Princípios de Programação

Trabalho 2

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2021/2022

Qualquer eleição é um momento importante numa sociedade democrática, pois dá a todos os elementos dessa sociedade uma voz para, em conjunto, poderem chegar a um consenso. Devido à sua importância, é necessário que um procedimento eleitoral ocorra da forma mais transparente e eficiente possível. Neste segundo trabalho vamos implementar um **sistema de votação posicional** para decidir o vencedor de uma eleição.
Num sistema de votação posicional, cada eleitor pode votar em múltiplos candidatos. O sistema que vamos usar é uma simplificação do que é utilizado no Festival Eurovisão da Canção🎤. Para os efeitos deste trabalho, a votação decorre da seguinte forma.

1. Cada eleitor deve escolher os seus três candidatos preferidos, indicando-os por ordem.
2. Cada candidato recebe, de cada eleitor: 3 pontos por ser o preferido desse eleitor; 2 pontos por ser o segundo preferido; e 1 ponto por ser o terceiro preferido.
3. Vence o candidato que tiver obtido mais pontos.

Nome	Primeira preferência	Segunda preferência	Terceira preferência
Bélgica	Portugal	Espanha	França
Dinamarca	Islândia	França	Bélgica
Espanha	Portugal	França	Grécia
França	Islândia	Portugal	Espanha
Grécia	Portugal	Islândia	França
Islândia	Espanha	Dinamarca	Portugal
Portugal	Islândia	França	Espanha

A tabela acima exemplifica uma eleição entre sete países europeus. Lendo a primeira linha, observamos que a Bélgica decidiu dar 3 pontos a Portugal, 2 pontos à Espanha e 1 ponto à França. Contando os votos, observaríamos que Portugal ficou três vezes em primeiro lugar, uma vez em segundo lugar e uma vez em terceiro lugar, obtendo assim um total de $3 \times 3 + 1 \times 2 + 1 \times 1 = 12$ pontos.
Contabilizando os pontos, obteríamos os seguintes resultados.

Nome	Vezes em 1º	Vezes em 2º	Vezes em 3º	Pontos
Bélgica	0	0	1	1
Dinamarca	0	1	0	2
Espanha	1	1	2	7
França	0	3	2	8
Grécia	0	0	1	1
Islândia	3	1	0	11
Portugal	**3**	**1**	**1**	**12**

Assim, neste exemplo, Portugal seria o país vencedor.
Vamos representar os votos de um eleitor usando o tipo de dados

```
(String,(String,String,String))
```

O primeiro elemento do par é uma string com o nome do eleitor; o segundo elemento do par é um triplo com os nomes dos candidatos colocados em primeira, segunda, e terceira preferência respetivamente. Assim, a tabela descrita no exemplo acima pode ser representada por

```
votos :: [(String,(String,String,String))]
votos = [
  ("Bélgica", ("Portugal", "Espanha", "França")),
  ("Dinamarca", ("Islândia", "França", "Bélgica")),
  ("Espanha", ("Portugal", "França", "Grécia")),
  ("França", ("Islândia", "Portugal", "Espanha")),
  ("Grécia", ("Portugal", "Islândia", "França")),
  ("Islândia", ("Espanha", "Dinamarca", "Portugal")),
  ("Portugal", ("Islândia", "França", "Espanha"))]
```

O seu objetivo é implementar as seguintes funções:

```
votosCandidato :: [(String,(String,String,String))] ->
   String -> Int
```

que, recebendo os votos dos vários eleitores e o nome de um candidato, devolve o número de pontos recebidos por esse candidato;

```
vencedorEleicao :: [(String,(String,String,String))] ->
   String
```

que, recebendo os votos dos vários eleitores, devolve o candidato vencedor.
Por exemplo:

```
> votosCandidato votos "Portugal"
12
> votosCandidato votos "Grécia"
1
> vencedorEleicao votos
"Portugal"
```

Tome em atenção os seguintes pontos:

1. O input é dado por uma lista de pares; cada par é composto por uma string que indica o nome do eleitor e um triplo de strings que indicam as preferências desse eleitor.
2. Em caso de empate, o vencedor deverá ser o candidato que votou em primeiro lugar, isto é, aquele cujo nome aparece primeiro como primeira componente de um par da lista.
3. Para obter a cotação máxima, deverá utilizar *pelo menos uma* função de ordem superior (**`map`**, **`filter`**, **`foldl`**, **`foldr`**, etc.), e deverá implementar *pelo menos uma* função recursivamente.
4. Para simplificar a sua implementação, pode assumir que: a lista de votos é uma lista não-vazia; que nesta lista, para cada candidato há um e um só tuplo correspondente aos seus votos; que nenhum candidato vota em si próprio; e que nenhum candidato vota noutro candidato duas vezes.

Notas

1. Deverá submeter um ficheiro com o nome `t2_fcXXXXX.hs`, onde `XXXXX` é o seu número de aluno.
2. Os trabalhos serão avaliados automaticamente. Respeite o nomes e o tipos das funções enunciadas acima.
3. Cada função que escrever deverá vir sempre acompanhada de uma assinatura. Isto é válido para as funções enunciada acima bem como para outras funções ajudantes que decidir implementar.
4. Para resolver estes problemas deve utilizar *apenas* a matéria dos seis primeiros capítulos do livro (até "Higher Order Functions" inclusivé). Pode usar qualquer função constante no **`Prelude`**.
5. Voltamos a informar: para obter a cotação máxima, deverá utilizar *pelo menos uma* função de ordem superior (**`map`**, **`filter`**, **`foldl`**, **`foldr`**, etc.), e deverá implementar *pelo menos uma* função recursivamente.
6. Lembre-se que as boas práticas de programação Haskell apontam para a utilização de várias funções simples em lugar de uma função única mas complicada.

Entrega. Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 1 de novembro de 2021.

Plágio. Os trabalhos de todos os alunos serão comparados por uma aplicação computacional. Relembramos aqui um excerto da sinopse: "Alunos detetados em situação de fraude ou plágio, plagiadores e plagiados, ficam reprovados à disciplina (sem prejuízo de ser acionado processo disciplinar concomitante)".