Teste Final de Programação Funcional – 1° Ano, MIEI / LCC / MIEF
11 de Janeiro de 2020 (Duração: 2 horas)

1. Apresente uma definição recursiva de cada uma das seguintes funções (pré-definidas) sobre listas:

   (a) `intersect :: Eq a => [a] -> [a] -> [a]` que retorna a lista resultante de remover da primeira lista os elementos que não pertencem à segunda. Por exemplo, `intersect [1,1,2,3,4] [1,3,5]` corresponde a `[1,1,3]`.

   (b) `tails :: [a] -> [[a]]` que calcula a lista dos sufixos de uma lista. Por exemplo, `tails [1,2,3]` corresponde a `[[1,2,3],[2,3],[3],[]]`.

2. Para armazenar conjuntos de números inteiros, optou-se pelo uso de sequências de intervalos.
   Assim, por exemplo, o conjunto $\{1, 2, 3, 4, 7, 8, 19, 21, 22, 23\}$ poderia ser representado por `[(1,4),(7,8),(19,19),(21,23)]`.

```
type ConjInt = [Intervalo]
type Intervalo = (Int,Int)
```

   (a) Defina uma função `elems :: ConjInt -> [Int]` que, dado um conjunto, dá como resultado a lista dos elementos desse conjunto.

   (b) Defina uma função `geraconj :: [Int] -> ConjInt` que recebe uma lista de inteiros, ordenada por ordem crescente e sem repetições, e gera um conjunto. Por exemplo, `geraconj [1,2,3,4,7,8,19,21,22,23] = [(1,4),(7,8),(19,19),(21,23)]`.

3. Para armazenar uma agenda de contactos telefónicos e de correio electrónico definiram-se os seguintes tipos de dados.
   Não existem nomes repetidos na agenda e para cada nome existe uma lista de contactos.

```
data Contacto = Casa Integer
              | Trab Integer
              | Tlm Integer
              | Email String
    deriving (Show)
type Nome = String
type Agenda = [(Nome, [Contacto])]
```

   (a) Defina a função `acrescEmail :: Nome -> String -> Agenda -> Agenda` que, dado um nome, um email e uma agenda, acrescenta essa informação à agenda.

   (b) Defina a função `verEmails :: Nome -> Agenda -> Maybe [String]` que, dado um nome e uma agenda, retorna a lista dos emails associados a esse nome. Se esse nome não existir na agenda a função deve retornar `Nothing`.

   (c) Defina a função `consulta :: [Contacto] -> ([Integer],[String])` que, dada lista de contactos, retorna o par com a lista de números de telefone (tanto telefones fixos como telemóveis) e a lista de emails, dessa lista. Implemente a função de modo a fazer uma única travessia da lista de contactos.

   (d) Defina a função `consultaIO :: Agenda -> IO ()` que, dada uma agenda, lê do teclado o nome que pretende consultar e apresenta no ecran os contactos associados a esse nome na agenda.

4. Relembre o tipo `RTree a` definido nas aulas.

```
data RTree a = R a [RTree a] deriving (Show, Eq)
```

   (a) Defina a função `paths :: RTree a -> [[a]]` que dada uma destas árvores calcula todos os caminhos desde a raíz até às folhas. Por exemplo, `paths (R 1 [R 2 [], R 3 [R 4 [R 5 [], R 6 []]], R 7 []])` deve corresponder à lista `[[1,2],[1,3,4,5],[1,3,4,6],[1,7]]`.

   (b) Defina a função `unpaths :: Eq a => [[a]] -> RTree a` inversa da anterior, i.e., tal que `unpaths (paths t) == t`, para qualquer árvore `t :: Eq a => RTree a`.