TP n^{o}6 : Schémas de programmes

Q1 La chauffe

(a) Réécrivez les fonctions suivantes sur les listes en utilisant le schéma de programme *reduce* d'Haskell (`foldr`) :
- somme des éléments
- produit des éléments d
- longueur
- tri dans l'ordre croissant
- concaténation de deux listes

(b) Sur le même principe écrivez des fonctions calculant pour un vecteur
- la norme
- l'incrémentation de tous ses éléments
- l'écart-type

(c) Réécrivez les 3 parcours d'un arbre binaire en utilisant un schéma de programme *reduce*

(d) Sur le même principe écrivez des fonctions calculant pour un arbre binaire
- somme des éléments
- l'incrémentation des éléments

Q2 Les sous-listes

Une *sous-liste* d'une liste l est obtenue
- en enlevant n éléments de l (avec éventuellement $n = 0$)
- en conservant les éléments dans l'ordre de l

Exemple : les sous-listes de `[1,2,3]` sont
`[[],[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]]`.
Contre exemple : `[3,2]` n'est pas une sous-liste de `[1,2,3]`.

Faites une fonction `sous_listes` qui retourne une liste contenant toutes les sous-listes d'une liste. Utilisez les schémas *reduce* et *map* de Haskell (`foldr` et `map`).

Q3 Les matrices creuses

Une *matrice creuse* est une matrice qui comporte un grand nombre de valeurs nulles.
Exemple :

$$\begin{pmatrix} 0 & 0 & 7 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \end{pmatrix}$$

On choisit de représenter une matrice creuse sous la forme : $((L, C), [(i, j), a_{ij} \ldots])$ avec

- L : nombre de lignes de la matrice
- C : nombre de colonnes
- $((i, j), a_{ij})$: valeur non nulle se situant à la ligne i et à la colonne j

Exemple : la matrice ci-dessus est représentée par
`((4,6),[((1,3),7),((1,6),4),((4,2),5)])`

(a) Écrire une fonction `transpose` qui calcule la transposée d'une matrice creuse (inversion des lignes et des colonnes)

(b) Écrire une fonction `addition` qui additionne deux matrices creuses. Remarques sur l'addition de 2 matrices creuses A et B :

- $A + B = (a_{ij} + b_{ij})$
- A et B doivent avoir les mêmes dimensions
- la matrice résultat ne doit pas avoir 2 valeurs aux mêmes indices

Q4 Les matrices fonctionnelles

On reprend la représentation des *matrices fonctionnelles* vues dans le TP3 sur les Fonctionnelles.

(a) Écrire une fonction `diagonale` qui retourne la liste des éléments de la diagonale d'une matrice

(b) Écrire une fonction `produit_matrix` qui retourne le produit de 2 matrices. Par définition le produit de 2 matrices $A \times B = C$ avec $c_{ij} = \sum_{k=1..n}(a_{ik} \times b_{kj})$

Q5 Le chiffrement de César

On se propose de faire un décodage simplifié du *chiffrement de César* : chaque lettre d'un message en clair est remplacée par une autre lettre à distance fixe dans l'alphabet.
Exemple avec un décalage de 3 :

- 'A' est remplacée par 'D'
- 'B' est remplacée par 'E'
- ...

Un message est une liste de mots. Un mot est une liste de lettre. *Exemple* de message chiffré de 7 mots :

```
[['P','I'],['W','S','P','I','M','P'],['W','I'],['P','I','Z','I'],['E'],['P'],['I','W','X']]
```

(a) Écrire une fonction `nb_occ` qui retourne la liste des occurences de chaque lettre d'un message sous forme de couples.
Exemple : avec le message ci-dessus la fonction devra retourner `[('P',5),('I',6),('W',3),('S',1),('M',1),('Z',1),('E',1),('X',1)]`
Pour cela utiliser le schéma *fold left* (`foldl` en Haskell) :

- d'une part pour appliquer une fonction de concaténation de chaine de caractères sur une liste de mots afin d'obtenir une unique liste de caractères
- d'autre part pour appliquer une fonction qui compte le nombre d'occurrences d'une lettre dans une liste de caractères

(b) Écrire une fonction `freq` qui donne la fréquence d'apparition de chaque lettre du message sous forme de couples. La *fréquence d'apparition* d'une lettre est son nombre d'occurrences divisé par le nombre total de lettres dans le message.On peut utiliser pour cela le schéma *map* (`map` en Haskell) qui applique le calcul de la fréquence à tous les éléments de la liste d'occurences. *Exemple* : avec le message ci-dessus la fonction devra retourner

```
[('P',0.26...),('I',0.31...),('W',0.15...),('S',0.05...),('M',0.05...),('Z',0.05...),
('E',0.05...),('X',0.05...)]
```

(c) Écrire une fonction `trouve_cle` qui calcule le décalage (la clé) utilisé dans le chiffrement du message codé.

- Pour cela on supposera que la lettre la plus fréquemment utilisée dans le message codé correspond au 'E' en clair (lettre la plus fréquente du français). Il suffira donc de calculer la distance entre cette lettre la plus fréquence et le 'E' (évidemment ça ne marchera pas à tous les coups).
- Pour calculer cette distance on peut se servir de la fonction

prédéfinie `Data.Char.ord` qui donne le code *ASCII* d'un caractère. Attention il faudra, pour utiliser cette fonction, ajouter `import Data.Char` dans votre programme/module Haskell.
- On peut ici utiliser le schéma *fold left* pour appliquer une fonction qui recherche la fréquence max dans une liste de fréquences de lettres.

(d) Écrire enfin la fonction `dechiffre` qui déchiffre un message codé par le chiffrement de César.
- Pour cela utiliser le schéma *map*
  * d'une part pour appliquer la fonction qui ajoute la clé à toutes les lettres d'un mot
  * d'autre part pour appliquer cette précédente fonction à tous les mots d'un message
- Vous pouvez utiliser la fonction Haskell prédéfinie `Data.Char.chr` qui affiche le caractère d'un code *ASCII*

(e) Faites un programme de test pour vérifier vos fonctions avec le message de l'exemple plus celui ci :

```
[['N','B','S','D','I','F'],['T','J'],['W','P','U','S','F'],['N','F','T','T','B','H','F'],['B'],
['C','F','B','V','D','P','V','Q'],['E','F'],['F']]
```