Ordres et tris

Objectifs Manipulation des listes et définitions de fonctions de tri.

Exercice 1 - Relations d'ordre

Un relation binaire $\mathcal{R}$ sur un ensemble E est une *relation d'ordre* si elle est :
- réflexive ($\forall x \in E.\ x\mathcal{R}x$)
- transitive ($\forall x, y, z \in E.\ x\mathcal{R}y$ et $y\mathcal{R}z$ implique $x\mathcal{R}z$)
- antisymétrique ($\forall x, y \in E.\ x\mathcal{R}y$ et $y\mathcal{R}x$ implique $x = y$)

Par exemple, sur les entiers naturels, les relations "inférieur ou égal" et "est multiple de" sont des relations d'ordre. Dans cet exercice, nous allons définir des relations d'ordre sur les listes.

Ordre lexicographique

L'ordre lexicographique est la relation d'ordre qui permet de trier les mots dans un dictionnaire. Sur les listes, l'ordre lexicographique est défini de la façon suivante :

$$[x_1\ ;\ \ldots\ ;x_n] \leq [y_1\ ;\ \ldots\ ;y_m] \text{ ssi } \begin{cases} \exists i.x_i \leq y_i \wedge \forall j < i.x_j = y_j \\ \text{ ou} \\ \forall j \leq n.x_j = y_j \wedge n \leq m \end{cases}$$

1. Écrire la fonction `lexico: 'a list -> 'a list -> bool` qui compare deux listes selon l'ordre lexicographique.

Ordre multi-ensemble

Un multi-ensemble est un ensemble dans lequel des éléments peuvent être répétés. Ainsi, un multi-ensemble m construit à partir d'éléments d'un ensemble E peut être défini comme une fonction qui à chaque élément de E retourne le nombre de fois qu'il apparaît dans m. On note $m(x)$ le nombre d'occurrences de x dans m. L'ordre multi-ensemble est défini de la façon suivante :

$$m_1 \leq m_2 \text{ ssi } \begin{cases} m_1 = m_2 \text{ ou} \\ \forall x \in E.m_2(x) < m_1(x) \Rightarrow \exists y \in E.x < y \wedge m_1(y) < m_2(y) \end{cases}$$

Les listes étant une implantation des multi-ensembles, on peut définir l'ordre multi-ensemble sur les listes de la façon précédente.

1. Écrire une fonction `occurrence: 'a -> 'a list -> int` qui compte le nombre d'occurrences d'un élément dans une liste.
2. Écrire la fonction `supprime: 'a -> 'a list -> 'a list` qui supprime toutes les occurrences d'un élément dans une liste. Cette fonction ne doit pas nécessairement préserver l'ordre de la liste.
3. Écrire la fonction `multiset_of_list: 'a list -> ('a * int) list` qui à partir d'une liste crée le multi-ensemble correspondant où à chaque élément différent de la liste, on lui associe son nombre d'occurrences dans la liste d'origine. Par exemple, `multiset_of_list [1; 6; 3; 3; 1; 3]` retourne la liste `[(1,2); (6, 1); (3; 3)]`.
4. Écrire la fonction `is_sup: ('a * int) -> ('a * int) list -> bool` qui vérifie la condition suivante :
$$\texttt{is\_sup (x,n) m} = m(x) < n$$
5. Écrire la fonction `sup: ('a * int) list -> ('a * int) list -> 'a option` qui retourne le plus grand élément de la première liste selon la condition précédente.
6. Écrire la fonction `multiset: 'a list -> 'a list -> 'a` qui compare deux listes selon l'ordre multi-ensemble.

Exercice 2 - Tri

Tri par sélection

1. Écrire une fonction `selection_min: 'a list -> 'a option * 'a list` qui sépare le plus petit élément d'une liste du reste de la liste. Cette fonction ne doit pas nécessairement préserver l'ordre de la liste.
2. À partir de la fonction précédente, définir une fonction `tri_selection: 'a list -> 'a list` qui tri une liste par ordre décroissant.

Tri à bulles

1. Écrire une fonction `bubble: 'a -> 'a list -> 'a list` qui est telle que par exemple `bubble 6 [4; 7; 1; 8; 2]` retourne la liste `[4; 6; 1; 7; 2; 8]`.
   La fonction `bubble x l` fait remonter la valeur `x` dans la liste `l` jusqu'à ce que `x` rencontre une valeur `y` supérieur. À ce moment, `x` prend la place de `y` et `y` remonte dans la suite de la liste. Ainsi, la plus grande valeur de la liste `l` se trouve à la fin de la liste retournée par la fonction `bubble`.
   Si on appelle la fonction `bubble` sur la liste obtenue précédemment les deux dernières valeurs de la liste sont triées (`bubble 4 [6; 1; 7; 2; 8] = [4; 1; 6; 2; 7; 8]`).
2. Écrire une fonction `bubble_sort: 'a list -> 'a list` qui appelle la fonction `bubble` jusqu'à l'obtention d'un *point fixe* (`bubble (List.hd l) (List.tl l) = l`).
3. Quelle est la complexité de la fonction `bubble_sort` lorsqu'on l'appelle sur une liste triée dans l'ordre croissant et sur une liste triée dans l'ordre décroissant ?