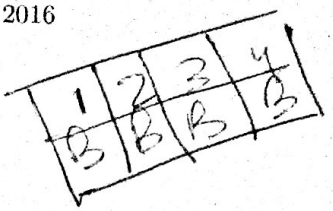
UBA - Facultad de Ciencias Exactas y Naturales - Departamento de Computación- Taller de Álgebra I 3
Parcial - Mayo 2016

LU o DNI
Apellidos:
Nombres:



Diego

Aclaraciones: Para aprobar se requiere al menos tener 2 ejercios bien resueltos.

Ejercicio 1
Programe la función `esFibonacci :: Integer -> Bool`, tal que `esFibonacci n` devuelve `True` si `n` pertenece a la sucesión de Fibonacci. Por ejemplo:
`esFibonacci 5` ⇝ `True`
`esFibonacci 7` ⇝ `False`
Recordar la definición recursiva de la sucesión de Fibonacci: `fib (n+2) = fib n + fib (n+1)`.
Donde además: `fib (1) = fib (2) = 1`

Ejercicio 2
Programar la función `mezcla :: [Integer] -> [Integer] -> [Integer]` tal que mezcla dos listas ordenadas `l1` y `l2`, y devuelve una lista ordenada obtenida de mezclar las dos listas `l1` y `l2` . Por ejemplo:
`mezcla [1,3,3,5] [2,5,9]` ⇝ `[1,2,3,3,5,5,9]`

Ejercicio 3
Programe la función `longitudCamino :: [(Integer,Integer)] -> Float`, tal que `longitudCamino c` es la suma de las distancias entre las tuplas de enteros, que se pueden interpretar como puntos en un plano. Por ejemplo:
`longitudCamino [(1,2)]` ⇝ `0.0`
`longitudCamino [(1,2),(4,6)]` ⇝ `5.0`
`longitudCamino [(1,2),(4,6),(7,10)]` ⇝ `10.0`
La función no esta definida para la lista de tuplas vacía.
Ayuda: Puede ser de utilidad pensar en una función auxiliar distancia entre dos tuplas. La distancia entre dos tuplas (a, b) y (c, d) se considera como $\sqrt{(c-a)^2 + (d-b)^2}$.

Ejercicio 4
Programe la función `incmin :: [Integer]-> [Integer]` tal que `incmin l` es la lista obtenida a partir de sumar, a cada elemento de `l`, el valor mínimo de `l` . Por ejemplo:
`incmin [3,7,4,8,5,9,2,6]` ⇝ `[5,9,6,10,7,11,4,8]`

Ejercicio 1

```
esFibonacci :: Integer → Bool
esFibonacci n = esFibAux n 1

esFibAux :: Integer → Integer → Bool
esFibAux n s
  | fib s == n = True
  | fib s > n = False
  | fib s < n = esFibAux n (s+1)

fib :: Integer -> Integer
fib 1 = 1
fib 2 = 1
fib n = fib (n-1) + fib (n-2)
```

B ✓ ✓

Ejercicio 2

```
mezcla :: [Integer] -> [Integer] -> [Integer]
mezcla xs ys
  | length xs == 0 = ys
  | length ys == 0 = xs
  | head xs <= head ys = head xs : (mezcla (tail xs) ys)
  | otherwise = head ys : (mezcla xs (tail ys))
```

B

Ejercicio 3

```
longitudCamino :: [(Integer, Integer)] -> Float
longitudCamino c
  | length c == 1 = 0
  | otherwise = distancia (head c) (head (tail c)) + longitudCamino (tail c
```

B

```
distancia :: (Integer, Integer) -> (Integer, Integer) -> Float
distancia d h = sqrt (((fst h - fst d)^2) + ((snd h - snd d)^2))
```

Ejercicio 4

```
incmin :: [Integer] -> [Integer]
incmin xs
  | length xs == 0 = []
  | otherwise = inc xs (min xs)

inc :: [Integer] -> Integer -> [Integer]
inc xs i
  | length xs == 0 = []
  | otherwise = ((head xs) + i) : inc (tail xs) i

min :: [Integer] -> Integer
min xs = minAux xs (head xs)
```

✓ ✓ ✓

B

```
minAux :: [Integer] -> Integer -> Integer
minAux xs m
  | length xs == 0 = m
  | head xs < m = minAux (tail xs) (head xs)
  | otherwise = minAux (tail xs) m
```