Questões de Haskell

Listas

Problema 3

(*) Encontre o K-ésimo elemento de uma lista.

O primeiro elemento da lista é o número 1. Exemplo:

```
* (element-at '(a b c d e) 3)
c
```

Exemplo em Haskell:

```
λ > elementAt [1,2,3] 2
2

λ> elementAt "haskell" 5
'e'
```

Problema 6

(*) Escreva o código da função isPalindrome, que determina se uma lista/palavra é ou não um palíndromo. Um palíndromo pode ser lido da esquerda para a direita ou da direita para a esquerda; por exemplo, (x a m a x).

Exemplo em Haskell:

```
λ> isPalindrome [1,2,3]
False

λ> isPalindrome "madamimadam"
True

λ> isPalindrome [1,2,4,8,16,8,4,2,1]
True
```

Problema 8

(*) Escreva a função compress, que elimina duplicatas consecutivas de uma lista de elementos. Se uma lista contém elementos repetidos consecutivamente, eles devem ser recolocados por uma única cópia do elemento. A ordem dos elementos não deve ser alterada.
Exemplo em Haskell:

```
λ> compress "aaaabccaadeeee"
"abcade"
```

Problema 18

(**) Extraia um slicing de uma lista. Dado dois índices, um i e um k, o slicing é uma nova lista contendo os elementos entre o i-ésimo e o k-ésimo elemento da lista original (ambos os limites inclusos). Comece contando os elementos com 1.

Exemplo em Haskell:

```
λ> slice ['a','b','c','d','e','f','g','h','i','k'] 3 7
"cdefg"
```

Problema 19

(**) Rotacione uma lista N posições para a esquerda.Rotacionar uma lista significa “girar” seus elementos, movendo itens do início para o fim sem alterar a ordem relativa dos valores. Dica: utilize as funções predefinidas length e (++).

Exemplo em Haskell:

```
λ> rotate ['a','b','c','d','e','f','g','h'] 3
"defghabc"

λ> rotate ['a','b','c','d','e','f','g','h'] (-2)
"ghabcdef"
```

Problema 26

(**) Gere combinações de K objetos distintos escolhidos a partir dos N elementos de uma lista.

De quantas maneiras um comitê de 3 pessoas pode ser escolhido em um grupo de 12 pessoas? Sabemos que existem C(12,3) = 220 possibilidades (C(N,K) denota os conhecidos coeficientes binomiais). Para matemáticos puros, esse resultado pode ser ótimo. Mas o que realmente queremos é gerar todas as possibilidades em uma lista.

Exemplo:

```
* (combinations 3 '(a b c d e f))
((A B C) (A B D) (A B E) ... )
```

Exemplo em Haskell:

```
λ> combinations 3 "abcdef"
["abc","abd","abe",...]
```

A solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/26

Problema 28

Ordenar uma lista de listas de acordo com o comprimento das sublistas.

a) Suponhamos que uma lista contenha elementos que também são listas. O objetivo é ordenar os elementos dessa lista de acordo com seu comprimento. Por exemplo, listas curtas primeiro,

listas longas depois, ou vice-versa.

Exemplo:

```
* (lsort '((abc) (de) (fgh) (de) (ijkl) (mn) (o)))
((O) (DE) (DE) (MN) (ABC) (FGH) (IJKL))
```

Exemplo em Haskell:

```
λ> lsort ["abc","de","fgh","de","ijkl","mn","o"]
["o","de","de","mn","abc","fgh","ijkl"]
```

b) Novamente, supomos que uma lista contenha elementos que também são listas. Mas desta vez o objetivo é ordenar os elementos dessa lista de acordo com a frequência de seu comprimento ; ou seja, no padrão, em que a ordenação é feita em ordem crescente, as listas com comprimentos raros são colocadas primeiro, e as com comprimentos mais frequentes vêm depois.

Exemplo:

```
* (lfsort '((abc) (de) (fgh) (de) (ijkl) (mn) (o)))
((ijkl) (o) (abc) (fgh) (de) (de) (mn))
```

Exemplo em Haskell:

```
λ> lfsort ["abc", "de", "fgh", "de", "ijkl", "mn", "o"]
["ijkl","o","abc","fgh","de","de","mn"]
```

Solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/28

Aritmética

Problema 35

(**) Determine os fatores primos de um número inteiro positivo dado.

Construa uma lista simples contendo os fatores primos em ordem crescente.

Exemplo:

```
* (prime-factors 315)
(3 3 5 7)
```

Exemplo em Haskell:

```
λ> primeFactors 315
[3, 3, 5, 7]
```

A solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/35

Problema 39

(*) A list of prime numbers in a given range.

Given a range of integers by its lower and upper limit, construct a list of all prime numbers in that range.

Example in Haskell:

```
λ> primesR 10 20
[11,13,17,19]
```

A solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/39

Problema 40

```
# Problema 40

A conjectura de Goldbach afirma que todo número par positivo maior que 2 é a
soma de dois números primos. Exemplo: 28 = 5 + 23. É um dos fatos mais
famosos da teoria dos números que ainda não foi comprovado para o caso
geral. Sua veracidade foi confirmada numericamente até números muito grandes
(muito maiores do que os que podemos processar com nosso sistema Prolog).
Escreva um predicado para encontrar os dois números primos cuja soma resulta
em um dado número inteiro par.
```

```
λ> goldbach 28
(5, 23)
```

solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/40

Lógica e Códigos

Problema 46

(**) Tabelas verdade para expressões lógicas.Soluções

Defina predicados and/2, or/2, nand/2, nor/2, xor/2, impl/2 e equ/2 (para equivalência lógica) que têm sucesso ou falham de acordo com o resultado de suas respectivas operações; por exemplo, and(A,B) terá sucesso se e somente se A e B tiverem sucesso.

Uma expressão lógica em duas variáveis pode então ser escrita como no exemplo a seguir: and(or(A,B),nand(A,B)).

Agora, escreva uma tabela de predicados/3 que imprima a tabela verdade de uma dada expressão lógica em duas variáveis.

Exemplo:

```
(table A B (and A (or A B)))
true true true
true fail true
fail true fail
fail fail fail
```

Exemplo em Haskell:

```
λ> table (\a b -> (and' a (or' a b)))
True True True
True False True
False True False
False False False
```

Problema 49

(**) Códigos Gray.Soluções

Um código Gray de n bits é uma sequência de cadeias de n bits construídas de acordo com certas regras. Por exemplo,

```
n = 1: C(1) = ['0','1'].
n = 2: C(2) = ['00','01','11','10'].
n = 3: C(3) = ['000','001','011','010','110','111','101','100'].
```

Descubra as regras de construção e escreva um predicado com a seguinte especificação:

```
% gray(N,C) :- C is the N-bit Gray code
```

É possível aplicar o método de "cache de resultados" para tornar o predicado mais eficiente quando ele for usado repetidamente?

Exemplo em Haskell:

```
λ> gray 3
["000","001","011","010","110","111","101","100"]
```

Árvores Binárias

Problema 56

Chamamos uma árvore binária de simétrica se for possível traçar uma linha vertical através do nó raiz de modo que a subárvore direita seja a imagem espelhada da subárvore esquerda. Escreva um predicado ‘symmetric/1’ que verifique se uma dada árvore binária é simétrica.

Dica: escreva primeiro um predicado ‘mirror/2’ para verificar se uma árvore é a imagem espelhada de outra. Estamos interessados apenas na estrutura da árvore, e não no conteúdo dos nós.

Example in Haskell:

```
λ> symmetric (Branch 'x' (Branch 'x' Empty Empty) Empty)
False
λ> symmetric (Branch 'x' (Branch 'x' Empty Empty) (Branch 'x' Empty Empty))
True
```

solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/56

Problema 57

(**) Árvores de busca binária.

Use a definição de árvore binária de busca (BST) para escrever um função” construa” que constrói uma árvore binária de busca a partir de uma lista de números inteiro

Example:

```
* construct([3,2,5,7,1],T).
T = t(3, t(2, t(1, nil, nil), nil), t(5, nil, t(7, nil, nil)))
```

Then use this predicate to test the solution of Problem 56.

Example:

```
* test-symmetric([5,3,18,1,4,12,21]).
Yes
* test-symmetric([3,2,5,7,4]).
No
```

Example in Haskell:

```
λ> construct [3, 2, 5, 7, 1]
Branch 3 (Branch 2 (Branch 1 Empty Empty) Empty) (Branch 5 Empty (Branch 7 Empty Empty))
λ> symmetric . construct $ [5, 3, 18, 1, 4, 12, 21]
True
λ> symmetric . construct $ [3, 2, 5, 7, 1]
True
```

solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/57

Problema 61

Conte as folhas de uma árvore binária. Uma folha é um nó sem sucessores (isto é, sem filhos).

Escreva um predicado ‘count_leaves/2’ para contar o número de folhas da árvore.

Example:

```
% count_leaves(T,N) :- the binary tree T has N leaves
```

Example in Haskell:

```
λ> countLeaves tree4
2
```

solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/61

Problema 61

Colete os nós internos de uma árvore binária em uma lista. Um nó interno de uma árvore binária tem um ou dois sucessores não vazios. Escreva um predicado internals/2 para coletá-los em uma lista.

Example:

```
% internals(T,S) :- S is the list of internal nodes of the binary tree T.
```

Example in Haskell:

```
λ> internals tree4
[1,2]
```

solução: https://wiki.haskell.org/index.php?title=99_questions/Solutions/62

Problemas Diversos

Problema 95

(**) Números em inglês por extenso

Em documentos financeiros, como cheques, os números às vezes precisam ser escritos por extenso. Exemplo: 175 deve ser escrito como um-sete-cinco. Escreva um predicado `full-words/1` para imprimir números inteiros (não negativos) por extenso.

Exemplo em Haskell:

Problemas Indicadas

EVERTON DANIEL DE LIMA ROMUALDO

Problema Indicado 1

```
# Dando o troco, Literalmente

Imagine uma máquina de vendas automática. O sistema dela precisa de
tempos em tempos devolver um troco que sobrou da compra de um cliente.
Implemente uma função "dandoOTroco" que dar o troco ao cliente usando o
menor número de moedas possíveis. A função recebe 2 parâmetros:
   ● Um valor inteiro representando o troco total a ser pago
   ● Uma lista de inteiros das moedas disponíveis no cofre da máquina.
     Assuma que você pode usar a mesma moeda várias vezes e que a lista
     sempre está ordenada do maior para o menor.
```

Problema indicado 2

```
# O roma é complicado

Crie um programa capaz de perceber que **roma** é um anagrama para
**amor**.

Dizemos que duas palavras são anagramas quando elas são compostas
exatamente pelas mesmas letras e na mesma quantidade, variando apenas a
ordem. Exemplo: "roma" e "amor".

Escreva a função detectaAnagrama que recebe duas Strings e retorna um
valor booleano indicando se elas são anagramas uma da outra.
```

Problema Indicado 3

```
# Calculadora Básica

Escreva um programa que simula uma calculadora básica. Deve receber como
entrada uma lista string que representa uma expressão aritmética com as
operações básicas de: soma(+), subtração(-), divisão(/),
multiplicação(*).

Por exemplo: 1+2*3 que resulta em 7.

O programa deve ser capaz de realizar a operação corretamente aplicando
as regras de precedências das operações. Desconsidere a existência de
qualquer outra coisa que não seja as 4 operações e os valores inteiros.
Assuma também que só existem números positivos maiores que 0.
```