

A woman with blue eyes and dark hair is looking upwards. Overlaid on her hand and face are glowing digital UI elements, including circular gauges with numbers like 20, 30, 75, and 3C, and various lines and icons. The text 'ALGO 3/5' is in large blue letters, and 'studi' is in smaller grey letters with a yellow 'i'.

ALGO 3/5

studi

Auteur : Sébastien Inion

<https://github.com/SebInfo/AlgoStudi3>

LE PROBLÈME DU RENDU DE MONNAIE

- ▶ Le problème du rendu de monnaie consiste à déterminer le nombre minimal de billets et/ou pièces nécessaires pour rendre une somme donnée.
- ▶ L'entrée est constituée de l'ensemble des valeurs des pièces et de la somme à rendre.
- ▶ La sortie est soit une liste de pièces, minimale pour atteindre cette somme, soit simplement le nombre minimal de pièces nécessaires (sans avoir la liste).
- ▶ Par exemple, la meilleure façon de rendre 7 euros est de rendre un billet de cinq et une pièce de deux, même si d'autres façons existent (rendre 7 pièces de un euro, par exemple).

LE RENDU DE MONNAIE

Définition [\[modifier \]](#) [\[modifier le code \]](#)

Les billets et pièces jouant ici le même rôle, on suppose pour simplifier qu'il n'y a que des pièces. Un *système de pièces* est alors un **n-uplet**

$$S = (c_1, c_2, \dots, c_n),$$

où c_i représente la valeur de la i^{e} pièce. On suppose que ces valeurs sont des entiers strictement croissants, et que $c_1 = 1$ (sinon certaines sommes ne peuvent être rendues — voir le [problème des pièces de monnaie](#)).

Étant donné un système S et un **entier positif** v , le problème de rendu de monnaie est le problème d'[optimisation combinatoire](#) qui consiste à trouver un n-uplet d'entiers positifs $T = (x_1, x_2, \dots, x_n)$ qui minimise

$$\sum_{i=1}^n x_i$$

sous la contrainte

$$\sum_{i=1}^n x_i c_i = v.$$

Ainsi, v représente la somme de monnaie à rendre et x_i le nombre de pièces c_i à utiliser. La quantité à minimiser est donc le nombre total de pièces rendues, la condition à vérifier traduisant simplement le fait qu'il faut bien rendre la somme v .

On note $M_S(v)$ le nombre minimal de pièces d'un système $S(v)$.

source : wikipédia

LE RENDU DE MONNAIE

Exemple [[modifier](#) | [modifier le code](#)]

Dans la [zone euro](#), le système en vigueur est, en mettant de côté les centimes d'euros :

$$S = (1, 2, 5, 10, 20, 50, 100, 200, 500).$$

Il y a par exemple six triplets de pièces (ou billets) de 1, 2 et 5 euros qui permettent de rendre 7 euros (les billets de 10 euros ou plus étant inutiles) :

$$(7,0,0), (5,1,0), (3,2,0), (1,3,0), (2,0,1), (0,1,1).$$

La solution au problème de rendu de monnaie $(S,7)$ est alors le triplet (x_1, x_2, x_3) qui minimise le nombre total $x_1 + x_2 + x_3$ de pièces rendues, soit $(0, 1, 1)$, c'est-à-dire une pièce de 2 euros et une de 5 (un billet). On a donc $M_{(1,2,5)}(7) = 2$.

source : wikipédia

ALGO GLOUTON

LE RENDU DE MONNAIE

- ▶ Ce problème est **NP-complet** (Un problème NP-complet est un problème qui n'admet pas d'algorithmes capables de trouver une solution en un temps polynomial) dans le cas général, c'est-à-dire difficile à résoudre.
- ▶ Cependant pour certains systèmes de monnaie dits canoniques, l'**algorithme glouton** est optimal, c'est-à-dire qu'il suffit de rendre systématiquement la pièce ou le billet de valeur maximale – ce tant qu'il reste quelque chose à rendre.
- ▶ Une monnaie canonique rend le minimum de pièces avec l'algo glouton

ALGO GLOUTON

- ▶ Il s'agit d'un algorithme qui prend des décisions étape par étape en choisissant à chaque étape la solution localement optimale à ce moment-là, sans se préoccuper des conséquences à long terme.
- ▶ L'idée fondamentale derrière un algorithme glouton est de faire les meilleurs choix locaux à chaque étape, en espérant que cela conduira à une solution globalement optimale.
- ▶ Important : parfois, ils peuvent donner des solutions sous-optimales ou incorrectes.

ALGO GLOUTON

- ▶ Dans le problème du rendu de monnaie, l'algorithme consistant à répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante est un **algorithme glouton**.
- ▶ Par exemple pour rendre 7 euros le minimum est 2 : une pièce de 5 plus une pièce de 2.
- ▶ Si l'on prend le système non canonique (1,3,4), l'algorithme glouton calcule $6 = 4+1+1$ alors qu'on peut rendre une pièce en moins en remarquant que $6 = 3+3$.
- ▶ Pour votre culture la livre sterling avant 1971 était une monnaie non canonique. De nos jours il n'en n'existe plus.

IMPLEMENTATION EN PYTHON

```
valeurs = [1,2,5,10,20,50,100,200,500]

def renduMonnaie(sommeARendre):
    liste = []
    indice = len(valeurs) - 1

    while sommeARendre > 0:
        piece = valeurs[indice] #glouton
        if piece > sommeARendre:
            indice -= 1
        else:
            liste.append(piece)
            sommeARendre -= piece

    return liste

print(renduMonnaie(7))
```

LA RÉCURSIVITÉ

LA RÉCURSIVITÉ

- Une fonction récursive est une fonction qui s'appelle elle-même.
- Prenons l'exemple classique de factorielle :

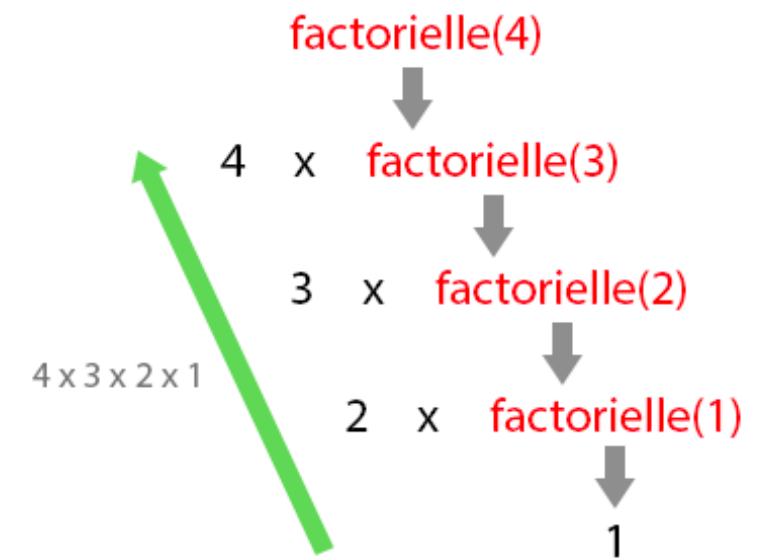
[illegible]

- ▶ On peut aussi dire que $5! = 5 * 4!$ et que $4! = 4 * 3!$ etc.
- ▶ On a ici une fonction récursive se terminant avec $0! = 1$

LA RÉCURSIVITÉ

► Algo :

```
fonction factorielle(n):  
    si n est égal à 0:  
        retourne 1  
    sinon:  
        retourne n multiplié par factorielle(n - 1)
```



► Ce qui donne naturellement en Python :

```
def factorielle(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorielle(n - 1)
```


FACTORIELLE SANS RÉCURSIVITÉ

► Algo :

```
fonction factorielle(n):  
    résultat = 1  
    pour i de 1 à n:  
        résultat = résultat * i  
    retourne résultat
```

► Ce qui donne en Python :

```
def factorielle(n):  
    resultat = 1  
    for i in range(1, n + 1):  
        resultat *= i  
    return resultat
```

AVANTAGES ET INCONVENIENTS

▶ Avantages :

- ▶ Le code paraît presque limpide et ressemble à une définition
- ▶ Pour certains problèmes (Tours de Hanoï par exemple) la récursivité est la seule issue raisonnable.

▶ Inconvénients :

- ▶ Légèrement plus lent que l'itératif
- ▶ Consomme un peu plus de mémoire ce qui est gênant lors d'appels très nombreux.

RENDU DE MONNAIE EN RÉCURSIF

- ▶ On cherche à déterminer, pour un ensemble P de pièces et une somme s **le nombre minimal de pièces** de P pour atteindre la somme s . On note $\text{pieces}_P(s)$ ce minimum. (le problème est le même que celui du début du cours)
- ▶ On décrit une formule récursive :
 - ▶ Si $s = 0$, alors $\text{pieces}_P(s) = 0$ de manière évidente.
 - ▶ Sinon, on peut rendre **n'importe laquelle** des pièces p de P si $p \leq s$. Si on a sélectionné p , il reste la somme $s - p$ à rendre : cette somme nécessite par hypothèse $\text{pieces}_P(s - p)$

RENDU DE MONNAIE EN RÉCURSIF

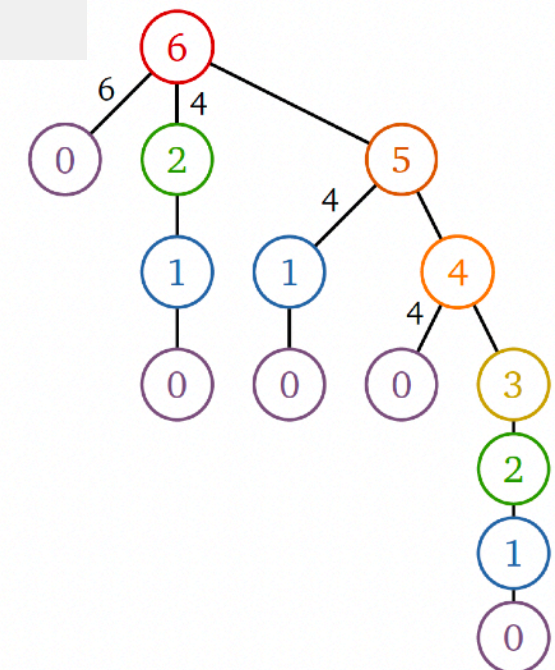
$$\text{pieces}_p(s) = \begin{cases} 0 & \text{si } s = 0, \text{ et} \\ 1 + \min\{\text{pieces}_p(s - p) : p \in P, p \leq s\} & \text{sinon.} \end{cases}$$

Entrées : Ensemble P d'entiers, entier s

Sortie : Nombre minimal de pièces de P dont la somme vaut s

- 1 Si $s = 0$: renvoyer 0
- 2 $n \leftarrow +\infty$
- 3 Pour chaque pièce $p \in P$:
- 4 Si $p \leq s$:
- 5 $n_p \leftarrow \text{RENDUNAIF}(P, s - p)$
- 6 Si $n_p \leq n$: $n \leftarrow n_p$
- 7 Renvoyer $1 + n$

RENDUNAIF($\{6, 4, 1\}, 6$)



RENDU DE MONNAIE EN RÉCURSIF

Entrées : Ensemble P d'entiers, entier s

Sortie : Nombre minimal de pièces de P dont la somme vaut s

```
1 Si  $s = 0$  : renvoyer 0
2  $n \leftarrow +\infty$ 
3 Pour chaque pièce  $p \in P$  :
4   Si  $p \leq s$  :
5      $n_p \leftarrow \text{RENDUNAIF}(P, s - p)$ 
6     Si  $n_p \leq n$  :  $n \leftarrow n_p$ 
7 Renvoyer  $1 + n$ 
```

```
def rendu_monnaie_nbrPiece(s, P):
    if s == 0:
        return 0
    n = float('inf')
    for p in P:
        if p <= s:
            np = rendu_monnaie_nbrPiece(s - p, P)
            if (np < n):
                n=np
    return 1+n
```

EXERCICES

- ▶ Transformer le programme précédent afin d'obtenir cette fois la liste des pièces et non plus le nombre de pièce.
- ▶ Tester avec un système non canonique.
- ▶ Est-ce que cela fonctionne mieux que les algorithmes glouton ?



**MERCI POUR
VOTRE ATTENTION**