



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, la función `stripSuffix :: Eq a => BTree a -> BTree a -> Maybe (BTree a)`, que elimine el sufijo dado en una secuencia, devolviendo `Nothing` si la secuencia no termina con el sufijo dado o `Just xs` donde `xs` es la secuencia que queda antes del sufijo.

Por ejemplo,

```
stripSuffix('b', 'a', 'r')('f', 'o', 'o', 'b', 'a', 'r') = Just('f', 'o', 'o')
stripSuffix('b', 'a', 'r')('f', 'o', 'o', 'b', 'a', 'r', 'r') = Nothing
```

2. Cierta periódico quiere manipular los resultados de las elecciones de las paso en Argentina asociando los resultados por ciudades cercanas de manera conveniente a un partido X.

Para ello, dada una secuencia `s` con los balances de votos de un partido X, se desea encontrar la suma de una subsecuencia de `s` que maximice las sumas de dichos balances. Definir una función `maxBalance :: Seq Int -> Int` que resuelva el problema.

Por ejemplo,

```
maxBalance(-2, -4, 6, -1, 5, -7, 2) = 10
```

Definir `maxBalance` usando funciones del TAD secuencias, con profundidad de orden $\lg n$.

3. Dado el TAD `Array`, con las siguientes operaciones:

```
tad Array (V : Set) where
import Int, Bool
new    : Int -> V -> Array V           -- dados n y v crea un arreglo de dimension n e inicializa
                                           -- cada elemento con v
set    : Int -> V -> Array V -> Array V -- modifica el valor asociado a un índice del arreglo
dimension : Array V -> Int
get     : Int -> Array V -> V           -- obtiene el valor almacenado en un índice del arreglo
slicing : Int -> Int -> Array V -> Array V -- dados i y j enteros y un arreglo a, crea un arreglo con
                                           -- valores a[i] .. a[j]
```

Dar una especificación algebraica para el TAD.

4. (Lólo libres) Dadas las siguientes definiciones:

```
data Rose a = E | N a [Rose a]
flatten :: Rose a → [a]
flatten E = []
flatten (N x xs) = x : concat (map flatten xs)
mapRose :: (a → b) → Rose a → Rose b
mapRose f E = E
mapRose f (N x xs) = N (f x) (map (mapRose f) xs)
```

probar por inducción estructural que $\text{map } f \circ \text{flatten} = \text{flatten} \circ \text{mapRose } f$

Ayuda: Puede utilizar el siguiente lema: $\text{concat} \circ \text{map} (\text{map } f) = \text{map } f \circ \text{concat}$