▶ **MPSI – Option Informatique**
Année 2002, Deuxième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Multi-ensembles

*Nous nous intéressons aujourd'hui à une notion courante en informatique théorique, celle de multi-ensemble fini. Un multi-ensemble est en quelque sorte un ensemble fini avec une notion d'*ordre de multiplicité *de chaque élément. Par exemple* $\{7, 5, 9\}$ *et* $\{9, 5, 5, 7\}$ *représentent des multi-ensembles différents : l'ordre de multiplicité de* 5 *est* 1 *dans le premier cas et* 2 *dans le second.*
Dans ce sujet, on s'intéresse à l'écriture de fonctions implémentant des opérations de base sur ces objets. On se contentera généralement d'algorithmes naïfs ayant une complexité polynomiale en la taille des entrées, même si des solutions plus efficaces peuvent souvent être proposées.

1 Ensembles finis

On représente souvent un ensemble fini $\{x_1, \ldots, x_n\}$ en *Caml* par une liste de la forme $[x_1; \ldots; x_n]$. Cette représentation est surjective : toute liste correspond à un ensemble (mais elle n'est pas injective : un ensemble peut être représenté par plusieurs listes). Notons que l'on n'interdit pas ici qu'un même élément apparaisse plusieurs fois dans la liste représentant un ensemble.

▶ **Question 1** *Écrivez une fonction* mem *telle que* mem x e *renvoie un booléen indiquant si l'élément* x *est dans l'ensemble* e *ou non.*

value mem : $\alpha \to \alpha\ list \to bool$

▶ **Question 2** *Programmez trois fonctions* union, intersect *et* subtract *calculant respectivement l'union, l'intersection et la différence de deux ensembles.*

value union : $\alpha\ list \to \alpha\ list \to \alpha\ list$
value intersect : $\alpha\ list \to \alpha\ list \to \alpha\ list$
value subtract : $\alpha\ list \to \alpha\ list \to \alpha\ list$

▶ **Question 3** *Écrivez une fonction* card *qui calcule le cardinal d'un ensemble.*

value card : $\alpha\ list \to int$

2 Multi-ensembles finis

2.1 Présentation

Si D est un ensemble, un multi-ensemble fini M d'éléments de D est une application de D dans $\mathbb{N}$ qui est nulle sauf pour un nombre fini d'éléments de D. De manière moins formelle, un multi-ensemble M est un ensemble fini dans lequel chaque élément a un ordre de multiplicité. Par exemple $M = \{0; 0; 2; 2; 2; 3\}$ est un multi-ensemble d'entiers. L'ordre de multiplicité de 2 dans M est 3 ($M(2) = 3$), celui de 3 est 1 ($M(3) = 1$). On peut changer l'ordre dans lequel on écrit les éléments (si on ne change pas le nombre de fois qu'ils apparaissent). Par exemple $M = \{0; 2; 0; 2; 3; 2\}$ mais $M \neq \{0; 2; 3\}$. Dans la suite, on sous-entend que les (multi-)ensembles sont finis.

▶ **Question 4** *Expliquez clairement la différence entre les notions (finies) d'ensemble et de multi-ensemble, les notions de liste et de multi-ensemble.*

▶ **Question 5** *Écrivez une fonction Caml* multi *telle que* multi x m *calcule l'ordre de multiplicité de* x *dans* m.

value multi : $\alpha \to \alpha\ list \to int$

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 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 $\{0,1\}$.

▶ **Question 6** *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 : $\alpha \to \alpha\ list \to \alpha\ list$

▶ **Question 7** *Programmez quatres fonctions* sum_m, union_m, intersect_ , *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 : $\alpha\ list \to \alpha\ list \to \alpha\ list$

2.3 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 $\bar{m} = [x_{\sigma(1)}; \ldots; x_{\sigma(n)}]$ telle que σ soit une permutation de $[\![1, n]\!]$ et $\bar{m}$ soit triée (i.e. $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(n)}$). 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.

▶ **Question 8** *Essayez d'appliquer cet algorithme « à la main » sur un exemple simple.*

▶ **Question 9** *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.*

▶ **Question 10** *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 ?*

▶ **Question 11** *Déduisez-en une fonction testant l'égalité de deux multi-ensembles.*

▶ **MPSI – Option Informatique**
Année 2002, Deuxième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Multi-ensembles

Un corrigé

▶ Question 2

```
let rec union e1 = function
    [] -> e1
  | t :: q -> union (t :: e1) q
;;

let rec inter e1 = function
    [] -> []
  | t :: q ->
      if mem t e1 then t :: inter e1 q
      else inter e1 q
;;

let rec subtract e1 e2 =
  match e1 with
    [] -> []
  | t :: q ->
      if mem t e2 then subtract q e2
      else t :: subtract q e2
;;
```

▶ Question 3

```
let rec card = function
    [] -> 0
  | t :: q ->
      if mem t q then card q
      else 1 + card q
;;
```

▶ Question 5

```
let rec multi x = function
    [] -> 0
  | t :: q ->
      if x <> t then multi x q
      else 1 + multi x q
;;
```

▶ Question 6

```
let rec remove x = function
    [] -> []
  | t :: q ->
      if x = t then q
      else t :: remove x q
;;
```

▶ Question 7

```
let sum_m m1 m2 = union m1 m2;;

let rec union_m m = function
    [] -> m
  | t :: q -> t :: (union_m (remove t m) q)
;;

let rec intersect_m m = function
    [] -> []
  | t :: q when mem t m ->
      t :: (intersect_m (remove t m) q)
  | t :: q -> intersect_m m q
;;

let rec subtract_m m1 m2 =
  match m1 with
    [] -> []
  | t :: q when mem t m2 ->
      subtract_m q (remove t m2)
  | t :: q -> t :: (subtract_m q m2)
;;
```

▶ Question 9

```
let rec insert x = function
    [] -> [x]
  | t :: q when x <= t -> x :: t :: q
  | t :: q -> t :: insert x q
;;
```

▶ Question 10

```
let rec sort = function
    [] -> []
  | t :: q -> insert t (sort q)
;;
```

▶ Question 11

```
let rec equal m1 m2 =
  (sort m1) = (sort m2)
;;
```