Option informatique : TP 4

Ce TP est une collection d'exercices divers et variés destinés à vous aider à réviser. Vous pouvez traiter les exercices dans le désordre.

1 Parcours de listes

(a) Écrire une fonction `average : int list -> float` qui calcule la moyenne d'une liste d'entiers. On pourra renvoyer `0.` dans le cas de la liste vide.
(b) Écrire une fonction `sorted : int list -> bool` qui teste si une liste est triée dans l'ordre croissant non-strict. Écrire une seconde version pour l'ordre croissant strict.
(c) Écrire une fonction `maxmin : int list -> int * int` qui, étant donné une liste non vide `l`, trouve la paire `(a,b)` telle que : (i) `b` apparaît dans `l` après `a` (ii) `b - a` est maximal (iii) la fonction est linéaire en temps.

```
maxmin [1;2;3;0;6;2;6] = (0,6);;
maxmin [2;3;7;0;5;2;6;-1] = (0,6);;
maxmin [1;3;7;2;4;9;0;3;0;2] = (1,9);;
maxmin [1] = (1,1);;
```

(d) Tri par insertion basique : on veut trier une liste de type `'a list` (on suppose que les éléments sont comparables avec `<`). Pour cela on commencera par écrire une fonction `insert : 'a list -> 'a -> 'a list` telle que `insert l x` ajoute `x` dans la liste triée`l` et renvoie une liste triée (contenant un élément de plus).
Ensuite, écrire une fonction `sort : 'a list -> 'a list` qui ajoute successivement chaque élément de son argument à une liste maintenue triée.

```
insert [2;4] 1 = [1;2;4];;
insert [2;4] 3 = [2;3;4];;
insert [] 17 = [17];;
insert [10;11;19] 17 = [10;11;17;19];;
sort [1;2;3] = [1;2;3];;
sort [2;4;6;1;2;4] = [1;2;2;4;4;6];;
sort [10;9;8;7;5;0;11] = [0;5;7;8;9;10;11];;
```

2 Multi-ensembles

On s'intéresse à la notion mathématique de multi-ensemble. Un multi-ensemble sur un type α est une fonction $\alpha \to \mathbb{N}$, qui à chaque valeur du type α associe sa *multiplicité*. Un ensemble est un multi-ensemble tel que la multiplicité de chaque élément est dans $\{0, 1\}$. On dit qu'un élément x appartient à un multi-ensemble m ssi $m(x) > 0$ (avec $m(x)$ la multiplicité de x dans m). On appelle *support* du multi-ensemble l'ensemble des valeurs dont la multiplicité est non nulle.

2.1 Multi-Ensembles Finis

Un multi-ensemble est fini si son support est un ensemble fini. On représente les multi-ensembles finis en Caml par le type α list. La multiplicité d'un élément dans le multi-ensemble est son nombre d'occurrence dans la liste.
(a) Expliquez clairement la différence entre les notions (finies) d'ensemble et de multi-ensemble, les notions de liste et de multi-ensemble.
(b) Écrire une fonction `mem : 'a list -> 'a -> bool` qui teste si un élément appartient à un multi-ensemble fini.
(c) Écrire une fonction `multiplicity : 'a list -> 'a -> int` qui calcule la multiplicité d'un élément dans un multi-ensemble.
(d) Écrire une fonction `support : 'a list -> 'a list` qui calcule le support (une liste où chaque élément n'apparait qu'une fois) d'une multi-ensemble fini donné.

2.2 Opérations

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 multiplicité 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 multiplicité 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 $\{0,1\}$.

(a) Écrire une fonction `add : 'a list -> 'a -> int -> 'a list` telle que `add m x i` ajoute `x` avec la multiplicité `i` au multi-ensemble `m`.
(b) Écrire une fonction `combine : (int -> int -> int) -> 'a list -> 'a list -> 'a list` qui, à une fonction f et deux multi-ensembles finis M_1 et M_2, associe le multi-ensemble M tel quel $\forall x,\ M(x) = f(M_1(x), M_2(x))$.
(c) Implémenter quatre fonctions qui correspondent respectivement à la somme, l'union, la différence et l'intersection de multi-ensembles finis représentés par des listes.
(d) Si on représente les multi-ensembles par des fonctions Caml de type `'a -> int`, peut-on écrire les quatre opérations précédentes facilement ? Si oui, les implé-

menter. On remarquera qu'une application partielle judicieuse permet de passer de la représentation par une liste à la représentation par une fonction.

2.3 Forme compacte

La forme compacte d'un multi-ensemble fini est une liste d'association de type `('a * int) list`, qui associe à chaque élément présent dans le multi-ensemble sa multiplicité.
(a) Écrire une fonction `compact : 'a list -> ('a * int) list` qui retourne la forme compacte d'un multi-ensemble.
(b) Écrire une fonction qui, étant donné un multi-ensemble compact, en enlève une occurrence d'un élément (et l'élément lui-même si sa multiplicité était de 1). La fonction aura le type `('a * int) list -> 'a -> ('a * int) list`.

Note : une partie des exercices sont inspirés des TPs de Vincent Simonet qui sont sous license libre. Ce document est donc sous license GPL.