TP noté : corrigé

PTSI Lycée Eiffel

21 décembre 2018

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

Question 1

Il s'agit d'un simple algorithme de recherche d'un maximum dans une liste, il faut donc recopier ce qu'on a vu en cours, en faisant attention au fait qu'on doit effectuer les comparaisons sur les valeurs des objets, donc sur le deuxième élément de la liste à deux éléments représentant chaque objet. Exceptionnellement, j'ai initialisé la variable v à la valeur 0 dans le programme qui suit car on sait que les valeurs des objets sont toutes positives.

```
def valmax(l) :
    v=0
    for i in range(len(l)) :
        if l[i][1]>v :
            v=l[i][1]
    return v
```

Question 2

À nouveau un algorithme de recherche de maximum, mais cette fois on veut afficher l'indice correspondant, on doit donc utiliser la deuxième version vue en cours.

```
def poidsmin(l) :
    p=l[0][0]
    j=0
    for i in range(len(l)) :
        if l[i][0]<p :
            p=l[i][0]
            j=i
    return j
```

Question 3

En mode gros paresseux, on peut appliquer deux fois de suite le programme précédent, en éliminant l'élément le plus léger après la première application. Pour ne pas modifier la liste prise en argument lors de l'application du programme (même si ici il est peu probable qu'on s'en resserve ensuite tant il ne sert à rien), je la recopie d'abord dans une variable m :

```
def poidsminbis(l) :
    m=[i[ :] for i in l]
```

```
    i=poidsmin(m)
    m[i][0]=m[0][0]+1
    j=poidsmin(m)
    return i,j
```

Question 4

Il s'agit cette fois d'un calcul de somme, là encore un algorithme très classique que tout le monde devrait maîtriser :

```
def poidstotal(l) :
    s=0
    for i in l :
        s+=i[0]
    return s
```

Question 5

Il suffit ici de comparer le poids de chaque objet au poids P pour savoir lesquels garder (dans mon programme, on stocke les objets conservés dans une nouvelle liste m) :

```
def troplourds(l,P) :
    m=[]
    for i in l :
        if i[0]<=P :
            m.append(i)
    return m
```

Question 6

On additionne tous les poids des objets appartenant à la sous-liste (donc ceux correspondant aux indices pour lesquelles la liste sl prend la valeur 1), puis on compare au poids maximal autorisé pour choisir ce qu'on retourne. Je me suis permis de retourner directement le test $s \leqslant P$, ce qui est tout à fait bien compris par Python (techniquement, un test comme celui-ci est justement une variable Python dont le type est un booléen, True ou False).

```
def compatible(l,P,sl) :
    s=0
    for i in range(len(l)) :
        if sl[i]==1 :
            s+=l[i][0]
    return (s<=P)
```

Question 7

L'énoncé laisse sous-entendre qu'un programme bien bourrin suffira, puisqu'on se limite à des listes de longueur 4. On peut donc tout simplement créer toutes les sous-listes possibles, qui sont de la forme $[i, j, k, z]$, avec chacune des quatre variables prenant soit la valeur 0, soit la valeur 1, puis appliquer le programme précédent à chacune et stocker les sous-listes compatibles dans une grande liste ici appelée liste :

```
def souslistescompatibles(l,P) :
    liste=[]
    for i in range(2) :
        for j in range(2) :
            for k in range(2) :
                for z in range(2) :
                    test=[i,j,k,z]
                    if compatible(l,P,test) :
                        liste.append(test)
    return liste
```

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

Question 8

Un programme tout simple qui, tant qu'aucune des deux listes n'est vide, compare le premier élément de chaque liste, case le plus petit dans la liste fusionnée notée m, et le supprime de la liste auquel il appartenait. À la fin, je retourne la concaténation des trois listes pour ne pas oublier les éléments restant dans la liste qui n'a pas encore été vidée (l'une des deux listes $l1$ et $l2$ est donc vide à ce moment-là, et la concaténation ne risque pas de détruire l'ordre).

```
def fusion(l1,l2) :
    m=[]
    while len(l1)>0 and len(l2)>0 :
        if l1[0]>l2[0] :
            m.append(l2[0])
            del(l2[0])
        else :
            m.append(l1[0])
            del(l1[0])
    return m+l1+l2
```

Question 9

On se contente de faire ce que dit l'énoncé :

```
def trifusion(l) :
    if len(l)==1 :
        return l
    n=len(l)//2
    l1=l[ :n]
    l2=l[n :]
    return fusion(trifusion(l1),trifusion(l2))
```

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

Question 10

Cette question étant vraiment très facile, pas de commentaire. Ah si, sur de vieilles versions de Python, il faudrait mettre un float autour du quotient pour qu'il ne fasse pas une division entière.

```
def interets(l) :
    return [i[1]/i[0] for i in l]
```

Question 11

Une façon brutale de faire : on trie la liste des intérêts, on la retourne pour l'avoir dans le sens décroissant, puis on liste les indices correspondants.

```
def triobjets(l) :
    i=interets(l)
    liste=trifusion(i)
    liste.reverse()
    return [i.index(j) for j in liste]
```

Question 12

On effectue l'algorithme décrit dans l'énoncé : on applique d'abord la fonction précédente pour obtenir la liste des indices des objets par intérêt décroissant, puis on les considère dans cet ordre, en créant une variable p pour conserver sous la main le poids qu'on a déjà mis dans le sac. Chaque objet qui ne fait pas dépasser le poids total est ajouté dans le sac (représenté ici par la liste m) :

```
def sacados(l,P) :
    liste=triobjets(l)
    p=0
    m=[]
    for i in liste :
        if p+l[i][0]<=P :
            p=p+l[i][0]
            m.append(i)
    return m
```

4 Tentative d'amélioration.

Question 13

Le programme suivant commence par calculer le poids total de la liste avant les tentatives d'échange. Puis on considère chaque objet de la liste, on regarde si le nouveau poids obtenu en enlevant cet objet et en ajoutant l'objet o à la place ne devient pas trop grand, et on vérifie également si la valeur de o est supérieure à la valeur de l'objet qu'on souhaite enlever. Si les deux conditions sont vérifiées, on effectue l'échange et on ressort tout de suite la nouvelle liste (on ne veut surtout pas échanger notre objet o avec plusieurs objets de la liste !). Si aucun échange n'a été effectué, on retourne l en fin de boucle.

```
def echange(l,p,o) :
    a=poidstotal(l)
```

```
for i in range(len(l)) :
        if a-l[i][0]+o[0]<=p and o[1]>l[i][1] :
                l[i]=o
                return l
return l
```

Question 14

On commence ici par résoudre le problème à l'aide de fonction sacados programmée plus haut. La variable a représente la liste des indices des objets qui sont dans le sac à dos, la variable b représente la liste des indices des objets qui ne sont PAS dans le sac à dos. On sépare donc liste initiale en deux listes disjointes, $l1$ qui représente le contenu provisoire du sac à dos, et $l2$ qui représente les objets laissés de côté par notre remplissage. Pour chacun des ces objets de la lkiste $l2$, on tente alors des échanges avec la liste $l1$, puis on retourne cette liste éventuellement modifée par les échanges. Un test sur la liste initiale objets montre qu'on obtient un résultat différent (et meilleur) pour un poids maximal de 18. Notons que ce dernier programme est en fait très mauvais, il serait nettement meilleur de modifier directement la fonction sacados pour construire simultanément les deux listes $l1$ et $l2$.

```
def sacadosbis(l,P) :
        a=sacados(l,P)
        b=[i for i in range(len(l)) if not (i in a)]
        l1=[l[i] for i in a]
        l2=[l[i] for i in b]
        for i in l2 :
                l1=echange(l1,P,i)
        return l1
```