

TD 5 : Listes – Part 2

NB : quelques exercices de ce TD proviennent du site

https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems (qui est fortement recommandé)

1 Gray code

Le code de Gray de n bits est une séquence de tous les nombres binaires de n chiffres avec la propriété suivante : chaque deux nombres consécutifs dans la liste ne diffèrent que dans une position.

Par exemple, le code de Gray pour $n = 4$ est

0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000.

Le premier élément du code est toujours $00\dots 0$ et le dernier est toujours $100\dots 0$. En utilisant cette propriété, une façon de construire le code pour n bits est de construire deux copies du code pour $n - 1$ bits, inverser la deuxième copie, mettre 0 au début de chaque élément de la première copie, 1 au début de chaque élément de la deuxième copie, et prendre la concaténation.

1. Donner une implémentation d'une fonction récursive `gray :: Int -> [String]` qui produit le code de Gray, en utilisant la compréhension de listes.
2. Donner une implémentation qui utilise `map`.
3. Donner une implémentation qui utilise `foldr`.

2 Run-length Encoding

On est donné une liste avec beaucoup de doublons. On aimerait produire une version compressée de cette liste qui, à la place de chaque séquence d'éléments identiques, contient une paire de la forme `(element, frequency)`.

Exemple :

```
*Main> rle "aabbbaacaad"
[( 'a' ,2), ( 'b' ,3), ( 'a' ,1), ( 'c' ,1), ( 'a' ,2), ( 'd' ,1)]
```

Naturellement, il nous faut aussi une fonction qui fait la conversion inverse :

```
*Main> unrle $ rle "aabbbaacaad"
"aabbbaacaad"
```

1. Écrire deux implémentations récursives pour les fonctions `rle`, `unrle`
2. Écrire une implémentation de `rle` qui utilise `foldr`
3. Écrire une implémentation de `unrle` qui utilise `foldr`
4. Écrire une implémentation de `unrle` qui utilise `foldl`
5. Écrire une implémentation de `unrle` qui utilise la compréhension de listes et `concatMap`.

NB : les fonctions `fst :: (a,b) -> a` et `snd :: (a,b) -> b` retournent respectivement le premier et le deuxième élément d'une paire.

3 List Frequency

Donner une fonction `listFreq` qui, étant donné une liste, retourne une deuxième liste qui signifie pour chaque élément de la liste donnée, sa fréquence.

Exemple :

```
*Main> listFreq "aabaabacb"  
[( 'a' , 5) , ( 'b' , 3) , ( 'c' , 1) ]
```

1. Donner une implémentation qui utilise la compréhension de listes
2. Donner une implémentation qui utilise `filter`

4 Permutations

Donner une fonction qui, étant donné une liste qui contient des éléments distincts, retourne une liste de toutes ses permutations.

Exemple :

```
*Main> allperms "abc"  
["abc", "bac", "bca", "acb", "cab", "cba"]
```