Centro de Informática - UFPE
Lista de Exercícios — IF686 (23.1)

---

1. Desenvolva o que se pede:
   (a) Implemente a função primos :: [**Int**] −> [**Int**] que, dada uma lista de inteiros, retorna uma nova lista tal que nenhum dos elementos dessa lista é um divisor de qualquer dos elementos posteriores a ele na lista. A função primos pode ser recursiva, considere os seguintes casos: lista vazia e lista da forma(x:xs) Para este último, a solução é a lista (x:l), em que l é obtida aplicando a função primos, recursivamente, a uma lista que não contêm múltiplo de p.
   (b) Defina a função primosN que, dado um inteiro n, retorna uma lista com os primos de 1 a n.

2. Defina uma função `sublistas :: [a] -> [[a]]` que retorna todas as sublistas de uma lista dada como argumento.

   > sublistas [1,2,3]

   [[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]

3. Implemente a função `filtrarEInserir :: [[Int]] -> Int -> ([[Int]], Int)` que retorna uma tupla. O primeiro elemento da tupla é constituído de listas de inteiros tais que a soma dos números ímpares é maior que a soma dos números pares. O segundo elemento consiste no produto entre o segundo argumento da função `filtrarEInserir` e a multiplicação da maior soma obtida das listas retornadas. Utilize obrigatoriamente `filter`.

   > filtrarEInserir [[2,3,4,5,6], [1, 2, 3], [9]] 5
   ([[1,2,3], [9]], 45)

   > filtrarEInserir [[2,3,4,5], []] 7
   ([[2,3,4,5]],98)

   > filtrarEInserir [] 5
   ([],0)

4. Faça o que se pede:
   (a) Defina um tipo algébrico polimórfico Pilha que pode ser exibido.
   (b) Defina as funções: *push*, *pop* e *top*.

5. Considere uma função polinomial de grau 2 ($f(x) = ax^2 + bx + c$), onde a, b e c são os coeficientes do polinômio.
   (a) Defina a função poli :: **Integer** −> **Integer** −> **Integer** −> **Integer** −> **Integer** que recebe como argumentos os coeficientes de uma função polinomial de grau 2 e devolve uma função de inteiro para inteiro (um polinômio)
   (b) Defina a função listaPoli :: [(**Integer**,**Integer**,**Integer**)] −> [**Integer**−>**Integer**] que aguarda uma lista de triplas de inteiros (coeficientes de um polinômio de segundo grau) e devolve uma lista de funções de inteiro para inteiro (polinômios) .
   (c) Defina a função appListaPoli :: [**Integer**−>**Integer**] −> [**Integer**] −> [**Integer**] que recebe uma lista de funções de polinômios e uma lista de inteiros. Esta função devolve uma lista de inteiros que resultam da aplicação de cada polinômio da primeira lista aplicada ao inteiro correspondente na segunda lista. Utilize compreensão de listas.

6. Como conjuntos não levam em consideração ordem e repetições dos elementos, podemos utilizar uma lista ordenada e sem repetições de elementos para a implementação de conjuntos. Por exemplo, o conjunto {2,1,5,7,5,8,5,1} é representado pela lista [1,2,5,7,8]. Considere o seguinte tipo

   ```
   data CInt = Conjunto [Int] deriving (Show)
   ```

   (a) Defina a função makeSet :: [**Int**] −> CInt que, dada uma lista qualquer de inteiros, resulta em um conjunto de inteiros. Considere que a função **sort** (Data.**List**) está disponível
   > makeSet [2,1,5,7,5,8,5,1] = Conjunto [1,2,5,7,8]

(b) Defina a função **union** :: CInt −> CInt −> CInt tal que
union (Conjunto [1,2,3]) (Conjunto [2,3,4,5]) = Conjunto [1,2,3,4,5]

(c) Defina a função mapSet :: (**Int** −> **Int**) −> CInt −> CInt tal que
mapSet (+1) (Conjunto [1,2,3]) = Conjunto [2,3,4]
Utilize a função **map** na implementação da solução.