Princípios de Programação

Trabalho para casa 2

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 precisamos 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
```

B. Escreva a função recursiva `fundePercursos` que recebendo dois percursos e um ponto inicial, devolve um único percurso que começa com o ponto inicial seguido de todos pontos de ambos os percursos. O percurso final deverá

- Manter a ordem relativa dos pontos em cada percurso dado,
- Minimizar a distância total do percurso resultado, e
- Em caso de empate o ponto da primeira lista aparece em primeiro lugar na lista resultado.

No resultado do exemplo abaixo podemos ver os pontos do percurso `[(1,3), (1,4), (5,5)]` a aparecer pela ordem dada, possivelmente entrecalados com pontos do outro percurso. A mesma observação pode ser feita para os pontos do percurso `[(1,1), (4,4)]`. Finalmente note que todas as outras possíveis fusões dos dois percursos têm uma distância não inferior à distância do percurso resultado.

```
ghci> fundePercursos [(1,3), (1,4), (5,5)] [(1,1), (4,4)] (0,0)
[(0.0,0.0), (1.0,1.0), (1.0,3.0), (1.0,4.0), (4.0,4.0), (5.0,5.0)]
```

C. Dado um determinado percurso e um ponto de interesse, escreva a função recursiva `adicionaParagem` que devolve um novo percurso que respeite a ordem dos pontos iniciais, mas com uma nova paragem no ponto de interesse, algures entre o primeiro e o último ponto. O novo percurso deverá minimizar a distância total. Em caso de empate, coloque o novo ponto na primeira posição mais próxima do início do trajeto (e que minimize a distância total). Assuma que o percurso tem pelo menos dois pontos. Por exemplo:

```
ghci> let p = [(0,0), (0,2), (2,2), (0,2), (0,4)]
ghci> adicionaParagem p (2,4)
[(0.0,0.0), (0.0,2.0), (2.0,4.0), (2.0,2.0), (0.0,2.0), (0.0,4.0)]
```

Notas

1. Os trabalhos serão avaliados automaticamente. Respeite os nomes e os tipos das *três* funções `distancia`, `adicionaParagem` e `adicionaParagem`.

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 recursão (*Recursion*). As definições das funções devem ser recursivas e não poderá usar nenhuma função de ordem superior. Pode usar qualquer função constante no **`Prelude`**.

4. 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 23 de outubro 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)".