Problème du rendu de monnaie

Chap. 23,01

1 Algorithme glouton

Les algorithmes gloutons forment une catégorie d'algorithmes permettant de parvenir à une solution pour un **problème d'optimisation** qui vise à *maximiser/minimiser une quantité* (plus court chemin (GPS), plus petite durée d'exécution, meilleure organisation d'un emploi du temps, etc.)

Le principe d'un algorithme glouton est le suivant :

- Résoudre un problème étape par étape ;
- À chaque étape, *faire le choix optimal de moindre coût* (de meilleur gain).

Le choix effectué à chaque étape n'est jamais remis en cause, ce qui fait que cette stratégie permet d'aboutir rapidement à une solution au problème de départ. C'est en ce sens que l'adjectif **greedy** (*glouton/avare*) caractérise cet algorithme : *il se termine rapidement (glouton) sans fournir beaucoup d'efforts (avare).*

2 Le problème du rendu de monnaie

2.1 Énoncé

Vous êtes commerçant et devez rendre de la monnaie à vos clients de *façon optimale*, c'est-à-dire avec **le nombre minimal de pièces et de billets**[1].

- On suppose que les clients ne vous donnent que des sommes entières en euros (pas de centimes pour simplifier) ;
- Les valeurs des pièces et billets à votre disposition sont : 1, 2, 5, 10, 20, 50, 100, 200 et 500. On suppose que vous avez autant d'exemplaires que nécessaire de chaque pièce et billet ;
- Dans la suite, afin de simplifier, nous désignerons par « pièces » à la fois les pièces et les billets.

2.2 Stratégie gloutonne

Un client nous achète un objet qui coûte 53 euros. Il paye avec un billet de 200 euros. On doit donc lui rendre 147 euros. Une façon de lui rendre la monnaie est de le faire avec un billet de 100, deux billets de 20, un billet de 5 et une pièce de 2.

Pour minimiser le nombre de pièces à rendre, il apparaît la stratégie suivante :

- On commence par rendre la pièce de plus grande valeur possible ;
- On déduit cette valeur de la somme (encore) à rendre ;

1. Version originale de ce document : http://info-mounier.fr/1nsi/seq10/algorithmes_gloutons.php

– On recommence, jusqu'à obtenir une somme nulle.

En procédant ainsi, on se rend compte que l'**on résout le problème étape par étape et qu'un choix optimal est fait à chaque étape** (*la pièce de plus grande valeur*). *Cette stratégie entre donc bien dans la catégorie des algorithmes gloutons.*

2.3 Implémentation en Python

Le corrigé se trouve à cette adresse : https://repl.it/@dlatreyte/rendudemonnaie.

1. Importer le module typing au début du fichier :

```
from typing import List
```

2. Préparer la fonction `main` suivante et étudier sa structure :

```
def main():
    # valeurs des pièces
    pieces = [1, 2, 5, 10, 20, 50, 100]

    prix = int(input("Quel est le prix ? "))
    client = int(input("Combien le client donne-t-il ? "))

    a_rendre = somme_a_rendre(prix, client)
    comment = pieces_a_rendre(a_rendre, pieces)

    reponse = ["{} piece(s) de {}".format(comment[i], pieces[i]) for i in
range(len(pieces))]
    print("Je dois rendre : {}".format(a_rendre))
    print("Je donne donc : {}".format(reponse))
```

3. Définir la fonction `somme_a_rendre` dont la spécification est :

```
def somme_a_rendre(prix: int, montant_client: int) -> int:
    """
    Détermine la somme à rendre à partir du prix et du montant donné par
    le client.
    """
```

4. Définir la fonction `pieces_a_rendre` dont la spécification est :

```
def pieces_a_rendre(somme: int, pieces: List[int]) -> List[int]:
    """
    Détermine les pièces (et leur nombre) à choisir pour rendre la somme
    passée en argument.
    Utilise la division euclidienne et le modulo de façon à pouvoir
    traiter tous les cas.
    """
```

5. Appeler la fonction `main`.

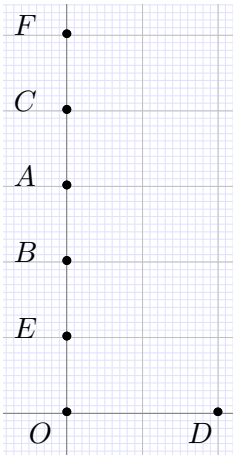
3 Un algorithme glouton conduit-il toujours à la solution optimale ?

La stratégie gloutonne consiste à trouver la **solution optimale locale** à chaque étape, dans l'espoir de trouver la **solution optimale globale**.

On peut se demander si cette stratégie donne nécessairement la meilleure solution globale, autrement dit si la solution globale obtenue est la meilleure ? Voici quelques exercices pour répondre à cette question.

Exercice 1. (Plus court chemin)

- On part du point O.
- On veut atteindre *en parcourant la distance la plus petite possible* tous les points : A, B, C, D, E et F.
- L'ordre de parcours des points n'a pas d'importance.



1. Tracer le chemin à suivre en adoptant une stratégie gloutonne.
2. La stratégie gloutonne est-elle la meilleure ? Si tel n'est pas le cas, tracer le chemin optimal.

Exercice 2. (Rendu de monnaie)

On doit rendre 8 euros à l'aide des pièces (imaginaires) : 6, 4 et 1 euros. On cherche à manipuler le moins de pièces possible.

1. Quelles pièces sont rendues en appliquant l'algorithme glouton ? Combien de pièces sont rendues ?
2. La solution précédente n'est pas la solution optimale. Trouver cette solution optimale.

Remarque. On peut montrer (ce n'est pas facile) que l'algorithme glouton retourne la solution optimale globale pour le rendu de monnaie... à la condition d'utiliser un système de monnaie tel que l'euro.

La stratégie gloutonne ne conduit pas toujours à la solution optimale d'un problème.

4 Un algorithme glouton conduit-il toujours à une solution ?

Exercice 3. (Rendu de monnaie)

On doit rendre 8 euros à l'aide des pièces (imaginaires) : 5 et 2 euros. On cherche à manipuler le moins de pièces possible.

1. Quelles pièces doivent être rendues si on applique un algorithme glouton ?

2. Conclusion ?

La stratégie gloutonne ne conduit pas toujours à la solution exacte d'un problème.

5 Pourquoi se contenter d'une stratégie gloutonne ?

Comme nous le verrons dans le document Chap. 23,02, *la stratégie gloutonne est utilisée dans le cas où la complexité des algorithmes donnant à coup sûr la réponse correcte est exponentielle* (pour le problème du sac à dos, $O(n) = 2^n$). De tels algorithmes sont très souvent inexploitables.

6 Code Python du programme principal

```
"""
Mise en œuvre de la stratégie gloutonne pour le
problème du rendu de monnaie.
"""
from typing import List


def somme_a_rendre(prix: int, client: int) -> int:
    """
    Détermine la somme à rendre à partir du prix et du montant donné par
    le client.
    """
    return client - prix


def pieces_a_rendre(somme: int, pieces: List[int]) -> List[int]:
    """
    Détermine les pièces (et leur nombre) à choisir pour rendre la somme
    passée en argument.
    Utilise la division euclidienne et le modulo de façon à pouvoir
    traiter tous les cas.
    """
    a_rendre = []
    indice_piece = len(pieces) - 1
    while indice_piece >= 0:
        rendu = somme // pieces[indice_piece]
        encore_a_rendre = somme % pieces[indice_piece]
        a_rendre.insert(0, rendu)
        somme = encore_a_rendre
        indice_piece -= 1
    return a_rendre
```

```
def main():
    # valeurs des pièces
    pieces = [1, 2, 5, 10, 20, 50, 100]

    prix = int(input("Quel est le prix ? "))
    client = int(input("Combien le client donne-t-il ? "))

    a_rendre = somme_a_rendre(prix, client)
    comment = pieces_a_rendre(a_rendre, pieces)

    reponse = ["{} piece(s) de {}".format(comment[i], pieces[i]) for i in range(len(pieces))]
    print("Je dois rendre : {}".format(a_rendre))
    print("Je donne donc : {}".format(reponse))


main()
```