Chapitre 7 : Algorithme glouton

Les algorithmes gloutons suivent une démarche simple : lorsqu'à chaque étape d'un programme, un choix est à faire, c'est le choix optimal à ce moment qui est fait (en générale, cette meilleure solution doit également satisfaire une ou des contraintes). Attention cependant, cette somme de choix locaux optimaux n'aboutit pas forcément à une démarche globale optimale.

I- Problème du rendu de monnaie

a) Position du problème :

- Le rendu r n'utilisera que des pièces de valeurs croissantes $S = (p_0, p_1, p_2, \dots p_{n-1})$ où p_i est un entier représentant la valeur en centime d'une pièce avec $p_0 = 1$.
- Le problème du rendu de monnaie consiste donc à trouver une liste d'entiers positifs $[x_0, x_1, x_3, \dots, x_{n-1}]$ qui vérifie $r = x_0p_0 + x_1p_1 + \cdots x_{n-1}p_{n-1}$ (cette condition sera la contrainte à vérifier)
- $\sum_{i=0}^{i=n-1} x_i$ donne le nombre de pièces à utiliser et va fixer le choix optimal (ici un nombre minimum de pièces à rendre)

b) Principe de l'algorithme :

On cherche, par valeur décroissante en partant de la pièce qui a la plus forte valeur, la 1e pièce qui a une valeur inférieure ou égale à r. On prend cette pièce, on retranche sa valeur v à r et on recommence en partant de la pièce prise en cherchant celle qui a une valeur inférieure ou égale à $r - v$.

```
def monnaie(p,r):
    n=len(p)
    reste=r
    x=n*[0]
    i=n-1
    while reste>0:
        while reste-p[i]<0:
            i=i-1
        x[i]=x[i]+1
        reste=reste-p[i]
    return x
```

L'algorithme glouton en respectant la contrainte et en proposant un rendu avec un minimum de pièces fournit la solution optimale.

II- Problème du sac à dos

a) Position du problème

- On considère un ensemble de n objets $objet_i$ de masse m_i et de valeur v_i :

```
tab = [["objet1",4,3],
       ["objet2",3,2],
       ["objet3",1,1],
       ["objet3",9,4]]
```

L'objet $objet_i$ est représenté par une liste du type ["objeti", v_i, m_i]. On utilise les trois fonctions suivantes (prenant un objet en argument):

```
def valeur(objet):
    return objet[1]
def masse(objet):
    return objet[2]
def rapport(objet):
    return objet[1]/objet[2]
```

- Il s'agit d'emporter dans son sac l'ensemble d'objets qui a la plus grande valeur (solution optimale) sachant que le sac supporte une masse maximale M (contrainte avec $m_i < M$).
- L'algorithme glouton consiste, à chaque étape, à choisir parmi des objets celui qui représente le choix optimal. Soit m_i la masse de cet objet, l'algorithme continu parmi les objets dont la masse est inférieure à $M - m_i$.
- Il y a trois manières de définir le meilleur choix. Parmi les objets qui n'ont pas été encore pris, soit on choisit un objet qui a la valeur maximale, soit un objet qui a la masse minimale, soit un objet qui a le rapport valeur/masse maximal

b) Principe de l'algorithme

Nous définissons une fonction *glouton* qui prend en argument une liste d'objets *tab*, une masse maximale M, le type de choix optimal.

- Cette fonction trie de manière décroissante la liste suivant le type de choix avec la fonction *sorted*.
- On parcourt la liste triée et ajoutons les noms des objets un par un tant que la masse totale ne dépasse pas M
- La valeur totale et le poids total sont stockés dans deux variables *valeur* et *poids*

```
def glouton(tab,M,choix):

    copie=sorted(tab,key=choix,reverse=True)#choix est une fonction
    #qui s'applique à chaque élément de tab
    #c'est ce qui fixe le choix d'optimisation
    reponse=[]
    valeur=0
    masse=0
    i=0
    while i<len(tab) and masse<=M:
        nom,valeur_i,masse_i=copie[i]
        if masse +masse_i <=M:
            reponse.append(nom)
            masse=masse+masse_i
            valeur=valeur+valeur_i
        i=i+1
    return reponse, valeur, masse

print(glouton(tab,8,valeur))#renvoie la solution optimale
print(glouton(tab,8,masse))#renvoie la solution optimale
print(glouton(tab,8,rapport))#ne renvoie pas la solution optimale
```

Commenté [AM1]: Avec de euros :
$S = (1,2,5,10,20,50,100,200,500)$
Pour rendre 8 centimes, il existe plusieurs possibilités
[8,0,0], [3,0,1], [2,3,0], [1,1,1]*et* [0,4,0]
Voici une fonction qui renvoie tous les triplets pour $r = 8$

```
def test():
    p=(1,2,5)
    for i in range(9):
        for j in range(5):
            for k in range(2):
              s=i*p[0]+j*p[1]+k*p[2]
              if s==8:
                  print ([i,j,k])
```

Ici le choix optimal est [1,1,1]

Commenté [AM2]: Si les pièces disponibles sont (1,3,4) et que $r = 6$ alors l'algorithme glouton propose [2,0,1] alors que la solution optimale est [0,2,0]. L'algorithme glouton fournit la solution optimale qu'avec un nombre de pièces suffisants.

Commenté [AM3]: On pourra retenir que les algorithmes gloutons peuvent ne pas fournir un résultat global optimisé. Avec le choix basé sur le rapport valeur/masse, il est tout de même possible d'obtenir un résultat final optimisé en utilisant un processus récursif qui couvre, lors de ses différents appels, l'ensemble des combinaisons possibles : l'analyse n'est alors plus locale (ce n'est plus un algorithme glouton).