Época Especial de Programação Funcional – 1º Ano, MIEI / LCC / MIEF

23 de Junho de 2021 (Duração: 2 horas)

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

   (a) `posImpares :: [a] -> [a]` que determina os elementos de uma lista que ocorrem em posições impares. Considere que o primeiro elemento da lista ocorre na posição 0 e por isso é par. Por exemplo, `posImpares [10,11,7,5]` corresponde a `[11,5]`.

   (b) `isPrefixOf :: Eq a => [a] -> [a] -> Bool` que testa se uma lista é prefixo de outra. Por exemplo, `isPrefixOf [10,20] [10,20,30]` corresponde a `True` enquanto que `isPrefixOf [10,30] [10,20,30]` corresponde a `False`.

2. Considere a sequinte definição para representar matrizes: `type Mat a = [[a]]`

   Por exemplo, a matriz $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$ seria representada por `[[1,2,3], [0,4,5], [0,0,6]]`

   (a) Defina a função `zeros :: Num a => Mat a -> Int` que conta o número de zeros da matriz.

   (b) Defina a função `addMat :: Num a => Mat a -> Mat a -> Mat a` que adiciona duas matrizes.

   (c) Defina a função `transpose :: Mat a -> Mat a` que calcula a transposta de uma matriz.

3. Relembre o tipo `BTree a` definido como `data BTree a = Empty | Node a (BTree a) (BTree a)`

   (a) Defina a função `replace :: Eq a => BTree a -> a -> a -> BTree a`, que recebe uma árvore binária, um valor a substituir, e o valor pelo qual se deve substituir, e que irá devolver uma nova árvore binária em que todas as ocorrências do valor a substituir são alteradas pelo novo valor.

   (b) Considere agora que se usa este tipo para guardar informação sobre uma turma (o número e o nome de cada aluno) numa *árvore binária de procura*. Defina a função `insere :: Integer -> String -> BTree (Int,String) -> BTree (Int,String)` que insere na árvore informação de um novo aluno. Se o número já existir actualiza o nome.

4. Considere a seguinte estrutura para manter um dicionário, onde as palavras estão organizadas de forma alfabética.
   Cada árvore agrupa todas as palavras começadas numa dada letra. As palavras constroem-se descendo na árvore a partir da raiz. Quando uma palavra está completa, o valor associado à última letra é `Just s`, sendo `s` uma string com a descrição da palavra em causa (que corresponde ao caminho desde a raiz até aí). Caso contrário é `Nothing`.
   Por exemplo, `d1` é um dicionário com as palavras: *cara*, *caras*, *caro* e *carro*.

```
data RTree a = R a [RTree a]
type Dictionary = [ RTree (Char, Maybe String) ]

d1 = [R ('c',Nothing) [
        R ('a',Nothing) [
            R ('r',Nothing) [
                R ('a',Just "...") [
                    R ('s',Just "...") []
                    ],
                R ('o',Just "...") [],
                R ('r',Nothing) [
                    R ('o',Just "...") []
] ] ]   ]   ]   ]
```

   (a) Defina a função `consulta :: String -> Dictionary -> Maybe String`, que dá a informação associada a uma palavra no dicionário, caso ela exista.

   (b) Defina a função `palavras :: Dictionary -> [String]` que devolve a lista de palavras (completas) que existem no dicionário.

   (c) Defina a função `apresenta :: Dictionary -> IO ()` que, pede ao utilizador para inserir uma letra, e lista no ecrã todas as palavras (completas) começadas por essa letra e a informação associada que está no dicionário. Apresente uma palavra por linha.