# Algoritmos y Estructuras de Datos

## Primer Parcial – Martes 10 de octubre de 2023

| #Orden | Libreta | Apellido y Nombre | E1 | E2 | E3 | E4 | Nota Final |
|---|---|---|---|---|---|---|---|
| 22 | 88/23 | OTAZUA ARCE, MATEO | 20 | 27 | 20 | 30 | 97 |

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen
- 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. Especificación de problemas [20 pts]

Se desea especificar el problema *positivosAumentados* que dada una secuencia $s$ de enteros devuelve la secuencia pero con los valores positivos reemplazados por su valor multiplicado por el valor de la posición en que se encuentra. Si lo desea puede ayudarse escribiendo predicados y funciones auxiliares.

**Ejemplos**

- $positivosAumentados([0, 1, 2, 3, 4, 5]) = [0, 1, 4, 9, 16, 25]$
- $positivosAumentados([-2, -1, 5, 3, 0, -4, 7]) = [-2, -1, 10, 9, 0, -4, 42]$

## E2. Modelado con TADs [30 pts]

Queremos modelar el funcionamiento de un centro ecuestre. El centro puede comprar caballos de diferentes razas. Al comprar un caballo, el vendedor nos indica cuántos "puntos de vigor" posee el caballo.

Es posible aumentar el vigor de los caballos entrenándolos. Cada entrenamiento dura un cierto tiempo y el caballo gana un punto de vigor por cada hora entrenada.

Luego, a los caballos se los hace competir en el Hipódromo de Palermo. En cada competencia el caballo puede ganar o perder. Si gana, aumenta 10 puntos de vigor. Si pierde, por la deshonra de la derrota, pierde 10 puntos.

Se quiere conocer en todo momento cuántos puntos de vigor tiene cada caballo y cuál es el mejor caballo de cada raza, siendo el mejor aquel que tiene más vigor. Si en alguna raza hay más de un caballo con el vigor máximo, alcanza con conocer a cualquiera de ellos.

a) [10 pts] Indique las operaciones (procs) del TAD con todos sus parámetros.
b) [15 pts] Describa el TAD en forma completa, indicando sus observadores, los requiere y asegura de las operaciones. Puede agregar los predicados y funciones auxiliares que necesite, con su correspondiente definición
c) [5 pts] Supongamos ahora que, luego del primer entrenamiento, los entrenamientos sucesivos para un mismo caballo van perdiendo efectividad: si a un caballo se lo entrena muchas veces, con cada entrenamiento el caballo recibe un 10 % menos de puntos de vigor que en el entrenamiento anterior. ¿qué cambios debería realizar en su TAD? Descríbalos con palabras.

## E3. Precondición más débil [20 pts]

Dado el siguiente condicional determinar la precondición más débil que permite hacer valer la poscondición (Q) propuesta

**if** $s[i] \neq 2^i$ **then**
  $s[i] = 2 * s[i-1]$
**else**
  $s[0] = 1$;
**end**

$Q \equiv (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow_L s[j] = 2^j)$

## E4. Correctitud del ciclo [30 pts]

Dado el siguiente ciclo, su precondición ($P_c$) y su postcondición ($Q_c$),

$P_c \equiv \{s = S_0 \wedge i = 0 \wedge 0 \leq d < |s|\}$

**while** $i < d$ **do**
  $s[i] := e$;
  $i := i + 1$;
**end**

$Q_c \equiv \{(\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = e) \wedge (\forall j : \mathbb{Z})(d \leq j < |s| \rightarrow_L s[j] = S_0[j])\}$

a) [10 pts] Proponer un invariante ($I$) y una función variante ($f_v$) para el ciclo
b) [20 pts] Demostrar los siguientes pasos de la demostración de correctitud del ciclo
   i) [5 pts] $P_c \rightarrow I$
   ii) [10 pts] $(I \wedge \neg B) \rightarrow Q_c$
   iii) [5 pts] $(I \wedge f_v \leq 0) \rightarrow \neg B$

# Ejercicio 1

```
proc positivosAumentados (s: seq<ℤ>): seq<ℤ> {
    requiere {True}
    asegura {|res| = |s|}
    asegura {(∀i ∈ ℤ) (0 ≤ i < |s| ⟶L
            if (s[i] > 0) then
                res[i] = s[i] * i
            else
                res[i] = s[i]
            endif}
}
```

20

6

# Ejercicio 2 (a y b)

```
TAD Club {
    obs caballos : conj<Nombre>
    obs razas : dict<Nombre, Raza>
    obs puntos : dict<Nombre, Puntos>
    obs gimnasio : dict<Nombre, Minutos>
    obs gana(caballo: Nombre): bool   # solo la uso en competir()

    proc abrirClub (): Club {
        requiere {True}
        asegura {res.caballos = {}}
        asegura {res.razas = {}}
        asegura {res.puntos = {}}
        asegura {res.gimnasio = {}}
    }
    proc comprarCaballo (inout e: Club, in c: Nombre, in r: Raza, in p: Puntos) {
        requiere {c ∉ e.caballos}
        requiere {p ≥ 0}
        asegura {e.caballos = old(e).caballos + {c}}
        asegura {e.razas = setKey(old(e).razas, c, r)}
        asegura {e.puntos = setKey(old(e).puntos, c, p)}
        asegura {e.gimnasio = setKey(old(e).gimnasio, c, 0)}
    }
    proc entrenar (inout e: Club, in c: Nombre, in m: Minutos){
        requiere {c ∈ e.caballos}
        requiere {m > 0}
        asegura {e.gimnasio = setKey(old(e).gimnasio, c, (old(e).gimnasio[c] + m) mod 60)}
        asegura {e.puntos = setKey(old(e).puntos, c, old(e).puntos[c] + ((old(e).puntos[c] + m) div 60))}
        asegura {e.caballos = old(e).caballos}
        asegura {e.razas = old(e).razas}
    }
```

Nombre es string
Raza es string
Puntos es $\mathbb{Z}$
Minutos es $\mathbb{Z}$

¿PARA QUÉ USAR MINUTOS EN VEZ DE HORAS? ¡OJO ESTÁ BIEN.

...

e.gana(c) no se usa así.
Es un obs, vos tenés que decir cómo funciona (qué devuelve). Quién ganó es un parámetro de entrada

...

```
proc competir (inout e: Club, in c: Nombre){
    requiere {c ∈ e.caballos}
    asegura {e.caballos = old(e).caballos}
    asegura {e.razas = old(e).razas}
    asegura {e.gimnasio = old(e).gimnasio}
    asegura {if (e.gana(c)) then
                e.puntos = setKey(old(e).puntos, c, old(e).puntos[c] + 10)
             else
                e.puntos = setKey(old(e).puntos, c, max{0, old(e).puntos[c] - 10})
             endif}
}
proc puntosDeCaballo (in e: Club, in c: Nombre, out res: Puntos){
    requiere {c ∈ e.caballos}
    asegura {res = e.puntos[c]}
}
proc mejorDeRaza (in e: Club, in r: Raza, out res: Nombre){
    requiere {(∃c ∈ e.caballos)(e.razas[c] = r)}
    asegura {(∃c ∈ e.caballos)(e.razas[c] = r ∧
             (∀c2 ∈ e.caballos)(e.razas[c2] = r → e.puntos[c] ≥ e.puntos[c2]) ∧
             res = c)}
}
}
```

(c) Entiendo que la eficiencia del entrenamiento baja si son entrenamientos sucesivos, y, por lo tanto, solo importaría quién fue el último caballo entrenado y cuántas veces sucesivas entrenó. ~~Por esto, agregaría un observador ultEntrenado : struct<c: Nombre, veces: ℤ>~~. Para simplificar el proceso, agregaría un observador ultEntrenado : struct<c: Nombre, f: Eficiencia>. (Eficiencia es ℤ). Con cada entrenamiento se actualizaría con quien fue el último entrenado y qué eficiencia tuvo. Si el caballo a entrenar es distinto al ultEntrenado.c, su eficiencia es 100, pero si es igual a ultEntrenado.c, entonces esta vez su eficiencia será old(ultEncantado).f * 0,9, que equivale a restarle 10%.
Se multiplicaría por (ultEntrenado.f)/100 los puntos a asignar. También en abrirClub() habría que inicializarlo con algún nombre y en comprarCaballo() requerir que ese nombre no se use.

# Ejercicio 3

$S \equiv$ if $(s[i] \neq 2^i)$ then
$s[i] = 2 * s[i-1]$
else
$s[0] = 1$
endif

$Q \equiv (\forall j : \mathbb{Z})\ (0 \leq j < |s| \rightarrow_L s[j] = 2^j)$

Nos piden calcular $wp(S, Q)$.

Separamos y reescribimos $S$, en su guarda $(B)$ y sus casos $(S_T$ y $S_F)$.

$B \equiv s \neq setAt(s, i, 2^i) \equiv s[i] \neq 2^i$ # mejor expresarlo así

SE EXPRESA ASÍ, EL SETAT ES PARA LA INSTRUCCIÓN.

$S_T \equiv s = setAt(s, i, 2 * s[i-1])$

$S_F \equiv s = setAt(s, 0, 1)$

Como $S$ es un condicional, por axioma 4 sabemos que

$wp(S, Q) \equiv def(B) \wedge ((B \wedge wp(S_T, Q)) \vee (\neg B \wedge wp(S_F, Q)))$ ✓

Calculamos por separado sus partes.

$def(B) \equiv def(s) \wedge def(i) \wedge_{\natural} 0 \leq i < |s|$

Por axioma 1, de asignación, sabemos que

$wp(S_T, Q) \equiv Q^s_{setAt(s,i,2*s[i-1])} \wedge def(s) \wedge def(i) \wedge 0 < i < |s|$

LOS DEFS VAN ANTES Y CON UN $\wedge_L$

Reemplazamos en Q correspondientemente

$Q^s_{setAt(s,i,2*s[i-1])} \equiv (\forall j : \mathbb{Z})\ (0 \leq j < |s| \rightarrow_L setAt(s, i, 2 * s[i-1])[j] = 2^j)$

Separamos en casos $(i = j)$ y $(i \neq j)$

$Q^s_{setAt(s,i,2*s[i-1])} \equiv (\forall j : \mathbb{Z})\ ((0 \leq j < |s| \wedge i = j \rightarrow_L 2 * s[i-1] = 2^j) \wedge$
$(0 \leq j < |s| \wedge i \neq j \rightarrow_L s[j] = 2^j))$ ✓

Notemos que por $def(setAt(s, i, 2 * s[i-1]))$, $0 < i < |s|$, y por lo tanto, el caso $(i = j)$ no se indefine.
Notemos también que el caso $(i = j)$ está implicado por el caso $(i \neq j)$, entonces podemos quedarnos con $(i \neq j)$. ✓

$Q^s_{setAt(s,i,2*s[i-1])} \equiv (\forall j : \mathbb{Z})\ (0 \leq j < |s| \wedge i \neq j \rightarrow_L s[j] = 2^j)$

Ya tenemos todas las partes de $wp(S_T, Q)$. Calculamos ahora $wp(S_F, Q)$.

Sabemos por axioma 1, de asignación:

$wp(S_F, Q) \equiv Q^S_{setAt(s,0,1)} \wedge def(s) \wedge 0 < |s|$

Reemplazamos correspondientemente:

$Q^S_{setAt(s,0,1)} \equiv (\forall j : \mathbb{Z})(0 \leq j < |s| \rightarrow setAt(s,0,1)[j] = 2^j)$

Separamos en casos $(i = j)$ y $(i \neq j)$, siendo $(i = 0)$ en ambos casos

$Q^S_{setAt(s,0,1)} \equiv (\forall j : \mathbb{Z})((0 \leq j < |s| \wedge 0 = j \rightarrow 1 = 2^0) \wedge$
$(0 \leq j < |s| \wedge j \neq 0 \rightarrow s[j] = 2^j))$

Podemos ver que el caso $(j = 0)$ es una tautología, y por lo tanto, podemos obviarlo.

$Q^S_{setAt(s,0,1)} \equiv (\forall j : \mathbb{Z})(0 < j < |s| \rightarrow s[j] = 2^j)$

Ya tenemos todas las partes de $wp(S_F, Q)$.

Podemos ahora escribir $wp(S, Q)$ con todas sus partes.

$wp(S, Q) \equiv def(s) \wedge def(i) \wedge 0 \leq i < S \wedge$
$((s[i] \neq 2^i \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \wedge i \neq j \rightarrow_L s[j] = 2^j)) \wedge def(s) \wedge def(i) \wedge 0 < i < |s|) \vee$
$(s[i] = 2^i \wedge (\forall j : \mathbb{Z})(0 < j < |s| \rightarrow_L s[j] = 2^j) \wedge def(s) \wedge 0 < |s|))$

Podemos simplificar la expresión como

$wp(S, Q) \equiv def(s) \wedge def(i) \wedge 0 \leq i < S \wedge$
$((s[i] \neq 2^i \wedge (\forall j : \mathbb{Z})(0 \leq j < |s| \wedge i \neq j \rightarrow_L s[j] = 2^j) \wedge i \neq 0) \vee$
$(s[i] = 2^i \wedge (\forall j : \mathbb{Z})(0 < j |s| \rightarrow_L s[j] = 2^j)))$

# Ejercicio 4

$P_c \equiv s = S_0 \wedge i = 0 \wedge 0 \leq d < |s|$

C ≡ while (i < d) do → $B_c \equiv i < d$
  s[i] := e; → $C_1 \equiv s := setAt(s,i,e)$
  i := i+1 → $C_2 \equiv i := i+1$
endwhile

$Q_c \equiv (\forall j : \mathbb{Z})(0 \leq j < d \rightarrow_L s[j] = e) \wedge (\forall j : \mathbb{Z})(d \leq j < |s| \rightarrow_L s[j] = S_0[j]) \wedge |s| = |S_0|$

simplifico $Q_c \equiv (\forall j : \mathbb{Z})((0 \leq j < d \rightarrow_L s[j] = e) \wedge (d \leq j < |s| \rightarrow_L s[j] = S_0[j])) \wedge |s| = |S_0|$

Propongo invariante:

$I \equiv 0 \leq i \leq d \wedge 0 \leq d \leq |s| \wedge |s| = |S_0| \wedge$
$(\forall j : \mathbb{Z})((0 \leq j < i \rightarrow_L s[j] = e) \wedge (i \leq j < |s| \rightarrow_L s[j] = S_0[j]))$

Propongo función variante:

$fv \equiv d - i$

Nos piden probar: $P_c \rightarrow I$, $(I \wedge \neg B_c) \rightarrow Q_c$ y $(I \wedge fv \leq 0) \rightarrow \neg B$.

Probamos $P_c \rightarrow I$

$P_c \equiv s = S_0 \wedge i = 0 \wedge 0 \leq d < |s|$
$\Rightarrow$
$I \equiv 0 \leq i \leq d \wedge 0 \leq d < |s| \wedge |s| = |S_0| \wedge$
$(\forall j : \mathbb{Z})((0 \leq j < i \rightarrow_L s[j] = e) \wedge (i \leq j < |s| \rightarrow_L s[j] = S_0[j]))$

Veamos por partes:

* $i = 0 \Rightarrow 0 \leq i \leq d$
* $0 \leq d < |s| \Rightarrow 0 \leq d < |s|$
* $s = S_0 \Rightarrow |s| = |S_0|$
* $s = S_0 \wedge i = 0 \Rightarrow (\forall j : \mathbb{Z})((0 \leq j < i \rightarrow_L s[j] = e) \wedge (i \leq j < |s| \rightarrow_L s[j] = S_0[j]))$ ✱
  podemos notar que dentro del "para todo" al reemplazar i por 0 queda una tautología y el segundo queda $(0 \leq j < |s| \rightarrow_L s[j] = S_0[j])$ y esto se encuentra implicado por $(s = S_0)$.

6

Probamos $(I \wedge \neg B) \rightarrow Q_c$

Observemos primero $(I \wedge \neg B)$:

$$I \equiv 0 \leq i \leq d \wedge 0 \leq d < |s| \wedge |s| = |s_0| \wedge (\forall j : \mathbb{Z})((0 \leq j < i \rightarrow_L s[j] = e) \wedge (i \leq j < |s| \rightarrow_L s[j] = s_0[j]))$$

$$\wedge$$

$$\neg B_c \equiv i \geq d$$

Notemos que $0 \leq i \leq d \wedge i \geq d \equiv i = d$. ✓

Podemos entonces simplificar la expresión como:

$$I \wedge \neg B \equiv i = d \wedge 0 \leq d < |s| \wedge |s| = |s_0| \wedge (\forall j : \mathbb{Z})((0 \leq j < d \rightarrow_L s[j] = e) \wedge (d \leq j < |s| \rightarrow_L s[j] = s_0[j]))$$ ✓

Probemos ahora que esto implica $Q_c$.

$$Q_c \equiv (\forall j : \mathbb{Z})((0 \leq j < d \rightarrow_L s[j] = e) \wedge (d \leq j < |s| \rightarrow_L s[j] = s_0[j])) \wedge |s| = |s_0|$$

Se puede ver la implicancia a simple vista. ✓

Probamos $(I \wedge f_v \leq 0) \rightarrow \neg B$

Observemos primero $(I \wedge f_v \leq 0)$

$$I \equiv 0 \leq i \leq d \wedge 0 \leq d < |s| \wedge |s| = |s_0| \wedge (\forall j : \mathbb{Z})((0 \leq j < i \rightarrow_L s[j] = e) \wedge (i \leq j < |s| \rightarrow_L s[j] = s_0[j]))$$

$$\wedge$$

$$f_v \leq 0 \equiv d - i \leq 0$$

Notemos que $(d - i \leq 0) \equiv (d \leq i)$ y $(0 \leq i \leq d) \wedge (d \leq i) \equiv (i = d)$.

Vemos que no hay contradicción y con $(i = d)$ basta para implicar $(\neg B)$. ✓

$i = d \Rightarrow i \geq d$. ✓

BIEN!