Devoir surveillé n°1

Durée : 2 heures, calculatrices et documents interdits

Optimisation de rendement d'une entreprise de livraison

Partie I - Optimisation du chargement

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 :

$$\sum_{i \in I} p_i \leqslant P_{\max} \quad \text{(respect du poids maximal)}$$

et

$$\sum_{i \in I} v_i \quad \text{soit maximal (optimisation du profit pour l'entreprise).}$$

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

I.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 $= [4, 3, 1, 9]$.

1) 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.

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

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

I.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.

4) 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.

5) 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
```

6) 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`.

7) Ce tri fonctionnerait-il pour une chaîne de caractères dont chaque caractère est un entier ? Justifier.

8) 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.

9) 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]` n'est pas autorisée.

10) 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.

11) 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.

12) Compléter la définition de la 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.

13) On souhaite appliquer cette méthode en utilisant les listes de poids et de valeurs de la sous-partie I.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.

I.3 - Une méthode récursive

Nous gardons les notations du cas général de la sous-partie I.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 $i \in \{0, 1, \ldots, n\}$ et $\omega \in \{0, 1, \ldots, P_{\max}\}$, 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 :

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

14) Justifier les relations précédentes dans les trois cas.

15) 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 $P_{\max}$.

16) 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.

17) En vous basant sur la relation (3), 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 $S(i, \omega)$.

18) Donner une série d'instructions utilisant la fonction `recur` et permettant de déterminer le profit de la sous-partie I.1.

I.4 - Amélioration de la méthode récursive

On souhaite améliorer la méthode récursive de la sous-partie I.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 sous-partie I.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.

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

20) 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 :

— $\mathrm{P} = [5, 3, 3, 3]$;

— $\mathrm{V} = [4, 3, 1, 1]$;

— $P_{\max} = 8$;

— `Memoire` un tableau de taille 5×9 ;

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

Partie II - Données liées aux livraisons conservées par l'entreprise

À chaque livraison, l'entreprise stocke des données relatives à celle-ci. L'entreprise dispose de 20 locaux, numérotés de 1 à 20, disposant chacun d'un certain nombre de camions. Pour faciliter ses livraisons, l'entreprise découpe la France en 30 zones et associe à chaque local, trois zones possibles de livraisons.
Ces données sont enregistrées dans une base de données contenant trois tables :
La table `livraison` constituée des champs suivants :

— date : date de la livraison au format "jj-mm-aaaa" (chaîne de caractères) ;

— heure : heure de la livraison au format : "hh-mm-ss" (chaîne de caractères) ;

— id_client : identifiant du client recevant la livraison (entier) ;

— id_local : identifiant du local de l'entreprise (entier compris entre 1 et 20).

La table `client` constituée des champs suivants :

— id : identifiant du client (entier) ;

— zone : entier compris entre 1 et 30.

La table `local` constituée des champs suivants :

— id : identifiant du local (entier compris entre 1 et 20) ;

— zone1 : entier ;

— zone2 : entier ;

— zone3 : entier.

21) Donner une clé primaire pour la table `livraison`.

22) Écrire une requête SQL permettant d'obtenir les identifiants des clients livrés le 10 janvier 2021.

23) Écrire une requête SQL permettant de récupérer les dates et les heures de toutes les livraisons ayant eu lieu dans la zone 5 le 2 mars 2021.

24) Écrire une requête SQL permettant de compter le nombre de livraisons effectuées le 3 février 2021 par des camions dont les locaux ne livrent que dans des zones possibles inférieures ou égales à dix.

Chaque identifiant de client est stocké par l'entreprise en codage binaire (avec 8 bits). Par exemple, 00010111 est associé au client dont l'identifiant est le numéro 23.

25) Donner le codage associé au client 39.

Afin de retrouver l'identifiant de chaque client à l'aide de son code binaire, la fonction suivante est proposée :

```
def Identifiant(Bin):
    '''Bin est une chaîne de caractères
    constituée de 0 et 1'''
    S = 0
    for i in range(len(Bin)):
        S = S + Bin[i] * 2 ** (len(Bin) - i)
    return S
```

26) Trouver les deux erreurs dans le code de la fonction précédente.

On suppose maintenant la fonction précédente corrigée. La ligne 6 pose un problème de complexité : à chaque itération de boucle, la puissance de 2 est recalculée entièrement.

27) Écrire une fonction `Identifiant2`, qui donne le même résultat que la fonction `Identifiant` avec une meilleure complexité. Le nombre de multiplications devra être linéaire suivant la longueur de `Bin`.

28) Si l'entreprise souhaite aussi stocker la zone de chaque client en codage binaire, donner le nombre de bits minimal nécessaire.