Tarea 3: Listas y Árboles
Puntaje máximo: 8 puntos

Entrega por Aulas: 10 de junio de 2024 hasta las 21:00 hrs

IMPORTANTE: Deberá subirse a Aulas un único archivo de Haskell (`.hs`) con los ejercicios resueltos, aquellos que no son de programar funciones se entregará como código comentado ({`-- --`}). El archivo debe incluir el nombre y número de estudiante al principio del mismo.

?1.

1. Defina la función `prefijo :: (a -> Bool) -> [a] -> [a]` que reciba un predicado `p` y una lista `xs` y retorne el prefijo `ys` más largo de `xs`, tal que todos los elementos de `ys` cumplen con `p`.
   Ejemplo: `prefijo` (<5) [1,4,5,2] = [1,4]
2. Defina la función `sufijo :: (a -> Bool) -> [a] -> [a]` que devuelve el sufijo restante de la precedente.
   Ejemplo: `sufijo` (<5) [1,4,5,2] = [5,2]
3. Demuestre por inducción que $(\forall xs :: [a])(\forall p :: (a -> Bool))\ ((prefijo\ p\ xs)\ ++\ (sufijo\ p\ xs) = xs)$

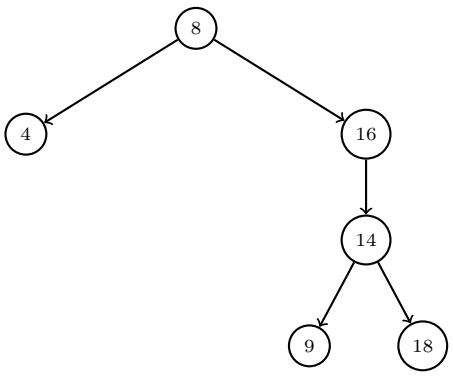
?2.

1. Defina la función `incluido :: Eq a => [a] -> [a] -> Bool`, que dadas dos listas `l1` y `l2`, retorna `True` si todos los elementos de la lista `l1` se encuentran también en l2. Puede hacer uso de la función `elem :: Eq a => a -> [a] -> Bool` del preludio de Haskell.
   Ejemplos:
   `incluido` [1,2] [1,4,5,2] = True
   `incluido` [1,2,3] [1,4,5,2] = False
2. Defina la función `interseccion :: Eq a => [a] -> [a] -> [a]`, que dadas dos listas `l1` y `l2`, devuelve una nueva lista que contiene sólamente los elementos que ambas listas tienen en común (sin elementos repetidos). Puede hacer uso de la función `elem :: Eq a => a -> [a] -> Bool` del preludio de Haskell.
   Ejemplo: `interseccion` [1,2,6,2] [1,4,5,2] = [1,2]
3. Demuestre por inducción que $(\forall\ l1, l2 :: [a])\ incluido\ (interseccion\ l1\ l2)\ \ l2\ =\ True$

?3. Considere la siguiente definición de árboles.

data Tree = L **Int** | U **Int** Tree | B Tree **Int** Tree

1. Codifique el siguiente árbol como una expresión `t :: Tree`.

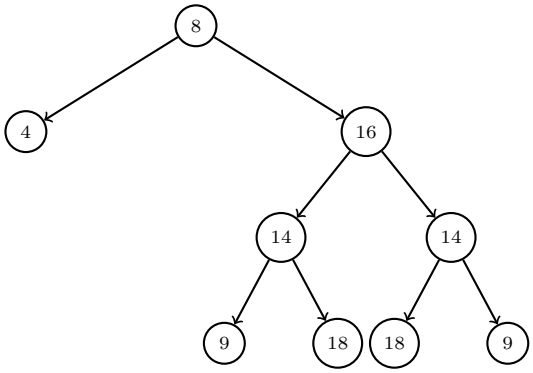


2. Defina la función `listarElems :: Tree -> [Int]`, que dado un árbol lista la información de todos los nodos.
   Ejemplo: `listarElems t = [8,4,16,14,9,18] (en este o cualquier otro orden)`.

3. Defina la función `esBinario :: Tree -> Bool`, que verifica si todos los nodos del árbol son binarios.
   Ejemplo: `esBinario t = False`.

4. Defina la función `espejo :: Tree -> Tree`, que recibe un árbol y lo devuelve espejado.

5. Defina la función `convertirEnBinario :: Tree -> Tree`, que recibe un árbol y lo convierte de modo que los nodos unarios se transforman en binarios manteniendo el subárbol actual como el izquierdo y agregando un nuevo subárbol a la derecha espejando el subárbol original.
   Por ejemplo, `convertirEnBinario t` da como resultado el siguiente árbol:



6. Demuestre por inducción que `(∀t::Tree) esBinario (convertirEnBinario t) = True`
   Puede hacer uso del siguiente lema: L1. `(∀t::Tree) esBinario t => esBinario (espejo t)`