Princípios de Programação

Trabalho Terceiro

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

2020/2021

Neste trabalho vamos desenvolver uma estrutura de dados em Haskell que nos permita acelerar pesquisas em estruturas ordenáveis.

A. Listas de Alguidares são listas de listas, sendo que cada sub-lista, ou alguidar, está ordenada e o seu primeiro elemento é superior ou igual ao último elemento da sub-lista anterior.
`[[1..200],[200..300],[350..800]]` é um exemplo de uma Lista de Alguidares. `[[1..200] ++ [200..300] ++ [350..800]]` também é um exemplo válido mas menos útil, visto que para encontrarmos algo precisamos de procurar num alguidar grande, enquanto no exemplo anterior, podemos encontrar o alguidar que interessa e procurar o elemento apenas nessa. É comum definir-se um tamanho máximo para cada alguidar.
1. Escreva a expressão correspondente a uma Lista de Alguidares vazia, chamada `listaDeAlguidaresVazia`.
2. Escreva a função polimórfica `adicionaAListaDeAlguidares` que receba o tamanho máximo dos alguidares, o elemento a adicionar e uma Lista de Alguidares, e devolva uma nova Lista de Alguidares. Pode assumir que na Lista de Alguidares dada, cada alguidar tem tamanho entre 1 e o tamanho máximo passado como argumento.
Note bem: a sua solução deverá observar primeiro a cabeça do alguidar seguinte, antes de tentar adicionar o elemento ao alguidar atual.
No exemplo seguinte adicionamos vários elementos a uma lista inicial, sempre mantendo o máximo de 3 elementos por alguidar.

```
> e = listaDeAlguidaresVazia
> e1 = adicionaAListaDeAlguidares 3 10 e
> e1
[[10]]
> e2 = adicionaAListaDeAlguidares 3 20 e1
```

```
> e2
[[10,20]]
> e3 = adicionaAListaDeAlguidares 3 30 e2
> e3
[[10,20,30]]
> e4 = adicionaAListaDeAlguidares 3 13 e3
> e4
[[10,13,20],[30]]
> e5 = adicionaAListaDeAlguidares 3 20 e4
> e5
[[10,13,20],[20,30]]
```

3. Escreva, recorrendo a pelo menos uma função de ordem superior, a função `elemListaDeAlguidares` que se comporta como a função **`elem`** do `Predule` mas que determina se o elemento se encontra numa Lista de Alguidares em vez de numa lista convencional.

```
> elemListaDeAlguidares 20 e5
True
> elemListaDeAlguidares 21 e5
False
```

4. Escreva, recorrendo a pelo menos uma função de ordem superior, a função `removerDaListaDeAlguidares` que remove um determinado elemento de uma Lista de Alguidares. A função recebe os seus argumentos pela ordem indicada.

```
> removerDaListaDeAlguidares 13 e5
[[10,20],[20,30]]
> removerDaListaDeAlguidares 10 $ removerDaListaDeAlguidares 13 e5
[[20],[20,30]]
> removerDaListaDeAlguidares 5 e5
[[10,13,20],[20,30]]
> removerDaListaDeAlguidares 20 e5
[[10,13],[30]]
```

5. Escreva, recorrendo a uma variante do fold, a função `fromList` que, recebendo o tamanho máximo das sub-listas, converte uma dada lista numa Lista de Alguidares.

```
> fromList 3 [1..20]
[[1,2,3],[4,5,6],[7,8,9],[10,11,12],[13,14,15],[16,17,18],[19,20]]
> fromList 3 $ map (^2) [-5..5]
[[0,1,1],[4,4,9],[9,16,16],[25,25]]
> fromList 4 []
[]
```

6. Defina a função de ordem superior `mapListaDeAlguidares` que, recebendo o tamanho máximo dos novos alguidares, uma função de um tipo ordenável noutro também ordenável, e uma Lista de Alguidares, retorna uma nova Lista de Alguidares com o resultado da aplicação da função a cada elemento da Lista de Alguidares antiga. De notar que a nova Lista de Alguidares tem de estar ordenada correctamente de acordo com os novos valores.

```
> mapListaDeAlguidares 2 (*2) e5
[[20,26],[40,40],[60]]
> mapListaDeAlguidares 2 (\x -> (x - 5) ^ 2) $ fromList 3 [1..10]
[[0,1],[1,4],[4,9],[9,16],[16,25]]
```

B. Key-Value Store Servidores de *Key-Value Store* são essenciais no desenho de sistemas distribuídos escaláveis. Têm como exemplos o Redis, Memcached, Riak, RocksDb, Tokyo Cabinet, Dynamo. Neste exercício vamos construir um *Key-Value Store* em Haskell, usando as nossas Listas de Alguidares para obter um melhor desempenho quando comparado com as listas nativas do Haskell.
1. Escreva uma função `createFastCache` que recebe o tamanho máximo das sub-listas (também chamados de *buckets*), e duas listas de chaves e valores, em que cada posição na lista de chaves corresponde à mesma posição na lista de valores. Esta função deverá devolver uma Lista de Alguidares.

```
> db = createFastCache 2 [10,20,20,30,50] ['a'..]
> db
[[(10,'a'),(20,'b')],[(20,'c'),(30,'d')],[(50,'e')]]
```

2. Escreva a função `fastGet` que, recorrendo a uma variante do fold, recebe uma Lista de Alguidares e um elemento do tipo de chave, e devolve uma lista com os vários valores associados a essa chave na Lista de Alguidares.
Note bem: não deverá percorrer todos os elementos da Lista de Alguidares, mas apenas cabeças dos alguidares. Os únicos alguidares que deve visitar na integra são aqueles onde poderá encontrar o elemento que procura.

```
> fastGet db 3
""
> fastGet db 10
"a"
> fastGet db 20
"bc"
```

Notas

1. Os trabalhos serão avaliados automaticamente. Respeite os nomes e os tipos de todas as funções enunciadas acima.

2. Cada função (ou expressão) que escrever deverá vir sempre acompanhada de uma assinatura. Isto é válido para as funções enunciadas acima bem como para outras funções ajudantes que decidir implementar.

3. 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`.

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 9 de novembro de 2020.

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)".