mediante lianas que les permitía a los simios trasladarse de un árbol a otro. Si no había liana conectando un par de árboles los simios se tiraban directamente y se les computaba una falta a su propia evolución. El ser supremo vigilante del universo deseaba controlar el desarrollo evolutivo de este mundo monitoreando el comportamiento de los simios y eliminando a los menos evolucionados, es decir, los que poseían mayor cantidad de faltas evolutivas. Por ende, cada tanto realizaban requisas en los árboles aplicando el principio *darwiniano* de selección natural y eliminando al simio menos evolucionado del árbol requisado. El siguiente TAD, modela el comportamiento mencionado.

TAD Monasterio

TAD simio es string, **TAD** árbol es nat

género Mon **exporta** generadores, observadores

generadores

crear : dicc(árbol, conj(simio)) u × dicc(árbol, conj(árbol)) l ⟶ Mon
$\left\{(\forall a_1 \neq a_2 \in claves(u) \Rightarrow_L obtener(a_1) \cap obtener(a_2) = \emptyset) \wedge (claves(l) \cup unir(significados(l)) \subseteq claves(u))\right\}$

saltar : simio s × árbol a × Mon m ⟶ Mon
$\{a \in bosque(m) \wedge s \in todosLosSimios(m) \wedge a \neq \text{suÁrbol}(s, m)\}$

requisar : árbol a × Mon m ⟶ Mon $\{a \in bosque(m) \wedge \neg\emptyset(\text{simiosColgados}(a, m))\}$

observadores básicos

simiosColgados : árbol a × Mon m ⟶ conj(simio) $\{a \in bosque(m)\}$

#faltas : simio s × Mon m ⟶ nat $\{s \in \text{todosLosSimios(m)}\}$

⊗ bosque : Mon m ⟶ conj(árbol)

⊗ hayLiana : árbol a_1 × árbol a_2 × Mon m ⟶ bool $\{\{a_1, a_2\} \subseteq bosque(m)\}$

otras operaciones

todosLosSimios : Mon ⟶ conj(simio)

suÁrbol : simio s × Mon m ⟶ árbol $\{s \in \text{todosLosSimios(m)}\}$

menosEvol : conj(simio) s × Mon m ⟶ simio $\{cs \subseteq \text{todosLosSimios}(m) \wedge \neg\emptyset(cs)\}$

Nota: los axiomas con el símbolo ⊗ no están especificados aquí.

axiomas

simiosColgados(a,crear(ds,li)) ≡ obtener(a,ds)

simiosColgados(a,saltar(s,a',m)) ≡ (simiosColgados(a,m) ∪ **if** a = a' **then** {s} **else** ∅ **fi**) - (**if** a = suÁrbol(s,m) **then** {s} **else** ∅ **fi**)

simiosColgados(a,requisar(a',m)) ≡ simiosColgados(a,m) - (**if** a = a' **then** {menosEvol(simiosColgados(a, m), m)} **else** ∅ **fi**)

menosEvol(cs,m) ≡ **if** ∅?(sinUno(cs)) **then** dameUno(cs) **else if** #faltas(dameUno(cs),m) > #faltas(menosEvol(sinUno(cs),m),m) **then** dameUno(cs) **else** menosEvol(sinUno(cs),m) **fi fi**

#faltas(s,crear(ds,li)) ≡ 0

#faltas(s,saltar(s',a,m)) ≡ #faltas(s,m) + β(s=s' ∧¬ hayLiana(suÁrbol(s,m),a)))
#faltas(s,requisar(a,m)) ≡ #faltas(s,m)
todosLosSimios(m) ≡ $\bigcup_{a \in bosque(m)}$ simiosColgados(a,m)
suÁrbol(s,m) ≡ a / ($\exists a$: árbol $s \in$ simiosColgados(a,m))

Fin TAD

Contexto de uso

Los árboles están numerados de 0 a N-1, las lianas son direccionales y puede haber árboles sin lianas entre ellos.

Complejidades requeridas

- saltar(s, a_d) es $\mathcal{O}(|s| + \log(S_{a_o}) + \log(S_{a_d}))$ donde a_o es el árbol donde está el mono s
- requisar(a) es $\mathcal{O}(\log(S_a))$
- todosLosSimios es $\mathcal{O}(1)$, debe devolver un iterador cuya operación AVANZAR puede no ser $\mathcal{O}(1)$
- #faltas(s) es $\mathcal{O}(|s|)$
- hayLiana(a_1, a_2) es $\mathcal{O}(1)$

donde S_x representa la cantidad de monos en el árbol x y $|s|$ es la longitud del string s.

Se pide:

(a) Diseñar una estructura para representar el TAD Monasterio. Indicar en castellano el invariante de representación de la estructura propuesta, explicando para qué sirve cada parte, o utilizando nombres autoexplicativos.
(b) 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.
(c) Escribir los algoritmos correspondientes a las operaciones CREAR y SALTAR, justificando las complejidades temporales obtenidas.
(d) **(opcional)** Explicar en castellano cómo se podrían implementar los algoritmos de todas las operaciones exportadas (observadores y generadores) con la estructura propuesta.

Ej. 2. Ordenamiento

Sea el conjunto de naturales $U = \{1, ..., n\}$ y una partición $\mathcal{F} = \mathcal{F}_1..\mathcal{F}_k$ de U; o sea $U = \mathcal{F}_1 \cup ... \cup \mathcal{F}_k$ y $(\forall i, j \in \{1...k\} \wedge i \neq j \Rightarrow \mathcal{F}_i \cap \mathcal{F}_j = \emptyset)$ y $\mathcal{F}_i \neq \emptyset$.

Para un conjunto dado $C \subseteq U$ interesa obtener, si existe, un $\mathcal{F}_i \subseteq \mathcal{F}$ tal que $C \subseteq \mathcal{F}_i$. Por ejemplo, si $\mathcal{F} = \{\mathcal{F}_1 = [1, 2, 5], \mathcal{F}_2 = [7, 6, 3], F_3 = [9, 10, 8], F_4 = [4]\}$ y $C = [4, 10, 9, 1]$, no existe solución, pero para $C = [3, 7]$ la respuesta es $F_2 = [7, 6, 3]$.

Se pide implementar la operación cubrimiento(in F: arreglo(arreglo(nat)), in C: arreglo(nat)) $\rightarrow res$: nat de manera que $res = 0$ si no hay solución, y $res = i$ con $i \in \{1..k\}$ si $C \subseteq F_i$. Para el ejemplo anterior, con $F = [[1, 2, 5], [7, 6, 3], [9, 10, 8], [4]]$ y $C = [3, 7]$, cubrimiento(F,C) = 2. Puede asumir que cada arreglo $F[i]$ no tiene valores repetidos y que F es una partición válida de U.

a) Dar un algoritmo que resuelva el problema en tiempo $O(n)$ en el peor caso.
b) Justificar detalladamente que el algoritmo cumple con la complejidad pedida y que calcula correctamente el resultado deseado.

Ej. 3. Dividir y conquistar

Se tiene un arreglo ordenado de n números naturales consecutivos con k *huecos*, es decir, en el arreglo están presentes todos los elementos dentro de un rango determinado salvo una cantidad k de ellos. Por ejemplo, el arreglo $A = [11, 12, 13, 15, 16, 19, 20, 21]$ tiene *huecos* en los valores $[14, 17, 18]$. Se pide encontrar los valores de todos los *huecos* de un arreglo.

a) Dar un algoritmo que use la técnica de *Divide and Conquer* y resuelva el problema en tiempo $\mathcal{O}(k \log(n))$ en el peor caso.
b) Marcar claramente qué partes del algoritmo se corresponden a *dividir*, *conquistar* y *unir* subproblemas.

c) Justificar detalladamente que el algoritmo cumple con la complejidad pedida, asumiendo (solamente para el análisis de complejidad) que $k = \mathcal{O}(1)$.