

Ejercicio 1 ★

Considerar las siguientes definiciones de funciones:

```

• - max2 (x, y) | x >= y = x
  | otherwise = y
• - normaVectorial (x, y) = sqrt (x^2 + y^2)
• - subtract = flip (-)
• - predecesor = subtract 1
  
```

```

• - evaluarEnCero = \f -> f 0
• - dosVeces = \f -> f . f
• - flipAll = map flip
• - flipRaro = flip flip
  
```

i. ¿Cuál es el tipo de cada función? (Suponer que todos los números son de tipo FLOAT).

ii. Indicar cuáles de las funciones anteriores *no* están currificadas. Para cada una de ellas, definir la función currificada correspondiente. Recordar dar el tipo de la función.

• PREG

• (DE O Q ESTA BIEN)

• NO HACER

• BIEN

1) i) a) INPUT OUTPUT
 $(\text{FLOAT}, \text{FLOAT}) \rightarrow \text{FLOAT}$

b) $(\text{FLOAT}, \text{FLOAT}) \rightarrow \text{FLOAT}$

c) FLIP (-) HA VUELTA LOS ARG DE (-). ACÁ (-) ESTÁ APLICADA PARCIALMENTE?

$$\hookrightarrow g: \text{SUBTRACT } 1 2 \equiv \text{FLIP}(-) 1 2 \equiv (-) 2 1 \equiv 2$$

$\text{FLOAT} \rightarrow (\text{FLOAT} \rightarrow \text{FLOAT}) \rightarrow$ Existe 1 ARG de dentro función? Luego, un ARG.
¿O PREDECESOR?

d) $\text{FLOAT} \rightarrow \text{FLOAT}$. SUBTRACT APlica PARCIALMENTE.

e) EVALUARENZERO = $(\lambda f \rightarrow f) 0 \equiv f 0 \equiv f \$ 0$

\Rightarrow No sé qué escribe f. Yo sé qué escribe FLOAT.

$(\text{FLOAT} \rightarrow \alpha) \rightarrow \alpha$

f INPUT SALIDA
 $\text{DOSVECES} = \lambda f \rightarrow f \cdot f \equiv f \cdot f$ $(\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$

g) TRIPLE ::= $\text{FLOAT} \rightarrow \text{FLOAT}$

$$\text{TRIPLE} = (*) 3$$

$$\hookrightarrow \text{TRIPLE} \cdot \text{TRIPLE} = \text{TRIPLE}(\text{TRIPLE}(\text{m}))$$

h) FLIPALL = MAP FLIP $\equiv \text{MAP}(\text{FLIP}) \equiv \text{MAP} \$ \text{FLIP}$.

\hookrightarrow Cambia orden de los dags de cada función.

MAP: $(\alpha \rightarrow \beta) \rightarrow \text{FOLDABLE}[\alpha \rightarrow \beta]$

FLIP: $(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \beta \rightarrow \alpha \rightarrow \gamma$

$$\{ : [(-), *] \cap d \rightarrow e \rightarrow f \equiv e \rightarrow f \rightarrow f$$

$$\hookrightarrow \alpha \rightarrow \beta \rightarrow \gamma \equiv \beta \rightarrow \alpha \rightarrow \gamma$$

$[\alpha \rightarrow \beta]$

\hookrightarrow ALIDAD? SUPONGO A 2

\hookrightarrow SIN NO AUNA FLIP.

\hookrightarrow Me importa que función se le aplique FLIP? \hookrightarrow CAPAZ CADA UNA TIENE DIF TIPO.

\hookrightarrow igual (no que Al PASAR POR HAB. DEVUELVEN TODOS EL MISMO.

(- Asocia i 2Q

A VER: $(\text{MAP FLIP})[f_1, f_2, \dots, f_m]$ NO. MAL PARÉNSIS. FLIP da f, lista THB UNA.

$\text{MAP}(\text{FLIP} [f_1, f_2, \dots, f_m])$

$$\hookrightarrow \text{FLIP} f_1 \text{ FLIP} f_2 \dots$$

$$\text{FLIP} f_0 \rightarrow [f_1' f_2' \dots f_m']$$

$f_1, f_2, \dots, f_n \rightarrow f_1, f_2, f_3$

MAP: $(a \rightarrow b) \rightarrow \text{FOLDABLE } x, a \rightarrow b$

FLIP: $(a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c$

FUNC: $[a \rightarrow b \rightarrow c] \rightarrow [b \rightarrow a \rightarrow c]$

MAP FLIP $\{(-), (+)\}$

h) FLIP FLIP \equiv FLIP \$ FLIP \equiv FLIP(FLIP)

$$\begin{aligned} \text{FLIP(FLIP)} &= \text{FLIP}((a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c) \\ &= b \rightarrow (a \rightarrow b \rightarrow c) \rightarrow a \rightarrow c \end{aligned}$$

1) ii) a) $\text{MAX}_2 :: \text{FLOAT} \rightarrow \text{FLOAT} \rightarrow \text{FLOAT}$

$$\text{MAX}_2 x y = \dots$$

b) NORMAVECTORIAL $:: \text{FLOAT} \rightarrow \text{FLOAT} \rightarrow \text{FLOAT}$

$$\text{NORMAVECTORIAL } x y = \dots$$

CREO Q LAS OTROS ESTÁN CURRIFICADAS.

Ejercicio 2 *

- i. Definir la función curry, que dada una función de dos argumentos, devuelve su equivalente currificado.
- ii. Definir la función uncurry, que dada una función currificada de dos argumentos, devuelve su versión no currificada equivalente. Es la inversa de la anterior.
- iii. ¿Se podría definir una función curryN, que tome una función de un número arbitrario de argumentos y devuelva su versión currificada?

Sugerencia: pensar cuál sería el tipo de la función.

i) CURRY APlica.
CURRY Ayuda A marcar ARGs SEPARADOS Y APlicarlos COMO TUPLA.

CURRY $:: ((a, b) \rightarrow c) \rightarrow a \rightarrow b \rightarrow c$

CURRY $f a b = f(a, b)$
RES TIPO C

CURRY $\underbrace{f}_{\text{f}} \underbrace{a}_{\text{a}} \underbrace{b}_{\text{b}} \equiv \text{MAX}_2(1, 2)$

ii) UNCURRY APlica.
UNCURRY Ayuda A marcar ARGs JUNTOS Y APlicarlos SEPARADOS

UNCURRY $:: (a \rightarrow b \rightarrow c) \rightarrow (a, b) \rightarrow c$

Uncurry $f(a,b) = f \underbrace{a \ b}_{\text{RES TIPo C}}$

Uncurry subtract (2,1)

iii) CURRY MAX 1 2 3 4 5 6 7

De lo que se pide. No hay tipos de n elemento dinámico

ESQUEMAS DE RECURSIÓN

Ejercicio 3 *

- i. Redefinir usando foldr las funciones sum, elem, (++), filter y map.
- ii. Definir la función mejorSegún :: ($a \rightarrow a \rightarrow \text{Bool}$) $\rightarrow [a] \rightarrow a$, que devuelve el máximo elemento de la lista según una función de comparación, utilizando foldr1. Por ejemplo, maximum = mejorSegún (>).
- iii. Definir la función sumasParciales :: Num a $\rightarrow [a] \rightarrow [a]$, que dada una lista de números devuelve otra de la misma longitud, que tiene en cada posición la suma parcial de los elementos de la lista original desde la cabeza hasta la posición actual. Por ejemplo, sumasParciales [1,4,-1,0,5] $\sim [1,5,4,4,9]$.
- iv. Definir la función sumaAlt, que realiza la suma alternada de los elementos de una lista. Es decir, da como resultado: el primer elemento, menos el segundo, más el tercero, menos el cuarto, etc. Usar foldr.
- v. Hacer lo mismo que en el punto anterior, pero en sentido inverso (el último elemento menos el anteúltimo, etc.). Pensar qué esquema de recursión conviene usar en este caso.

3)i)

1) SUM :: Num a $\rightarrow [a] \rightarrow a$

$\text{SUM} :: \text{Num } a \rightarrow a \rightarrow a$ nistro tipos ¿cómo fueran inicializadas las ramas?

$\text{SUM} = \text{FOLDR}(\lambda m \text{ REC} \rightarrow m + \text{REC}) 0 \equiv \text{FOLDR}(+) 0$

$\text{Sum } [1,2,3] = \text{FOLDR}(\lambda m \text{ REC} \rightarrow 1 + \text{FOLDR}(\lambda m \text{ REC} \rightarrow 2 + \text{FOLDR}(\lambda m \text{ REC} \rightarrow 3 + 0)))$

$$6 = \overbrace{1}^{\text{REC3}} + \overbrace{5}^{\text{REC2}} = \overbrace{2}^{\text{REC2}} + \overbrace{3}^{\text{REC1}}$$

$\text{C1} \leftarrow \text{RESP}$

2) EVEN :: Eq a $\rightarrow a \rightarrow [a] \rightarrow \text{Bool}$

\sim FALSE PEOR CASO

$\text{EVEN} :: \text{Eq } a \rightarrow a \rightarrow [a] \rightarrow \text{Bool}$

PARA LA RECURSIÓN

$\text{EVENP} = \text{FOLDR}(\lambda x \text{ REC} \rightarrow \text{IF } x == e \text{ THEN TRUE ELSE REC}) \text{ FALSE}$

LAS RAMAS TIENEN

$\text{EVEN 1 } [2,1] = \text{FOLDR}(\lambda x \text{ REC} \rightarrow \text{IF } 2 == 1$

THEN TRUE

ELSE FOLDR($\lambda x \rightarrow \text{IF } 2 == 2$

THEN TRUE

ELSE FALSE)

$\text{FOLDR}(\lambda x \text{ REC} \rightarrow \text{IF } 2 == 1 \text{ THEN TRUE ELSE TRUE})$

TRUE

F.1: Forma Lógica

F.2: Forma Cíclica

3) $(++) :: [a] \rightarrow [a] \rightarrow [a]$

$(++) :: [a] \rightarrow [a] \rightarrow [a]$

$(++) l_1 l_2 = \text{FOLDL}(\lambda x \text{ REC} \rightarrow x : \text{REC}) l_2 l_1$

REASON: FLIP $\$ (\text{FOLDL}(:))$ \rightsquigarrow ENT Ahi tom l_1 & l_2 $\xrightarrow{\text{CB}}$.

Condition

4) $\text{FILTER} :: (a \rightarrow \text{BOOL}) \rightarrow [a] \rightarrow [a]$

$\text{FILTER } f = \text{FOLDL}(\lambda e \text{ REC} \rightarrow \text{IF } f e \text{ THEN } e : \text{REC} \text{ ELSE REC}) []$

$\text{FILTER } (\lambda x \rightarrow \text{EVEN } x) [3, 2]$

$\text{FOLDL}(\lambda e \text{ REC} \rightarrow \text{IF EVEN } 3 \text{ THEN } 3 : \text{FOLDL}(\lambda e \text{ REC} \rightarrow \text{IF EVEN } 2 \text{ THEN } 2 : [] \text{ ELSE } []) \text{ ELSE FOLDL} \dots)$

$\text{FOLDL}(\lambda e \text{ REC} \rightarrow \text{IF EVEN } 3 \text{ THEN } 3 : \text{FOLDL}(\lambda e \text{ REC} \rightarrow [2]) \text{ ELSE FOLDL}(\lambda e \text{ REC} \rightarrow [2]))$

$\text{FOLDL}(\lambda e \text{ REC} \rightarrow \text{IF EVEN } 3 \text{ THEN } 3 : [2] \text{ ELSE } [2])$

RES = [2]

5) $\text{MAP} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

$\text{MAP } f = \text{FOLDL}(\lambda e \text{ REC} \rightarrow f e : \text{REC}) []$

3) ii)

$\text{MESORSEGON} :: (a \rightarrow a \rightarrow \text{BOOL}) \rightarrow [a] \rightarrow a$

$\text{MESORSEGON } f l = \text{FOLDL}(\lambda e \text{ REC} \rightarrow \text{IF } (f e \text{ REC})$
 $\text{THEN } e \text{ ELSE REC})$

3) iii) $[1, 4, -1, 0, 5] = \text{REVERSE} = [5, 0, -1, 4, 1]$

$= [5, 0, -1, 4, 1] = \text{REVERSE} = [1, 5, 4, 0]$

$\xrightarrow{\text{REVERSE REC}}$

$\xrightarrow{\text{SI REC ES VACIA ESTA EN CB. LISTA VACIA}}$

$\text{SUMASPARCIALES} :: \text{Num } a \Rightarrow [a] \rightarrow [a]$

$\xrightarrow{S : (a : \text{REC}) \rightarrow \text{O T H E N } e + \text{HEAD}(e) : \text{REC}}$

$\text{SUMASPARCIALES } L = \text{REVERSE}(\text{FOLDL}(\lambda e \text{ REC} \rightarrow e + \text{HEAD}(e) : \text{REC})[]) \text{ (REVERSE}(L)\text{)}$

$\xrightarrow{= S + \text{HEAD}(0 + \text{HEAD}(-1 + \text{HEAD}(4 + \text{HEAD}(1 + \text{HEAD}([]) : \text{REC}) : \text{REC}) : \text{REC}) : \text{REC}}$

$\xrightarrow{4 + \text{HEAD}([1]) : \text{REC}}$

$\xrightarrow{-1 + \text{HEAD}([5, 1]) : \text{REC}}$

$\xrightarrow{0 + \text{HEAD}([4, 5, 1]) : \text{REC}}$

$\xrightarrow{S + \text{HEAD}([4, 5, 1]) : \text{REC}}$

$\xrightarrow{\text{REVERSE} [4, 5, 1]}$

$\xrightarrow{[1, 5, 4, 5, 1]}$

CON FOLCL: REVERSE-FOLCL (\AC X -1 IF LENGTH AC > 0 THEN HEAD AC + X) : AC

x7 oggi PUNTER ESEM AL RIPAL.

F(LSF X:AC) []

[S, 9, 3, 2, 1]

LASSA CAR PUNTER ESEM PER ANIMA MARIA ATLAS.

↳ S.[4:3:(2:(1:[]))]

$$3) \text{ iv) } [1, 2, 3] = P_1 - P_2 + P_3 - P_4 + P_5 = 2$$

$$\begin{aligned} &= [1, 2, 3] = 2 - 3 \xrightarrow{\text{N.C.}} -1 \\ &\quad = 1 - (-1) = 2 \quad \text{CB: O} \\ &\quad \quad \quad \text{N.C.} \end{aligned}$$

SUMAALT :: NUM a. $\Rightarrow [a] \rightarrow a$

SUMAALT = FOLDL (-) 0 ✓

$$\begin{aligned} &= (-) 1 ((-) 2 ((-) 3 0)) \\ &\quad - 1 = (-) 2 3 \\ &\quad = (-) 1 - 1 \end{aligned}$$

2 =

$$3) \text{ v) } [1, 2, 1, 4] = 4 - 1 + 2 - 1 = 4$$

$$= 3 + 2 = 5$$

$$= 5 - 1$$

$$= 4$$

ACUM = 0

FOLCL :: (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow [] \rightarrow b

FOLDL f AC [] = AC

FOLDL f AC (x:xs) = f FOLDL f AC (x) xs

$$1 - 2 - 1 - 4 = -1 - (-1) = 0$$

$$= 0 - (-4) = 4$$

$$F.L: [1, 2, 3] = 3 - 2 + 1 = 2$$

$$F.C: 1 - 2 - 3 = -1 - (-3) = 2$$

SUMAALT INVERSA :: INT \rightarrow INT

FOLCL :: (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow [] \rightarrow b

SUMAALT INVERSA = FOLDL (FLIP (-)) 0

FOLDL f AC [] = AC

FOLDL f AC (x:xs) = FOLDL f (f AC x) xs

$$= FOLDL (-) 0 [1, 2, 3]$$

$$FOLDL (FLIP (-)) 0 [1, 2, 3]$$

$$= FOLDL (-) (-1) [2, 3]$$

$$FOLDL (FLIP (-)) 1 [2, 3]$$

$$= FOLDL (-) (-3) [3]$$

$$FOLDL (FLIP (-)) 1 [3]$$

$$= \text{FOLDL}(-) (-6) []$$

$$= -6$$

$$\text{FOLDL}(\text{FLIP}(-)) 2 []$$

$$\Rightarrow 2.$$

En este caso $\text{FOLDL}(-)(\text{ACC}-x) \times x$

RESULTADO ANTES DE SEGUIR AVANZANDO.

FOLDL RESULTADO AL FINAL.

LISTAS INF: PREG ESTE TIPO DE LAZI.

TAKE 2 (FOLDL (:)[] [1..])

SABE Q MULTIPLICAR POR DOS. NO GUERRA NADA

El TAKE 2 le dice al FOLDL que necesita una lista con 2 elementos

Entonces el FOLDL hace: $(1:(2:[]))$

Y luego TAKE 2 [1,2] = [1,2]

FLIP

TAKE 2 (FOLDL (:)[] [1..])

$\text{FOLDL}(:)((:)[] 1) [2..]$

$\text{FOLDL}(:)[1] [2..]$

$\text{FOLDL}(:)((:)[1]:2) [3..] \rightarrow$ SE CUELG A XQ NO HA LLEGADO A
Q LISTA INF SEA VACIA

FOLDL ES LAZY. DEVEREUF. RESULTADOS PARCIALES SIN EVALUAR

TODO LA LISTA.

FOLDL ES ESTÁTICO. RECORRE TODO LA LISTA ANTES DE DAR RESULTADO.

FOLDL ASOCIA A IZQ DE FUNCIÓN f, FOLDR ASOCIA A DERECHA.

PUNTO ARG ACUM

PUNTO ARG REC.

Ejercicio 5 ★

Considerar las siguientes funciones:

- `elementosEnPosicionesPares :: [a] -> [a]`
`elementosEnPosicionesPares [] = []`
`elementosEnPosicionesPares (x:xs) = if null xs`
`then [x]`
`else x : elementosEnPosicionesPares (tail xs)`
- `entrelazar :: [a] -> [a] -> [a]`
`entrelazar [] = id`
`entrelazar (x:xs) = \ys -> if null ys`
`then x : entrelazar xs []`
`else x : head ys : entrelazar xs (tail ys)`

Indicar si la recursión utilizada en cada una de ellas es o no estructural. Si lo es, reescribirla utilizando foldr.

- 1) No es recursión estructural porque la recursión es sobre ($x:xs$) y se le da un complemento algo de xs y No se le da el resto de parámetros al llamado recursivo. null xs
- 2) Es recursión estructural porque se hace la recursión sobre ($x:xs$) y xs No se da el resto como si el resto no lo pase en el llamado recursivo.

PARA DECIR S. ES ESTO NO SOLO ME IMPRESA SOBRE "LA RECURSIÓN MÁS GRANDE"
¿Qué pasó si se hace recursión sobre ($x:xs$) y ($y:ys$) u lo q?

ENTRARAZAR = FOLDL(||x REC YS -> IF (NULL YS) THEN X:REC []
ELSE X:REC (TAIL YS)) (CONS [])
si recurs ($x:xs$) y llegué a la base. RETORNO []. Ej: f: CONS [] ys = [] ys.
¿Qué tendría el 2do arg? ¿ ys ? Sí :) En la segunda llamada

Ejercicio 6 ★

El siguiente esquema captura la recursión primitiva sobre listas.

```
recr :: (a -> [a]) -> b -> b -> [a] -> b  
recr _ z [] = z  
recr f z (x : xs) = f x xs (recr f z xs)
```

- a. Definir la función sacarUna :: Eq a => a -> [a] -> [a], que dados un elemento y una lista devuelve el resultado de eliminar de la lista la primera aparición del elemento (si está presente).
- b. Explicar por qué el esquema de recursión estructural (foldl) no es adecuado para implementar la función sacarUna del punto anterior.
- c. Definir la función insertarOrdenado :: Ord a => a -> [a] -> [a] que inserta un elemento en una lista ordenada (de manera creciente), de manera que se preserva el ordenamiento.

[z, 3, 5, 6, 8]
[]

[6, 8] [6]

0-) CUANDO LLEGO AL EUEN CONO NECESSARIO. PROCESAR + COLO

SACARUNA :: EQ a => a -> [a] -> [a]

Tipo b TIENES NEC. YA NICE TODO.

SACARUNA e = RECR(||x XS REC -> IF (x == e) THEN XS

Tipo b

ELSE X:REC) []

Algo tipo b

SACARUNA 2 [1, 2, 3, 2] = RECR(||1 [2, 3, 2] ? -> IF (1 == 2) THEN [2, 3, 2]

ELSE 1: (NECR(||2 [3, 2] ? -> IF 2 == 2

THEN [3, 2])

ELSE ...))

RESULTADO A IZQ

= RECR(||1 [2, 3, 2] ? -> IF (1 == 2) THEN [2, 3, 2]

ELSE 1: ([3, 2]))

$$= [1, 3, 2] \text{ RES2.}$$

RES

M1 Duda: ¿Por qué el resultado es una lista con la primera ocurrencia si se imprime a resolver del final? ¿"El camino elegido lo almacena antes de ejecutar?"

$[1, 2, 3, 2]$

RES: \downarrow OSEA ALMA IZO A DEL Y A TOMANDO LA DECISIÓN Y ALMACENA EL CAMINO. DSP RESUELVE "ALMACENA"

$1 \neq 2, 2 = 2$ ENTONCES IF. IF \rightarrow FALSE \rightarrow IF TRUE $\rightarrow [3, 2]$ EL RESULTADO. SÍ. VA IZO A DEL, TOMA ECISIONES
 $\rightarrow 1 : [3, 2] = [1, 3, 2]$ Y DESPUÉS RESUELVE.

b) Porque necesitamos xs para "terminar" la recursión. Si no, sacaría las otras apariciones.

c) INSERTAR ORDENADO:: ORD $a \geq b \rightarrow a -> [a] \rightarrow [a]$

INSERTAR ORDENADO e = DECR (|x xs REC \rightarrow IF ($x > e$) THEN $e : x : xs$ $\stackrel{b}{\nearrow}$ $\stackrel{x \neq TERMINAR REC.}{\nearrow}$ YA HICIE ROBO.
 $\text{ELSE } e : \text{REC} |$ $\stackrel{b}{\nearrow}$ $\stackrel{b}{\nearrow}$

Ejercicio 7 ★

Definir las siguientes funciones para trabajar sobre listas, y dar su tipo. Todas ellas deben poder aplicarse a listas finitas e infinitas.

- I. mapPares, una versión de map que toma una función currificada de dos argumentos y una lista de pares de valores, y devuelve la lista de aplicaciones de la función a cada par. **Pista:** recordar curry y uncurry.
- II. armarPares, que dadas dos listas arma una lista de pares que contiene, en cada posición, el elemento correspondiente a esa posición en cada una de las listas. Si una de las listas es más larga que la otra, ignorar los elementos que sobran (el resultado tendrá la longitud de la lista más corta). Esta función en Haskell se llama zip. **Pista:** aprovechar la currificación y utilizar evaluación parcial.
- III. mapDoble, una variante de mapPares, que toma una función currificada de dos argumentos y dos listas (de igual longitud), y devuelve una lista de aplicaciones de la función a cada elemento correspondiente de las dos listas. Esta función en Haskell se llama zipWith.

MAPPARES :: $(a \rightarrow b \rightarrow c) \rightarrow [(a, b)] \rightarrow [c]$

MAPPARES f = MAP \$ UNCURRY f



ARMAR PARES :: $[a] \rightarrow [b] \rightarrow [(a, b)]$

ARMAR PARES = FOLDL (|x REC (y:ys) \rightarrow (x, y) : REC ys) (CONST [])



MAPDOBLE :: $(a \rightarrow b \rightarrow c) \rightarrow [a] \rightarrow [b] \rightarrow [c]$

MAPDOBLE f l1 l2 = MAPPARES f \$ ARMAR PARES l1 l2



- I. Escribir la función `sumaMat`, que representa la suma de matrices, usando `zipWith`. Representaremos una matriz como la lista de sus filas. Esto quiere decir que cada matriz será una lista finita de listas finitas, todas de la misma longitud, con elementos enteros. Recordamos que la suma de matrices se define como la suma celda a celda. Asumir que las dos matrices a sumar están bien formadas y tienen las mismas dimensiones.

`sumaMat :: [[Int]] -> [[Int]] -> [[Int]]`

- II. Escribir la función `trasponer`, que, dada una matriz como las del ítem I, devuelva su traspuesta. Es decir, en la posición i, j del resultado está el contenido de la posición j, i de la matriz original. Notar que si la entrada es una lista de N listas, todas de longitud M , la salida debe tener M listas, todas de longitud N .

`trasponer :: [[Int]] -> [[Int]]`

Ejercicio 9 ★

- Definir y dar el tipo del esquema de recursión `foldNat` sobre los naturales. Utilizar el tipo `Integer` de Haskell (la función va a estar definida sólo para los enteros mayores o iguales que 0).
- Utilizando `foldNat`, definir la función `potencia`.

$$\text{FR} + \text{CB} + \text{INPUT} + \text{NES}$$

(E, REC, NES, REC, NES)

misma tipo

`FOLDNAT :: (INTEGER -> b -> b) -> b -> INTEGER -> b`

`FOLDNAT f z 0 = z`

`FOLDNAT f z m = f m (FOLDNAT f z (m-1))`

$$7^4 = 4 * 4 * 4 * 4 * 4$$

`POTENCIA :: INTEGER -> INTEGER -> INTEGER`

`POTENCIA m = FOLDNAT (\x REC -> m * REC) 1`

$$2 * (2 * (2 * (2 * (1)))) = 8 \quad \text{CB}$$

$$\begin{matrix} m=3 & m=2 & m=1 & m=0 \\ NREC=8 & NREC=4 & NREC=2 & NREC=1 \end{matrix}$$

$$\begin{matrix} m=1 & 2^3 = 2 \cdot 2 \cdot 2 \\ m=2 & m=3 \end{matrix}$$

m es el que nos lleva hacia llegar a 0 ~ ya preparó las multiplicaciones.

Ejercicio 10

- Definir la función `genLista` :: $a \rightarrow (a \rightarrow a) \rightarrow \text{Integer} \rightarrow [a]$, que genera una lista de una cantidad dada dada de elementos, a partir de un elemento inicial y de una función de incremento entre los elementos de la lista. Dicha función de incremento, dado un elemento de la lista, devuelve el elemento siguiente.
- Usando `genLista`, definir la función `desdeHasta`, que dado un par de números (el primero menor que el segundo), devuelve una lista de números consecutivos desde el primero hasta el segundo.

$$\boxed{[n, n+1, \dots]}$$

`GENLISTA :: a -> (a -> a) -> INTEGER -> [a]`

`GENLISTA f m = TAKE m [x | x < [i, f i ..]]`

La función ($a \rightarrow a$) `GENLISTA` pude hacer que sea

$\underbrace{f \text{ ALGO}}_{\text{CONST 1}} = 1$

TAKE M SERIA MAYOR - MENOR

Si AVE! PERO ME PIDE (ENUM a); ESTÁ BIEN? ENTENDO QUE SI

DESDE HASTA :: (a, a) → [a] MAL (ABOLIR). → EL PANAM le da en DESDE HASTA.

DESDE HASTA (a, b) = GENLISTA a ($\overbrace{i+1}^{(b-a)}$) ($i \in [1, b-a]$) ESTA MÁLTIBA qd sería [S, 1] qd sería m. leia [m, m+1] ID a qd sería [a, a...]

¿QUEDA (+1)? ENT [i, i+1] = [S, 6...]

Bien aplica. f ESPERA UN SOLO ARG qd ESTA APLICADA PARCIALMENTE.

= GENLISTA a ($\overbrace{i+1}^{(+1)}$) ($b-a$)

La IMPLEMENTACIÓN ESTABA BIEN, me falleó el tipo, FORZAR con 3 PANAM int.

~ DESDE HASTA :: (INTEGRAL a) → (a, a) → [a]

DESDE HASTA (a, b) = a. (+1) (FRON INTEGRAL (b-a+1))

TOMA NOMBRE ENTEND Y LO CONVIENE EN NOMBRE MAS GRAL.

MJON IMPLEMENTACIÓN: LIMITAR M A INTEGRAL. CONVERTIR EN FUNCIÓN.

GENLISTA :: (INTEGRAL b, ENUM a) → a → (a → a) → b → [a]

GENLISTA i / m = TAKE (FRON INTEGRAL m :: INT) [x | x <= [i, f i..]]
INT → [a] qd m lo convierte desde num a INT.
↑ si m lo paga, BASICALL INFIERNE.

DESDE HASTA :: (INTEGRAL a) → (a, a) → [a]

DESDE HASTA (a, b) = GENLISTA a. (+1) (b-a+1)

FRON INTEGRAL FUNCIONA SOLO CON INTEGRAL b.



USAR WHEN y CASE.

Ejercicio 11

Definir el esquema de recursión estructural para el siguiente tipo:

```
data Polinomio a = X->b  
| Cte a->(a->a)  
| Suma (Polinomio a) (Polinomio a) ->(b->b->b)  
| Prod (Polinomio a) (Polinomio a) ->(b->b->b)
```

Luego usar el esquema definido para escribir la función evaluar :: Num a => a -> Polinomio a -> a que, dado un número y un polinomio, devuelve el resultado de evaluar el polinomio dado en el número dado.

FOLDPOLI :: b -> (a -> b) -> (b -> b -> b) -> (b -> b -> b) -> Polinomio a -> b

FOLDPOLI f x f CT. f SON FPNOB POLI = CASE POLI OF

x -> fx

FOR CONSTRUCCIONES ALG b.

FOR CN ALG b.

CTE a -> f CTE a

SUMA a b -> fSUM (REC a) (REC b)

PROD a b -> fPROD (REC a) (REC b)

WHEN REC = FOLDPOL: fx fCTE fSUM fPROD

$$\begin{aligned}
 2x^2 + x + 1 &\equiv \text{PROD}(2)(x^2) \\
 &\equiv \text{SUM}(\text{PROD}(2)(x^2) \times) \\
 &\equiv \text{SUM}\left(\left(\text{SUM}\left(\text{PROD}(2)(x^2) \times\right)\right) 1\right)
 \end{aligned}$$

1
 2
 3

EVALUAR :: Num a => a -> Polinomio a -> a

EVALUAR m = FOLDPOL: m id (+)(*) m id (+)(*) SI ME ENCUENTRE UN NÚMERO ID g=3

(El mismo tipo que SUMA. Si no que FOLD POLINOMIO ES DE FOLD.

Ejercicio 12 ★ 6

Considerar el siguiente tipo, que representa los árboles binarios:

data AB a = Nil | Bin (AB a) a (AB a)

- I. Usando recursión explícita, definir los esquemas de recursión estructural (foldAB) y primitiva (recAB), y -> FOLD EN BASE A REC dar sus tipos.
- II. Definir las funciones esNil, altura y cantNodos (para esNil puede utilizarse case en lugar de foldAB o recAB).
- III. Definir la función mejorSegún :: (a -> a -> Bool) -> AB a -> a, análoga a la del ejercicio 3, para árboles. Se recomienda definir una función auxiliar para comparar la raíz con un posible resultado de la recursión para un árbol que puede o no ser Nil.
- IV. Definir la función esABB :: Ord a => AB a -> Bool que chequea si un árbol es un árbol binario de búsqueda. Recordar que, en un árbol binario de búsqueda, el valor de un nodo es mayor o igual que los valores que aparecen en el subárbol izquierdo y es estrictamente menor que los valores que aparecen en el subárbol derecho.
- V. Justificar la elección de los esquemas de recursión utilizados para los tres puntos anteriores.

i)

FOLDAB :: b -> (b -> a -> b -> b) -> AB a -> b

FOLDAB z fBin Nil = z

FOLDAB z fBin (Bin; rd) = fBin (REC(i)) r (REC(d))

WHEN REC = FOLDAB z fBin

RECAB :: b -> (b -> a -> AB a -> AB a -> b -> b) -> AB a -> b

RECAB z fBin Nil = z

$\text{RECAB} \in \text{fBIN}(\text{BiPir}\alpha) = \text{fBIN}(\text{NEC}\alpha) \vdash \text{id}(\text{NEC}\alpha)$

WHEN $\text{REC} = \text{RECAB} \in \text{fBIN}$

DEFIN. FOLDAB' EN BASE A RECAB ?

$\text{FOLDAB}' :: b \rightarrow (b \rightarrow a \rightarrow b \rightarrow b) \rightarrow AB\alpha \rightarrow b$

$\text{FOLDAB}' \in \text{fBIN} \alpha = \text{RECAB} \in (\text{BiPir}\alpha \text{ fBIN} \alpha) \checkmark$

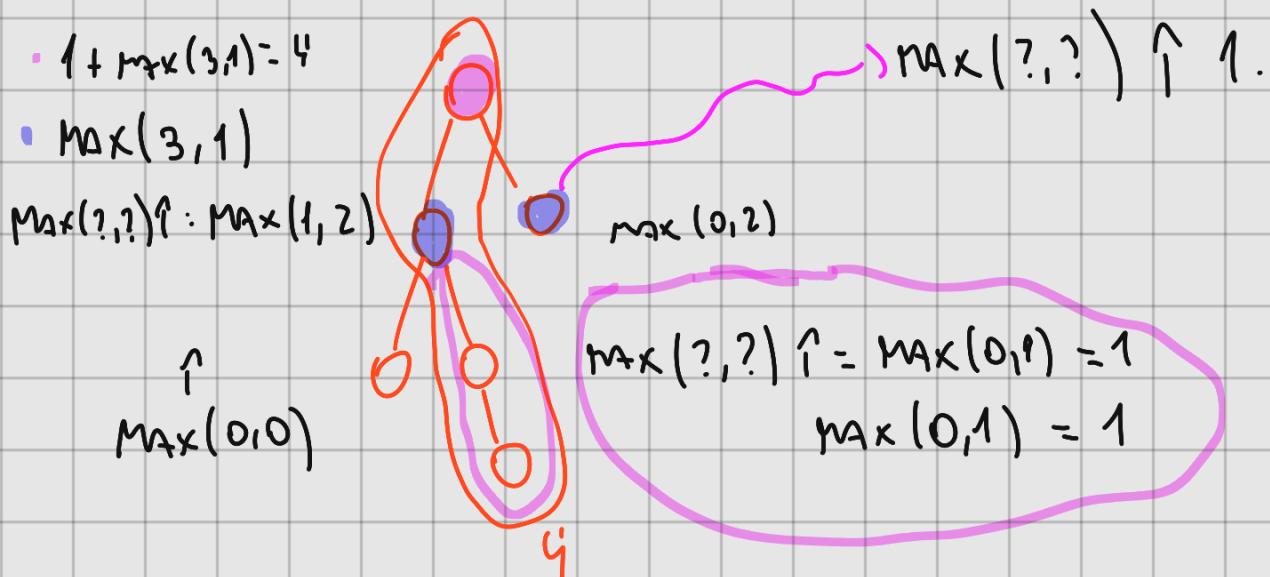
iii) $\text{ESPIL} :: AB\alpha \rightarrow \text{BOOL}$

$\text{ESNIL Nil} = \text{TRUE}$

$\text{ESNIL } - = \text{FALSE}$

$\text{ALTURA} :: AB\alpha \rightarrow \text{INT}$

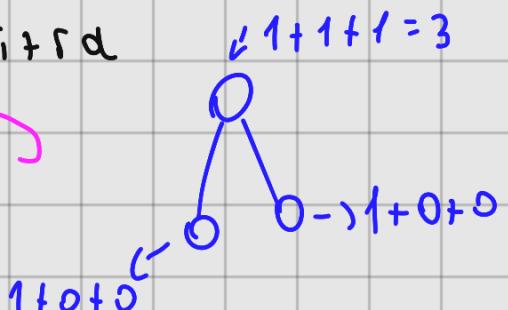
$\text{ALTURA} = \text{FOLDAB} 0 (\text{BiPir}\alpha \rightarrow 1 + \text{MAX}(\text{BiPir}\alpha))$



PRIMEROS PASOS. COMPARA NUMERO CON DER Y AL FINAL COMPARA NUMERO CON LA RAIZ DEL NODO MAS ALTA QUE EL NODO

CANTIDADES: HISTORIAL PIRAMIDE $\stackrel{\text{nodo}}{\sim} 1 + \text{BiPir}\alpha$

$$\begin{aligned} 1 + \text{BiPir}\alpha &= 1 + (1+0+0) + (1+0+0) \\ &= 1 + 1 + 1 \end{aligned}$$



iii) MEJONSEGUUN: $[1, 10, 3] (>)$

FUNCIÓN

que es lo

$\text{IF}(>1(\text{IF}(>10(\text{IF}(>3 \text{ ? THEN } 3 \text{ ELSE?}) \text{ ELSE } (\text{IF}(>3 \text{ ? THEN } 3 \text{ ELSE?}))$)

$\text{IF}(>1(\text{IF}(>10 \text{ THEN } 10 \text{ ELSE...})$

$\text{IF}(>1110 \text{ THEN } 1 \text{ ELSE } (\text{IF}(>103 \text{ THEN } 10 \text{ ELSE...}))$

$\text{IF}(>103 \text{ THEN } 10 \text{ ELSE...}) = 10$ ↑
 ACA Iban JPN
 PROGRAMACIÓN DINÁMICA

El ? en si el CB toma el ULTIMO DE LISTA.
 Otro algoritmo en el ULTIMO.

SIMPON: DEVUELVE VALOR (Num).

CB. Miz Compar.

En el orden serán el ÚLTIMO PUEC

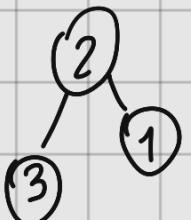
MESONSEGUNAB:: ($a \rightarrow a \rightarrow \text{BOOL}$) $\rightarrow \text{AB} \rightarrow a$

MESONSEGUNAB f Nil = Entón "nil" \rightarrow ESTA CASO HACE MESONSEGUN FONCIÓN TOTAL.

MESONSEGUNAB f (Bin(lrd)) = FOL3AB f (lrd) \rightarrow if (frr; & frr) THEN

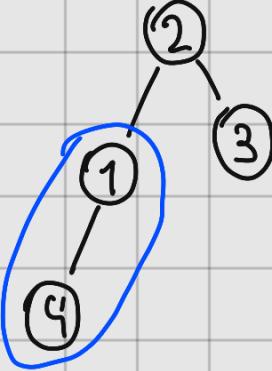
ELSE if (frr) THEN f ELSE f (Bin lrd)
 PIPON PIPON
 ejecutar ejecutar
 r

• ESABB:



$2 > 3$? NO. NO ESABB.

$2 < 1$? NO.



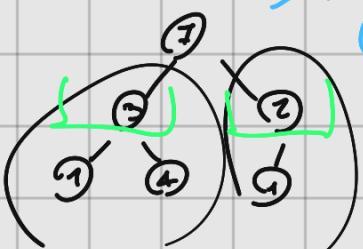
(NO).

↑
 FALSE

$2 > 1 > 4$
 Pero $2 >$ FALSE?

ACA ES ACON XQ NO P OFIJO

Comparar Non
 BOOL BOOL



Si delo CONFORM RAIZ CON FAVOR i70 + DMR NEC.
 Si NO delo CONFORM RAIZ CON FAVOR i70 + DMR TENDO NEC i70, NEC DEL

Ejercicio 13

Dado el tipo AB a del ejercicio 12:

- Definir las funciones **ramas** (caminos desde la raíz hasta las hojas), **cantHojas** y **espejo**.
- Definir la función **mismaEstructura** :: AB a -> AB b -> Bool que, dados dos árboles, indica si éstos tienen la misma forma, independientemente del contenido de sus nodos. **Pista:** usar evaluación parcial y recordar el ejercicio 7.

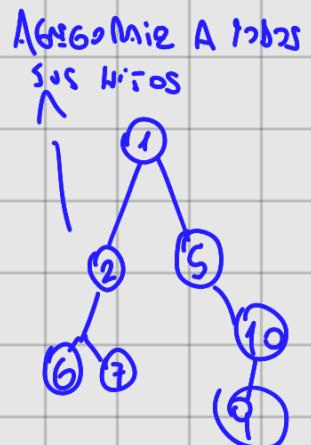
RAMAS :: AB a -> [[a]]

HO.

RAMAS = FOLDAB [] (| r : r rd -> MAP (:) r : r ++ MAP (:) r rd)

MAP (r :) r : ++ MAP (r :) r rd

OK.



[[1, 2, 6], [1, 2, 7], [1, 5, 10, 9]]

OJO: Si FALSE en CASO BASE [[r]] el MAP lo ejecuta si la lista tiene algo, pero [] no tiene nada.

DEJAN AGREGAR AL HIJOS UNO.

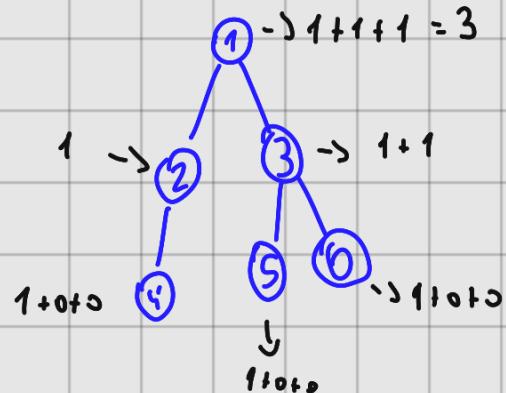
¿Cómo podríamos prescindir de esto?

RAMAS = FOLDAB [] (| r : r rd -> IF NULL (r :) == & & NULL (rd))

THEN [[r]]

ELSE MAP (r :) r : ++ MAP (r :) r rd)

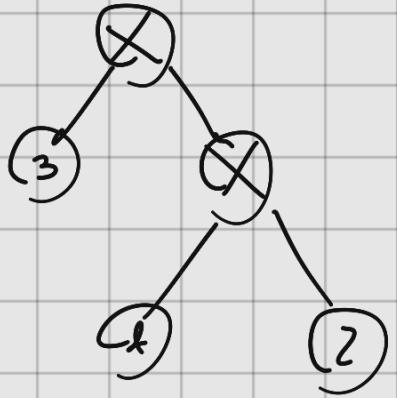
CANTHOJAS = FOLDAB 0 (| r : r rd -> (IF (== 0) r : & & (== 0) rd THEN 1 ELSE 0) + r : rd)



ESPEJO :: AB a -> AB a

ESPEJO = FOLDAB N : L (| r : r rd -> BIN (r : r : r))

MISMA ESTRUCTURA



Ejercicio 14

Se desea modelar en Haskell los árboles con información en las hojas (y sólo en ellas). Para esto introduciremos el siguiente tipo:

```
data AIH a = Hoja a | Bin (AIH a) (AIH a)
```

- a) Definir el esquema de recursión estructural `foldAIH` y dar su tipo. Por tratarse del primer esquema de recursión que tenemos para este tipo, se permite usar recursión explícita.
- b) Escribir las funciones `altura :: AIH a -> Integer` y `tamaño :: AIH a -> Integer`. Considerar que la altura de una hoja es 1 y el tamaño de un AIH es su cantidad de hojas.

$$a) \text{FOLDAIH} :: (a \rightarrow b) \rightarrow (b \rightarrow b \rightarrow b) \rightarrow \text{AIH } a \rightarrow b$$

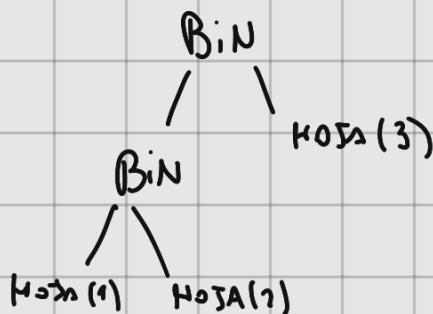
$$\text{FOLDAIH } f_{\text{HOJA}} \ f_{\text{AIH}} (\text{HOJA } a) = f_{\text{HOJA}} a \xrightarrow{\text{(Const } a)} b$$

$$\text{FOLDAIH } f_{\text{HOJA}} \ f_{\text{AIH}} (\text{BIN } id) = f_{\text{AIM}} (\text{REC } l) (\text{REC } r)$$

$$\text{WHERE REC} = \text{FOLDAIH } f_{\text{HOJA}} \ f_{\text{AIH}}$$

$$b) \text{a) ALTURA} :: \text{AIH } a \rightarrow \text{INTEGER}$$

$$\text{ALTURA} = \text{FOLDAIH}(\text{const } 0)(\text{if } id \rightarrow 1 + \max_{\substack{\text{non } l \\ \text{non } r}} \text{id})$$



$$b) \text{TAMAÑO} :: \text{AIH } a \rightarrow \text{INTEGER}$$

$$\text{TAMAÑO} = \text{FOLDAIH}(\text{const } 1)(\text{if } id \rightarrow \text{id}) \xrightarrow{\text{IGNORA VALOR}} \text{SUMA } \text{const } \text{BASER } (\text{HOJAS})$$

IGNORA VALOR

$$\text{nodo HOJA. (const } 1 \text{ VAL)} = 1$$

Ejercicio 15 *

- i. Definir el tipo RoseTree de árboles no vacíos, con una cantidad indeterminada de hijos para cada nodo.
- ii. Escribir el esquema de recursión estructural para RoseTree. **Importante** escribir primero su tipo.
- iii. Usando el esquema definido, escribir las siguientes funciones:
 - a) hojas, que dado un RoseTree, devuelva una lista con sus hojas ordenadas de izquierda a derecha, según su aparición en el RoseTree.
 - b) distancias, que dado un RoseTree, devuelva las distancias de su raíz a cada una de sus hojas.
 - c) altura, que devuelve la altura de un RoseTree (la cantidad de nodos de la rama más larga). Si el RoseTree es una hoja, se considera que su altura es 1.

Si fuera $[[\Gamma]]$ como nega Evar
Aquí?

i) DATA RoseTree: $\alpha = \text{ROSE } \alpha [\text{ROSETREE } \alpha]$

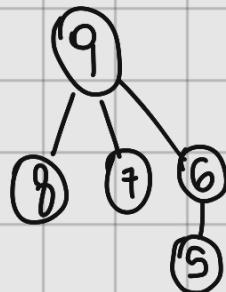
ii) FOLDROSE:: $(\alpha \rightarrow [b] \rightarrow b) \rightarrow \text{ROSETREE } \alpha \rightarrow b$

$$\text{FOLDROSE } f_{\text{ROSE}} (\text{ROSE } \alpha \text{ hijos}) = f_{\text{ROSE}} \alpha (\underbrace{\text{MAP REC hijos}}_{[b]})$$

WHEN REC = FOLDROSE f_ROSE

iii) HOJAS:: ROSETREE $\alpha \rightarrow [\alpha]$ $((::0) \cdot \text{LENGTH } \text{REC}$

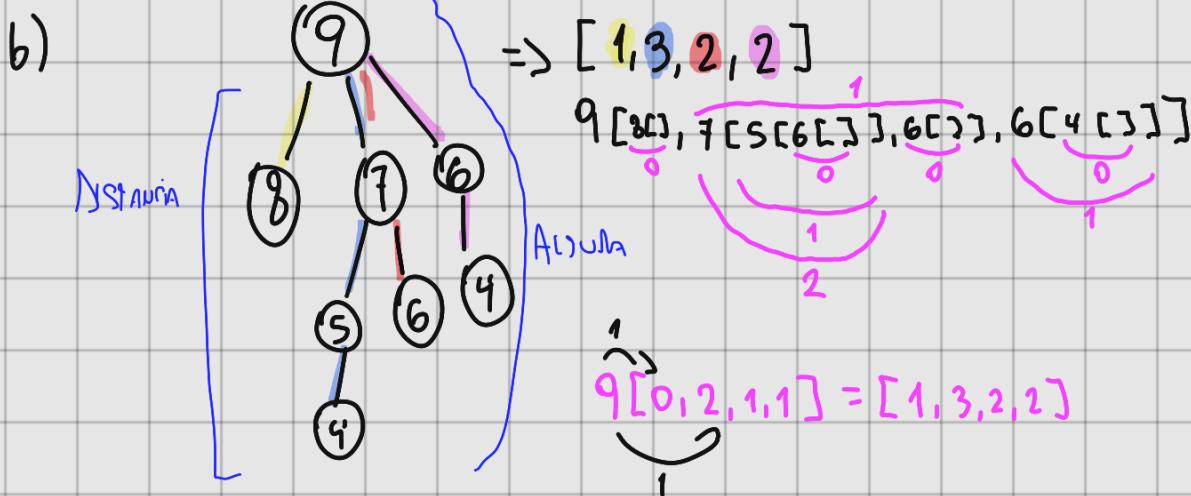
$\text{HOJAS} = \text{FOLDROSE } (| \Gamma_{\text{REC}} \rightarrow \text{IF LENGTH } (\text{REC}) == 0 \text{ THEN } [\Gamma] \\ \text{ELSE } (\text{CONCAT } \text{REC}))$



$9 [\text{ROSE } 8[], \text{ROSE } 7[], \text{ROSE } 6[\text{ROSE } 5[]]]$

- $\text{ROSE } 8[] = [8]$
- $\text{ROSE } 7[] = [7]$
- $\text{ROSE } 6[\text{ROSE } 5[]] = \text{ROSE } 6[5] = [5]$

$\{ [8], [7], [5] \} = [8, 7, 5]$



$(B: [0])$

DISTANCIAS:: ROSETREE $\alpha \rightarrow [\alpha]$

$(B: [0])$ LA MÍ² ESTÁ A DIST 0 J.P. SI MISMO.

CONCATMAP: CONCATENA RESULTADO DE HACER MAP A ESA LISTA

DISTANCIAS:: $(\text{Num } \alpha) \Rightarrow \text{ROSETREE } \alpha \rightarrow [\alpha]$

DISTANCIAS = FOLDLPOSE((lambda \rightarrow 0: COPCARMAP(MAP(+1)) REC),
CB)

C) $1 + \text{MAX}(\text{DISTANCIAS})$

ALTURA :: (DADA, EQ a) \Rightarrow ALTURAEQ a \rightarrow 0

ALTURA R = MAXIMUM \$ MAP(+1)(DISTANCIAS R)

GENERACIÓN INFINITA

Para resolver los ejercicios de esta parte se recomienda leer el apartado "Las tres leyes de la generación infinita", que se encuentra en la sección Util del campus.

Ejercicio 17

¿Cuál es el valor de esta expresión?

$\{x \mid x < [1..3], y < [1..3], (x+y) \bmod 3 = 0\}$

Genera $x=1, y=[1,2,3]$

Sigue dando $(x+y)$, y le pides con los que son divisible por 3.

Ej: $1+2 \Rightarrow 3$ es un 1 mole.

$$\begin{aligned} \text{En Serie, } & \left. \begin{array}{l} x=1 \text{ pos } y \\ x=2 \text{ pos } y \\ x=3 \text{ pos } y \end{array} \right\} \Rightarrow [1] \\ & \left. \begin{array}{l} \\ \\ \end{array} \right\} \Rightarrow [] \\ & \left. \begin{array}{l} \\ \\ \end{array} \right\} \Rightarrow [3] \end{aligned} \quad [1,3]$$

Ejercicio 18

Definir la lista infinita paresDeNat :: [(Int, Int)], que contenga todos los pares de números naturales: (0,0), (0,1), (1,0), etc.

Generación 2 diferente de 1.

CP (0,1)
 ↓ ↓

(0,0) (1,2) \rightarrow (0,1) manda solo.

[(0,0), (0,1), (1,0), (0,2), (2,0)]

Ejercicio 19

Una tripla pitagórica es una tripla (a, b, c) de enteros positivos tal que $a^2 + b^2 = c^2$.

La siguiente expresión intenta ser una definición de una lista (infinita) de triples pitagóricas:

```
pitagóricas :: [(Integer, Integer, Integer)]
pitagóricas = [(a, b, c) | a <- [1..], b <- [1..], c <- [1..], a^2 + b^2 == c^2]
```

Explicar por qué esta definición no es útil. Dar una definición mejor.

PORQUE NO SE TERMINA DE GENERAR Q => IGUAL A MÍC.