

Algoritmos y Estructuras de Datos

Segundo Parcial - Viernes 22 de Noviembre de 2024

#Orden	Libreta	Apellido y Nombre	E1	E2	E3	E4	Nota Final
126	752/23	HUGO ERNST FEDENICO	20	19	24	26	98-

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, más los dos apuntes de la materia
- El parcial se aprueba con 50 puntos y hay que tener 60 puntos de promedio entre los dos parciales

Cacerola Santiago
Requejada Santiago

E1. Complejidad (20 pts)

Sean $f, g, h : \mathbb{Z} \rightarrow \mathbb{Z}$. Indique si las siguientes afirmaciones son verdaderas o falsas. En caso de ser verdaderas, justifique formalmente con las definiciones o propiedades vistas en clase. En caso de ser falsas, dé un contrajeemplo.

- Si $f \times g \in \Theta(h)$, entonces $h \in O(g)$
- $f \in O(g) \Rightarrow O(g) \cap \Omega(f) \neq \emptyset$
- Si el mejor caso de un algoritmo es $\Theta(n)$, el peor no puede ser $\Omega(\log(n))$

E2. Rep&Abs (20 pts)

Científico, Paper es \mathbb{Z}

```

TAD CitasCientificas {
    obs publicaciones: dict(Paper, seq(Científico))
    obs citas: dict(Paper, conj(Paper))
    proc iniciarSistema() : CitasCientificas
        asegura(El sistema comienza con ambos diccionarios vacíos)
    proc registrarPaper(inout cc : CitasCientificas, in p : Paper, in cs : seq(Científico))
        requiere(El paper no se encuentra en el sistema previamente)
        asegura(Registra el paper en publicaciones con su lista de autores y en citas con vacío)
    proc citarPaper(inout cc : CitasCientificas, in p : Paper, in c : Paper)
        requiere(Algunos papers existen y el paper c no lo había sido citado previamente por el paper p)
        asegura(Se integra c al conjunto de citas de p en citas)
}

```

```

Módulo CitasCientificasImpl implementa CitasCientificas <
var publicaciones: Diccionario<Científico, Conjunto<Paper>>
// Todos los científicos que han publicado algún paper asociados a todos sus papers
var autores: Diccionario<Paper, Vector<Científico>>
// Para cada paper, la lista de científicos en el orden que aparecen en la publicación
var citas: Conjunto<Tupla<Paper, Paper>>
// Tuplas donde la primera componente es el paper que cita y la segunda el paper citado
var masCitados: ColaDePrioridadMax<Tupla<int, Paper>>
// La primera componente de las tuplas es la cantidad de veces que el paper de la segunda fue citado.
// La prioridad está determinada solamente por la primera componente
>

```

- Indique si cada una de las sentencias propuestas pertenece al invariante de representación para la estructura propuesta
 - Dados dos científicos distintos en **publicaciones**, sus significados son disjuntos
 - Para todo paper de **autores**, no hay repetidos en el vector de su significado
 - Para todo par de tuplas del tipo **citas** si existe (a,b) no existe (b,a)
 - No hay tuplas repetidas en **citas**
 - Hay la misma cantidad de claves distintas en **autores** que elementos en **citas**
 - Para cada tupla de **citas**, ambos papers son claves de **autores**
- Proponga dos sentencias más para el invariante de representación (en castellano preciso)
- Escribir la función de abstracción en lógica de primer orden

E3. Diseño (30 pts)

Un instituto de cocina quiere organizar las inscripciones a su Escuela Culinaria de Invierno (ECI) donde ofrecen varias materias con capacidad limitada y nombres únicos de largo acotado. Los alumnos pueden intentar inscribirse a las materias y el sistema les informará si han quedado inscriptos o no (en caso que los inscriptos a la materia ya alcanzaran su capacidad). Si no pudieran inscribirse, igualmente quedan registrados en el sistema y serán agregados a la materia si algún inscripto se da de baja, por estricto orden de inscripción.

El sistema deberá implementar las siguientes operaciones en la complejidad pedida, donde a_m es la cantidad de alumnos inscriptos a la materia m . En todos los casos se puede asumir que la materia está registrada en el sistema. Materia es string. Alumno y Capacidad son int.

- proc abrirMateria(inout i : Inscripciones, in m : Materia, in c : Capacidad)
 - Descripción: Abre la inscripción a una materia
 - Complejidad: $O(1)$
- proc IntentarInscripcion(inout i : Inscripciones, in m : Materia, in a : Alumno) : Bool
 - Descripción: Intenta inscribir a un alumno a una materia a la que no está inscripto. Devuelve si el alumno quedó inscripto o no
 - Complejidad: $O(\log(a_m))$
- proc DarseDeBaja(inout i : Inscripciones, in m : Materia, in a : Alumno)
 - Descripción: Saca al alumno de los inscriptos (donde ya debería estar) y agrega al siguiente de la lista de espera
 - Complejidad: $O(\log(a_m))$
- proc InscriptosMateria(in i : Inscripciones, in m : Materia) : Conjunto(Alumno)
 - Descripción: Devuelve el conjunto de alumnos inscriptos en la materia
 - Complejidad: $O(1)$

Se pide:

- Plantear la estructura de representación del módulo ECIInscripcionesImpl, que provea las operaciones mencionadas cumpliendo las complejidades requeridas
- Escribir el algoritmo de DarseDeBaja. Justifique detalladamente que se cumple la complejidad requerida
- Teórica. ¿Qué características de un Heap permiten representarlo con un arreglo? ¿Qué permite saber cuáles son los nodos hijos y cuál es el padre? Desarrolle en no más de 15 renglones

E4. Sorting (30 pts)

- Teórica. Dado un arreglo de tamaño n ¿Por qué todos los algoritmos de ordenamiento tienen complejidad $\Omega(n^2)$? ¿Por qué todos los algoritmos de ordenamiento tienen complejidad $\Omega(n \log(n))$ cuando no se tiene información previa sobre sus elementos? Desarrolle en no más de 15 renglones
- Se está por realizar un gran censo de perros en la Ciudad de Buenos Aires. La información recaudada llegará en forma de un arreglo de tuplas `<Nombre, Raza, Edad, Peso>`. Las razas son strings de largo acotado L , pero no sabemos cuántas razas pueden existir, aunque sí sabemos que entre ellas está la mejor raza de todas: raza Perro. Las edades serán enteros positivos entre 1 y 20 y los pesos serán de tipo float.

Proponga un algoritmo que organice la información en dos arreglos:

- Uno con los perros raza Perro ordenados por edad y, en caso de empate, por peso, ambos en forma ascendente
- Otro arreglo con la información de las razas y la cantidad de perros de cada una en forma de tuplas `<Raza, int>`, ordenadas en forma decreciente por la cantidad y, en caso de empate, alfabéticamente por la raza

La complejidad temporal del algoritmo debe ser $O(n + p \log(p))$, donde n es el total de perros y p el total de perros raza Perro.

Ejemplo

Si recibimos:

```
[<Nulus, Perro, 4, 15.3>, <Joe, Cocker, 14, 8.0>, <Lady, Caniche, 10, 6.1>, <Dobby, Perro, 7, 18.3>
<Res, Dogo, 3, 50.0>, <Haru, Perro, 7, 30.0>, <Clementina, Perro, 14, 10.0>, <Choqui, Caniche, 16, 5.0>]

Deberíamos devolver:
P: [ <Nulus, 4, 15.3>, <Dobby, 7, 18.3>, <Haru, 7, 30.0>, <Clementina, 14, 10.0>]
R: [ <Caniche, 2>, <Cocker, 1>, <Dogo, 1>]
```

a) FEDERICO HUBERTH

Hojas ① ②

a) Falso. Sea $f(n) = n$ y $g(n) = n$. ($f: \mathbb{Z} \rightarrow \mathbb{Z}$ y $g: \mathbb{Z} \rightarrow \mathbb{Z}$)
 $(f \times h)(n) = n \cdot n = n^2$.

Sea h una función, $h: \mathbb{Z} \rightarrow \mathbb{Z}$, $h(n) = n^2$
Entonces $(f \times h) \in \Theta(h)$ (trivialmente)

Con estos datos cumplimos la hipótesis.

Pero ~~$h \notin O(g)$~~ $n^2 \notin O(n)$ ✓

b) Es verdadero, intentaré demostrarlo con palabras
 ~~$f \in O(g)$, o bien g es de la misma familia~~
que f , o $g \gg f$.

~~$\Omega(f)$ no a ser el conj de todas las funciones tq $h \in \Omega(f) \Leftrightarrow \exists k > 0$, no tq $n > n_0 \Rightarrow h(n) \geq k f(n)$.~~

~~Ocasionalmente~~

c) Falso.

Sea A un arreglo de enteros y el sig
algoritmo

• var j: int

j := 0

while (j < A.length()) do // $\Theta(n)$, n = |A|

j := j + 1

end while

Aquí, tanto el mejor caso, como el peor son $\Theta(n)$. Es decir, ~~el peor caso~~

Por def $\Theta(n) = \Omega(n) \cap \mathcal{O}(n)$.

O sea, si el peor caso está en $\Omega(n)$,

también lo está en $\Omega(\log(n))$

($n \in \Omega(\log(n))$). ✓

FEDERICO HUERTA

Hoja ①
10

1) b) $f \in O(g)$

Y ademas, trivialmente, $f \in \Omega(f)$ ✓

O sea que $f \in O(g)$ y $f \in \Omega(f)$

O sea que $O(g) \cap \Omega(f)$ tiene al

menos un elemento, o sea

$O(g) \cap \Omega(f) \neq \emptyset$, es verdadero. ✓

1) Q)

FEDERICO HUERTH

Haga

I) Falso ✓ Puede haber más de un Científico por paper.

Si los Científicos son autores del mismo Paper, entonces sus valores no son distintos.

II) ~~Verdadero, pero~~

Si Llam no tiene sentido que haya repetidos, en el reg ~~no hay~~ del TAD no hay nada que lo impida.

Y al registrar Paper, el param "is" es de tipo seq, con lo cual podría tener repetidos.

Respuesta: Por sentido común Verdadero !! ✓

III) Verdadero ✓ Se deduce del reg.

Sea $p = \text{paper}^1$ y $t = \text{paper}^2$

Si cito $\text{citarPaper}(p, c)$, cuando quiero $\text{citarPaper}(c, p)$, el ahora "p" posee a ser el c, y "c" posee a ser el p. Básicamente pide que $c \neq p$ no hay si el citado por(p), no cumple el reg.

Además, no tiene sentido, siempre un "paper" sale antes que otro. ¿Cómo hacen para citarse mutuamente?

IV) Verdadero. Además que no tiene sentido poder ver que los valores de cada key del obs citos son conj. y como los dict No tienen claves repetidas, no se puede dar.

V) Falso. Yo vimos que si un paper p cito a un paper c, éste no lo puede citar.

Imaginemos |autores|=2 (P y C)
pero como max puede haber 1 cita
(o ninguna)

VI) Verdadero. Se deduce del requerimiento de citar Paper.

b) ~~Primer~~ Definimos la relación entre publicaciones y autores. Para cada Paper de cada Científico en publicaciones, en esa clave Paper existe un autores, y en el vector encontraremos el científico.

Para cada Científico de cada Paper en autores la clave científica existe, y además paper pertenece al cfo. ✓

FEDERICO HOERTH Hoje ③ ②

Es un "doble implíco" así que no
habría faltado agregar más. No puede haber
elementos en uno que no estén en el otro
& viceversa

Segundo: Hablemos sobre lo wle Do Prioridad Mox

La primera ~~esta~~ comp debo ser la sumatoria
de todos los cíts donde ese paper se
encuentra en lo 2^{do} coord del Conjunto cíts.
Además, podríamos pedir que a la primera comp.
sea mayor estricto que el. ✓ ~~recomiendo~~

c) ~~pedir absrcto citar científicas + imp, el titánico (150)~~
Voy a asumir que el invRep está bien armado Y que
me garantizo que las relaciones estén OK!
Así que voy a reloacionar obs publicaciones → var autores
& obs cíts ↔ var cíts.

Pred absf: ~~ElasCientificosImpl & CitoCientificos~~ { FEDERICO HOERTT Hoya ②

Pred mismas Claves (f: OtrosCientificosImpl, c: OtrosCientificos) {
($\forall p: \text{Paper}$) ($p \in c.\text{publicaciones} \leftrightarrow p \in c.\text{autores}.data$) ✓

{

Pred largos OK (c: CitasCientificasImpl, c': CitasCientificas) {
($\forall p: \text{Paper}$) ($p \in c.\text{publicaciones} \rightarrow |c.\text{publicaciones}[j] = p.\text{autores}.data[p]$) ✓

{

~~Pred mismos Elementos (c: OtrosCientificosImpl, c': OtrosCientificos)~~

Pred mismos Elementos (c: OtrosCientificosImpl, c': OtrosCientificos) {
($\forall p: \text{Paper}$) ($\forall j: \mathbb{N}$) ($\forall e: \text{publicaciones}_n$) ($e[j] \in c.\text{publicaciones}[p]$)

{

Pred mismos Citos (c: CitasCientificas, c': CitasCientificasImpl) {

($\forall p_1, p_2: \text{Paper}$) ($\langle p_1, p_2 \rangle \in c.\text{citos-elems} \leftrightarrow ($
 $p_1 \in c.\text{citos} \wedge \cancel{p_2 \in c'.\text{citos}[p_1]} \quad /$
)) }

pred abs(c:CitasCientificosImpl, c':citasCientificos) {
 mismos ~~Claves~~ Claves(c, c') \wedge_L lorgasOR(c, c') \wedge_L
 mismos Elementos(c, c') \wedge mismos Otas(c, c')
}



~~Elementos~~

3) Nos dicen que los materiales son strong y de largo alcance: no depende de ningún parám.
 En el contexto de éste ej, voy a considerar que definir y obtener de $\text{DictDigital}[\text{Materia}, \dots]$ son $O(1)$ (considero objetos acotados) /

MateriaData es Struct<Inscriptos; ConjuntoHoj<Alumno>;
 en Espacio: ColaSobrehasta<Alumnos>> ✓

Alumno es int X faltan la capacidad.

Materia es int

Modulo Inscripciones implementa InscripcionesECI {

Var materias: DiccionarioDigital<

> Materia, ~~Materia~~ MateriaData

Proc DarseDeBaja(inout i: Inscripciones, in m: Materia,
 in a: Alumno)

Var materiaData: MateriaData.

materiaData := i.materias.obtener(m) // $O(1)$

~~i.materias~~

materiaData.inscriptos.sacar(a) // $O(\log |Q_m|)$

... ~

if (!materiaData.entsperaRatio()) then $O(1)$
materiaData.inscriptosDefinirRatio() // $O(\log(Qm))$
materiaData.antsperaDesencolar()
)
endif

Complexidad final: ~~$O(\log(Qm))$~~

$$O(\log(Qm) + \log(Qm)) = O(2\log(Qm))$$
$$2\log(Qm) \in O(\log(Qm))$$

Complejidad final: $O(\log(Qm))$

FEDERICO HYERTA → semi-completos Hoja ① ③

3 c) Que sea un arbol perfectamente balanceado,
además como se completan de izq a der
es muy fácil y práctico ver si un nodo (índice)
es hoja o no.

Si tenemos un índice, ~~sabemos~~ que
pertenece a un nivel y que el siguiente nivel tiene
el doble de nodos, y... al mult. por 2 nos
"paramos" en el nodo anterior al izq del
sig. nivel. Esto matemáticamente funciona.
Análogamente para ir a la derecha dir. por
2 para subir el nivel.

¿ Para ir a derecha?

4a) ^{únicos} y ^{peor} ^{caso} ~~mejor~~ ^{Hasta} ~~se~~ ^{que} ~~que~~ ^{de} ~~que~~ ^θ ~~θ~~

ser $O(n)$ (por ende $\Omega(n)$), son aquellos donde conocemos información extra y podemos evitar hacer comparaciones entre elementos (ej: RadixSort con dígitos ocultos 0-9 y un largo maxi de díj.)

Todos los demás algoritmos deben realizar comparaciones entre elementos, y como vimos en la teoría, por el árbol de decisiones, tienen un peor caso no mejor que $\Omega(n \log n)$, algunos superaron esa cota, ej: insertion $O(n^2)$ en el peor caso. Otros tienen $O(n \log n)$ como mergeSort, pero no mejor que eso.

[✓] (no menciona de dónde sale el $n \log n$)

FEDERICO
HOERTH

b) Los rozos son string de largo oculto
(y ~~tamb~~ tamb considero alfabeto oculto).

Como expliqué en el ej. de estructuras, utilizaré
Dict Digital en $O(1)$ ✓

Lo idea del algoritmo es separar los "perros,"
de los otros rozos ($O(n)$). ✓

Luego ordenar los 2 conjuntos.

Contador es struct <rozo: Rozo, cant: int>

Proc OrdenarPerros(in A: Array<Nombre, Rozo, Edad, Peso>)

Var mejorRozo: ArregloTupla<Nombre, Edad, Peso>

Var otrosRozos: ArregloTupla<Rozo, int>. \rightarrow ¿qué tamaño?

Ver cantPorRozo Dict: DiccionarioDigital<~~dict<Rozo, int>~~>

Roza, Contador ✓

Var mejoresPerros: ListaEnlazada<Tupla<Nombre, Edad, Peso>>

Ver centPorRozaLista: ListaEnlazada<Contador>

~~contPorRozos~~

contPorRoza = diccionarioVacio()

contPorRoza Dict := diccionarioVacio() // $O(1)$

contPorRozaLista := listaVacia() // $O(1)$

~~contPerros~~

mejoresPerros := listaVacia() // $O(1)$

Var i: int

i := 0

while ($y < A.length$) do // $O(n)$, Haga ⑧ ④
 if ($A[y][1] == "Perro"$) then // $O(1)$
 if (! (contPorRaza.Diccionario.esta ($A[y][1]$))) then // $O(1)$
~~contPorRaza.Diccionario.agregar (A[y][1], 1)~~
 var contador: Contador // $O(1)$
 contador := new Contador (roza = $A[y][1]$, cont = 1)
 contPorRaza.Diccionario.definirRapido ($A[y][1]$, contador) // $O(1)$
contPorRaza.listo.agregarAtras (contador) // $O(1)$
 endif
 Vkl contador: Contador // $O(1)$
 $O(P) \leftarrow$ contador := contPorRaza.Diccionario.obtener ($A[y][1]$)
 contador.cant := contador.cant + 1 // $O(1)$
 else
~~separar~~ ~~mejoresPerros~~ ~~otrosPerros~~ \rightarrow ~~mejoresPerros~~ \rightarrow $O(1)$
~~mejoresPerros~~ \rightarrow ~~otrosPerros~~ \rightarrow $O(1)$
 endif
 $y := y + 1$ // $O(1)$
 end While

~~mejoresPerros := new Array (<Contador>) (contPorRaza.listo.obtener (1))~~
~~mejoresPerros := new Array (<Contador>) (contPorRaza.listo.obtener (1))~~
~~otrosPerros := new Array (<Contador>) (contPorRaza.listo.obtener (2))~~

FEDERICO
HUERTH

~~Los otros que~~

var otrosRazasArr: Array<Contador> // deberia ir el diccionario
desde acá.

~~Razas~~

$O(n)$

otrosRazasArr := toArray(KantPorRazaHist)

// Ahora necesito ordenar esta lista en $O(n)$

// Podemos aplicar doble radix sabiendo que

// la edad está rotada y/o rezo tamb

→ Hay q' ordenar por cantidad de apariciones, no por edad.

radixSort(otrasRazasArr) // Ordenar por desempeño:
✓ // Orden alfabetica (raza)
 $O(n)$

radixSort(otrasRazasArr) // Ordenar por cant

x

// de mayor a menor
 $O(n)$ $O(n \cdot m^n)$

~~Vamos a usar~~

Var mejoresPerrosArr: Array<Tupla<Nombre, Edad, Peso>

mejoresPerrosArr := toArray(mejoresPerros) // $O(p)$

mergeSort(mejoresPerrosArr) // Ordenar por desempeño

Coord Peso Asc.

$O(p \cdot \log(p))$

mergeSort(mejoresPerrosArr) // ordenar por
edad asc (estable)
// $O(p \log(p))$

[Ahora sólo me guarda los perros de las razas que
quiero otrasRazasRes: = new Array(otrasRazasArr.length)]

var otrasRazasRes: Array<TuplaRaza, int>
otrasRazasRes: = new Array(otrasRazasArr.length)
// $O(n)$

j := 0

while (j < otrasRazasArr) do // $O(n)$

otraRazasRes[j] := <otrasRazasArr[j].raza,
otrasRazasArr[j].cant> // $O(1)$

j := j + 1 // $O(1)$ → no entiendo qué cambia

end While

return mejoresPerrosArr, otrasRazasRes
/* complejidad final: $O(n * p \log(p))$

Análisis Complejidad final

$$O(n + n + n + n + p + p \log(p) + p \log(p)) \\ + h + \cancel{p}n = O(6n + p + p \log(p))$$

~~h~~, $6n \in O(n)$

$$p \in O(p \log(p))$$

Complejidad final: $O(n + p \log(p))$