Lazy evaluation, listas e listas infinitas

4.1 Usando a definição da lista infinita *primos* apresentada nas aulas teóricas, escreva uma função $factores :: Int \to [Int]$ para factorizar um inteiro positivo em primos. Exemplo: $factores\ 100 = [2, 2, 5, 5]$ porque $100 = 2 \times 2 \times 5 \times 5$.

4.2 Considere duas séries (i.e. somas infinitas) que convergem para π:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots \tag{1}$$

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \cdots \tag{2}$$

Escreva duas funções `calcPi1, calcPi2 :: Int -> Double` que calculam um valor aproximado de π usando o número de parcelas dado como argumento; investigue qual das séries converge mais depressa para π.

Sugestão: construa listas infinitas para os numeradores e denominadores dos termos separadamente e combine-as usando *zip*/*zipWith*.

4.3 Escreva uma definição recursiva duma função $intercalar :: a \to [a] \to [[a]]$ tal que $intercalar\ x\ ys$ que obtém todas formas possíveis de intercalar x com os elementos em ys. Exemplo: $intercalar\ 1\ [2, 3] = [[1, 2, 3],\ [2, 1, 3],\ [2, 3, 1]]$.

4.4 Escreva uma definição da função $perms :: [a] \to [[a]]$ que obtém todas as permutações de uma lista (a ordem das permutações não é importante). Exemplo: $perms\ [1, 2, 3] = [[1, 2, 3], [2, 1, 3], [2, 3, 1], [1, 3, 2], [3, 1, 2], [3, 2, 1]]$.

Sugestão: utilize a função *intercalar* do exercício anterior para obter todas as formas de intercalar um elemento numa lista.

4.5 Podemos tornar a cifra de César da Folha 2 um pouco mais difícil de quebrar usando uma palavra chave em vez de um deslocamento único. Começamos por repetir a palavra-chave (por exemplo, "LUAR") ao longo do texto da mensagem; cada letra da chave de 'A' a 'Z' fazemos corresponder um índice de deslocamento de 0 a 25 (e.g., "LUAR" corresponde aos deslocamentos 11, 20, 0 e 17).

A	T	A	Q	U	E	D	E	M	A	D	R	U	G	A	D	A
L	U	A	R	L	U	A	R	L	U	A	R	L	U	A	R	L
L	N	A	H	F	Y	D	V	X	U	D	I	F	A	A	U	L

Escreva uma função $cifraChave :: String \to String \to String$ que implemente esta variante da cifra de César.

4.6 Neste exercício pretende-se definir o triângulo de Pascal completo como uma *lista infinita* $pascal :: [[Int]]$ das linhas do triângulo.

(a) Escreva uma definição de *pascal* usando a função *binom* do Exercício 5 da Folha 1. Note que que $pascal\,!!\,n\,!!\,k = binom\ n\ k$, para quaisquer n e k tais que $n > 0$ e $0 \le k \le n$.

(b) Escreva outra definição que evite o cálculo de factoriais usando as seguintes propriedades de coeficientes binomiais:

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad\qquad \binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \quad (\text{se } n > k)$$

Árvores de pesquisa

Nos exercícios seguintes considere a definição do tipo de árvores de pesquisa apresentada na aulas teórica:

$$\mathit{data}\ \mathit{Arv}\ a = \mathit{Vazia} \ \mid\ \mathit{No}\ a\ (\mathit{Arv}\ a)\ (\mathit{Arv}\ a)$$

4.7 Escreva uma definição recursiva da função $\mathit{sumArv} :: \mathit{Num}\ a \Rightarrow \mathit{Arv}\ a \rightarrow a$ que soma todos os valores duma árvore binária de números.

4.8 Baseado-se na função $\mathit{listar} :: \mathit{Arv}\ a \rightarrow [a]$ apresentada na aula teórica, escreva a definição duma função para listar os elementos duma árvore de pesquisa por *ordem decrescente.*

4.9 Escreva uma definição da função $\mathit{nivel} :: \mathit{Int} \rightarrow \mathit{Arv}\ a \rightarrow [a]$ tal que $\mathit{nivel}\ n\ \mathit{arv}$ é a lista ordenada dos valores da árvore no nível n, isto é, a uma altura n (considerando que a raiz tem altura 0).

4.10 Escreva uma definição da função de ordem superior $\mathit{mapArv} :: (a \rightarrow b) \rightarrow \mathit{Arv}\ a \rightarrow \mathit{Arv}\ b$ tal que '$\mathit{mapArv}\ f\ t$' aplica uma função f a cada valor duma árvore t.

4.11 Usando as duas funções *construir* (usando inserções simples e partições binárias) dadas nas teóricas experimente usar o interpretador de Haskell para calcular a altura de árvores de pesquisa com n valores.

(a) usando o método de partições binárias, e.g. $\mathit{construir}\ [1..n]$;

(b) usando inserções simples, e.g. $\mathit{foldr}\ \mathit{inserir}\ \mathit{Vazia}\ [1..n]$;

Experimente com $n = 10$, 100 e 1000 e compare a altura obtida com o minorante teórico: uma árvore binária com n nós tem altura $\geq \log_2 n$.

4.12 Neste exercício pretende-se implementar uma variante da remoção de um valor duma árvore de pesquisa simples.

(a) Baseando-se na função $\mathit{mais\_esq} :: \mathit{Arv}\ a \rightarrow a$ apresentada na aula teórica, escreva uma definição da função $\mathit{mais\_dir} :: \mathit{Arv}\ a \rightarrow a$ que obtém o valor mais à direita numa árvore (i.e., o maior valor).

(b) Usando a função da alínea anterior, escreva uma definição alternativa da função $\mathit{remover} :: \mathit{Ord}\ a \Rightarrow a \rightarrow \mathit{Arv}\ a \rightarrow \mathit{Arv}\ a$ que use o valor mais à direita da sub-árvore esquerda no caso de um nó com dois descentes não-vazios.