Proyecto 2

Algoritmos y Estructuras de Datos I - Laboratorio
Recursión, alto orden, expresiones lambda y otros elementos de Haskell

14 de abril de 2010

En este proyecto hay una serie de funciones para programar en Haskell. En todos los ejercicios escribir el tipo (y la clase) al que pertenece la función.

1. Hacer una función que devuelva el mínimo de una lista. Primero hacerla con caso base e inductivo y después utilizando el funcional `foldr`. Para este último programa ver si existe una función en el Preludio de Haskell que devuelva el mínimo de dos números y usarla.

   Ayuda. ¿En qué clase debería encontrarse esta función?

   Además, para definir el caso base puede haber un problema. Investigar a qué otra clase debería pertenecer el tipo de los elementos de la lista para poder hacerlo y poner el tipo de la función con ambas clases.

2. Escribir las siguientes funciones con caso base e inductivo y después utilizando el funcional `map`. Además para esto último, el programa debe constar de una sola línea y sin definiciones locales:

   a) `sumarList :: Num a => a -> [a] -> [a]` que toma un número y una lista de números y le suma a cada elemento de la lista el primer parámetro. Por ejemplo:

   ```
   sumarList 3 [4,6,7] = [7,9,10]
   ```

   b) `encabs :: a -> [[a]] -> [[a]]` que toma un elemento y lo pone en la cabeza de cada elemento de las listas del segundo parámetro. Por ejemplo:

   ```
   encabs 3 [[2,1],[],[4,7]] = [[3,2,1],[3],[3,4,7]]
   ```

3. Programar una función `encuentra` que dado un valor de tipo `Int` y una lista de pares con tipo `[(Int,String)]`[1] devuelve el segundo elemento del primer par cuyo primer elemento es igual al primer parámetro. En el caso que ningún elemento de la lista cumpla con esto, devolver el string vacío. Por ejemplo:

   ```
   encuentra 103 [(40,"tos"),(103,"vela"),(16,"taza")] = "vela"
   encuentra 102 [(40,"tos"),(103,"vela"),(16,"taza")] = ""
   encuentra 102 [] = ""
   ```

   Ayuda. Pensar inducción sobre la lista.

[1] En haskell `String` es lo mismo que lista de `Char`.

Pregunta: ¿Se puede programar la función con el funcional `foldr`? Si se puede, prográmelo.

4. La función `primIgualesA` toma un valor y una lista y devuelve el tramo inicial de la lista cuyos elementos son iguales al valor. Por ejemplo:

```
primIgualesA 3 [3,3,4,1] = [3,3]
primIgualesA 3 [4,3,3,4,1] = []
primIgualesA 3 [] = []
primIgualesA 'a' "aaadaa" = "aaa"
```

   a) Programar esta función con caso base e inductivo.

   b) Programar esta función con `foldr`.

   c) Buscar en `Prelude.hs` si existe una función que la generaliza y utilizarla para programarla en una sola linea.

   d) Programar la función `primIguales` que toma una lista y devuelve el tramo inicial de la lista cuyos elementos son iguales. Por ejemplo:

```
primIguales [3,3,4,1] = [3,3]
primIguales [4,3,3,4,1] = [4]
primIguales [] = []
primIguales "aaadaa" = "aaa"
```

5. La función `lineas :: String -> [String]` toma una texto y devuelve las lineas del mismo en el orden que aparecen. Se define una linea de un texto como una mayor subsecuencia de caracteres del mismo, que no contenga el caracter `'\n'`. Para programar esta función seguir los siguientes pasos:

   a) Programar la función `subsecs:: (a -> Bool) -> [a] -> [[a]]` que dado un predicado y una lista devuelve las subsecuencias más largas cuyos elementos hagan verdadero el predicado, en el orden que aparecen. Por ejemplo:

```
subsecs (==0) [0,0,2,0,3,0,0,0] = [[0,0],[0],[0,0,0]]
subsecs (==0) [1,0,0,1,0,0] = [[],[0,0],[0,0]]
subsecs (==0) [1,0,0,1,1,0,0] = [[],[0,0],[],[0,0]]
subsecs (/='\n') "hola que\ntal" = ["hola que","tal"]
subsecs (/='\n') "" = [""]
subsecs (/='\n') "\n" = ["",""]
```

   b) Con esta función definir la función `lineas`.

   c) Con estas funciones definir la función `palabras` que toma un texto y devuelve sus palabras. Se define una palabra de un texto como una mayor subsecuencia de caracteres del mismo, que no contenga espacios ni saltos de línea y que contenga por lo menos un caracter.

   Probar esta función con archivos de texto. Para ello utilizar la función que toma el nombre de un archivo y devuelve un string:

```
-- extensión para poder leer archivos en Strings
import Hugs.IOExts(unsafePerformIO)

leeArchivo :: String -> String
leeArchivo nombre = unsafePerformIO $ readFile nombre
```

6. Programar la función

```
intercal :: a -> [a] -> [[a]]
```

donde `intercal x ys` devuelve todas las posibles intercalaciones de `x` en la lista `ys`. Por ejemplo:

```
intercal 100 [6,3] = [[100,6,3],[6,100,3],[6,3,100]]
intercal 1 [3,4,5] = [[1,3,4,5],[3,1,4,5],[3,4,1,5],
                      [3,4,5,1]]
intercal 'u' "clo" = ["uclo","culo","cluo","clou"]
intercal 3 [] = [[3]]
```

Ayuda. Es posible que necesite crear una función auxiliar.

7. Programar la función `intercal` del ejercicio 6, utilizando el funcional `map` y sin utilizar funciones auxiliares.