T.P. 04.2
Multi-ensembles finis

■ ■ ■

Partie I : Présentation

Formellement, pour un ensemble D, un multi-ensemble fini de D est une fonction de D dans $\mathcal{N}$ nulle presque partout. Un peu moins formellement, c'est un sous-ensemble fini de D dont les éléments ont un ordre de multiplicité.
Informatiquement, n-uplet, multi-ensembles et ensembles finis peuvent tous être stocké sous forme de listes. Les opérations doivent par contre maintenir des invarients spécifique dans chacun des cas.
Pour les multi-ensembles par exemple, les listes `[4; 1; 3; 1; 1; 3]` et `[1; 1; 3; 4; 3; 1]` sont 2 représentations du même objet mais `[1;4;3;4]` est différent.
On peut définir 4 opérations ensemblistes sur les multi-ensembles :

- **La somme** $M +_m N$: l'ordre de multiplicité d'un élément x de $(M +_m N)$ est la somme de son ordre de multiplicité dans M et dans N (i.e. $\forall x,\ (M +_m N)(x) = M(x) + N(x)$).
  Par exemple, $\{0,0,1,2\} +_m \{0,2,2,2\}$ est égal à $\{0,0,1,2,0,2,2,2\} = \{0,0,0,1,2,2,2,2\}$.
- **L'union** $M \cup_m N$: l'ordre de multiplicté d'un élément x de $(M \cup_m N)$ est le maximum des deux ordres de multiplicité de x dans M et N (i.e. $\forall x,\ (M \cup_m N)(x) = \max(M(x), N(x))$).
  Par exemple, $\{0,0,1,2\} \cup_m \{0,2,2,2\}$ est égal à $\{0,0,1,2,2,2\}$.
- **L'intersection** $M \cap_m N$: l'ordre de multiplicté d'un élément x de $(M \cap_m N)$ est le minimum des deux ordres de multiplicité de x dans M et N (i.e. $\forall x,\ (M \cap_m N)(x) = \min(M(x), N(x))$).
  Par exemple, $\{0,0,1,2\} \cap_m \{0,2,2,2\}$ est égal à $\{0,2\}$.
- **La différence** $M -_m N$: l'ordre de multiplicité d'un élément x de $M -_m N$ est égal à $\max(0, M(x) - N(x))$.
  Par exemple, $\{0,0,1,2\} -_m \{0,2,2,2\}$ est égal à $\{0,1\}$.

Exercice 1. Définissez une fonction remove prenant pour argument un élément x et une liste m. Cette fonction retournera la liste m dans laquelle la première occurence de x aura été supprimée. Si x n'apparaît pas dans m, la liste sera retournée inchangée.

value remove : $\alpha \to \alpha\ list \to \alpha\ list$

Exercice 2. Programmez quatres fonctions sum_m, union_m, intersect_m et subtract_m implémentant les opérations décrites ci-dessus.

value sum_m : $\alpha\ list \to \alpha\ list \to \alpha\ list$
value union_m : $\alpha\ list \to \alpha\ list \to \alpha\ list$
value intersect_m : $\alpha\ list \to \alpha\ list \to \alpha\ list$
value subtract_m : $\alpha\ list \to \alpha\ list \to \alpha\ list$

Partie II : Représentation canonique d'un multi-ensemble

On suppose dans cette partie que l'ensemble de base D est totalement ordonné. On appelle représentant canonique d'un multi-ensemble $m = \{x_1, \ldots, x_n\}$ l'unique liste $\overline{m} = [x_{\sigma(1)}; \ldots; x_{\sigma(n)}]$ telle que σ soit une permutation de $[\![1, n]\!]$ et $\overline{m}$ soit triée (i.e. $x_{\sigma(1)} \leqslant \cdots \leqslant x_{\sigma(n)}$). Calculer le représentant canonique d'un multi-ensemble revient donc à trier n'importe quelle liste le représentant.

Nous allons écrire pour cela un algorithme de tri simple. Le principe du *tri par insertion* est le suivant : pour trier une liste $[x_0; x_1; \ldots; x_{n-1}]$, on part de la liste vide [ ] à laquelle on ajoute successivement x_0, puis x_1, ..., puis x_{n-1} à la "bonne place", c'est-à-dire de manière à ce que la liste formée reste triée. Ce mécanisme peut s'énoncer d'une manière équivalente faisant apparaître un schéma récursif : pour trier la liste $t :: q$, il suffit de trier q et d'insérer t à sa place.

Exercice 3. Essayez d'appliquer cet algorithme « à la main » sur un exemple simple.

Exercice 4. Écrivez une fonction insert qui prenne pour arguments un objet x et une liste m supposée déja triée et insère x dans la liste à une place convenable.

Exercice 5. Déduisez-en une fonction sort prenant une liste pour argument et la retournant triée. Quel st le coût de cette algorithme de tri en fonction de la longueur de la liste à trier ?

Exercice 6. Déduisez-en une fonction testant l'égalité de deux multi-ensembles.

Partie III : ensemble des représentants d'un multi-ensemble

Exercice 7. Ecrivez une fonction qui à partir d'un représentant d'un multi-ensemble renvoie la liste de tous ces représentants.
Pour cela, pensez que réccursivement, construire les permutations de `t :: q` c'est insérer `t` à toutes les positions possibles dans toutes les permutations de `q`.

Partie IV : itérateurs (bonus)

Plutôt que d'utiliser du filtrage pour manipuler des listes, il est possible de tout écrire à partir des fonctions :
value it_list : $(\alpha \to \beta \to \alpha) \to \alpha \to \beta\ list \to \alpha$
`it_list f a [b1; ...; bn]` is `f (... (f (f a b1) b2) ...) bn`.
value list_it : $(\alpha \to \beta \to \beta) \to \alpha\ list \to \beta \to \beta$
`list_it f [a1; ...; an] b` is `f a1 (f a2 (... (f an b) ...))`.
que l'on appelle les itérateurs sur les listes.

Exercice 8. Sauriez vous réécrire un troisième itérateur classique sur les listes :
value map : $(\alpha \to \beta) \to \alpha\ list \to \beta\ list$
`map f [a1; ...; an]` applies function `f` to `a1`, ..., `an`, and builds the list `[f a1; ...; f an]` with the results returned by `f`.
à partir des précédants.
Tentez le même jeu avec la fonction
value list_append : $\alpha\ list\ list \to \alpha\ list$
qui prend une iste de liste et renvoie la liste de toutes les listes mises bout à bout.