Lista de Exercícios II

Universidade Federal do Ceará
Campus de Quixadá
Ciência da Computação
Programação Funcional
Prof.º Ricardo Reis

8 de Abril de 2014

Utilizando Haskell, construir as funções seguintes.

1. **paridade**
   INPUT: Lista u de valores booleanos
   OUTPUT: Se o total de *True*s é ímpar então retorne *True* e do contrário *False*
   PROT:
   ```
   paridade ::  [Bool] -> Bool
   ```
   EX(S):
   ```
   paridade [True, True, False] => False
   ```

2. **rev**
   INPUT: Um inteiro positivo x
   OUTPUT: Um inteiro positivo equivalente a x, mas com os dígitos na ordem inversa
   PROT:
   ```
   rev ::  Int -> Int
   ```
   EX(S):
   ```
   rev 3491 ==> 1943
   ```

3. **delete'**
   INPUT: Valor x e lista u
   OUTPUT: Versão de u com a primeira aparição de x removida.
   PROT:
   ```
   delete' ::  (Eq a) => a -> [a] -> [a]
   ```
   EX(S):
   ```
   delete' 5 [1,5,6,9] ==> [1,6,9]
   ```

4. **swap**
   INPUT: Lista u de tipo arbitrário e dois inteiros p e q que representam posições de elementos de u.
   OUTPUT: Versão de u com os elementos das posições p e q trocados
   PROT:
   ```
   swap ::  [a] -> Int -> Int -> [a]
   ```
   EX(S):
   ```
   swap [5,6,7,8,9] 0 3 ==> [8,6,7,5,9]
   ```

5. **nextPerm**
   INPUT: Lista u de elementos ordenáveis.
   OUTPUT: Próxima permutação lexicográfica de u ou lançar exceção se não for possível. A *próxima permutação lexicográfica* de uma lista u de elementos ordenáveis é obtida aplicando-se o seguinte algoritmo,
   (a) Obter o maior valor de índice i de u tal que $u[i] < u[i+1]$ (pode não existir! Neste caso deve-se disparar a exceção).
   (b) Obter o maior índice j de u, com $j > i$, tal que $u[j] > u[i]$.
   (c) Trocar $u[i]$ com $u[j]$.
   (d) Reverter em u a sub-lista que se estende da posição $i+1$ até o final da lista.

   PROT:
   ```
   nextPerm ::  (Ord a) => [a] -> [a]
   ```
   EX(S):
   ```
   nextPerm [1,3,5,2] ==> [1, 5, 2, 3]
   ```

6. **allPerms**
   INPUT: Lista u de chaves não repetidas
   OUTPUT: Todas as permutações possíveis de u em ordem lexicográfica
   PROT:
   ```
   allPerm ¨ [a] -> [[a]]
   ```
   EX(S):
   ```
   allpEM [1,2,3] ==> [[1,2,3], [1,3,2],
   [2,1,3], [2,3,1], [3,1,2], [3,2,1]]
   ```

7. **buscabin**
   INPUT: Lista u de chaves ordenadas ascendentemente e valor x de mesmo tipo base de u
   OUTPUT: Posição de u onde se encontra x ou -1 se $x \notin x$. A busca deve ser binária.
   PROT:
   ```
   buscaBin ::  [a] -> a -> Int
   ```
   EX(S):
   ```
   buscaBin [1,3,5,6,8] 5 ==> 2
   ```

8. **factors**
   INPUT: Número n inteiro positivo
   OUTPUT: Lista de tuplas (f, p) que representam os fatores primos de n omde f denota o fator propriamente dito e p seu respectivo expoente. (Todo número x, tal que $x \in \mathbb{N}$, pode ser reescrito como o produto de potências de bases primas e expoentes naturais. Por exemplo, o número 3361743 pode ser reescrito na forma,

$$3361743 = 3^4 \cdot 7^3 \cdot 11^2$$

Os números 3, 7 e 11 são denominados fatores primos de 3361743 e 4, 3 e 2 seus respectivas expoentes.)
PROT:
```
factors ::  (Integral a) => a -> [(a,a)]
```
EX(S):
```
factors 3361743 => [(3,4),(7,3),(11,2)]
```

9. **listacc**
INPUT: Lista u de inteiros
OUTPUT: Versão v acumulativa de u. Na versão acumulativa a k-ésima chave, v_k é determinada somando-se as todas as chaves de u até a posição k. Matematicamente,

$$v_k = \sum_{j=0}^{k} u_k$$

PROT:
```
listacc ::  (Num a) => [a] -> [a]
```
EX(S):
```
listacc [1,2,3,4] ==> [1,3,6,10]
```

10. **maxsseq**
INPUT: Lista u de inteiros (podem ser positivos, negativos ou zero)
OUTPUT: Sublista de u de elementos consecutivos cuja soma é máxima
PROT:
```
maxsseq (Ord a, Num a) => [a] ->[a]
```
EX(S):
```
maxsseq [2,1,-4,9,7,-1,5] ==> [9,7,-1,5]
```