Departamento de Computação
Universidade Federal de Ouro Preto
Programação Funcional
Prof. Lucília Figueiredo

Lista de Exercícios 02 - Funções de ordem superior

1. Defina a função `segments :: (a -> Bool) -> [a] -> [[a]]` que, dado um predicado `p` e uma lista `l`, divide essa lista em segmentos que satisfazem o predicado `p`. Por exemplo:

   ```
   segments (>=0) [1,2,3.-1,4,-2,-3,5] ▷ [[1,2,3],[4],[5]]
   ```

   Usando essa função, defina as seguintes funções:

   (a) `lines :: String -> [String]` que divide um texto em linhas, considerando como uma linha toda seqüência de caracteres até que seja encontrado o caractere de fim de linha ou o final do texto.

   (b) `words :: String -> [String]` que separa uma linha em palavras, usando a função `isSpace :: Char -> Bool` para determinar se um caractere deve ser considerado como um separador de palavras.

2. A função `foldn :: (a -> a) -> a -> Int -> a` é definida do seguite modo (para valores inteiros não negativos):

   ```
   foldn f e 0 = e
   foldn f e n = f (foldn f e (n-1))
   ```

   (a) Use essa função para definir a função `replica :: Int -> a -> [a]` que, dado um valor inteiro não-negativo `n` e um valor `v`, de tipo `a`, retorna uma lista em que `v` ocorre `n` vezes.

   (b) Usando a definição acima e a função `map`, defina uma função `dupS :: String -> String` que retorna o string obtido duplicando cada um dos caracteres do string dado como argumento. Por exemplo, `dupS "alo"` retorna `"aalloo"`.

3. A aplicação da função `foldr (⊕) e xs` resulta em um termo no qual os elementos da lista são combinados, por meio do operador `(⊕)`, usando associatividade à direita:

   $$\texttt{foldr (}\oplus\texttt{) e [a}_1\texttt{,a}_2\texttt{,}\cdots\texttt{,a}_n\texttt{] = a}_1 \oplus \texttt{(a}_2 \oplus \cdots \texttt{(a}_n \oplus \texttt{e)} \cdots \texttt{)}$$

   (a) Defina a função dual de `foldr`, `foldl`, que combina os elementos da lista usando associatividade à esquerda, isto é:

   $$\texttt{foldl (}\oplus\texttt{) e [a}_1\texttt{,a}_2\texttt{,}\cdots\texttt{,a}_n\texttt{] = (}\cdots \texttt{(e} \oplus \texttt{a}_1\texttt{)} \oplus \texttt{a}_2\texttt{)} \cdots \oplus \texttt{a}_n\texttt{)}$$

   (b) Dê um exemplo de uma função `f` tal que `foldr f` $\neq$ `foldl f`.

   (c) Defina a função `map` usando `foldr`

4. Defina cada uma das funções a seguir, usando `foldr` e `foldl`:

   (a) `elem :: a -> [a] -> Bool`, que determina se um valor é uma elemento de uma lista.

   (b) `remdups :: Eq a => [a] -> [a]` que remove da lista elementos duplicados adjacentes.

5. Defina cada uma das funções a seguir, usando `foldr`:

   (a) `todos :: (a -> Bool) -> [a] -> Bool` que, dado um predicado `p :: a -> Bool` e uma lista de valores do tipo `a`, determina se todos os elementos da lista satisfazem o predicado `p`.

   Usando essa função, defina a função `positivos :: [Int] -> Bool`, que determina se todos os elementos da lista dada como argumento são inteiros positivos.

   (b) `algum :: (a -> Bool) -> [a] -> Bool` que, dado um predicado `p :: a -> Bool` e uma lista de valores do tipo `a`, determina se algum dos elementos da lista satisfaz o predicado `p`.

   Usando essa função, defina a função `membro :: a -> [a] -> Bool` que determina se o valor dado como primeiro argumento ocorre na lista dada como segundo argumento.

6. Dada uma lista de números `xs = [`x_1`,`x_2`,`$\cdots$`,`x_n`]`, a seqüência sucessiva máxima `ssm` de `xs` é a maior subseqüência `[`x_{j_1}`,`x_{j_2}`,`$\cdots$`,`x_{j_k}`]` tal que $j_1 = 1$ e $x_{j_s} < y_{j_t}$ para $j_s < j_t$. Por exemplo: `ssm [3,1,3,4,9,2,10,7]` é `[3,4,9,10]`. Defina a função `ssm` usando `foldr`.

7. A função `zipWith` é definida como:

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith _ []     _      = []
zipWith _ _      []     = []
zipWith f (x:xs) (y:ys)= f x y :  zipWith xs ys
```

   Usando a função `addLists :: Numa a => [a] -> [a] -> [a]`, definida como `addLists = zipWith (+)`, podemos definir listas infinitas:

```
ones, nats ::  [Int]
ones = 1 :  ones
nats = addLists ones nats
```

   (a) Defina a lista dos númneros de Fibonacci `fibs = [0,1,1,2,3,5,8,13,..]` usando essa idéia. Qual seria a complexidade do tempo de execução de `take n fibs` (em função de `n`), usando a sua definição de `fibs`?

   (b) Usando a função `addLists`, defina a função `runningSums :: [Int] -> [Int]` que, dada uma lista `[`a_0`,`a_1`,`a_2`,`$\cdots$`]` calcula a soma `[0,`a_0`,`a_0`+`a_1`,`a_0`+`a_1`+`a_2`,`$\cdots$`]`