Algoritmos y Estructuras de Datos II

Primer Parcial – Viernes 3 de abril de 2024

#Orden	Libreta	Apellido y Nombre	E1	E2	E3	E4	Nota Final
	363/23	Martínez, Fausto Nicolás	18	28	20	30	96

- Es posible tener una hoja (2 carillas), escrita a mano, con los anotaciones que se deseen, más los dos apuntes de TADs
- 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 70 puntos

E1. Especificación de problemas [20 pts]

Se llama números perfectos a aquellos naturales mayores a cero que son iguales a la suma de sus divisores positivos propios (divisores incluyendo al 1 y sin incluir al propio número). Se desea especificar el problema `reemplazarNúmerosPerfectos`, que dada una secuencia de enteros devuelve la secuencia pero con los valores que se corresponden con números perfectos reemplazados por el índice donde se encuentran.

Ejemplos

- $reemplazarNúmerosPerfectos([0,3,9,6,4,28,7]) = [0,3,9,3,4,5,7]$

Los únicos números reemplazados son el 6 y el 28 porque son los únicos números perfectos de la secuencia.

E2. TADs [30 pts]

Dada la siguiente especificación del TAD `Diccionario<K,V>`:

```
TAD Diccionario<K,V> {
    obs data: seq<tupla<K,V>>

    proc diccionarioVacío(): Diccionario<K,V>
        asegura {res.data = ⟨⟩}

    proc está(in d: Diccionario<K,V>, in k: K): bool
        asegura {res = true ↔ estaClave(d.data, k)}

    proc definir(inout d: Diccionario<K,V>, in k: K, in v: V)
        requiere {d = D0}
        asegura {estaClave(D0.data, k) → |d.data| = |D0.data|}
        asegura {¬estaClave(D0.data, k) → |d.data| = |D0.data| + 1}
        asegura {⟨k, v⟩ ∈ d.data}
        asegura {(∀i : ℤ)((0 ≤ i < |D0.data| ∧L D0.data[i]0 ≠ k) →L D0.data[i] ∈ d.data)}

    proc obtener(in d: Diccionario<K,V>, in k: K): V
        requiere {estaClave(d.data, k)}
        asegura {(∃i : ℤ)(0 ≤ i < |d.data| ∧L d.data[i]0 = k ∧ res = d.data[i]1)}

    pred estaClave(data: seq<tupla<K,V>>, k: K)
        {(∃i : ℤ)(0 ≤ i < |data| ∧L data[i]0 = k)}
}
```

a) ¿Es necesario escribir el predicado de igualdad? Si es necesario escríbalo. Si no, explique por qué.

b) Agregue a la especificación el procedimiento `obtenerODefinir`. Este procedimiento recibe un diccionario, una clave y un valor. Si la clave ya está definida en el diccionario, devuelve su valor asociado. En cambio, si la clave no está definida en el diccionario, la define con el valor pasado por parámetro y devuelve ese mismo valor.

c) ¿Cómo se podría modificar el TAD para que sea posible conocer la cantidad de veces que se consultó cada clave? Explique detalladamente en castellano qué cosas se agregarían y cuáles se modificarían.

E3. Precondición más débil [20 pts]

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

```
if (a mod 2 = 0)
    a := a * a
else
    a := -|a|
endif
```

$Q \equiv (\exists j : \mathbb{Z})(j \geq 0 \wedge j^2 = a)$

a) Describir en palabras la WP esperada
b) Derivarla formalmente a partir de los axiomas de precondición más débil. Para obtener el puntaje máximo deberá simplificarla lo más posible.

E4. Correctitud del ciclo [30 pts]

Dado el siguiente programa con su especificación

$P_c \equiv \{n > 0 \wedge n \ mod \ 2 = 0 \wedge i = 1 \wedge res = 1\}$

```
While(i <= n/2){
    res := res * i * (n+1-i);
    i := i+1;
}
```

$Q_c \equiv \{res = n!\}$

Contamos con el siguiente invariante, que sabemos que es incorrecto:
$I \equiv \{1 \leq i \leq n/2 + 1 \wedge res = \prod_{j=1}^{2(i-1)} j\}$

a) Señale qué axiomas del teorema del invariante no se cumplen. Justifique con palabras en forma precisa.
b) Escriba un invariante que resulte correcto.
c) Proponga una función variante y demuestre formalmente que es correcta.

U 363/23 | Fausto Martínez - 1° Parcial Algoritmos y Estructuras de Datos II | 3/5/2024 Hoja 1

1) aux sumaDivisores (n:ℤ) ℤ = $\sum_{i=0}^{n-1}$ if n mod i = 0 then i else 0 fi

pred esPerfecto (n:ℤ) {

n > 0 ∧ n = sumaDivisores(n)

}

proc reemplazarPerfectos (s: seq⟨ℤ⟩) seq⟨ℤ⟩ {

requiere {True}

asegura {(∀i:ℤ)(0≤i<|s| ∧ esPerfecto(s[i]) → res[i] = i}

asegura {(∀i:ℤ)(0≤i<|s| ∧ ¬esPerfecto(s[i]) → res[i] = s[i]}

asegura {|s| = |res|} (esto debería ir arriba para que se indeterminen los otros)

}

LU 363/23 | Fausto Martínez - 1° Parcial Algoritmos y Estructuras de Datos II | 3/5/2024 | Hoja 2

2) a) Es necesario escribir el predicado de igualdad, ya que, por default, la igualdad se da solo si coinciden todos los observadores. Y, usando el concepto clásico de diccionario, no hay un orden en los elementos del mismo, sin embargo las secuencias sí tienen orden y ese es el problema.

Para explicarlo, veamos el sig. ejemplo:

Quisieramos que {"Argentina": 1, "Francia": 2} y {"Francia": 2, "Argentina": 1} sean el mismo diccionario, pero sin cambiar la definición de la igualdad no lo lograríamos.

Por eso, defino el siguiente predicado de igualdad:

pred sonIguales (d_1: Diccionario<K,V>, d_2: Diccionario<K,V>) {
$((\forall i:\mathbb{Z})(0 \leq i < |d_1.data| \rightarrow_L d_1.data[i] \in d_2.data) \wedge$
$(\forall i:\mathbb{Z})(0 \leq i < |d_2.data| \rightarrow_L d_2.data[i] \in d_1.data))$
}

De esta manera, me aseguro de fijarme lo que me importa: que las secuencias tengan los mismos elementos, más allá del orden

b) proc obtener ODefinit (inout d: Diccionario<K,V>, k:K, v:V): V {
requiere {d = D_0}
asegura {estaClave(k, D_0) $\rightarrow$ $|d.data| = |D_0.data| \wedge$
$(\forall i:\mathbb{Z})(0 \leq i < |D_0.data| \rightarrow_L D_0.data[i] \in d.data) \wedge$
$(\exists i:\mathbb{Z})(0 \leq i < |d.data| \wedge_L d.data[i]_0 = K \wedge res = d.data[i]_1)$}
asegura {$\neg$ estaClave(k, D_0) $\rightarrow$ $|d.data| = |D_0.data| + 1$
$\wedge$ estaClave(k,d) $\wedge$ $\langle K,V \rangle \in d.data \wedge$
$(\forall i:\mathbb{Z})((0 \leq i < |D_0.data| \wedge_L D_0.data[i]_0 \neq k) \rightarrow_L D_0.data[i] \in d.data)$
$\wedge$ res = V}
}

c) En el caso en el que quisiera conocer la cantidad de veces que se "consultó" cada clave, implementaría el siguiente observador:

obs Contador: dict<K, Z> ✓

Luego, dentro de los procedimientos que yo considere que involucran "consultar" la clave (en este caso, serían todos los proc que reciben k como parámetro, es decir, esHet, definir, obtener y eventualmente obtenerO Definir)

Debería asegurar que, si es la primera vez que accedo a la clave, que la agrego a Contador y la inicio en 1

Y si es una vez posterior, (esto se deduciría de la naturaleza del proc) sumaría 1 al contador asociado a la clave. ✓

Siempre, por supuesto habría que aclarar que los Contadores asociados a las claves que no fueron consultadas se mantendrían en el mismo valor. ✓

Faltaría un pre para consultar el contador

LU 363/23 | Facundo Martínez - 1° Parcial Algoritmos y Estructuras de Datos II | 3/5/2024 Hoja 3

3) a) Si a es par, le asigno su cuadrado, así que a efectivamente será un cuadrado por default

Si a es impar, le asigno un número negativo, así que nunca será un cuadrado perfecto.

Todo parece indicar que la wp será a mod 2 = 0 ✓ Chequeemoslo.

b) wp(if ... endif, Q)

≡ def(a mod 2 = 0) ∧_L ((a mod 2 = 0 ∧ wp(a = a+a, Q) ∨ (a mod 2 = 1 ∧ wp(a = -|a|, Q)))

(≡ Axioma 4)

≡ (a mod 2 = 0 ∧ wp(a = a+a, Q)) ∨ (a mod 2 = 1 ∧ wp(a = -|a|, Q))

① ②

- ① a mod 2 = 0 ∧ wp(a = a+a, Q)

≡ a mod 2 = 0 ∧ def(a+a) ∧_L Q^a_{a+a}

(Ax 1) ≡ a mod 2 = 0 ∧ (∃j:ℤ)(j ≥ 0 ∧ j² = a²)

④

⊙ Observación: Este predicado es siempre True, pues tomando j = |a|, deducimos que existe j.

① ≡ a mod 2 = 0 //

- ② a mod 2 = 1 ∧ wp(a := -|a|, Q)

≡ a mod 2 = 1 ∧ def(-|a|) ∧_L Q^a_{-|a|}

(Ax 1) ≡ a mod 2 = 1 ∧ (∃j:ℤ)(j ≥ 0 ∧ j² = -|a|)

④

⊙ Obs: Este predicado es siempre False, pues -|a| es un número menor o igual que 0, y en particular menor que 0 pues a mod 2 = 1. Luego, j² nunca puede ser estrictamente menor que 0, sea cual sea el j, por lo que deducimos que no existe ningún j que cumple ⇒ ② ≡ False //

Calificación

Volviendo, teníamos ① ∨ ②

que es a mod 2 = 0 ∨ False

que reduce a: **a mod 2 = 0**

Weakest Precondition

Que es, efectivamente lo que sospechaba

∨

MS!

LU 363/23 | Fausto Martínez - 1° Parcial Algoritmos y Estructuras de Datos II | 3/5/2024 Hoja 4

4) a) Podemos ver fácilmente tanto

- $P_c \to I$ pues si $n>0 \wedge n \bmod 2 = 0 \Rightarrow n \geq 2$
$\Rightarrow \frac{n}{2}+1 \geq 2$

y como $i = 1$, entonces
$1 \leq i \leq 2 \leq n/2+1$ ✓

Aparte, como $res = 1 \wedge i = 1$
$\Rightarrow res = \prod_{j=1}^{2(i-1)} j = \prod_{j=1}^{2(1-1)} j$ ✓

- $I \wedge \neg B \to Q_c$
pues $I \wedge \neg B \Rightarrow i = n/2+1 \Rightarrow res = \prod_{j=1}^{2(\frac{n}{2}-1+1)} j = \prod_{j=1}^{n} j = n!$

Por descarte, la propiedad que no se cumple es
- $\{I \wedge B\}\ S\ \{I\}$

Veamos como se comporta el ciclo tomando como ejemplo $n=8$

$res = 1$

- 1° Iteración $i=1 \to res = 1*1*(9-1) \Rightarrow res = 8$
- 2° Iteración $i=2 \to res = 8*2*(9-2) \Rightarrow res = 8*2*7$

Acá ya podemos ver que el invariante no vale, pues el mismo nos dice que $res = \prod_{j=1}^{2(i-1)} j$ ✓
y acá se ve bastante directo que res no es eso. Sin embargo, sigamos, sospecho que me va a servir para el (6)

- 3° Iteración $i=3 \to res = (8*2*7)*3*(9-3) \to res = 8*7*6*3*2$
- 4° Iteración $i=4 \to res = (8*7*6*3*2)*4*(9-4) \to$
$res = 8!$ $\wedge$ $i=5$ //

Parece que va agarrando los números como en un sospechoso orden...

$8! = 8*7*6*5*4*3*2*1$ ✓

Por lo pronto, lo que puedo asegurar, es que el invariante que nos propusieron no se mantiene al final de cada iteración, y por tanto, no sirve $\{I \wedge B\}\ S\ \{I\}$

b) Basado en lo observado en el inciso anterior estoy tentado a elegir el invariante

$$I_2 \equiv 0 \leq i \leq \frac{n}{2}+1 \wedge res = \prod_{j=n+2-i}^{n} j * \prod_{j=2}^{i-1} j$$

Si me da el tiempo, intento probar que es correcto al terminar el inciso c) (no hace falta).

c) Busco una fv que se vaya reduciendo y cumpla

4) $\{I_2 \wedge B \wedge fv = V_0\}\ S\ \{fv < V_0\}$

5) $I_2 \wedge fv \leq 0 \longrightarrow \neg B$

Propongo $fv = \frac{n}{2} - i + 1$ ✓

Veamos que cumple ④ y ⑤ (voy a usar el invariante I_2 pero creo que no va a afectar mucho el resultado)

④ Para ver $\{I_2 \wedge B \wedge fv = V_0\}\ S\ \{fv < V_0\}$, veamos por Teorema, que $I_2 \wedge B \wedge fv = V_0 \Rightarrow \underbrace{wp(S, fv < V_0)}_{*}$

$* wp(res := res * i * (n+1-i);\ i := i+1,\ fv < V_0)$

Ax3 $\equiv wp(res := res * i * (n+1-i),\ wp(i := i+1,\ \frac{n}{2} - i < V_0))$

Ax1 $\equiv wp(res := res * i * (n+1-i);\ \frac{n}{2} - i < V_0)$

Ax1 $\equiv \frac{n}{2} - i < V_0$ ✓

Luego $I_2 \wedge B \wedge fv = V_0 \Rightarrow fv = V_0 \equiv \frac{n}{2} - i + 1 = V_0$

$\Rightarrow V_0 - 1 < V_0 \Rightarrow \frac{n}{2} - i < V_0 \equiv wp(S, fv < V_0)$ ✓

$\therefore \{I_2 \wedge B \wedge fv = V_0\}\ S\ \{fv < V_0\}$

⑤ $I_2 \wedge fv \leq 0 \longrightarrow fv \leq 0 \Rightarrow \frac{n}{2} - i + 1 \leq 0$ ✓

$\Rightarrow i \geq \frac{n}{2} + 1 \Leftrightarrow i > \frac{n}{2} \equiv \neg B$

$\therefore I_2 \wedge fv \leq 0 \Rightarrow \neg B$, luego el ciclo termina y fv es correcto.

LV 363/23 | Fausto Martínez - 1º Parcial Algoritmos y Estructuras de Datos II | 3/5/2024 Hoja 5

Bien, ahora intentaré demostrar que I_2 es un invariante correcto (no por cancherear si no para chequear si está bien o no)

Quiero ver que ① $P_c \Rightarrow I_2$

② $I_2 \wedge B \Rightarrow wp(S, I_2)$

③ $I_2 \wedge \neg B \Rightarrow Q_c$

① Como $n>0 \wedge m \bmod 2 = 0 \Rightarrow n \geq 2$
$\Rightarrow \frac{n}{2} + 1 \geq 2$.

luego como $i=1 \Rightarrow 1 \leq i \leq 2 \leq \frac{n}{2}+1$ ✓

Como $res = 1 \wedge i = 1$

$\Rightarrow res = \prod_{j=n+2-1}^{n} j * \prod_{j=2}^{1} j$

$= \prod_{j=n+2-i}^{n} j * \prod_{j=2}^{i-1} j$ //

$\therefore P_c \Rightarrow I_2$

③ $I_2 \wedge \neg B \Rightarrow Q_c$

$0 \leq i \leq \frac{n}{2}+1 \wedge res = \prod_{j=n+2-i}^{n} j * \prod_{j=2}^{i-1} j \wedge i > n/2$

$\Rightarrow i = n/2+1 \wedge res = \prod_{j=n+2-i}^{n} j * \prod_{j=2}^{i-1} j$

$\Rightarrow res = \prod_{j=n+2-(n/2+1)}^{n} j * \prod_{j=2}^{n/2+1-1} j$

$= \prod_{j=n/2+1}^{n} j * \prod_{j=2}^{n/2} j = \prod_{j=2}^{n} j = n!$

$\therefore I_2 \wedge \neg B \Rightarrow Q_c$

Papel de fibra de caña de azúcar.

$\equiv wp(res := res + i*(n+1-i),\ 0 \le i+1 \le \frac{n}{2}+1 \wedge res = \prod_{j=n+1-i}^{n} j * \prod_{j=2}^{i} j)$

$\equiv \{0 \le i+1 \le \frac{n}{2}+1 \wedge res*i*(n+1-i) = \prod_{n+1-i}^{n} j * \prod_{j=2}^{i} j\}$

$\equiv \{0 \le i+1 \le \frac{n}{2}+1 \wedge res = \prod_{n+2-i}^{n} j * \prod_{j=2}^{i-1} j\}$

$I_2 \wedge B \equiv 0 \le i \le \frac{n}{2}+1 \wedge res = \prod_{n+2-i}^{n} j * \prod_{j=2}^{i} j \wedge i \le \frac{n}{2}$

$\Rightarrow i+1 \ge 0 \wedge i+1 \le \frac{n}{2}+1 \wedge res = \prod_{n+2-i}^{n} j * \prod_{j=2}^{i} j$

$\equiv wp(S, I_2)$

$\therefore \{I_2 \wedge B\}\ S\ \{I_2\}$

Luego, mi invariante I_2 es correcto :), Ah y el programa es correcto respecto a su especificación :)

NOTA