Algoritmos y Estructuras de Datos II

Práctica 5 – Diseño: Elección de estructuras de datos

Notas preliminares

- En esta práctica se utilizan los términos abreviados *Rep* y *Abs* para referirse al invariante de representación y a la función de abstracción, respectivamente.
- Por "diseñar" se entiende el hecho de elegir una estructura de representación adecuada, esquematizar Rep, Abs y (en pseudocódigo) los algoritmos más importantes y justificar que se cumplen los requerimientos pedidos.

Ejercicio 1

Considere el TAD Multiconjunto(Nat).

1. Diséñelo utilizando listas enlazadas de naturales. Calcule el orden de complejidad (de peor caso) de las operaciones de inserción, borrado y cálculo de la cantidad de repeticiones de un elemento.[1]
2. Piense en otras estructuras –más allá de las listas– y estime los órdenes de complejidad de peor caso que tendrían las operaciones mencionadas.

Ejercicio 2 ★

Escriba algoritmos para las operaciones $\cap$ y $\cup$, considerando las siguientes representaciones para conjuntos:

1. Sobre listas enlazadas (ordenadas y no ordenadas).
2. Sobre árboles binarios de búsqueda.
3. Sobre *heaps*.

En cada inciso, calcule los órdenes de complejidad de peor caso de las operaciones.

Ejercicio 3

Confeccione una tabla comparativa de las complejidades de peor caso de las operaciones de pertenencia, inserción, borrado, búsqueda del mínimo y borrado del mínimo para conjuntos de naturales sobre las siguientes estructuras:

1. Lista enlazada.
2. Lista enlazada ordenada.
3. Árbol binarios de búsqueda.
4. Árbol AVL.
5. *heap*.
6. *trie* (usar la expresión binaria del natural).

Ejercicio 4 ★

El TAD Matriz infinita de booleanos tiene las siguientes operaciones:

- *Crear*, que crea una matriz donde todos los valores son falsos.
- *Asignar*, que toma una matriz, dos naturales (fila y columna) y un booleano, y asigna a este último en esa coordenada. (Como la matriz es infinita, no hay restricciones sobre la magnitud de fila y columna.)
- *Ver*, que dadas una matriz, una fila y una columna devuelve el valor de esa coordenada. (Idem.)
- *Complementar*, que invierte todos los valores de la matriz.
- *Intersecar*, que toma dos matrices y realiza la conjunción coordenada a coordenada.

[1] Note que cuando se expresa el orden de complejidad, se hace en base a cierto n que representa el "tamaño" de la estructura en algún sentido. ¿Qué representa n en los incisos a) y b)?

Diseñe este TAD de modo que las operaciones *Crear*, *Ver* y *Complementar* tomen $O(1)$ tiempo en peor caso.

Ejercicio 5 ★

Una matriz finita posee las siguientes operaciones:

- *Crear*, con la cantidad de filas y columnas que albergará la matriz.
- *Definir*, que permite definir el valor para una posición válida.
- *#Filas*, que retorna la cantidad de filas de la matriz.
- *#Columnas*, que retorna la cantidad de columnas de la matriz.
- *Obtener*, que devuelve el valor de una posición válida de la matriz (si nunca se definió la matriz en la posición solicitada devuelve cero).
- *SumarMatrices*, que permite sumar dos matrices de iguales dimensiones.

Diseñe un módulo para el TAD Matriz finita de modo tal que dadas dos matrices finitas A y B,

∗ *Definir* y *Obtener* aplicadas a A se realicen cada una en $\Theta(n)$ en peor caso, y
∗ *SumarMatrices* aplicada a A y B se realice en $\Theta(n+m)$ en peor caso,

donde n y m son la cantidad de elementos no nulos de A y B, respectivamente.

Ejercicio 6

Considere el TAD Diccionario con historia, cuya especificación es la siguiente:

TAD Diccionario con historia(Clave, Significado)

parámetros formales
géneros κ, σ

$\bullet = \bullet : \kappa \times \kappa \longrightarrow$ bool (relación de equivalencia)

igualdad observacional

$$(\forall d, d' : \text{diccHist}(\kappa, \sigma)) \left(d =_{\text{obs}} d' \iff \left(\begin{array}{l} (\forall k : \kappa)(\text{Definido?}(k,d) =_{\text{obs}} \text{Definido?}(k,d') \wedge_{\text{L}} \\ (\text{Definido?}(k,d) \Rightarrow_{\text{L}} | \\ \quad \text{Significado}(k,d) =_{\text{obs}} \text{Significado}(k,d') \wedge \\ \quad \text{BorrarSignificado}(k,d) =_{\text{obs}} \text{BorrarSignificado}(k,d'))) \end{array} \right) \right)$$

observadores básicos

Definido?	$: \kappa \times \text{diccHist}$	$\longrightarrow$ bool	
Significado	$: \kappa\ c \times \text{diccHist}\ dh$	$\longrightarrow \sigma$	$\{\text{Definido?}(c, dh)\}$
BorrarSignificado	$: \kappa\ c \times \text{diccHist}\ dh$	$\longrightarrow$ diccHist	$\{\text{Definido?}(c, dh)\}$

generadores

Vacío	$:$	$\longrightarrow$ diccHist
Definir	$: \kappa \times \sigma \times \text{diccHist}$	$\longrightarrow$ diccHist

otras operaciones

CantSignificados	$: \kappa\ c \times \text{diccHist}\ dh$	$\longrightarrow$ nat	$\{\text{Definido?}(c, dh)\}$

axiomas $\forall dh$: diccHist$\forall c, c'$: $\kappa \forall s$: σ

Definido?(c, Vacío) $\equiv$ false
Definido?(c, Definir(c', s, dh)) $\equiv$ $(c = c') \vee$ Definido?(c, dh)

Significado(c, Definir(c', s, dh)) $\equiv$ **if** $c = c'$ **then** s **else** Significado(c, dh) **fi**

BorrarSignificado(c, Definir(c', s, dh)) $\equiv$ **if** $c = c'$ **then**
dh
else
Definir(c', s, BorrarSignificado(c, dh))
fi

CantSignificados(c, Definir(c',s,dh)) $\equiv$ **if** $c = c'$ **then** 1 **else** 0 **fi** +
if Definido?(c, dh) **then** CantSignificados(c, dh) **else** 0 **fi**

Fin TAD

Se debe diseñar este TAD respetando los siguientes órdenes de ejecución en el peor caso:

- Definido? $O(n)$
- Vacío $O(1)$
- Significado $O(n)$
- Definir $O(n)$
- BorrarSignificado $O(n)$
- CantSignificados $O(n)$

donde n es la cantidad de claves que están definidas en el diccionario.

Ejercicio 7

1. Diseñe el TAD Árbol binario(α), teniendo en cuenta que las operaciones *nil*, *bin*, *nil?*, *izq*, *der* y *raíz* deben tener orden de complejidad $O(1)$ en peor caso.
2. Especifique la precondición, postcondición y el aliasing de las operaciones mencionadas.
3. Dar algoritmos para resolver las siguientes operaciones, y calcular su orden de ejecución en peor caso:
   a) NivelCompleto? : nat $n \times$ ab(α) $a \longrightarrow$ bool $\{n \leq \mathrm{h}(a)\}$
   b) Completo? : ab(α) $\longrightarrow$ bool

Ejercicio 8

Diseñe el TAD Cola de prioridad, escribiendo los algoritmos e indicando sus órdenes de complejidad de peor caso sobre las siguientes estructuras:

1. Árbol binario con invariante de *heap*.
2. Arreglo con invariante de *heap*.

Ejercicio 9 ★

Diseñar un conjunto de naturales que satisfaga los siguientes requerimientos de complejidad temporal:

- Ag debe costar tiempo $O(1)$ si el elemento a agregar es menor que el mínimo elemento del conjunto hasta el momento o mayor que el máximo, y tiempo lineal en la cantidad de elementos del conjunto en cualquier otro caso.
- $\cup$ y $\cap$ deben tomar tiempo $O(1)$ si cada elemento de uno de los conjuntos pasados como parámetro es mayor que todos los del otro, y tiempo $O(n+m)$ en cualquier otro caso, siendo n y m las cardinalidades de los dos conjuntos pasados como parámetros.

1. Elegir una estructura adecuada, justificando la decisión, y escribir los algoritmos para las tres operaciones anteriores. Documentar claramente los aspectos de aliasing.
2. ¿Puede usarse este mismo diseño para conjuntos que no sean de naturales? ¿Qué condición debe cumplir el tipo al que pertenecen los elementos para poder usar este módulo?

Ejercicio 10 ★

Diseñar las relaciones binarias entre números naturales. Para una relación $\mathcal{R}$, se quiere preguntar:

- Dado un número a, cuáles son los números b tales que $a\mathcal{R}b$.
- Dado un número a, cuáles son los números b tales que $b\mathcal{R}a$.

Las relaciones se construyen por extensión, es decir, agregando y borrando parejas de la relación.

1. Suponiendo que las relaciones son entre números de 0 a 100, elegir estructuras de datos adecuadas. Justificar. Dar el invariante de representación y la función de abstracción de las estructuras. Implementar las operaciones sobre las estructuras elegidas. Indicar el orden de complejidad de las operaciones.
2. Elegir las estructuras de datos adecuadas si el rango sobre el cual se una relación será fijado al momento de la creación de la misma. Justificar.

3. Elegir las estructuras de datos adecuadas si las relaciones son entre naturales cualesquiera (i.e., no hay un rango prefijado), de manera que ninguna operación tenga un costo demasiado alto. Justificar.

Ejercicio 11 ★

Se desea diseñar un sistema de estadísticas para la cantidad de personas que ingresan a un banco. Al final del día, un empleado del banco ingresa en el sistema el total de ingresantes para ese día. Se desea saber, en cualquier intervalo de días, la cantidad total de personas que ingresaron al banco. La siguiente es una especificación del problema.

TAD IngresosAlBanco

observadores básicos
totDias : iab $\longrightarrow$ nat
cantPersonas : iab i × nat d × nat h $\longrightarrow$ nat $\{1 \leq d \wedge d \leq h \wedge h \leq totDias(i)\}$

generadores
Comenzar : $\longrightarrow$ iab
TerminaDia : iab × nat $\longrightarrow$ iab

axiomas ...
totDias(Comenzar) $\equiv$ 0
totDias(TerminaDia(i, n)) $\equiv$ 1 + totDias(i)
cantPersonas(TerminaDia(i, n),d, h) $\equiv$ **if** totDias(i) $< h$ **then** n **else** 0 **fi** + **if** totDias(i) $< d$ **then**
0
else
cantPersonas(i, d, mín(h,totDias(i)))
fi

Fin TAD

1. Dar una estructura de representación que permita que la función *cantPersonas* tome $O(1)$.
2. Calcular cuánta memoria usa la estructura, en función de la cantidad de días que pasaron n.
3. Si el cálculo del punto anterior fue una función que no es $O(n)$, piense otra estructura que permita resolver el problema utilizando $O(n)$ memoria.
4. Agregue al diseño del punto anterior una operación *mediana* que devuelva el último (mayor) día d tal que cantPersonas($i, 1, d$) $\leq$ cantPersonas($i, d+1$,totDias(i)), restringiendo la operación a los casos donde dicho día existe. Si este ítem no sale, se recomienda dejarlo pendiente hasta avanzada la práctica de dividir y conquistar. No olvidar retomarlo luego.

Ejercicio 12

El servicio meteorológico ha decidido generar un sistema que registre el nivel de smog instante a instante, durante un año. Cuenta con diversos empleados que van ingresando las distintas mediciones. Se desea que el ingreso de las mediciones sea muy eficiente. También se desea obtener el promedio mensual. Además, se está estudiando premiar al empleado más productivo de cada mes (el que más mediciones registre), por lo que es necesario conocerlo.

La especificación es la siguiente:

TAD Servicio meteorológico

generadores
CrearSM : conj(empleado) × timestamp $\longrightarrow$ sm
IngresarMedición : sm s × empleado e × timestamp t × nat n $\longrightarrow$ sm
$\{e \in \text{Empleados}(s) \wedge \text{Mes}(t) = \text{MesActual}(s) \wedge \neg\text{HayValor?}(s, t)\}$
CerrarMes : sm $\longrightarrow$ sm

observadores básicos
MesActual : sm $\longrightarrow$ mes
Empleados : sm $\longrightarrow$ conj(empleado)

HayValor? : sm × timestamp ⟶ bool
Valor : sm s × timestamp t ⟶ nat {HayValor?(s,t)}
#MedicionesDelEmpleado : sm s × empleado e × mes m ⟶ nat {$e \in$ Empleados(s)}

otras operaciones
PromedioMensual : sm s × mes m ⟶ nat
{$\sum_{e \in \text{Empleados}(s)}$ #MedidicionesDelEmpleado$(s,e,m) > 0$}
MejorEmpleado : sm s × mes m ⟶ empleado
{$(\exists e$: empleado$)(e \in$ Empleados$(s) \wedge_{\text{L}}$ #MedicionesDelEmpleado$(s,e,m) > 0$}

axiomas ...

Fin TAD

Empleado ES String[20], Mes ES Nat (entre 1 y 12) y Timestamp ES Nat (con precisión arbitraria y proyectores para ver día, mes, etc.).

Diseñar un sistema eficiente, que responda a las siguientes complejidades temporales en peor caso:

- IngresarMedición: $O(1)$.
- Valor: $O(n \log(m' + n))$, siendo n la cantidad de mediciones ingresadas desde la última invocación a la función y m' la máxima cantidad de mediciones correspondientes a una misma hora (es decir, las 9 del día 1, las 10 del día 1, ..., las 9 del día 2, etc.),
- MejorEmpleado y PromedioMensual: $O(1)$.

Pista: no se olvide de *CerrarMes*.

Ejercicio 13 ★

Considere el siguiente TAD que describe la estructura de un texto:

TAD Texto

observadores básicos
cantPalabras : txt ⟶ nat
enPos : txt t × nat n ⟶ palabra {$1 \leq n \leq$ cantPalabras(t)}

generadores
enBlanco : ⟶ txt
agPalabra : txt × palabra ⟶ txt

otras operaciones
cambiarPalabra : txt × palabra × palabra ⟶ txt
posiciones : txt × palabra ⟶ conj(nat)
subtexto : txt t × nat ini × nat fin ⟶ txt {$1 \leq ini \leq fin \leq$ cantPalabras(t)}
masRepetidas : txt ⟶ conj(palabra) {cantPalabras$(t) > 0$}

otras operaciones (no exportadas)
masRepeticiones : txt ⟶ nat

axiomas
cantPalabras(enBlanco) ≡ 0
cantPalabras(agPalabra(t,p)) ≡ 1 + cantPalabras(t)

enPos(agPalabra(t,p),n) ≡ **if** cantPalabras$(t) = n-1$ **then** p **else** enPos(t,n) **fi**

cambiarPalabra(enBlanco, p, p') ≡ enBlanco
cambiarPalabra(agPalabra(t,p''), p, p') ≡ agPalabra(cambiarPalabra(t,p,p'),
if $p = p''$ **then** p' **else** p'' **fi**)

posiciones(enBlanco, p') ≡ $\emptyset$
posiciones(agPalabra(t,p),p') ≡ **if** $p = p'$ **then**
Ag(cantPalabras(t)+1, posiciones(t,p')))
else
posiciones(t,p')
fi

```
subtexto(t, i, f) ≡ if i = f then
                        agPalabra(enPos(t, i), enBlanco)
                    else
                        agPalabra(enPos(t, f), subtexto(t, i, f − 1))
                    fi

masRepetidas(agPalabra(t,p)) ≡ if #posiciones(t, p) = masRepeticiones(t) then
                                   Ag(p, ∅)
                               else
                                   masRepetidas(t) ∪
                                   if #posiciones(t, p) + 1 = masRepeticiones(t) then
                                       Ag(p, ∅)
                                   else
                                       ∅
                                   fi
                               fi

masRepeticiones(t) ≡ if ∅?(masRepetidas(t)) then
                         0
                     else
                         #posiciones(t, dameUno(masRepetidas(t)))
                     fi
Fin TAD
```

Se desea diseñar el módulo correspondiente al TAD texto. En particular, asumimos que trabajaremos sólo con textos en español, y por lo tanto podemos dar una cota para la longitud de la palabra más larga que puede aparecer en el texto.

Se requiere que las operaciones que se listan a continuación cumplan con la complejidad temporal indicada:

- *subtexto(in inicio:nat, in fin:nat, in t:txt)* $\rightarrow$ *txt*

  Devuelve el texto correspondiente al fragmento de t que comienza en la posición *inicio* y finaliza en la posición fin.

  $O(fin - inicio)$ en el peor caso
- *cambiarPalabra(in anterior:palabra, in nueva:palabra, inout t:text)*

  Cambia todas las ocurrencias en el texto de la palabra anterior por la nueva.

  $O(k)$ en el peor caso, donde k es la cantidad de veces que se repite la palabra a cambiar.
- *palabrasMasRepetidas (in t:txt)* $\rightarrow$ *conj(palabras)* Devuelve el conjunto de palabras que más se repiten en el texto.

  $O(1)$ en el peor caso. Puede generar aliasing.

a) Describir la estructura a utilizar, documentando claramente cómo la misma resuelve el problema y cómo cumple con los requerimientos de eficiencia. El diseño debe incluir sólo la estructura de nivel superior. Para justificar los órdenes de complejidad, describa las estructuras soporte. **Importante**: si alguna de las estructuras utilizadas requiere que sus elementos posean una función especial (por ejemplo, comparación) deberá describirla.

b) Escribir una versión en lenguaje imperativo del algoritmo *cambiarPalabra*. Justifique la complejidad sobre el código.

Ejercicio 14 ★

Se desea diseñar un sistema para monitorear una planta industrial que cuenta con un conjunto de alarmas asociadas a distintos sensores. Cada sensor está asociado a una única alarma pero cada una de estas puede ser activada por distintos sensores. Una alarma está activa cuando la medición de al menos uno de sus sensores asociados supera un valor umbral definido para ese sensor. Los sensores y las alarmas se identifican con un código alfanumérico (para los sensores puede asumir que este es de longitud acotada, mientras que para las alarmas no)

El siguiente TAD es una especificación para este problema.

TAD PLANTA

observadores básicos

esAlarma : planta × alarma ⟶ bool
esSensor : planta × sensor ⟶ bool
alarmaSensor : planta p × sensor s ⟶ alarma {esSensor(p,s)}
umbral : planta p × sensor s ⟶ nat {esSensor(p,s)}
medicion : planta p × sensor s ⟶ nat {esSensor(p,s)}

generadores

crear : conj(alarma) ⟶ planta
agSensor : planta p × sensor s × nat u × alarma a ⟶ planta
{¬ esSensor(p,s) ∧ esAlarma(p,a) ∧ $u > 0$}
nuevaMedicion : planta p × sensor s × nat ⟶ planta {esSensor(p,s)}

otras operaciones

encendidosPorAlarma : planta p × alarma a ⟶ conj(sensor) {esAlarma(p,a)}
encendida : planta p × alarma a ⟶ bool {esAlarma(p,a)}

axiomas

esAlarma(crear(c), a) ≡ $a \in c$
esAlarma(agSensor(p, s, u, a'),a) ≡ esAlarma(p, a)
esAlarma(nuevaMedicion(p, s, n),a) ≡ esAlarma(p, a)

esSensor(crear(c),s) ≡ false
esSensor(agSensor(p, s', u, a), s) ≡ $s = s' \vee$ esSensor(p, s)
esSensor(nuevaMedicion(p, s', n), s) ≡ esSensor(p, s)

alarmaSensor(agSensor(p, s', u, a), s) ≡ **if** $s = s'$ **then** a **else** alarmaSensor(p, a) **fi**
alarmaSensor(nuevaMedicion(p, s', n), s) ≡ alarmaSensor(p, s)

umbral(agSensor(p, s', u, a), s) ≡ **if** $s = s'$ **then** u **else** umbral(p, a) **fi**
umbral(nuevaMedicion(p, s', n), s) ≡ umbral(p, s)

medicion(agSensor(p, s', u, a), s) ≡ **if** $s = s'$ **then** 0 **else** medicion(p, a) **fi**
medicion(nuevaMedicion(p, s', n), s) ≡ **if** $s = s'$ **then** n **else** medicion(p, a) **fi**

encendidosPorAlarma(crear(c),a) ≡ $\emptyset$
encendidosPorAlarma(agSensor(p, s, u, a'), a) ≡ encendidosPorAlarma(p, a)
encendidosPorAlarma(nuevaMedicion(p, s, n), a) ≡ **if** alarmaSensor(p, s) = $a \wedge n >$ umbral(p, s) **then**
Ag(s, encendidosPorAlarma(p, a))
else
encendidosPorAlarma(p, a) − {s}
fi

encendida(p, a) ≡ #encendidosPorAlarma(p, a) > 0

Fin TAD

Se desea diseñar el sistema propuesto, teniendo en cuenta que la operación *nuevaMedicion* debe realizarse con complejidad temporal $O(1)$ y *encendida* con complejidad temporal $O(\ell a)$ en el peor caso, donde a es la cantidad de alarmas de la planta y ℓ es la longitud del identificador de alarma más largo.

Se pide:

a) Describir la estructura a utilizar, documentando claramente cómo la estructura resuelve el problema y cómo cumple con los requerimientos de eficiencia. El diseño debe incluir sólo la estructura de nivel superior. (De ser necesario para justificar los órdenes de complejidad, describa las estructuras soporte.)
b) Escribir una versión en lenguaje imperativo del algoritmo *nuevaMedición*, indicando la complejidad de cada uno de los pasos. Tener en cuenta que esta operación puede activar una alarma (cuando la medición supera el umbral) o desactivarla (cuando la medición pasa a ser menor que el umbral y no hay otros sensores activándola).

Ejercicio 15

Se desea diseñar un sistema para manejar el ranking de posiciones de un torneo deportivo. En el torneo hay un conjunto fijo de equipos que juegan partidos (posiblemente más de un partido entre cada pareja de equipos) y el ganador de cada partido consigue un punto. Para el ranking, se decidió que entre los equipos con igual cantidad de puntos no se desempata, sino que todos reciben la mejor posición posible para ese puntaje. Por ejemplo, si los puntajes son: A: 5 puntos, B: 5 puntos, C: 4 puntos, D: 3 puntos, E: 3 puntos, las posiciones son: A: 1ro, B: 1ro, C: 3ro, D: 4to, E: 4to.

El siguiente TAD es una especificación para este problema.

TAD EQUIPO es NAT.
TAD TORNEO

observadores básicos

equipos	: torneo	$\longrightarrow$	conj(equipo)	
puntos	: torneo t × equipo e	$\longrightarrow$	nat	$\{e \in \text{equipos}(t)\}$

generadores

nuevoTorneo	: conj(equipo) c	$\longrightarrow$	torneo	$\{\neg\emptyset?(c)\}$
regPartido	: torneo t × equipo g × equipo p	$\longrightarrow$	torneo	$\{g \in \text{equipos}(t) \land p \in \text{equipos}(t) \land g \neq p\}$

otras operaciones

pos	: torneo t × equipo e	$\longrightarrow$	nat	$\{e \in \text{equipos}(t)\}$
#masPuntos	: torneo t × conj(equipo) c × nat	$\longrightarrow$	nat	$\{c \subseteq \text{equipos}(t)\}$

axiomas

equipos(nuevoTorneo(c))	$\equiv$	c
equipos(regPartido(t, g, p))	$\equiv$	equipos(t)
puntos(nuevoTorneo(c),e)	$\equiv$	0
puntos(regPartido(t, g, p),e)	$\equiv$	puntos(t, e) + **if** $e = g$ **then** 1 **else** 0 **fi**
pos(t, e)	$\equiv$	1+#masPuntos(t,equipos(t),puntos(t, e))

#masPuntos(t, c, p)	$\equiv$	**if** $\emptyset?(c)$ **then**
		0
		else
		if puntos(t,dameUno(c)) $> p$ **then** 1 **else** 0 **fi** +
		#masPuntos(t,sinUno(c),p)
		fi

Fin TAD

Se desea diseñar el sistema propuesto, teniendo en cuenta que las operaciones *puntos*, *regPartido* y *pos* deben realizarse en $O(\log n)$, donde n es la cantidad de equipos registrados.

a) Describir la estructura a utilizar.

b) Escribir un pseudocódigo del algoritmo para las operaciones con requerimientos de complejidad.

Ejercicio 16

Se nos encargó el diseño del nuevo Sistema Unificado de Estadísticas para Colectivos y Omnibuses (SUECO). Dicho sistema recolecta los datos de todos los boletos pagados mediante el sistema electrónico y genera estadísticas de la cantidad de boletos expedidos y la cantidad de plata electrónica que ingresó al sistema en un determinado período. Al agregar un boleto al sistema se especifica su precio y además un natural que representa el momento en que fue sacado ese boleto. Dado que el sistema de pago electrónico no se sincroniza al instante, es posible que los boletos se agreguen al sistema no ordenados cronológicamente. Al consultar, siempre se da un intervalo de tiempo y se debe responder la cantidad de boletos expedidos o plata recaudada en el intervalo propuesto.

El siguiente TAD es una especificación para este problema.

TAD SUECO

géneros sueco

exporta sueco, generadores, observadores

igualdad observacional

$$(\forall s, s' : \text{sueco}) \left(s =_{\text{obs}} s' \iff \begin{pmatrix} (\forall d{:}\text{nat})(\forall h{:}\text{nat}) \\ \#\text{Intervalo}(s,d,h) =_{\text{obs}} \#\text{Intervalo}(s',d,h) \\ \text{plataIntervalo}(s,d,h) =_{\text{obs}} \text{plataIntervalo}(s',d,h) \end{pmatrix} \wedge \right)$$

observadores básicos

#Intervalo	: sueco × nat × nat	⟶	nat	
plataIntervalo	sueco × nat × nat	⟶	nat	

generadores

iniciarDia	:	⟶	sueco	
agregarBoleto	: sueco × nat × nat p	⟶	sueco	$\{0 < p\}$

axiomas

#Intervalo(iniciarDia,d,h)	≡	0
#Intervalo(agregarBoleto(s, t, p),d,h)	≡	#Intervalo(s, d, h) + **if** $d \leq t \wedge t < h$ **then** 1 **else** 0 **fi**
plataIntervalo(iniciarDia,d,h)	≡	0
plataIntervalo(agregarBoleto(s, t, p),d,h)	≡	plataIntervalo(p, d, h) + **if** $d \leq t \wedge t < h$ **then** p **else** 0 **fi**

Fin TAD

Se pide dar una estructura de representación que permita realizar la operación *agregarBoleto* en $O(n)$, y *#Intervalo* en $O(\log n)$, dónde n es la cantidad de boletos agregados hasta ese momento.

a) Describir la estructura a utilizar.

b) Para ambas operaciones, explicar como se usa la estructura de manera de cumplir con la complejidad pedida (no hace falta dar pseudocódigo de cada operación, pero se puede darlo si resulta conveniente para la explicación).

c) Agregar lo que sea necesario para proveer la operación *plataIntervalo* también en $O(\log n)$. Describa las modificaciones a la estructura, a las operaciones del punto anterior que precisen modificaciones, y la forma de implementar *plataIntervalo*.