TD noté, mercredi 3 décembre 2008

Le programme construit au fur et à mesure des questions devra être imprimé à la fin du TD et rendu au chargé de TD. Il ne faut pas oublier de mentionner son nom en commentaires au début du programme et l'ajouter sur chaque page. Les réponses autres que des parties de programme seront insérées sous forme de commentaires.

Le président de la république souhaite créer une pièce de monnaie à son effigie à condition que celle-ci facilite la vie des citoyens. Il voudrait que celle-ci soit d'un montant compris entre 1 et 10 euros et qu'elle permette de construire la somme de 99 euros avec moins de pièces qu'actuellement. Les questions qui suivent ont pour but de décomposer en pièces n'importe quel montant avec un jeu de pièces donné.

1) On considère que `pieces` est une liste de pièces triées par ordre croissant. Compléter le programme suivant afin d'obtenir la liste triée par ordre décroissant [1]. (2 points)

```
pieces = [1,2,5,10,20,50]
...
```

2) On souhaite écrire une fonction qui prend comme argument un montant `m` et une liste `pieces` toujours triée en ordre décroissant. Cette fonction doit retourner la plus grande pièce inférieure ou égale au montant `m`. (3 points)

3) En utilisant la fonction précédente, on veut décomposer un montant `m` en une liste de pièces dont la somme est égale au montant. On construit pour cela une seconde fonction qui prend aussi comme argument un montant `m` et une liste de pièces `pieces`. Cette fonction retourne une liste de pièces dont la somme est égale au montant. Une même pièce peut apparaître plusieurs fois. L'algorithme proposé est le suivant :

1. La fonction cherche la plus grande pièce inférieure ou égale au montant.
2. Elle ajoute cette pièce au résultat puis la soustrait au montant.
3. Si la somme restante est toujours positive, on retourne à la première étape.

Cet algorithme fonctionne pour le jeu de pièces donné en exemple et décompose 49 euros en $49 = 20 + 20 + 5 + 2 + 2$. (5 points)

4) Prolongez votre programme pour déterminer avec cet algorithme le plus grand montant compris entre 1 et 99 euros inclus et qui nécessite le plus grand nombre de pièces ? (2 points)

5) On ajoute la pièce 4 à la liste des pièces :

```
pieces = [1,2,4,5,10,20,50]
```

Que retourne l'algorithme précédent comme réponse à la question 3 avec cette nouvelle liste et pour le montant 98 ? Est-ce la meilleure solution ? (2 points)

1. On peut s'aider de tout type de document. Il est possible d'utiliser les méthodes associées à la classe `list` qu'on peut aussi retrouver via un moteur de recherche internet avec la requête *python list*.

6) Comme l'algorithme précédent ne fournit pas la bonne solution pour tous les montants, il faut imaginer une solution qui puisse traiter tous les jeux de pièces. On procède par récurrence. Soit $P = (p_1, \ldots, p_n)$ l'ensemble des pièces. On définit la fonction $f(m, P)$ comme étant le nombre de pièces nécessaire pour décomposer le montant m. On vérifie tout d'abord que :

1. La fonction $f(m, P)$ doit retourner 1 si le montant m est déjà une pièce.
2. La fonction $f(m, P)$ vérifie la relation de récurrence suivante :

$$f(m, P) = \min \{f(m - p, P) + 1 \text{ pour tout } p \in P\}$$

Cet algorithme permet de construire la meilleure décomposition en pièces pour tout montant. La dernière expression signifie que si le montant m est décomposé de façon optimale avec les pièces (p_1, p_2, p_3, p_4) alors le montant $m - p_4$ est aussi décomposé de façon optimale avec les mêmes pièces (p_1, p_2, p_3).

Il ne reste plus qu'à implémenter la fonction [2] $f(m, P)$. (5 points)

7) Qu'obtient-on avec le jeu de pièces `pieces` $= [1, 2, 4, 5, 10, 20, 50]$? Prolongez le programme pour déterminer tous les choix possibles du président parmi les pièces [3,4,6,7,8,9] ? (1 point)

8) Quel est le coût de votre algorithme ? (facultatif)

Correction

```
# coding: latin-1

###################################
# question 1 : retourner un tableau
###################################

pieces = [1,2,5,10,20,50]
pieces.reverse ()
print pieces # affiche [50, 20, 10, 5, 2, 1]

# il existait d'autres solutions
pieces.sort (reverse = True)

# ou encore l'utilisation d'une fonction compare modifiée
# qui s'inspire de l'exemple
# http://www.xavierdupre.fr/enseignement/initiation/...
# ...initiation_via_python_ellipse/chap2_type_tex_-_tri-_3.html

# on encore un tri programmé
# http://www.xavierdupre.fr/enseignement/initiation/...
# ...initiation_via_python_ellipse/chap3_syntaxe_tex_-_tri-_27.html

#####################################################################
# question 2 : trouve la plus grande pièce inférieure à un montant
#####################################################################

def plus_grande_piece (montant, pieces) :
    # on suppose que les pièces sont triées par ordre décroissant

    for p in pieces :
```

2. Même si l'algorithme est présenté de façon récurrente, il peut être implémenté de façon récurrente ou non. La méthode récurrente est toutefois déconseillée car beaucoup plus lente.

```
        if p <= montant : return p

    # on peut ajouter la ligne
    return 0
    # qui correspond au fait qu'aucune pièce plus petite ou égale au montant
    # n'a été trouvé --> le montant est négatif ou nul

# on vérifie
print "plus_grande_piece (80, pieces) =", plus_grande_piece (80, pieces)
                 # affiche 50

####################################
# question 3 : décomposer un montant
####################################

def decomposer (montant, pieces) :
    # on suppose que les pièces sont triées par ordre décroissant
    res = [ ]  # contiendra la liste des pièces pour le montant
    while montant > 0 :
        p = plus_grande_piece (montant, pieces)  # on prend la plus grande pièce
                                                 # inférieure ou égale au montant
        res.append (p)                           # on l'ajoute à la solution
        montant -= p                             # on ôté p à montant :
                                       # c'est ce qu'il reste encore à décomposer
    return res

print "decomposer (98, pieces) =", decomposer (98, pieces)
               # affiche [50, 20, 20, 5, 2, 1]

print "decomposer (99, pieces) =", decomposer (99, pieces)
               # affiche [50, 20, 20, 5, 2, 2]

#####################################################
# question 4 : trouver la décomposition la plus grande
#####################################################

def maximum_piece (pieces) :
    # détermine le nombre maximum de pièces à utiliser
    maxi    = 0
    montant = 0
    for m in range (1, 100) :
        r = decomposer (m, pieces)
        if len (r) >= maxi :    # si on remplace cette ligne par if len (r) > maxi :
            maxi    = len (r)   # on trouve le plus petit montant
                                # avec la pire décomposition
            montant = m         # et non le plus grand montant
                                # avec la pire décomposation
    return maxi, montant

print "maximum_piece (pieces) =", maximum_piece (pieces) # affiche (6, 99)

#################################
# question 5 : décomposition de 98
#################################

pieces4 = [1,2,4,5,10,20,50]
pieces4.reverse ()

print "decomposer (98, pieces) = ", decomposer (98, pieces)    # [50, 20, 20, 5, 2, 1]
```

```
print "decomposer (98, pieces4) = ", decomposer (98, pieces4)  # [50, 20, 20, 5, 2, 1]

"""
Les deux décompositions sont identiques.
Or il existe une décomposition plus courte avec la pièce 4 :
98 = 50 + 20 + 20 + 4 + 4 = 5 pièces

L'algorithme fait la même erreur lorsqu'il décompose le montant 8.
Il cherche toujours la plus grande pièce inférieure au montant qui est 5.
Il lui est alors impossible d'utiliser la pièce 4  pour décomposer 8.

Cet algorithme ne fournit pas la bonne solution avec ce nouveau jeu de pièces.
"""

################################
# question 6 : algorithme optimal
################################

# version récursive : très longue
def decomposer_optimal (montant, pieces) :
    if montant in pieces :
        return [ montant ]
    else :
        r = [ 1 for m in range (0, montant) ]
        for p in pieces :
            if montant > p : # si ce test n'est pas fait, la récurrence peut être infinie
                             # car les montants négatifs ne sont pas pris en compte
                             # par le premier test
                dec = decomposer_optimal (montant - p, pieces) + [p]
                if len (dec) < len (r) :
                    r = dec

        return r

# print "decomposer_optimal (98, pieces4) =", decomposer_optimal (98, pieces4)
# trop long

# version non récursive
def decomposer_optimal (montant, pieces) :
    memo = [ [ 1 for l in range (0, m) ] for m in range (0, montant+1) ]
    # memo [i] contient la pire décomposition du montant i (que des pièces de un)

    # pour les pièces de pieces, on sait faire plus court
    for p in pieces :
        if p < len (memo) :
            memo [p] = [ p ]

    for m in range (1, montant+1) :
        for p in pieces :
            if m > p :
                # on calcule la nouvelle décomposition
                dec = [p] + memo [m-p]
                # si elle est plus courte, on la garde
                if len (dec) < len (memo [m] ) :
                    memo [m] = dec

    # on retourne la meilleur décomposition pour montant
    return memo [ montant ]
```

```
# beaucoup plus rapide
print "decomposer_optimal (98, pieces4) =",  decomposer_optimal (98, pieces4)
             # affiche [50, 20, 20, 4, 4]


######################
# question 7 : résultat
######################


# pour trouver la décomposition la plus longue avec n'importe quel jeu de pièces
# on reprend la fonction maximum_piece et on remplace decomposer par decomposer optimale

def maximum_piece (pieces) :
    # détermine le nombre maximum de pièces à utiliser
    maxi    = 0
    montant = 0
    for m in range (1, 100) :
        r = decomposer_optimal (m, pieces)
        if len (r) >= maxi :    # si on remplace cette ligne par if len (r) > maxi :
            maxi    = len (r)   # on trouve le plus petit montant
                                # avec la pire décomposation
            montant = m         # et non le plus grand montant
                                # avec la pire décomposation
    return maxi, montant

print "maximum_piece (pieces) =", maximum_piece (pieces) # affiche (6, 99)
print "maximum_piece (pieces4) =", maximum_piece (pieces4) # affiche (5, 99)


# on teste pour toutes les pièces [3,4,6,7,8,9]
# ajoutées au jeu de pièces standard [1,2,5,10,20,50]
ensemble = [3,4,6,7,8,9]

for ajout in [3,4,6,7,8,9] :
    pieces = [1,2,5,10,20,50] + [ ajout ]
    pieces.sort (reverse = True)
    print "maximum_piece (" + str (pieces) + ") = ", maximum_piece (pieces)

# résultat :
"""
maximum_piece ([50, 20, 10, 5, 3, 2, 1]) =  (6, 99)
maximum_piece ([50, 20, 10, 5, 4, 2, 1]) =  (5, 99)  # 4, ok
maximum_piece ([50, 20, 10, 6, 5, 2, 1]) =  (6, 99)
maximum_piece ([50, 20, 10, 7, 5, 2, 1]) =  (5, 99)  # 7, ok
maximum_piece ([50, 20, 10, 8, 5, 2, 1]) =  (5, 99)  # 8, ok
maximum_piece ([50, 20, 10, 9, 5, 2, 1]) =  (5, 98)  # 9, ok
"""


############################
# question 8 : décomposition
############################

"""
On cherche ici le coût de la fonction decomposer_optimal en fonction du montant.
Il n'y a qu'une seule boucle qui dépend du montant, le coût de la fonction est en O(n).

Dans la version récursive, le coût est le résultat d'une suite récurrente :
```

```
u(n) = u(n-1) + ... + u(n-d)
où d est le nombre de pièces.

Le coût est donc en O(a^n) où a est la plus grande des racines du polynôme :

P (x) = x^d - x^(d-1) - ... - 1

P(1) < 0 et lim P(x) = infini lorsque x tend vers infini,
donc la plus grande des racines est supérieure à 1.
Le coût de la fonction decomposer_optimal récursive est en O(a^n) avec 1,96 < a < 1,97.

Pour des explications plus conséquentes, voir la page
http://fr.wikipedia.org/wiki/Suite_r%C3%A9currente_lin%C3%A9aire
sur les suites récurrentes et l'exercice 12.3.3 du livre :
http://www.xavierdupre.fr/enseignement/initiation/...
 ...initiation_via_python_ellipse.pdf
(ou 13.3.3 http://www.xavierdupre.fr/enseignement/...
 ...initiation/initiation_via_python_small.pdf).
"""
```

L'algorithme qui décompose un montant de façon optimale quelque soient le montant et le jeu de pièces s'apparente à la programmation dynamique. C'est le même algorithme que celui qui permet de chercher le plus court chemin dans un graphe où les nœuds serait les montants de 0 à 99 et où il y aurait autant d'arcs que de pièces partant de chaque nœuds. Les arcs seraient tous de même poids : 1. Avec cette description, trouver la meilleure décomposition revient à trouver le chemin incluant le moins d'arcs possible.