Algorithmes gloutons

Certains problèmes peuvent se résoudre en déterminant toutes les solutions possibles.

Celles-ci existent en général et on est alors amener à trouver la (les) meilleure(s) solution(s), si on a un critère de comparaison distinguant une solution d'autre.

Prenons un jeu de 32 cartes classiques, chaque figure correspondant à un certain nombre de points, et on tire une main en contenant 8.

Objectif : déterminer la meilleure main.

On a $32 \times 31 \times 30 \times 29 \times 28 \times 27 \times 26 \times 25 = 424097856000$ mains différentes possibles.

On calcule la valeur de chacune des mains et on récupère la plus grande valeur et la (les) main(s) associée(s).

Cette méthode est basée sur la *force brute* : elle génère un coût algorithmique important (on est au-delà des coûts polynomiaux).

Dans ce genre de situation, on privilégie en général une méthode itérative moins coûteuse : on utilise un algorithme glouton.

Dans notre cas, on prend une carte, celle ayant la plus grande valeur, puis la suivante avec le même critère, et ceci jusqu'à la huitième.

1 Principe

On considère un **problème d'optimisation** dont on peut déterminer les solutions et pour lesquelles on associe une valeur.

La première approche *naïve* :

On cherche toutes les solutions ayant la valeur la plus intéressante, la valeur **optimale**.

La technique la plus simple consiste à énumérer toutes les solutions de manière **exhaustive**, puis de choisir la (les) meileure(s).

C'est une approche par **force brute** qui a un coût algorithmique trop important (en général, exponentiel).

Remarque : on retrouve cette complexité en cryptographie pour les mots de passe par exemple.

Une alternative est d'utiliser une méthode dite **gloutonne** :

— on va effectuer une succession de choix, chacun d'eux semblant être le meilleur sur le moment (dans notre exemple, choisir la carte restante de plus grande valeur) ;

— on résout donc un sous-problème à chaque itération en créant une solution partielle, appelée solution optimale ;

La méthode sera donc itérative et aboutira à une solution potentielle (dans notre exemple, une main avec une valeur optimale).

*La solution trouvée est-elle la **meilleure** ?*

On va voir que cela dépend.

Cette année, trois situations se résolvant par des algorithmes gloutons :

— Le rendu de monnaie (dans ce cours)

— Le voyageur de commerce (en TP)

— Le sac à dos (en TP)

2 Le rendu de monnaie

Vous êtes au marché, on doit vous rendre 9 €.

On a à notre disposition (pour faire simple) des pièces de 1 et 2 €, des billets de 5 €.

Voilà les solutions possibles :
— neuf pièces de 1
— sept pièces de 1 et une de 2
— cinq pièces de 1 et deux de 2
— trois pièces de 1 et trois de 2
— une pièce de 1 et quatre de 2
— un billet de 5 et quatre pièces de 1
— un billet de 5 et deux pièces de 1 et une de 2
— un billet de 5 et deux pièces de 2

On remarque ici que la valeur ne correspond pas à un critère de choix : on nous rend à chaque fois 9 €.

Par contre, pour le caissier le nombre de billets et de pièces rendus sont à minimiser : c'est la valeur à optimiser dans ce problème (c'est le dernier cas avec 3 pièces-billet).

De manière général, le problème est **pour un montant donné de construire une liste de billets et de pièces équivalent avec le minimum de pièces et billets**.

La résolution dépend maintenant du système monétaire, autrement dit les pièces et billets disponibles. On se place dans le cas où pour notre système, tout est disponible ce qui revient à une liste :

listeM = [200,100,50,20,10,5,2,1,0.5,0.2,0.1,0.05,0.02,0.01]

Comme dans l'exemple, le principe est de chercher la monnaie à rendre en commençant par la plus grande (sur le principe de la division) et on a l'algorithme suivant :

```
listeM = [200,100,50,20,10,5,2,1,0.5,0.2,0.1,0.05,0.02,0.01]
resultat=[0]*longueur(listeM)
i=0
Tant que montant>=0.01 faire
        resultat[i]=quotient entier de montant par listeM[i]
        montant=montant-resultat[i]*listeM[i]
        i=i+1
Fin Tant que
retourner resultat
```

Si on reprend le cas du montant de 9 euros :
— resultat[i] = 0 pour i de 0 à 4 puisque 9<10
— on divise 9 par 5 : on obtient 1 quotient entier et resultat[5]=1
— il reste 4 euros
— on divise 4 par 2 : on obtient 2 quotient entier et resultat[6]=2
— il reste 0
— donc la condition de la boucle n'est plus vérifiée
— on obtient resultat = [0,0,0,0,0,1,2,0,0,0,0,0,0,0]

Remarque : la solution obtenue est optimale, on ne peut pas dans ce système en trouver de meilleure. Cependant, si on change le système monétaire, on peut avoir des résultats non optimaux (voir exercice 1).

.....

Exercice 1 : on crée dans notre système un billet de 7 euros. Montre qu'avec un montant de 14 €, la solution de l'algorithme n'est plus optimale

Exercice 2 : crée une fonction Python correspondant à l'algorithme et qui calcule en plus le nombre de monnaies rendues. Tu documenteras ta fonction en y ajoutant 3 exemples dont un avec des centimes.