Corregido por Jorge R. Manassi f. —

Introducción a la Programación
Algoritmos y Estructuras de Datos I

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

Nota: 10

Parcial - 1er cuatrimestre 2023

1.1	1.2	2.1	2.2	3	4.1	4.2
B	B	B	B	B	B	B

Apellido OTAZUA ARCE ..... **Nombre** MATEO .....

LU 88/23 ..... **Turno** TARDE .....

Cant. de hojas entregadas (sin contar ésta) 4 .....

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: $seq\langle Char \times Char\rangle$, in c: $seq\langle Char\rangle$) : $seq\langle Char\rangle$ {
   `requiere x:` $\{ (\forall i, j : \mathbb{Z})((0 \leq i < |b| \land 0 \leq j < |b| \land i \neq j) \rightarrow b[i]_0 \neq b[j]_0\}$
   `requiere y:` $\{ (\forall i, j : \mathbb{Z})((0 \leq i < |b| \land 0 \leq j < |b| \land i \neq j) \rightarrow b[i]_1 \neq b[j]_1\}$
   `requiere z:` $\{ (\forall i : \mathbb{Z})(0 \leq i < |c| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |b| \land b[j]_0 = c[i]))\}$
   `asegura u:` $\{ |resultado| = |c| \}$
   `asegura w:` $\{ (\forall i : \mathbb{Z})(0 \leq i < |c| \rightarrow (\exists j : \mathbb{Z})(0 \leq j < |b| \land b[j]_0 = c[i] \land b[j]_1 = resultado[i])) \}$
   }

2. [1 punto]

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

   Dados los inputs b: $seq\langle Char \times Char\rangle$, m: $seq\langle seq\langle Char\rangle\rangle$ y n: $seq\langle seq\langle Char\rangle\rangle$, 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: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ {
`requiere:` $\{|notas| > 0\}$
`requiere:` $\{(\forall i : \mathbb{Z})(0 \leq i < |notas| \rightarrow 0 \leq notas[i] \leq 10)\}$
`asegura:` $\{result = 1 \leftrightarrow$ todos los elementos de *notas* son mayores o iguales a 4 y el promedio es mayor o igual a 7$\}$
`asegura:` $\{result = 2 \leftrightarrow$ todos los elementos de *notas* son mayores o iguales a 4 y el promedio está entre 4 (inclusive) y 7$\}$
`asegura:` $\{result = 3 \leftrightarrow$ alguno de los elementos de *notas* es menor a 4 o el promedio es menor a 4$\}$
}

```
def aprobado(notas: list[int]) -> int:
L1:    suma_notas: int = 0
L2:    i: int = 0
L3:    while i < len(notas):
L4:        if notas[i] < 4:
L5:            return 3
L6:        suma_notas = suma_notas + notas[i]
L7:        i = i + 1
L8:    if suma_notas >= 7 * len(notas):
L9:        return 2
L10:   else:
L11:       if suma_notas > 4 * len(notas):
L12:           return 2
L13:       else:
L14:           return 3
```

1. Dar el diagrama de control de flujo (control-flow graph) del programa `aprobado`.
2. Escribir un test suite que ejecute todas las líneas del programa `aprobado`.
3. Escribir un test suite que tenga un cubrimiento de **al menos** el 50 por ciento de decisiones ("branches") del programa.
4. Explicar cuál/es es/son el/los error/es en la implementación. ¿Los test suites de los puntos anteriores detectan algún defecto en la implementación? De no ser así, modificarlos para que lo hagan.

Ejercicio 4. 2 puntos

1. [1 punto] Suponga las siguientes dos especificaciones de los problemas p1 y p2:

```
problema p1(x:Int)=res:Int {
  requiere A;
  asegura C;
  }
```

```
problema p2(x:Int)=res:Int {
  requiere B;
  asegura C;
  }
```

   Si A es más fuerte que B, ¿Es cierto que todo algoritmo que satisface la especificación p1 también satisface la especificación p2? ¿Y al revés?, es decir, ¿Es cierto que todo algoritmo que satisface la especificación p2 también satisface la especificación p1? Justifique.

2. [1 punto] ¿Es posible que haya un test suite con 100 % de cubrimiento de nodos que todos los test pasen pero que igual el programa tenga un bug? Justifique.

OTAZUA ARCE, MATEO (88/23)

Ejercicio 1

1. ✓

#"palabra" que puede tener números o símbolos
(nombre aprobado por Rubinstein)

problema encriptar (in clave: seq⟨Char × Char⟩, in palabra: seq⟨Char⟩) : seq⟨Char⟩ {

requiere sinRepetidosOrigenClave: {(∀i,j:ℤ)((0≤i<|clave| ∧ 0≤j<|clave| ∧ i≠j) → clave[i]$_0$ ≠ clave[j]$_0$)}

requiere sinRepetidosDestinoClave: {(∀i,j:ℤ)((0≤i<|clave| ∧ 0≤j<|clave| ∧ i≠j) → clave[i]$_1$ ≠ clave[j]$_1$)}

requiere todoDePalabraEnClave: {(∀i:ℤ)(0≤i<|palabra| → (∃j:ℤ)(0≤j<|clave| ∧ clave[j]$_0$ = palabra[i]))}

asegura mismoLargo: {|resultado| = |palabra|}

asegura cadaCaracterEncriptado: {(∀i:ℤ)(0≤i<|palabra| →
(∃j:ℤ)(0≤j<|clave| ∧ clave[j]$_0$ = palabra[i] ∧ clave[j]$_1$ = resultado[i]))}

}

2. ✓

problema sonPalabrasEncriptadasConClave (in clave: seq⟨Char × Char⟩, in palabras: seq⟨seq⟨Char⟩⟩, in palabrasEncriptadas: seq⟨seq⟨Char⟩⟩) : Bool {

requiere sinRepetidosOrigenClave: {(∀i,j:ℤ)((0≤i<|clave| ∧ 0≤j<|clave| ∧ i≠j) → clave[i]$_0$ ≠ clave[j]$_0$)}

requiere sinRepetidosDestinoClave: {(∀i,j:ℤ)((0≤i<|clave| ∧ 0≤j<|clave| ∧ i≠j) → clave[i]$_1$ ≠ clave[j]$_1$)}

requiere todoDePalabrasEnClave: {(∀k:ℤ)(0≤k<|palabras| →
(∀i:ℤ)(0≤i<|palabras[k]| → (∃j:ℤ)(0≤j<|clave| ∧ clave[j]$_0$ = palabras[k][i])))}

requiere mismoLargoListas: {|palabras| = |palabrasEncriptadas|}

asegura respuestaCorrecta: {resultado = True ↔ (∀i:ℤ)(0≤i<|palabras| → palabrasEncriptadas[i] = encriptar(palabras[i]))}

↑ el problema encriptar necesita también recibir una clave

Ejercicio 2

OTAZUA ARCE, MATEO (88123)

1.

```
encriptar :: [(Char,Char)] -> [Char] -> [Char]
encriptar clave [] = []
encriptar clave (char:chars) = (encriptarAux clave char) : (encriptar clave chars)

encriptarAux :: [(Char,Char)] -> Char -> Char
encriptarAux (clave:claves) char
    | origen == char  = destino
    | otherwise  = encriptarAux claves char
    where (origen, destino) = clave
```

2.

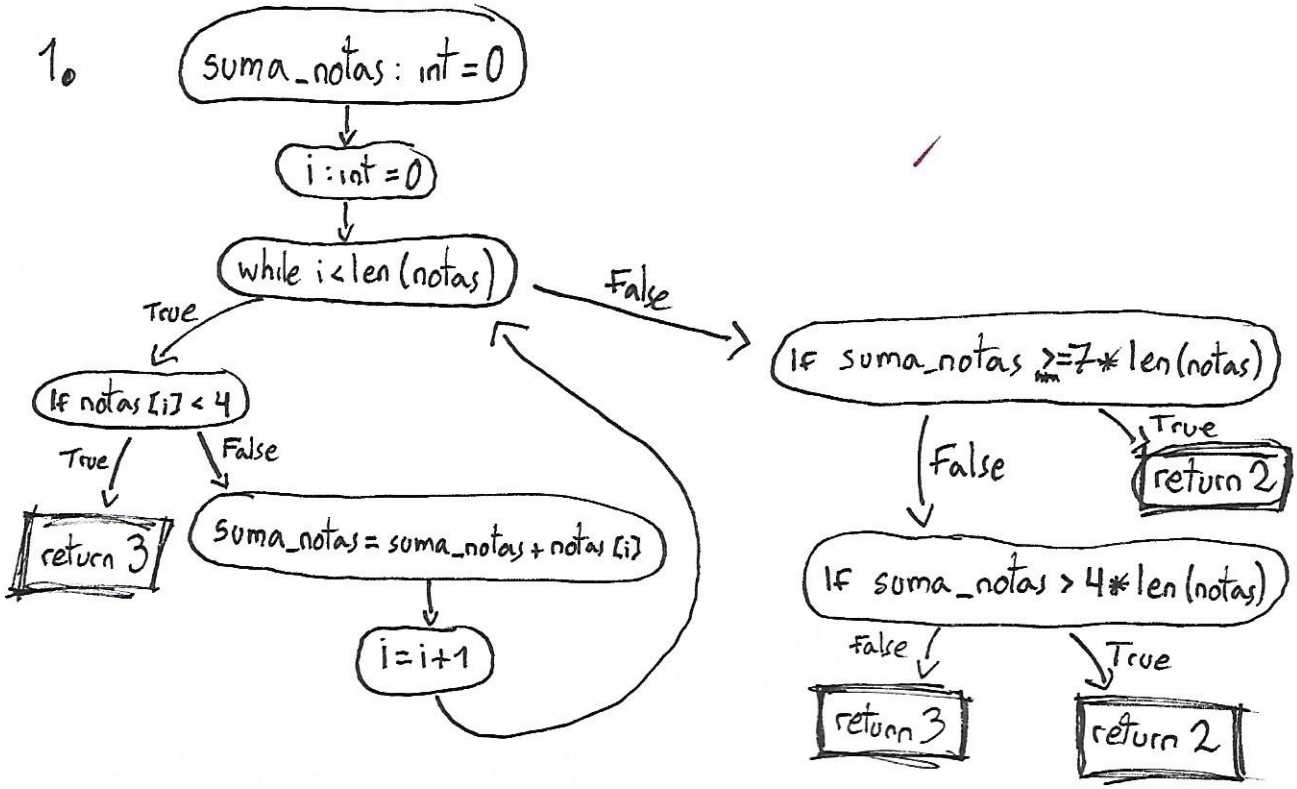
```
def encriptar(claves: [(str,str)], palabra: str) -> str:
    resAux: [str] = []
    for char in palabra:
        for clave in claves:
            if clave[0] == char:
                resAux.append(clave[1])

    res: str = resAux.join()
    return res
```

"" .join(resAux)

OTAZUA ARCE, MATEO (88/23)

Ejercicio 3

1.



2.

salidas esperadas

- caso notaMenorA4: [2] → 3
- caso notaMayorA7: [8] → 1
- caso notaEntre4y7: [5] → 2
- caso nota4: [4] → 2

3. ídem 2

- caso notaMenorA4: [2] → 3
- caso notaMayorA7: [8] → 1
- caso notaEntre4y7: [5] → 2
- caso nota4: [4] → 2

4. Hay dos errores en la implementación:

- Nunca devuelve 1, en su lugar devuelve 2
- Cuando el promedio es 4 debería devolver 2, pero devuelve 3.

Ambos test suites detectan estos defectos.

Ejercicio 4

OTAZUA ARCE, MATEO (88/23)

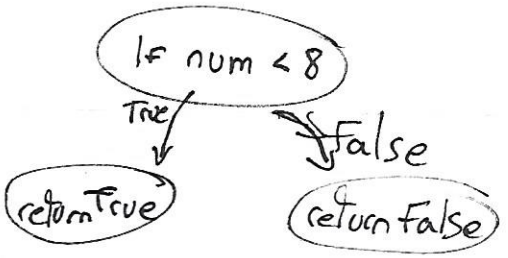
1. Que A sea más fuerte que B significa que $A \rightarrow B$, o sea, si A es True, B necesariamente es True también. Pero no siempre que B sea True, necesariamente A lo será.

~~Todo algoritmo que cumpla la especificación de p1 cumplirá por ende la especificación de p2. Pero no todo algoritmo que cumpla la de p2 cumplirá la de p1.~~

Si A y B se encontrasen en los asegura, todo algoritmo que cumple con la especificación de p1 cumpliría con la de p_2, pero no todo algoritmo que cumple con la especificación de p_2 cumpliría con la de p_1. Pero A y B se encuentran en los requiere, por lo tanto, p_2 es una subespecificación de p_1 y todo algoritmo que cumpla con la especificación de p_2 cumplirá la de p_1 pero no necesariamente al revés.

2. Es posible, y se puede demostrar con un ejemplo:

```
def esNegativo (num: int) -> bool :
    IF num < 8 : return true
    else : return false
```



TEST SUITE

caso negativo : -1 → true

caso positivo : 9 → False

Este ejemplo recorre todos los nodos sin encontrar el defecto.