Informatique 2ème année

Vous partez en vacances en avion et vous avez une limite : votre valise ne doit pas peser plus de 24 kg.

Chacun des n objets de votre chambre a une utilité u_i, et un poids p_i, indiqués ci-dessous.

objet	poids	utilité
Ordinateur	3	7
Matelas gonflable	4	8
Tente	5	12
Vêtements	5	10
Cadeaux	7	13
Cours de maths	8	15
Vélo pliant	8	17

Question 1 : Quels articles faut-il mettre dans la valise pour maximiser l'utilité totale sans dépasser le poids limite de $L = 24$ kg ?

Réponse : Le mieux est de prendre l'ordinateur, la tente, le cours de maths et le vélo pliant, pour une utilité totale de 51.

On cherche un algorithme qui prend en entrée les poids des objets, leurs utilités, et la limite de poids, et détermine comment maximiser l'utilité totale.

Une stratégie naturelle consisterait à examiner les objets par ordre d'utilité décroissante, et à prendre un objet dans la valise à chaque fois qu'il ne dépasse pas le poids restant disponible.

Question 2 : Comment appelle-t-on une telle stratégie ?

Réponse : C'est un algorithme glouton : il fait à chaque étape le choix qui apporte le plus grand gain immédiat, sans prendre en compte les conséquences pour la suite.

Question 3 : Dans la situation ci-dessus, quelle combinaison d'objets résulte de cette stratégie ? Est-ce une stratégie optimale ?

Réponse : En appliquant cette stratégie, on prendrait Vélo pliant, Cours de Maths, Cadeaux, et on s'arrêtrait là, avec une utilité totale de 45 alors que l'optimum est 51. Cette stratégie n'est donc pas optimale.

Question 4 : Une variation de cette stratégie serait d'examiner les objets non pas par ordre d'utilité décroissante, mais selon leur rapport utilité/poids, leur «utilité par kg». Cette stratégie donne-t-elle la réponse optimale ? Justifier.

Réponse : On peut par curiosité appliquer cette stratégie à notre exemple, et elle choisit la tente (utilité de 2,4 par kg), l'ordinateur (2,14 par kg), le vélo pliant (2,125), le matelas gonflable (2) ou les vêtements (2), puis aucun autre objet ne peut être ajouté. L'utilité totale atteint alors 44 ou 46, alors que l'optimal est 51. Cette stratégie n'est pas optimale

Question 5 : Pour trouver une solution optimale, on peut énumérer toutes les combinaisons d'objets, éliminer celles qui dépassent la limite de poids, et déterminer parmi celles qui restent, laquelle a la plus grande utilité totale. Quelle serait, en fonction du nombre d'objets n, la complexité de cette approche ?

Réponse : Il y a 2^n combinaisons possibles à énumérer. Pour chacune il faut évaluer son poids total et éventuellement son utilité totale, ce qui a une complexité en $O(n)$. La complexité totale est en $O(n2^n)$. Il faut ensuite rechercher, parmi les combinaisons qui respectent la limite de poids, celle de plus grande utilité totale : comme il y a au plus 2^n combinaisons, c'est une simple recherche de maximum en $O(2^n)$, ce qui est dominé par le coût de l'énumération.

La complexité de l'approche par recherche exhaustive est donc en $O(n2^n)$.

Pour les trois prochaines questions, on suppose fixés le nombre d'objets n, les poids $p_1, \dots p_n$, les utilités $u_1, \dots u_n$, et la limite de poids L.

Pour $t \in [\![0, L]\!]$ et $i \in [\![0, n]\!]$, on pose $M(t, i)$ l'utilité totale maximale qu'on peut obtenir en choisissant des objets parmi les i premiers objets, et sans dépasser un poids total de t.

Question 6 : Quelles valeurs de t et i choisir pour que $M(t, i)$ réponde à notre problème de valise ?

Réponse : En posant $t = L$ et $i = n$ on retrouve notre problème de départ. Il faut donc déterminer $M(L, n)$.

Question 7 : Soit $i > 0$, exprimer $M(t, i)$ en fonction des $M(t', i-1), t' \leqslant t$. Justifier (On pourra distinguer plusieurs cas).

Réponse : On peut distinguer deux manières de choisir des objets parmi les i premiers, sans dépasser le poids total t :

- En prenant l'objet i. On doit alors compléter avec une sélection d'objets parmi les $i-1$ premiers, dont le poids total ne dépasse pas $t - p_i$. En choisissant la meilleure sélection pour cela, on a une utilité totale $u_i + M(t - p_i\ ,\ i-1)$
- En ne prenant pas l'objet i. En choisissant de façon optimale parmi les objets restants, on aura par définition une utilité totale $M(t\ ,\ i-1)$

Si $p_i > t$ on n'a que la seconde possibilité : $M(t, i) = M(t, i-1)$
Sinon, on a les deux possibilités et on choisit entre les deux la meilleure utilité totale :
$M(t, i) = \max(\ M(t\ ,\ i-1)\ ,\ M(t - p_i\ ,\ i-1)\ )$.

Question 8 : Pour $i = 0$, et $t \in [\![0, L]\!]$, quelle est la valeur de $M(t, i)$?

Réponse : La relation précédente est vraie si $i > 0$. Si $i = 0$, on doit choisir parmi un sous-ensemble vide d'objets, donc on obtient une utilité totale nulle : $M(t, i) = 0$.

On ne suppose plus les données n, u_i, p_i et L fixées.

Question 9 : Écrire une fonction `tableauM(u,p,L)` qui prend en argument les données du problème (une liste d'utilités `u` , une liste de poids `p` , et un poids total limite `L`), et renvoie un tableau (une liste de listes) contenant toutes les valeurs $M(t, i)$.

(Attention : le poids et l'utilité du premier objet, p_1 et u_1, sont désignés en python par les expressions `p[0]` et `u[0]` . Il faudra tenir compte de ce décalage d'indices dans le programme, ou modifier les listes `u` et `p` pour que les données du problème soient bien aux indices 1 à n.)

Réponse :

```
1  def tableauM(u,p,L):
2      n=len(p)
3      M=[]
4      for t in range(0,L+1):
5          M.append( [0]*(n+1) )
6      for i in range(1,n+1):
7          for t in range(0,L+1):
8              if p[i-1]>t:
9                  M[t][i]=M[t][i-1]
10             else:
11                 x=M[t][i-1]
12                 y=u[i-1] + M[t-p[i-1]][i-1]
13                 M[t][i]=max(x,y)
14     return M
```

- lignes 3-5 : On crée d'abord un tableau `M` de la bonne taille, rempli de zéros, en évitant les problèmes d'alias
- lignes 6-7 : Ensuite, on doit le remplir. Comme $M(t, i)$ s'appuie sur des valeurs $M(t', i-1)$, on doit remplir la colonne $i-1$ avant de remplir la colonne i.
  La colonne 0 est déjà remplie de zéros, il faut alors remplir colonne par colonne, en commençant par la colonne 1.
- lignes 8-13 : Pour le calcul de $M(t, i)$ on distingue deux cas selon si $p_i < t$, comme expliqué à la question 7. Attention au décalage : on accède à p_i et u_i par les expressions `p[i-1]` et `u[i-1]` .

Question 10 : Cette fonction donne la plus grande valeur d'utilité totale possible, mais ne dit pas quels objets il faut mettre dans la valise pour atteindre cette utilité totale. Écrire une fonction `solution(u,p,L)` qui détermine une liste d'objets à mettre dans la valise pour maximiser l'utilité totale.

Réponse : Pour savoir si on doit choisir l'objet i pour réaliser l'optimum $M(t,i)$, le plus simple ici est de vérifier si oui ou non $M(t,i) > M(t,i-1)$
pour reconstruire une liste d'objets optimale, on prend ou pas l'objet i, et on complète selon le cas, avec une liste qui correspond à $M(t,i-1)$ ou à $M(t-p_i\ ,\ i-1)$.

Ainsi on commence avec $i=n$ pour décider si on prend l'objet n, puis on décrémente i pour statuer sur chaque objet, en mettant à jour le poids t restant disponible lorsqu'on ajoute un objet à la valise.

```
def solution(u,p,L):
    M=tableauM(u,p,L)
    S=[]
    i=len(u)
    t=L
    while i>0:
        if M[t][i] > M[t][i-1]:
            S.append(i)          # on ajoute un objet dans la valise
            t=t-p[i-1]           # on met a jour le poids disponible
        i=i-1
    return S
```

Question 11 : Dans la situation du début de l'énoncé, on imagine que la limite passe à 26 kg. Il y a alors plusieurs façons distinctes d'obtenir d'obtenir l'utilité totale maximale. Combien, et lesquelles ?

Réponse : Il y a deux façons d'obtenir l'optimum, qui vaut 54 : (Tente, Vêtements, Cours de Maths, Vélo Pliant) ou (Ordinateur, Matelas gonflable, Tente, Vêtements, Vélo Pliant).

Question 12 : On pose $V(t,i)$ le nombre de combinaisons distinctes qui permettent de maximier l'utilité totale, sous les mêmes conditions que pour $M(t,i)$.
Indiquer comment calculer toutes les valeurs de $V(t,i)$.

Réponse :

- si $i=0$, il n'y a qu'une solution possible donc $V(t,i)=1$
- si $i>0$, il y a trois cas :
  - soit on est obligé de prendre l'objet i pour obtenir l'optimal $M(t,i)$. Alors $V(t,i) = V(t-p_i, i-1)$
  - soit on est obligé de NE PAS prendre l'objet i pour obtenir l'optimal. Alors $V(t,i) = V(t,i-1)$
  - soit on peut obtenir cet optimum de ces deux façons. Alors $V(t,i) = V(t,i-1) + V(t-p_i, i-1)$

Question 13 : Écrire une fonction python `nbsolutions(u,p,L)` qui renvoie le nombre de combinaisons optimales distinctes.

Réponse : On doit calculer un tableau des valeurs de $V(t,i)$ pour obtenir finalement la réponse qui nous intéresse : $V(L,n)$. La construction de ce tableau des $V(t,i)$ se déroule à peu près comme pour le tableau des $M(t,i)$.

```
def nbsolutions(u,p,L):
    M=tableauM(u,p,L)
    n=len(p)
    V=[]
```

```
for t in range(0,L+1):
   V.append( [1]*(n+1) )
for i in range(1,n+1):
    for t in range(0,L+1):
        if p[i-1]>t:
           V[t][i]=V[t][i-1]
        else:
           x=M[t][i-1]
           y=u[i-1] + M[t-p[i-1]][i-1]
           if x>y:
               V[t][i] = V[t][i-1]
           elif y>x:
               V[t][i] = V[t-p[i-1]][i-1]
           else:
               V[t][i] = V[t][i-1] + V[t-p[i-1]][i-1]
return V[L][n]
```