Trabajo Práctico # 2

Programación Funcional
Tecnicatura en Programación Informática
Universidad Nacional de Quilmes

1. Árboles

Ejercicio 1

Defina las siguientes funciones sobre árboles binarios:

1. `sizeT`: dado un árbol, devuelve la cantidad de elementos.
2. `leaves`: dado un árbol, devuelve el número de hojas.
3. `nodes`: dado un árbol, devuelve el número de nodos que no son hojas.
4. `mirrorT`: dado un árbol, retorna su imagen especular; esto es, el árbol resultante de intercambiar el hijo izquierdo con el derecho de cada nodo.
5. `levelT`: dado un árbol y un número n, devuelve una lista con los nodos de nivel n.
6. `traverseInOrder`: dado un árbol, devuelve una lista que representa el resultado de recorrerlo en modo *in-order*.
7. `traversePreOrder`: dado un árbol, devuelve una lista que representa el resultado de recorrerlo en modo *pre-order*.
8. `treeToListPosOrder`: dado un árbol, devuelve una lista que representa el resultado de recorrerlo en modo *post-order*.
9. `treeToListPerLevel`: dado un árbol, devuelve una lista de listas en la que cada elemento representa un nivel de dicho árbol.
10. `occursT`: dado un árbol y un elemento, devuelve si ese elemento pertenece a el árbol o no.
11. `minT`: dado un árbol, devuelve su elemento mínimo.
12. `maxT`: dado un árbol, devuelve su elemento máximo.
13. `heightT`: dado un árbol, devuelve su altura.
    Nota: la altura de un árbol es la cantidad máxima de nodos entre la raíz y alguna de sus hojas.
14. `widthT`: dado un árbol, devuelve su ancho.
    Nota: el ancho de un árbol es el máximo de las cantidades de elementos de cada nivel.
15. `balanced`: dado un árbol, retorna si está balanceado.
    Nota: un árbol está balanceado si la distancia entre su raíz y su hoja más lejana menos la distancia de su raíz y su hoja más cercana es a lo sumo 1.
16. `heightBalanced`: dado un árbol, retorna si está balanceado en altura.
    Nota: un árbol está balanceado en altura si para todo nodo, la diferencia de altura entre sus subárboles izquierdo y derecho es a lo sumo 1.

17. `weightBalanced`: dado un árbol, retorna si el árbol recibido está balanceado en peso.
Nota: un árbol está balanceado en peso si para todo nodo, la diferencia entre la cantidad de nodos de sus subárboles izquierdo y derecho es a lo sumo 1.

Ejercicio 2

Defina las siguientes funciones sobre árboles binarios:

1. `sumarN`: recibe un árbol de números y un número n y devuelve otro árbol que resulta de sumarle n a cada uno de los nodos del árbol recibido.
2. `siguienteColorT`: recibe un árbol de colores, y devuelve otro cambiando cada color por el siguiente.
3. `direccionOpuestaT`: recibe un árbol de direcciones, y devuelve otro cambiando cada dirección por la opuesta.
4. `suma`: recibe un árbol de números y calcula la suma de todos sus nodos.
5. `mayorQueHijos`: recibe un árbol de enteros y devuelve uno de booleanos donde cada nodo del árbol nuevo es verdadero si el nodo correspondiente en el árbol original es mayor o igual a la suma de sus hijos.
6. `podarNegro`: recibe un árbol de colores y poda todos los subárboles cuya raíz sea `Negro`.
7. `estaOrdenado`: recibe un árbol de números y devuelve verdadero si para cada nodo, todos los nodos de su hijo izquierdo son menores que él y todos los nodos de su hijo derecho mayores que él.

Ejercicio 3

Sea el tipo `Exp`, modelando expresiones aritméticas:

```
data Exp = Const Int | Var String | UnExp UnOp Exp | BinExp BinOp Exp Exp
data UnOp = Neg
data BinOp = Add | Sub | Mul | Div
type Env = [(String,Int)]
```

Implementar las siguientes funciones:

1. `eval`, que dada una expresión y una lista relacionando nombres de variables con su valor (de tipo `Env`), evalúe la expresión y retorne su valor. ¿Qué casos hacen que `eval` sea una función parcial? Implementarla como función total mediante el tipo `Maybe`. Esto es, `eval` es de tipo $Exp \to Env \to Maybe\ Int$.
2. `simplify`, que simplifique una expresión aritmética según los siguientes criterios:
   a) $0 + x = x + 0 = x$
   b) $x - 0 = x$
   c) $0 - x = -x$
   d) $x \times 1 = 1 \times x = x$
   e) $x \times 0 = 0 \times x = 0$
   f) $x \div 1 = x$
   g) $0 \div x = 0,\ x \neq 0$

h) Calcular el resultado de la expresión si todos los operandos son constantes (este tipo de simplificación se denomina *constant folding*).

Agregamos ahora dos nuevos operadores unarios a nuestro tipo de expresiones:

```
data UnOp = Neg | Inc | Dec  -- Inc: incremento, Dec: decremento
```

Reimplementar `eval` y `simplify` incluyendo el manejo de estas nuevas operaciones unarias. Para `simplify`, considerar estos criterios de simplificación adicionales:

1. Reemplazar $x + 1 = 1 + x$ por `Inc x`
2. Reemplazar $x - 1$ por `Dec x`
3. Reemplazar $1 - x$ por `-(Dec x)`