problème du sac à dos (structures, tableaux, fonctions, niveau 2)

Cadre général

Le problème dit « du sac à dos » consiste à trouver, parmi un ensemble initial d'objets ayant chacun un poids et une valeur, quels objets mettre dans le sac de sorte que celui-ci ai une valeur maximale, mais ne pèse pas plus qu'un poids fixé au départ.

Par exemple, on chercher à remplir au mieux (valeur maximale) le sac à dos à partir des objets suivants, mais sans dépasser un poids total de 9 kg :

- boîte de 4 kg de valeur 8 ;
- bouteille de 2 kg de valeur 3 ;
- paquet de 5 kg de valeur 7 ;
- tente de 6 kg de valeur 10.

A noter que c'est bien la contrainte du poids maximal à ne pas dépasser qui rend ce problème « intéressant ». Sans une telle contrainte, il suffirait de tout mettre dans le sac pour maximiser sa valeur !

Ce problème « du sac à dos » est un problème NP-complet, c.-à-d. qu'à ce jour on ne connaît pas de meilleur algorithme pour le résoudre que celui qui consiste à essayer toutes les solutions.

Face à ce genre de problèmes, on peut alors chercher à ne pas le résoudre de façon optimale, mais utiliser un algorithme plus efficace que la recherche exhaustive, mais qui ne donnera qu'une solution approximative (une bonne solution mais qui n'est pas garantie pour être la meilleure).

Par exemple, un algorithme simple non optimal consiste à simplement remplir le sac avec les objets dans l'ordre dans lequel ils sont donnés/posés sur la table, si c'est possible (c.-à-d. ajouter l'objet ne fait pas dépasser le poids maximal autorisé).

Un autre algorithme simple non optimal consiste remplir le sac en mettant l'objet le plus lourd en premier si c'est possible, puis ensuite le 2^e objet le plus lourd si c'est possible, etc. C'est le même algorithme que le précédent mais où l'on a au préalable trié les objets par poids décroissant. On appelle cet algorithme, un algorithme « glouton » car il « veut » consommer le plus en premier.

Si l'on applique le premier algorithme à l'exemple ci-dessus :

- on met la boîte de 4 kg de valeur 8 : poids total = 4 kg, valeur totale = 8 ;
- on met la bouteille de 2 kg de valeur 3 : poids total = 6 kg, valeur totale = 11 ;
- on ne peut pas mettre le paquet de 5 kg car le poids total serait trop grand (11 kg) ;
- on ne peut pas non plus mettre la tente de 6 kg.

Au final, on arrive avec un poids total = 6 kg et une valeur totale de 11.

Si l'on applique l'algorithme glouton :

- on met la tente de 6 kg de valeur 10 (plus grande valeur) : poids total = 6 kg, valeur totale = 10 ;
- on ne peut pas mettre la boîte de 4 kg de valeur 8 (2^e plus grande valeur) car le poids total serait trop grand (18 kg) ;
- on ne peut pas mettre le paquet de 5 kg non plus ;
- on met la bouteille de 2 kg de valeur 3 : poids total = 8 kg, valeur totale = 13.

Au final, on arrive avec un poids total = 8 kg et une valeur totale de 13, ce qui est déjà mieux.

Mais la solution optimale consiste ici à prendre la boîte et le paquet. Le poids total est de 9 kg et la valeur totale de 15.

Mise en pratique

Le but est de programmer ces trois algorithmes. Commencez pour cela par représenter la liste des objets possibles, par exemple comme un ensemble de paires de valeurs.

Implémentez ensuite la recherche naïve (premier algorithme) dans une fonction qui prend en paramètres une liste d'objets et un poids maximal. Cette fonction affichera la valeur maximale trouvée et le poids restant libre dans le sac (avec notre exemple, elle afficherait : « valeur : 11, poids restant : 3 kg »).

Implémenez ensuite l'algorithme glouton en :

1. triant l'ensemble d'objets par valeurs décroissantes ;
2. appliquant la fonction précédente.

La partie la plus difficile est la recherche exacte. Celle-ci se fera de façon récursive en considérant le meilleur résultat des deux solutions consistant à prendre ou non le premier objet de la liste puis rechercher sur la suite de la liste des objets.
Plus précisément : écrire une fonction (récursive) qui prend une liste d'objets, un index de début et un poids maximum et qui

1. si la liste est vide retourne 0 (valeur d'une liste d'objets vide) ;
2. sinon, lance la recherche sur la liste privée du premier élément (c.-à-d. en passant simplement l'index de départ à un de plus que celui reçu) et mémorise la valeur obtenue (appelons-la « *valeur 1* ») ;
3. puis, si le poids du 1er objet est inférieur au poids maximum possible :
    - relance la recherche sur la liste privée du premier élément (en passant l'index de départ plus 1) avec un poids maximal réduit du poids de ce 1er objet ;
    - si la somme de la valeur obtenue par cette nouvelle recherche et de la valeur du 1er objet est plus grande que celle obtenue précédemment (« *valeur 1* »), alors retourner cette somme, sinon retourner la « *valeur 1* ».