Trabalho de PLP 2024/1

Esta atividade avaliativa deve ser resolvida por grupos de até dois alunos. Várias funções devem implementadas em Haskell e entregues num único arquivo para avaliação no Campus Virtual.

As implementações entregues deverão ser fruto de trabalho próprio: os grupos não devem ver as implementações uns dos outros, não devem usar implementações de outros (sejam disponíveis na internet ou produzidas por IA generativa). Será realizada verificação de plágio nas respostas. Será verificado se os recursos utilizados condizem com os recursos apresentados em aula. Cada grupo implementará apenas algumas das funções a seguir. Para isso, será divulgado, via Campus Virtual, uma tabela ou regra que identifica quais das funções deverão ser entregues.

O arquivo entregue deve ter um cabeçalho com o nome completo e número de matrícula de cada membro do grupo, além do número do grupo (definido na determinação de quais funções entregar). As funções devem ter comentários indicando seu número no enunciado.

Ao criar funções auxiliares, verifique se elas devem estar expostas para serem usadas em qualquer programa ou se deve estar escondidas para serem usadas apenas por outra função. Implemente de acordo.

Algumas das funções indicadas estão disponíveis na biblioteca padrão da linguagem. Entretanto, nesses casos, você obviamente não pode só chamar a função da biblioteca.

As funções a implementar são:

1. unica-ocorrencia: recebe um elemento e uma lista e verifica se existe uma única ocorrência do elemento na lista
   ex.: unica_ocorrencia 2 [1,2,3,2] ⇒ False
   unica_ocorrencia 2 [3,1] ⇒ False
   unica_ocorrencia 2 [2] ⇒ True
2. maiores-que: recebe um número e uma lista de números, retorna uma lista com os números que são maiores que o fornecido
   ex.: maiores_que 10 [4,6,30,3,15,3,10,7] ⇒ (30 15)
3. concatena: recebe duas listas quaisquer e retorna um terceira lista com os elementos da primeira no início e os elementos da segunda no fim
   ex.: concatena [] [] ⇒ []
   concatena [1,2] [3,4] ⇒ [1,2,3,4]
4. remove: recebe um elemento e uma lista e retorna a lista sem a primeira ocorrência do elemento
5. remover_ultimo: recebe uma lista e retorna a lista sem o último elemento
6. remover_repetidos: recebe uma lista e retorna outra lista sem repetição de elementos
   ex.: remover_repetidos [7,4,3,5,7,4,4,6,4,1,2] ⇒ [7,4,3,5,6,1,2]
7. maiores: recebe um numero natural *n* e uma lista de números, retorna uma lista com os *n* maiores números sem alterar a ordem entre os elementos
   ex.: maiores 4 [9,3,5,7,8,4,4,7] ⇒ [9,7,8,7]
8. gera_sequencia: recebe um número inteiro *n* positivo e retorna a lista [1,-1,2,-2,3,-3, ... ,*n*,-*n*]

9. inverte: recebe uma lista e retorna outra, que contém os mesmos elementos da primeira, em ordem invertida
10. divide: recebe uma lista e um número natural *n*, retorna um par onde o primeiro elemento é uma lista com os *n* primeiros números da lista original e o segundo elemento é uma lista com o resto dos elementos da lista original
    ex.: divide [1,2,3,4] 0 ⇒ ([],[1,2,3,4])
    divide [1,2,3,4] 2 ⇒ ([1,2],[3,4])
11. somatorio: recebe uma lista de números e retorna a soma deles
    ex.: somatorio [] ⇒ 0
    somatorio [2,3] ⇒ 5
12. intercala: recebe duas listas e retorna outra lista com os elementos das listas originais intercalados.
    ex.: intercala [1,2,3] [4,5] ⇒ [1,4,2,5,3]
    intercala [] [1,2,3] ⇒ [1,2,3]
13. uniao: recebe duas listas que não contenham elementos repetidos e retorna uma nova com todos os elementos das duas listas originais (sem repetição)
    ex.: uniao [3,6,5,7] [2,9,7,5,1] ⇒ [3,6,5,7,2,9,1]
14. `interseccao`: recebe duas listas sem elementos repetidos e retorna uma lista com os elementos que são comuns às duas
    ex.:
    `interseccao [3,6,5,7] [9,7,5,1,3] ⇒ [3,5,7]`
15. mesmos_elementos: recebe duas listas e verifica se elas tem os mesmos elementos
16. sequencia: recebe dois números naturais *n* e *m*, e retorna uma lista com *n* elementos, onde o primeiro é *m*, o segundo é *m*+1, etc...
    ex.: sequencia 0 2 ⇒ []
    sequencia 3 4 ⇒ [4,5,6]
17. insere_ordenado: recebe uma lista de números em ordem crescente e um número qualquer, retorna uma lista de números em ordem crescente com os elementos da lista inicial mais o número passado.
18. ordenado: recebe uma lista de números e verifica se eles estão ordenados (ordem crescente) ou não.
    ex.: ordenada [3,7,7,8,9] ⇒ True
19. ordena: recebe uma lista com números e retorna outra lista com os números ordenados
    ex.: ordena [7,3,5,7,8,4,4] ⇒ [3,4,4,5,7,7,8]
20. mediana: recebe uma lista de números e retorna a mediana deles.
21. picos: recebe uma lista de números e retorna os números que são maiores que seus vizinhos. Considere que a lista é circular, ou seja, o início e o fim estão ligados.
    ex.: picos [2,3,5,10,5,5,6,2,3] ⇒ [10,6,3]
22. rodar_esquerda: recebe um número natural, uma lista e retorna uma nova lista onde a posição dos elementos mudou como se eles tivessem sido "rodados"
    ex.: rodar_esquerda 0 [1,2,3,4,5] ⇒ [1,2,3,4,5]
    rodar_esquerda 1 [1,2,3,4,5] ⇒ [2,3,4,5,1]
    rodar_esquerda 3 [1,2,3,4,5] ⇒ [4,5,1,2,3]
    rodar_esquerda 9 [1,2,3,4,5] ⇒ [5,1,2,3,4]

23. rodar_direita: recebe um número natural, uma lista e retorna uma nova lista onde a posição dos elementos mudou como se eles tivessem sido "rodados"
    ex.: rodar_direita 0 [1,2,3,4,5] ⇒ [1,2,3,4,5]
    rodar_direita 1 [1,2,3,4,5] ⇒ [5,1,2,3,4]
    rodar_direita 3 [1,2,3,4,5] ⇒ [3,4,5,1,2]
    rodar_direita 4 [1,2,3,4,5] ⇒ [2,3,4,5,1]
24. todas_maiusculas: Recebe uma string qualquer e retorna outra string onde todas as letras são maiúsculas. Pode ser útil saber os seguintes códigos de representação de caracteres: a=97, z=122, A=65, Z=90, 0=48, 9=57, espaço=32.
    ex.: todas_maiusculas "abc 123" = "ABC 123"
25. primeiras_maiusculas: recebe uma string qualquer e retorna outra string onde somente as iniciais são maiúsculas
    ex.: primeiras_maiusculas "FuLaNo bElTrAnO silva" ⇒"Fulano Beltrano Silva"
26. media: Recebe uma lista de números e retorna a média aritmética deles.
27. variancia: recebe uma lista de números racionais e retorna a variância (populacional) deles.
28. mediana: calcula a mediana de uma lista de números racionais.
29. seleciona: recebe uma lista qualquer e uma lista de posições, retorna uma lista com os elementos da primeira que estavam nas posições indicadas
    ex.: seleciona [’a’,’b’,’c’,’d’,’e’,’f’] [1,4,3,4] ⇒ [’a’,’d’,’c’,’d’]
30. separa: separa os elementos de uma lista de números nas posições com zero.
    ex.: separa [3,4,7,-1,0,4,7,3,0,0,9,8] ⇒ [[3,4,7,-1],[4,7,3],[],[9,8]]
31. palindromo?: recebe uma string e verifica se ela é uma palíndromo ou nao
    ex.: palindromo? "ana" ⇒True
    palindromo? "abbccbba" ⇒True
    palindromo? "abbdbbaa" ⇒False
32. primo: verifica se um número é primo ou não
33. soma_digitos: recebe um número natural e retorna a soma de seus dígitos
    ex.: soma_digitos 328464584658 ⇒ 63
34. bolha: recebe uma lista de números e retorna a lista ordenada, pelo método da bolha (bolha burra – aquela versão em que não se verifica se houve alguma troca para parar mais cedo)
35. compactar: recebe uma lista de números e transforma todas as repetições em sub-listas de dois elementos: sendo o primeiro elemento o número de repetições encontradas e o segundo elemento é o número que repete na lista original. Os números que não repetem na lista original não devem ser alterados.
    ex.: compactar [2,2,2,3,4,4,2,9,5,2,4,5,5,5] ⇒ [3,2],[3],[2,4],[2],[9],[5],[2],[4],[3,5]]
36. Dizemos que um *quadrado perfeito* é um número cuja raiz quadrada é um número inteiro. Sabemos o que a raiz quadrada é um cálculo lento quando comparado à operações como adição ou multiplicação. Implemente uma função que verifica se um número é um quadrado perfeito sem usar uma função que calcula raiz quadrada.
37. Faça um programa que encontra a representação de um número natural numa base *b* qualquer (1 < b < 37).
    ex: muda_base 17 2 ⇒ "10001"
    muda_base 26 16 ⇒ "1A"

38. O conjunto de todos os subconjuntos de um segundo conjunto é denominado *conjuntos das partes* desse segundo conjunto. Faça um programa que encontra o conjunto das partes de uma lista. Exemplo:
partes [2,3,2,31] ⇒ [[],[2],[3],[31],[2,2],[2,3],[2,31],[3,31],[2,2,3],[2,2,31],[2,3,31], [2,2,3,31]]