INFORMATIQUE

Préliminaires :

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 III « 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. Il est rappelé que cette épreuve est une épreuve d'informatique, et que l'absence de programmes sera donc sanctionnée comme il se doit.

Les algorithmes demandés seront décrits en français ; les programmes seront rédigés en langage Caml ou en langage Pascal. ***Les candidats indiqueront impérativement au tout début de leur copie le langage de programmation qu'ils ont choisi d'utiliser.*** *Une approche modulaire est vivement conseillée, comme l'a indiqué René Descartes, il convient « de diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait, et qu'il serait requis pour les mieux résoudre ». Lorsque les candidats choisiront d'écrire une fonction annexe non demandée,* ***il leur est demandé avec insistance d'expliquer*** **avant** ***les spécifications de cette fonction.***

Certaines questions ou remarques sont réservées soit aux candidats ayant choisi le langage Caml, soit à ceux ayant choisi le langage Pascal. Dans ce cas, la question ou remarque est annoncée par un encadré, et se termine par un trait de séparation pointillé ou un nouvel encadré.

Pour les questions de complexité, on demande d'évaluer les coûts globalement en terme d'opérations élémentaires telles que des affectations, des opérations arithmétiques, comparaisons, tests... En particulier, on ne s'attachera pas à des considérations concernant la taille des données.

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 $m \in \mathbb{N}^* = \mathbb{N} \setminus \{0\}$, $\mathbb{R}^m$ est muni de sa structure euclidienne canonique en posant, pour $u, v \in \mathbb{R}^m$: $u \cdot v = \sum_{i=1}^{m} u_i v_i$.

Filière MP

- 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 $c = (c_i)_{1 \leq i \leq m}$ *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*:

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

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

Partie I - Systèmes de pièces

Note à l'attention des candidats qui ont choisi le langage Caml

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* $(6, 3, 1)$.

La question suivante est destinée aux candidats qui ont choisi le langage Caml

I.A - Rédiger en Caml une fonction de signature

```
est_un_systeme : int list -> bool
```

spécifiée comme suit : `est_un_systeme c` indique si la liste `c` est bien un système. Par exemple, `est_un_systeme [5;2;1]` rendra la valeur `true`, tandis que `est_un_systeme [5;7;1]`rendra la valeur `false`, tout comme le fera `est_un_systeme [7;5;2].`

Note à l'attention des candidats qui ont choisi le langage Pascal.

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. Le type utilisé est défini comme suit :

```
type Liste = ^Cellule;
type Cellule =
  record
   contenu : integer;
   suivant : Liste
  end ;
```

La liste vide est représentée par la constante `nil`. *Pour créer des listes, nous disposons d'une procédure et d'une fonction dont les en-têtes sont* :

```
procedure pCons(x:integer; var c:Liste);
function fCons(x:integer; c:liste):Liste;
```

spécifiées comme suit : `pCons(x,c)` a*joute en tête de liste* `c` *une cellule dont le champ* `contenu` *vaut* `x`. `fCons(x,c)` *rend une liste dont la première cellule contient* `x` *dans son champ* `contenu`, *et dont le champ* `suivant` *vaut* `c`.

Par exemple, le programme suivant construit les listes $L_1 = (5, 2, 1)$, $L_2 = (5, 7, 1)$ *et* $L_3 = (7, 5, 2)$ *dont il sera question dans la suite .*

```
var L1,L2,L3:Liste;
begin
  L1:=nil; pCons(1,L1); pCons(2,L1) ; pCons(5,L1);
  L2:=fCons(5,fCons(7,fCons(1,nil)));
  L3:=fCons(7,fCons(5,fCons(2,nil)));
end;
```

La question suivante est destinée aux candidats qui ont choisi le langage Pascal.

I.B - Rédiger en Pascal une fonction d'en-tête

```
function est_un_systeme(c:Liste):boolean;
```

spécifiée comme suit : `est_un_systeme(c)` indique si la liste `c` est bien un système. Par exemple, `est_un_systeme(L1)` rendra la valeur `true`, tandis que `est_un_systeme(L2)` rendra la valeur `false`, tout comme le fera `est_un_systeme(L3)`.

Partie II - Représentations de poids minimal

Soient $c = (c_1, ..., c_m)$ 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*.

II.A - Prouver l'encadrement

$$\left\lceil \frac{x}{c_1} \right\rceil \leq M(x) \leq x .$$

II.B - Exhiber un système c et un nombre $x \in \mathbb{N}^*$ possédant plusieurs représentations dans ce système.

II.C - Soient c un système et $x \in \mathbb{N}^*$. Montrer que $M(x) \leq 1 + M(x - c_j)$ pour tout indice j tel que $c_j \leq x$.

II.D - Soient c un système, $x \in \mathbb{N}^*$, et j tel que $c_j \leq x$. Montrer **soigneusement** 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 $k_j > 0$).

II.E - Soit $x > 1$; notons s le plus petit indice i tel que $c_i \leq x$. Justifier l'égalité

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

II.F - Soit $x > 0$. Montrer que l'on peut construire la liste des valeurs de $M(y)$ pour $y \leq x$, pour un coût $O(mx)$ (au sens précisé dans les préliminaires).

La question suivante est destinée aux candidats qui ont choisi le langage Caml.

II.G - Rédiger en Caml une fonction de signature :

```
poids_minimaux : int -> int list -> int list
```

spécifiée comme suit : `poids_minimaux x c` construit la liste des valeurs de $M_c(y)$ pour $1 \leq y \leq x$. Par exemple, `poids_minimaux 5 [5;2;1]` rendra la liste `[1;1;2;2;1]`. Cet exemple fournit d'ailleurs l'ordre dans lequel on souhaite que les $M_c(y)$ apparaissent dans la liste résultat.

On utilisera la formule de la question II.E). De plus, on pourra utiliser, si on le souhaite, les fonctions `list_of_vect, vect_of_list et make_vect` de signatures respectives :

```
list_of_vect :'a vect -> 'a list
vect_of_list :'a list -> 'a vect
make_vect : int -> 'a -> 'a vect
```

La question suivante est destinée aux candidats qui ont choisi le langage Pascal.

II.H - Rédiger en Pascal une fonction d'en-tête

```
function poids_minimaux(x:integer; c:Liste):Liste;
```

spécifiée comme suit : `poids_minimaux(x,c)` rendra une liste des valeurs de $M_c(y)$, pour $1 \leq y \leq x$. Par exemple, `poids_minimaux(5,L1)` rendra la liste $(1,1,2,2,1)$. Cet exemple fournit d'ailleurs l'ordre dans lequel on souhaite que les $M_c(y)$ apparaissent dans la liste résultat.

On utilisera la formule de la question II.E, et on pourra employer librement la constante `mMax`, le type `vecteur`, la fonction `vecteur_vers_liste` et la procédure `liste_vers_vecteur` qui pourraient être définis ainsi :

```
const mMax =1000;

type vecteur = array[1..mMax] of integer;

function vecteur_vers_liste(v:vecteur; r:integer):Liste;
var
  p:Liste;
  k:integer;
begin
  p:= nil;
  for k:=r downto 1 do pCons(v[k],p);
```

```
    vecteur_vers_liste:=p
  end;

  procedure liste_vers_vecteur (L:Liste; var v:vecteur;var
  taille:integer);
  begin
    taille:=0;
    while L<>nil do
     begin
         taille:=taille+1
         v[taille]:=L^.contenu;
         L:=L^.suivant
     end
  end;
```

Ainsi, `vecteur_vers_liste(v,r)` retourne la liste des $(v_i)_{1 \leq i \leq r}$ sous réserve que l'encadrement $1 \leq r \leq \texttt{mMax}$ soit vérifié. Si $r \leq 0$, elle rend la liste vide ; si $r > \texttt{mMax}$, le résultat n'est pas spécifié.

L'appel `liste_vers_ vecteur(L,v,t)` place quant à lui les éléments de la liste L dans un vecteur v, en mettant à jour la variable t représentant le nombre d'éléments de la liste.

———————————————————————————————————

Partie III - 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 $q = \left\lfloor \frac{x}{c_1} \right\rfloor$, cet algorithme rend q pièces de valeur c_1, puis rend le montant $x - qc_1$ en utilisant le système $(c_2, \ldots, c_m)$.

La question suivante est destinée aux candidats qui ont choisi le langage Caml.

III.A - Rédiger en Caml une fonction de signature :

```
glouton : int -> int list -> int list
```

spécifiée comme suit : `glouton x c` construit la représentation de x dans le système c en utilisant l'algorithme glouton.

Par exemple, `glouton 13 [5;2;1]` rendra la liste `[2;1;1]`.

La question suivante est destinée aux candidats qui ont choisi le langage Pascal.

III.B - Rédiger en Pascal une fonction d'en-tête

```
function glouton(x:integer; c:Liste):Liste;
```

spécifiée comme suit : `glouton(x,c)` retourne une liste contenant la représentation de x dans le système c, en appliquant l'algorithme glouton. Par exemple, `glouton (13,L1)` retournera la liste (2,1,1).

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 $M(x) = G_c(x)$ pour tout $x \in \mathbb{N}$.

III.C - Montrer que tout système $c = (c_1, c_2)$ est canonique.

III.D - Exhiber un système $c = (c_1, c_2, c_3)$ non canonique (en justifiant).

III.E - Soient q et n deux entiers ≥ 2. Montrer que le système $(q^n, q^{n-1}, \ldots, q, 1)$ est canonique.

III.F - Montrer que le système « Euro » $(200, 100, 50, 20, 10, 5, 2, 1)$ est canonique.

On pourra montrer que dans une représentation minimale de x : $k = (k_1, \ldots, k_8)$, on a $k_8 \leq 1$, $2k_7 + k_8 \leq 4$, $5k_6 + 2k_7 + k_8 \leq 9$ ...

III.G - Avant la réforme de 1971 (introduisant un système décimal), le Royaume-Uni utilisait le système $(30, 24, 12, 6, 3, 1)$. Montrer que ce système n'est pas canonique.

Partie IV - L'algorithme de Kozen et Zaks

Nous allons voir ici un algorithme efficace permettant de déterminer si un système c 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, $m \geq 3$.

IV.A - Soit c un système non canonique. Il admet donc des contre-exemples. Montrer que le plus petit d'entre-eux, disons x, vérifie :

$$c_{m-2} + 1 < x < c_1 + c_2 .$$

Pour la seconde inégalité, on pourra raisonner par l'absurde et utiliser la question II.D.

IV.B - Soit $q \geq 3$. Montrer que le système $(q+1, q, 1)$ n'est pas canonique. Quel est le plus petit contre-exemple pour ce système ? Justifier.

IV.C - Soit $q \geq 3$. Déterminer $\alpha(q) > q$ tel que le système $(\alpha(q), q, 1)$ ne soit pas canonique, et admette $\alpha(q)+2$ comme plus petit contre-exemple.

IV.D - Que vient-on de faire dans les deux questions précédentes ?

Le résultat de la question IV.A 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 $G(x - c_i) < G(x) - 1$.*

IV.E - Montrer que tout témoin est un contre-exemple.

IV.F - Montrer que le résultat précédent n'admet pas de réciproque.

On pourra travailler dans le système (5,4,1).

IV.G - 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 suffit 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.

IV.H - En utilisant l'algorithme de Kozen et Zaks, montrer que les systèmes $(q, 2, 1)$ $(q \geq 3)$ et $(7, 4, 1)$ sont canoniques, alors que $(6, 5, 1)$ ne l'est pas.

La question suivante est destinée aux candidats qui ont choisi le langage Caml.

IV.I - Rédiger en Caml une fonction de signature :

```
kozen_zaks : int list -> bool
```

spécifiée comme suit : `kozen_zaks c` indique si la liste c est canonique. Par exemple, `kozen_zaks [5;2;1]` rendra la valeur `true` et `kozen_zaks [6;5;1]` rendra la valeur `false`. On utilisera l'algorithme de Kozen et Zaks.

La question suivante est destinée aux candidats qui ont choisi le langage Pascal.

IV.J - Rédiger en Pascal une fonction d'en-tête

```
function kozen_zaks(c:Liste):boolean;
```

spécifiée comme suit : `kozen_zaks(c)` indique si la liste c est canonique. Par exemple, `kozen_zaks(L1)` rendra la valeur `true`. On utilisera l'algorithme de Kozen et Zaks.

— —

IV.K - Montrer que le coût de l'algorithme de Kozen et Zaks peut être exponentiel, par rapport au nombre m de pièces du système.

On exhibera deux systèmes (l'un canonique, l'autre pas) pour lesquels le coût de l'algorithme est exponentiel en m .

••• FIN •••

À titre culturel, on signale l'existence d'un algorithme, dû à Pearson, résolvant le même problème mais dont le coût est polynômial en m dans le pire des cas.