TP n° 1

MP Lycée Camille Jullian

17/24 septembre 2021

Ce TP est consacré à l'étude d'un problème d'algorithmique classique, le problème du sac à dos, décrit dans la première partie du sujet. Les questions sont organisées pour être de difficulté relativement progressive, et les différentes parties du sujet sont largement indépendantes. Par ailleurs, l'énoncé indique clairement quels programmes écrits pour des questions antérieures peuvent être utiles à la résolution d'une question donnée, on pourra généralement tenter de répondre à chaque question même si on n'a pas réussi à faire tout ce qui précède.

Sauf mention du contraire, aucune méthode ou fonction permettant de manipuler les listes n'est autorisée, à l'exception de la fonction **len** et de la méthode **append**. En particulier, il est hors de question d'utiliser des commandes comme **sum** ou **sort** qui rendraient totalement immédiate la résolution de certaines questions.

1 Le problème du sac à dos : présentation et programmation de quelques fonctions simples.

Le problème du sac à dos est un problème d'algorithmique consistant à essayer d'optimiser le remplissage d'un sac à dos de contenance limitée. On dispose donc d'un sac à dos pouvant contenir un poids maximal P_{max}, ainsi que d'une liste d'objets (qu'on numérotera par exemple de 0 à $n-1$, pour être cohérents avec les habitudes de Python) ayant chacun un poids p_i et une valeur v_i. Le but est de remplir le sac à dos avec certains des objets de la liste, de façon à ce que le poids total de ces objets ne dépasse pas le poids maximal P_{max} (mais on a bien sûr le droit de prendre un poids moindre que P_{max}), et à ce que la valeur totale des objets soit la plus grande possible. En Python, nous représenterons un objet par une liste à deux éléments $[p_i, v_i]$ contenant son poids suivi de sa valeur, et la liste de tous les objets disponibles sera donc une liste de n élements qui seront eux-mêmes des listes à deux éléments.

Ainsi, si on dispose de la liste d'objets suivante : $[[5,12],[8,15],[3,3],[7,13]]$, nous avons à notre disposition quatre objets. Le premier a un poids de 5 kg et une valeur de 12 euros, le deuxième a un poids de 8 kilos et une valeur de 15 euros etc. Si on dispose d'un sac de contenance maximale $P_{max} = 15$ kg, on peut par exemple décider de prendre les objets numéros 1 et 3, pour un poids total de 15 kilos et une valeur totale de 28 euros ; ou de prendre les objets 0, 2 et 3 pour un poids total et une valeur totale identiques. Si on augmente la capacité du sac à dos à $P_{max} = 16$ kilos, on peut cette fois prendre les trois premiers objets, pour une valeur totale de 30 euros.

L'énoncé de chaque question du problème sera suivi d'un exemple qui sera traité sur la liste Python nommée **objets** et définie par **objets=[[5,12],[8,15],[3,3],[7,13]]**. Naturellement, les programmes écrits devront fonctionner sur n'importe quelle liste d'objets et pas uniquement sur cet exemple précis.

Question 1 : Écrire une fonction Python **valmax** qui prend comme argument une liste d'objets et qui retourne la valeur maximale parmi les objets de la liste.

Exemple : valmax(objets) doit renvoyer la valeur 15.

Question 2 : Écrire une fonction Python **poidsmin** qui prend comme argument une liste d'objets et qui retourne le numéro de l'objet le plus léger de la liste.

Exemple : poidsmin(objets) doit renvoyer la valeur 2.

Question 3 : Écrire une fonction Python **poidsminbis** qui prend comme argument une liste d'objets et qui retourne les numéros des deux objets les plus légers de la liste. On pourra réutiliser le programme écrit pour la question précédente, ou en écrire un complètement nouveau.

Exemple : poidsminbis(objets) doit renvoyer les valeurs 0, 2.

Question 4 : Écrire une fonction Python **poidstotal** qui prend comme argument une liste d'objets et qui retourne la somme des poids de tous les objets de la liste.

Exemple : poidstotal(objets) doit renvoyer la valeur 23.

Question 5 : Écrire une fonction Python **troplourds** qui prend comme arguments une liste d'objets et un poids maximal, et qui supprime de la liste les objets dont le poids est strictement supérieur au poids total souhaité.

Exemple : troplourds(objets,6) doit renvoyer $[[5, 12], [3, 3]]$.

On dit qu'une sous-liste de la liste d'objets est **compatible** avec le poids P_{max} du sac à dos si la somme des poids des objets la constituant est inférieure ou égale à P_{max}. Une sous-liste sera représentée en Python par une liste de même longueur que la liste d'objets, remplie de 0 et de 1, les 1 étant situés aux emplacements correspondant aux objets appartenant à la sous-liste. Ainsi, pour notre liste objets de longueur 4, la liste $[0, 1, 0, 1]$ représentera la sous-liste constituée des objets numéro 1 et 3.

Question 6 : Écrire une fonction **compatible(l,P,sl)** qui prend comme arguments une liste d'objets l, un poids maximal P et une liste sl de 0 et de 1 représentant une sous-liste de la liste d'objets, et qui renvoie True ou False selon la compatibilité de la sous-liste.

Exemple : compatibilite(objets,16,[0,1,0,1]) doit renvoyer True, mais compatibilite(objets,12,[0,1,0,1]) doit renvoyer False.

Question 7 : Écrire une fonction Python **souslistescompatibles(l,P)** prenant comme arguments une liste d'objets l et un poids maximal P et qui renvoie la liste de toutes les sous-listes d'objets compatibles. Pour simplifier on écrira un programme qui ne fonctionne qu'avec des listes constituées d'exactement quatre objets. On pourra bien sûr utiliser dans cette question le programme écrit à la question précédente.

Exemple : souslistescompatibles(objets,14) devrait renvoyer
$[[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 1, 1], [0, 1, 0, 0], [0, 1, 1, 0], [1, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 0], [1, 1, 0, 0]]$.

2 Programmation d'un algorithme de tri : le tri fusion.

Pour résoudre le problème du sac à dos, nous allons utiliser une méthode qui fait intervenir une procédure de tri. Dans cette partie, nous allons programmer un tel algorithme, dans une version appelée tri fusion. Le principe consiste à séparer la liste en deux sous-listes de longueur égales (ou quasiment égales si le nombre d'éléments de la liste est impair), à trier les deux sous-listes (en

effectuant à nouveau un tri fusion, de façon récursive), puis à « fusionner » les deux sous-listes triées en une grande liste triée. Pour celà, puisque ces deux sous-listes sont déjà triées, il suffit de comparer leur premier élément, placer le plus petit des deux en tête de la grande liste et le supprimer de la sous-liste auquel il appartenait, puis recommencer jusqu'à avoir tout trié.

Question 8 : Écrire une fonction **fusion** qui prend comme arguments deux listes triées et qui renvoie la fusion de ces deux listes. On pourra ici utiliser la fonction **del** ou la méthode **remove** pour effectuer les suppressions de termes dans les sous-listes.

Exemple : fusion([1,4,6],[2,5,9]) doit renvoyer la liste $[1, 2, 4, 5, 6, 9]$.

Pour écrire ensuite l'algorithme de tri proprement dit, on va définir une fonction **trifusion(l)** et utiliser un algorithme récursif. Le programme doit simplement faire les choses suivantes :

- si la liste l contient un seul élément, on retourne la liste l elle-même (qui est forcément déjà triée).
- sinon on découpe l en deux sous-listes l1 et l2 de longueur similaires.
- puis on retourne simplement la liste obtenue en fusionnant les deux listes trifusion(l1) et trifusion(l2)

Question 9 : Écrire la fonction Python correspondante.

Pas besoin d'exemple, la fonction doit simplement renvoyer la liste triée !

3 Résolution du problème du sac à dos : méthode naïve.

Pour résoudre le problème du sac à dos, on va utiliser l'algorithme suivant (qui ne donne pas toujours la solution optimale, mais qui a l'avantage d'être relativement rapide à exécuter) :

- on calcule pour chaque objet de la liste le rapport $\frac{v_i}{p_i}$ de sa valeur sur son poids, qu'on appellera **intérêt** de l'objet.
- on classe les objets par ordre d'intérêt décroissant.
- on considère les objets l'un après l'autre (dans l'ordre imposé par le point précédent), et pour chacun d'entre eux, on le met dans notre sac si et seulement s'il nous reste assez de poids disponible.

Si on reprend notre exemple habituel, on aurait des intérêts de nos objets égaux à 2.4 ; 1.875 ; 1 et 1.86. Le classement des objets par intérêt décroissant donne donc $0, 1, 3, 2$. Si on dispose par exemple d'un sac de 18 kg, on va placer l'objet 0, ajouter l'objet 1 (on a maintenant un poids de $5 + 8 = 13$ kg dans le sac), on ne met pas l'objet 3 (trop lourd), puis on met l'objet 2 (on a assez de place), et on a finalement pris les objets 0, 1 et 2.

Question 10 : Écrire une fonction **interets** qui prend comme argument une liste d'objets et qui renvoie la liste de leurs intérêts (non triée pour l'instant).

Question 11 : Écrire une fonction **triobjets** qui prend comme argument une liste d'objets et qui ressort la liste des indices des objets triée par intérêt décroissant (dans notre exemple la fonction renverrait donc la liste $[0, 1, 3, 2]$). Si on n'a pas réussi à écrire le programme **trifusion** de la partie précédente, on aura le droit d'utiliser la méthode **sort** pour trier la liste.

Question 12 : Écrire une fonction **sacados** qui prend comme arguments une liste d'objets et un poids maximal, et qui résout le problème du sac à dos par la méthode décrite ci-dessus (la fonction renverra par exemple la liste des numéros des objets qu'on embarque dans notre sac à dos). En plus de la fonction programmée à la question précédente, on pourra utiliser pour cette question la méthode **index** si on en ressent le besoin.

4 Tentative d'amélioration.

Comme on l'a signalé, la méthode utilisée à la question précédente ne donne pas toujours un résultat optimal. Pour tenter d'améliorer notre résolution, on peut procéder de la façon suivante : on commence par appliquer la méthode de résolution approchée de la troisième partie, puis on prend un par un les objets qu'on a laissés de côté, et on tente de les échanger avec chacun des objets qu'on a mis dans le sac. Si un de ces échanges améliore le résultat (le poids reste inférieur au poids maximal autorisé, et la valeur augmente) alors on effectue cet échange. Une fois tous les objets examinés, on considère la nouvelle solution comme correcte (même si en pratique elle ne sera toujours pas optimale dans certains cas).

Question 13 : Écrire une fonction **echange** qui prend comme premier argument une liste d'objets, comme deuxième argument un poids maximal, et comme troisième argument un objet n'appartenant pas à cette liste, et qui échange ce dernier avec un des objets de la liste de départ si celà améliore la valeur totale obtenue en gardant un poids convenable.

Question 14 : Écrire une fonction **sacadosbis** qui résout le problème du sac à dos en appliquant l'amélioration décrite dans cette dernière partie à l'algorithme programmé dans la partie précédente. Dans toute cette dernière partie, on pourra réutiliser certaines fonctions écrites dans les premières parties du problème.