

*Arvo*  
**Introducción a la Programación  
Algoritmos y Estructuras de Datos I**

8,4

Parcial - 1er cuatrimestre 2023

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

1.1	1.2	2.1	2.2	3	4.1	4.2
B-	B-	B	R+	B	R-	B

Apellido ..... *Fuentes Urdapilleta*..... Nombre ..... *Pedro*.....  
 LU .. ..... Turno ..... *Tarde*.....  
 Cant. de hojas entregadas (sin contar ésta) ..... *5*.....

El parcial se aprueba con 5 puntos. Entregar cada ejercicio en hoja separada. No se permite consultar ningún material durante el examen.

### Ejercicio 1. 2 puntos

#### 1. [1 punto]

Completar en las siguientes especificaciones nombres adecuados para el problema *a*, los parámetros *b* y *c*, y las etiquetas *x*, *y*, *z*, *u* y *w*.

```
problema a (in b: seq(Char × Char), in c: seq(Char)) : seq(Char) {
    requiere x: { (Vi, j : Z)((0 ≤ i < |b| ∧ 0 ≤ j < |b| ∧ i ≠ j) → b[i]0 ≠ b[j]0}
    requiere y: { (Vi, j : Z)((0 ≤ i < |b| ∧ 0 ≤ j < |b| ∧ i ≠ j) → b[i]1 ≠ b[j]1}
    requiere z: { (Vi : Z)(0 ≤ i < |c| → (Ej : Z)(0 ≤ j < |b| ∧ b[j]0 = c[i]))}
    asegura u: { |resultado| = |c| }
    asegura w: { (Vi : Z)(0 ≤ i < |c| → (Ej : Z)(0 ≤ j < |b| ∧ b[j]0 = c[i] ∧ b[j]1 = resultado[i])) }
}
```

#### 2. [1 punto]

Especificar el siguiente problema (se puede especificar de manera formal o semi-formal):

Dados los inputs *b*: seq(Char × Char), *m*: seq(seq(Char)) y *n*: seq(seq(Char)), retornar verdadero si *n* es igual al resultado de aplicar el problema *a* (del punto 1.1) a cada elemento de la secuencia *m*.

### Ejercicio 2. 4 puntos

#### 1. [2 puntos]

Programar en Haskell una función que satisfaga la especificación del problema *a* del Ejercicio 1. Recordá escribir los tipos de los parámetros.

#### 2. [2 puntos]

Programar en Python una función que satisfaga la especificación del problema *a* del Ejercicio 1. Recordá escribir los tipos de los parámetros y variables que uses en tu implementación.

### Ejercicio 3. 2 puntos

Sea la siguiente especificación del problema aprobado y una posible implementación en lenguaje imperativo:

```
problema aprobado (in notas: seq(Z)) : Z {
    requiere: {notas > 0}
    requiere: {(Vi : Z)(0 ≤ i < |notas| → 0 ≤ notas[i] ≤ 10)}
    asegura: {result = 1 ↔ todos los elementos de notas son mayores o iguales a 4 y el promedio es mayor o igual a 7}
    asegura: {result = 2 ↔ todos los elementos de notas son mayores o iguales a 4 y el promedio está entre 4 (inclusive) y 7}
    asegura: {result = 3 ↔ alguno de los elementos de notas es menor a 4 o el promedio es menor a 4}
}
```

(Adro Fuentz, EJ 4) 1)

Urifey

R-

1.) Ni A es más fuerte que B. Significa que  $A \rightarrow B$  es una Tautología.  
Hojas. Luego, ni se cumple la especificación de  $p_1$  significa que A es True y C  
y True. Entonces, se cumpliría B es True para  $A \rightarrow B$  pero  $A \rightarrow B$  es una Tautología  
y A es True. Luego, C es True porque se cumple la especificación  
de  $p_1$ .

2.) Al revisar no funciona, pues no se cumple la especificación de  
 $p_2$ , que B y C son True. Ahora, para que se cumpla  $p_1$ , recuerda  
que A y C son True, y como puedes determinar el valor de B A en base  
al valor de B pues no sé si  $B \rightarrow A$  es Tautología.

Graduaria

2) Ni es posible que haya un Test suite con 100% de cubrimiento de  
casos que pase todos los Tests pero que igual el programa tenga un bug.  
Esto se debe a la naturaleza propia del Testing, uno no puede nunca saber  
si hay un valor que pueda hacer fallar ~~desarrollar~~ el programa (esta,  
que el resultado no sea el esperado acorde a la especificación) debido a  
que no se pueden probar todos los valores (a menos que el  
programa tenga una cantidad de inputs válidos infinita). Entonces,  
siempre puede haber un valor que haga fallar el programa. Ello es el  
Testing es una manera de acercarnos lo mejor posible de la muestra más  
eficiente, de acuerdo a Nuevos computacionales y Tiempo, al mayor  
cubrimiento de ~~los~~ posibles situaciones críticas que hacen fallar el  
programa.

B

entre Fuentes

Urfeld

EJ 3) 3) Busco un Test suite que al menos tenga un cubrimiento de al menos el 50% de las branches.

) Para ser más clara, numeraré las branches con Nro en mi CFG.

Haga 4

.) Para alcanzar un cubrimiento de al menos el 50% de las branches necesito cubrir al menos 4 branches. La ① y ② son irrelevantes pues siempre se ejecutan con cualquier input válido. Noté que con un solo test case puedo alcanzar a la ⑥ y a la ⑦.

Test #3: Promedio entre 4 y 7

notas = [5, 6] . Reservado = 2

Paradójico

Este Test alcanza las branches ①, ②, ⑥ y ⑦. De modo, un test suite con este único test case alcanza para cubrir el 50% de las branches.

.) El Test case cubre las branches ⑥ y ⑦ pues "entra y sale" del while.

(.) El test case cubre las la branch ① pues

;) Esto se debe a que no ejecuta la LS pues ninguna de las notas es  $\leq 4$ .

.) El Test case cubre las branches ② ⑥ pues "sale" del while y la suma de sus notas no es  $\geq 7 + \text{len(notas)}$

.) Luego, cubre la branch ⑦ pues la suma de sus notas es  $\geq 7 + \text{len(notas)}$ .

③ non relevantes para cualquier ejecución de notas, ya que no tiene notas menores a 4.

4) Hay dos errores en la implementación. El primero está en L9; donde dice `return 2` debería decir `return 1`. Este error es detectado por el Test #2, cuyo resultado esperado es 1 y el obtenido es 2.

El otro error está en L10; donde dice `if suma-notas > 4 * len(notas)`: debería decir `if suma-notas >= 4 + len(notas)`, esto se debe a que en el 2do asegura que  $\{result = 2 \Leftrightarrow$  todos los elementos de notas son mayores o iguales a 4 y el promedio está entre 4 (inclusive) y 7 $\}$ . Este error no afecta al test #4, cuyo resultado esperado es 2 y su resultado obtenido es 3.

⑤. Esto se debe a que en el primer asegura que  $\{result = 1 \Leftrightarrow \dots\}$ .

EJ3) 1)  
EXERCISE

Adro Fuentes

Arbolg

Hojas.

Paradójico

L1 SUMA\_NOTAS: INT = 0



L2 i: INT = 0



False

L3 while: < len(notas)



True ①

L4 if notas[i] < 4

True ②

is return 3

False ④ X

L6 suma\_notas = suma\_notas + notas[i]

L7 i := i + 1

L8 if suma\_notas  $\geq 7 * \text{len(notas)}$ :

True ⑤

L9 return 2

False ⑥

L10 if suma\_notas  $> 4 * \text{len(notas)}$ :

True ⑦

L11 Return 2

False ⑧ L12 return 3.

2) Busco un Test suite que ejecute Today las líneas del programa apropiado

.) Néque cualquier Test case de input valido ejecutara las líneas.

L1, L2, L3, L4, L6, L7, L8. Busco Test cases que ejecuten las restantes.

.) Test #1: Nota menor a 4

nota<sub>ay</sub> = [3]. Res esperado = 3.

.) Este Test case ejecuta L5.

.) Test #2: Promedio mayor o igual a 7 y que ninguna nota sea menor a 4.

nota<sub>ay</sub> = [7, 8]. Res esperado = 7.

.) Este Test case ejecuta L9.

.) Test #3: Promedio entre 4 y 7

nota<sub>ay</sub> = [5, 6]. Res esperado = 2.

.) Este Test case ejecuta L10, L11

.) Test #4: Promedio menor o igual a 4.

nota<sub>ay</sub> = [4]. Res esperado = 2

.) Este Test case ejecuta L10, L12

.) Aclaración: los numeros de linea son en base al CFG, no a la consigna.

$\text{list}[\text{tuple}[\text{char}, \text{char}]]$

Pedro Fuentes 2) b) defg devolver lista Snd ( $\text{listaTuplas} : \text{list}[\text{tuple}[\text{char}, \text{char}]]$ )  $\rightarrow$   $\text{list}[\text{char}]$ ,  
Urfeig  $\text{listaFnx} : \text{list}[\text{char}] \rightarrow \text{list}[\text{char}] =$

$\text{listaSnd} : \text{list}[\text{char}] = []$

for i in range (len (listaFnx)):

index : int = buscar En Primeros Elementos ( $\text{listaFnx}[i]$ ,  $\text{listaTuplas}$ )

$\text{listaSnd.append}(\text{listaTuplas}[\text{index}][1])$

return listaSnd

$\text{def buscarEnPrimerosElementos}(\text{char} : \text{char}, \text{l} : \text{list}[\text{tuple}[\text{char}, \text{char}]])) \rightarrow \text{int} :$

index, mP = 0 # El programa asume que hay un index que cumple.

for i in range (len(l)):

if l[i][0] == char:

index = i

break

return index.

2) a) devolver listaSnd ::  $[(\text{char}, \text{char})] \rightarrow [\text{char}] \rightarrow [\text{char}]$

devolver listaSnd  $\text{l}[] = []$

devolver listaSnd  $\text{l}(x: x_s)$  |,  $\text{len}(x: x_s) \geq 1 =$  : El mas Segundo Elemento (index, l) : devolver listaSnd  $\text{l}(x_s)$

otherwise []

where index = buscarEnPrimerosElementos (x, l)

$\text{len} : [\text{char}] \rightarrow \text{int}$

$\text{len}[] = 0$

$\text{len}(x: x_s) = 1 + \text{len}(x_s)$

$\text{aux Segundo Elemento} :: \text{Int} \rightarrow [(\text{Char}, \text{Char})] \rightarrow \text{Char}$

$\text{aux Segundo Elemento } i = \text{aux Segundo Elemento } i + 1$ .

$\text{aux Segundo Elemento} :: \text{Int} \rightarrow [(\text{Char}, \text{Char})] \rightarrow \text{Int} \rightarrow \text{Char}$

~~$\text{aux Segundo Elemento } i : [ ] k = k$~~

$\text{aux Segundo Elemento } i (x : x_s) k \quad | \quad i = k = \text{val } x$

$\text{otherwise} = \text{aux Segundo Elemento } i x_s (k+1)$ .

$\text{Buscar en Primeros Elementos} :: \text{Char} \rightarrow [(\text{Char}, \text{Char})] \rightarrow \text{Int}$

$\text{Buscar en Primeros Elementos} = \text{Buscar Primitivo } c \text{ l.o.}$

~~esta~~

$\text{Buscar Primitivo} :: \text{Char} \times [(\text{Char}, \text{Char})] \rightarrow \text{Int} \rightarrow \text{Int}$

$\text{Buscar Primitivo } c [ ] k = k$

$\text{Buscar Primitivo } c (x : x_s) k \quad | \quad c = \text{fst } x = k$

$\text{otherwise} = \text{Buscar Primitivo } c x_s (k+1)$

-- Aclaración: Todas las funciones arriba que van a encontrar lo que buscan

-- y que todo función de acuerdo a lo que requiere.

Este Fuentes  
Algoritmos

Hasta 1

1) a)

- .) Llamo al problema a "desolver ListaSnd"
- .) Llamo al parámetro b lista Tuplas
- .) Llamo al parámetro c lista FxT
- .) Llamo a la etiqueta x no Hay FxT Repetidos (lista Tuplas)
- .) Llamo a la etiqueta y no Hay And Repetidos (lista Tuplas)
- .) Llamo a lo siguiente z lista FxT Pertenece A lista Tuplas (lista Tuplas, lista FxT)
- .) Llamo a la etiqueta u misma longitud (n, c)
- .) Llamo a la etiqueta w res Es lista Snd en Mismo Orden que lista FxT (lista Snd)

b) problema es lista And de lista Tuplas (inc lista Tuplas, reg < char x char),

in lista FxT : reg < reg < char >,

in lista Snd : reg < reg < char > ) ; No es {

Requiere: lista FxT Pertenece A lista Tuplas

requiere { | lista FxT | = | lista Snd | }

arregla { res = True  $\leftarrow$  (Hj, Z) | 0  $\leq$  j  $\leq$  l }

① ( lista Tuplas, lista FxT, lista Snd )

② .) Llamo a la etiqueta w res Es lista Snd de lista Tuplas ( lista Tuplas )

③ .) Llamo a la etiqueta w : res Es lista Snd en Mismo Orden que lista FxT ( lista Tuplas, lista FxT ).

b) problema es lista Snd De Lstas Tuples (in b: NegChar <char>, in m: NegChar <char>, in n: NegChar <char>) : bool {

NegChar : lista Fox pertenece A lista Tuples (b, n)

requiere:  $\{ |n| = |m| \}$ .

segura:  $\{ res = \text{True} \Leftrightarrow (\forall i: \mathbb{Z}) (0 \leq i \leq |n| \rightarrow (\exists j: \mathbb{Z}) (0 \leq j \leq |m| \rightarrow$

son Iguales (devolver lista Snd (b, m[i]), n[j])) \}) \}.

o no son Iguales (in l1: NegChar, in l2: NegChar) {

res = True  $\Leftrightarrow (|l_1| = |l_2|) \wedge (\forall i: \mathbb{Z}) (0 \leq i \leq |l_1| \rightarrow l_1[i] = l_2[i]) \}.$

.) Aclaración: En el o no son iguales p.d q res sea en el mismo orden para las listas, pues en igualdad que tiene q ser por la expresación del j.  
En cambio, en el seguro del 1.b se no se da orden para los elementos de m y el resultado de aplicar la función a cada elemento de m.