Exame de Programação Funcional – 1° Ano, MIEI / LCC / MIEF
25 de Janeiro de 2019 (Duração: 2 horas)

1. Apresente uma definição recursiva das seguintes funções sobre listas:

   (a) `isSorted :: (Ord a) => [a] -> Bool` que testa se uma lista está ordenada por ordem crescente.

   (b) `inits :: [a] -> [[a]]` que calcula a lista dos prefixos de uma lista. Por exemplo, `inits [11,21,13]` corresponde a `[[],[11],[11,21],[11,21,13]]`.

2. Defina `maximumMB :: (Ord a) => [Maybe a] -> Maybe a` que dá o maior elemento de uma lista de elementos do tipo `Maybe a`. Considere `Nothing` o menor dos elementos.

3. Considere o seguinte tipo para representar árvores em que a informação está nas extermidades:

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

   (a) Defina a função `listaLT :: LTree a -> [a]` que dá a lista das folhas de uma árvore (da esquerda para a direita).

   (b) Defina uma instância da classe `Show` para este tipo que apresente uma folha por cada linha, precedida de tantos pontos quanta a sua profundidade na árvore. Veja o exemplo ao lado.

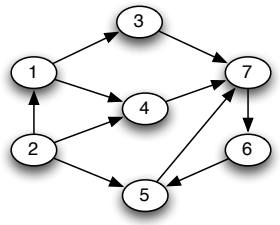
   ```
   > Fork (Fork (Tip 7) (Tip 1)) (Tip 2)
   ..7
   ..1
   .2
   ```

4. Utilizando uma função auxiliar com acumuladores, optimize a seguinte definição que determina a soma do segmento inicial de uma lista com soma máxima.

   ```
   maxSumInit :: (Num a, Ord a) => [a] -> a
   maxSumInit l = maximum [sum m | m <- inits l]
   ```

5. Uma relação binária entre elementos de um tipo `a` pode ser descrita como um conjunto (lista) de pares `[(a,a)]`. Outras formas alternativas consistem em armazenar estes pares agrupados de acordo com a sua primeira componente. Considere os seguintes três tipos para estas representações.

   ```
   type RelP a = [(a,a)]
   type RelL a = [(a,[a])]
   type RelF a = ([a], a->[a])
   ```

   Por exemplo, a relação representada na figura 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`
   - `([1,2,3,4,5,6,7],f) :: RelF Int`, em que `f` é uma função tal que `f 1 = [3,4]`, `f 2 = [1,4,5]`, `f 3 = [7]`, `f 4 = [7]`, `f 5 = [7]`, `f 6 = [5]`, e `f 7 = [6]`.

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

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

   Defina a função de conversão `convPL :: (Eq a) => RelP a -> RelL a`, inversa da anterior. Isto é, tal que `convPL (convLP r) = r`, para todo o `r`.

   (b) Defina a função `criaRelPint :: Int -> IO (RelP Int)`, tal que `criaRelPint n` permite ao utilizador criar (interactivamente) uma relação de inteiros com `n` pares.

   (c) Defina as funções de conversão entre as representações `RelF a` e `RelP a`,

   i. `convFP :: (Eq a) => RelF a -> RelP a`

   ii. `convPF :: (Eq a) => RelP a -> RelF a`

   tal que `convFP (convPF r) = r`, para todo o `r`.