ALGORITHMES GLOUTONS

Un problème d'optimisation classique consiste à maximiser ou minimiser une fonction tout en respectant une ou plusieurs *contraintes*. Prenons un exemple : on veut passer par 3 villes A, B, C (c'est la contrainte) en parcourant un minimum de distance (fonction à minimiser) pour économiser l'essence. On part du point O, cf le schéma ci-dessous (1 graduation = 1 km) :

Pour trouver le trajet optimal, on peut tester tous les trajets possibles et prendre le meilleur. On trouve que c'est $O \to B \to A \to C$, qui permet de passer par toutes les villes en parcourant seulement 9 km. Cependant, pour un problème plus compliqué, tester tous les cas possibles relève de l'impossible : le coût en calcul est trop grand.

L'objectif d'un *algorithme glouton* est d'obtenir une solution rapidement, mais cette solution n'est pas forcément optimale. À chaque étape de l'algorithme, on lui propose un ensemble de choix : un algorithme glouton va prendre le meilleur choix « à court terme ». Dans notre exemple, l'algorithme glouton va choisir à chaque étape d'aller à la ville la plus proche selon la position actuelle. Le trajet sera donc $O \to A \to B \to C$, pour un total de 11 km. Ainsi, *faire une suite de choix optimaux à court terme ne conduit pas forcément à la solution optimale sur le long terme.* Notons toutefois que si C était à gauche de B, l'algorithme glouton donnerait la solution optimale.

1 Problème du rendu de monnaie

Un client paye des marchandises avec un montant plus élevé que le prix réel. Il faut alors rendre une certaine somme. Comment rendre la monnaie en utilisant le minimum de pièces ?

La réponse dépend de la valeur des pièces que l'on peut utiliser. On appelera cela un « système » de pièces et on le représente par un n-uplet $P = (p_0, p_1, \cdots, p_{n-1})$, où p_i représente la valeur de la pièce d'indice i. Dans le système de la zone euro, par exemple, nous avons en centimes les pièces suivantes :

$$P_{euro} = (1, 2, 5, 10, 20, 50, 100, 200, 500)$$

où 100, 200 et 500 représentent respectivement les pièces de 1, 2 et 5 euros.

Par exemple, si on doit rendre 8 euros, la solution optimale est de rendre une pièce de 5 euros, une de 2 euros et une de 1 euro, soit trois pièces. Une solution non optimale serait de rendre quatre pièces de 2 euros.

Le problème du rendu de monnaie consiste à rendre une somme d'argent r avec le minimum de pièces. Pour tout i, on note x_i le nombre de pièces rendues de valeur p_i. Le problème consiste donc à déterminer une n-liste $[x_0, \cdots, x_{n-1}]$ de sorte que :

- On rende une somme égale à r, donc il faut que .....
- On minimise le nombre de pièces utilisées, càd on minimise .....

Résoudre ce problème pour un système de pièce P quelconque n'est pas trivial. C'est pour cela que des solutions gloutonnes peuvent être intéressantes en premier lieu.

Solution gloutonne. Un algorithme glouton va choisir les pièces une par une : on rendra la pièce de plus haute valeur qui est juste en-dessous de la valeur restante. Par exemple pour rendre une somme $r = 19$ centimes :

1. La pièce de plus haute valeur avant 19 est 10. On la rend une fois. Il reste $19-10=9$ centimes à rendre.
2. La pièce de plus haute valeur avant 9 est 5. On la rend une fois. Il reste $9-5=4$ centimes à rendre.
3. La pièce de plus haute valeur avant 4 est 2. On la rend une fois. Il reste $4-2=2$ centimes à rendre.
4. La pièce de plus haute valeur avant 2 est 2. On la rend une fois. Il reste $2-2=0$ centimes à rendre. Fini.

Dans cet exemple, on retourne ainsi 2 pièces de deux centimes, 1 pièce de cinq centimes et 1 pièce de dix centimes. On attend donc que l'algorithme retourne `[0,2,1,1,0,0,0,0,0]`.

Exercice 1. Compléter le script `monnaie(p,r)` ci-dessous qui retourne la solution de l'algorithme glouton pour un système de pièces `p` et une somme `r` à rendre. Tester si `monnaie( (1,2,5,10), 19)` retourne bien `[0,2,1,1]`.

```
def monnaie(p,r):
    n = len(p)
    sol = n*[0]     # stockera la solution
    while r>0:
        # tant qu'il reste de l'argent à rendre (r>0)
        # on détermine la pièce avec la plus grande valeur ≤ r
        # puis on rend cette pièce : on met à jour sol et r
    return sol
```

Exercice 2. Tester différentes valeurs de r avec $p = (1,2,5,10,20,50)$. Est-ce que l'algorithme glouton vous semble optimal ?

Jusqu'en 1971, le système monétaire utilisé en Angleterre comportait une multitude de pièces : 1, 3, 4, 6 pence, etc.

Question 1. On veut rendre 8 pence avec ce système. Que donne la solution gloutonne ? Vous semble-t-elle optimale ?

2 Stations d'essence

Un automobiliste part en vacances et doit parcourir un long trajet. Il prend la route avec le plein de carburant. Son véhicule peut parcourir une distance maximale d_{max} avec un plein. La route empruntée comporte n stations-services. La première est à une distance d_0 du départ, la deuxième est à une distance d_1 de la première, et ainsi de suite. Le point d'arrivée est considéré comme une station, et est à une distance d_{n-1} de la station précédente.

L'automobiliste pourra effectuer son trajet si et seulement si on a toujours $d_i \leq d_{max}$. Nous supposons que cette condition est remplie. **L'objectif de l'automobiliste est de s'arrêter le minimum de fois dans une station-service.**

Un algorithme glouton va, lorsque le plein est fait, parcourir la plus grande distance possible sans faire le plein. La valeur d_{max} étant fixée, on va aller à la dernière station-service qui est atteignable sans faire le plein (s'il y a d'autres stations avant, on ne s'y arrêtera pas).

Par exemple, si $[d_0, d_1, \cdots, d_{n-1}] = [2,2,3,1,5,1]$ et $d_{max} = 5$, alors :

- On va jusqu'à la station 1 car $d_0 + d_1 = 4 \leq d_{max}$ mais $d_0 + d_1 + d_2 = 7 > d_{max}$. On fait le plein à la station 1.
- Partant de la station 1, on va jusqu'à la station 3 car $d_2 + d_3 = 4 \leq d_{max}$ mais $d_2 + d_3 + d_4 = 9 > d_{max}$. On fait le plein à la station 3.
- Partant de la station 3, on va jusqu'à la station 4 car $d_4 = 5 \leq d_{max}$ mais $d_4 + d_5 = 6 > d_{max}$. On fait le plein à la station 4.
- Partant de la station 4, on va jusqu'à la station 5 (la dernière, c'est l'arrivée) car $d_5 = 1 \leq d_{max}$. Fin de l'algorithme.

On s'est arrêté aux stations 1, 3, 4, 5. Sur cet exemple, l'algorithme doit retourner :

```
>>> essence( [2,2,3,1,5,1] , 5 )
[1,3,4,5]
```

Exercice 3. Compléter le programme ci-dessous et le tester avec l'exemple ci-dessus. *Note : vous pouvez aussi coder l'algorithme d'une autre manière.*

```
def essence(dist,dmax):
    n = len(dist)
    stations = []  # numéros des stations où on s'est arrêté
    i = -1         # n° de la dernière station où on est passé, en
       s'arrêtant ou non (-1 est le point de départ)
    res = dmax     # on a assez d'essence dans le réservoir pour parcourir
       dmax

    while i!=n-1: # tant qu'on n'a pas atteint la dernière station,

        j = i  # numéro des stations où l'on va passer depuis i

        while ... :  # on vérifie qu'on peut aller jusqu'à la station j
            j = j+1  # on passe à la station j
            ...      # on met à jour res

        i = j  # on s'arrête à la station j
        ...    # on met à jour stations et res
        ...
    return stations
```

Question 2. L'algorithme glouton vous semble-t-il optimal ?

Idée de la preuve d'optimalité de l'algorithme glouton. On peut montrer qu'effectivement, l'algorithme glouton fournit toujours une solution optimale. L'heuristique de la preuve est la suivante. On considère :

- Une solution $g := [g_0, g_1, \ldots, g_{m-1}, g_m]$ obtenue avec l'algorithme glouton.
- Une solution optimale quelconque $S := [S_0, S_1, \ldots, S_p]$.

Comme S est optimale, on a nécessairement $p \leq m$. Prouvons que $p = m$, càd que la stratégie gloutonne fait aussi bien qu'une stratégie optimale donnée. Si les listes g et S sont égales, il n'y a rien à prouver. Supposons donc que $g \neq S$.

Soit k le plus petit entier tel que $g_k \neq S_k$. La solution S peut donc s'écrire $S = [g_0, g_1, \ldots g_{k-1}, S_k, S_{k+1}, \ldots, S_p]$. On montre alors que $S' := [g_0, g_1, \ldots g_{k-1}, g_k, S_{k+1}, \ldots, S_p]$ est une solution réalisable car on peut aller de la station g_k à S_{k+1} au même titre que de S_k à S_{k+1} au vu de la construction de l'algorithme glouton. Par conséquent, S' est aussi une solution optimale puisqu'il y a $p+1$ arrêts comme pour S.

En itérant ce processus, on finit par arriver à la conclusion que $[g_0, g_1, \ldots, g_p]$ est une solution optimale. Alors, si $p < m$, on aurait une contradiction car on pourrait retirer les arrêts $p+1, \ldots, m$ de la liste g, et avoir encore une solution, ce qui est impossible par construction de l'algorithme glouton. Donc $p = m$.

3 Approfondissement : problème du sac à dos

Nous sommes devant un ensemble de n objets. Chaque objet noté o_i a une valeur notée v_i et un poids noté p_i. Il s'agit d'emporter dans son sac à dos l'ensemble d'objets qui a la plus grande valeur sachant que le sac supporte un poids maximum P_{max}. Comment résoudre ce problème, quels objets doit-on prendre ?

Pour appliquer une stratégie gloutonne, nous devons définir ce que nous entendons par le meilleur choix à chaque étape. Il y a trois manières ici de définir un meilleur choix. Parmi les objets qui n'ont pas encore été pris :

A) on choisit un objet qui a la valeur maximale,

B) on choisit un objet qui a le poids minimal,

C) on choisit un objet qui a le ratio valeur / poids maximal.

L'algorithme glouton consiste, à chaque étape, à choisir parmi ces objets celui qui représente le meilleur choix (selon le critère A), B) ou C)). Ce meilleur choix aura un poids qu'on note $P1$. Ensuite, nous recommençons parmi les objets de poids $p_i \leq P_{max} - P1$. Et ainsi de suite. Prenons un exemple : le sac à dos peut contenir 15Kgs. Les poids des objets sont en Kg, les valeurs en euro.

Objet	Valeur	Poids	Ratio Valeur / Poids
Vélo	126	14	9
Vase	80	8	10
Livre	32	2	16
Nourriture	20	5	4
Vêtements	18	6	3
Bouteille	5	1	5

Un objet est représenté par une liste comme `['Velo',126,14]`.

Exercice 4. Avec un script Python, appliquer l'algorithme glouton pour chacun des trois critères (A), B) ou C)) pour la liste d'objets ci-dessus.

Question 3. Lequel des trois critères est le plus performant ? Est-ce une solution optimale ?