Disciplina: Paradigmas de Programação
Professor: Maicon Rafael Zatelli
Entrega: *Moodle*

Trabalho I - Programação Funcional - Haskell

Atenção: Faça um ZIP com todos os arquivos de solução. Use o nome do arquivo de maneira a entender qual problema você está resolvendo. Por exemplo, problema1.hs, problema2.hs e assim por diante. Responda todas as questões referentes ao problema 1, no arquivo do problema 1. Faça o mesmo para as questões referentes ao problema 2, e assim por diante.

- A legibilidade, organização do código, bem como o uso de comentários também serão considerados na avaliação.
- Note que a somatória dos pontos passa de 10, porém mesmo acertanto todos os problemas, a nota máxima será 10.

1. **(1.0)** Crie uma função com assinatura `mesmosElementos :: [Int] -> [Int] -> Bool`, a qual retorne se duas listas possuem os mesmos elementos, independentemente da ordem em que eles aparecem e do número de vezes que eles possam se repetir. **Não utilize** nenhuma função pronta to Haskell para esta tarefa.
2. Motifique o arquivo `arvore.hs` (disponível no Moodle) de forma a adicionar novas operações a nossa árvore:
   - **A (0.5):** Crie uma função com a seguinte assinatura: `ocorrenciasElemento :: Arvore -> Int -> Int`, a qual recebe um número e deve retornar a quantidade de ocorrências dele na árvore.
   - **B (0.5):** Crie uma função com a seguinte assinatura: `maioresQueElemento :: Arvore -> Int -> Int`, a qual recebe um número e deve retornar a quantidade de números maiores que ele na árvore.
   - **C (0.5):** Crie uma função com a seguinte assinatura: `subtraiParesImpares :: Arvore -> Int`, a qual deve retornar a soma de todos os números pares - (menos) a soma de todos os números ímpares.
   - **D (1.0):** Crie uma função com a seguinte assinatura: `igual :: Arvore -> Arvore -> Bool`, a qual recebe duas árvores como parâmetro e deve retornar se elas são iguais. Duas árvores são iguais se elas possuem os mesmos elementos, dispostos da mesma forma.
   - **E (1.0):** Crie uma função com a seguinte assinatura: `altura :: Arvore -> Int`, a qual recebe uma árvore como parâmetro e deve retornar a sua altura. Uma árvore com apenas o nó raiz possui altura 0.
   - **F (1.0):** Crie uma função com a seguinte assinatura: `folhas :: Arvore -> Int`, a qual recebe uma árvore como parâmetro e deve retornar a quantidade de folhas na árvore. Um nó é uma folha se ele não possui filhos.
   - **G (1.0):** Crie uma função com a seguinte assinatura: `emOrdem :: Arvore -> [Int]`, a qual recebe uma árvore como parâmetro e deve retornar a lista de números visitados em percorrimento em ordem.
   - **H (1.0):** Crie uma função com a seguinte assinatura: `menoresQueElemento :: Arvore -> Int -> [Int]`, a qual recebe um número e deve retornar os números pares menores que ele na árvore. ATENÇÃO: retornar os números, não a quantidade.
3. Crie um tipo (**data** ou **type**) para um ponto 2D e resolva os seguintes problemas:
   - **A (1.0):** Crie uma função com a seguinte assinatura: `colineares :: Ponto -> Ponto -> Ponto -> Bool`, a qual recebe três pontos e retorna se os mesmos são colineares. DICA: os três pontos serão colineares se o determinante de suas coordenadas for igual a 0. Use esta dica para resolver o problema.
   - **B (2.0):** Crie uma função com a seguinte assinatura: `menorDistancia :: [Ponto] -> Float`, a qual recebe uma lista de pontos e retorna a distância entre os dois pontos mais próximos. Explique seu raciocínio em um comentário para este problema.