

Início	sábado, 4 de dezembro de 2021 às 14:10
Estado	Prova submetida
Data de submissão:	sábado, 4 de dezembro de 2021 às 15:25
Tempo gasto	1 hora 15 minutos
Nota	14,00 de um máximo de 14,00 (100%)

Informação

- Funções solicitadas em alíneas anteriores podem ser utilizadas em alíneas seguintes, mesmo que não as tenha implementado.
- Se implementar funções auxiliares, deve incluí-las na resposta de todas as alíneas onde sejam usadas.
- Não precisa copiar qualquer código fornecido no enunciado para a caixa de texto de resposta às questões.
- Documente o código e use nomes de variáveis intuitivos, de forma a clarificar a interpretação do programa.
- Não pode usar funções de bibliotecas / importar módulos em nenhuma alínea.

Pergunta 1

Respondida

Pontuou 1,500 de 1,500

Escreva uma definição da função **maxpos :: [Int] -> Int** que calcula o máximo dos inteiros positivos numa lista; se a lista não tiver números positivos o resultado deve ser 0.

Exemplos:

```
maxpos [1,2,3,4,5] == 5
maxpos [-1,-2,-3,4,-5] == 4
maxpos [2] == 2
maxpos [] == 0
```

```
maxpos :: [Int] -> Int
maxpos [] = 0
maxpos (x:xs)
  | x > next = x
  | otherwise = next
where
  next = maxpos xs
```

Pergunta 2

Respondida

Pontuou 1,500 de 1,500

Escreva uma definição da função **dups :: [a] -> [a]** que duplica os valores em posições alternadas numa lista (isto é, duplica o primeiro elemento, não duplica o segundo, duplica o terceiro, e assim sucessivamente).

Exemplos:

```
dups "abcdef" == "aabccdeef"
dups [0,1,2,3,4] == [0,0,1,2,2,3,4,4]
dups [] == []
```

```
dups :: [a] -> [a]
dups xs = dupsAux xs True

dupsAux :: [a] -> Bool -> [a]
dupsAux [] _ = []
dupsAux (x:xs) r
  | r = x : x : dupsAux xs False
  | otherwise = x : dupsAux xs True
```

Pergunta 3

Respondida

Pontuou 1,500 de 1,500

A linguagem dos Ps é um jogo de palavras em que duplicamos cada vogal (letras 'a', 'e', 'i', 'o', 'u') e colocamos um 'p' entre elas; todos os outros caracteres e letras ficam inalterados. Assim, por exemplo, a frase "ola, mundo!" fica "opolapa, mupundopo!". Escreva uma definição da função **transforma :: String -> String** que transforma uma frase para a linguagem dos Ps. Para simplificar, assuma que a frase não contém letras maiúsculas nem acentuadas.

Exemplos:

```
transforma "ola, mundo!"      = "opolapa, mupundopo!"
transforma "4 gatos e 3 ratos" = "4 gapatopos epe 3 rapatopos"
```

```
transforma :: String -> String
transforma [] = []
transforma (x:xs)
  | x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u' = x : 'p' : x : transforma xs
  | otherwise = x : transforma xs
```

Informação

Considere os seguintes tipos nas questões abaixo:

```
type Vector = [Int]
type Matriz = [[Int]]
```

Nota: as matrizes são retangulares, ou seja, o comprimento de todas as sublistas é idêntico.

Pergunta 4

Respondida

Pontuou 2,000 de 2,000

Defina a função **transposta :: Matriz -> Matriz**, que calcula a matriz transposta.

Exemplo:

```
transposta [[1,2], [3,4]] = [[1,3], [2,4]]
transposta [[1,2,3], [4,5,6]] = [[1,4], [2,5], [3,6]]
```

```
transposta :: Matriz -> Matriz
transposta [] = []
transposta m = [head x | x <- m] : transposta [tail x | x <- m, tail x /= []]
```

Pergunta 5

Respondida

Pontuou 1,000 de 1,000

Defina a função **prodInterno :: Vector -> Vector -> Int**, que calcula o produto interno de dois vectores (a soma dos produtos de elementos com o mesmo índice). Assume-se que ambos os vetores têm o mesmo comprimento.

Exemplo:

```
prodInterno [1,2,3] [4,3,2] = 16
```

```
prodInterno :: Vector -> Vector -> Int
prodInterno [] [] = 0
prodInterno x1 x2 = head x1 * head x2 + prodInterno (tail x1) (tail x2)
```

Pergunta 6

Respondida

Pontuou 2,000 de 2,000

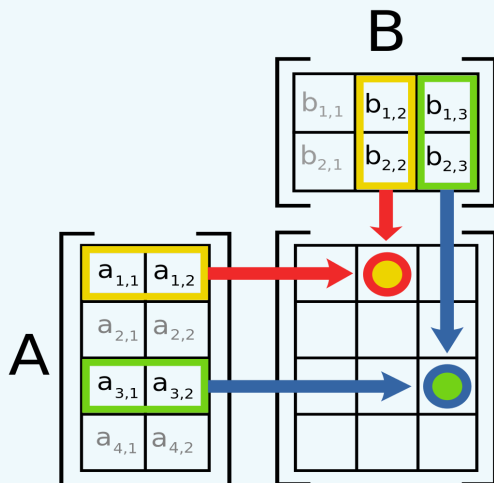
Defina a função **prodMat :: Matriz -> Matriz -> Matriz**, que calcula o produto de duas matrizes.

Exemplo:

```
prodMat [[1,2], [3,4]] [[2,3], [2,4]] = [[6,11], [14,25]]
```

(note que $6 = 1*2 + 2*2$ e $11 = 1*3 + 2*4$)

Diagrama ilustrativo de multiplicação de matrizes:



```
prodMat :: Matriz -> Matriz -> Matriz
prodMat m1 m2 = prodMatAux m1 (transposta m2)
```

```
prodMatAux :: Matriz -> Matriz -> Matriz
prodMatAux [] [] = []
prodMatAux m1 m2 = [[prodInterno v1 v2 | v2 <- m2] | v1 <- m1]
```

Informação

Considere a seguinte definição de um tipo de dados para árvores binárias em que N é o construtor de nós e F é o construtor de folhas.

```
data Arv a = F | N a (Arv a) (Arv a)
    deriving(Show)
```

Pergunta 7

Respondida

Pontuou 1,500 de 1,500

Escreva uma definição da função **alturas :: Arv a -> Arv Int** que transforma uma árvore binária noutra com a mesma estrutura mas em que o valor de cada nó é dado pela sua altura (isto é, o maior comprimento dum caminho desse nó até uma folha).

Exemplo (com uma árvore de strings):

```
alturas (N "joão" (N "abel" F F) (N "pedro" F F)) = N 2 (N 1 F F) (N 1 F F)
```

```
alturas :: Arv a -> Arv Int
alturas F = F
alturas (N a a1 a2) = N (alturasAux (N a a1 a2)) (alturas a1) (alturas a2)
```

```
alturasAux :: Arv a -> Int
alturasAux F = 0
alturasAux (N a a1 a2) = 1 + max (alturasAux a1) (alturasAux a2)
```

Pergunta 8

Respondida

Pontuou 2,000 de 2,000

Defina uma outra função ***equilibrada :: Arv a -> Bool*** que exprima a condição duma árvore ser equilibrada, isto é, as alturas das sub-árvores de cada nó diferem no máximo de 1 unidade.

Exemplo (com a mesma árvore):

```
equilibrada (N "joão" (N "abel" F F) (N "pedro" F F)) = True
```

-- por indicação do professor (auxiliar utilizada nas alturas)

alturasAux :: Arv a -> Int

alturasAux F = 0

alturasAux (N a a1 a2) = 1 + max (alturasAux a1) (alturasAux a2)

equilibrada :: Arv a -> Bool

equilibrada F = True

equilibrada (N a a1 a2) = equilibradaAux (alturas (N a a1 a2))

equilibradaAux :: Arv Int -> Bool

equilibradaAux F = True

equilibradaAux (N a F F) = True

equilibradaAux (N a F (N b b1 b2)) = b == 1

equilibradaAux (N a (N b b1 b2) F) = b == 1

equilibradaAux (N a (N b b1 b2) (N c c1 c2)) = b == c && equilibradaAux (N b b1 b2) && equilibradaAux (N c c1 c2)

Pergunta 9

Respondida

Pontuou 1,000 de 1,000

Escreva a definição de uma função *f*, em Haskell, que tenha o tipo:

f :: (a -> b -> c) -> b -> a -> c

(nota: não complique - a função pretendida não é complicada!)

f :: (a -> b -> c) -> b -> a -> c

f fun b a = fun a b