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

Rendu de monnaie

D'après la composition d'Informatique du concours *Centrale-Supélec* 2002

Le sujet traite du problème du monnayeur : comment rendre la monnaie en utilisant le plus petit nombre possible de pièces ? Les deux premières parties mettent en place le formalisme et les outils qui serviront pour la suite. On étudiera dans la partie 3 « l'algorithme glouton ». La dernière partie présente un algorithme permettant de décider si l'algorithme glouton est optimal pour un système de pièces donné. Cette dernière partie utilise les résultats établis dans les parties précédentes.

Notations

Lorsque E est un ensemble fini, $|E|$ désigne son cardinal.
Lorsque $x \in \mathbb{R}$, $\lfloor x \rfloor$ désigne la partie entière inférieure de x et $\lceil x \rceil$ sa partie entière supérieure : ce sont les uniques entiers relatifs vérifiant $\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1$ et $\lceil x \rceil - 1 < x \leq \lceil x \rceil$.
Si $p, q \in \mathbb{Z}$ avec $p \leq q$, $[\![p, q]\!]$ désigne l'ensemble des entiers relatifs compris entre p et q : $[\![p, q]\!] = [p, q] \cap \mathbb{Z}$.
Si $m \in \mathbb{N}^* = \mathbb{N}\backslash\{0\}$, $\mathbb{R}^m$ est muni de sa structure euclidienne canonique en posant, pour $u, v \in \mathbb{R}^m$:

$$u \cdot v = \sum_{i=1}^{m} u_i v_i$$

Si $1 \leq i \leq m$, e_i désigne le m-uplet dont toutes les composantes sont nulles sauf la i-ème qui vaut 1.

Formalisation du problème

Soit $(c_i)_{1 \leq i \leq m}$ un m-uplet d'entiers vérifiant $c_1 > c_2 > \ldots > c_m = 1$. Nous utiliserons le terme *système* pour désigner un tel m-uplet. Les c_i sont les valeurs faciales des pièces en service. Par exemple, le système utilisé en zone Euro est $(200, 100, 50, 20, 10, 5, 2, 1)$.
Pour chaque $i \in [\![1, m]\!]$, nous disposons d'une quantité illimitée de pièces de valeur (c_i). Soit x un entier (le montant à rendre). Une représentation de x dans le système c est un m-uplet $k = (k_1, \ldots, k_m)$ vérifiant :

$$x = \sum_{i=1}^{m} k_i c_i = k \cdot c$$

k_i est donc le nombre de pièces (c_i) qui seront rendues. Pour épargner les poches des clients, nous souhaitons minimiser le *poids* de cette représentation, c'est-à-dire la quantité :

$$\|k\| = \sum_{i=1}^{m} k_i = k \cdot \mathbf{1}$$

avec $\mathbf{1} = (1, \ldots, 1)$ (le m-uplet dont toutes les composantes sont égales à 1).

1 Systèmes de pièces

Nous utiliserons des listes d'entiers pour représenter aussi bien un système (liste de valeurs de pièces) qu'une représentation d'un montant dans ce système. Par exemple, la liste `[4; 1; 3]` est une représentation de 30 dans le système $(6, 3, 1)$.

▶ **Question 1** *Rédiger en Caml une fonction* `est_un_systeme` *telle que* `est_un_systeme c` *rende un booléen indiquant si la liste* `c` *est bien un système.*

`value est_un_systeme` : *int list* → *bool*

Par exemple, `est_un_systeme [5; 2; 1]` rendra la valeur `true`, tandis que `est_un_system [5; 7; 1]` rendra la valeur `false`, tout comme le fera `est_un_systeme [7; 5; 2]`.

2 Représentations de poids minimal

Soient $c = (c_1, \ldots, c_m)$ un système et $x \in \mathbb{N}$. Nous notons $M_c(x)$ (ou même $M(x)$ lorsqu'aucune ambiguïté n'est à craindre) le plus petit nombre de pièces nécessaires pour représenter x dans le système c :

$$M(x) = \min\{\|k\| \mid k \in \mathbb{N}^m \text{ et } k \cdot c = x\}$$

Nous nous intéresserons aux représentations de poids minimal de x : ce sont les représentations k telles que $\|k\| = M(x)$. Pour alléger la rédaction, nous parlerons simplement de *représentations minimales.*

▶ **Question 2** *Démontrer que* $\lceil x/c_1 \rceil \leq M(x) \leq x$.

▶ **Question 3** *Montrer que* $M(x) \leq 1 + M(x - c_j)$ *pour tout indice* j *tel que* $c_j \leq x$. *Montrer que* $M(x) = 1 + M(x - c_j)$ *si et seulement si il existe une représentation minimale* k *de* x *faisant intervenir* c_j *(c'est-à-dire telle que* $k_j > 0$*).*

▶ **Question 4** *Notons* s *le plus petit indice* i *tel que* $c_i \leq x$. *Justifier l'égalité*

$$M(x) = 1 + \min_{s \leq i \leq m} M(x - c_i)$$

▶ **Question 5** *Déduisez-en une fonction Caml* poids_minimaux *telle que* poids_minimaux x c *construit la liste des valeurs de* $M_c(y)$ *pour* $1 \leq y \leq x$ *dans l'ordre des* y *croissants.*

value poids_minimaux : *int → int list → int list*

Indication : vous pourrez commencer par construire le résultat sous forme d'un tableau, puis le convertir en liste en utilisant la fonction list_of_vect de la bibliothèque standard.

3 L'algorithme glouton

Avertissement : *Dans cette partie, on travaillera obligatoirement sur des listes sans passer par des vecteurs.*

L'algorithme glouton pour rendre une somme $x > 0$ consiste à choisir le plus grand $c_i \leq x$, puis à rendre récursivement $x - c_i$. Par exemple, avec le système $c = (10, 5, 2, 1)$, l'algorithme décomposera 27 en 10+10+5+2. Avec le formalisme proposé, la solution fournie par l'algorithme glouton est donc $k = (2, 1, 1, 0)$. Le fonctionnement de l'algorithme glouton peut être accéléré par la remarque suivante : notant $q = \lfloor x/c_1 \rfloor$, cet algorithme rend q pièces de valeur c_1, puis rend le montant $x - qc_1$ en utilisant le système $(c_2, \ldots, c_m)$.

▶ **Question 6** *Rédiger en Caml une fonction* glouton *telle que* glouton x c *construit la représentation de* x *dans le système* c *en utilisant l'algorithme glouton.*

value glouton : *int → int list → int list*

Soient $c = (c_1, \ldots, c_m)$ un système et $x \in \mathbb{N}$. Nous noterons $\Gamma_c(x)$ la représentation gloutonne de x dans le système c, (c'est-à-dire la représentation obtenue en appliquant l'algorithme glouton), et $G_c(x)$ (ou même $G(x)$ lorsqu'aucune ambiguïté n'est à craindre) le nombre de pièces utilisées par l'algorithme glouton : $G_c(x) = \|\Gamma_c(x)\|$.
Nous dirons qu'un système c est *canonique* lorsque l'algorithme glouton nous donne toujours une représentation minimale ; on a alors $M_c(x) = G_c(x)$ pour tout $x \in \mathbb{N}$.

▶ **Question 7** *Montrer que tout système* $c = (c_1, c_2)$ *est canonique. Exhiber un système* $c = (c_1, c_2, c_3)$ *non canonique.*

4 L'algorithme de Kozen et Zaks

Nous allons voir ici un algorithme efficace permettant de déterminer si un système $c = (c_1, \ldots, c_m)$ est canonique. Nous dirons qu'un entier x est un *contre-exemple* pour c si $M(x) < G_c(x)$. Un système canonique n'admet donc pas de contre-exemple.
Dans la suite, on suppose $m \geq 3$.

▶ **Question 8** *Soit* c *un système non canonique. Il admet donc des contre-exemples. Montrer que le plus petit d'entre-eux, disons* x*, vérifie* $c_{m-2} + 1 < x < c_1 + c_2$.

Ce résultat nous donne un algorithme déterminant si un système est canonique : il suffit de rechercher un contre-exemple dans l'intervalle discret $[\![c_{m-2} + 2, c_1 + c_2 - 1]\!]$. Ceci nécessiterait la construction (coûteuse) des représentations minimales de chacun des éléments de cet intervalle.
Nous allons étudier un algorithme plus efficace, dû à Kozen et Zaks. Leur méthode repose sur la notion de *témoin*. Nous dirons qu'un entier x est un témoin pour le système $c = (c_1, \ldots, c_m)$ s'il existe un indice i tel que $c_i < x$ et $G(x - c_i) < G(x) - 1$.

▶ **Question 9** *Démontrer que tout témoin est un contre-exemple. Montrer que le résultat précédent n'admet pas de réciproque. (On pourra considérer le système* $(5, 4, 1)$*.)*

▶ **Question 10** *Montrer que si le système* c *admet des contre-exemples, alors le plus petit d'entre eux est un témoin.*

Il résulte de cette étude que, pour savoir si un système est canonique, il sufit de vérifier l'inégalité $G(x) \leq G(x - c_i) + 1$ pour tout $x \in [\![c_{m-2} + 2, c_1 + c_2 - 1]\!]$, et tout indice $i \in [\![1, m]\!]$ tel que $c_i < x$. C'est le principe de l'algorithme de Kozen et Zaks.

▶ **Question 11** *Rédiger en Caml une fonction* kozen_zaks *prenant pour argument un système et indiquant si celui-ci est canonique.*

value kozen_zaks : *int list → bool*

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

Rendu de monnaie

Un corrigé

▶ **Question 1**

```
let rec est_un_systeme = function
    [] -> false
  | [1] -> true
  | [_] -> false
  | c_1 :: c_2 :: c' ->
     c_1 > c_2 && (est_un_systeme (c_2 :: c'))
;;
```

▶ **Question 2** *Montrons tout d'abord que* $\lceil x/c_1 \rceil \leq M(x)$. Soit k une représentation pour laquelle le minimum (dans la définition de $M(x)$) est atteint. On a $M(x) = \|k\|$ et $k \cdot c = x$. Par définition $k \cdot c = \sum_i k_i c_i$ et, pour tout $i \in [\![1, m]\!]$, $c_i \leq c_1$. On en déduit que $x \leq \sum_i k_i c_1 = c_1 \sum_i k_i = c_1 \|k\|$. Il vient $x/c_1 \leq M(x)$ et, $M(x)$ étant un entier, $\lceil x/c_1 \rceil \leq M(x)$.

Vérifions maintenant que $M(x) \leq x$. Posons $c = e_m$. Soit k défini par $k_m = x$ et, pour tout $i \in [\![1, m-1]\!]$, $k_i = 0$. On a $k \cdot c = x$ et $\|k\| = x$. On en déduit que $M(x) \leq x$.

▶ **Question 3** Soit k' une représentation minimale de $x - c_j$. $k = k' + e_j$ est une représentation de x. On en déduit que $M(x) \leq \|k' + e_j\| = M(x - c_j) + 1$.
Supposons $M(x) = 1 + M(x - c_j)$ et considérons toujours une représentation minimale k' de $x - c_j$. On a $\|k' + e_j\| = M(x)$. $k = k' + e_j$ est donc une représentation minimale de x qui fait intervenir c_j.
Inversement, si k est une représentation minimale de x telle que $k_j \geq 1$, $k' = k - e_j$ est une représentation de $x - c_j$. On en déduit que $M(x - c_j) \leq \|k'\| = \|k\| - 1 = M(x) - 1$. On conclut que $M(x - c_j) + 1 \leq M(x)$ et grâce à la première inégalité on obtient $M(x) = M(x - c_j) + 1$.

▶ **Question 4** La question 3 donne immédiatement

$$M(x) \leq 1 + \min_{x \leq i \leq m} M(x - c_i)$$

Considérons k une représentation minimale de x et j tel que $k_j \geq 1$. La question 3 nous assure que $M(x) = 1 + M(x - c_j)$. On en déduit l'égalité.

▶ **Question 5**

```
let poids_minimaux x c =
  let m = make_vect x 1 in
  let rec mini y = function
      [] -> failwith "systeme vide"
    | c_1 :: [] -> m.(y - c_1 - 1)
    | c_1 :: c' when c_1 > y -> mini y c'
    | c_1 :: _ when c_1 = y -> 0
    | c_1 :: c' -> min m.(y - c_1 - 1) (mini y c')
  in
  for y = 2 to x do
    m.(y - 1) <- 1 + mini y c
  done;
  list_of_vect m
;;
```

▶ **Question 6**

```
let rec glouton x = function
    [] -> []
  | c_1 :: c' ->
    let q = x / c_1 in
    q :: glouton (x - q * c_1) c'
;;
```

▶ **Question 7** Montrons tout d'abord que tout système à deux éléments $c = (c_1, 1)$ est canonique. Soit $x \in \mathbb{N}$. Effectuons la division euclidienne de x par c_1 : $x = qc_1 + r$ avec $r < c_1$. L'algorithme glouton propose $k = (q, r)$ comme représentation minimale. Soit $k' = (q', r')$ une autre représentation de x dans le système c. On a nécessairement $q' < q$ (sinon $k = k'$, puisque $q'c_1 \leq x$. On a alors $r' - r = (q - q')c_1 \geq 1 \times 2$. On en déduit que $\|k'\| > \|k\|$.
On vérifie que le système $(4, 3, 1)$ n'est pas canonique : pour $x = 6$, l'algorithme glouton propose de rendre 3 pièces, $(1, 0, 2)$, alors qu'il est possible de n'en rendre que 2 : $(0, 2, 0)$.

▶ **Question 8** Considérons x le contre-exemple minimal de c.
Montrons tout d'abord que $c_{m-2} + 1 < x$. Si $x < c_{m-2}$ alors $x \leq c_{m-1}$. Toute décomposition de x ne fait donc

intervenir que c_{m-1} et c_m. On en déduit, par la question 7, que x n'est pas un contre-exemple (puisque l'algorithme glouton est optimal sur tout système à deux éléments).
Pour $x = c_{m-2}$, l'algorithme glouton propose de rendre une pièce, ce qui est optimal. De même, pour $x = c_{m-2} + 1$, l'algorithme propose 1 pièce si $x = c_{m-3}$ et 2 pièces sinon, ce qui est optimal également. On en déduit que tout contre-exemple pour le système c est strictement supérieur à $c_{m-2} + 1$.

Montrons maintenant que $x < c_1 + c_2$. Raisonnons par l'absurde et supposons $x \geq c_1 + c_2$. On a $G(x) > M(x)$ et $G(x) = 1 + G(x - c_1)$. x étant un contre exemple minimal, on a $G(x - c_1) = M(x - c_1)$. Soit j tel que c_j apparaisse dans une décomposition minimale de x. $M(x) = 1 + M(x - c_j) = 1 + G(x - c_j)$. On en déduit que $M(x-c_1) > M(x-c_j)$. On a $M(x) < 1+M(x-c_1)$ donc, par la question 3, on sait qu'il n'existe pas de décomposition minimale de x faisant intervenir c_1. On en déduit que $c_j \leq c_2$ et $x - c_j \geq x_1$. Or x est le contre-exemple minimal donc $x - c_j$ n'est pas un contre-exemple. L'algorithme glouton donne donc une décomposition minimale de $x-c_j$, qui fait intervenir c_1. On en déduit qu'il existe une décomposition minimale de x faisant intervenir c_1, d'où la contradiction.

▶ **Question 9** Soit x un témoin pour le système c est i tel que $c_i < x$ et $G(x-c_i) < G(x) - 1$. $G(x - c_i)$ est le poids d'une décomposition de $x - c_i$. On en déduit que $M(x - c_i) \leq G(x - c_i)$. Or $M(x) \leq M(x - c_i) + 1$ donc $M(x) < G(x)$. x est donc un contre-exemple.

▶ **Question 10** Soit x un contre-exemple minimal de c. Supposons que x n'est pas un témoin. Soit i tel que $c_i < x$. On a $G(x - c_i) \geq G(x) - 1$. Or x est un contre-exemple minimal, donc $G(x - c_i) = M(x - c_i)$. On en déduit que $M(x - c_i) + 1 \geq G(x) > M(x)$. Il n'existe pas de décomposition minimale de x faisant intervenir c_i. On en déduit que $x = c_j$, puis que $G(x) = 1 = M(x)$, d'où la contradiction.

▶ **Question 11**

```
let kozen_zaks c =
  match c with
    [] | [_] | [_; _] -> true
  | _ ->
    let v = vect_of_list c in
    let m = vect_length v in
    let g = make_vect (v.(0) + v.(1)) 0 in

    let j = ref (m - 1) in
    for x = 1 to v.(0) + v.(1) - 1 do
      while !j >= 0 && v.(!j) <= x do j := !j - 1 done;
      g.(x) <- 1 + g.(x - v.(!j + 1))
    done;

    let res = ref true in
    for x = v.(m - 3) + 2 to v.(0) + v.(1) - 1 do
      for i = 0 to m - 1 do
        if v.(i) < x && g.(x) > g.(x - v.(i)) + 1
        then res := false
      done;
    done;
    ! res
;;
```