1ª Lista de Exercícios de Paradigmas de Linguagens Computacionais

Professores: Fernando Castor, Paulo Borba

Monitores: Marcos Paulo Barros Barreto (mpbb)
Tulio Paulo Lages da Silva (tpls)

CIn –UFPE - 2014.2
Disponível em: 15/10/2014
Entrega: 29/10/2014

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 3** das respostas estão corretas não foi entregue. A entrega da lista com **pelo menos 7** 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.

1ª) Um fazendeiro cultiva um tipo de planta onde, após a colheita, cada folha é vendida por alguns centavos. Infelizmente, sua plantação sofre ataque de algumas espécies de insetos. Cada espécie de inseto ataca em um horário distinto do dia e só atacam plantas com uma quantidade de folhas dentro de um certo range (não conseguem atacar plantas muito vigorosas e não se interessam pelas que estão debilitadas).

Faça uma função que receba os seguintes parâmetros: o valor de venda de cada folha; o valor de pulverização de uma planta (protegendo-a contra todos os insetos); uma lista de plantas, cada uma com quantidade de folhas inicial e quantidade de folhas que nascem por dia; uma lista de insetos, com número médio de folhas consumidas por planta/dia e range de folhas para eles atacarem uma planta (ou seja, um inseto (3,(10,15)) retiraria 3 folhas de cada planta com 10 a 15 folhas), essa lista de insetos já está na ordem de ataque; número de dias até a colheita. Como resultado a função deve devolver uma lista de plantas que devem ser pulverizadas para maximizar o lucro do fazendeiro.

2ª) Faça uma função **func1 :: a->String->(String->Bool)->(a,String)**, que simplesmente devolve o par formado pelo seu primeiro argumento e o segundo caso o segundo argumento aplicado ao terceiro seja verdadeiro e devolva (a,"") caso contrário.
Então faça uma **func2 :: [String->e]->[String]->[String->Bool]->[e]**, que aplique cada elemento da lista em seu primeiro argumento a func1. Então monte uma lista com todas as possibilidades de aplicar seu segundo e terceiro argumento à lista de funções parciais encontrada no primeiro passo. Com essa lista em mãos, use obrigatoriamente a função map sobre ela e obtenha [e].

3ª) Faça uma função chamada **bucketSort**, que recebe n como número de buckets que deve criar, uma lista de inteiros e uma lista de funções de ordenação. Cada bucket deve ser construído como uma lista de inteiros presente na lista original, ficando dentro do seguinte intervalo: [(ma/n * (b-1) , (ma/n *b)], onde ma é o maior inteiro presente na lista original, n é o número de bucktes e b é o "índice" do bucket (considerando índices de 1 a n)

Uma vez que os buckets e seus valores estiverem definidos, cada bucket será passado a uma função de ordenação (se houver menos funções do que buckets, todos os buckets que sobrarem

devem ser jogados novamente em bucketSort). Por fim concatena as listas ordenadas.

4ª) Faça uma função **gerarListaCombinacoes :: Int -> Int -> [[Int]]** que gere todas as possíveis combinações de **K (segundo parâmetro) números distintos** escolhidos de uma lista contendo de **0 até N-1 (primeiro parâmetro) números**.
Ex:
Prelude>gerarListaCombinacoes 3 2
[[0, 1],[0, 2],[1, 2]].

5ª) O método de criptografia Diffie-Hellman foi um dos primeiros exemplos práticos de métodos de troca de chaves implementado dentro do campo da criptografia. Ele permite que duas partes que não se conhecem compartilharem uma chave secreta em um canal de comunicação inseguro. Sabendo disso, crie a função
gerarChaveDiffieHellman :: Integer -> Integer -> Integer -> Integer -> Integer,
na qual os parâmetros são, respectivamente:

- Um número primo;
- Uma raiz primitiva módulo o número primo escolhido;
- Dois números inteiros.

Ex:
Prelude>gerarChaveDiffieHellman 23 5 6 15
2

6ª) A festa de formatura dos graduandos do CIn do período letivo de 20XY.Z está para acontecer, e todos combinaram de que haverá um baile. O que ninguém confirmou ainda foi como serão formados os casais, o que deixa Robelito um pouco apreensivo. Sabendo que a quantidade de mulheres em relação a quantidade de homens em sua sala é bem baixa e, apesar de seu charme, Robelito sabe que dificilmente conseguirá um par feminino a tempo do baile. Por isso ele deve decidir rápido se ele continuará em sua busca por um par feminino, buscando apenas mulheres, se passará a não se importar, dançando com qualquer um que aparecer, ou se desistirá logo e irá pro baile com sua fantasia das Virgens de Olinda, para que pelo menos na foto com seu par provavelmente masculino tenha alguma presença feminina.
Com isso, crie uma função **filtroGerarPares** para ajudar Robelito a decidir como proceder. A função deve receber os seguintes parâmetros:

- Uma lista de tuplas, da forma [(String, Char, Bool)], respectivamente o nome do aluno, um caractere indicando se o aluno é mulher ou homem (‘m’ e ‘h’, respectivamente) e um booleano indicando se esta pessoa toparia ou não dançar com Robelito;
- Um Char indicando a decisão de Robelito (‘m’ para só querer dançar com mulheres, ‘x’ para tanto faz e ‘v’ para desistir e dançar com algum amigo, vestindo sua fantasia das Virgens.

Ela deve retornar a lista com os nomes dos candidatos que toparem dançar com Robelito e estiverem dentro do seu conjunto de decisão.
Ex:
Prelude>filtroGerarPares [(“Glaucio”, ‘h’, True), (“Zeno”, ‘h’, False), (“Josiela”, ‘m’, True), (“Juliana” ‘m’, False), (“Priscillo”, ‘h, True), (“Robson”,’m’,False)] ‘m’
[(“Josiela”)]

7ª) Crie uma função **checaOrdenacao :: [String] -> (String -> String-> Boolean) -> ([String], Boolean)**, que recebe uma lista de Strings e uma função e retorne uma tupla com a lista de Strings original ordenada de acordo com a função fornecida e um booleano indicando se a lista já estava ordenada de acordo com a função (True) ou se foi preciso reordená-la (False).

8ª) Crie a função **buscaPalavra :: String -> [String] -> [String]**, que retorne todas as palavras (apenas letras, sem espaços ou caracteres especiais) da lista que comecem com a palavra passada como parâmetro (ignorando se é maiúscula ou minúscula).
Ex:
Prelude>buscaPalavra “pont” [“ponta”, “apontar”, “Ponteiro”, “porto”]
[“ponta”, “Ponteiro”].