**NSI**

2

08

**Automne**



ALGORITHMES GLOUTONS

 [Stéphane BEAUET – Frédéric PEURIERE]

 **Un problème d´optimisation**

Une grande quantité de problèmes à traiter par les algorithmes se résume à une **question d´optimisation**: plus courte distance entre deux points, le plus court trajet entre deux villes, à pied en vélo ou en voiture... Une première méthode pour trouver une solution à un problème qui comporte une grande quantité de solutions possibles consiste à **explorer toutes les possibilités**, les comparer et ne retenir que la meilleure. Le problème est que cette méthode (dite par "*force brute*") nécessite des temps de calculs insurmontables.

Les **algorithmes gloutons** proposent de trouver la solution optimale en décomposant le problème en différentes étapes et de retenir le meilleur choix à chaque étape. Est ce que le choix final obtenu est optimal? Parfois mis pas toujours...

 **UN EXEMPLE: LE rendu de monnaie**

Dans le système de la zone euro il existe 8 pièces différentes que nous pouvons représenter par un tuple: **S=(1,2,5,10,20,50,100,200).**100 et 200 représentent les pièces de 1 et 2 euros.

****

1. **Comment rendre huit centimes avec le moins de pièces possibles?**

Cherchons d´abord à connaître toutes les combinaisons possibles pour **rendre 8 centimes** avec des pièces. Nous n´avons besoin que de pièces de 1,2 ou 5 centimes. Considérons donc le tuple: **S=(1,2,5)**.

Cherchons tout d´abord à connaître **toutes les combinaisons possibles** de somme d´argent que l'on peut rendre avec des pièces de:

 *- 1 centime, dont le nombre peut varier de 0 à 8.
 - 2 centimes, dont le nombre peut varier de 0 à 4.
 - 5 centimes, dont le nombre peut varier de 0 à 1.*

 a) Déterminez par le calcul le nombre de combinaisons possibles:

……………………………………………………

Ecrivons maintenant le code PYTHON permettant d'afficher toutes ces combinaisons et testez-le dans Thonny (fichier monnaie1.py sur le site).
*Une pièce de 2 centimes correspond par exemple à S[1].*

S=(1,2,5)

**for** i **in** range (9): **#de 0 à 8 pour les pièces de 1 centime**

    **for** j **in** range (5): **#de 0 à 4 pour les pièces de 2 centimes**

        **for** k **in** range (2): **#de 0 à 1 pour les pièces de 5 centimes**

            somme=i\*S[0]+j\*S[1]+k\*S[2]

            **print**([i,j,k])

b) Modifiez ce code afin qu´il affiche toutes les combinaisons possibles pour **rendre 8 centimes** et testez-le dans Thonny.

……………………………………………………

……………………………………………………

……………………………………………………

……………………………………………………

……………………………………………………

……………………………………………………

……………………………………………………

……………………………………………………

……………………………………………………

c) Combien existe-t-il de solutions? ………………………………

d) Quelle est la solution optimale? ………………………………

1. **Généralisation du problème de rendu de monnaie**

On nous demande d´écrire un algorithme pour une machine à café qui doit rendre la monnaie avec un nombre minimal de pièces.

Nous allons chercher, par valeur décroissante en partant de la pièce qui a la plus grande valeur, la première pièce qui a une valeur inférieure ou égale à la somme à rendre que nous appellerons r.
On prend cette pièce, on retranche sa valeur v à r. On recommence en partant de la pièce prise en cherchant celle qui a une valeur inférieure ou égale à la nouvelle somme à rendre r-v.

La fonction **monnaie()** peut s´écrire comme ceci (fichier monnaie2.py sur le site):

1. Seuro=(1,2,5,10,20,50,100,200)
2. **def** monnaie (S,r):
3. n=len(S)
4. solution=n\*[0] #crée la liste: [0,0,0,0,0,0,0,0] si n=8
5. **for** i **in** range(n-1,-1,-1): # de n-1 à 0 compris
6. v=S[i]
7. **while** r>=v:
8. r=r-v
9. solution[i]=solution[i]+1
10. **return** solution

a) Ecrire l´instruction à entrer dans la console pour obtenir le jeu de pièce optimal pour **rendre 77 centimes**. Faites le test dans Thonny et écrivez le résultat:

………………………………...............................

b) Vérifiez avec notre algorithme le résultat de l´activité précédente (rendu de 8 centimes). Ecrivez la solution:

………………………………...............................

La stratégie employée ici est gloutonne car on recherche **la solution optimale à chaque tour de boucle** (la plus grande pièce inférieure à la somme restant à rendre).

Mais la solution trouvée est-elle globalement optimale? On peut prouver que c´est le cas dans le système de pièce européen.

🖙 Dans le Royaume Uni des années 60 on trouvait des jeux de pièces de 1 penny, 3 pence, 4 pence et 6 pence. Ce système est représenté par le tuple: Spenny=(1,3,4,6)

c) Quelle solution donne notre algorithme pour rendre **8 pence** par exemple?

 ………………………………..............................................

d) Quelle est la solution optimale (le moins de pièces possible)?

 ………………………………..............................................

Un algorithme glouton ne permet pas toujours de trouvez une solution globalement optimale. Pour y remédier, on a recours à la **programmation dynamique**. Elle est étudiée en terminale.