

Teste

Programação Funcional – 1º Ano, LEI / LCC
22 de Janeiro de 2013

Duração: 2 horas (teste completo) / 1 hora (Parte II)

Parte I

Esta parte do teste representa 12 valores da cotação total. Cada alínea está cotada em 2 valores. **A não obtenção de uma classificação mínima de 8 valores nesta parte implica a reprovação no teste.**

1. Defina uma função `merge :: Ord a => [a] -> [a] -> [a]` que constrói uma lista ordenada a partir de duas listas ordenadas. Por exemplo, `merge [1,3,4,5] [2,4,6] = [1,2,3,4,4,5,6]`.
2. Defina a função `triplos :: [a] -> [(a,a,a)]` que recebe uma lista e vai construindo triplos com os elementos dessa lista (seguindo a mesma ordem em que os elementos estão na lista). Quando já não existirem elementos suficientes para formar triplos, esses elementos são desprezados. Por exemplo:
`triplos "abcdefghijkl" = [('a','b','c'),('d','e','f'),('g','h','i')]`
3. Apresente uma definição alternativa da seguinte função, usando recursividade explícita em vez de funções de ordem superior.

```
fun :: Num a => [a] -> [a]
fun l = map (*2) l
```

4. Considere as seguintes definições de tipos de dados para representar uma base de dados de filmes:

```
type Filme = (Titulo,Realizador,[Actor],Ano,Duracao)
type Titulo = String
type Realizador = String
type Actor = String
type Ano = Int
type Duracao = Int
```

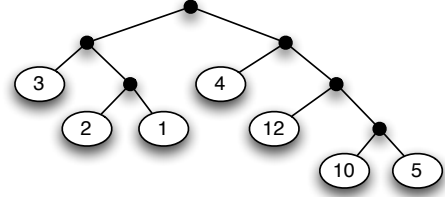
- (a) Defina a função `doActor :: Actor -> [Filme] -> [(Titulo,Ano)]` que lista o nome dos filmes (e o respectivo ano) em que um dado actor participa.
- (b) Defina a função `total :: [Títulos] -> [Filme] -> Int`, que dada uma lista de títulos e a base de dados de filmes, calcula o tempo de duração total dessa lista de filmes. Se um filme não existir na base de dados, a sua duração será 0. Não assuma qualquer tipo de ordenação das listas.

5. Considere o seguinte tipo para representar árvores de folhas:

```
data LTree a = Leaf a | Fork (LTree a) (LTree a)
```

A cada folha de uma árvore pode ser associado o seu "caminho", que não é mais do que uma lista de Booleanos (**False** esquerda e **True** direita). Por exemplo, na árvore da figura temos a seguinte correspondência entre nodos e caminhos:

- O caminho **[False,False]** corresponde à folha 3.
- O caminho **[False,True,True]** corresponde à folha 1.
- O caminho **[True,True]** não corresponde a nenhuma folha da árvore.



Defina uma função **select** :: **LTree a** -> **[Bool]** -> (**Maybe a**) que determina a folha seleccionada por um caminho. Se o caminho não seleccionar nenhuma folha a função deve retornar **Nothing**.

Parte II

1. Recorde a estrutura de dados (árvore de folhas) da alínea 5 da Parte I.

- (a) Defina uma função **procura** :: **Eq a** => **LTree a** -> **a** -> **Maybe [Bool]** que, procura um elemento numa árvore e, em caso de sucesso calcula o caminho correspondente. Por exemplo, se **a** for a árvore representada na figura, **procura a 1 = Just [False,True,True]**.
- (b) Considere a seguinte função que lista as folhas de uma árvore, juntamente com a sua profundidade;

```
travessia :: LTree a -> [(a,Int)]
travessia (Leaf x) = [(x,0)]
travessia (Fork e d) = map (\(x,n) -> (x,n+1)) (travessia e ++ travessia d)
```

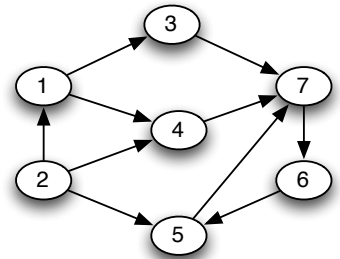
Defina uma função **build** :: **[(a,Int)]** -> **LTree a** inversa da anterior, i.e., tal que, **build (travessia a) = a** para toda a árvore **a**.

2. Uma relação binária entre elementos de um tipo **a** pode ser descrita como um conjunto (lista) de pares **[(a,a)]** ou, agregando todos os pares que têm a primeira componente em comum, por uma lista do tipo **[(a,[a])]**.

```
type RelP a = [(a,a)]
type RelL a = [(a,[a])]
```

Assim, a relação representada ao lado pode ser implementada por

- **[(1,3), (1,4), (2,1), (2,4), 2,5), (3,7), (4,7), (5,7), (6,5), (7,6)] :: RelP Int**
- **[(1,[3,4]), (2,[1,4,5]), (3,[7]), (4,[7]), (5,[7]), (6,[5]), (7,[6])] :: RelL Int**



(a) Considere a seguinte função de conversão entre representações:

```
converteLP :: RelL a -> RelP a
converteLP l = concat (map junta l)
  where junta (x,xs) = map (\y->(x,y)) xs
```

Defina a função de conversão **convertePL** :: (**Eq a**) => **RelP a** -> **RelL a** inversa da anterior, i.e., tal que **convertePL(converteLP l) = l** para todo **l**.

(b) Defina uma função **converso** :: **RelL a** -> **RelL a** que calcula a relação inversa de uma relação. Por exemplo, para o exemplo apresentado a relação inversa deve dar como resultado uma lista com os seguintes elementos:

```
[(1,[2]), (3,[1]), (4,[1,2]), (5,[2,6]), (6,[7]), (7,[3,4,5])]
```