Préparation aux oraux CCINP n°4

Rendu de monnaie

Dans ce sujet, on considère le problème de rendu de monnaie : étant donnée une somme v, donner un nombre n minimal de pièces dont la somme vaut v. Dans la suite, un billet pourra être considéré comme une pièce de même valeur. Nous appellerons donc les billets des pièces, sans distinction avec les pièces réelles.

Par exemple, avec le système de pièce européen, pour rendre $v = 32{,}48$ €, on peut donner :
- 6×5 € $+ 2$ € $+ 2 \times 0{,}20$ € $+ 0{,}05$ € $+ 3 \times 0{,}01$ €, soit 12 pièces ;
- 32×1 € $+ 2 \times 0{,}20$ € $+ 0{,}05$ € $+ 3 \times 0{,}01$ €, soit 38 pièces ;
- $3248 \times 0{,}01$ €, soit 3248 pièces ;
- 20 € $+ 10$ € $+ 2$ € $+ 2 \times 0{,}20$ € $+ 0{,}05$ € $+ 3 \times 0{,}01$ €, soit 9 pièces.

La solution avec 9 pièces est la solution optimale : elle contient le nombre minimum de pièces.

Dans l'exemple ci-dessus, on utilise l'*euro*. Mais, dans la suite, nous ne nous limiterons pas à un tel système. En effet, on définit un *système monétaire* comme un sous-ensemble fini de $\mathbb{D}^+$.[1] Ainsi, l'*euro* est un système monétaire défini comme :

$$\textbf{Euro} = \left\{ \underbrace{500, 200, 100, 50, 20, 10, 5}_{\text{billets}}, \overbrace{2, 1, \frac{1}{2}, \frac{1}{5}, \frac{1}{10}, \frac{1}{20}, \frac{1}{50}, \frac{1}{100}}^{\text{pièces}} \right\}.$$

On peut également définir d'autres systèmes monétaires, comme le système monétaire Américain par exemple :

$$\textbf{USD} = \left\{ \underbrace{100, 50, 20, 10, 5, 2, 1}_{\text{billets}}, \overbrace{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{10}, \frac{1}{20}, \frac{1}{100}}^{\text{pièces}} \right\}.$$

En OCaml, on représentera un système monétaire par un tableau d'entiers (type `currency`), afin d'éviter des problèmes d'erreurs d'arrondis de flottants. Ainsi, 5 € sera représenté par $500 = 5 \times 100$; $0{,}01$ € sera représenté par $1 = 0{,}01 \times 100$.

Un rendu sera représenté par un tableau d'entiers (type `change`). Chaque entier c_i de ce tableau (à l'indice i) représentera le nombre de pièces à rendre, dont la valeur sera donnée par le système monétaire : il s'agit de la i-ème pièce, dans sa représentation par un tableau.

```
type currency = int array
type change = int array
```

On définit donc formellement le problème RENDU-MONNAIE comme

RENDU-MONNAIE :	**Entrée.** Un système monétaire $\mathbf{M}$, une valeur v **Sortie.** Le nombre minimal de pièces de $\mathbf{M}$ de somme v.

[1] On rappelle que $\mathbb{D}$ est l'ensemble des nombres décimaux, c'est à dire les nombres rationnels admettant une écriture décimale *finie*.

Q1. Corriger la fonction `verifie` qui vérifie si un assortiment de pièces rend bien la bonne somme.

Q2. Écrire une fonction récursive `rendu_glouton` qui, à chaque fois, essaie de sélectionner la pièce de plus grande valeur.

Q3. Commenter le fonctionnement de `rendu_glouton` sur des entrées données.

Observons la structure récursive du problème.

- Pour rendre 0 €, on n'a qu'une possibilité : ne rendre aucune pièce.
- Pour rendre une somme s, on calcule pour toute pièce $p \in \mathbf{M}$ (avec $s \geq p$) le rendu de $s - p$. Chaque rendu est une liste finie $Q_p = \left(q_{p,1}, q_{p,2}, \ldots, q_{p,n_p}\right)$ de longueur n_p. On choisit un rendu avec le plus petit nombre de pièces, de somme $s - p^\star$. On crée ainsi un rendu de longueur $n_{p^\star} + 1$ de somme s :

$$\Big(\underbrace{q_{p^\star,1}, q_{p^\star,2}, \ldots, q_{p^\star,n_{p^\star}}}_{\text{noté } (\ldots Q_{p^\star} \ldots)}, p^\star\Big).$$

  Ce rendu est de longueur minimale pour la somme s.

Q4. On décide d'utiliser de la programmation dynamique afin d'améliorer les performances de la stratégie précédente. Pourquoi ?

Dans cette approche, on introduit une liste finie $(\ell_k)_{k \in [\![0,s]\!]}$ où chaque ℓ_k est un rendu (sous la forme d'une liste de pièces) minimal de la somme k.

Q5. Donner une relation de récurrence entre les ℓ_k. Quel(s) est(/sont) le(s) cas de base ?

Q6. Écrire une fonction `rendu_prog_dyn` qui applique la stratégie de programmation dynamique.

Indication C. On rappelle l'existence de la fonction `Array.make_matrix` qui permet de créer une matrice (vue comme un tableau de tableaux) de taille `n` par `m` avec la valeur `x` sur chaque coefficient en l'appelant comme :

```
Array.make_matrix n m x
```

Q7. Quelle est la complexité de la fonction précédente ?

Q8. À l'aide d'un argument bien choisi, expliquer pourquoi la fonction précédente est correcte.