Primeiro EE de PLC (IF686)

Data: 08 de julho de 2013. **Tempo disponível: 2h00m.** **Prof.:** Fernando Castor

Parzival está em uma busca desesperada pelo *easter egg* deixado por Anorak. A coleta de pistas envolve o estudo aprofundado de uma enorme quantidade de material (jogos, filmes, seriados, comerciais de TV, músicas, etc., todos dos anos 80 e, em menor escala, 90) que lhe forneça pistas para descobrir onde as três chaves (cobre, jade e cristal) e os três portões se encontram no virtualmente infinito OASIS. Para tornar o registro de informações mais eficiente e passar na frente dos outros *gunters*, Parzival resolveu construir um sistema de software em Haskell, já que ele ouviu falar que essa linguagem, criada no final da década de 80, era uma paixão secreta (e não documentada) de James Halliday (o nome real do supracitado Anorak). Esse sistema manterá informações cronológicas (em termos do momento em que ele as registrou) e permitindo que consultas sejam realizadas com flexibilidade. O problema é que Parzival não entende muito sobre Haskell e gostaria de uma ajuda para escrever pelo menos a parte principal do sistema. Este é o ponto em você pode ajudá-lo. READY PLAYER ONE!

1. (1.0 pto.) Antes de qualquer coisa, Parzival precisa definir quais serão os dados que ele precisa armazenar. Mostre para ele como escrever isso em Haskell definindo um tipo de dados chamado `Dica` capaz de representar, ao mesmo tempo, filmes, músicas e jogos, associando até quatro (no mínimo dois!) pedaços de informação relevantes a cada uma dessas categorias de informação. Por exemplo, filmes podem ter um nome, um diretor e um ano de lançamento e jogos podem ter um nome, um ano de lançamento e o nome do *publisher*. Complementarmente, Parzival às vezes cruza informações com características diferentes (um filme e um jogo, dez músicas da trilha sonora de um filme e este último, etc.). Logo, uma `Dica` também pode ser um `Relacionamento` entre `Dica`s. Seu tipo de dados deve ser definido de modo que valores desse tipo possam ser transformados em `String`s e comparados para saber se são iguais com facilidade.

```
data Dica = Relaciomamento [Dica] | Filme (String, String, Int) | Jogo (String, Int, String) | Musica (String, String, Int) deriving (Eq, Show)
-- Filme: nome, diretor, ano de lançamento. Jogo: nome, ano de lançamento, publisher. Musica: nome, autor, ano de lançamento
```

2. (2.0 pto.) Para manter o registro histórico dos dados, Parzival pretende empregar uma fila. Mostre a ele como se implementa uma fila simples em Haskell e escreva as funções para `inserir` e `remover` elementos em uma fila. Cada elemento incluído deve ter um identificador único associado. Esse identificador é fornecido no momento da inclusão. Crie também uma função para saber se um elemento está `presente` em uma fila. Sua fila deve ser tão genérica quanto possível (funcionar também com valores de tipos diferentes de `Dica`s) e não mais.

```
data Filas t u = Fila [(t, u)] deriving (Eq, Show) -- t = identificador e u = elemento (Dica, por exemplo)

filaExemplo = Fila ([(1, Filme ("Avengers", "Diretor", 2015)),(2, Musica ("Clocks", "Coldplay", 2010)),(3, Relaciomamento [(Musica ("Titanic", "Diretor", 2000)), (Filme ("My Heart Will Go On", "Celine Dion", 2010))])])

inserir :: Filas t u -> t -> u -> Filas t u
inserir (Fila dicas) identificador dica = Fila (dicas ++ [(identificador,dica)])

-- inserir filaExemplo 4 (Musica ("Paradise", "ColdPlay", 2012))
-- Fila [(1,Filme ("Avengers","Diretor",2015)),(2,Musica ("Clocks","Coldplay",2010)),(3,Relaciomamento [Musica ("Titanic","Diretor",2000),Filme ("My Heart Will Go On","Celine Dion",2010)]),(4,Musica ("Paradise","ColdPlay",2012))]

remover :: Eq u => Filas t u -> u -> Filas t u
remover (Fila dicas) d = Fila ([(id, dica) | (id, dica) <- dicas, (dica /= d)])

-- remover filaExemplo (Filme ("Avengers", "Diretor", 2015))
-- Fila [(2,Musica ("Clocks","Coldplay",2010)),(3,Relaciomamento [Musica ("Titanic","Diretor",2000),Filme ("My Heart Will Go On","Celine Dion",2010)])]
```

```
presente :: (Eq t, Eq u) => Filas t u -> u -> Bool
presente (Fila dicas) d = ([(id, dica) | (id, dica) <- dicas, (dica == d)] /= [])

-- presente filaExemplo (Musica ("Clocks", "Coldplay", 2010)) -- True
-- presente filaExemplo (Musica ("The Scientist", "Coldplay", 2010)) -- False
```

3. (3.5 ptos.) O histórico é terreno fértil para Parzival realizar consultas tentando correlacionar informações, algo fundamental na vida de qualquer *gunter*. Informações estão correlacionadas quando têm um alto nível de similaridade. Entretanto, os significados de "alto nível" e "similaridade" dependem da consulta a ser realizada. Por isso, Parzival precisa de flexibilidade ao realizar consultas. Você deve ajudá-lo a ver o quanto Haskell pode tornar simplificar sua vida. Implemente uma função chamada `consultar` que devolve uma `Dica` que é um `Relacionamento` envolvendo todas as `Dica`s do histórico cujo nível de similaridade com alguma outra `Dica` seja maior que um certo limiar. Tenha em mente que seu programa deve ser flexível, tanto em termos do critério de similaridade quanto em termos do limiar.