

Programação Funcional

Folha de Exercícios 03

Funções de Alta ordem

Prof. Wladimir Araújo Tavares

1. Escreva a função `divideMedia :: [Double] -> ([Double], [Double])` tal que `(divideMedia xs)` devolve o par (ys, zs) , onde ys é formado pelos elementos de xs estritamente menores que a média, enquanto zs é formado pelos elementos de xs estritamente maiores que a média. Por exemplo,

```
divideMedia [6,7,2,8,6,3,4] == ([2.0,3.0,4.0],[6.0,7.0,8.0,6.0])
```

Dica: defina a função `media` e use a função `filter :: (a->Bool) -> [a] -> [a]`.

2. A função do prelúdio `scanl` é uma variante do `foldl` que produz a lista com os valores acumulados:

```
scanl f z [x1, x2, ...] = [z, f z x1, f (f z x1) x2, ...]
```

Por exemplo:

```
scanl (+) 0 [1, 2, 3] = [0, 0 + 1, 0 + 1 + 2, 0 + 1 + 2 + 3] = [0, 1, 3, 6]
```

- (a) Use a função `scanl` para definir a função `fatAcc n` que retorna uma lista de com $[1!, 2!, \dots, n!]$
- ```
fatAcc 10 == [1,2,6,24,120,720,5040,40320,362880,3628800]
```
- (b) Use a função `fatAcc` para definir a função `fatorial`.

3. Escreva uma função `partir :: Int -> [a] -> [[a]]` tal que `partir n` de decompõe uma lista em sub-listas cuja concatenação dá a lista original e tal que cada sub-lista tem comprimento  $n$  (exceto, possivelmente, a última).

Exemplo: `partir 5 "abcdefghijk" = ["abde", "fghij", "kl"]`

Dica: use a função `take, drop` e `(:)`

4. Defina uma função `group` em Haskell que recebe como argumento uma lista  $l$  de valores e agrupa os valores repetidos, retornando uma lista de pares nos quais o primeiro componente é um valor de  $l$  e o segundo é o número de repetições consecutivas desse valor.

Por exemplo:

```
group [1,1,2,2,2,3,4,5,5,1] == [(1,2),(2,3),(3,1),(4,1),(5,2),(1,1)]
```

- (a) Usando a função `takeWhile`, encontre uma sub-cadeia inicial que contém todos os números iguais.
- (b) Usando a função `dropWhile`, remova uma subcadeia inicial que contém todos os números iguais.
- (c) Escreva uma definição recursiva para a função `group`.

5. Considere a seguinte definição de função em Haskell:

```
until p f x = if p x then x else until p f (f x)
```

- (a) Qual é o tipo da função `until`?
- (b) Qual é o resultado da avaliação da expressão `until (< 10) square 2`?
- (c) Use a função `until` para definir uma função que, dado um string  $s$ , retorne o string obtido removendo-se todos os caracteres iguais a branco que ocorrem no início de  $s$ .

6. Defina a função `relacionados :: (a->a->Bool) -> [a] -> Bool` tal que `(relacionados p xs)` verifica se todo par  $(x, y)$  de elementos consecutivos de  $xs$  satisfazem o predicado  $p$ . Por exemplo,

```
relacionados (<) [2,3,7,9] = True
relacionados (<) [2,3,1,9] = False
```

- (a) Escreva uma definição recursiva para `relacionadosRec`.
- (b) Escreva uma definição usando compreensão de listas `relacionadosComp` com a função `and`.

7. Defina a função `agrupa :: Eq a => [[a]] -> [[a]]` tal que `(agrupa xss)` é uma lista de listas obtidas agrupando os primeiros elementos, os segundos elementos, ... de forma que o comprimento das listas dos resultados seja igual a lista mais curta de  $xss$ . Por exemplo,

```
agrupa [[1..6],[7..9],[10..20]] == [[1,7,10],[2,8,11],[3,9,12]]
agrupa [] == []
```

8. O que a função `mystery` faz?

```
mystery xs = foldr (++) [] (map sing xs)
sing x = [x]
```

9. Escreva a definição da função `concatenaFold :: [[a]] -> [a]` que concatena uma lista de listas usando a função `foldr :: (a -> b -> b) -> b -> [a] -> b`

Dica: Considere a seguinte definição recursiva:

```
concatena :: [[a]] -> [a]
concatena [] = []
concatena (x:xs) = x ++ concatena xs
```

10. Escreva a definição da função `inverteFold :: [a] -> [a]` que inverte uma lista usando a função `foldr :: (a -> b -> b) -> b -> [a] -> b`

Dica: Considere a seguinte definição recursiva:

```
inverte :: [a] -> [a]
inverte [] = []
inverte xs = (last xs):inverte (init xs)
```

11. Escreva a definição da função `tamanhoFold :: [a] -> Int` que retorna o tamanho de uma lista usando a função `foldr :: (a -> b -> b) -> b -> [a] -> b`.

Dica: Considere a seguinte definição recursiva:

```
tamanho :: [a] -> Int
tamanho [] = 0
tamanho (x:xs) = 1 + length xs
```

12. Escreva a definição da função `elementoFold a :: [a] -> Bool` que determina se um elemento pertence ou não a uma lista usando a função `foldr :: (a -> b -> b) -> b -> [a] -> b`.

Dica: Considere a seguinte definição recursiva:

```
elemento :: Eq a => a -> [a] -> Bool
elemento y [] = False
elemento y (x:xs) = (x == y) || elemento y xs
```

13. Escreva uma função `paridade :: [Bool] -> Bool` que calcule a paridade de uma sequência de bits (representados como uma lista de booleanos): se o número de bits de valor `True` for ímpar então a paridade é `True`, caso contrário é `False`.

Exemplo: `paridade [True,True, False,True] = True`

- (a) Escreva uma definição recursiva para a função `paridade`.
- (b) Escreva uma definição usando o `foldr` para a função `paridadeFold`.

14. A função `duplicar :: String -> String` repete duas vezes cada vogal (letras 'a', 'e', 'i', 'o', 'u' minúsculas ou maiúsculas) numa cadeia de caracteres; os outros caracteres devem ficar inalterados.

Exemplo: `duplicar "Ola, mundo!" == "O0laa, muundo0!"`

Dica: Crie uma lista com as vogais minúsculas e maiúsculas.

- (a) Escreva uma definição recursiva para a função `duplicar`.
- (b) Escreva uma definição usando o `foldr` para a função `duplicarFold` que faz o mesmo que `duplicar`.

15. Defina a função `filtraAplica :: (a->b) -> (a->Bool)->[a]->[b]` tal que `(filtraAplica f p xs)` é uma lista obtida aplicando a função `f` aos elementos de  $xs$  que satisfazem o predicado  $p$ . Por exemplo,

```
filtraAplica (4+) (<3) [1..7] => [5,6]
```

Defina a função:

- (a) Usando compreensão de listas.
- (b) Usando a função `map` e `filter`.
- (c) Usando recursão
- (d) Usando a função `foldr`

16. Considere um polinômio  $P(X) = c_0 + c_1 z + \dots + c_n z^n$  representado pela lista dos seus coeficientes  $[c_0, c_1, \dots, c_n]$ . Podemos calcular o valor do polinômio num ponto de forma eficiente usando a forma de Horner :

$$P(z) = c_0 + c_1 z + \dots + c_n z^n = c_0 + z * (c_1 + z * (\dots + z * (c_{n-1} + z * c_n) \dots)) \quad (1)$$

Note que usando a expressão não necessitamos de calcular potências: para calcular o valor dum polinômio de grau  $n$  usamos apenas  $n$  adições e  $n$  produtos.

Complete a seguinte definição recursiva tal que `horner cs z` calcula o valor do polinômio com lista de coeficientes  $cs$  no ponto  $z$  usando a forma de Horner.

```
horner :: [Double] -> Double -> Double
horner [] z = 0
horner (c:cs) z =
```

A forma de Horner também pode ser expressa como aplicação da função de ordem superior `foldr`. Complete a definição seguinte de forma a que a igualdade seja correta.

```
horner cs z = foldr _____ cs
```

17. Defina a função `mapFold f :: [a] -> [a]` usando a função `foldr`.
18. Defina a função `filterFold p :: [a] -> [a]` usando a função `foldr`.
19. As funções `foldl1` e `foldr1` do prelúdio-padrão são variantes de `foldl` e `foldr` que só estão definidas para listas com pelo menos um elemento (i.e. não-vazias). `foldl1` e `foldr1` têm apenas dois argumentos (uma operação de agregação e uma lista) e o seu resultado é dado pelas equações seguintes.

$$foldl1(\oplus)[x_1, \dots, x_n] = (\dots (x_1 \oplus x_2) \oplus x_3 \dots) \oplus x_n$$
$$foldr1(\oplus)[x_1, \dots, x_n] = x_1 \oplus (\dots (x_{n-1} \oplus x_n) \dots)$$

Defina as funções `maximoFold`, `minimoFold :: Ord a => [a] -> a` do prelúdio-padrão (que calculam, respectivamente, o maior e o menor elemento duma lista não-vazia) usando `foldl1` e `foldr1`.

20. Escreva definição recursiva da função `insert :: Ord a => a -> [a] -> [a]` para inserir um elemento numa lista ordenada na posição correta de forma a manter a ordenação.
- Exemplo: `insert 2 [0, 1, 3, 5] == [0, 1, 2, 3, 5]`
21. Usando a função `insert`, escreva uma definição também recursiva da função `insertSort :: Ord a => [a] -> [a]` que implementa ordenação pelo método de inserção:
- a lista vazia já está ordenada;
  - para ordenar uma lista não vazia, recursivamente ordenamos a cauda e inserimos a cabeça na posição correta.
22. Defina a função `insertSortFold :: Ord a => [a] -> [a]` que ordena uma lista pelo método de inserção usando a função `foldr` e `insert`.
23. A função `scanSum :: [Int] -> [Int]` recebe uma lista de inteiros e retorna uma lista com as somas das somas acumuladas. Por exemplo,
- ```
scanSum [1,2,3,4] == [1,3,6,10]
scanSum [1,3,5,7] == [1,4,9,16]
```
- (a) Defina uma função `prefixos :: [Int] -> [[Int]]` que retorna uma lista de prefixos de todos os prefixos de uma lista.
- ```
prefixos [1,2,3,4] == [[1], [1,2], [1,2,3], [1,2,3,4]]

prefixos [] = -----
prefixos xs = prefixos ----- ++ [xs]
```
- (b) Defina `scanSum` usando compreensão de listas com o auxílio da função `prefixos`.
- (c) Defina a função `scanSum` recursivamente. Dica: use uma função auxiliar