IF686 - 2020.3
Lista de exercícios
Tipos Algébricos

1. Implemente uma função para divisão segura, ou seja, retorna um resultado com Just normalmente, mas Nothing se o divisor for zero

2. Uma variação da divisão segura define que um resultado bem sucedido deve ser retornado como Right resultado, enquanto a divisão por zero deve retornar Left "123/0" (substitua x pelo valor do argumento x)

3. Implemente a função mapMaybe que se comporta como map e filter . Os argumentos são uma lista ([a]) e uma função de tipo a -> Maybe b. A função é aplicada a todos os valores da lista. Caso retorne Just x, x estará na lista resultante; se a função retornar Nothing, nada será adicionado à lista resultante.

   Exemplos

   ```
   let f x = if x>0 then Just (2*x) else Nothing
    in mapMaybe f [0,1,-1,4,-2,2]                    ==> [2,8,4]

   mapMaybe Just [1,2,3]   ==> [1,2,3]

   mapMaybe (\x -> Nothing) [1,2,3]   ==> []
   ```

4. Defina a função classifica que recebe uma lista de valores Either a b e retorna uma lista de valores Left e uma lista de valores Right

   ```
   classifica  [Left 1, Right True, Left 0, Right False]
     ==> ([1,0],[True,False])
   ```

5. Nas próximas questões, use um tipo para árvore binária como este abaixo.

   ```
   data Tree a = Leaf | Node a (Tree a) (Tree a)
               deriving (Show, Eq)
   ```

   (a) Imprlemente a função valAtRoot que retorna o valor na raiz da árvore. O valor retornado é do tipo Maybe a pois a árvore pode estar vazia.

   (b) Implemente uma função que computa o tamanho de uma árvore, ou seja, o número de nós da árvore.

   (c) Devolva o valor mais à esquerda na árvore. O valor retornado é do tipo Maybe a, pois a árvore pode estar vazia.

   Exemplos

```
leftest Leaf    ==> Nothing
leftest (Node 1 (Node 2 (Node 3 Leaf Leaf) Leaf) Leaf)
   ==> Just 3
leftest (Node 1 (Node 2 Leaf (Node 3 Leaf Leaf)) (Node 4 Leaf Leaf))
   ==> Just 2
```

_

(d) Implemente a função map para árvores.

Exemplos

```
mapTree (+1) Leaf  ==>  Leaf
mapTree (+2) (Node 0 (Node 1 Leaf Leaf) (Node 2 Leaf Leaf))
 ==> (Node 2 (Node 3 Leaf Leaf) (Node 4 Leaf Leaf))
```

(e) Defina uma função que insere um valor na posição mais à esquerda possível. Deve-se retornar uma nova árvore.

Exemplos

```
insertL 0 Leaf
   ==> Node 0 Leaf Leaf
insertL 0 (Node 1 Leaf Leaf)
   ==> Node 1 (Node 0 Leaf Leaf) Leaf)

insertL 0 (Node 1
             (Node 2
               Leaf
               (Node 3 Leaf Leaf))
             (Node 4 Leaf Leaf))

         ==> Node 1
             (Node 2
               (Node 0 Leaf Leaf)
               (Node 3 Leaf Leaf))
             (Node 4 Leaf Leaf)
```

(f) Implemente a função medida que recebe um árvore como entrada e retorna uma árvore com o mesmo formato, mas com o valor em cada nó sendo o tamanho da subárvore começando naquele nó

Exemplos

```
medida (Node 'a' Leaf Leaf)
 ==> Node 1 Leaf Leaf
medida (Node 'a' (Node 'b' Leaf Leaf) Leaf)
 ==> Node 2 (Node 1 Leaf Leaf) Leaf
medida (Node 0 (Node 0 Leaf Leaf) Leaf)
 ==> Node 2 (Node 1 Leaf Leaf) Leaf
```

```
medida (Node 0 (Node 0 Leaf Leaf)
                (Node 0 (Node 0 Leaf Leaf)
                        (Node 0 Leaf
                                (Node 0 Leaf Leaf))))
  ==> Node 6 (Node 1 Leaf Leaf)
             (Node 4 (Node 1 Leaf Leaf)
                     (Node 2 Leaf
                             (Node 1 Leaf Leaf)))
```

(g) A função foldr :: (a -> b -> b) -> b -> [a] -> b é usada para comprimir uma lista em um único valor. Implemente a função foldTree que funciona como foldr, mas para árvores

Exemplo

```
foldTree f l (Node 3 Leaf Leaf)
  ==> f 3 l l
foldTree f l (Node 'a' (Node 'b' (Node 'c' Leaf Leaf)
                                 Leaf)
                       (Node 'd' Leaf Leaf))
  ==> f (f 'a' (f 'b' (f 'c' l l)
                      l)
               (f 'd' l l))
```

(h) Com base nas funções foldTree e sumt

```
sumt :: Int -> Int -> Int -> Int
sumt x y z = x+y+z
```

defina a função treeSum