Rendu de monnaie (d'après X et Centrale MP 2002)

Durée : 2 heures

Le sujet traite du problème du monnayeur : comment rendre la monnaie en utilisant le plus petit nombre de pièces ? La première partie met en place le formalisme et résout le problème à l'aide de la programmation dynamique, en supposant disposer d'un nombre illimité de pièces de chaque type. On étudiera dans la deuxième partie l'*algorithme glouton*, qui, on le verra, n'est pas toujours optimal. Enfin, la dernière partie tiendra compte du contenu du portefeuille disponible pour rendre la monnaie.

Formalisation du problème

On appelle *système* un m-uplet d'entiers $c = (c_i)_{1\leqslant i\leqslant m}$ (avec $m \geqslant 2$) vérifiant : $c_1 > c_2 > \cdots > c_m = 1$. Les c_i sont les valeurs faciales des pièces (ou billets) en service. Par exemple, le système utilisé en zone Euro est : $(500, 200, 100, 50, 20, 10, 5, 2, 1)$.

Dans les deux premières parties de ce problème on supposera disposer pour tout $i \in [\![1, m]\!]$ 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_i est donc le nombre de pièces c_i qui seront rendues.

Enfin, pour épargner les poches des clients, nous souhaitons minimiser le poids de cette représentation, c'est à dire la quantité :

$$w(k) = \sum_{i=1}^{m} k_i.$$

Partie I. Représentations de poids minimal

Nous utiliserons des listes d'entiers pour représenter aussi bien un système 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 Python une fonction de signature :

```
def est_un_systeme(c: [int]) -> bool
```

spécifiée comme suit : `est_un_systeme(c)` indique si la liste c est bien un système.

Question 2. Rédiger en Python une fonction de signature :

```
def poids(k: [int]) -> int
```

qui calcule le poids $w(k)$ d'une représentation k d'un certain montant.

Question 3. Rédiger en Python une fonction de signature :

```
def montant(k: [int], c: [int]) -> int
```

qui prend pour arguments une représentation k et un système c et qui renvoie le montant x représenté par k dans le système c.

Soient $c = (c_1, \ldots, c_m)$ un système, et $x \in \mathbb{N}$. Nous notons $\mathrm{M}(x)$ le plus petit nombre de pièces nécessaires pour représenter x dans le système c :

$$\mathrm{M}(x) = \min\left\{ w(k) \;\middle|\; k \in \mathbb{N}^m \text{ et } x = \sum_{i=1}^{m} k_i c_i \right\}.$$

Nous nous intéresserons aux représentations de poids minimal de x : ce sont les représentations k telles que $w(k) = \mathrm{M}(x)$.

Question 4.

a) Montrer que pour tout indice j tel que $c_j \leqslant x$ on a : $\mathrm{M}(x) \leqslant 1 + \mathrm{M}(x - c_j)$.

b) Montrer qu'on a : $\mathrm{M}(x) = 1 + \mathrm{M}(x - c_j)$ si et seulement s'il existe une représentation minimale k de x faisant intervenir c_j, c'est à dire telle que $k_j > 0$.

c) Soit s le plus petit indice i vérifiant : $c_i \leqslant x$. Justifier l'égalité :

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

Question 5. Rédiger en Python une fonction de signature :

```
def poids_minimaux(x: int, c: [int]) -> [int]
```

spécifiée comme suit : si $x \in \mathbb{N}$ est un entier naturel et c un système, alors `poids_minimaux(x, c)` renvoie le tableau des valeurs de $\mathrm{M}(y)$ pour $0 \leqslant y \leqslant x$. Par exemple, `poids_minimaux(5, [5, 2, 1])` renvoie la liste `[0, 1, 1, 2, 2, 1]`. Cet exemple fournit d'ailleurs l'ordre dans lequel on souhaite que les $\mathrm{M}(y)$ apparaissent dans le tableau résultat.

Question 6. Rédiger en Python une fonction de signature :

```
def representation_minimale(x: int, c:[int]) -> [int]
```

spécifiée comme suit : `representation_minimale(x, c)` renvoie une représentation minimale de x dans le système c. Par exemple, `representation_minimale(6, [10, 5, 2, 1])` renvoie la liste `[0, 1, 0, 1]`.
On utilisera la fonction `poids_minimaux` définie à la question précédente.

Partie II. L'algorithme glouton

L'algorithme glouton pour rendre une somme $x > 0$ consiste à choisir le plus grand $c_i \leqslant x$, 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 $q = \left\lfloor \dfrac{x}{c_1} \right\rfloor$, cet algorithme rend q pièces de valeur c_1, puis rend le montant $x - qc_1$ en utilisant le système $(c_2, \ldots, c_m)$.

Question 7. Rédiger en Python une fonction de signature :

```
def glouton(x: int, c: [int]) -> [int]
```

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])` renvoie la liste `[2, 1, 1]`.

Systèmes canoniques

Nous noterons $\Gamma(x)$ la représentation gloutonne de x dans le système c, et $\mathrm{G}(x)$ le nombre de pièces utilisées par l'algorithme glouton : $\mathrm{G}(x) = w(\Gamma(x))$.
Nous dirons que le système c est *canonique* lorsque l'algorithme glouton nous donne toujours une représentation minimale ; on a alors $\mathrm{M}(x) = \mathrm{G}(x)$ pour tout $x \in \mathbb{N}$.

Question 8.

a) Montrer que tout système (c_1, c_2) est canonique.

b) Exhiber un système (c_1, c_2, c_3) non canonique (en justifiant).

c) 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.

Nous dirons qu'un entier x est un *contre-exemple* pour c lorsque $\mathrm{M}(x) < \mathrm{G}(x)$. Un système canonique n'admet donc pas de contre-exemple. *Nous admettrons que si un système n'est pas canonique, il existe un contre-exemple $x < c_1 + c_2$.*

Question 9. En déduire une fonction de signature :

```
est_canonique(c:[int]) -> bool
```

qui détermine si le sytème c est canonique.

Partie III. Paiement exact avec une ressource finie

Dans cette dernière partie, on tient compte des ressources du payeur. Pour cela on introduit la notion de *portefeuille* : c'est la liste des pièces ou billets détenues par le payeur. Par exemple, un portefeuille contenant trois billets de 10 euros, deux billets de 5 euros et une pièce de 1 euro sera représenté par la liste $(10, 10, 10, 5, 5, 1)$. Par convention, les espèces seront présentées par valeurs faciales décroissantes.

Question 10. Montrer, à l'aide d'un exemple utilisant le système européen, que la stratégie gloutonne peut échouer pour un prix donné, même s'il est possible de payer compte tenu du contenu du portefeuille.

Question 11. Rédiger une fonction de signature :

```
def paye_glouton(x: int, p: [int]) -> [int]
```

qui prend pour arguments un montant à payer x et un portefeuille p et qui renvoie un sous-ensemble de p dont la somme est égale à x. Cette fonction suivra la démarche gloutonne et si cette démarche échoue, la fonction `paye_glouton` doit renvoyer `None`.
Par exemple, `paye_glouton(8, [5, 5, 2, 2, 1])` doit renvoyer `[5, 2, 1]` et `paye_glouton(8, [7, 4, 4])` renvoyer `None`.

Question 12. Écrire enfin une fonction :

```
def compte_paiements(x: int, p: [int]) -> int
```

qui prend en arguments un montant x et un portefeuille `p` et qui renvoie le nombre de façons de payer `x` à l'aide des espèces de `p`. On adoptera pour ce faire une démarche basée sur la programmation dynamique.

Remarque. Deux espèces de même dénomination sont distinguées. Ainsi, si le portefeuille est $[2, 2, 2]$ et le montant 2, alors il y a trois façons de payer.