Haskell no muerde (Práctica 1)

Sergio García Macías

ÍNDICE

Ejercicios:

1. **cambia_el_primero(a,b):** cambia el primer valor de la lista a por el valor de a

2. **cambia_el_n(a,n,b):** cambia el valor de la posición n de la lista a por el valor de a

3. **get_mayor_abs(a):** devuelve el mayor número en valor absoluto de la lista a

4. **num_veces(a,b):** devuelve la cantidad de veces que aparece el valor a en la lista b

5. **palabras_mayores_n(n,a):** devuelve una lista con las palabras mayores que n

Ejercicios propuestos por mi:

1. **spri(a):** suma todos los valores pares y le resta la suma de todos los valores impares

2. **tiene_divisor(a):** devuelve verdadero en caso de que tenga algún divisor menor que él

3. **genera_n_primos(n):** genera los primeros ‘n’ números primos

4. **compon_foldr_de_n(n):** expande el foldr de la función suma de los primeros n números naturales

5. **spnnp:** suma los primeros n **NO** primos

Resolución:

1. **get_mayor_abs**:

   Aplicar la función abs para todo |*e*| de la lista y después con "*maximum*" obtener el valor mayor.

2. **num_veces**:

   Aplicar *filter* para todo |*e*| de la lista tal que *(e == valor)* y obtener la longitud de la lista generada.

3. **cambia_el_primero**:

   Usando la notación de listas de Haskell *(head:tail)* sustituir head por valor *"valor:tail"*

4. **cambia_el_n**:

   Mediante las funciones *take* y *drop* obtener los *n-1* primeros elementos, concatenarlo con el operador ++ al valor deseado y volver a concatenar ignorando los primeros *n* elementos.

5. **palabras_mayores_n**:

   Aplicar la función *filter* para cada elemento |*e*| tal que:

   (length e > n)

Resolución de mis ejercicios:

1. **spri**:

   Mediante aplicar la función sum a la lista generadas por:

   (filter(\x -> mod x 2 == 0) values) (pares)
   (filter(\x -> mod x 2 == 1) values) (impares)

   y después restar los valores obtenidos.

2. **tiene_divisor**:

   Haciendo uso de la función *"any"* comprobar que exista un elemento |*e*| en el dominio *[2,3 .. ⌊$\sqrt{n}$⌋]* que cumpla *(n%e == 0)*

   (El dominio llega hasta la raíz de *n* porque esta función va a ser usada como "Mickey-herramienta"[1] más adelante)

3. **genera_n_primos**:

   Usando la función *"take"* para obtener los primeros *n* elementos del dominio *[1,2 ..]* que cumplan:

   (\e -> tiene_divisor e == False)

4. **compon_foldr_de_n**:

   Para componer/expandir la suma de *foldr* haremos uso de *foldr*. Como queremos devolver *[Char]* debemos usar como estado inicial algo de

[1] Mickey-herramienta: Expresión lúdica utilizada en el contexto de la serie de animación Mickey Mouse para describir las herramientas y objetos utilizados por Mickey Mouse y sus amigos en diversas situaciones.

tipo *[Char]* por lo que nuestro estado inicial será “0” al suponer que el estado inicial del foldr que queremos componer es 0.

Para cada |*e*| del dominio *[1,2..n]* aplicaremos:

(\x y -> “({x} + {y})”)

Donde con {x}, {y} me refiero a insertar el valor de las variables como cadena. Con esto conseguimos que al estado del foldr que vamos manteniendo en cada iteración le añadamos “*(x+{estado_anterior}*” por lo que se obtiene un resultado como el siguiente:

"(1+(2+(3+0)))"

5. **spnnp**:

Volviendo a usar las funciones *sum* y *filter* sobre el dominio *[1,2..n]* obtenemos los elementos |*e*|tal que cumplan:

(tiene_divisor e == True)

Y por último aplicamos *sum* al nuevo dominio generado.

Código de los ejercicios:

```haskell
get_mayor_abs:: [Integer] -> Integer
get_mayor_abs values = maximum (map (abs) values)

num_veces::(Eq a) => [a] -> a -> Int
num_veces values val = length ( filter (\x -> x == val) values )

cambia_el_primero::[a] -> a -> [a]
cambia_el_primero (_:resto) val = val:resto

cambia_el_n::a -> Int -> [a] -> [a]
cambia_el_n new_val n list = take (n-1) list ++ [new_val] ++ drop (n) list

palabras_mayores_n:: Int -> [[Char]] -> [[Char]]
palabras_mayores_n minimo palabras = filter (\x -> length x > minimo) palabras
```

Código de mis ejercicios:

```haskell
-- Suma Pares Resta Impares
spri:: [Integer] -> Integer
spri values = sum (filter(\x -> mod x 2 == 0) values) - sum (filter(\x -> mod x 2 == 1) values)

-- Tiene algun divisor mayor que pero menor que el
tiene_divisor:: Integer -> Bool
tiene_divisor n = any(\x -> mod n x == 0) [2,3 .. floor(sqrt(fromIntegral n))]
--                                                  ^^^^^^^^^^^^
--                                              Esto lo he sacado de la documentacion
--                                              de Haskell que me hacia falta para la sqrt


-- Haskell es lentillo y mi algoritmo es super basico asi que
-- por si acaso no le pidas mas de 100 numeros primos
genera_n_primos:: Int -> [Integer]
genera_n_primos n = take(n) (filter(\x -> tiene_divisor x == False) [1,2..])


-- Esto esta "to wapo"
compon_foldr_de_n:: Integer -> [Char]
compon_foldr_de_n n = foldr (\x y -> "(" ++ show x ++ "+" ++ y ++ ")" ) "0" [1,2 .. n]

-- Suma los Primeros N No Primos
spnnp:: Integer -> Integer
spnnp n = sum (filter(\x -> tiene_divisor x == True) [1,2 .. n])
```