Princípios de Programação

Trabalho para casa 3

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

2019/2020

Neste trabalho pretende-se continuar a construir partes de um sistema de condução autónoma. Para tal, voltamos a precisar de manipular a posição geográfica de veículos e de outros objectos. Vamos representar posições geográficas como pares de números `Float`, isto é pelo tipo `(Float, Float)`, a que chamamos simplesmente *pontos*. Vamos também precisar de manipular *percursos* que representamos como listas de pontos, isto é através do tipo `[(Float, Float)]`.

A. Escreva uma função recursiva `distancia` que devolve a distância total de um percurso calculada como a soma dos comprimentos dos segmentos dados por pontos consecutivos no percurso. Considere que a `distancia` de um percurso com menos de dois pontos é zero. Um exemplo de utilização é:

```
ghci> distancia [(0,0), (0,0), (1,0), (1,10)]
11.0
ghci> distancia [(1,1), (3,4)]
3.6055512
```

Atente que, apesar de estar perante um enunciado idêntico ao do trabalho 2, o trabalho 3 exige funções de ordem superior, como poderá consultar nas notas abaixo.

B. Escreva a função `minimaDistanciaA` que recebendo um ponto inicial e uma lista de pontos, devolve a distância mínima entre o ponto inicial e cada um dos pontos.

```
ghci> minimaDistanciaA (0,0) [(1,1), (1,0), (0,3)]
1.0
```

C. Escreva a função recursiva `evitaPontos` que, recebendo

- uma distância d, por exemplo, 2
- uma lista de pontos a evitar xs, por exemplo, `[(0,0),(3,3)]`
- um percurso ys, por exemplo, `[(1,1),(3,4),(10, 10),(3, 30)]`

devolve todos os pontos do percurso ys, pela ordem original, que estejam a uma distância igual ou superior a d de cada um dos pontos de xs. Por outras palavras, dado um percurso ys, a função `evitaPontos` elimina os pontos de ys que estão a uma distância inferior a d de cada um dos pontos a evitar, xs.

No exemplo abaixo, o ponto `(1,1)` e o ponto `(3,4)` são evitados no percurso original porque estão a uma distância inferior a 2 dos pontos `(0,0)` e `(3,3)`, respectivamente. Os restantes pontos do percurso não estão próximos (isto é, não estão a uma distância inferior a 2) de nenhum ponto a evitar, por isso aparecem no percurso final.

```
ghci> evitaPontos 2 [(0,0), (3,3)] [(1,1), (3,4), (10, 10), (3, 30)]
[(10.0,10.0), (3.0,30.0)]
```

Notas

1. Os trabalhos serão avaliados automaticamente. Respeite os nomes e os tipos das *três* funções `distancia`, `minimaDistanciaA` e `evitaPontos`.
2. Não se esqueça de apresentar uma assinatura para cada função que escrever.
3. Para resolver estes problemas deve utilizar *apenas* a matéria dos capítulos do livro até ao capítulo funções de ordem superior (*Higher Order Functions*).
4. Na resolução de cada uma das funções tem de usar pelo menos uma função de ordem superior.
5. No conjunto das três funções deverá usar pelo menos um **`map`**, um **`filter`** e uma variante do `fold` (incluindo **`foldr1`** e **`foldl1`**). Por cada uma destas funções em falta será descontado 1/3 da nota obtida.
6. Como é normal, pode ainda usar qualquer função constante no **`Prelude`**.
7. 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 6 de novembro de 2019.

Ética. Os trabalhos de todos os alunos serão comparados por uma aplicação computacional. Lembre-se: "Alunos detetados em situação de fraude ou plágio, plagiadores e plagiados, ficam reprovados à disciplina (sem prejuízo de ser acionado processo disciplinar concomitante)".