LENGUAJES DE PROGRAMACIÓN 1
Tarea 1 de Haskell
5 puntos

Fernando Lovera; Sergio Terán

Enero-Marzo 2018

1 Generalidades.

Al resolver la tarea, no sólo concentrate en crear código que funcione si no también en crear código que tenga buen estilo de programación, así te acostumbraras a programar "bien" desde el principio.

Intenta crear pequeñas funciones que tengan resultados precisos y luego combinar todas esas funciones en vez de crear una mega función que lo resuelva todo ("step by step" o "divide and conquer").

Utiliza nombres nemónicos, utiliza funciones del preludio de Haskell (lee el preludio de Haskell, para que se te facilite la vida), intenta escribir todos los tipos en tus funciones (los tipos son muy importantes), estas en todo tu derecho de crear funciones adicionales/auxiliares.

La tarea vale 5 puntos en total y tienen una semana a partir de hoy para entregarla. Mánden su tarea a mi correo principal (flovera1@gmail.com) y al correo de Sergio Terán (sergioteran.zambrano@gmail.com), su correo debe indicar sus números de carnet de la siguiente manera:
carnet1carnet2, por ejemplo: 07-4112607-41125.

Está terminantemente prohibido la comunicación entre grupos, no aceptamos copias de tareas; es preferible que entreguen algo incompleto pero suyo que algo completo pero con cosas copiadas, ésto será inaceptable.

2 Validar Números de tarjetas de crédito. 2.5 puntos

La forma en que se validan las tarjetas de crédito en los sitios web es a través de una fórmula de checksum.
En esta sección ustedes van a intentar implementar un algoritmo que valide dichos números. El algoritmo sigue los siguientes pasos:

1. Duplicar el valor de cada segundo dígito comenzando por la derecha. El último dígito permance sin cambios.
   Por ejemplo: [1,2,3,4] se transforma en [2,4,6,4].

2. Agregar los dígitos de los valores duplicados y los dígitos sin duplicar del número original.
   Por ejemplo, [2,3,16,6] se convierte en 2 + 3 + 1 + 6 + 6 = 18.

3. Calcular el resto cuando la suma es dividida por 10. Si el resultado es igual a 0, entonces el número es válido.

2.1

Define las funciones:

(a) aDigits :: Integer -¿ [Integer]

(b) aDigitosRev :: Integer -¿ [Integer]

aDigitos convierte enteros positivos a listas con sus dígitos: 123 sería: [1,2,3]
aDigitosRev hace lo mismo pero con los dígitos en reversa!.
-Qué hago con el caso base?
aDigitos 0 == []
-Qué hago con números negativos?
aDigitos(-42) == []

2.2

Duplicamos cada otro dígito cuando todos los dígitos están en orden, define una función:
duplicarCadaOtro :: [Integer]-¿[Integer]
Debes empezar a duplicar cada otro número a partir de la derecha, por ejemplo:
duplicarCadaOtro [8,7,6,5] == [16,7,12,5]

2.3

El output de duplicarCadaOtro tiene una combinación de números de un dígito y dos dígitos. Definir la función:
sumDigitos :: [Integer] -¿ Integer
Ésta debe calcular la suma de todos los dígitos.

2.4

Define la función que valida:
validar :: Integer -¿ Bool

Ésto indica si un entero puede o no ser una tarjeta de crédito válida. Esta función utiliza todas las funciones definidas anteriormente.
Example: validate 4012888888881881 = Verdadero
Example: validate 4012888888881882 = Falso

3 Recursividad. 2.5 puntos

Aún no hemos introducido en tema de recursividad en clase, pero sí conocemos tipos recursivos, entonces lo que cambia es un poco la sintaxis. Tenemos la siguiente definición de árbol binario:

– árbol binario data ArbBinario = L — N BinTree BinTree deriving (Eq, Show)

– ésta función crea un árbol binario completo de tamaño 2 ^ $(n + 1) - 1$

```
crearArbBinario 0 = L
crearArbBinario n = N (crearArbBinario (n-1)) (crearArbBinario (n-1))
```

– esta función calcula el tamaño del árbol
tam L = 1
tam (N t1 t2) = 1 + tam t1 + tam t2

En la definición de ArbBinario, cada nodo N interno debe tener dos subárboles. Por lo tanto, no hay forma de construir un árbol de tamaño 2. Si quieres un árbol binario que puede ser de cualquier tamaño, entonces necesitas introducir un nuevo constructorN1 en ArbBinario que permita un nodo interno con un solo subárbol. Haga esta modificación a ArbBinario.

Acomda la definición de tamaño para tener en cuenta esta modificación.

Escriba una profundidad de función que proporcione la profundidad del árbol de ArbBinario modificado. La profundidad de un árbol de tamaño 1 es 1.

Escriba una función hacerABinTree de modo que hacerABinTree s construya un ArbBinario de tamaño exactamente n