Ejercicio Integrador

Programación Funcional, Universidad Nacional de Quilmes

Three Tree

Un `ThreeT` es un árbol cuyos elementos se encuentran sólo en las hojas y cada nodo se bifurca en tres ramas.

Su definición es la siguiente:

```
data ThreeT a = Leaf a | Branch a (ThreeT a) (ThreeT a) (ThreeT a)
```

1. Definir las siguientes funciones usando recursión explícita sobre la estructura de `ThreeT`:
   a) `sizeTT :: ThreeT a -> Int`
   b) `sumTT :: ThreeT Int -> ThreeT Int`
   c) `leavesTT :: ThreeT a -> [a]`
   Devuelve los elementos que están en las hojas
   d) `mapTT :: (a -> b) -> ThreeT a -> ThreeT b`
   e) `maxTT :: Ord a => ThreeT a -> a`
   f) `findTT :: Eq a => (a -> Bool) -> ThreeT (a,b) -> Maybe b`
   g) `levelNTT :: Int -> ThreeT a -> [a]`
   h) `listPerLevelTT :: ThreeT a -> [[a]]`
   Devuelve una lista de listas donde en cada lista están los elementos de cada nivel.
2. Dar una definición de `fold` (llamada `foldTT`) en base a la estructura del tipo `ThreeT`.
3. Definir las funciones del primer punto usando la definición dada de `fold` para `ThreeT`.
4. Demostrar las siguientes equivalencias usando las funciones definidas en el punto 1.
   a) `sizeTT = sumTT . mapTT (const 1)`
   b) `sum . leavesTT = sumTT`
   c) `sizeTT . mapTT f . mapTT g = sizeTT . mapTT (f . g)`
   d) `maximum . leavesTT = maxTT`