



Nombre y Apellido:

Legajo:

Examen Final

1. Se representan secuencias mediante árboles binarios dados por el siguiente tipo de datos:

`data BTree a = E | L a | N Int (BTree a) (BTree a)`

donde el recorrido *inorder* del árbol da el orden de los elementos de la secuencia y el valor entero guardado en los nodos representa la longitud de la misma.

Definir en Haskell, de manera eficiente, las siguientes funciones:

- a) `mapIndex :: (a → Int → b) → BTree a → BTree b`, que dada una función f y una secuencia s aplique f a cada elemento de la secuencia y a su posición. Es decir, `mapIndex f ⟨x0, x1, ..., xn⟩ = ⟨f x0 0, f x1 1, ..., f xn n⟩`.
- b) `fromSlow :: Int → Int → Int → BTree Int`, que dados tres naturales n , m y k , construya una secuencia de m elementos, comenzando por n y sumando 1 cada k elementos.

Por ejemplo,

`fromSlow n 10 3 = ⟨n, n, n, n + 1, n + 1, n + 1, n + 2, n + 2, n + 2, n + 3⟩`
`fromSlow n 10 1 = ⟨n, n + 1, n + 2, n + 3, n + 4, n + 5, n + 6, n + 7, n + 8, n + 9⟩`

Ayuda : Usar la función `mapIndex`.

2. Dada una secuencia cronológica de ventas realizadas en diferentes horarios y un monto de pesos X , se desea determinar la hora en que las ventas acumuladas alcanzaron el valor X .

Escribir una función `alarma :: Seq (Pesos, Hora) → Pesos → Maybe Hora` que calcule la hora en que se llega al monto X o retorne *Nothing* si nunca se acumuló ese monto. Suponer que se definieron los tipos `Pesos` y `Hora`.

Definir la función usando la función `scan`, con profundidad de orden $O(\lg n)$, donde n es el largo de la secuencia. Justificar los costos.

Por ejemplo:

`alarma ((22, "8:00"), (30, "10:00"), (20, "10:40"), (33, "11"), (40, "11:20")) 100 = Just "11"`

3. Dado el siguiente tipos de datos usado para representar árboles generales:

`data GTree a = Node a [GTree a]`

y las siguientes definiciones:

`elem x [] = False`
`elem x (y : ys) = x == y ∨ elem x ys`
`concatMap f = concat ∘ map f`
`allNodes (Node n xs) = n : concatMap allNodes xs`
`isNode y (Node n xs) = y == n ∨ (map (isNode y) xs)`

Probar por inducción estructural sobre t que:

`elem n (allNodes t) = isNode n t`

para cualquier n .

Ayuda: Puede usar la siguiente propiedad `elem x (concat xss) = or (map (elem x) xss)`, sin demostrarla.