

Devoir maison n°2 - A rendre le jeudi 8 janvier 2026

Problème : Un peu d'optimisation

Une entreprise de livraison dispose de plusieurs locaux en France et chacun possède plusieurs camions de livraisons. Celle-ci souhaite optimiser le chargement de ses camions pour diminuer ses frais de fonctionnement.

Chaque camion de l'entreprise peut charger une cargaison jusqu'à un poids maximal noté P_{\max} . L'entreprise dispose de différentes informations provenant de ses clients :

- le poids de chaque produit p_i (chaque client propose un seul produit) ;
- la valeur v_i associée au transport de chaque produit : c'est-à-dire l'argent gagné par l'entreprise si elle réalise le transport de ce produit.

En considérant que l'entreprise dispose de n clients, l'entreprise cherche donc à trouver une liste d'indices notée I contenue dans $\{1, \dots, n\}$ telle que :

$$\sum_{i \in I} p_i \leq P_{\max} \quad (\text{respect du poids maximal}) \quad \text{et} \quad \sum_{i \in I} v_i \text{ soit maximal (optimisation du profit pour l'entreprise)} .$$

Dans toute la suite, les poids seront donnés en centaines de kilogrammes et les valeurs en centaines d'euros.

1 Un exemple

Dans cette sous-partie, on suppose que $n = 4$ et que $P_{\max} = 8$ centaines de kilogrammes.

On stocke alors les différentes informations dans trois listes :

- Pr est la liste des produits proposés par les clients numérotés de 1 à 4 : $Pr = [1, 2, 3, 4]$;
- P est la liste des poids associés : $P = [3, 2, 1, 4]$;
- V est la liste des valeurs associées : $V = [4, 3, 1, 9]$.

Par exemple, le produit 2 a un poids de deux centaines de kilogrammes et une valeur de trois centaines d'euros.

Les questions **Q1**, **Q2** et **Q3** se traitent à l'aide de calculs simples.

Q1. Expliquer pourquoi une cargaison constituée d'un, de deux ou de quatre produits ne répond pas au problème posé, c'est-à-dire ne maximise pas le profit fait par l'entreprise en respectant la condition donnée sur le poids maximal.

Q2. Donner toutes les cargaisons de trois produits respectant le poids maximal. On donnera à chaque fois le profit fait par l'entreprise.

Q3. Quelle est la cargaison maximisant le profit de l'entreprise ? Que vaut le profit dans ce cas ?

2 Une méthode intuitive pour la résolution du problème

On garde les notations de la sous-partie précédente dans le cas général :

- Pr est la liste des produits (numérotés de 1 à n inclus) ;
- $P = [p_1, \dots, p_n]$ est la liste des poids associés aux produits ;
- $V = [v_1, \dots, v_n]$ est la liste des valeurs associées aux produits.

Q4. Définir une fonction `ListeProduits` ayant pour argument un entier naturel non nul n et renvoyant la liste Pr .

Une méthode intuitive pour tenter d'optimiser le profit de l'entreprise est la suivante : on calcule les ratios $\frac{v_i}{p_i}$, puis on trie les objets par ordre décroissant suivant ces valeurs. Les produits sont alors classés par rentabilité : le premier produit devient le plus rentable " au poids " et ainsi de suite. On ajoute progressivement chaque produit dans la cargaison, dans cet ordre, sans dépasser la limite du poids maximal.

Q5. Définir une fonction `Ratio` ayant pour arguments deux listes P, V où P correspond à la liste des poids et V correspond à la liste des valeurs, renvoyant la liste des ratios $\frac{v_i}{p_i}$.

La fonction suivante est associée à une méthode de tri :

```
def Tri(L):  
    ''' L est une liste de nombres réels '''  
    for i in range(1, len(L)):  
        x=L[i]  
        j=i  
        while j>0 and x<L[j-1]:  
            L[j]=L[j-1]  
            j=j-1  
        L[j]=x  
    return L
```

Q6. On exécute `Tri(L)` avec $L = [3, 5, 2, 1]$. Combien y a-t-il d'itérations de la boucle `for` ? Donner la valeur de L à la fin de chaque itération de la boucle `for`.

Q7. Quelle est la méthode de tri utilisée dans la fonction `Tri` ? Donner sa complexité (en nombre de comparaisons) dans le pire des cas et dans le meilleur des cas. On justifiera soigneusement les complexités données.

Q8. Définir une fonction `Inverse` ayant pour argument une liste de nombres réels L et renvoyant l'inverse de celle-ci. Par exemple, l'inverse de $[1, 5, 3, 4]$ est $[4, 3, 5, 1]$. *L'utilisation de $L[::-1]$ ou de la méthode `reverse` n'est pas autorisée.*

Q9. On souhaite trier une liste de poids P et une liste de valeurs V associées à une liste de produits en suivant l'ordre décroissant de la liste des ratios $\frac{v_i}{p_i}$. Justifier que les fonctions `Ratio`, `Tri` et `Inverse` ne permettent pas de répondre simplement au problème posé.

Q10. Écrire, à l'aide des fonctions `Ratio` et `Inverse`, une fonction `Tri2` ayant pour arguments une liste de poids P et une liste de valeurs V associées à une liste de produits. Cette fonction renverra les listes de poids et de valeurs triées par ordre décroissant de la liste des ratios.

Q11. Écrire fonction `Vmax` ayant pour arguments les listes de poids P et de valeurs V et le poids maximal P_{\max} du chargement, et renvoyant la valeur maximale du profit de l'entreprise en suivant la méthode proposée.

Q12. On souhaite appliquer cette méthode en utilisant les listes de poids et de valeurs de la partie 1. Donner la liste des ratios, les listes de poids et de valeurs obtenues à l'aide de la fonction `Tri2` ainsi que le profit obtenu. Commenter le résultat.

3 Une méthode récursive

Nous gardons les notations du cas général de la partie 2 : Pr, P et V. On considère pour simplifier que les poids des produits sont des entiers, ainsi que P_{\max} .

Nous introduisons une méthode récursive pour résoudre le problème d'optimisation :

- pour chacun des produits, deux choix sont possibles : il fait partie de la cargaison ou non ;
- la récursivité s'effectuera sur la liste des indices de Pr : le premier appel de la fonction se fera en utilisant l'indice n , puis l'indice $n - 1$ et ainsi de suite jusqu'à l'indice 0 (correspondant au cas où il n'y a plus de produits) ;
- pour $i \in \{0, 1, \dots, n\}$ et $\omega \in \{0, 1, \dots, P_{\max}\}$, on note $S(i, \omega)$ la valeur maximale cumulée des produits que l'on peut placer dans un camion d'une capacité maximale (en poids) de ω avec la liste constituée des i premiers produits de Pr.

On pose alors la relation de récursivité suivante :

$$S(i, \omega) = \begin{cases} 0 & \text{si } i = 0 \\ S(i - 1, \omega) & \text{si } i > 0 \text{ et } p_i > \omega \\ \max(S(i - 1, \omega), v_i + S(i - 1, \omega - p_i)) & \text{si } i > 0 \text{ et } p_i \leq \omega \end{cases} \quad (3)$$

Q13. Justifier les relations précédentes dans les trois cas.

Q14. Justifier la terminaison de l'algorithme associé à la relation de récursivité précédente, sachant que la première valeur donnée pour i sera n et la première valeur pour ω sera P_{\max} .

Q15. Définir une fonction **Max** ayant pour arguments deux réels et renvoyant le maximum parmi ces deux valeurs. Il est interdit d'utiliser la fonction **max** prédéfinie dans Python.

Q16. En vous basant sur la relation (3), recopier et compléter la définition de la fonction récursive **recur** ayant pour arguments les listes de poids et de valeurs P et V, un indice i , un poids ω , et renvoyant $S(i, \omega)$.

```
def recur(P,V,i,w) :  
    if i==0:  
        return .....  
    if P[i-1]>w:  
        return recur(P,V,i-1,w)  
    else :  
        .....  
        .....
```

Q17. Donner une série d'instructions utilisant la fonction **recur** et permettant de déterminer le profit de la partie 1.

4 Amélioration de la méthode récursive

On souhaite améliorer la méthode récursive de la partie 3. Nous allons procéder en mémorisant des calculs déjà effectués. Voici le principe : nous allons stocker les valeurs $S(i, \omega)$ dans un tableau **Memoire**, de taille $(n + 1) \times (P_{\max} + 1)$, initialisé au départ avec des éléments tous égaux à -1 .

Si la valeur de $S(i, \omega)$ a déjà été calculée, l'élément d'indice (i, ω) de ce tableau **Memoire** ne sera plus égal à -1 : on renverra donc directement la valeur. Sinon, on la calculera en suivant le principe de la partie 3 et on la stockera dans le tableau avant de la renvoyer.

Nous faisons le choix de représenter les tableaux comme des listes de listes.

Q18. Donner l'instruction permettant de créer le tableau **Memoire** initialisé avec des coefficients égaux à -1 , en supposant P_{\max} et n connus.

Q19. Recopier et compléter la fonction **recur2** en suivant le principe expliqué et permettant d'améliorer la fonction **recur**. La variable **Memoire** sera utilisée comme une variable globale.

Avec les données suivantes :

- $P = [5, 3, 3, 3]$
- $V = [4, 3, 1, 1]$
- $P_{\max} = 8;$
- **Memoire** un tableau de taille $5 \times 9;$

et en exécutant **recur2 (P, V, len(P), Pmax, Memoire)**, on obtient alors la valeur 7.

```
def recur2(P,V,i,w,Memoire) :
    if i==0:
        return 0
    if Memoire[i][w]>-1:
        return .....
    if P[i-1]>w:
        Memoire[i][w]=recur2(P,V,i-1,w,Memoire)
        return Memoire[i][w]
    else:
        if Memoire[i-1][w]==-1:
            Memoire[i-1][w]= .....
            if .....
                Memoire[i-1][w-P[i-1]]=recur2(P,V,i-1,w-P[i-1],Memoire)
        a=max(Memoire[i-1][w],V[i-1]+Memoire[i-1][w-P[i-1]])
        Memoire[i][w]=a
        return .....
```