

Tarea 3: Listas y Árboles

Puntaje máximo: 8 puntos

Entrega por Aulas: 23 de noviembre de 2025 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 `unir :: [a] -> [a] -> [a]`, que dadas dos listas `l1` y `l2`, devuelve una nueva lista que contiene los elementos de `l1` y `l2` manteniendo el orden, estando primero los elementos de `l1` y luego los de `l2`.
Ejemplos:
`unir [2,8] [0,9] = [2,8,0,9]`
`unir [] [8,11] = [8,11]`
2. Defina la función `producto :: [Int] -> Int`, que dada una lista de enteros, devuelve el resultado de aplicarle la función `producto (*)` a todos los elementos de la lista.
Ejemplos:
`producto [] = 1`
`producto [1,3,4] = 1*3*4 = 12`
3. Demuestre por inducción que $(\forall l1, l2 :: [Int])(producto(unir\ l1\ l2) = (producto\ l1)*(producto\ l2))$

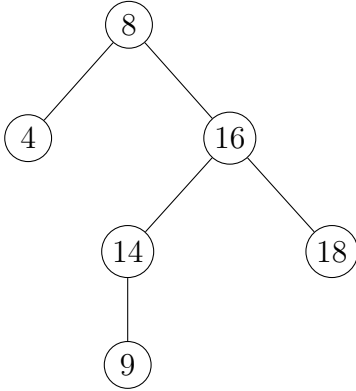
?2.

1. Defina la función `cumplen :: (a -> Bool) -> [a] -> [a]`, que reciba un predicado `p` y una lista `xs` y retorna una lista con todos los elementos que cumplen con `p`.
Ejemplos:
`cumplen even [2,3,5,7,8,9,11,12] = [2,8,12]`
`cumplen (>2) [4,2,-3,1,5,6] = [4,5,6]`
2. Defina la función `descartar :: (a -> Bool) -> [a] -> [a]`, que reciba un predicado `p` y una lista `xs` y retorna una lista con todos los elementos que no cumplen con `p`.
Ejemplos:
`descartar even [2,3,5,7,8,9,11,12] = [3,5,7,9,11]`
`descartar (>2) [4,2,-3,1,5,6] = [2,-3,1]`
3. Usando la siguiente definición de `length`:
`length :: [a] -> Int`
`length [] = 0`
`length (.:xs) = 1 + length xs`
4. Demuestre por inducción que:
 $(\forall xs :: [a])(\forall p :: (a \rightarrow Bool)) (length(cumplen\ p\ xs) + length(descartar\ p\ xs) = length\ xs)$

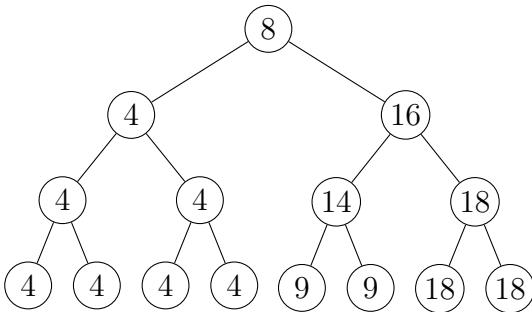
?3. Considere la siguiente definición de árboles.

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

1. Codifique el siguiente árbol como una expresión $t :: \text{Tree}$.



2. Defina la función $\text{sumTree} :: \text{Tree} \rightarrow \text{Int}$, que retorna la suma de todos los enteros del árbol. Ejemplo: $\text{sumTree } t = 69$
3. Defina la función $\text{treeToList} :: \text{Tree} \rightarrow [\text{Int}]$, que convierte el árbol en una lista siguiendo el recorrido **inorder**.
Ejemplo: $\text{treeToList } t = [4, 8, 9, 14, 16, 18]$ (en ese orden).
4. Defina la función $\text{treeHeight} :: \text{Tree} \rightarrow \text{Int}$, que retorna la altura del árbol, o sea la cantidad de niveles que posee. Ejemplo: $\text{treeHeight } t = 4$
5. Defina la función $\text{completeLevel} :: \text{Tree} \rightarrow \text{Int} \rightarrow \text{Tree}$, que recibe un árbol y su altura, y retorna el árbol completando los nodos/hojas faltantes para que sea un árbol binario completo, es decir, todos los niveles completamente llenos.
Ejemplo: $\text{completeLevels } t =$



6. Demuestre por inducción que $(\forall t :: \text{Tree})(\text{treeHeight}(\text{completeLevels } t) = \text{treeHeight } t)$