Algorithmes gloutons

Introduction :

Un cambrioleur possède un sac à dos d'une contenance maximum de 30 Kg. Au cours d'un de ses cambriolages, il a la possibilité de dérober 4 objets A, B, C et D. Voici un tableau qui résume les caractéristiques de ces objets :

Caractéristiques des objets :

objet	A	B	C	D
masse	13 kg	12 kg	8 kg	10 kg
valeur marchande	700 €	400 €	300 €	300 €

Déterminez les objets que le cambrioleur aura intérêt à dérober, sachant que :

- **Tous les objets dérobés devront tenir dans le sac à dos (30 Kg maxi)**
- **Le cambrioleur cherche à obtenir un gain maximum.**

Ce genre de problème est un grand classique en informatique, on parle de problème d'optimisation. Il existe toujours plusieurs solutions possibles à un problème d'optimisation (dans le problème du sac à dos, on peut choisir toutes les combinaisons à partir du moment où la masse totale ne dépasse pas 30 kg), mais on ne cherche pas n'importe quelle solution, on cherche une solution dite optimale. Dans notre exemple on cherche le plus grand gain possible. Souvent, dans les problèmes d'optimisation, il n'existe pas une solution optimale, mais plusieurs solutions optimales, résoudre un problème d'optimisation c'est donc trouver une des solutions optimales.

Il existe différentes méthodes algorithmiques permettant de trouver une solution optimale à un problème d'optimisation : il peut, en effet, être intéressant "d'automatiser" la résolution des problèmes d'optimisation à l'aide d'algorithme.

En apparence, la solution la plus simple dans le cas du sac à dos serait d'écrire un algorithme qui teste toutes les combinaisons d'objets possibles et qui retient les solutions qui offrent un gain maximum. Dans notre cas précis, avec seulement 4 objets, cette solution pourrait être envisagée, mais avec un plus grand nombre d'objets, le temps de calculs, même pour un ordinateur très puissant, deviendrait trop important. En effet l'algorithme qui testerait toutes les combinaisons possibles aurait une complexité en temps en $O(a^n)$ avec a une constante et n le nombre d'objets. On parle d'une complexité exponentielle. Les algorithmes à complexité exponentielle ne sont pas efficaces pour résoudre des problèmes, le temps de calcul devient beaucoup trop important quand n devient très grand.

À la place de cette méthode "je teste toutes les possibilités", il est possible d'utiliser une méthode dite **gloutonne** (**greedy** en anglais).

1. Qu'est-ce qu'une méthode gloutonne ?

La résolution d'un problème d'optimisation se fait généralement par étapes : à chaque étape on doit faire un choix.

Par exemple, dans le problème du sac à dos, nous ajoutons les objets un par un, chaque ajout d'un objet constitue une étape: à chaque étape on doit choisir un objet à mettre dans le sac.

Le principe de la méthode gloutonne est de, à chaque étape de la résolution du problème, faire le choix qui semble le plus pertinent sur le moment, avec l'espoir qu'au bout du compte, cela nous conduira vers une solution optimale du problème à résoudre.

Autrement dit, on fait des choix localement optimaux dans l'espoir que ces choix mèneront à une solution globalement optimale (le "localement" signifie ici "à chaque étape de la résolution du problème").

Appliquons une méthode gloutonne à la résolution du problème du sac à dos :

- Sachant que l'on cherche à maximiser le gain, commençons par établir un tableau nous donnant la "valeur massique" de chaque objet (pour chaque objet on divise sa valeur par sa masse) :

Valeur massique des objets

objet	**A**	**B**	**C**	**D**
valeur massique	54 €/kg	33 €/kg	38 €/kg	30 /kg

- On classe ensuite les objets par ordre décroissant de valeur massique soit : A - C - B -D
- Enfin, on remplit le sac en prenant les objets dans l'ordre et en s'arrêtant dès que la masse limite est atteinte. C'est ici que se fait "le choix glouton", à chaque étape, on prend l'objet ayant le rapport "valeur-masse" le plus intéressant au vu des objectifs :
    - 1re étape : A (13 kg)
    - 2e étape : C (13+8=21 kg)
    - 3e étape : B (13+8+12=33 kg) => impossible, on dépasse les 30 kg.

Le sac est donc composé de 2 objets : A et C pour un montant total de 1000 € et une masse totale de 21 kg.

Cette méthode gloutonne peut être "automatisée", il est donc possible d'écrire un algorithme glouton (un algorithme qui est basé sur une méthode gloutonne) afin de trouver une solution au problème du sac à dos avec n'importe quelles valeurs (nombre d'objets, masse des objets, valeur des objets, masse maximum).

2. La solution trouvée ci-dessus est-elle optimale ?

On constate rapidement que la réponse est non, car le couple A+B permet d'atteindre une valeur de 1100 € pour une masse de 25 kg. Dans notre problème, la méthode gloutonne ne nous donne pas une solution optimale.

Plus généralement, il est important de bien comprendre qu'un algorithme glouton ne donne pas forcement une solution optimale.

3. Le problème du rendu de monnaie

Pour simplifier, nous considèrerons des sommes entières en euros. Nous avons à notre disposition un nombre illimité de pièces ou billets de :

- 1 €
- 2 €
- 5 €
- 10 €
- 20 €
- 50 €
- 100 €
- 200 €
- 500 €

Nous devons rendre la monnaie à un client à l'aide de ces pièces. **La contrainte est d'utiliser le moins de pièces possible.**

Le principe de l'algorithme consiste à répéter le choix de la pièce dont la valeur est la plus grande et qui ne dépasse pas la somme restante.

On dit qu'il s'agit d'un algorithme glouton car il choisit la plus grosse pièce à chaque étape, sans réfléchir à la suite !

Voici un algorithme possible qui prend en argument une somme à rendre et un tableau qui contient les différentes pièces et billets disponibles.

```
VARIABLES

pieces : tableau de nombres entiers (liste des pièces et billets disponibles)
choix : tableau de nombres entiers (liste qui affiche le types de pièces et/ou billets)
quantite : tableau de nombres entiers (liste qui affiche le nombre de pièces et/ou billets)
somme: nombre entier (somme à rendre)
i : nombre entier
n : nombre entier


DEBUT

i ← 1
n ← longueur(pieces)
```

```
pour i allant de 1 à n

      tant que somme >= pieces[i]
            somme ← somme - pieces[i]
            quantite[i] ← quantite[i] +1
            choix[i] ← pieces[i]
      fin tant que

retourner quantité, choix

FIN
```

Implémentez cet algorithme en python puis testez-le avec les sommes suivantes :

388 €, 210 €, 76 €, 33 €. Combien de pièces/billets avez-vous utilisés ?

4. Optimisation de l'algorithme

Dans le système de pièces européen, l'algorithme glouton donne toujours une solution optimale.

Mais ce n'est pas toujours le cas, les algorithmes gloutons ne donnent pas toujours une solution optimale. (Voir l'introduction avec l'exemple du sac à dos.)

Pour rester dans le cas du rendu de monnaie, considérons maintenant une liste de pièces fictives de 4 €, 3 € et 1 € et l'on souhaite rendre la somme de 6 euros.

Que retourne le programme précédent avec ces données ?

La solution obtenue est-elle optimale ?