

Introduction à l'algorithme glouton

Un problème d'optimisation est un problème pour lequel on cherche la meilleure solution (selon un critère défini) dans un ensemble de solutions possibles.

Quelques exemples de problèmes d'optimisation :

- Rendre la monnaie avec un minimum de pièces.
- Ranger le maximum d'éléments : sac à dos, colis dans un camion, etc.
- Allocation de salles pour des cours.
- Sélection d'activité
- Et d'autres.

On parle de solution optimale toute solution qui fait partie des solutions possibles et qui est la meilleure des solutions selon le critère défini.

- Pour un rendu de monnaie, le critère pour la solution optimale sera le nombre de pièces et/ou billets.

Les algorithmes gloutons sont souvent utilisés pour résoudre ces problèmes d'optimisation . On cherche une solution optimale en effectuant le meilleur choix possible à chaque étape de l'algorithme. Dans ce type de résolution, il n'y a pas de retour en arrière. Lorsqu'un choix est fait, il n'est pas modifié par la suite. On se retrouve donc à chaque étape, avec un problème de plus en plus petit à résoudre.

Exemple classique : le rendu de monnaie

Nous allons étudier un problème d'optimisation classique : le problème du rendu de monnaie de manière optimale. On cherche à rendre la monnaie avec un nombre minimal de pièces et billets.

Voici notre système de monnaie (exprimé en euros) :

- Pièces : 0,01 ; 0,02 ; 0,05 ; 0,1 ; 0,2 ; 1 ; 2 .
- Billets : 5 ; 10 ; 20 ; 50 ; 100 ; 200; 500.

On cherche par exemple à rendre 53 euros. On peut dans un tableau énumérer quelques solutions possibles et choisir celle qui minimise le nombre de pièces et de billets.

Rendus de monnaie	Nombre de pièces et de billets
$5300 \times 0,01\text{€}$	5 300
$53 \times 1\text{€}$	53
$5 \times 10 + 1 \times 2 + 1 \times 1\text{€}$	7
$2 \times 20 + 2 \times 5 + 1 \times 2 + 1 \times 1\text{€}$	6
$1 \times 50 + 1 \times 1 + 1 \times 2\text{€}$	3

L'utilisation de ce type de méthode est très coûteux en temps de calcul car il faut explorer toutes les possibilités avant de trouver la solution optimale.

Solution et solution optimale

Un algorithme glouton consiste à effectuer des choix "locaux" (dans le cas du rendu de monnaie : rendre la plus grande valeur possible), choix qui ne seront plus jamais remis en cause mais qui permettent de réduire le problème à un problème plus "simple" (dans le cas du rendu de monnaie : rendre la somme diminuée de la valeur maximale)

Les algorithmes gloutons constituent une méthode algorithmique, parmi d'autres, pour résoudre des problèmes, en particulier d'optimisation.

Lorsqu'un algorithme glouton permet d'obtenir une solution, celle-ci n'est pas forcément la solution optimale au problème.

Un algorithme glouton peut ne trouver aucune solution bien que des solutions existent.

Implémentation de l'algorithme glouton

1. Programmer l'algorithme glouton de rendu de monnaie en langage python.

Pour cela, programmer une fonction `rendu_monnaie_glouton` qui possède comme paramètres

- un système de pièces et de billet : « système » sous forme de liste de nombres ;
- une somme à rendre : `somme`.

Cette fonction renvoie une liste de nombres qui caractérise la monnaie à rendre.

- penser à trier par ordre décroissant la liste des pièces et billets.
- ### 2. Améliorer cette fonction pour prendre en compte le fait que l'algorithme ne trouve pas de solution dans certains cas.

Voici l'algorithme glouton python associé :

```
def renduMonnaieGlouton(x):  
    pieceEtBillets = [500,200,100,50,20,10,5,2,1]  
    i = 0  
    while(x>0):  
        if(pieceEtBillets[i] > x):  
            i = i+1  
        else:  
            print(str(pieceEtBillets[i]))  
            x -= pieceEtBillets[i];
```

`renduMonnaieGlouton(33)` #Exemple sur 33 euros

La sortie pour 33 euro est alors :

20 10 2 1

Autre exemple avec 55 euro d'algorithme glouton python : `renduMonnaieGlouton(55)` #Exemple sur 55 euros

La sortie est alors :

50 5

Autre méthode

```
# valeurs des pièces
systeme_monnaie = [1,2,5,10,20,50,100]
```

Définissons le système de pièces à l'aide d'un tableau de valeurs des pièces classées par valeurs croissantes S.

[numbers=None] Pour stocker les pièces à rendre, une liste Python initialement vide peut être utilisée.

```
# liste des pièces à rendre
lst_pieces = []
```

La première pièce à rendre est potentiellement la dernière pièce du tableau `systeme_monnaie`. Une variable

```
# indice de la première pièce comparer à la somme à rendre
i = len(systeme_monnaie) - 1
```

de type entier est initialisée avec l'indice du dernier élément de ce tableau.

Chaque fois qu'une pièce de S n'est plus utilisable, la valeur de i sera diminuée de 1. Le programme s'arrête quand i atteint la valeur 0. Ce qui mène à l'écriture d'une boucle conditionnelle pour remplir la liste des pièces choisies. La somme à rendre est initialement stockée dans la variable `somme_a_rendre`.

```
# somme à rendre
somme_a_rendre = 87

# boucle de construction de la liste des pièces
while somme_a_rendre > 0:
    valeur = systeme_monnaie[i]
    if somme_a_rendre < valeur:
        i -= 1
    else:
        lst_pieces.append(valeur)
        somme_a_rendre -= valeur
```

Pour finir, le code précédent peut être encapsulé dans une fonction qui reçoit deux arguments - la somme à rendre et le système de monnaie - et qui renvoie la liste des pièces choisies par l'algorithme glouton.

```

def pieces_a_rendre(somme_a_rendre,systeme_monnaie):# liste des
pieces à rendre

lst_pieces = []

# indice de la première pièce à comparer à la somme à rendre =
len(systeme_monnaie) - 1

while somme_a_rendre > 0:

    valeur = systeme_monnaie[i]

    if somme_a_rendre < valeur:

        i -= 1

    else:

        lst_pieces.append(valeur) somme_a_rendre -=
        valeur

return lst_pieces

```

Problème du sac à dos

Pour partir en randonnée, vous disposez d'un sac à dos. Afin de pouvoir marcher longtemps chaque jour, vous décidez de limiter la masse totale transportée à 17kg. Cette limitation ne vous permet pas de pouvoir transporter tous les éléments que vous aimeriez emporter.

Vous décidez de donner une note "utilité" à chacun des objets.

Objet	eau	nourriture	nécessaire de cuisine	couverture	matelas	tente	gaz	lampe	console de jeu	change	jumelle	cartes	pull	machette	livre de chevet
Masse	3	4.3	2.2	1	1.5	3.4	0.8	0.5	2.1	2.8	1.3	0.5	0.8	2.4	0.3
Utilité	20	15	12	6	4	18	13	10	7	11	9	8	8	5	2

Vous cherchez à maximiser l'utilité totale des objets emportés dans votre sac à dos.

1. Le fait de prendre ou non chaque objet est un choix binaire.
Si vous voulez tester toutes les possibilités pour remplir votre sac à dos afin de trouver la ou les solutions optimales, combien de cas devez-vous étudier ?
2. Si vous utilisez un algorithme glouton avec comme critère l'optimisation de l'utilité, quelle sera la composition de votre sac à dos ?

N'étant pas certain de la pertinence du résultat obtenu, vous décidez de choisir les objets non plus en fonction de leur seule utilité mais en fonction de leur utilité massique c'est-à-dire en fonction du rapport utilité/masse.

De plus, plutôt que de tout recalculer à la main, vous préférez utiliser un programme informatique en langage Python.

Proposer une implémentation en langue Python de l'algorithme glouton permettant de remplir le sac à dos suivant le critère choisi.