

Corrigido: Ramón

# Algoritmos y Estructuras de Datos

## Segundo Parcial - Sábado 25 de Noviembre de 2023

#Orden	Libreta	Apellido y Nombre	E1	E2	E3	Nota Final
83	1088122	Fuentes Urquiza Pedro	40	28	25	93

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, además de los apuntes de la cátedra.
- Cada ejercicio debe entregarse en hojas separadas.
- Incluir en cada hoja el número de orden asignado, número de libreta, número de hoja, apellido y nombre.
- El parcial se aprueba con 60 puntos. Para promocionar es necesario tener al menos 70 y ningún ejercicio con 0 puntos (en ambos parciales).

### E1. Elección de estructuras (40 pts)

Se quiere implementar el TAD BIBLIOTECA que modela una biblioteca con su colección de libros. Por el momento la biblioteca cuenta con una sola estantería, dentro de la cual cada libro ocupa una posición. La biblioteca cuenta con un registro de socios que pueden retirar y devolver libros en cualquier momento. Por restricciones del sistema que se utiliza, un socio no puede registrarse con un nombre de más de 50 caracteres.

Cuando la biblioteca adquiere un nuevo libro o cuando un libro es devuelto, éste es insertado en el primer espacio libre de la estantería. Es decir, si los lugares ocupados son 1, 2, 3, 4 y se presta el libro en la posición 2, al agregar un nuevo libro al catálogo éste será ubicado en la posición 2. Cuando el libro que estaba originalmente en la posición 2 sea devuelto, será ubicado en la primera posición libre, que será la 5.

Dadas las siguientes operaciones y de acuerdo a las complejidades temporales de peor caso indicadas, donde  $L$  es la cantidad de libros en la colección,  $r$  es la cantidad de libros que el socio en cuestión tiene retirados y  $k$  la cantidad de posiciones libres en la estantería, respectivamente:

- proc AgregarLibroAlCatálogo(inout b: Biblioteca, in l: idLibro)
   
Requiere: {l no pertenece a la colección de libros de b}
   
Descripción: la biblioteca adquiere un nuevo libro, lo suma a su catálogo y lo pone en la estantería en el primer espacio disponible.
   
Complejidad:  $O(\log(k) + \log(L))$
- proc PedirLibro(inout b: Biblioteca, in l: idLibro, in s: Socio)
   
Requiere: {s es socio de la biblioteca y el libro l no está entre los libros prestados}
   
Descripción: el socio pasa a retirar un libro que se retira de la estantería y se acumula en sus libros prestados.
   
Complejidad:  $O(\log(r) + \log(k) + \log(L))$
- proc DevolverLibro(inout b: Biblioteca, in l: idLibro, in s: Socio)
   
Requiere: {s es socio de la biblioteca y el libro l está entre sus libros prestados}
   
Descripción: el socio pasa a devolver un libro que previamente había tomado prestado. Vuelve a la estantería en el primer espacio disponible.
   
Complejidad:  $O(\log(r) + \log(k) + \log(L))$
- proc Prestados(in b: Biblioteca, in s: Socio): Conjunto<Libro>
   
Requiere: {s es socio de la biblioteca}
   
Descripción: este procedimiento retorna los libros que el socio tomó prestados de la biblioteca y aún no devolvió.
   
Complejidad:  $O(1)$
- proc UbicaciónDeLibro(in b: Biblioteca, in l: idLibro): Posicion
   
Requiere: {l pertenece a la colección de libros de b y no está prestado}
   
Descripción: obtiene la posición del libro en la estantería.
   
Complejidad:  $O(\log(L))$

Se pide:

- Plantea la estructura de representación del módulo BibliotecaImpl, que provea las operaciones mencionadas. Se debe explicar detalladamente qué información se guarda en cada parte, las relaciones entre ellas, y cómo se aseguraría que la información registrada es consistente.
- Justificar cómo se cumplen las complejidades pedidas con esta estructura por cada operación. Indicar suposiciones sobre la implementación de las estructuras usadas y aclaraciones sobre aliasing.
- Escribir los algoritmos de AgregarLibroAlCatálogo y DevolverLibro, justificando detalladamente que se cumplen las cotas de complejidad requeridas.

## E2. Invariante de representación y función de abstracción (30 pts)

Tenemos un TAD que modela las ventas minoristas de un comercio. Cada venta es individual (una unidad de un producto) y se quieren registrar todas las ventas. El TAD tiene un único observador:

```

TAD Comercio {
    obs ventasPorProducto: dict<Producto, seq<tupla<Fecha, Monto>>>
}

Producto es string
Monto es int
Fecha es int (segundos desde 1/1/1970)

```

**ventasPorProducto** contiene, para cada producto, una secuencia con todas las ventas que se hicieron de ese producto. Para cada venta, se registra la fecha y el precio. Se puede considerar que todas las fechas son diferentes. Este TAD lo vamos a implementar con la siguiente estructura:

```

Modulo ComercioImpl implementa Comercio {
    var ventas: Secuencia<tupla<Producto, Fecha, Monto>>
    var totalPorProducto: Diccionario<Producto, Monto>
    var ultimoPrecio: Diccionario<Producto, Monto>
}

```

- **ventas** es una secuencia con todas las ventas realizadas, indicando producto, fecha y monto.
- **totalPorProducto** asocia cada producto con el dinero total obtenido por todas sus ventas.
- **ultimoPrecio** asocia cada producto con el monto de su última venta registrada.

Se pide:

- Escribir en forma coloquial y detallada el invariante de representación y la función de abstracción.
- Escribir ambos en el lenguaje de especificación.

## E3. Sorting (30 pts)

Se nos pide ayudar a un herborista que quiere poder organizar sus ingredientes para determinar qué hierbas le conviene recolectar. Para ello cuenta con su propio inventario. Como no es una persona muy organizada, puede tener distintas hierbas del mismo tipo en distintas alacenas o cofres. Luego de realizar una inspección de su lugar de trabajo, nos entrega una secuencia de  $n$  tuplas que constan de una hierba, identificada por su nombre, y la cantidad que se encontró. El nombre de cada hierba tiene como máximo 100 caracteres, de acuerdo al estándar de la Organización Mundial de Herboristas. El herborista cuenta a su vez con su libro de creaciones, que le permite saber en cuántas recetas se utiliza cada hierba.

Se necesita saber cuáles son las hierbas que se usan en más creaciones y, en caso de empate, deberían aparecer primero aquellos de las que tiene menos reservas. La complejidad esperada en el *peor caso* es de  $O(n + h \log(h))$ , donde  $h$  es la cantidad de hierbas distintas con las que cuenta el herborista.

```
proc Recolectar(in s:Vector<tupla<string,int>>, in u:Diccionario<string,int>):Vector<string>
```

Ejemplo:

```

stock = [ ("Diente de León", 10), ("Menta", 4), ("Margarita", 13),
          ("Lavanda", 12), ("Diente de León", 5), ("Margarita", 6) ]

usos = {"Diente de León": 5, "Menta": 1, "Margarita": 3, "Lavanda": 5}

Recolectar(stock, usos) = ["Lavanda", "Diente de León", "Margarita", "Menta"]

```

Los primeros son la lavanda y el diente de león porque ambos tienen 5 usos, pero aparece primera la lavanda porque hay menos stock. En tercer lugar tenemos la margarita que tiene 3 usos. Finalmente, en último lugar está la menta, que tiene un solo uso.

- Se pide escribir el algoritmo de Recolectar. Justificar detalladamente la complejidad y escribir todas las suposiciones sobre las implementaciones de las estructuras usadas, entre otras.
- ¿Cuál sería el *mejor caso* para este problema? ¿Cuál sería la cota de complejidad más ajustada?

Pedro Fuentes Urquiza  
1088/22

Al. en Ondas de la Computación

Hojas 1

### 1) módulo Biblioteca Impl implementa Biblioteca {

var ~~préstados~~: Diccionario Digital < Socio, Conjunto~~log~~ < libro > ;  
var ~~posibles~~: Cola Prioridad < Personas > // la prioridad es creciente, o sea menor  $N - \text{Hog}$   
var ~~libros~~: Diccionario~~log~~ < libro, int > // con el mismo elemento en la primera posición.  
Personas

// Personas es int.

Explicación: En ~~préstados~~ se guarda, para cada socio, el conjunto de los libros que presta. Puedo prestarlo. En la implementación de Pedir libro se asegura que no tenga un libro en el diccionario libros y en préstamo a la vez. Uso un Diccionario Digital para devolver los libros prestados en  $O(1)$ . Para hacer ~~llenar~~ prioridad ~~esta~~ cumple con la complejidad. El conjunto se para haciendo ~~llenar~~.

- En ~~posibles~~ se encuentran las personas libro en una cola de Prioridad, para acceder a la mínima y sacarla en  $O(\log k)$ . También me permite agregar una persona en  $O(\log k)$ . Esta cola de Prioridad tiene el menor número como el de menor prioridad.
- En libro se encuentra el libro como clave y su posición como valor. De este modo, puedo buscar, encontrar, eliminar y obtener en  $O(\log l)$ .

### Explicación y justificación de complejidad por operación

- ✓) Agregar libro: Para agregar un libro, busco la mínima posición libro, me lo guardo y lo elimino de posibles. Esto lo hace en  $O(\log k)$  por la ColaPrioridad. Luego, lo agrego en mi diccionario~~log~~ libros ~~de~~ con el libro que me pasan como parámetro como clave y la primera posición libro como valor (lo que me guardé antes). Esto lo hace en  $O(\log l)$  por ~~de~~ agregar en mi Diccionario~~log~~.
- ✓) Pedir libro: Primero, agrego el libro a los libros prestados de los socios. Esto lo hace en  $O(\log R)$  pues accedo a una clave en un Diccionario Digital en  $O(1)$  en el caso, y agrego un elemento a un Conjunto~~log~~, en este caso, en  $O(\log R)$ . Luego, busco la posición del libro en mi Diccionario~~log~~. Esto toma  $O(\log l)$  ~~de~~ <sup>me to</sup> agrega.

lugar la él memo del Diccionario con  $O(\log L)$ . Por ultimo, agrego la posición del libro (la que me guardé) en la Cola Prioridad por libro. Esta me Toma  $O(\log L)$

) Dar libro : Primero, busco la muestra posición libro en pos libro, me la guardo, y la saco de la Cola Prioridad. Esto me Toma  $O(\log L)$  para sacar el libro en  $O(1)$  y liberarla (desencolar) en  $O(\log h)$ . Luego, si quiero devolver el libro de los invitados del resto. Accedo al diccionario Digital prioritario en  $O(1)$  y elimino el libro que me pasan como parámetro, en  $O(\log R)$ . Por ultimo, agrego la clave en  $(idLibro)$  en mi diccionario Log libros, y pongo como valor la posición libro que me guardé anterior. Esto ultimo lo hago en  $O(\log L)$ .

) Prioridad : Accedo a mi Diccionario Digital de invitados con  $O(1)$  la clave que me pasan como parámetro y devuelvo el valor de libro prioritaria, cuando llamo. Esto me Toma  $O(1)$  por el Diccionario Digital con claves ordenadas por su segundo llave.

) Proc UtilizarLibro : Busco el valor del id libro que me pasan como parámetro en mi Diccionario Log libros. Esto me Toma  $O(\log L)$ .

c) Proc AgregarLibroAlCatalogo (met b: Biblioteca Ingel, m l: idLibro) {  
 $O(1)$  n pos = b. posLibros. proximo () //Armo que proximo solo devuelva el valor y desencolarlo de la lista.  
 $O(\log h)$  b. posLibros. Desencolar ()  
 $O(\log L)$  b. libros. definir (libro: l, pos)  
}

Proc DevolverLibro (met b: Biblioteca Ingel, m l: idLibro, m s: \$000) {  
 $O(1)$  pos = b. posLibros. proximo ()  
 $O(\log h)$  b. posLibros. Desencolar ()  
 $O(\log L)$  b. definir (l, pos) b. libro. definir (l, pos)  
 $O(\log R)$  b. ~~definir~~ b. prioritario. libro (s, b. prioritario. libro (s). eliminar (l))  
}

10/11/22

2) A continuación mire la explicación en castellano del Inventario de Representación y en su versión en lenguaje formal.

1) Todos los productos de Total Por Producto tienen como valor la suma total de las ventas de cada una de sus entradas en la memoria Ventas.

2) Todos los productos de Ultimo Precio tienen como valor el monto de la última venta (por lo tanto que la memoria

3) Todos los productos de Ultimo Precio tienen como valor el monto de la entrada (de ese mismo producto) con menor valor de fecha (con la misma entrada de fecha entre las entradas de ese producto).

4) Finalmente) Todas las ventas. Todos los productos de las ventas están en el diccionario de Total Por Producto, y su valor es la suma de ventas que tiene ese producto.

5) Vuelta) Todos los productos de ventas tienen una venta con mayor fecha. Ese producto estará en Ultimo Precio y su valor será el monto de aquella entrada de ese producto con mayor valor de fecha (entre las entradas de ese producto).

Nota: Usé los términos "entrada" y "venta" referiéndome a posiciones de la memoria, las triples.

Prod InvRep ( $d: \text{Comercio Ingel}$ ) {

( $\forall p: \text{Producto}$ ) ( $p \in d.\text{Total Por Producto} \Rightarrow d.\text{Total Por Producto}[p] = \text{SumaTotal}(d.\text{ventas}, p)$ )

( $\forall i \in \mathbb{Z})(0 \leq i \leq |d.\text{ventas}|) \Rightarrow d.\text{ventas}[i][0] \in d.\text{Total Por Producto} \wedge d.\text{Total Por Producto}[d.\text{ventas}[i][0]] = \text{SumaTotal}(d.\text{ventas}, d.\text{ventas}[i][0])$  ).

(nº 4)

- $\forall p: \text{Producto} \quad (p \in d.\text{UltimoPrecio}) \xrightarrow{\text{precio}} (j: \mathbb{N}) (0 \leq i < d.\text{ventas}) \wedge d.\text{ventas}[i][0] = p$   
 $\wedge (\forall j: \mathbb{N}) (0 \leq j < d.\text{ventas}) \wedge j \neq i \cancel{\Rightarrow} \cancel{(d.\text{ventas}[i][j] < d.\text{ventas}[j][i])} \wedge d.\text{ventas}[i][j] = d.\text{ultimoPrecio}[p] = d.\text{ultimoPrecio}[i]$   
 $\Rightarrow d.\text{ventas}[i][0] \Rightarrow d.\text{ventas}[j][i] < d.\text{ventas}[i][j]$   
 $(V_i: \mathbb{N}) (0 \leq i < d.\text{ventas}) \wedge (\forall j: \mathbb{N}) (0 \leq j < d.\text{ventas} | j \neq i) \wedge d.\text{ventas}[i][j] = d.\text{ventas}[j][i] \Rightarrow d.\text{ventas}[i][j] < d.\text{ventas}[j][i]$   
 $\Rightarrow d.\text{ventas}[i][0] \in d.\text{ultimoPrecio} \cap d.\text{ultimoPrecio} (d.\text{ventas}[i][0]) = d.\text{ventas}[i][0]$

### Función de Abstracción

La idea es que ~~Todos los~~ Todas las ~~productos~~ Productos que están como clave en mi diccionario UltimosPrecios (o también TotalPorProductos o ~~Todos los~~ Productos DeVentas) están como clave en res.ventasPorProducto. Luego, si las tuplas que se encuentran en la memoria de ese producto estaban allí solamente porque el producto es el mismo que la clave y se están en la memoria de ventas.

FuncAbst (en d. ComercioIny)  $\frac{q}{\vdash \text{Comercio}}$

avanza  $(\forall T: \text{Tupla} \in \text{Productos}, \text{Fecha}, \text{Monto}) (T_0 \in d.\text{UltimoPrecio} \Rightarrow$

$T_0 \in \text{res.ventasPorProducto} \wedge (T_0 \in \text{res.ventasPorProducto}[T_0] \iff T_0 \in d.\text{ventas}))$

$\hookrightarrow$  Tendrás que hacer otra tukt

aux Polymorphic sumaTotal ( $\frac{ins}{reg}$  c.Tupla  $\in \text{Productos}, \text{Fecha}, \text{Monto} \rightarrow$   $\text{m. } f(\text{Fecha})$ ) {  $\frac{prod}{com s\&lo}$  Ficha y monto }

$$\sum_{i=0}^{10^4} (if (S[i][0] == p) then f(S[i][2]) else 0 fi)$$

3) Aclaraciones: Asume que el diccionario u os de la implementación DiccionarioDigital, ~~es~~ es debido a que las claves están rotadas por una constante, las operaciones que voy a utilizar serán  $O(1)$ . Ademá, iré a los vectores como si fueran Arreglos.

1) Idea: .) Cuanto el StockTotal de cada flor en un DiccionarioDigital, llamado Stock. Es  $O(n)$  pues el costo de desear y obtener un elemento es  $O(1)$ . Ademá, mantendré un contador h que cuente los elementos usados. Ademá, cuento los elementos

- Implementar un Arreglo de únicos en un contador h
- .) Implementar un Arreglo de triples < Stock, int, int> de tamaño h. Esto es  $O(h)$ .

.) Recorrer todas las claves de mi Diccionario Stock. A cada una de ellas le agrego una posición del arreglo. En esta posición guardare la clave del elemento actual, su valor, y en la tercera posición guardare el valor de la clave actual en el Diccionario. Todo esto cuesta  $O(h)$  pues acceder al array es  $O(1)$ , así como acceder a los valores de los diccionarios digitales.

.) Por último, ordeno este array con Merge Sort, una versión modificada. Ordenaré por el valor del ~~3<sup>er</sup>~~ clave de la 3<sup>er</sup> posición de la Triple de manera decreciente, y en caso de empate, ordenar de manera creciente por el valor del ~~2<sup>do</sup>~~ <sup>2<sup>da</sup> posición de la triple.  $\Theta$ .</sup>

Proc Recolección (in s: Vector<Triple<Stock, int>>, in u: Diccionario<Stock, int>):

Vector<Stock> {

n := s.length

h := 0

Stock := DiccionarioDigital.DiccionarioValor()

for (int i = 0; i < n; i++) {

if (Stock.raja(s[i][0])) {

Stock.raja(s[i][0], Stock.dime(s[i][0]) + 1)

$O(n)$

} else {

Stock.raja(s[i][0], 1)

h++}

}

resTriples := new Array< Tuple<String, int, int> ()

slus Agregader : 70.

for each val in Jack {

*Tryphos* (*clausa*)  $\rightarrow$  Tryphos (val), *T. obscurus* (val), *T. obscurus* (vd) } 0(1)  
*clausa* + +

Merge Sort Heapsort (recTriples). //In comparamento con applicazione antecedente. (b) (b)

```
res := new Array<String>(width)
```

```
for (int j=0; j<h; j++) {
```

$$\text{res}[j] = \text{restrictives}[j][0]$$

$\mathcal{O}(n)$

o(h)

return our

7

⑧ Finalmente, como solo tienen el nombre de cada herbo, nos tiene sobre el array ya ordenado y pongo en otro Array de Strings cada posición en el mismo orden que estaba ordenado en el array original, pero solo la primera clase. Además, es necesario ordenar que el algoritmo Merge Sort Modificado es  $O(h \log h)$  para ordenar un array de  $h$  elementos.

Finalmente, la complejidad total del algoritmo es  $O(h \log h + n)$ .

b) El mejor caso para este problema sera un array de  $n$  enteros repetidos, de este modo Tarea  $O(h \log h)$  puede que no. No habrá mejor caso que el peor caso Merge Sort tiene una complejidad constante de  $\log n$ , siendo  $n$  la tamañada.

7) b) Para pensar el mejor caso, puedo ver que sucede con mi dispositivo con ambos extremos, con  $h=1$  y con  $h=n$ .

→ Mejor caso no consiste en fijar una variable a una constante. Con  $h=1$ , el costo total sería de  $O(n)$ , puesto que se recorrerá el arreglo original para sumar la cantidad de herbas. Luego, todas las operaciones dependen exclusivamente de  $h$ ; Armar el Array con Triples, Recorrer el diccionario stock, Merge Sort, armar el array con  $n$  iteraciones sobre los Triples.  $N h=1$ . Todo esto es una constante, teniendo que el costo total sea  $O(n)$ .

Luego, con  $h=n$  queda todo lo contrario. Recorrer el array inicial sigue siendo  $O(n)$ , pero el resto de operaciones toman más tiempo, haciendo que la complejidad sea de  $O(n \log n)$  en total.

↳ Si, pero también lo faltó

① Armar el Array con Triples, es visto como  $O(h \cdot \log h)$ , faltó  $\log h$   
② Merge Sort es lo que me eleva la complejidad, ya que toma  $O(n \log n)$  ! Pase de ~~de~~

Luego, el mejor caso para este problema es un arreglo que tenga solo una herba repetida  $n$  veces. Revisar si esto

Una cosa de complejidad exacta para el mejor caso sería de  $\Theta(n)$ , ya que independientemente al arreglo siempre se recorrerá en su totalidad y el resto de las operaciones quedan como constantes.