DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Práctica 2

2do cuatrimestre 2023 — Algoritmos y Estructuras de Datos I / Introducción a la Programación

Integrante	LU	Correo electrónico
Fausto N. Martínez	363/23	fnmartinez@dc.uba.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Práctica 2

1.1. Ejercicio 1

(a) $1 \to 2$ (pues *resultado* es el doble de x)

(b) $4 \to 2$ (pues *resultado* es la raíz cuadrada de x)

(c) $\pi \to 3, -5 \to 0$ (pues *resultado* es el entero positivo más cercano a los x dados)

(d) $\langle 4, 9\rangle \to \langle 2, 3\rangle$ (pues no hay elementos repetidos en s, y son todos positivos, aparte *resultado* tiene la misma cantidad de elementos que s, y estos son la raíz cuadrada de cada elemento de s, en el mismo orden que en s)

(e) $\langle 4, 9\rangle \to \langle 3, 2\rangle$ (es el mismo problema que antes, pero ahora los *asegura* no nos piden que los elementos de *resultado* estén en el mismo orden que en s, asi que estos son valores de entrada y salida que cumplen la especificación)

(f) $\langle 4, 9\rangle \to \langle 5, 2\rangle$ (es el mismo problema que antes, pero ahora los *asegura* no nos pide que apliquemos raizCuadrada() a *todos* los elementos de s, asi que podemos aplicar raizCuadrada() solo a un elemento, por ejemplo, y que esto siga siendo un ejemplo de valores de entrada y salida que cumplen la especificación)

(g) $\langle 3, -1\rangle \to \langle\sqrt{3}\rangle$ (Esto es un ejemplo válido de valores de entrada y salida que cumplen la especificación, pues ahora la misma no me pide que los valores de entrada sean positivos, pero si nos pide que *resultado* sea la salida de aplicar raizCuadrada() solo a los elementos positivos de s)

(h) $\langle 4, 9, 16\rangle \to \langle 2, 3, 4, 0, 0, 0\rangle$ (Esto cumple la especificación pues ahora el *asegura* no nos pide que *resultado* tenga la misma longitud que s (esto tambien ocurría en el item anterior))

(i) $\langle 4, 9, 16\rangle \to \langle 2, 3\rangle$ (Ahora, como el *asegura* nos pide que la longitud de *resultado* sea como *máximo* la longitud de s, esto nos permite que un valor de salida con longitud menor que el de entrada sea un resultado que cumpla la especificación)

1.2. Ejercicio 2

1. No, pues no queremos darle a raizCuadrada() como valor de entrada un número negativo.

2. Como mencionamos antes, la diferencia entre *raicesCuadradasUno*(d) y *raicesCuadradasDos*(e) es que en *raicesCuadradasDos* los valores de salida pueden estar en cualquier orden, cosa que no pasa en *raicesCuadradasUno*

3. Si, un algoritmo que cumpla *raicesCuadradasUno*, cumple tambien *raicesCuadradasDos*, pues esta admite cualquier orden de los valores de salida (inclusive el orden "correcto"), no ocurre lo mismo al revés, pues por ejemplo, si el algoritmo que resuelve *raicesCuadradasDos* diera los valores de salida que pusimos como ejemplo en (e), este no cumpliría la especificación de *raicesCuadradasUno*

4. La diferencia entre *raicesCuadradasCinco*(h) y *raicesCuadradasSeis*(i) es que el *asegura* extra de *raicesCuadradasSeis* no nos permite que la longitud del valor de salida sea mayor que el de entrada, mientras *raicesCuadrasCinco*, al no aclararlo, lo permite.
   Luego, $\langle\sqrt{3}, \sqrt{9}\rangle$ es una salida válida para ambos problemas dado $s = \langle 3, 9, 11, 15, 18\rangle$ pues cumple el asegura de que cada posición de *resultado* es la salida de aplicarle raizCuadrada() a esa misma posición de s, y, para el caso de *raicesCuadradasSeis*, tambien cumple, que la longitud de *resultado* es, como máximo la misma que s, es decir, que la misma es menor o igual que la de s.
   Aparte $\langle\sqrt{3}, \sqrt{9}, \sqrt{11}, \sqrt{13}\rangle$ es una salida válida de *raicesCuadradasCinco* para $s = \langle 3, 9, 11\rangle$ pues este problema no tiene ningún *asegura* que nos restrinja la longitud de *resultado*

5. Como *raicesCuadradasCuatro*(g) tiene que seguir cumpliendo que se le aplique raizCuadrada() solo a los numeros positivos, ponerle un *asegura* que diga que la longitud de *resultado* sea la misma que s, simplemente haría que, si s tiene algún numero no positivo, entonces, nos sería indistinto en resultado poner su raizCuadrada() o cualquier otra cosa, siempre cumpliendo que la longitud de *resultado* sea lo mismo que s. Por ejemplo $\langle 2, -1, -2\rangle \to \langle\sqrt{2}, 0, 8000\rangle$ serían valores de entrada y salida válidos

6. No, de hecho *raicesCuadradasDos*(e) es más restrictivo que *raicesCuadradasTres*(f), pues nos pide que la salida sea aplicarle raizCuadrada() a *todos* los elementos de s, mientras que *raicesCuadradasTres* nos dice que la salida sea aplicársela *uno o varios* elementos de s. En particular, aplicársela a todos, sería una salida válida para *raicesCuadradasTres*, pero claramente no son el mismo problema, pues hay valores de entrada y salida que cumplen la especificación de *raicesCuadradasTres*, y no la de *raicesCuadradasDos*, por ejemplo los que dimos en (f)

7. Si, esos son valores de entrada y salida válidos, pues se puede ver que cumplen la especificación de *raicesCuadradasDos*, si le sacamos a ese problema el *requiere* que nos pide que no haya elementos repetidos

1.3. Ejercicio 3

(a) Si, es una salida válida pues cumple los *asegura*

(b) No es válida, pues 3 no es estrictamente mayor que 3, entonces deberíamos agregar un *requiere* que prohíba elementos repetidos en s.

(c) Es válida, lo podríamos arreglar agregando un *asegura* que pida que la cantidad de elementos de *resultado* sea igual a la cantidad de elementos de s

(d) Con el *asegura* agregado anteriormente, ya solucionamos ese problema, asi que no debemos agregar otro *asegura*.

(e) `problema ordenar` (s:$seq(\mathbb{Z})$) : $seq(\mathbb{Z})${
`requiere`:{No hay elementos repetidos en s}
`asegura`:{La longitud de *resultado* es igual a la de s}
`asegura`:{*resultado* contiene los mismos elementos que s}
`asegura`:{*resultado* es una secuencia en la cual cada elemento es estrictamente mayor al anterior}
}

1.4. Ejercicio 4

(a) Por empezar, en ningun lugar aclara que se debe duplicar los valores de entrada, luego $\langle 2, 2\rangle \rightarrow \langle 3, 4\rangle$ son resultados de entrada y salida que cumplen la especificación por ejemplo, y para nada son los que queremos que la cumplan.

(b)
- El primer asegura, nos dice que hay *algún* valor de *resultado* que es la salida de duplicar (x) para cada $x \in s$. No nos sirve pues queremos que sean todos.
- El segundo asegura, ni siquiera aclara que el resultado sea el doble que cada elemento de s, solo nos pide que sea mayor, no nos sirve
- El tercer asegura, sin embargo, es claramente lo que queremos para nuestra función, que el valor de cada posición de *resultado* sea la salida de aplicarle duplicar() a esa misma posición de s.
- El cuarto asegura, nos pide que todos los numeros de *resultado* sean pares, lo cual es cierto que necesitamos, pero ya está implícito con el tercer asegura, y poner este sería sobreespecificar (especificar de más), cosa que no queremos hacer

1.5. Ejercicio 5

1.5.A. Inciso A

`problema cantidadColectivosLinea` (linea:$\mathbb{Z}$,bondis:$seq(\mathbb{Z})$) : $\mathbb{Z}${
`requiere`:{Todos los elementos de la lista *bondis* pertenecen a (28, 33, 34, 37, 45, 107, 160, 166)}
`requiere`:{*linea* es alguno de estos numeros: 28, 33, 34, 37, 45, 107, 160, 166}
`asegura`:{*resultado* es la cantidad de veces que aparece *linea* en *bondis*}
}

1.5.B. Inciso B

`problema compararLineas`($l1 : \mathbb{Z}, l2 : \mathbb{Z}, bondis : seq(\mathbb{Z})) : \mathbb{Z}${
`requiere` : { Todos los elementos de la lista *bondis* pertenecen a (28, 33, 34, 37, 45, 107, 160, 166)}
`requiere` : {$l1$ es alguno de los siguientes numeros: 28, 33, 34, 37, 45, 107, 160, 166}
`requiere` : {$l2$ es alguno de los siguientes numeros: 28, 33, 34, 37, 45, 107, 160, 166}
`asegura` : {$resultado = l1 \leftrightarrow cantidadColectivosLinea(l1) \geq cantidadColectivosLinea(l2)$}
`asegura` : {$resultado = l2 \leftrightarrow cantidadColectivosLinea(l1) < cantidadColectivosLinea(l2)$}
}

1.6. Ejercicio 6

1.6.A. Inciso A

`problema promedioDeAlumno`($notas : seq(String \times \mathbb{Z})) : \mathbb{R}$\{
`requiere` : { Todos las $2 - uplas$ de $notas$ tienen en su segunda posición un entero entre 1 y 10}
`asegura` : {res = El resultado de sumar todas las segundas posiciones de las $2 - uplas$ de $notas$, y dividirlas por la longitud de $notas$ }
}

1.6.B. Inciso B

`problema mejorMateria`($notas : seq(String \times \mathbb{Z})) : String$\{
`requiere` : { Todos las $2 - uplas$ de $notas$ tienen en su segunda posición un entero entre 1 y 10}
`asegura` : {res = La primera coordenada ($String$) de la $2 - upla$ con la mayor segunda coordenada ($\mathbb{Z}$) de todas las $2 - uplas$ }
}

- Luego, si queremos devolver una lista de materias, tendríamos que hacer unas pares de modificaciones, la mas importante de ellas, cambiar el tipo de datos de salida de $String$ a $seq(String)$

1.6.C. Inciso C

`problema alumnosAprobados`($alumnos : seq(String \times seq(String \times \mathbb{Z})), materia : String) : seq(String)$\{
`requiere` : { Todos las $2 - uplas$ de $notas$ tienen en su segunda posición un entero entre 1 y 10}
`requiere` : { Cada primer elemento de $alumnos$ (es decir su nombre y apellido) debe ser único en la secuencia}
`asegura` : {res = Una lista con los $String$ correspondientes a los alumnos que tienen una nota mayor o igual a 4 en su secuencia asociada $notas$, en la posicion donde $materia$ es el primer elemento de la tupla perteneciente a $notas$ }
}

- Para resolver el problema del orden alfabético simplemente habría que agregar otro $asegura$ que pida que los elementos de la lista res estan ordenados en forma alfábetica. Eso reduciría la cantidad de algoritmos que resuelven el problema pues sería un problema mas *restrictivo*

1.6.D. Inciso D

`problema promedioDeTodosLosAlumnos`($alumnos : seq(String \times seq(String \times \mathbb{Z})), materia : String) : seq(String \times \mathbb{R})$\{
`requiere` : { Todos las $2 - uplas$ de $notas$ tienen en su segunda posición un entero entre 1 y 10}
`requiere` : { Cada primer elemento de $alumnos$ (es decir su nombre y apellido) debe ser único en la secuencia}
`asegura` : {res tiene la misma longitud que $alumnos$}
`asegura` : {res es la salida de aplicar promedioDeAlumno() a cada segunda coordenada ($notas$) de la lista $alumnos$}
}