Árboles

Taller de Álgebra I

Segundo cuatrimestre de 2016

Nuevas estructuras recursivas

En clases anteriores definimos una estructura de datos equivalente a la lista `[]` de Haskell.

```
data Lista a = ListaVacia | Agregar a (Lista a)
```

¿Qué otras estructuras podemos construir con tipos de datos recursivos?

Árboles estrictamente binarios de enteros

Un **árbol estrictamente binario de enteros** es una estructura muy utilizada en computación y matemática. Veamos cómo podemos representarlos:

```
data Arbol = Hoja Integer | Rama Arbol Integer Arbol
```

Ejercicio

Determinar el tipo de las siguientes expresiones:

- `Hoja 10`
- `Rama (Hoja 20) 10 (Hoja 30)`
- `[Rama (Hoja 2) 5 (Rama (Hoja 1) 10 (Hoja 0)), Hoja 0]`
- `Rama (Hoja 3) 5 (Rama (Hoja 1))`

Más sobre los árboles

Árboles estrictamente binarios de enteros

```
data Arbol = Hoja Integer | Rama Arbol Integer Arbol
```

Si lo necesitan, puede agregar `deriving (Eq, Show)` al final de la definición.

Ejercicios

Implementar las siguientes funciones:

- `esHoja :: Arbol -> Bool`, que determina si un árbol es o no una hoja.
- `sumaNodos :: Arbol -> Integer`, que devuelve la suma de los valores del árbol.
- `altura :: Arbol -> Integer`, que devuelve la altura de un árbol.
- `pertenece :: Integer -> Arbol -> Bool`, que indica si un elemento pertenece o no a un árbol.
- dado el tipo `data Dir = Der | Izq`,
  implementar `busqueda :: [Dir] -> Arbol -> Integer`,
  que recorre el árbol siguiendo la lista de instrucciones y devuelve el valor que se encuentre luego de recorrerlo (asumir que la lista lleva a un elemento y no se termina el árbol antes de encontrarlo).

Árboles genéricos

¿Y si quisiera que mis árboles sean de `Char`, o de `String`?

¿Y qué tal si usamos tipos genéricos?

```
data Arbol t = Hoja t | Rama (Arbol t) t (Arbol t)
```

Estamos definiendo infinitos tipos (uno por cada posible tipo `t`).

Ejemplos

- `Hoja 20 :: Arbol Integer`
- `Rama (Hoja 10) 2 (Hoja 10) :: Arbol Integer`
- `Rama (Hoja 'b') 'a' (Hoja 'c') :: Arbol Char`
- `Rama (Hoja "10") "Algebra" (Hoja "10") :: Arbol String`
- `Rama (Rama (Hoja 10) 2 (Hoja 10)) 6 (Hoja 10) :: Arbol Integer`
- `Rama (Hoja (Hoja 10)) (Rama (Hoja 10) 2 (Hoja 10)) (Hoja (Hoja 10)) :: Arbol (Arbol Integer)`

Ejercicios

Implementar las siguientes funciones:

1. `esHoja :: Arbol a -> Bool`
   que determina si el árbol es una hoja.
2. `cantHojas :: Arbol a -> Integer`
   que calcula la cantidad de hojas que tiene un árbol.
3. `maximo :: Ord a => Arbol a -> a`
   que devuelve el máximo elemento de un árbol de elementos con orden.
4. `raiz :: Arbol a -> a`
   que devuelve el valor del nodo principal del árbol.
5. `todosIguales :: Eq a => Arbol a -> Bool`
   que determina si todos los nodos del árbol tienen el mismo valor.
6. `espejar :: Arbol a -> Arbol a`
   que invierte el árbol de manera que esté espejado.
7. `esHeap :: Ord a => Arbol a -> Bool`
   que valga verdadero en un árbol si cada nodo (salvo la raíz) es mayor o igual que su padre.