AGR. Algorithmes. Gloutons

I/ Introduction

Un problème d'optimisation a deux caractéristiques : une fonction que l'on doit maximiser ou minimiser et une série de contraintes que l'on doit satisfaire.
Exemple : Quel est le chemin le plus court entre Lille et Toulon sachant que l'on doit passer par Dijon ?

On pourrait essayer d'écrire un algorithme qui énumère toutes les possibilités afin de sélectionner la meilleure par simple comparaison. Cette technique, appelée **force brute**, est rarement utilisable du fait de sa complexité très élevée (exponentielle le plus souvent).
Un algorithme **glouton** est souvent une solution intéressante mais pas toujours optimal.

Pour comprendre cette notion d'algorithme glouton, il faut expliciter la notion de **choix optimal**. Un choix peut être **globalement** optimal, c'est-à-dire que c'est le meilleur de tous ou seulement **localement** optimal, c'est le meilleur choix mais parmi un ensemble restreint de choix.
A chaque étape exécutée, l'algorithme glouton va choisir le meilleur choix parmi plusieurs propositions.

A savoir : un choix glouton est seulement localement optimal. Une fois le choix effectué, il ne sera plus considéré.

II/ Problème du sac à dos

Pour appliquer une stratégie gloutonne, il faut définir le **concept** de **meilleur choix**.
Dans le problème du sac à dos, on dispose de *n* objet, chacun ayant une masse et une valeur mais on peut ne porter qu'une masse totale maximale dans ce sac.

Il y a trois manières de définir un meilleur choix :

- On trie les objets en fonction de leur valeur décroissante.
- On trie les objets en fonction de leur masse croissante.
- On trie les objets en fonction du rapport valeur/masse décroissant.

En fonction du meilleur choix privilégié, l'algorithme glouton consiste à choisir parmi les objets celui qui représente le choix optimal.
On parle de variante entière si l'on est obligé de choisir l'objet en entier ou de variante fractionnaire si l'on peut n'en prendre qu'une partie (par exemple des objets en poudre ou liquide). Un algorithme glouton donne une solution optimale à ce dernier cas.

Exemple : on suppose qu'un sac à dos peut contenir jusqu'à 15 kg d'objet. Voilà un tableau qui regroupe les caractéristiques des objets disponibles (*valeur* en euro et *masse* en kg).

Objets	Valeur	Masse	Valeur / Masse
Objet 1	126	14	9
Objet 2	32	2	16
Objet 3	20	5	4
Objet 4	5	1	5
Objet 5	18	6	3
Objet 6	80	8	10

Un objet est représenté par une liste du type [‘Numéro objet’, valeur, masse]. L’objet 1 aura alors comme liste celle-ci : [‘Objet 1’, 126, 14].

On définit maintenant :

- Trois fonctions *valeur(obj)*, *masse(obj)* et *rapport(obj)* qui vont représenter chacune un « meilleur choix » pour l’objet passé en argument dans le sens décroissant.
- Une fonction *glouton* qui va trier la liste d’objet selon le critère « meilleur choix » choisi. On utilise la fonction *sorted* pour l’effectuer.
- La liste d’objets issus du tableau ci-dessus.

Les trois fonctions valeur(obj), masse(obj) et *rapport(obj)*

```python
# Choix de la valeur
# Plus la valeur de l'objet est élevée, plus il est intéressant
def valeur(obj) :
    return obj[1]

# Choix par l'inverse de la masse
# Plus un objet est léger, plus il est intéressant
def masse(obj) :
    return 1/obj[2]

# Choix par le rapport valeur/masse
# Plus le rapport est élevé, plus l'objet est intéressant
def rapport(obj) :
    return obj[1]/obj[2]
```

La fonction glouton

```python
# Implémentation de la stratégie gloutonne
# A noter que 'choix' est le prototype d'une fonction
def glouton(liste_obj, masse_max, choix) :
    liste_triee = sorted(liste_obj, key = choix, reverse = True)
    resultat = []
    valeur = 0
    masse = 0
    i = 0 # Compteur pour l'indice de la liste

    # Tant qu'il reste des objets et que la limite (masse)
    # n'est pas atteinte
    while i < len(liste_obj) and masse < masse_max :
        nom, val, mas = liste_triee[i]
        if masse + mas <= masse_max :
            resultat.append(nom)
            masse += mas
            valeur += val
        i += 1

    return resultat, valeur
```

Liste d'objets et résultats obtenus

```
# Liste d'objets
objets = [ ['Objet 1', 126, 14], ['Objet 2', 32, 2], ['Objet 3', 20, 5],
         ['Objet 4', 5, 1], ['Objet 5', 18, 6], ['Objet 6', 80, 8] ]

# Test des différents "meilleur choix"
print("Choix par valeur : ", glouton(objets, 15, valeur))
print("Choix par masse : ", glouton(objets, 15, masse))
print("Choix par rapport : ", glouton(objets, 15, rapport))
```

```
Choix par valeur :  (['Objet 1', 'Objet 4'], 131)
Choix par masse :  (['Objet 4', 'Objet 2', 'Objet 3', 'Objet 5'], 75)
Choix par rapport :  (['Objet 2', 'Objet 6', 'Objet 4'], 117)
```

Le critère « valeur » est ici le plus intéressant. Mais ce n'est pas la meilleure solution, on peut par exemple choisir les objets 2, 3 et 6 pour une valeur totale de 132 euros et une masse de 15 kg.

III/ Problème du rendu de monnaie

Dans un système monétaire, on dispose de pièces (ou billets) d'une certaine valeur. Le but de rendre la monnaie en utilisant un minimum de pièces.

Si on considère le système de la zone euro, on dispose de pièces de valeurs (en centimes d'euros) suivantes :
S = {1, 2, 5, 10, 20, 50, 100, 200, 500 } où 100, 200 et 500 représentent respectivement des pièces de 1, 2 et 5 euros.

Si on souhaite rendre la monnaie de 8 centimes, les pièces pouvant être rendues sont de 1,2 ou 5 centimes d'euro. Il y a plusieurs solutions, en voici quelques-unes :

- 8 pièces de 1 centime.
- 4 pièces de 2 centimes.
- 1 pièce de 5 centimes, 1 pièce de 2 centimes, 1 pièce de 1 centime.
- 2 pièces de 2 centimes et 4 pièces de 1 centime.
- Etc.

Le triplet qui minimise le nombre de pièces à donner est [1, 1, 1] avec 3 pièces en tout.

A savoir : Le principe glouton du rendu de monnaie consiste à chaque fois à rendre la pièce (ou billet) du plus grand montant possible.

Exemple : pour rendre 63 euros, on rend d'abord un billet de 50 euros, puis de 10 euros, une pièce de 2 euros et enfin 1 pièces de 1 euro.

Voici le programme glouton du rendu de monnaie correspondant en langage Python

```
def rendu_monnaie(systeme,reste_a_rendre) :
    i = len(systeme) - 1
    piece_rendues = len(systeme)*[0]

    while reste_a_rendre > 0 :  # Tant qu'il reste de la monnaie à rendre
        while systeme[i] > reste_a_rendre : # Tant que l'on peut rendre cette pièce
            i -= 1
        piece_rendues[i] += 1
        reste_a_rendre -= systeme[i]

    return piece_rendues
```

```
syst = (1, 2, 5, 10, 20, 50, 100, 200, 500)
print(rendu_monnaie(syst,8))
```

```
[1, 1, 1, 0, 0, 0, 0, 0, 0]
```

La stratégie gloutonne propose ici la meilleure solution pour le rendu de monnaie sur 8 centimes.

Dans le cas général, avec un système différent de celui montré en exemple, c'est un problème complexe à résoudre. Mais dans presque tous les systèmes de monnaie, l'algorithme glouton est optimal : ce système est dit **canonique**.

Le système décimal s'est imposé dans tous les domaines en France sous Napoléon Bonaparte. Le système monétaire en est devenu aussi un et sera exporté dans toute l'Europe sauf en Angleterre où en trouve encore dans les années 1960 une multitude de pièces. Ce n'est qu'en 1971 que le Royaume-Uni a reformé son système de monnaie.

En savoir plus :
Système monétaire décimal européen :
https://omnilogie.fr/O/Livre,_shilling,_penny..._le_syst%C3%A8me_mon%C3%A9taire_anglais_avant_la_d%C3%A9cimalisation