Corrigé : rendu de monnaie (d'après X et Centrale MP 2002)

Partie I. Représentations de poids minimal

Question 1.

```
1  def est_un_systeme(c):
2      for i in range(len(c) - 1):
3          if c[i] <= c[i + 1]:
4              return False
5      return c[-1] == 1
```

Les lignes 2-4 vérifient la condition $c_1 > c_2 > \cdots > c_m$, la ligne 5 la condition $c_m = 1$.

Question 2. On a $w(k) = \sum_{i=1}^{m} k_i$ donc il s'agit s'implement de faire la somme des éléments de k :

```
def poids(k):
    s = 0
    for n in k:
        s += n
    return s
```

Question 3. On a cette fois $x = \sum_{i=1}^{m} k_i c_i$, d'où la fonction :

```
def montant(k, c):
    x = 0
    for i in range(len(c)):
        x += k[i] * c[i]
    return x
```

Question 4.

a) Soit $k' = (k'_1, \ldots, k'_j, \ldots, k'_m)$ une représentation minimale de $x - c_j$. Alors $k = (k'_1, \ldots, k'_j + 1, \ldots, k'_m)$ est une représentation (non nécessairement minimale) de x donc : $\mathrm{M}(x) \leqslant 1 + \sum_{i=1}^{m} k'_i$, soit $\mathrm{M}(x) \leqslant 1 + \mathrm{M}(x - c_j)$.

b) Avec les notations précédentes, si $\mathrm{M}(x) = 1 + \mathrm{M}(x - c_j)$ alors $k = (k'_1, \ldots, k'_j + 1, \ldots, k'_m)$ est une représentation minimale de x faisant intervenir c_j.
Réciproquement, supposons qu'il existe une représentation minimale $k = (k_1, \ldots, k_j, \ldots, k_m)$ de x pour laquelle $k_j \geqslant 1$. Alors $k' = (k_1, \ldots, k_j - 1, \ldots, k_m)$ est une représentation (non nécessairement minimale) de $x - c_j$, et : $\mathrm{M}(x - c_j) \leqslant \sum_{i=1}^{m} k_i - 1$, soit $\mathrm{M}(x - c_j) \leqslant \mathrm{M}(x) - 1$. Compte tenu de la question précédente, on en déduit $\mathrm{M}(x) = \mathrm{M}(x - c_j) - 1$, ce qui achève la preuve de l'équivalence.

c) D'après la question 4a nous avons : $\mathrm{M}(x) \leqslant 1 + \min_{s \leqslant i \leqslant m} \mathrm{M}(x - c_i)$.
Soit maintenant $j \in [\![s, m]\!]$ tel que c_j intervienne dans une représentation minimale de x. D'après la question 4b, on a $\mathrm{M}(x) = 1 + \mathrm{M}(x - c_j)$, donc en définitive : $\mathrm{M}(x) = 1 + \min_{s \leqslant i \leqslant m} \mathrm{M}(x - c_i)$.

Question 5. Nous allons construire le tableau $[\mathrm{M}(0), \mathrm{M}(1), \cdots, \mathrm{M}(x)]$ en observant que la formule établie à la question 4 permet de remplir la case d'indice i une fois toutes les cases d'indices $j < i$ remplies.

```
1  def poids_minimaux(x, c):
2      M = [0 for _ in range(x + 1)]
3      for y in range(1, x + 1):
4          s = 0
5          while c[s] > y:
6              s += 1
7          mini = M[y - c[s]]
8          for i in range(s + 1, len(c)):
9              mini = min(mini, M[y - c[i]])
10         M[y] = 1 + mini
11     return M
```

les lignes 4-6 calculent l'entier s défini à la question 4c ; les lignes 7-10 calculent $\mathrm{M}(y)$ à l'aide de la formule établie à cette même question.

Question 6. Une fois le tableau $[\mathrm{M}(0), \mathrm{M}(1), \ldots, \mathrm{M}(x)]$ construit, il reste à déterminer une valeur de i pour laquelle $\mathrm{M}(x) = 1 + \mathrm{M}(x - c_i)$ puis à recommencer le processus avec $x - c_i$ tant que cette quantité est strictement positive.

```
def representation_minimale(x, c):
    M = poids_minimaux(x, c)
    k = [0 for _ in c]
    i = len(c) - 1
    while x > 0:
        while M[x] < 1 + M[x - c[i]]:
            i -= 1
        k[i] += 1
        x -= c[i]
    return k
```

Partie II. L'algorithme glouton

Question 7.

```
def glouton(x, c):
    k = [0 for _ in c]
    for i in range(len(c)):
        q = x // c[i]
        k[i] = q
        x = x - q * c[i]
    return k
```

Question 8.

a) Si $c = (c_1, c_2)$ avec $c_1 \geqslant 2$ et $c_2 = 1$, alors pour tout $x \in \mathbb{N}^*$, $\Gamma(x) = (q, r)$ avec $x = qc_1 + r$ et $0 \leqslant r \leqslant c_1 - 1$ (c'est le quotient et le reste de la division euclidienne de x par c_1). On a donc : $\mathrm{G}(x) = q + r$.
Considérons maintenant une représentation minimale (q', r') de x. Alors :

$$x = qc_1 + r = q'c_1 + r' \quad \text{et} \quad q' + r' \leqslant q + r.$$

De l'égalité : $q' + r' = q + r + (q - q')(c_1 - 1)$ on déduit : $q - q' \leqslant 0$. Or $q' > q$ implique $q'c_1 > x$, ce qui est absurde. On a donc $q = q'$ et par suite $r = r'$. Ainsi, $\mathrm{G}(x) = \mathrm{M}(x)$, et de plus la représentation gloutonne est l'unique représentation minimale de x.

b) Considérons le système $c = (4, 3, 1)$ et $x = 6$. Alors $\Gamma(x) = (1, 0, 2)$ donc $\mathrm{G}(x) = 3$, alors que $\mathrm{M}(x) = 2$ puisque $x = 3 + 3$. Plus généralement, tout système $c = (p, p-1, 1)$ avec $p > 3$ et $x = 2p - 2$ convient.

c) Considérons $x = 48$. On a $\Gamma(x) = (1, 0, 1, 1, 0, 0)$ donc $\mathrm{G}(x) = 3$, alors que $\mathrm{M}(x) = 2$ puisque $48 = 24 + 24$.

Question 9. Il s'agit ici de calculer tous les poids minimaux $\mathrm{M}(x)$ pour $x < c_1 + c_2$ puis à comparer ces derniers avec le résultat de l'algorithme glouton.

```
def est_canonique(c):
    M = poids_minimaux(c[0] + c[1] - 1, c)
    for x in range(c[0] + c[1]):
        if poids(glouton(x, c)) > M[x]:
            return False
    return True
```

Partie III. Paiement exact avec une ressource finie

Question 10. Pour un portefeuille égal à $(5,2,2,2)$, la stratégie gloutonne échoue à payer la somme de 6 euros bien qu'il soit possible de payer.

Question 11. Les espèces du portefeuille étant rangées par ordre décroissant, on applique la stratégie gloutonne en parcourant le portefeuille de la gauche vers la droite. Si à la fin du processus la somme restant à payer est nulle, c'est un succès ; dans le cas contraire la démarche gloutonne échoue.

```
def paye_glouton(x, p):
    montant = []
    for e in p:
        if e <= x:
            montant.append(e)
            x -= e
    if x == 0:
        return montant
```

Question 12. Adoptions une démarche récursive : notons $p = (e_1, \dots, e_m)$ le portefeuille et notons $t(i,j)$ le nombre de décompositions de l'entier j pris dans le portefeuille $(e_i, \dots, e_m)$. Si le portefeuille est vide, la seule décomposition possible concerne la somme nulle, donc $t(m+1,j) = \begin{cases} 1 & \text{si } j = 0 \\ 0 & \text{sinon} \end{cases}$

Dans les autres cas, on peut rendre ou ne pas rendre la dénomination e_i, ce qui conduit à la formule

$$t(i,j) = \begin{cases} t(i+1, j-e_i) + t(i+1,j) & \text{si } e_i \leqslant j \\ t(i+1,j) & \text{si } j < e_i \end{cases}$$

```
def compte_paiements(x, p):
    m = len(p)
    t = [[0 for j in range(x + 1)] for i in range(m + 1)]
    t[m][0] = 1
    for i in range(m - 1, -1, -1):
        for j in range(x + 1):
            if j < p[i]:
                t[i][j] = t[i + 1][j]
            else:
                t[i][j] = t[i + 1][j] + t[i + 1][j - p[i]]
    return t[0][x]
```