# Systèmes monétaires (d'après le concours Polytechnique)

La durée du devoir est de 3h, les calculatrices ne sont pas autorisées.

Les parties sont partiellement indépendantes, mais les définitions, en particulier dans l'introduction, valent pour l'intégralité du sujet. Toute fonction peut librement être réutilisée dans les questions suivantes, y compris les fonctions demandées par l'énoncé que vous n'êtes pas parvenus à écrire.

Les fonctions dont on demande une implémentation sont à écrire dans le langage OCaml. Toutes les fonctions de la bibliothèque standard sont autorisées. Cela inclue en particulier les fonctions de la forme `List.XXX` et `Array.XXX`, incluant (mais non limitées à) celles rappelées ci-dessous.

`List.length : 'a list -> int`

Renvoie le nombre d'éléments dans la liste fournie en argument. Complexité linéaire en la taille de la liste ($O(n)$).

`List.hd : 'a list -> 'a`

Renvoie le premier élément de la liste fournie en argument. Complexité constante ($O(1)$).

`List.tl : 'a list -> 'a`

Renvoie la liste en argument privée de son premier élément. Complexité constante ($O(1)$).

`List.rev : 'a list -> 'a list`

Retourne une nouvelle liste contenant les éléments de la liste fournie en argument en ordre inverse. Complexité linéaire en la taille de la liste ($O(n)$).

`List.init : int -> (int -> 'a) -> 'a list`

« `List.init n f` » crée la liste `[f 0; f 1; ...; f (n-1)]` à `n` éléments obtenus en appelant la fonction `f` successivement avec les entiers de `0` à `n-1` (dans cet ordre).

`List.mem : 'a -> 'a list -> bool`

« `List.mem x lst` » renvoie un booléen indiquant si au moins un élément de la liste `lst` est égal à `x`. Complexité dans le pire des cas linéaire en la taille de la liste ($O(n)$).

`List.iter : ('a -> unit) -> 'a list -> unit`

« `List.iter f lst` » exécute $f\,a_i$ successivement pour chacun des éléments $a_i$ de la liste `lst` (dans l'ordre).

`List.map : ('a -> 'b) -> 'a list -> 'b list`

« `List.map f lst` » renvoie la liste `[f`$a_0$`; f`$a_1$`; ...; f`$a_{n-1}$`]` où les $a_i$ sont les éléments de la liste `lst`. L'ordre d'évaluation des $f\,a_i$ n'est pas spécifiée.

`Array.length : 'a array -> int`

Renvoie la taille (longueur) du tableau passé en argument. Complexité constante ($O(1)$).

`Array.make : int -> 'a -> 'a array`

« `Array.make n elem` » construit et renvoie un tableau de taille `n` contenant `elem` dans chacune de ses cases. Complexité linéaire en `n` ($O(n)$).

`Array.copy : 'a array -> 'a array`

Renvoie un nouveau tableau, de même taille que celui passé en argument, et contenant les mêmes éléments. Complexité linéaire en la taille du tableau ($O(n)$).

`Array.map : ('a -> 'b) -> 'a array -> 'b array`

« `Array.map f arr` » renvoie un nouveau tableau dont les éléments $b_i$ sont obtenus en évaluant `f` $a_i$, où les $a_i$ sont les éléments du tableau passé en argument.

`Array.iter : ('a -> unit) -> 'a array -> unit`

« `Array.iter f arr` » exécute `f` $a_i$ successivement (dans l'ordre) sur chacun des éléments $a_i$ du tableau `arr`.

**Il est demandé d'attacher une attention toute particulière à la clarté et la lisibilité des fonctions.** On attend d'une fonction qu'elle soit juste, mais également qu'elle ne soit pas inutilement compliquée. Pour toute fonction de plus de quelques lignes, il convient d'en expliquer clairement le fonctionnement (ce que les noms désignent, ce que font les éventuelles fonctions auxilliaires et la signature de celles-ci, etc.). Il est recommandé de proposer des invariants de boucles chaque fois que cela fait sens.

**Un soin identique doit être apporté aux preuves demandées.** Aucune ne nécessite plus d'une dizaine de lignes, veillez à bien extraire les arguments importants et à faire ressortir les articulations logiques de votre raisonnement.

## 1 Introduction

Dans tout le problème un *système monétaire* est un ensemble de $n > 0$ entiers naturels non-nuls distincts $\mathcal{D} = \{d_0, d_1, \ldots, d_{n-1}\}$ vérifiant $\forall 0 \leqslant i < n-1,\ d_i > d_{i+1}$ et $d_{n-1} = 1$. Les $d_i$ sont les *dénominations* du système. 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 $[e_0; e_1; \ldots; e_{m-1}]$ de $m \geqslant 0$ entiers appartenant à $\mathcal{D}$, avec $\forall 0 \leqslant i < m-1,\ e_i \geqslant e_{i+1}$. Les éléments de cette suite sont des *espèces*. Notez 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.

La *valeur* d'une somme $\mathcal{S}$, notée $\mathcal{V}(\mathcal{S})$, est tout simplement la somme de ses espèces, tandis que sa taille, notée $|\mathcal{S}|$, 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 [10;10;10;5;5;1], sa valeur est 41 et sa taille 6.

Dans les sections 1 à 3 de ce problème, les systèmes monétaires et les sommes sont représentés en machine par des listes d'entiers. **Attention, cette représentation sera modifiée dans la section 4.**

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

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` sont non-vides, contiennent des entiers deux à deux distincts, et leur dernier élément est toujours 1.

**1.** Écrire la fonction `valeur`, de signature `somme -> int` (c'est-à-dire `int list -> int`) qui prend une somme $\mathcal{S}$ en argument et renvoie sa valeur $\mathcal{V}(\mathcal{S})$.

Une somme $\mathcal{S}$ est dite *extraite* d'une autre somme $\mathcal{P}f$ (dite portefeuille), si et seulement si $\mathcal{S}$ est une suite extraite de $\mathcal{P}f$. Intuitivement, « la somme $\mathcal{S}$ est extraite de $\mathcal{P}f$ » signifie que l'on paye sa valeur à l'aide d'espèces prises dans le portefeuille $\mathcal{P}f$. 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. Notez bien qu'un portefeuille est une somme.

**2.** Proposer une fonction `est_extraite`, de signature `somme -> somme -> bool` (c'est-à-dire `int list -> int list -> bool`) qui prend en argument une somme $\mathcal{S}$ et une somme $\mathcal{S}'$ et renvoie un booléen indiquant si $\mathcal{S}$ a pu être extraite de $\mathcal{S}'$.

# 2 Payer le compte exact

## 2.1 Cas de ressources infinies

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 $p$. Par exemple, pour $p = 42$, la somme [20;10;5;5;2], dont la valeur est 42, serait une solution possible pour le système de l'euro (parmi d'autres).

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.

**Dans un premier temps, on suppose que l'acheteur dispose toujours des espèces dont il a besoin** (il dispose d'une quantité infinie d'espèces pour chacune des dénominations du système). Pour $p = 42$ et le système de l'euro, on choisit donc dans un premier temps la dénomination 20 (la plus grande des dénominations du système inférieure ou égale à $p$), il reste alors $p = 22$. On choisit à nouveau la dénomination 20, et il reste $p = 2$. On choisit enfin la dénomination 2 et la somme est payée. Cela donne donc la somme [20;20;2].

**3.** Montrer soigneusement que dans ce cas, quelle que soit la somme $p \geqslant 0$, la démarche gloutonne réussit toujours.

**4.** Écrire une fonction `glouton` de signature `int -> systeme -> somme` (c'est-à-dire `int -> int list -> int list`) qui prend en argument un prix $p$ à payer et un système monétaire $\mathcal{D}$, et renvoie la somme (soit liste d'espèces) que l'on peut utiliser pour payer. La somme renvoyée sera calculée en suivant la démarche gloutonne.

## 2.2 Cas de ressources finies

**On tient dorénavant compte des ressources de l'acheteur.** Dans les termes du préambule, cela revient à essayer de trouver une somme $\mathcal{S}$ ayant pour valeur le prix à payer $p$ et extraite d'une somme donnée, dite portefeuille, et notée $\mathcal{P}f$.

**5.** Montrer, à l'aide d'un exemple utilisant le système européen, que la stratégie gloutonne peut échouer à trouver une somme convenable, pour un prix $p$ donné et un portefeuille donné $\mathcal{P}f$, même si il est possible de payer ce prix $p$ avec des espèces contenues dans le portefeuille $\mathcal{P}f$.

**6.** Écrire une fonction `paye_glouton` de signature `int -> somme -> somme` (donc `int -> int list -> int list`) qui prend en argument un prix $p$ et une somme représentant le contenu du portefeuille $\mathcal{P}f$, et qui renvoie, dans la mesure du possible, une somme extraite de $\mathcal{P}f$ 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. Par exemple,

```
# paye_glouton 42 [20; 10; 5; 5; 5; 2; 1];;
- : int list = [20; 10; 5; 5; 2]

# paye_glouton 42 [50; 20; 10; 5; 2; 2; 1];;
- : int list = []
```

**7.** Proposer une fonction `paye` de signature `int -> somme -> somme` (donc `int -> int list -> int list`) qui prend en argument un prix $p$ et une somme représentant le contenu du portefeuille $\mathcal{P}f$, et qui renvoie une somme extraite de $\mathcal{P}f$ et dont la valeur est $p$ si une telle somme existe, et une liste vide sinon. On ne suit pas ici la stratégie gloutonne, la fonction doit retourner une liste vide si, et seulement si, il n'existe aucune

somme extraite de $\mathcal{P}f$ de valeur $p$. *On demande ici une solution simple, pas nécessairement efficace en terme de complexité.*

**8.** Quelle est, dans le pire des cas, la complexité en temps de la fonction précédente ?

## 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, pour un portefeuille $\mathcal{P}f = [50;20;20;10;5;5;2]$, la somme $\mathcal{S} = [20;20;2]$ est optimale, mais la somme $\mathcal{S}' = [20;10;5;5;2]$ n'est pas optimale parmi les sommes de valeur 42 que l'on peut extraire de $\mathcal{P}f$.

Dans cette partie, un portefeuille $\mathcal{P}f$ est fixé, et on cherche une façon optimale de payer un prix $p$ à partir des espèces contenues dans le portefeuille.

Dans un premier temps, on établit quelques opérations sur les sommes, qui nous seront utiles dans la suite pour déterminer un représentant optimal.

Soit une somme $\mathcal{S}$ comprenant $k$ espèces de dénomination $d$ (et possiblement des espèces d'autres dénominations). On définit l'ajout d'une dénomination $d$ à $\mathcal{S}$, noté $[d]\oplus\mathcal{S}$, comme la somme qui comprend $k+1$ espèces de dénomination $d$ et qui est inchangée autrement.

**9.** Écrire la fonction `ajout` de signature `int -> somme -> somme`, qui prend en argument une dénomination $d$ et une somme $\mathcal{S}$ et qui renvoie la liste représentant la somme correspondant à l'ajout de $d$ à $\mathcal{S}$.

```
# ajout 10 [20; 10; 5; 5; 2];;
- : int list = [20; 10; 10; 5; 5; 2]
```

Soient deux sommes $\mathcal{S}$ et $\mathcal{S}'$. On définit la différence de $\mathcal{S}$ et $\mathcal{S}'$, notée $\mathcal{S}\ominus\mathcal{S}'$, comme suit. Pour toute dénomination $d$, soit $k$ le nombre d'espèces de dénomination $d$ comprises dans $\mathcal{S}$ et $k'$ le nombre d'espèces de dénominations $d$ comprises dans $\mathcal{S}'$ :

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

**10.** Écrire la fonction `diff` de signature `somme -> somme -> somme` (soit `int list -> int list -> int list`) qui prend deux sommes $\mathcal{S}$ puis $\mathcal{S}'$ en arguments et renvoie leur différence.

```
# diff [20; 10; 5; 5; 2] [10; 5];;
- : int list = [20; 5; 2]

# diff [20; 10; 5; 5; 2] [50; 10; 10; 2];;
- : int list = [20; 5; 5]
```

Soit un portefeuille $\mathcal{P}f$ et un entier naturel $i$. On note $\mathcal{T}(i)$ l'ensemble des entiers naturels $v$ tels que le ou les représentants optimaux de $v$ parmi les sommes extraites de $\mathcal{P}f$ soient de taille $i$.

**11.** Déterminer $\mathcal{T}(0)$, $\mathcal{T}(1)$ et $\mathcal{T}(2)$ pour le portefeuille $\mathcal{P}f = [50;20;20;10;5;5;2]$.

Pour déterminer une somme optimale, on propose la fonction suivante, de signature `int -> somme -> somme` (soit `int -> int list -> int list`), qui prend en argument un prix à payer $p$ et un portefeuille $\mathcal{P}f$, et retourne une somme optimale extraite de $\mathcal{P}f$ de valeur $p$ s'il en existe, et une liste vide sinon :

```
let optimal p pf =
  let tab_m = Array.make (p+1) []
  and t_i = ref [ 0 ] in
  while !t_i <> [] do
    t_i := suivant tab_m pf !t_i
  done;
  tab_m.(p);;
```

Dans cette fonction, la liste `!t_i` recevra successivement les valeurs de $\mathcal{T}(0)\cap[\![0\,..\,p]\!]$, $\mathcal{T}(1)\cap[\![0\,..\,p]\!]$, $\mathcal{T}(2)\cap[\![0\,..\,p]\!]$, etc. où $\cap$ représente l'intersection et $[\![0\,..\,p]\!]$ les entiers naturels inférieurs ou égaux à $p$.

Le tableau contiendra quant à lui, dans chaque case `tab_m.(k)` pour tout $k \leqslant p$, une liste représentant une somme optimale pour $k$ si une telle somme a pu être identifiée, et une liste vide sinon.

La fonction `suivant`, de signature `somme array -> somme -> int list -> int list` prend donc en argument le tableau `tab_m`, le portefeuille $\mathcal{P}f$ et l'ensemble des valeurs de $\mathcal{T}(i)\cap[\![0\,..\,p]\!]$, et retourne l'ensemble des valeurs $\mathcal{T}(i+1)\cap[\![0\,..\,p]\!]$, tout en ayant renseigné les cases de `tab_m` correspondant aux valeurs $\mathcal{T}(i+1)\cap[\![0\,..\,p]\!]$. Les listes contenant les valeurs de $\mathcal{T}(i)\cap[\![0\,..\,p]\!]$ n'ont pas à être triées.

**12.** Proposer une implémentation de la fonction `suivant`, que l'on détaillera avec soin.

## 4 Systèmes monétaires canoniques

Un système monétaire est dit *canonique* lorsque la stratégie gloutonne appliquée à tout prix $p$ produit une somme optimale parmi les représentants de $p$.

**13.** Montrer, en exhibant un prix à payer $p$ et la somme obtenue par l'algorithme glouton, que l'ancien système britannique $\mathcal{D}' = \{240,60,30,24,12,6,3,1\}$ n'est pas canonique.

Le but de cette dernière partie est de produire un programme permettant de déterminer si un système monétaire donné est canonique.

Dans cette étude on fixe un système monétaire $\mathcal{D}$ de $n$ dénominations, représenté cette

fois par un tableau (`int array`) de $n$ entiers naturels $\mathcal{D} = [\![d_0; d_1; \ldots; d_{n-1}]\!]$. Une somme $\mathcal{S}$ sera également représentée par un tableau de $n$ entiers naturels, $\mathcal{S} = [\![s_0; s_1; \ldots; s_{n-1}]\!]$, **mais $s_i$ est cette fois le nombre d'espèces de dénomination $d_i$ présentes dans la somme $\mathcal{S}$.**

Ainsi le système européen correspond au tableau $\mathcal{D} = [\![500; 200; 100; 50; 20; 10; 5; 2; 1]\!]$ et le portefeuille contenant trois billets de dix euros, deux billets de cinq euros et une pièce de un euro au tableau $\mathcal{P}f = [\![0; 0; 0; 0; 0; 3; 2; 0; 1]\!]$. Les définitions (et les notations) de la valeur et de la taille sont inchangées, soit :

$$\mathcal{V}(\mathcal{S}) = \sum_{i=0}^{n-1} s_i d_i \qquad |\mathcal{S}| = \sum_{i=0}^{n-1} s_i$$

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

```
type tsysteme == int array;;
type tsomme == int array;;
```

**14.** Écrire la fonction `t_taille`, de signature `tsomme -> int` (c'est-à-dire `int array -> int`) qui prend une somme $\mathcal{S}$ en argument et renvoie sa taille $|\mathcal{S}|$ (le nombre d'espèces contenues dans la somme).

**15.** Écrire la fonction `t_valeur`, de signature `tsysteme -> tsomme -> int` (c'est-à-dire `int array -> int array -> int`) qui prend un système monétaire et une somme $\mathcal{S}$ en argument et renvoie sa valeur $\mathcal{V}(\mathcal{S})$.

Par ailleurs, les sommes sont ordonnées (totalement) selon l'ordre lexicographique, noté $<_\ell$ et défini de la sorte : pour tous tableaux $\mathrm{U} = [u_0, \ldots, u_{n-1}]$ et $\mathrm{V} = [v_0, \ldots, v_{n-1}]$ reeprésentant deux sommes, on a $\mathrm{U} <_\ell \mathrm{V}$ si et seulement si :

- $\exists i \in [\![0 \,..\, n-1]\!] \mid u_i < v_i$ ;
- et $\forall\, j,\ 0 \leqslant j < i$, on a $u_j = v_j$.

On a par exemple $[\![0; 1]\!] <_\ell [\![0; 2]\!]$ et $[\![0; 4]\!] <_\ell [\![1; 0]\!]$. On note $\leqslant_\ell$ la relation d'ordre définie par $\mathrm{U} \leqslant_\ell \mathrm{V}$ si et seulement si $\mathrm{U} <_\ell \mathrm{V}$ ou $\mathrm{U} = \mathrm{V}$.

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

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

**16.** Montrer que si $\mathcal{S}$ est une somme et $\mathcal{S}'$ une somme non nulle, alors $\mathcal{S} \leqslant_\ell \mathcal{S} \oplus \mathcal{S}'$, où la somme $\oplus$ doit être comprise comme une réunion de sommes, que l'on peut voir comme l'addition, composante par composante, des deux tableaux représentant les sommes $\mathcal{S}$ et $\mathcal{S}'$. On notera $s_k$ le nombre d'espèces de dénomination $d_k$ dans $\mathcal{S}$ et $s'_k$ le nombre d'espèces de dénomination $d_k$ dans $\mathcal{S}'$ (on utilisera ces notations dans la suite). On pourra *commencer* par envisager le cas où $\mathcal{S}'$ ne contient qu'une seule espèce (soit $|\mathcal{S}'| = 1$).

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 $\mathrm{G}(p)$, est le plus grand pour l'ordre lexicographique $\leqslant_\ell$ des représentants de $p$ (cette définition correspond bien au même représentant glouton que précédemment). On notera $g_k(p)$ le nombre d'espèces de dénomination $d_k$ dans $\mathrm{G}(p)$.

Le représentant optimal de $p$, noté $\mathrm{M}(p)$, est le plus grand pour l'ordre $\sqsubseteq$ des représentants de $p$ (cette définition choisit pour représentant optimal de $p$ le plus grand, pour l'ordre lexicographique, parmi tous ceux qualifiés de représentants optimaux avec la définition de la section précédente). On notera $m_k(p)$ le nombre d'espèces de dénomination $d_k$ dans $\mathrm{M}(p)$.

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

**17.** Écrire la fonction `t_glouton` de signature `int -> tsysteme -> tsomme` (soit `int -> int array -> int array`) qui prend en argument un prix à payer $p$ et un système monétaire $\mathcal{D}$ (sous la forme d'un tableau de taille $n$, conformément au nouveau type), et qui renvoie le représentant glouton de ce prix. **Le coût de `t_glouton` doit être linéaire en n.**

**18.** Montrer que si $p < q$, alors $\mathrm{G}(p) <_\ell \mathrm{G}(q)$.

Soit $k$ un indice, $k \in [\![0 \,..\, n-1]\!]$. On note $\mathcal{I}_k$ la somme composée d'une seule espèce de dénomination $d_k$.

**19.** Montrer que pour deux sommes $\mathcal{S}$ et $\mathcal{S}'$ quelconques vérifiant $\mathcal{S} \leqslant_\ell \mathcal{S}'$, pour toute espèce $d_k$, on a $\mathcal{S} \oplus \mathcal{I}_k \leqslant_\ell \mathcal{S}' \oplus \mathcal{I}_k$.

**20.** Soit $p$ un entier naturel. On suppose que $\mathrm{G}(p)$ contient au moins une espèce de dénomination $d_k$. Montrer que $\mathrm{G}(p - d_k) = \mathrm{G}(p) \ominus \mathcal{I}_k$ où la soustraction $\ominus$ doit être comprise comme une différence de sommes, que l'on peut voir comme la soustraction, composante par composante, des deux tableaux représentant les sommes $\mathrm{G}(p)$ et $\mathcal{I}_k$.

**21.** Montrer, de même, que si $\mathrm{M}(p)$ contient au moins une espèce de dénomination $d_k$, on a $\mathrm{M}(p - d_k) = \mathrm{M}(p) \ominus \mathcal{I}_k$.

On suppose le système monétaire non-canonique, et on considère le contre-exemple $w$ minimal, c'est-à-dire l'entier $w$ tel que $\mathrm{M}(w) <_\ell \mathrm{G}(w)$ et $\forall\, w' < w$, $\mathrm{M}(w') = \mathrm{G}(w')$.

On note $\mathrm{M}(w) = [\![m_0; m_1; \ldots; m_{n-1}]\!]$ et $\mathrm{G}(w) = [\![g_0; g_1; \ldots; g_{n-1}]\!]$. On pose $i \in [\![0 \,..\, n-1]\!]$ l'indice minimal tel que $m_i > 0$ et $j \in [\![0 \,..\, n-1]\!]$ l'indice maximal tel que $m_j > 0$.

**22.** Montrer qu'il n'existe pas d'indice $k$ tel que $m_k > 0$ et $g_k > 0$.

**23.** Montrer que l'on a $i > 0$.

**24.** Montrer que $d_{i-1} < w$.

**25.** Montrer que $w < d_{i-1} + d_j$.

Toujours en supposant le système monétaire non-canonique, et en conservant les notations de la question précédente, on admet l'encadrement suivant, montré par D. Pearson en 1994, qui s'applique aux sommes :

$$\mathrm{M}(w) \ominus \mathrm{I}_j \leqslant_\ell \mathrm{G}(d_{i-1} - 1) <_\ell \mathrm{M}(w)$$

**26.** Montrer que, pour $i$ et $j$ donnés, l'encadrement permet de construire $\mathrm{M}(w)$ à partir de $\mathrm{G}(d_{i-1} - 1)$.

**27.** En déduire une fonction `canonique` de signature `tsysteme -> bool` prenant en argument un système monétaire, et retournant un booléen indiquant si celui-ci est canonique.

**28.** Quelle est, en fonction de $n = |\mathcal{D}|$, la complexité en temps dans le pire des cas de la fonction précédente ?