MC346 - Paradigmas de Programação

Prova Haskell - 11/05/2016

Em cada questão abaixo, você pode definir funções auxiliares à vontade, conforme ache conveniente.

Questão 1 (Valor 2,5) Escreva uma função `rotate k l` em Haskell que recebe um inteiro `k` e uma lista `l` e retorna uma nova lista do mesmo tamanho com os elementos deslocados em `k` posições para a esquerda em modo rotativo, ou seja, supondo que à esquerda da primeira posição está a última posição. Exemplo:

```
ghci> rotate 3 "abcde"
"deabc"
```

Sua função será também avaliada pela eficiência, portanto tente fazer uma função eficiente, isto é, com tempo de processamento proporcional ao tamanho da lista, mesmo que o parâmetro `k` seja bem maior do que este tamanho. Uma boa dica é usar a função `mod`.

Questão 2 (Valor 2,5) A função infixa `div` retorna o resultado da divisão inteira entre seus argumentos. Qual será o resultado da seguinte expressão:

```
map ($ 3) $ map (div) [1,2,3,4,5,6]
```

Explique seu raciocínio.

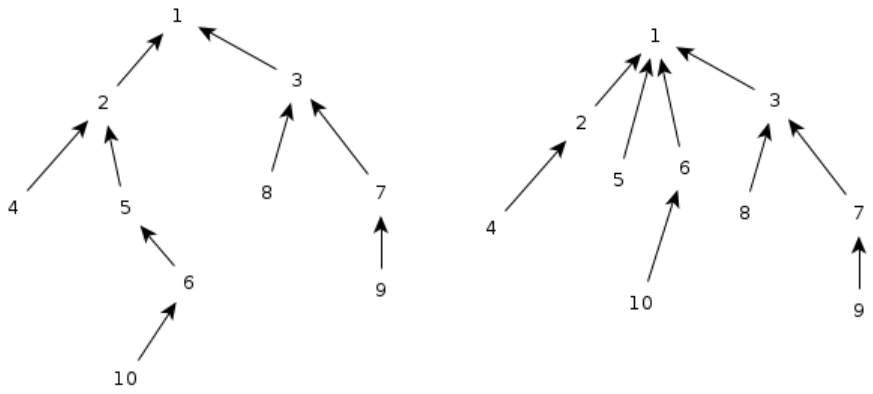
Questão 3 (Valor 2,5) Escreva um programa `quadra.hs` em Haskell que leia um número inteiro da entrada padrão e imprima seu quadrado na saída padrão.

Exemplo:

```
% runghc quadra.hs
Entre com um número:  16
Quadrado:  256
```

Questão 4 (Valor 2,5) Suponha que um mapeamento represente uma estrutura de árvore onde cada nó é mapeado em seu pai (veja exemplos nas figuras a seguir). A raiz não é mapeada em nada (`Nothing`). Escreva uma função `compress` em Haksell que recebe um elemento `x` e um tal mapeamento e retorne um novo mapeamento igual ao anterior só que com todos os elementos do caminho de `x` até a raiz (exceto a própria raiz) apontando diretamente para a raiz.

Por exemplo, se `m` se refere ao mapeamento que representa a figura à esquerda a seguir, o resultado de `compress 6 m` é a figura à direita a seguir.



No interpretador Haskell, supondo que a função `compress` tenha já sido definida, este exemplo seria:

```
ghci> :m + Data.Map
ghci> let t=[(2,1),(3,1),(4,2),(5,2),(8,3),(7,3),(9,7),(6,5),(10,6)]
ghci> let m = fromList t
ghci> compress 6 m
fromList [(2,1),(3,1),(4,2),(5,1),(6,1),(7,3),(8,3),(9,7),(10,6)]
```

Nota: a função

```
lookup :: (Ord k) => k -> Map k v -> Maybe v
```

pode ser útil. O resultado de `lookup x m` é `Just p` se `p` é o pai de `x` em `m` ou `Nothing` se `x` não tem pai (ou seja, é uma raiz).

Boa sorte!