Departamento de Computación – FCEyN – UBA

Taller de Álgebra I - Simulacro de parcial

SEGUNDO CUATRIMESTRE 2017 – SIMULACRO

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.
- No está permitido: alterar los tipos de datos presentados en el enunciado – utilizar técnicas no vistas en clase para resolver los ejercicios.

Ejercicio 1

Implementar una función `enProgresion :: Integer -> Integer -> Integer -> Bool` que dados a, b, $c \in \mathbb{N}_{>0}$ determine si los números están en progresión aritmética (en algún orden). Recordar que una sucesión es una progresión aritmética si la diferencia entre cada término y el anterior es constante. *Por ejemplo:*
`enProgresion 5 9 7` $\rightsquigarrow$ `True`
`enProgresion 1 2 4` $\rightsquigarrow$ `False`

Ejercicio 2

Implementar una función `valuacion2Adica :: Integer -> Integer` que dado $n \in \mathbb{Z}_{\neq 0}$ calcule el mayor $a \in \mathbb{N}_{\geq 0}$ tal que 2^a divide a n, es decir, calcule el exponente del 2 en la factorización de n.

Por ejemplo:
`valuacion2Adica 56` $\rightsquigarrow$ `3` (ya que $2^3 = 8$ divide a 56 y no existe otra potencia de 2 mayor a 8 que divida a 56)

Ejercicio 3

Se define recursivamente la sucesión

$$a_1 = 3, \quad a_{n+1} = 2a_n + 3, \qquad n \geq 1.$$

Implementar una función `cuantosTerminos :: Integer -> Integer` que dado $n \in \mathbb{N}_{>0}$ cuente cuántos términos de la sucesión $\{a_i\}_{i \in \mathbb{N}_{>0}}$ son menores que n. *Por ejemplo:*
`cuantosTerminos 13` $\rightsquigarrow$ `2`

Ejercicio 4

Programe la función `esTipoFibonacci :: [Integer] -> Bool` que, dada una lista de al menos tres elementos, devuelve `True` si todos los elementos a partir del tercero son la suma de los dos anteriores y `False` en otro caso.

Por ejemplo:

- `esTipoFibonacci [3,4,7,11,18]` $\rightsquigarrow$ `True`
- `esTipoFibonacci [3,4,6,9,14]` $\rightsquigarrow$ `False`

Ejercicio 5

Programe la función `desplazar :: Integer -> [a] -> [a]`, tal que `desplazar n l` devuelve el resultado de quitar los primeros `n` elementos de `l` y agregarlos al final. Asuma que vale $0 \leq$ `n` $\leq$ `length l`.

Por ejemplo: `desplazar 2 [1,2,3,4,5]` $\rightsquigarrow$ `[3,4,5,1,2]`