Lycée Chateaubriand
Informatique MP 2002/03
Mathieu Giraud

TP Caml n° 6
Petits exercices et systèmes monétaires

1 Quatre exercices

Ces petits exercices sont complètement indépendants. Vous avez une heure pour en faire autant que vous pouvez, dans l'ordre que vous voulez. Passez ensuite au problème.

1.1 Suites

a. Etant donné une suite récursive donnée par u_0 et une formule de récurrence $u_{n+1} = f(u_n)$, créer une fonction `calcul u0 f n` qui calcule u_n.

b. Quel est le coût de p appels successifs à la fonction précédente ? Pour le diminuer, écrivez une fonction `précalcul u0 f nmax` qui renvoie un tableau de $n_{\max} + 1$ éléments contenant tous les éléments de u_0 à $u_{n_{\max}}$.

1.2 Arithmétique

c. Écrire une fonction `pgcd : int -> int -> int` utilisant l'algorithme d'Euclide. Traiter aussi le cas du `ppcm`.

d. Réaliser une fonction `inverse n x` trouvant l'inverse éventuel de x pour $(\mathbb{Z}/n\mathbb{Z}, \times)$.

1.3 Analyse

e. On s'occupe des fonctions réelles `float -> float`. En utilisant éventuellement une constante globale `epsilon`, écrire les fonctions

- `dérivée` telle que `dérivée f x` calcule $f'(x)$,
- `intègre` telle que `intègre f a b` calcule $\int_a^b f(t)dt$,
- et `primitive` telle que `primitive f` renvoie la primitive de f qui s'annule en zéro.

1.4 Tours de Hanoi

On souhaite écrire un programme qui résout le problème des *tours de Hanoi*. On dispose de trois emplacements, et de n pièces toutes de tailles différentes, posées au début sur le premier emplacement. A chaque étape on a le droit de ne déplacer qu'une seul pièce, et on a le droit de poser une pièce que sur une pièce plus grande. On a gagné si, à la fin, toutes les pièces sont sur le dernier emplacement.

f. Écrire rapidement une fonction `affiche m n` qui affiche à l'écran qu'il faut déplacer une pièce de la colonne m vers la colonne n.

g. En remarquant que, pour bouger n pièces de l'emplacement 1 à l'emplacement 3, il suffit de bouger $n-1$ pièces de l'emplacement 1 à l'emplacement 2, puis la dernière pièce de l'emplacement 1 vers l'emplacement 3, et enfin les $n-1$ pièces de l'emplacement 2 à l'emplacement 3, écrire une fonction récursive `hanoi n départ intermédiaire arrivée` qui résoud le problème.

2 Problème : systèmes monétaires et porte-monnaie

Par convention, les listes manipulées dans ce problème seront en ordre décroissant et ne comprendront que des entiers strictements positifs.

Un *système monétaire* est une liste de *dénominations* distinctes $D = \langle d_1, d_2, ..., d_n \rangle$ avec $n > 0$ et $d_n = 1$. On posera par exemple en Caml :

```
type système == int list   et   let euros = [500; 200; 100; 50; 20; 1; 5; 2; 1]
```

Une *somme* (`type somme == int list`) est une suite finie d'entiers $S = \langle e_1, e_2, ..., e_m \rangle$ appartenant à D. Les éléments de cette suite sont les *espèces*. On définit de manière naturelle la *valeur* et la *taille* d'une somme. Dans le système européen, un exemple de somme est `let s1 = [20; 10; 10; 10; 2]`. Sa valeur est 62 et sa taille 5. On dit aussi que `s1` est un *représentant* de 5.

Certaines sommes sont des *portefeuilles* et représentent les espèces dont dispose un citoyen pour faire ses achats. On pourra alors dire que la somme `[10; 2]` est *extraite* de `s1`. Dans tout le problème, il s'agit de trouver comment payer exactement son achat sans rendu de monnaie, c'est à dire de trouver un représentant d'un prix p donné extrait d'un portefeuille donné.

h. Écrire une fonction `valeur` qui prend une somme en argument et renvoie sa valeur.

2.1 Payer gloutonnement le compte exact

Étant donné un prix p, on cherche à trouver un représentant de p en utilisant las stratégie gloutonne suivante :

– on commence par donner l'espèce d la plus élevée possible inférieure ou égale à p,
– et on recommence avec le prix $p - d$.

Évidemment, le processus s'arrête lorsque le prix est entièrement payé.

i. On suppose que l'acheteur dispose toujours de toutes les espèces du système en quantité illimitée. Écrire une fonction `glouton : système -> int -> somme` qui réalise la stratégie gloutonne. Réussit-elle toujours ?

j. On tient cette fois compte du portefeuille de l'acheteur. Écrire la fonction `paye_glouton : somme -> int -> somme`. Réussit-elle toujours ?

2.2 Payer le compte exact de toutes les manières possibles

k. Écrire une fonction `compte_paiements : somme -> int -> int` qui

compte le nombre de manières différentes de payer un prix p avec un portefeuille donné. Toutes les espèces seront distinguées : ainsi `compte_paiements [2;2;2] 4` doit valoir `3`.

2.3 Payer le compte exact et optimal

Une somme est *optimale* lorsque sa taille est minimale parmi un ensemble de sommes de valeur donnée. Par exemple, parmi les sommes de valeur 26 euros, la somme `[20; 5; 1]` est optimale, ce qui n'est pas le cas de $\langle 10, 10, 5, 1\rangle$. Dans le système européen, la solution gloutonne est en fait toujours optimale : on dit que le système est *canonique*.

l. Montrer que l'ancien système britannique $\langle 60, 30, 24, 12, 6, 3, 1\rangle$ n'est pas canonique.

La suite de cette partie se propose de coder une fonction trouvant une solution optimale $M(p)$ à tout prix p même si le système n'est pas canonique. On peut tout d'abord penser à utiliser un mécanisme similaire à la fonction `compte_paiements`, ce qui conduit à des temps de calcul exponentiels à chaque interrogation.

Nous allons ici étudier une méthode qui permet (comme dans la question **b**) de se souvenir de toutes les valeurs M_p lorsque p parcourt un intervalle $[0; p_{\max}]$ fixé en construisant au fur et à mesure la fonction $M : p \to M(p)$.

Étant donné un portefeuille, on note $T(n)$ l'ensemble des prix dont la solution optimale est de longueur n. Les liste $T(n)$ ne seront pas nécessairement triées. Ainsi, avec le portefeuille $\langle 10, 10, 10, 5, 1\rangle$, on a :

$$\begin{array}{rcl} T(0) & = & \{0\} \\ T(1) & = & \{1, 5, 10\} \\ T(2) & = & \{6, 11, 15, 20\} \end{array}$$

On note enfin M_i les restrictions successives de M aux différents $\cup_{j\leq i} T(j)$. Pour notre exemple, ces restrictions valent :

$$M_0 : \left| \begin{array}{l} 0 \longmapsto \langle\rangle \end{array} \right. \qquad M_1 : \left| \begin{array}{l} 0 \longmapsto \langle\rangle \\ 1 \longmapsto \langle 1\rangle \\ 5 \longmapsto \langle 5\rangle \\ 10 \longmapsto \langle 10\rangle \end{array} \right. \qquad M_2 : \left| \begin{array}{l} 0 \longmapsto \langle\rangle \\ 1 \longmapsto \langle 1\rangle \\ 5 \longmapsto \langle 5\rangle \\ 6 \longmapsto \langle 5, 1\rangle \\ 10 \longmapsto \langle 10\rangle \\ 11 \longmapsto \langle 10, 1\rangle \\ 15 \longmapsto \langle 10, 5\rangle \\ 20 \longmapsto \langle 10, 10\rangle \end{array} \right.$$

On suppose que le tableau M_i est représenté sous la forme d'une variable globale de taille suffisante `tab : somme vect`. Plus précisément, `tab.[n]` vaut $M(n) = M_i(n)$ si $M_i(n)$ est défini (c'est à dire si $n \in \cup_{j\leq i} T(j)$) et la liste vide sinon.

m. Écrire la fonction `étape : somme -> int -> int list -> int list` telle que, si `t` représente $T(i)$, `étape portefeuille pmax t` renvoie une liste représentant $T(i+1)$. On supposera en outre que `tab` encode M_i avant l'appel de `étape` et on demande que `tab` encode M_{i+1} au retour de `étape`.

n. En déduire la fonction `precalcul_optimal portefeuille pmax` qui remplit entièrement le tableau `tab` (on fera attention à ne pas dépasser la taille "suffisante" de ce tableau qui sera en fait $p_{\max} + 1$). Terminer en écrivant une fonction `paye_optimal p` qui renvoie une solution optimale pour un prix $p \leq p_{\max}$.

o.* Adapter les dernières fonctions pour que `tab` ne soit plus une variable globale.

Sources

- Exercices : sources diverses
- Problème : en gros les deux premières parties de X 2002 (qui en comptait trois, la dernière étant moins "programmatoire"). Remarquons au passage que le sujet de Centrale 2002 traitait le même thème.