ÉCOLE POLYTECHNIQUE **FILIÈRE MP**
OPTION INFORMATIQUE

CONCOURS D'ADMISSION 2002

COMPOSITION D'INFORMATIQUE

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

⋆⋆⋆

Dans tout le problème un *système monétaire* est un ensemble de n entiers naturels non-nuls distincts $D = \{d_1, d_2, \ldots, d_n\}$, avec $n > 0$ et $d_n = 1$. Les d_i sont les *dénominations* du système. Par convention, les dénominations sont présentées par ordre *décroissant*. Par exemple, le système de l'euro est l'ensemble $\{500, 200, 100, 50, 20, 10, 5, 2, 1\}$.

Une *somme* (d'argent) est une suite finie $\langle e_1, e_2, \ldots, e_m\rangle$ d'entiers appartenant à D. Les éléments de cette suite sont des *espèces*. On remarquera bien que deux espèces d'une somme peuvent porter la même dénomination, qu'une somme peut être vide, et qu'il n'y a pas nécessairement d'espèces de dénomination 1 dans une somme. Par convention, les espèces sont présentées par ordre de dénominations décroissantes.

La *valeur* d'une somme S, notée $\mathcal{V}(S)$, est tout simplement la somme arithmétique de ses espèces, tandis que sa *taille*, notée $|S|$, est le nombre de ses éléments (l'entier m ci-dessus). Étant donné un entier naturel v, une somme de valeur v est un *représentant* de v. Par exemple, le portefeuille d'un citoyen européen peut contenir 3 billets de 10 euros, deux billets de 5 euros et une pièce de 1 euro. Cette somme est notée $\langle 10, 10, 10, 5, 5, 1\rangle$, sa valeur est 41 et sa taille 6.

Une somme S est *extraite* d'une autre P dite portefeuille, si et seulement si S est une suite extraite de P. Intuitivement, « la somme S est extraite de P » signifie que l'on paye sa valeur à l'aide d'espèces prises dans le portefeuille P. Par exemple, notre citoyen européen peut payer exactement 15 euros en extrayant un billet de 10 euros et un autre de 5 euros de son portefeuille.

Programmation. Dans les deux premières parties du problème, les systèmes monétaires et les sommes sont représentés en machine par des listes d'entiers.

(Caml *)*

```
type systeme == int list ;;
type somme   == int list ;;
```

{ Pascal }

```
type
  liste = ^cellule ;
  cellule = record
    contenu : integer ; suivant : liste ;
  end ;
  systeme = liste ;
  somme = liste ;
```

En outre, on supposera (pour les arguments) et on garantira (pour les résultats) les propriétés suivantes :

- tous les entiers présents dans les listes sont strictement positifs ;
- toutes les listes sont triées en ordre décroissant ;

– les listes de type `systeme` contiennent des entiers distincts ; leur dernier élément est 1.

En Pascal, la liste vide est `nil` et l'on pourra pour construire les listes utiliser la fonction suivante :

```
function cons(contenu : integer ; suivant : liste) : liste ;
var r : liste ;
begin
   new(r) ;
   r^.contenu := contenu ;
   r^.suivant := suivant ;
   cons := r
end ;
```

Cette fonction est applicable pour construire les listes des deux types `somme` et `systeme`.

Question 1. Écrire la fonction `valeur` qui prend une somme en argument et renvoie sa valeur.

(* *Caml* *)	{ *Pascal* }
`valeur : somme -> int`	`function valeur(s:somme) : integer ;`

I. Payer le compte exact.

Dans cette partie on considère le problème du paiement exact. Dans les termes du préambule, cela revient, étant donné un entier naturel p, à trouver un représentant de p.

Une démarche possible pour payer exactement le prix p est la démarche dite gloutonne, que l'on peut décrire informellement ainsi :

– donner l'espèce la plus élevée possible — c'est à dire de la plus grande dénomination d disponible et telle que $d \leqslant p$;

– recommencer en enlevant l'espèce donnée au prix à payer — c'est dire poser p égal à $p - d$.

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

Question 2. Dans cette question, on suppose que l'acheteur dispose toujours des espèces dont il a besoin.

a) Montrer que la démarche gloutonne réussit toujours.

b) Écrire une fonction `glouton` qui prend en argument un système monétaire `sys` et un prix à payer `p` et renvoie la liste des espèces à utiliser pour payer. La sommme renvoyée sera calculée en suivant la démarche gloutonne.

(* *Caml* *)	{ *Pascal* }
`glouton : systeme -> int -> somme`	`function glouton` `  (sys:systeme ; p:integer):somme ;`

Question 3. On tient cette fois compte des ressources de l'acheteur. Dans les termes du préambule cela revient à trouver une somme ayant pour valeur le prix à payer et extraite d'une somme donnée, dite portefeuille.

a) Montrer, à l'aide d'un exemple utilisant le système européen, que la stratégie gloutonne peut échouer pour un prix donné, même si il est possible de payer compte tenu des ressources disponibles.

b) Écrire une fonction `paye_glouton` qui prend en argument une somme `pf`, représentant le contenu du portefeuille, ainsi qu'un prix `p` ; et qui renvoie une somme extraite de `pf` et dont la valeur est `p`. La somme renvoyée sera calculée en suivant la démarche gloutonne, si cette démarche échoue, la fonction `paye_glouton` doit renvoyer la liste vide.

(* *Caml* *)	{ *Pascal* }
`paye_glouton : somme -> int -> somme`	`function paye_glouton (pf:somme ; p:integer):somme ;`

Question 4. Écrire une fonction `compte_paiements` qui prend en arguments un portefeuille `pf` et un prix `p` ; et qui renvoie le nombre de façons de payer `p` à l'aide des espèces de `pf`. Autrement dit cette fonction renvoie le cardinal de l'ensemble des représentants de `p` extraits de `pf`.

(* *Caml* *)	{ *Pascal* }
`compte_paiements : somme -> int -> int`	`function compte_paiements (pf:somme ; p:integer):integer ;`

Remarque. Deux espèces de même dénomination sont distinguées. Ainsi, si le portefeuille est $\langle 2, 2, 2\rangle$ et le prix 2, alors il y a trois façons de payer.

II. 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, la somme $S = \langle 20, 20, 1\rangle$ montre que le portefeuille de notre citoyen européen ($P = \langle 10, 10, 10, 5, 5, 1\rangle$) n'est pas optimal parmi les sommes de valeur 41. En effet, la taille de S est 3 et elle est strictement inférieure à la taille de P.

Dans cette partie un portefeuille P est fixé. Pour tout entier naturel p, on pose $M(p)$ égal à un représentant de p optimal parmi les représentants de p extraits de P, si une telle somme existe ; ou la somme vide, si une telle somme n'existe pas. Le but de cette partie est la détermination de $M(p)$.

Question 5. Le but de cette question préalable est de préciser quelques opérations sur les sommes.

a) Soit une somme S comprenant k espèces de dénomination d, on définit l'ajout de d à S, noté $S + \langle d\rangle$, comme la somme qui comprend $k + 1$ espèces de dénomination d et qui est inchangée autrement. Écrire la fonction `ajoute` qui prend en arguments une somme `s` et une dénomination `d` et qui renvoie la liste représentant l'ajout de `d` à `s`.

(* *Caml* *)	{ *Pascal* }
`ajoute : somme -> int -> somme`	`function ajoute (s:somme ; d:integer):somme ;`

b) On définit la différence de deux sommes S et S', notée $S - S'$, comme suit. Pour toute dénomination d, soit k le nombre d'espèces de dénomination d comprises dans S et k' le nombre d'espèces de dénominations d comprises dans S' :

- si $k > k'$, alors $S - S'$ comprend $k - k'$ espèces de dénomination d ;
- sinon, $S - S'$ ne comprend aucune espèce de dénomination d.

Écrire la fonction `diff` qui prend deux sommes en arguments et renvoie leur différence.

(* *Caml* *)	{ *Pascal* }
`diff : somme -> somme -> somme`	`function diff(s,t:somme):somme ;`

Question 6. Soit i un entier naturel. On note $T(i)$ l'ensemble des entiers naturels p, tels que la somme $M(p)$ est de taille i. On définit également une suite de fonctions M_0, M_1, ... où la fonction M_i est la restriction à $\bigcup_{0 \leqslant j \leqslant i} T(j)$ de la fonction M.

Déterminer $T(i)$ et M_i dans le cas du portefeuille européen $\langle 10, 10, 10, 5, 5, 1\rangle$ et pour i égal à 0, 1 et 2.

Question 7. On suppose donné un tableau global de sommes, `tab`, de taille suffisante (cette condition est précisée par la suite) et dont chaque case vaut initialement la liste vide.

(* *Caml* *)	{ *Pascal* }
`tab : somme vect`	`tab : array [0..QMAX] of somme`

Pour un entier i donné, on encode simultanément l'ensemble $T(i)$ et la fonction M_i de la façon suivante :

- une liste d'entiers représente $T(i)$;
- si $M_i(p)$ est défini, alors la case d'indice p de `tab` contient $M_i(p) = M(p)$. Sinon, la case d'indice p de `tab` n'existe pas ou contient la liste vide.

a) Écrire la fonction `etape` qui prend en arguments, un portefeuille `pf`, un prix à payer `p` et une liste d'entiers représentant l'ensemble $T(i)$, et qui renvoie une liste d'entiers représentant $T(i+1)$. On supposera en outre que le tableau `tab` encode M_i avant l'appel de la fonction `etape` et on demande que le tableau `tab` encode M_{i+1} au retour de la fonction `etape`.

(* *Caml* *)	{ *Pascal* }

`etape :`
`  somme -> int -> int list`
`  -> int list`
	`function etape`
	`  (pf:somme ; p:integer ; ti:liste):liste ;`

On notera :

- les listes représentant $T(i)$ et $T(i+1)$ ne sont pas nécessairement triées ;
- la condition « `tab` est de taille suffisante » est précisée : le tableau `tab` est supposé tel qu'il existe bien des cases d'indice q, pour q inférieur ou égal à `p`.

b) En déduire une fonction `optimal` qui prend en argument un portefeuille `pf` et un prix `p` et qui renvoie une somme optimale de valeur `p` et dont les espèces sont extraites de `pf`. Si il n'est pas possible de faire l'appoint, la fonction `optimal` renverra la liste vide.

(* *Caml* *)	{ *Pascal* }
`optimal : somme -> int -> somme`	`function optimal(pf:somme ; p:integer):somme ;`

III. Étude des systèmes monétaires.

Un système monétaire est dit *canonique*, lorsque la stratégie gloutonne sans limitation de ressources (cf. question 2) appliquée à tout prix p produit une somme optimale parmi les représentants de p.

Question 8. Montrer que l'ancien système britannique $\langle 240, 60, 30, 24, 12, 6, 3, 1\rangle$ n'est pas canonique.

Le but de cette partie est de produire un programme qui décide si un système monétaire est canonique. Dans cette étude on fixe un système monétaire D de n dénominations, représenté cette fois par le vecteur de n entiers naturels $D = (d_1, d_2, \ldots, d_n)$. Une somme S sera également représentée par un vecteur de n entiers naturels, $S = (s_1, s_2, \ldots, s_n)$, *mais s_i est cette fois le nombre d'espèces de dénomination d_i présentes dans la somme S.*

Ainsi le système européen est le vecteur $(500, 200, 100, 50, 20, 10, 5, 2, 1)$ et le portefeuille du citoyen est le vecteur $(0, 0, 0, 0, 0, 3, 2, 0, 1)$. Les définitions (et les notations) de la valeur et de la taille sont inchangées :

$$\mathcal{V}(S) = \sum_{i=1}^{n} s_i \cdot d_i, \quad |S| = \sum_{i=1}^{n} s_i$$

Par ailleurs les sommes sont ordonnées (totalement) selon l'ordre lexicographique, noté $<_\ell$ et défini ainsi : pour tous vecteurs U et V, on a $U <_\ell V$ si et seulement si :

1. il existe i, $1 \leqslant i \leqslant n$, tel que $u_i < v_i$;
2. et, pour tout j, $1 \leqslant j < i$, on a $u_j = v_j$.

Ainsi on a par exemple $(0,1) <_\ell (0,2)$ $(i = 2)$ et $(0,4) <_\ell (1,0)$ $(i = 1)$. On note $\leqslant_\ell$ la relation d'ordre définie par $U \leqslant_\ell V$, si et seulement si $U <_\ell V$ ou $U = V$.

On définit un second ordre total sur l'ensemble des sommes, noté $\sqsubseteq$, de la façon suivante :

$$U \sqsubseteq V, \quad \text{si et seulement si} \quad |U| > |V| \text{ ou } (|U| = |V| \text{ et } U \leqslant_\ell V)$$

Les relations d'ordre introduites permettent les *définitions* suivantes des représentants gloutons et optimaux (il n'est pas demandé de comparer ces nouvelles définitions aux anciennes). Étant donné un entier naturel p, le représentant glouton de p, noté $G(p)$ est le plus grand selon l'ordre lexicographique $\leqslant_\ell$ des représentants de p. Tandis que le représentant optimal de p, noté $M(p)$, est le plus grand selon l'ordre $\sqsubseteq$ des représentants de p.

Dès lors le système D est canonique, si et seulement si on a $G(p) = M(p)$ pour tout entier naturel p. En revanche, D n'est pas canonique, si et seulement si il existe un ou des entiers naturels w, dits *contre-exemples*, tels que $M(w) \neq G(w)$, c'est à dire tels que $M(w) <_\ell G(w)$.

PROGRAMMATION. Dans cette partie on considère l'entier n (nombre de dénominations du système monétaire) fixé. Les systèmes monétaires et les sommes sont représentées en machine par des tableaux d'entiers.

(Caml *)*

```
let n = ... (* n:int *) ;;
type tsysteme == int vect ;;
type tsomme == int vect ;;
```

{ Pascal }

```
const
  n = ... ;
type
  tsysteme = array [1..n] of integer ;
  tsomme   = array [1..n] of integer ;
```

(En Caml, on supposera ces tableaux de taille $n+1$, de sorte que les indices des tableaux correspondent à ceux des vecteurs.)

Question 9. Écrire la fonction `tglouton` qui prend en argument un système monétaire `sys` selon le nouveau type et un prix `p` et qui renvoie le représentant glouton de ce prix. Le coût de `tglouton` (c'est-à-dire le nombre maximum d'instructions élémentaires utilisées lors d'un appel à `tglouton`) doit être linéaire en n.

(Caml *)*

```
tglouton :
  tsysteme -> int -> tsomme
```

{ Pascal }

```
function tglouton
  (sys:tsysteme ; p:integer) : tsomme;
```

Question 10.

a) Montrer que $p < q$ entraîne $G(p) <_\ell G(q)$.

Soit k, indice, avec $1 \leqslant k \leqslant n$. On pose I_k égal à la somme composée d'une seule espèce de dénomination d_k. Soit encore p, entier naturel.

b) On suppose que $G(p)$ comprend au moins une espèce de dénomination d_k. Montrer qu'alors on a $G(p - d_k) = G(p) - I_k$. (Le deuxième « $-$ » ci-dessus est la différence des sommes, que l'on peut simplement voir comme la soustraction appliquée aux vecteurs et définie composante par composante.)

c) De même, si p est tel que $M(p)$ comprend au moins une espèce de dénomination d_k, montrer que l'on a alors $M(p - d_k) = M(p) - I_k$.

Question 11. Dans cette question on suppose le système monétaire D non-canonique et on considère le contre-exemple minimal w, c'est à dire l'entier w tel que :

- $M(w) \neq G(w)$ (et donc $M(w) <_\ell G(w)$) ;
- et, pour tout entier $w' > w$, $M(w') = G(w')$.

On note $M(w) = (m_1, m_2, \ldots, m_n)$. Soient encore i l'indice minimal tel que $m_i > 0$ et j l'indice maximal tel que $m_j > 0$.

a) On note $G(w) = (g_1, g_2, \ldots, g_n)$. Montrer qu'il n'existe pas d'indice k, tel que $m_k > 0$ et $g_k > 0$.

b) Montrer que l'on a $i > 1$.

c) Montrer que l'on a $d_{i-1} < w < d_{i-1} + d_j$.

Question 12. Toujours en supposant le système D non-canonique et en conservant les définitions et notations de la question précédente, on admet l'encadrement suivant (qui cette fois s'applique aux sommes) :

$$M(w) - I_j \leqslant_\ell G(d_{i-1} - 1) <_\ell M(w).$$

a) Montrer que cet encadrement permet de connaître les composantes de $M(w)$ en fonction de celles de $G(d_{i-1} - 1)$, en supposant i et j connus.

b) Écrire une fonction `canonique`, qui prend en argument un système monétaire `sys` et décide si ce système est canonique.

(Caml *)*	*{ Pascal }*
`canonique : tsysteme -> bool`	`function canonique(sys:tsysteme) : boolean`

Le coût de `canonique` doit être en $O(n^3)$.

c) Montrer que le système européen est canonique.

* *
*

Rapport de M. Cyril GAVOILLE, correcteur.

De l'épreuve

Rappelons que, si l'ensemble des candidats qui ont choisi cette option passent l'épreuve d'informatique, seules les copies des candidats admissibles sont corrigées. J'ai corrigé 203 copies dont 20 étaient rédigées en Pascal. L'épreuve s'est révélée d'une difficulté moyenne, la compréhension de l'épreuve ayant été bien maîtrisée par beaucoup de candidats. La moyenne est de 10,9 pour un écart type de 4,0. La répartition des notes est classique. Son résumé est le suivant :

$0 \leqslant N < 4$	$4 \leqslant N < 8$	$8 \leqslant N < 12$	$12 \leqslant N < 16$	$16 \leqslant N \leqslant 20$
4%	19%	39%	28%	10%

La réussite des candidats ayant composé en Pascal est légèrement inférieure à celle des candidats ayant composé en Caml, mais cette différence statistique n'est pas significative pour l'échantillon donné. Toutes les questions ont été résolues, mais aucun candidat n'a résolu parfaitement toutes les questions. Trois candidats ont tout de même obtenu la note de 20/20.

De l'évaluation

Quelques rappels. La concision d'un programme est un point important. Un candidat est pénalisé si le code de sa fonction s'étend sur un nombre anormalement élevé de lignes alors qu'une solution de quelques lignes aurait suffit (pour la question 5 pourtant simple, des solutions de plus d'une page ont été données). Cependant ne pas confondre concision avec lisibilité. La sanction est d'autant plus grande si le code est abandonné au correcteur sans explication de l'algorithme sous-jacent. Inversement, il est inutile de justifier plus que nécessaire une fonction dont l'algorithme est évident d'après l'énoncé (comme la question 1 par exemple). L'excès de justification qui se révèle inexacte la plupart du temps, n'est pas sanctionné. C'est une mauvaise gestion du temps. Un bon commentaire peut dans certains cas lever le doute sur un code entaché d'erreurs de syntaxe ou d'étourderie. On demande donc au candidat d'ajouter ses explications là où il les pense utiles pour le correcteur.

Les erreurs de syntaxe sur un code bien indenté ne sont que très faiblement sanctionnées. Dans certains cas cependant (fonction dont le code est trivial, comme à la question 1 par exemple), elles peuvent être sanctionnées plus lourdement et faire la différence entre des copies équivalentes. Voici quelques erreurs que j'ai relevées (liées à la syntaxe de Caml, les erreurs de syntaxe en Pascal étant moins fréquentes) : "`; else`", " `:()`" au lieu de "`[]`", code inutile comme "`| p when p=0 ->`" à la place de "`| p=0 ->`", "`!i= !i+1`" dans un "`let i=ref 0 in`" ce qui est une double erreur de syntaxe, "`let n+1=(vect_length sys)`" (pour la déclaration d'un tableau à la question 9), ... Notez que `T_i` est un nom de variable possible, pas `T_(i+1)`.

Dans le rapport détaillé qui suit, est indiqué, pour chaque question, le pourcentage des candidats qui ont obtenu plus de la moitié des points.

Question 1. [96%] La question ne comportait aucune difficulté.

Question 2. [69%] La question 2b) de programmation a été bien mieux réussie que la question 2a) qui était en fait la justification de l'algorithme. Alors que la majorité des candidats ont écrit un programme parfaitement correct similaire à :

```
let rec glouton sys p = match sys,p with
| sys,0 -> []
| d::r,p -> ...
```

très peu ont écrit une récurrence correcte sur le prix à payer p, récurrence qui devait commencer à $p = 0$. Au final, à la question 2a), seulement 31% des candidats ont obtenu le nombre de points maximum, ce qui est fort peu par rapport à la difficulté de la récurrence.

Question 3. [81%] Pour la question 3a), un portefeuille=[5,2,2,2] et un prix=6 fournissaient le contre-exemple demandé (d'autres étaient possibles bien évidemment). Bien qu'évidente, plus de 20% des candidats ont mal compris cette question. Les exemples du type portefeuille=500 et prix=1 ne sont en rien des contre-exemples, certains candidats ayant insisté sur le fait qu'avec un billet de 500 Euros on peut toujours acheter un bien de 1 Euro. La question 3b) était à peine plus difficile que la question 2b), excepté le fait qu'il fallait penser à retirer l'espèce du portefeuille et gérer les paiements impossibles (renvoyer `[]` dans ce cas).

Question 4. [67%] Il s'agissait ici de compter le nombre de suites extraites dont la valeur était un prix donné, p. Notez que la fonction précédente `paye_glouton` n'était d'aucun secours, puisqu'à la question 3a) nous avons vu qu'elle pouvait échouer. En fait, en utilisant une simple récurrence on pouvait en même temps générer une suite extraite et tester que sa valeur soit p (chaque espèce d du portefeuille étant choisie pour appartenir ou non à la suite extraite en distinguant le cas où $d \leqslant p$ et $d > p$). Certains candidats (peut être trompés par l'exemple du sujet ?) ont essayé de générer uniquement les suites extraites de valeur p avec un algorithme polynômial en la taille du portefeuille, en vain ! Rappelons que le nombre de façons de payer p avec le portefeuille comportant $2p$ espèces égales à 1, est $C_{2p}^{p} \sim 4^{p+o(p)}$.

Question 5. [87%] Cette question ne comportait aucune difficulté réelle, et la concision a été appréciée. La question 5b) pouvait être résolue avec un filtrage à 4 cas, et non pas 1 page de code découpée en 6 sous-fonctions comme dans certaines copies.

Question 6. [80%] La plupart des erreurs sont des erreurs d'étourderie. On avait $T(0) = \{0\}$, $T(1) = \{1, 5, 10\}$, et $T(2) = \{6, 11, 15, 20\}$. On en déduisait les listes $M_i(p)$.

Question 7. [49%] Cette question difficile a été finalement moyennement réussie. Les algorithmes étaient plus longs que pour les questions précédentes : deux boucles `while` (ou deux fonctions récursives) pour la question 7a). Plusieurs solutions étaient possibles, les solutions qui sortaient de "l'esprit" de la question, dont le code était souvent long, devaient être expliquées. Sinon elles ont été sanctionnées.

Question 8 et 9. [82%] Très peu de candidats n'ont pas réussi la question 8 (l'algorithme glouton donnait $\langle 30, 12, 6\rangle$ pour $p = 48$ au lieu du paiement optimal $\langle 24, 24\rangle$).

La question 9 était classique : une boucle comprenant une division Euclidienne du prix (avec calcul du quotient et mise à jour du reste). Cette question a aussi été bien réussie, les quelques erreurs ayant essentiellement porté sur la syntaxe, le dimensionnement du tableau

par exemple. Une fonction Pascal ne pouvant pas retourner de tableau, les candidats rédigeant en Pascal devaient lire, comme certains l'on fait remarquer :

```
function tglouton(sys :system ; p :integer ; var t :tsomme) ;
```

Question 10. [59%] Plusieurs preuves correctes ont été proposées. C'est la clarté et l'enchaînement logique des arguments qui ont été jugés, un texte mathématique allant de l'hypothèse à la conclusion ne fournissant pas toujours une preuve correcte.

Question 11a et 11b. [81%] Les deux premières sous-questions de la question 11 ont été de loin les mieux réussies. Tous les candidats qui ont abordé cette question ont remarqué et corrigé d'eux même l'erreur typographique "$w' > w$" : il fallait lire $w' < w$, ce qui était clair d'après le texte introductif. La question 11a) se montrait très facilement par contradiction en posant $w' = w - d_k$.

La question 11b) était aussi très simple. Exemple de preuve : si $m_1 \geqslant 1$, alors d'après 11a) $g_1 = 0$. Il suit que $M(w) >_\ell G(w)$ ce qui est une contradiction.

Question 11c. [14%] Pour cette sous question, la réussite a été nettement moins bonne. Pour montrer que $d_{i-1} < w$, de nombreux candidats ont utilisé l'argument (faux) suivant : « $w = m_i d_i + m_{i+1} d_{i+1} + ...$, avec $m_i > 0$; ainsi $w \geqslant m_i d_i \geqslant d_i > d_{i-1}$ ». La dernière inégalité est fausse, l'ordre décroissant des éléments montrant plutôt que $d_{i-1} > d_i$ et pas du tout $d_{i-1} < w$.

Très peu ont su montrer $w < d_{i-1} + d_j$. On pouvait écrire par exemple : $M(w - d_j) = M(w) - I_j$ car $m_j \geqslant 1$. Comme $w - d_j < w$, on a aussi $M(w - d_j) = G(w - d_j)$. Ainsi $G(w - d_j) = M(w) - I_j$ et donc $G(w - d_j) = (0, ..., 0, m_i, ..., m_j - 1, 0, ..., 0)$. Il suit que $w - d_j < d_{i-1}$ car sinon la composante $i - 1$ de $G(w - d_j)$ serait non nulle.

Question 12. [27%] La seule solution pour l'inégalité proposée en 12a) est de poser $M(w) = (0, ..., 0, g_i, ..., g_j + 1, 0...0)$ où $(g_1, ..., g_n) = G(d_{i-1} - 1)$. En effet, tout représentant glouton $S = (u_1, ..., u_n)$ tel que $A \leqslant_\ell S <_\ell B$ avec $A = (0...0, v_i, ..., v_j, 0...0)$ et $B = (0...0, v_i, ..., v_j + 1, 0...0)$, vérifie $u_k = v_k$ pour tout $i \leqslant k \leqslant j$.

L'algorithme qui en découle est relativement simple. Le seul point délicat, et qui a fait chuter plusieurs candidats ayant pourtant répondu de manière très satisfaisante à toutes les questions précédentes, était le test : $w = \nu(M)$ est-il un contre-exemple ? où M est le représentant construit à partir de $G(d_{i-1} - 1)$. Il est clair que si w est un contre-exemple, alors $M(w) = M$ et donc $M <_\ell G(w)$ et aussi $M \neq G(w)$. Bien que $M \neq G(w)$, ou bien $M <_\ell G(w)$, soit facilement testable (une seule instruction en Caml), cela n'implique pas que w soit un contre-exemple. Pour s'en convaincre, il suffit de considérer le système Européen avec $w = 5$ pour $i = 8$ et $j = 9$. On a $M = [2, 1]$ (car $G(5 - 1) = [2, 0]$). Mais $G(5) = [1, 0, 0]$, ce qui montre que $M \neq G(5)$. Et pourtant 5 est loin d'être un contre-exemple ! Le test correct devait être basé sur la comparaison des tailles de M et de $G(w)$, fonction dont le code était évident.

Certains candidats sont arrivés à bout des calculs de la question 12c) pour le système Européen. Ils pouvaient être simplifiés en utilisant l'inégalité de la question 11c) et/ou en montrant que les systèmes $\{5, 2, 1\}$ et $\{100, 10, 1\}$ étaient canoniques, le système Européen étant une combinaison de ces systèmes.