TP1 - Révisions et bonnes habitudes

23 septembre 2011

Pour toutes questions, suggestions, remarques ou autres, n'hésitez pas à m'envoyer un mail à edbonnet@hotmail.com en mettant en objet [TP Caml].

1 Écrire du code clair

De par la nature des épreuves que vous allez affronter (essentiellement sur papier), il est important d'écrire du code propre et lisible. Pour cela, vous disposez de plusieurs armes :

- **l'indentation**, qui matérialise la structure des fonctions que vous écrivez, et les rendent donc plus lisibles.
- **les commentaires**, qui rendront reconnaissants les gens qui vous relisent.
- un choix **significatif** du nom des variables que vous définissez. Sans tomber dans des noms à rallonge, du genre `le_plus_petit_element_courant_du_tableau` vous pouvez trouver plus expressif que `x`. `mini` semble un bon compromis.

Même si le langage caml permet avec les références d'adopter un style impératif, le paradigme chouchou est le style fonctionnel. On évitera donc dans la mesure du possible d'utiliser les références.

2 La récursivité des listes

Par essence, la liste caml est un objet inductif :
`'a list = [] | 'a :: 'a list`. Elle se prête donc naturellement aux procédures récursives. On va commencer la séance par quelques échauffements. Oublions temporairement nos éventuelles connaissances des bibliothèques Caml.

Question 1 *Écrire une fonction* `extremes : int list -> int * int` *qui retourne le minimum et le maximum d'une liste d'entiers.*

Question 2 *Écrire une fonction* `somme : int list -> int` *qui calcule la somme des entiers d'une liste.*

Question 3 *Écrire une fonction* `miroir : 'a list -> 'a list` *qui retourne la liste miroir.*

Question 4 *Écrire une fonction* `concat : 'a list -> 'a list -> 'a list` *qui concatène deux listes.*

Question 5 *Écrire une fonction* `applique : 'a list -> ('a -> 'b) -> 'b list` *qui prend une liste et une fonction et qui rend la liste des images.*

Question 6 *Écrire une fonction* `partitionne : 'a list -> ('a -> bool) -> 'a list * 'a list` *qui prend une liste et une propriété et qui découpe la liste en deux : les éléments qui vérifient la propriété et ceux qui ne la vérifient pas.*

3 Quelques tris sur des tableaux

3.1 Tri par sélection

Le tri par sélection est sans doute l'approche la plus frontale du problème. Il consiste à chercher l'élément le plus petit et à l'échanger avec l'élément en première position, puis à chercher le deuxième plus petit élément et à l'échanger avec l'élément en deuxième position, et ainsi de suite.

Question 7 *Implémenter le tri par sélection.*

3.2 Tri par insertion

Le tri par insertion est le procédé que l'on fait à la main pour ranger ses cartes quand celles-ci nous sont données une par une. Quand on reçoit une nouvelle carte, on l'insère à sa place de sorte que la sous-partie triée grandit à chaque étape.

Question 8 *Implémenter le tri par insertion.*

3.3 Tri à bulles

Le tri à bulles consiste à parcourir le tableau en inversant si nécessaire deux éléments consécutifs. On itère cette opération jusqu'à ce qu'un parcours complet ne change plus le tableau.

Question 9 *Pourquoi ce procédé termine-t-il ? Dans le pire des cas, combien de parcours sont effectués ? Caractériser ce pire des cas.*

Question 10 *Implémenter le tri à bulles.*

3.4 Tri rapide

Vous avez étudié (ou étudierez) en TD le tri rapide. Si besoin est, j'en (re)parlerai. Pour la question suivante, il peut être plus agréable travailler avec des listes.

Question 11 *Implémenter le tri rapide.*

Vous avez vu (ou verrez) en TD que la complexité en moyenne du tri rapide est $O(n.log(n))$. C'est la meilleure complexité théorique pour un tri comparatif.

Question 12 *Écrire une fonction qui génère un tableau (ou une liste) aléatoire de taille et de bornes des entrées fixés en paramètre. Constater, par l'expérience, la suprématie du quasi-linéaire sur le quadratique.*

4 Décompositions en sommes d'un entier

La décomposition d'un entier en sommes est à la base de nombreux problèmes d'énumérations combinatoires (parcours sur une grille, nombre d'injections...). Ici on va s'intéresser aux décompositions qui ne tiennent pas compte de l'ordre. Ainsi 4 possède 5 décompositions : $\{1,1,1,1\}$, $\{1,1,2\}$, $\{1,3\}$, $\{2,2\}$, $\{4\}$.

Question 13 *À la main, combien y a-t-il de décompositions de l'entier 7 ?*

Question 14 *Écrire une fonction* `decomp : int -> int list list` *qui retourne toutes les décompositions de n. Pour quelle valeur de n, votre fonction n'est plus viable ?*

Question 15 *Combien y a-t-il de décompositions de l'entier 100 ?*

Question 16 *Combien y a-t-il de décompositions de l'entier 1000 ?*