Le problème du rendu de monnaie

On dispose de pièces de monnaie et de billets, par exemple dans le système monétaire européen nous avons des pièces de 1 et 2 euros, puis des billets de 5, 10, 20, 50, 100, 200 et 500 euros. Dans la suite, nous appellerons indifféremment "pièces" soit de vraies pièces de monnaie, soit des billets.

On souhaite rendre une somme d'argent s avec ces pièces mais en utilisant le moins de pièces possible, sachant que l'on n'est pas limité et qu'on dispose de toutes les pièces nécessaires.

1) Formalisation mathématique du problème

Il y a 9 types de pièces que nous numérotons de 1 à 9. Appelons m_i le montant des pièces de type n°i. Nous avons donc :

$m_1 = 1,\ \ m_2 = 2,\ \ m_3 = 5,\ \ m_4 = 10,\ \ m_5 = 20,\ \ m_6 = 50,\ \ m_7 = 100,\ \ m_8 = 200 \quad \text{et} \quad m_9 = 500$

Si on appelle n_i le nombre de pièces de montant m_i dans la somme s à rendre et n le nombre de pièces rendues, nous aurons :

$$s = \sum_{i=1}^{9} n_i\, m_i \quad (1) \quad \text{et} \quad n = \sum_{i=1}^{9} n_i \quad (2)$$

Fixons-nous une somme s à rendre et appelons $E(s)$ l'ensemble des solutions de (1) :

$$E(s) = \left\{ (n_1, n_2, \ldots, n_9) \in \mathbb{N}^9 \;\middle|\; s = \sum_{i=1}^{9} n_i\, m_i \right\}$$

Il est facile de vérifier que $E(s) \neq \varnothing$ pour tout $s \in \mathbb{N}$. Soit alors $V(s)$ l'ensemble :

$$V(s) = \left\{ n = \sum_{i=1}^{9} n_i \;\middle|\; (n_1, n_2, \ldots, n_9) \in E(s) \right\} \subset \mathbb{N}$$

En tant que partie non vide de $\mathbb{N}$, $V(s)$ admet un plus petit élément que nous notons $n_{\min}(s)$. De ce fait, nous pouvons introduire l'ensemble :

$$E^*(s) = \left\{ (n_1, n_2, \ldots, n_9) \in E(s) \;\middle|\; \sum_{i=1}^{9} n_i = n_{\min}(s) \right\}$$

ou encore :

$$E^*(s) = \left\{ (n_1, n_2, \ldots, n_9) \in \mathbb{N}^9 \;\middle|\; s = \sum_{i=1}^{9} n_i\, m_i \quad \text{et} \quad \sum_{i=1}^{9} n_i = n_{\min}(s) \right\}$$

$E^*(s)$ est appelé ensemble des **solutions optimales** de l'équation (1). Il s'agit d'un sous-ensemble de l'ensemble $E(s)$ des solutions de (1).

2) Unicité de la solution optimale

On peut montrer que, dans le système des euros, pour chaque somme s, il n'y a qu'une seule solution optimale ce qui signifie que $E^*(s)$ est un singleton ou encore que $\text{Card}(E^*(s)) = 1$.

Exercice

1. Quelles sont les solutions optimales pour $s = 1$, $s = 2$, $s = 3$, $s = 4$?

2. Supposons qu'il n'y ait qu'une seule solution optimale pour une somme $s > 0$ donnée. Dans quelles limites peuvent varier les nombres de pièces de 1 euros, 2 euros, 5 euros, etc ..., 500 euros qui définissent cette solution ?
3. On considère la somme $s+1$ obtenue en ajoutant 1 euro à s. Expliciter comment construire une solution optimale et montrer qu'il n'y a qu'une seule possibilité. Conclure.

* * * * * * * * * * * * *

Nous nous proposons maintenant d'étudier des algorithmes qui répondent aux deux questions suivantes :

1) Pour une somme s donnée, que vaut $n_{\min}(s)$?
2) Connaissant $n_{\min}(s)$, trouver la solution optimale, c'est à dire l'unique éléments de $E^*(s)$.

Il y a deux types d'algorithmes destinés à résoudre ce genre de problème :

- Les algorithmes gloutons vus en MPSI
- La programmation dynamique

3) Résolution par algorithme glouton

Un algorithme glouton résout le problème étape par étape :

- à chaque étape, il donne une solution optimale (ce qu'on peut appeler un optimum local)
- Il évalue ce qui lui reste à résoudre et passe à l'étape suivante.
- Les solutions données aux étapes précédentes ne sont jamais remises en question.

Ouvrir le fichier Capytale "Programmation dynamique : le problème du rendu de monnaie" et suivre les indications.

4) Résolution par programmation dynamique

On s'intéresse pour commencer au problème suivant : pour une somme s donnée, quel est le nombre minimal $n_{\min}(s)$ de pièces à rendre, c'est à dire le plus petit élément de $V(s)$.

On ne s'intéresse pas pour le moment à trouver la solution optimale qui réalise ce minimum (on la construira plus tard).

D'après la section **1)** on sait que $n_{\min}(s)$ est le plus petit élément de $V(s)$ défini par :

$$V(s) = \left\{ n = \sum_{i=1}^{9} n_i \;\middle|\; (n_1, n_2, \dots, n_9) \in \mathbb{N} \quad \text{et} \quad s = \sum_{i=1}^{9} n_i\, m_i \right\}$$

Exercice :

1. De quel(s) élément(s) est composé $V(0)$? Quel est alors son plus petit élément ?
2. Supposons maintenant que $s > 0$. Montrer qu'on a :

$$V(s) = \bigcup_{\substack{1 \leqslant k \leqslant 9 \\ m_k \leqslant s}} \{\, 1 + n \mid n \in V(s - m_k) \,\}$$

On pourra montrer la double inclusion en remarquant que si on dispose de n pièces de montant total $s > 0$ alors si l'une d'entre-elles vaut m_{k_0}, les $n' = n - 1$ autres pièces seront de montant total $s - m_{k_0}$.

3. En déduire que :

$$\boxed{\forall\, s > 0,\ n_{\min}(s) = \min_{\substack{1 \leqslant k \leqslant 9 \\ m_k \leqslant s}} (1 + n_{\min}(s - m_k)) \quad \text{et} \quad n_{\min}(0) = 0 \qquad (*)}$$

On pourra utiliser la propriété suivante (qu'on pourra chercher à démontrer) :

Si A et B sont deux parties non vides de $\mathbb{N}$ et que $a = \min A$ et $b = \min B$ alors :

$$\min(A \cup B) = \min(a, b)$$

ce qui peut se généraliser immédiatement par récurrence à toute réunion **finie** de parties de $\mathbb{N}$

L'équation $(*)$ peut être vue comme une sorte de récurrence sur $n_{\min}(s)$: l'expression de $n_{\min}(s)$ est donnée en fonction de celles de $n_{\min}(s - m_k)$; on peut dire que le problème $P =$ "trouver $n_{\min}(s)$" dépend de la résolution de k sous-problèmes $P_k =$ "trouver $n_{\min}(s - m_k)$"

De plus, chacun de ces sous-problèmes est *optimal* : on parle de **propriété de sous-structure optimale**.

Un programme résolvant l'équation $(*)$ peut être écrit aussi bien en version récursive qu'en version itérative. Revenons au fichier dans Capytale pour écrire ces programmes.

a) Version récursive naïve

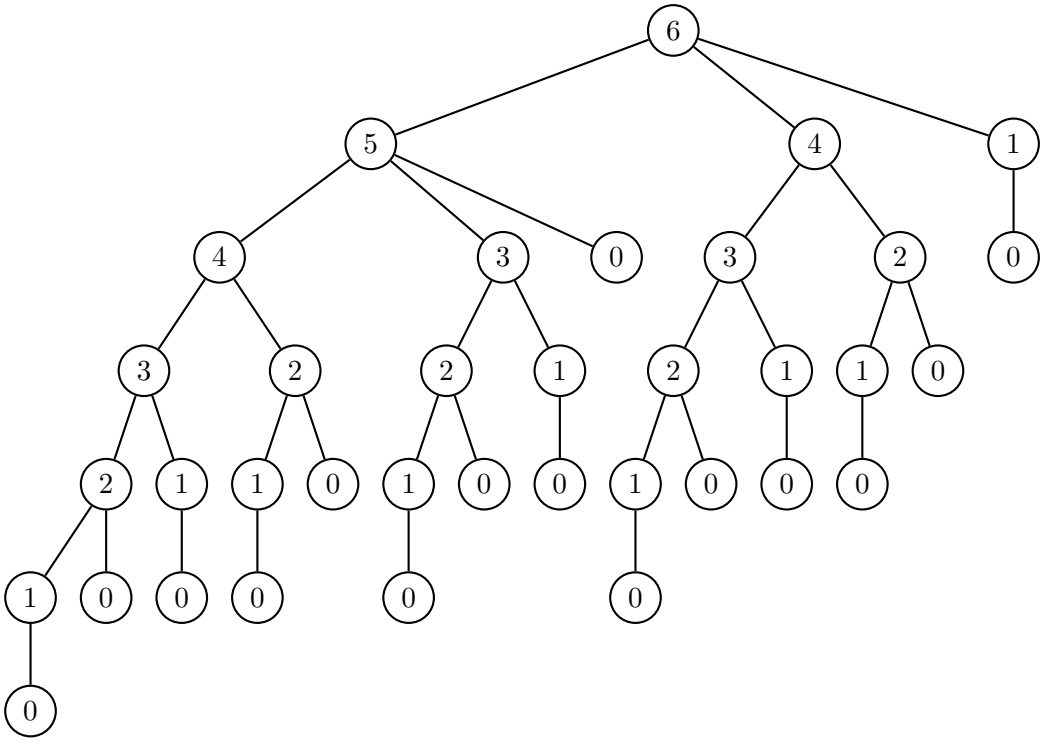

FIGURE 1 – Arbre des appels récursifs de n_min1(6)

En dessinant l'arbre des appels récursifs de n_min1(6) (cf. Figure 1) on observce qu'on appelle un **chevauchement** des sous-problèmes et une technique de mémoïsation est nécessaire pour accélérer les calculs.

b) Version récursive avec mémoïsation

On utilise un dictionnaire dont les éléments sont les couples $(s : n\_\min(s))$ pour stocker les résultats et ne pas avoir à les recalculer tout le temps.

c) Version itérative avec un tableau

Dans la programmation dynamique, il est toujours possible d'opter pour une version itérative de l'algorithme. On étudie dans le fichier Capytale une version qui utilise un tableau T de taille $s+1$: si i est l'indice des alvéoles de T alors $T[i]$ contient $n_{\min}(i)$.

5) Construction de la solution optimale

Pour le moment nous n'avons que $n_{\min}(s)$ pour tout $s \in \mathbb{N}$. On souhaite maintenant connaître la solution optimale, c'est à dire l'unique élément de $E^*(s)$ pour chaque somme s. Autrement dit :

Pour toute somme s à rendre, on souhaite obtenir la liste des pièces qui la constitue.

Dans ce but, il est nécessaire de compléter les algorithmes précédents en y ajoutant une *information supplémentaire* qui va nous permettre de **construire** cette solution optimale.

L'idée est que chacun des trois algorithmes précédents renvoie non seulement $n\_\min(s)$ mais aussi la valeur particulière m_p de m_k qui a rendu minimal $1 + \underset{\substack{1\leqslant k\leqslant 9\\ m_k\leqslant s}}{n_{\min}}(s-m_k)$ pour obtenir $n_{\min}(s)$.

On se reportera au fichier Capytal pour l'écriture de ces fonctions puis pour la construtions de la solution.

Reprise de la fin du TP info

Version itérative avec tableau T de taille $s+1$: i est l'indice des alvéoles de T et $T[i]$ contient $n_{\min}(i)$.

Si $s = 0$ renvoyer 0. Sinon créer un tableau T de taille $s+1$ et le remplir avec des zéros. Pour i variant de 1 à s faire : trouver le plus petit des $1 + T[i-m]$ lorsque m varie dans Monnaie de sorte que $m \leqslant i$ et mettre le résultat dans $T[i]$. Renvoyer $T[s]$.

```
def n_min3(s) :
   Monnaie = [1,2,5,10,20,50,100,200,500]
   if s == 0 :
      return 0
   else :
      T = (s+1)*[0]
      for i in range(1,s+1) :
         L = []
         for m in Monnaie :
            if m <= i :
               N = 1 + T[i-m]
               L.append(N)
            else :
               break
         T[i] = min(L)
      return T[s]
```

5) Construction de la solution optimale

Pour le moment nous n'avons que $n_{\min}(s)$ pour tout $s \in \mathbb{N}$. On souhaite maintenant connaître la solution optimale, c'est à dire l'unique élément de $E^*(s)$ pour chaque somme s. Autrement dit :

Pour toute somme s à rendre, on souhaite obtenir la liste des pièces qui la constitue.

Dans ce but, il est nécessaire de compléter les algorithmes précédents en y ajoutant une *information supplémentaire* qui va nous permettre de **construire** cette solution optimale.

L'idée est que chacun des 4 algorithmes précédents renvoie non seulement $n\_\min(s)$ mais aussi le montant particulier m_p qui a rendu minimal $1 + \underset{\substack{1\leqslant k\leqslant 9\\ m_k\leqslant s}}{n_{\min}}(s-m_k)$. Comment ?

```
def minimum(L) :       # Renvoie le minimum et l'indice de ce minimum
   mini, j = L[0], 0   # Initialisation
   n = len(L)
```

```
def n_min3_modif(s) :
   Monnaie = [1,2,5,10,20,50,100,200,500]
   if s == 0 :
      return
   else :
      T = (s+1)*[0]
      for i in range(1,s+1) :
         L = []
         for m in Monnaie :
            if m <= i :
               N = 1 + T[i-m]
               L.append(N)
            else :
               break
         T[i], indice = mininimum(L)
```

Exercice : écrire la version modifiée de l'algorithme récursif ave mémoïsation

La fonction `n_min3_modif` permet d'écrire la fonction `construire_rec` (version récursive) qui renvoie la liste des pièces rendues :

```
def construire_rec(s) :
   if s == 0 :
      return []
   else :
      n,m = n_min3_modif(s)
      liste_pieces = [m] + construire(s-m)
      return liste_pieces
```

Exercice : écrire une version itérative de cette fonction (corrigé sur Capytale)