TP n^{o}4 : Types structurés : les Listes

Q1 La chauffe

(a) Réécrivez dans un module `List.hs` les fonctions suivantes sans utiliser de schémas de programmes :
- somme des éléments d'une liste
- produit des éléments d'une liste
- longueur d'une liste
- insertion d'un élément dans une liste triée dans l'ordre croissant
- tri d'une liste dans l'ordre croissant
- concaténation de deux listes

(b) Faites la fonction `inverse` qui inverse les éléments d'une liste **sans utiliser** `reverse`

Q2 Les max

(a) Écrire une fonction `max_un` qui retourne le plus grand nombre d'une liste de nombres, **sans utiliser** la fonction `maximum`. Cette fonction sera partielle : elle ne s'appliquera pas à une liste vide.

(b) Écrire une fonction `max_deux` qui retourne les 2 plus grands nombres d'une liste de nombres.
- la fonction sera partielle : elle ne s'appliquera pas à des listes de moins de 2 éléments
- *Exemple* : `max_deux` pour `[23,8,7,12,20,1]` → `(23,20)`

(c) Écrire une fonction `max_trois` qui retourne les 3 plus grands nombres d'une liste de nombres

Q3 Les intervalles

(a) Écrire une fonction `intervalle_asc inf sup` qui retourne la liste des entiers compris entre `inf` et `sup` dans l'ordre ascendant.

(b) Même chose mais dans l'ordre descendant pour la fonction `intervalle_desc inf sup`

Q4 Préfixes et suffixes

(a) Écrire une fonction `prefixes` qui, à partir d'une liste, retourne tous les débuts de cette liste. *Exemple* : pour la liste `['a','b','c']` on obtient la liste `[[],['a'],['a','b'],['a','b','c']]`

(b) Écrire une fonction `suffixes` qui, à partir d'une liste, retourne tous les fins de cette liste. *Exemple* : pour la liste `['a','b','c']` on obtient la liste `[['a','b','c'],['b','c'],['c'],[]]`

Q5 Mots de Lyndon

Dans cet exercice

- on appellera "mot" une liste on vide de caractères. Par exemple `['b','i','e','r','e']`
- la relation d'ordre lexicographique (ordre des mots dans le dictionnaire) sera notée $\prec$.
  Par exemple `['b','i','e','r','e']` $\prec$ `['b','o','n','b','o','n']`

(a) Écrire la fonction `inferieur` qui, étant donné 2 mots u et v retourne `True` si $u \prec v$ et `False` sinon.
Exemple : si $u =$ `['b','i','e','r','e']` et $v =$ `['b','o','n','b','o','n']` alors `inferieur u v` retourne `True`.

(b) Écrire la fonction `conjugue` qui, pour un mot `u` de longueur `n` et un entier `i` (avec $1 \leqslant i \leqslant n$) retourne le conjugué du mot u qui débute par le i^{ieme} élément de u.
Un conjugué d'un mot u de la forme $[u_1, u_2, \ldots, u_n]$ est un mot de la forme $[u_i, u_{i+1}, \ldots, u_n, u_1, u_2, \ldots, u_{i-1}]$ où $1 \leqslant i \leqslant n$.
Exemples :

```
conjugue ['a','b','c','d'] 2
"bda"
conjugue ['a','b','c','d','e'] 4
"deabc"
```

(c) Écrire la fonction `lyndon` qui teste si un mot u est un mot de Lyndon.
Un mot u est un mot de Lyndon si pour tout conjugué v de u on a $u \prec v$.
Exemple : si u = `['a','a','a','b']` alors `lyndon u` retourne `True`.

On ne considère à partir d'ici que les mots de Lyndon ayant seulement les lettres '0' et '1'. Le but est maintenant d'obtenir une liste de tous les mots de Lyndon de longueur $n \geqslant 1$. Réalisez cela avec les questions suivantes :

(d) Écrire la fonction `insere_liste` qui, étant donné 2 listes de mots l_1 et l_2 renvoie une liste triée contenant tous les mots **distincts** contenus dans les listes l_1 et l_2 avec les contraintes suivantes
- l_1 n'est pas forcément triée
- l_2 est triée : les mots de la liste l_2 sont tous distincts et ordonnés selon l'ordre $\prec$.

Exemple :

```
insere_liste  [['0','1'],['1','0']] [['0','0','1'],['0','1']]
["001","01","10"]
```

(e) Écrire la fonction `fusion_liste` qui, étant donné 2 listes l_1 et l_2 de mots de Lyndon, renvoie la liste triée de tous les mots de Lyndon obtenus en concaténant un mot de l_1 avec les mots de l_2. *Exemple* :

```
fusion_liste [['0','0'],['0','1','1']] [['0','0','1'],['0','1']]
["00001","0001"]
```

(f) Écrire la fonction `genere` qui, pour un entier n (avec $n \geqslant 1$), retourne la liste de tous les mots de Lyndon de longueur n.

Pour cela on se sert d'une propriété qui dit que tout mot de Lyndon de longueur $\geqslant 2$ est la concaténation de 2 mots de Lyndon f et $g$tels que $f \prec g$.

- On peut donc obtenir les mots de Lyndon de longueur $n > 1$ à partir de 2 mots de Lyndon f de longueur l_f et g de longueur l_g avec
  - $f \prec g$
  - $l_f + l_g = n$
- Un cas pratique pour la récursion est que l'on peut construire les mots de Lyndon de longueur $n > 1$ à partir de 2 mots de Lyndon f et g tels que
  - $l_f = 1$ et $l_g = n - 1$ et aussi
  - $l_f = n - 1$ et $l_g = 1$

Vérifiez que les mots de Lyndon

- de longueur 1 sont `['0'],['1']`
- de longueur 2 sont `['0','1']`
- de longueur 3 sont `['0','0','1'],['0','1','1']`
- de longueur 4 sont `['0','0','0','1'],['0','0','1','1'],['0','1','1','1']`