2º Teste

Programação Funcional – 1° Ano, LEI / LCC / MIEF
21 de Janeiro de 2014

Duração: 90 min

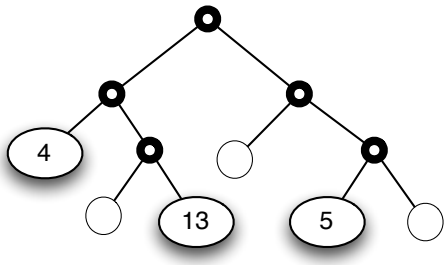
Considere o seguinte tipo para representar árvores:

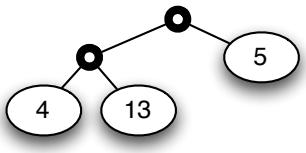
```
data Tree a = Empty
            | Leaf a
            | Fork (Tree a) (Tree a)
```

Neste tipo as definições

```
a1 = Fork (Fork (Leaf 4)
                (Fork Empty (Leaf 13)))
          (Fork Empty
                (Fork (Leaf 5) Empty))
a2 = Fork (Fork (Leaf 4) (Leaf 13))
          (Leaf 5)
```

correspondem às árvores apresentadas à direita.





1. Defina `Tree a` como uma instância da classe `Show` de forma a que `show a1` produza a string

   ```
   "((4 <-*-> (<> <-*-> 13)) <-*-> (<> <-*-> (5 <-*-> <>)))"
   ```

2. Defina a função `ultimo :: Tree a -> Maybe a` que calcula o elemento mais à direita de uma árvore. Se não existirem elementos a função deverá retornar `Nothing`. Por exemplo, para as árvores acima, a função deverá dar como resultado `Just 5`

3. Defina a função `apaga :: Eq a => a -> Tree a -> Tree a` que apaga todas as ocorrências de um dado elemento numa árvore (substituindo por `Empty`).

4. Defina a função `limpa :: Tree a -> Tree a` que remove todos os `Empty` de uma árvore não vazia. A função deverá retornar `Empty` sempre que não exista nenhum elemento na árvore. Por exemplo, `limpa a1` deve construir a árvore `a2`.

5. Defina uma função `randomRemove :: Tree a -> IO (Tree a)` que recebe uma árvore com pelo menos uma folha e retorna o resultado de remover (substituindo por `Empty`) uma folha aleatória dessa árvore. *Sugestão:* use a função `randomRIO :: Random a => (a,a) -> IO a` para gerar o número de ordem do elemento a remover.