1ª Lista de Exercícios de Paradigmas de Linguagens Computacionais
Professor: Fernando Castor
Monitores:

Ciência da computação: Alberto Costa Jr. (arcj), Diego Aragão (dca), Hugo Bessa (hrba), João Victor Leite (jvfl), Luana Martins (lms7), Luiz Fernando Sotero (lfss2), Vítor Antero (vham)

CIn-UFPE – 2013.1
Disponível desde: 06/06/2013
Entrega: 18/06/2013

A lista deverá ser respondida **em dupla**. A falha em entregar a lista até a data estipulada implicará na perda de 0,25 ponto na **média** da disciplina para os membros da dupla. Considera-se que uma lista na qual **menos que 10** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 17** das questões corretamente respondidas implica em um acréscimo de 0,125 ponto na média da disciplina para os membros da dupla. Se **qualquer situação de cópia de respostas** for identificada, os membros **de todas as duplas envolvidas** perderão 0,5 ponto na média da disciplina. O mesmo vale para respostas obtidas a partir da Internet. As respostas deverão ser entregues **exclusivamente em formato texto ASCII** (nada de .pdf, .doc, .docx ou .odt) e deverão ser enviadas para o monitor responsável por sua dupla, **sem** cópia para o professor. Devem ser organizadas em arquivos separados, um por questão, entregues em um único arquivo compactado, ou seja, um único arquivo .zip contendo as respostas para todas as questões. Um membro de cada dupla deve ir até a página da monitoria correspondente (CC ou EC) e registrar os nomes e logins dos membros da sua dupla sob o nome de um monitor. A escolha do monitor deve seguir uma política *round-robin* de modo a balancear a carga de duplas entre os monitores da maneira mais equitativa possível. A não-observância desta política pode resultar na transferência da dupla para outro monitor.

1) Crie uma função que, dados dois números x e y, retorne como resultado o m.d.c. de x e y.

```
*Main> mdc 15 6
3
```

2) Crie uma função que, dado um número inteiro positivo x, verifique se x é primo ou não. Lembre-se de utilizar o crivo de Eratóstenes para deixar a sua função mais otimizada.

```
*Main> ehPrimo 20
False
*Main> ehPrimo 13
True
```

3) Dados dois pontos num espaço tridimensional, faça uma função distância e um tipo ponto de tal forma que a função calcule a distância entre dois pontos passados como parâmetros e tenha a assinatura:

distancia :: Ponto -> Ponto -> Double

***Main>** distancia (1.0,2.0,3.0) (2.0,3.0,4.0)
1.7320508075688772

4) Simule a cifra ADFGVX. Para isto, crie a função *cifraMessage :: String -> String -> String*, sendo os parâmetros a mensagem a ser enviada e a chave para encriptação nessa ordem. O retorno é a mensagem criptografada. Utilize a seguinte tabela como *Quadrado de Políbio*:
table = [('a', "n478rp"), ('d', "1cekml"), ('f', "ht6u0j"), ('g', "d52asx"), ('v', "ziqovb"), ('x', "3y9fwg")]
Para melhor compreensão de *table*:
O char 'p' é criptografado para "xa", pois 'p' é o último char de "n478rp" e a chave é o char 'a'.

***Main>** cifraMessage "a enciclopedia livre" "muriel"
"ad xd df dv vg da ad av vd gd xf gv fd ga xv gd dd gv"

Observações importantes: Não precisa considerar os espaços em branco nessa encriptação. Considere que as strings passadas como parâmetro contêm apenas caracteres minúsculos.
Fonte: http://pt.wikipedia.org/wiki/Cifra_adfgvx

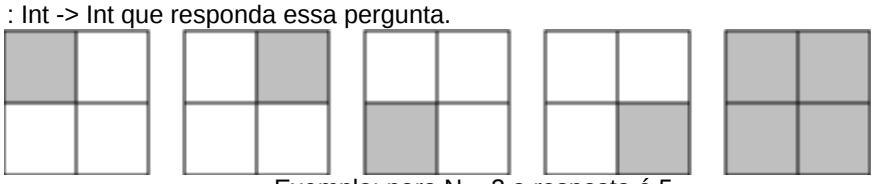
5) Desenvolva uma função palindromoDecimal :: String -> String que, recebendo um número inteiro positivo N em hexadecimal (representado como um String), seja capaz de verificar se este número, transformado para a base decimal, é lido exatamente da mesma forma de frente para trás ou de trás para frente, ou seja, se este número na base decimal é um palíndromo.

Obs.: Não é permitido o uso da função **reverse**.

- Exemplos:

***Main>** palindromoDecimal "1E1B9"
"123321 - PALINDROMO"
***Main>** palindromoDecimal "7C1"
"1985 - NAO-PALINDROMO"
***Main>** palindromoDecimal "7C13"
"31763 - NAO-PALINDROMO"
***Main>** palindromoDecimal "16FD"
"5885 - PALINDROMO"

6) Quantos quadrados diferentes existem em um quadriculado NxN(para N>=1)? Crie uma função : : Int -> Int que responda essa pergunta.



Exemplo: para N = 2 a resposta é 5.

***Main>** function 2
5
***Main>** function 1
1
***Main>** function 8
204

7) Você está coletando informações do Facebook para seu novo app e deseja filtrar os usuários de acordo com o gênero e a idade. As informações sempre vem em uma lista de tuplas (String username, Char gender, Int age). Utilizando compreensão de listas, liste o nome de todos usuários do sexo masculino com idade a partir de 18 anos. Note que o Char gender pode assumir apenas os valores 'F' ou 'M'.

***Main>** fbfilter [("Bidu",'M',16), ("Franjinha",'M',19), ("Magali",'F',17), ("Tina",'F',24), ("Pipa",'F',25), ("Cebolinha",'M',18), ("Monica",'F',17), ("Jeremias", 'M',20)]
["Franjinha","Cebolinha","Jeremias"]

8) - Imagine que você é um ladrão e acabou de assaltar uma loja. Você tem pouco tempo para decidir o que vai levar, pois a polícia está a caminho. Tudo o que você possui é uma mochila, de capacidade limitada, e pode escolher entre um conjunto de itens, cada um com um lucro e peso associados. Determine o maior lucro possível que você pode obter nesta situação criando uma função que receba uma lista de itens em que cada item é representado por uma 3-upla (String,Int,Int) composta por nome, peso e lucro, respectivamente, e um inteiro representando a capacidade da mochila e retorne um par (Int,[String]) sendo o primeiro elemento o maior lucro possível e o segundo a lista dos itens roubados.

knapsack :: [(String, Int, Int)] -> Int -> (Int, [String])

***Main>** knapsack [("A",5,1),("B",3,4),("C",1,3),("D",6,10),("E",2,6)] 8
(16,["E","D"])
***Main>** knapsack [("A",4,2),("B",2,1),("C",1,3),("D",2,4)] 5
(8,["D","C","B"])

9) Crie uma função de assinatura frequencia :: String -> [Char] -> [(Char,Int)]. Esta função receberá uma String, e uma lista de Char's. O retorno será uma lista de tuplas indicando o número de ocorrências na String de cada Char da lista recebida como parâmentro. Os primeiros elementos das tuplas serão os Char's da lista e os segundos elementos serão as ocorrências dos respectivos Char`s na String. A lista deverá ser ordenada em função do **número de ocorrências**, e caso este seja igual, em **ordem alfabética.**

***Main>** frequencia "ABELHUDA" ['A','L', 'E', 'C']
[('A',2),('E',1),('L',1),('C',0)]

10) Crie uma função que receba uma lista e uma função e ordenação e retorne a lista ordenada.

Ex.:
```
;a funcao ordenacaoStr retorna se a primeira string é menor que a outra lexicograficamente
ordenacaoStr :: String -> String -> Boolean
ordenacaoStr "" "" = False
ordenacaoStr x "" = False
ordenacaoStr "" y = True
ordenacaoStr (x:xs) (y:ys)
        | x == y = ordenacaoStr xs ys
        | x > y = False
        | x < y = True

ordenaçãoInt :: Int -> Int -> Boolean
ordenacaoStr x y
        | x >= y = False
        | otherwise True
```

***Main>** sortBy ["Pedro", "Amanda", "Alberto"] ordenacaoStr
["Alberto", "Amanda", "Pedro"]

***Main>** sortBy [1, -3, 10, -23] ordenacaoInt
[-23, -3, 1, 10]

11) Crie uma função *zipLists* que dada 3 listas de tipos quaisquer, gera como retorno uma lista de tuplas com elementos de cada lista passada como parâmetro.
***Main>** zipList [] [] []
[]
***Main>** zipList [1,2,3] [4,5,6] [7,8,9,10]
[(1,4,7),(2,5,8),(3,6,9)]
***Main>** zipList "Nome" [1,2,3,4] (cycle [True, False])
[('N',1,True),('o',2,False),('m',3,True),('e',4,False)]

12) Implemente uma versão otimizada do algoritmo Quicksort que não precisa passar pela lista de elementos de entrada duas vezes para cada chamada recursiva, diferentemente do que é feito na versão apresentada em sala de aula. Seu quicksort deve funcionar para listas de tipos cujos valores possam ser comparados para saber se um é maior que o outro.

13) Defina uma função comparaConjuntos :: (Eq t) => [t] -> [t] -> String que responda se o primeiro conjunto A contém o segundo conjunto B, se B contém A, se há interseção entre eles, se eles são disjuntos ou se eles são iguais. Caso A contenha B, a saída deve ser "A contem B"; caso B contenha A, a saída deve ser "B contem A"; caso haja interseção, mas nenhum conjunto contenha o outro, a saída deve ser "A interseciona B"; caso não haja nenhum elemento em comum, a saída deve "Conjuntos disjuntos"; caso os conjuntos sejam iguais, a saída deve ser "A igual a B".
Obs1: Nos conjuntos, a ordem e a quantidade de vezes que os elementos estão listados na coleção não é relevante.

***Main>** comparaConjuntos [1,2,3,4,5] [3,4]
"A contem B"
***Main>** comparaConjuntos "banana" "nanaba"
"A igual a B"
***Main>** comparaConjuntos "caco" "macaco"
"B contem A"
***Main>** comparaConjuntos [True] [False]
"Conjuntos disjuntos"
***Main>** comparaConjuntos ['b','o','l','a'] ['b','u','l','e']
"A interseciona B"

14) Crie uma função fSplitWith:: Eq a => [a] -> a -> Int -> [[a]] que separe os itens do primeiro argumento utilizando o separador indicado no segundo argumento. A função deverá então retornar uma lista contendo as sublistas que foram separadas dessa maneira, e somente aquelas cujo tamanho não excede o Int passado como parâmetro. Exemplo:

***Main>** fSplitWith "esfera cubo paralelepipedo" ' ' 8
["esfera","cubo"]
***Main>** fSplitWith [True, True, False, True, False, True] False 1
[[True],[True]]
***Main>** fSplitWith [True, True, False, True, False, True] False 2
[[True, True],[True],[True]]

15) Dada uma lista de tuplas formadas por um Double x e uma função unária f que retorna um Double, crie uma função retorne uma lista de reais, sendo cada um deles o resultado de f(x).
Obs: É obrigatório o uso da função **map** da biblioteca padrão de haskell.

***Main>** aplicaFuncoes [(1, (+2)) , (2, (*5)) , (3, (-7)) , (4, (/8))]
[3.0, 10.0, -4.0, 0.5]

16) Faça uma função ordenarTuplas que, dados uma lista de tuplas (Ex: [('a','b')]) e um inteiro como parâmetros, deve ordenar a lista a partir da chave indicada pelo inteiro. Essa chave é o campo da tupla que será usado como base para comparação, 0 se for o primeiro elemento, 1 se for o segundo.
Obs: Deve ser usada compreensão de listas.

***Main>** ordenarTuplas [('b',3),('c',2),('a',1)] 0
[('a',1),('b',3),('c',2)]
***Main>** ordenarTuplas [('b',3),('c',2),('a',1)] 1
[('a',1),('c',2),('b',3)]

17) Implemente a função *filtrarEInserir :: [[Int]] -> Int -> ([[Int]], Int)* que retorna uma tupla. O primeiro elemento da tupla são listas de inteiros tais que a soma dos números ímpares é maior que a soma dos números pares. O segundo elemento consiste no produto entre o segundo argumento da função filtrarInserir e a multiplicação da maior soma obtida das listas retornadas. Utilize obrigatoriamente *filter*.

***Main>** filtrarEInserir [[2,3,4,5,6], [1, 2, 3], [9]] 5
([[1,2,3], [9]], 45)
***Main>** filtrarEInserir [[2,3,4,5], []] 7
([[2,3,4,5]],56)
***Main>** filtrarEInserir [] 5
([],0)

18) Construa a função separaESoma que, dada uma lista de inteiros finita, retorne uma tupla que contenha a soma dos números pares como primeiro elemento e a soma dos números ímpares como segundo elemento da tupla. Deve-se usar a função map e alguma função que faça dobramento (foldr, foldr1, foldl...)

***Main>** separaESoma [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
(110,100)

19) Definidos os tipos Vector e Matrix como segue, crie a função multiplicaMatrizes que executa apenas a multiplicação entre duas matrizes (Matrix) quadradas. O retorno da função deve ser uma Matrix que contem o resultado da multiplicação entre os parâmetros. Eficiência não é uma preocupação para resolver essa questão.

type Vector = [Double]
type Matrix = [Vector]

***Main>** multiplicaMatrizes [[1, 2], [3, 4]] [[4, 3], [2, 1]]
[[8.0,5.0],[20.0,13.0]]