Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
Linguagens de Programação: Conceitos e Paradigmas

Exercícios de Programação em Haskell

Aluno (a):_____

1) Construa a função **ocorrencias::Int->[Int]->Int** que, dada uma lista de inteiros e um valor inteiro, retorne o número de ocorrências desse valor na lista. Exemplos de uso:

   ocorrencias 1 [1,3,4,1,3,1] = 3
   ocorrencias 2 [1,3,4,1,3,1] = 0

2) Construa a função **unicos::[Int]->[Int]** que, dada uma lista de inteiros, retorna uma lista contendo os valores que ocorrem apenas uma vez na lista de entrada. Exemplos de uso:

   unicos [1,3,4,1,3,1,2] = [4,2]
   unicos [1,3,1,3,1] = []

3) Defina a função **desloque::[Int]->[Int]** que, dada uma lista de inteiros, retorna uma lista correspondente à original deslocada uma posição à esquerda. Exemplo de uso:

   desloque [1,2,3,4,5] = [2,3,4,5,1]

4) Defina a função **pa::[Int]->[(Int,Int,Int)]** que, dada uma lista de inteiros, retorna uma lista contendo tuplas que descrevem progressões aritméticas formadas com os elementos de entrada. Cada tupla descreve dois termos consecutivos da progressão e a razão. Exemplo:

   pa [3,5,2] = [(3,5,2),(2,5,3)]

5) Construa uma função **elimina::[Int]->Int->[Int]** que, dada uma lista de inteiros e um número inteiro, retorne uma lista onde todas as ocorrências desse número foram eliminadas. Exemplo de uso:

   elimina [1,3,4,1,3,2] 1 = [3,4,3,2]

6) Uma lista é uma *sublista* de outra se todos os elementos da primeira estão presentes na segunda, na mesma ordem. Por exemplo, [1,3] é uma sublista de [1,2,3,4], mas não é uma sublista de [4,3,2,1]. Uma lista é uma *subseqüência* de outra se for uma sublista e todos os elementos ocorrerem em um único bloco. Por exemplo, [1,3] é uma subseqüência de [1,3,4], mas não é uma subseqüência de [1,2,3,4]. Defina funções para testar se uma lista é sublista e subseqüência de outra.

7) Dado um número inteiro positivo n, construa uma lista com todos os pares de inteiros positivos cuja soma de elementos é igual a n.

8) Uma lista associativa é uma lista de pares (tuplas) onde o primeiro item corresponde à chave e o segundo é o valor de interesse. Por exemplo, tratando-se de pessoas físicas, a chave pode ser o CPF e o valor pode ser um registro que guarda informações importantes sobre uma pessoa. Pensando nos casos excepcionais, defina:

   a) O tipo de dado **LAssoc a b** para representar listas associativas compostas por pares formados por uma chave de tipo a e um valor de tipo b, dados nesta ordem.

   b) Função **alist_find :: LAssoc a b -> a -> Bool** que, dada uma lista associativa l e uma chave k, verifica se existe algum registro com a chave k.

c) Função **alist_access :: LAssoc a b -> a -> b** que, dada uma lista associativa l e uma chave k, retorna o valor associado à chave k.

d) Função **alist_remove :: LAssoc a b -> a -> LAssoc a b** que, dada uma lista associativa l e uma chave k, retorna uma nova lista associativa onde o par com chave k foi removido.

e) Função **alist_insert :: LAssoc a b -> a -> b -> LAssoc a b** que, dada uma lista associativa l, uma chave k, e um registro r, retorna a lista associativa com o par (k,r) inserido. Se já existe um registro associado à chave k, este deve ser atualizado com o novo valor r.

9) Em Ciência da Computação, uma árvore de busca binária é uma árvore binária tal que:

(a) Cada vértice v possui um valor rotulo(v) associado;
(b) Seja v_e o filho à esquerda do vértice v. v_e corresponde à raiz da sub-árvore formada pelos vértices com valores menores que v;
(c) Seja v_d o filho à direita do vértice v. v_d é a raiz da sub-árvore formada pelos vértices com valores maiores que v.

Os valores são relevantes na árvore de busca binária, cujo objetivo é estruturar os dados de forma flexível permitindo pesquisa binária. A seguir, descrevemos algumas operações básicas sobre árvores de busca binária:

Buscas

Para buscar um valor chave em uma árvore binária, começamos examinando o vértice raiz v. Se chave for igual a rotulo(v), o valor foi encontrado e a busca se encerra. Se valor chave for menor que rotulo(v), a busca deve continuar na sub-árvore à esquerda de v (v_e) e assim recursivamente. Similarmente, se o valor chave for maior que rotulo(v), então devemos procurá-lo na sub-árvore à direita de v (v_d). Se a busca alcançar um vértice folha da árvore, concluímos que a árvore não possui o valor chave.

Inserções

A inserção começa com a busca pelo valor chave a ser inserido. Se o valor chave for encontrado, a árvore binária não será modificada e o processo de inserção se encerrará. Se o valor chave não existir na árvore, alcançaremos um vértice folha da árvore, onde iremos inserir o valor chave. Em outras palavras, examinamos a raiz v e introduzimos um vértice na sub-árvore à esquerda (v_e) se o valor chave é menor que rotulo(v), ou na sub-árvore à direita (v_d) se o valor chave for maior que rotulo(v).

Considerando que árvores de busca binária ou são vazias (**Vazia**) ou possuem vértices v com uma sub-árvore à esquerda (v_e), um valor inteiro (rotulo(v)) e uma sub-árvore à direita (v_d), tais que:

data Int_Arvbin = Vazia | Vertice (Int_Arvbin, Int, Int_Arvbin)

Pede-se a criação das seguintes funções:

- **busca :: Int -> Int_Arvbin -> Bool**, que realiza buscas por uma chave em uma árvore de busca binária.
- **insercao :: Int -> Int_Arvbin -> Int_Arvbin**, que toma uma árvore de busca binária e um valor, inserindo-o na árvore.
- **altura :: Int_Arvbin -> Int**, que calcula a altura de uma árvore binária, isto é, o maior número arestas (ligações) conectando a raiz da árvore a uma folha específica.
- **naovazios :: Int_Arvbin -> Int**, que calcula o número de nós não-vazios de uma árvore.