Departamento de Computación FCEyN UBA

Taller de Álgebra I - Parcial

PRIMER CUATRIMESTRE 2019 *1 de junio de 2019*

Aclaraciones

- El parcial se aprueba con tres ejercicios bien resueltos.
- Programe todas las funciones en lenguaje Haskell. El código debe ser autocontenido. Si utiliza funciones que no existen en Haskell, debe programarlas. Incluya la signatura de todas las funciones que escriba.
- No está permitido alterar los tipos de datos presentados en el enunciado, ni utilizar técnicas no vistas en clase para resolver los ejercicios.

Ejercicio 1

Implementar una función `estanRelacionados :: Integer -> Integer -> Bool` que dados a y $b \in \mathbb{Z}$ distintos de cero indique si $a \sim b$. Diremos que $a \sim b$ si y sólo si existe un $k \in \mathbb{Z}$ distinto de cero tal que $a^2 + abk = 0$.

Por ejemplo:
`estanRelacionados 8 2` $\rightsquigarrow$ `True` porque existe un $k = -4$ tal que $8^2 + 8 \times 2 \times (-4) = 0$.
`estanRelacionados 7 3` $\rightsquigarrow$ `False` porque no existe un k entero tal que $7^2 + 7 \times 3 \times k = 0$.

Ejercicio 2

Implementar la función `prod :: Integer -> Integer` que para $n \in \mathbb{N}_{>0}$ calcule la productoria $\prod_{i=1}^{2n}(i^2 + 2i)$.

Ejercicio 3

Implementar una función `esCapicua :: Integer -> Bool` que dado $n \in \mathbb{N}_{\geq 0}$ determine si n es un número capicúa.
Por ejemplo: `esCapicua 1212` $\rightsquigarrow$ `False` `esCapicua 307703` $\rightsquigarrow$ `True` `esCapicua 3` $\rightsquigarrow$ `True`
Ayuda: Puede suponer que ya está implementada la función `i_esimoDigito :: Integer -> Integer -> Integer` que dado un $n \in \mathbb{N}_{\geq 0}$ y un $i \in \mathbb{N}_{>0}$ menor o igual a la cantidad de dígitos de n, devuelve el i-ésimo dígito de n.

Ejercicio 4

Programar una función `zipPrimos :: [Integer] -> [Integer] -> [(Integer, Integer)]` que dadas dos listas de números naturales de igual longitud, devuelve una lista de tuplas formadas por números primos de ambas listas, tal que: a) la primer coordenada de las tuplas proviene de números de la primera lista pasada como parámetro y la segunda coordenada proviene de números de la segunda lista, b) sólo se deben agregar tuplas a la lista resultante cuando aparecen números primos en la misma posición en ambas listas pasadas como parámetro, y c) la lista resultante debe seguir el mismo orden de aparición de los números primos que en las listas pasadas como parámetro.

Por ejemplo:
`zipPrimos [3,4,5,1,5,3,2,10] [1,6,5,6,8,7,13,2]` $\rightsquigarrow$ `[(5,5),(3,7),(2,13)]`.

Ayuda: puede suponer que la función `esPrimo :: Integer -> Bool` que indica si un número natural es primo ya está implementada.

Ejercicio 5

Se define la sucesión $a_1 = 1$, $a_2 = 2$, $a_3 = 5$, $a_{n+1} = 3a_n^2 + 2a_{n-1} + a_{n-2}$, con $n > 3$. Programar la función `promedioTerminos :: [Integer] -> Float` que dada una lista de números naturales calcule el promedio de todos los términos de la sucesión a_n que aparecen en la lista. Si la lista no contiene términos de la sucesión la función debe retornar 0.
Por ejemplo:
`promedioTerminos [5,23489,5432,1,19212,345]` $\rightsquigarrow$ `6406`.

Ayuda: puede asumir que la función `longitud :: [Integer] -> Integer` que devuelve la longitud de una lista de números ya está implementada. Además puede utilizar la función `fromInteger :: Num a => Integer -> a` definida en el Preludio de Haskell para pasar un número de tipo `Integer` a `Float`.

B | B | B | B | B Aprobada

Corrigió Pablo T.

Ej 1:

-- Estan relacionados $\Leftrightarrow$ existe $k \in \mathbb{Z}$ $k \neq 0$ tal que

$a^2 + a.b.k = 0$ $a, b \in \mathbb{Z}$ $a, b \neq 0$

Busqué la forma de traducir esto en algo más claro para mí.

$a^2 + a.b.k = 0$

$a\,(a + b.k) = 0$

$a \neq 0 \Rightarrow a + b.k = 0$

$b.k = -a$

$k = -\frac{a}{b}$ porque $b \neq 0$

Entonces busco una función que me de True si $b \mid -a$ y False si $b \nmid -a$ porque $k \in \mathbb{Z}$

```
estanRelacionados :: Integer → Integer → Bool
estanRelacionados a b = (mod (-a) b == 0)
```

B

Ej 2:

```
prod :: Integer → Integer
prod 1 = 3 * 8
prod n = ((2 * n)^2 + 2 * (2 * n)) * ((2 * n - 1)^2 + 2 * (2 * n - 1))
       * prod (n - 1)
```

-- Aclaración: $3 \cdot 8 = \prod_{i=1}^{2} (i^2 + 2i)$

B

Ej 4:

```
zipPrimos :: [Integer] → [Integer] → [(Integer, Integer)]
zipPrimos [] [] = []
zipPrimos (x:xs) (y:ys)
  | (esPrimo x) && (esPrimo y) = (x,y) : zipPrimos xs ys
  | otherwise = zipPrimos xs ys
```

B

Ej 5:

-- la función esTérminoSuc la planteo de esta forma porque sé que a_n es creciente pues $a_1 < a_2 < a_3$ y para $n > 3$ los términos en a_{n+1} son sumandos positivos.

```
Sucesion :: Integer -> Integer
Sucesion 1 = 1
Sucesion 2 = 2
Sucesion 3 = 5
Sucesion n =  3* (sucesion (n-1))^2 + 2 * (sucesion (n-2)) +
        sucesion (n-3)

esTerminoDeSuc :: Integer -> Integer -> Bool
esTerminoDeSuc n m
  |(sucesion m) == n = True
  |(sucesion m) > n  = False
  |(sucesion m) < n  = esTerminoDeSuc n (m+1)

listaConTerminosSuc :: [Integer] -> [Integer]
listaConTerminosSuc [] = []
listaConTerminosSuc (x:xs)
  |(esTerminoDeSuc x 1 == True) = x : listaConTerminosSuc xs
  |otherwise = listaConTerminosSuc xs

sumaTerminos :: [Integer] -> Float
sumaTerminos [] = 0
sumaTerminos (x:xs) = x + sumaTerminos xs
```

hay que hacer fromInteger a x para que esta suma se pueda hacer.

```
promedioTerminos :: [Integer] -> Float
promedioTerminos xs
  | longitud (listaConTerminosSuc xs) == 0 = 0
  | otherwise = (sumaTerminos (listaConTerminosSuc xs)) /
(fromInteger (longitud (listaConTerminosSuc xs)))
```

B

Ej 3.

```
cantDigitos :: Integer -> Integer
cantDigitos n
  | n < 10 = 1
  | n > 10 = 1 + cantDigitos (div n 10)
```

```
esCapicuaAux :: Integer -> Integer -> Integer -> Bool
esCapicuaAux n m k
  | n < 10 = True
  | m < k = True
  | (iesimo n m == iesimo n k) = esCapicuaAux n (m-1) (k+1)
  | otherwise = False
```

B

```
esCapicua :: Integer -> Bool
esCapicua n = esCapicuaAux n (cantDigitos n) 1
```