Exercitium (curso 2013–14)

Ejercicios de programación funcional con Haskell

José A. Alonso Jiménez

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Sevilla, 8 de diciembre de 2018

Para Guiomar

Índice general

Introducción

> *"The chief goal of my work as an educator and author is to help people learn to write beautiful programs."*
>
> (Donald Knuth en Computer programming as an art)

Este libro es una recopilación de las soluciones de los ejercicios propuestos en el blog Exercitium [1] durante el curso 2013–14.

El principal objetivo de Exercitium es servir de complemento a la asignatura de Informática [2] de 1º del Grado en Matemáticas de la Universidad de Sevilla.

Con los problemas de Exercitium, a diferencias de los de las relaciones [3], se pretende practicar con los conocimientos adquiridos durante todo el curso, mientras que con las relaciones están orientadas a los nuevos conocimientos.

Habitualmente de cada ejercicio se muestra distintas soluciones y se compara sus eficiencias.

La dinámica del blog es la siguiente: cada día, de lunes a viernes, se propone un ejercicio para que los alumnos escriban distintas soluciones en los comentarios. Pasado 7 días de la propuesta de cada ejercicio, se cierra los comentarios y se publica una selección de sus soluciones.

Para conocer la cronología de los temas explicados se puede consultar el diario de clase [4].

El código del libro se encuentra en GitHub [5]

[1] https://www.glc.us.es/~jalonso/exercitium
[2] https://www.cs.us.es/~jalonso/cursos/i1m-13
[3] https://www.cs.us.es/~jalonso/cursos/i1m-13/ejercicios/ejercicios-I1M-2013.pdf
[4] https://www.glc.us.es/~jalonso/vestigium/category/curso/i1m/i1m2013
[5] https://github.com/jaalonso/Exercitium2013

Ejercicio 1

Iguales al siguiente

Ejercicio propuesto el 21–4–14

Definir la función

```
igualesAlSiguiente :: Eq a => [a] -> [a]
```

tal que (igualesAlSiguiente xs) es la lista de los elementos de xs que son iguales a su siguiente. Por ejemplo,

```
igualesAlSiguiente [1,2,2,2,3,3,4]  ==  [2,2,3]
igualesAlSiguiente [1..10]          ==  []
```

Soluciones

```
import Data.List (group)
import Test.QuickCheck

-- 1ª definición (por comprensión):
igualesAlSiguiente :: Eq a => [a] -> [a]
igualesAlSiguiente xs =
  [x | (x,y) <- zip xs (tail xs), x == y]

-- 2ª definición (por recursión):
igualesAlSiguiente2 :: Eq a => [a] -> [a]
```

```haskell
igualesAlSiguiente2 (x:y:zs)
  | x == y    = x : igualesAlSiguiente2 (y:zs)
  | otherwise = igualesAlSiguiente2 (y:zs)
igualesAlSiguiente2 _ = []

-- 3ª definición (con concat y comprensión):
igualesAlSiguiente3 :: Eq a => [a] -> [a]
igualesAlSiguiente3 xs = concat [ys | (_:ys) <- group xs]

-- 4ª definición (con concat y map):
igualesAlSiguiente4 :: Eq a => [a] -> [a]
igualesAlSiguiente4 xs = concat (map tail (group xs))

-- 5ª definición (con concatMap):
igualesAlSiguiente5 :: Eq a => [a] -> [a]
igualesAlSiguiente5 xs = concatMap tail (group xs)

-- 6ª definición (con concatMap y sin argumentos):
igualesAlSiguiente6 :: Eq a => [a] -> [a]
igualesAlSiguiente6 = concatMap tail . group

-- Equivalencia
-- ============

-- La propiedad es
prop_igualesAlSiguiente_equiv :: [Int] -> Bool
prop_igualesAlSiguiente_equiv xs =
  igualesAlSiguiente xs == igualesAlSiguiente2 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente3 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente4 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente5 xs &&
  igualesAlSiguiente xs == igualesAlSiguiente6 xs

-- La comprobación es
--    +++ OK, passed 100 tests.
--    (0.07 secs, 9,911,528 bytes)
```

Ejercicio 2

Ordenación por el máximo

Ejercicio propuesto el 22–4–18

Definir la función

```
ordenadosPorMaximo :: Ord a => [[a]] -> [[a]]
```

tal que (ordenadosPorMaximo xss) es la lista de los elementos de xss ordenada por sus máximos. Por ejemplo,

```
ghci> ordenadosPorMaximo [[3,2],[6,7,5],[1,4]]
[[3,2],[1,4],[6,7,5]]
ghci> ordenadosPorMaximo ["este","es","el","primero"]
["el","primero","es","este"]
```

Soluciones

```
import Data.List (sort)
import GHC.Exts  (sortWith)
import Test.QuickCheck

-- 1ª definición
ordenadosPorMaximo :: Ord a => [[a]] -> [[a]]
ordenadosPorMaximo xss =
  map snd (sort [(maximum xs,xs) | xs <- xss])
```

```haskell
-- 2ª definición
ordenadosPorMaximo2 :: Ord a => [[a]] -> [[a]]
ordenadosPorMaximo2 xss =
  [xs | (_,xs) <- sort [(maximum xs,xs) | xs <- xss]]

-- 3ª definición:
ordenadosPorMaximo3 :: Ord a => [[a]] -> [[a]]
ordenadosPorMaximo3 = sortWith maximum

-- Equivalencia
-- ============

verificaOrdenadosPorMaximo :: IO ()
verificaOrdenadosPorMaximo =
  quickCheck prop_ordenadosPorMaximo

prop_ordenadosPorMaximo :: [[Int]] -> Bool
prop_ordenadosPorMaximo xs =
  ordenadosPorMaximo ys == ordenadosPorMaximo2 ys
  where ys = filter (not . null) xs

-- Comprobación
--    λ> verificaOrdenadosPorMaximo
--    +++ OK, passed 100 tests.
```