Optimisation. Le problème du sac à dos

Le problème type

On dispose d'un sac pouvant supporter un poids maximal donné et de divers objets ayant chacun une valeur et un poids. Il s'agit de choisir les objets à emporter dans le sac afin d'obtenir la valeur totale la plus grande tout en respectant la contrainte du poids maximal. C'est un problème d'optimisation avec contrainte.
Ce problème peut se résoudre par force brute, c'est-à-dire en testant tous les cas possibles. Mais ce type de résolution présente un problème d'efficacité. Son coût en fonction du nombre d'objets disponibles croît de manière exponentielle.
Nous pouvons aussi envisager une stratégie gloutonne. Le principe d'un algorithme glouton est de faire le meilleur choix pour prendre le premier objet, puis le meilleur choix pour prendre le deuxième, et ainsi de suite.

Que faut-il entendre par meilleur choix ? Est-ce prendre l'objet qui a la plus grande valeur, l'objet qui a le plus petit poids, l'objet qui a le rapport valeur/poids le plus grand ?

Le problème à résoudre

Nous disposons d'une clé USB qui est déjà bien remplie et sur laquelle il ne reste que 5 Go de libre. Nous souhaitons copier sur cette clé des fichiers vidéos pour l'emporter en voyage. Chaque fichier a une taille et chaque vidéo a une durée. La durée n'est pas proportionnelle à la taille car les fichiers sont de format différents, certaines vidéos sont de grande qualité, d'autres sont très compressées.
Le tableau qui suit présente les fichiers disponibles avec les durées données en minutes.

Fichier	Durée	Taille
Video 1	114	4,57 Go
Video 2	32	630 Mo
Video 3	20	1,65 Go
Video 4	4	85 Mo
Video 5	18	2,15 Go
Video 6	80	2,71 Go
Video 7	5	320 Mo

Représentation des données

Un fichier est représenté par un triplet contenant son nom de type `str`, sa durée de type `int` et sa taille de type `float`. Les triplets obtenus sont les éléments d'une liste.

```
videos = [('Video 1', 114, 4.57), ('Video 2', 32, 0.630),
('Video 3', 20, 1.65), ('Video 4', 4, 0.085),
('Video 5', 18, 2.15), ('Video 6', 80, 2.71),
('Video 7', 5, 0.320)]
```

Force brute

Le principe est simple, il faut tester tous les cas possibles sans les répéter et sans en oublier un.
Une méthode consiste à associer le chiffre 1 à un fichier s'il est choisi et le chiffre 0 sinon. Nous obtenons ainsi un nombre entier écrit en binaire avec 7 bits. Le nombre 1001100 signifie que nous avons choisi les fichiers 1, 4 et 5. Le nombre 1111111 signifie que nous avons choisi tous les fichiers. A chaque nombre correspond exactement une possibilité pour construire une partie de l'ensemble des 7 fichiers.

1. **Calculer** le nombre de combinaison possible dans le cas d'un ensemble de 7 films.

Ensemble des parties

Il faut donc écrire une fonction qui prend en paramètres un ensemble, (dans notre problème de 7 fichiers) et renvoie l'ensemble des parties qui peuvent être constituées.

2. **Ecrire** une fonction `ens_des_parties` qui prend en paramètre un ensemble d'objets et renvoie une liste dont chaque élément est une partie de l'ensemble.

Exemple

```
>>> ens_des_parties([('Video 1', 114, 4.57), ('Video 2', 32, 0.630)])
[[('Video 2', 32, 0.63)], [('Video 1', 114, 4.57)], [('Video 1', 114, 4.57), ('Video 2', 32, 0.63)]]
```

3. **Ecrire** une fonction `recherche` qui prend en paramètres un ensemble de parties et la contrainte de taille max de type `float` et renvoie la partie correspondant au meilleur choix satisfaisant la contrainte après avoir parcouru toutes les parties.

Algorithme glouton

Objectif : Nous chercherons à définir une fonction glouton qui prend en paramètres une liste de fichiers, une taille maximale (celle que peut stocker la clé USB) et le type de choix utilisé (par durée, par taille, ou par durée/taille).

Méthode retenue

Une méthode consiste à construire une nouvelle liste en triant la liste passée en paramètre suivant le type de choix utilisé. La liste triée est parcourue et les noms des fichiers sont ajoutés un par un dans une variable `reponse` , tant que la taille totale ne dépasse pas la taille maximale. Cette variable est ensuite renvoyée en fin de fonction.

4. **Ecrire** une fonction glouton qui prend en paramètres une liste de fichiers, une taille maximale (celle que peut stocker la clé USB) et le type de choix utilisé (par durée, par taille, ou par durée/taille) et retourne la liste des noms de vidéo optimale selon le choix utilisé.
5. **Relever** le résultat obtenu pour chacun des trois critères définis (tailles, durée, durée/taille) dans le cas ou l'espace libre est de 5Go.
6. **Comparer** les stratégies durée et durée/taille et trouver des situations dans lesquelles la valeur trouvée est le plus éloignée de la valeur optimale.