

Corrigé de la feuille d'exercices 6

Programmation dynamique

Exercice 1

```
def matrice_max(A):
    n, m = len(A), len(A[0])
    M = [ [0] * m for k in range(n) ]
    M[0][0] = A[0][0]          # poids (0,0)
    for i in range(1,n):      # poids (i,0) bord gauche
        M[i][0] = M[i-1][0] + A[i][0]
    for j in range(1,m):      # poids (0,j) bord haut
        M[0][j] = M[0][j-1] + A[0][j]
    for i in range(1,n):
        for j in range(1,m):
            M[i][j] = A[i][j] + max(M[i-1][j], M[i][j-1])
    Chemin = [(n-1,m-1)]
    i, j = n-1, m-1
    while i>0 or j>0:
        if i>0 and M[i-1][j] > M[i][j-1]:
            Chemin.append((i-1,j))
            i -= 1
        else:
            Chemin.append((i,j-1))
            j -= 1
    return M[n-1][m-1], Chemin[::-1]
```

Exemple : (celui du cours).

```
In [1]: A = [[1,2,1,3,2,1],
             [2,1,1,1,2,2],
             [1,2,1,3,1,3],
             [1,1,2,1,3,1]]
```

```
In [2]: matrice_max(A)
```

```
Out[2]: (17, [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4),
              (1, 5), (2, 5), (3, 5)])
```

Exercice 2

1) Une application telle quelle programmation dynamique ne fonctionne pas.

D'abord il est préférable d'exprimer toutes sommes et valeurs en centimes afin d'éviter les problèmes de dépassement.

Ensuite une version telle quelle par récursivité dépasse les capacités de la pile de récursion :

```
L = [1,2,5,10,20,50,100,200,500,1000,2000,5000,100000,200000,
      500000]
D = {}
def f(s, L):## Marche pas !!!
    if s==0:
        return 0
    R = []
    for v in L:
        if s >= v:
            if s-v in D:
                R.append(D[s-v])
            else:
                x = f(s-v,L)
                D[s-v] = x
                R.append(x)
    return 1 + min(R)
```

```
In [1]: f(1500,L)
```

```
...
```

```
RuntimeError: maximum recursion depth exceeded in comparison
```

Deux modifications à apporter permettent l'obtention d'une version plus efficace :

– Changer la formule de récurrence en :

$$f(0, L) = 0 \text{ et } f(s, L) = \begin{cases} 1 & \text{si } s \in L \\ 1 + \min \{f(s - v, L) \mid s \geq v, v \in L\} & \text{sinon} \end{cases}$$

– Parcourir L de la dernière (plus grande) valeur à la première (plus petite).

```
L = [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 100000, 200000,
500000]
```

```
D = {}
```

```
def f(s, L):
    if s==0:
        return 0
    R = []
    for v in L[::-1]: # Modif !
        if s == v: # Modif !
            D[s] = 1 # Modif !
            return 1 # Modif !
        if s > v: # Modif !
            if s-v in D:
                R.append(D[s-v])
            else:
                x = f(s-v, L)
                D[s-v] = x
                R.append(x)
    return 1 + min(R)
```

```
In [1]: f(1500, L)
```

```
Out[1]: 2
```

2) Voici un algorithme glouton du plus grand d'abord :

```
def fg(s, L):
    n = 0
    while s > 0:
```

```
        for x in L[::-1]:
            if x <= s:
                s = s-x
                n += 1
            break
    return n
```

Exercice 3

1) L'algorithme glouton donne :

```
def sacADos(V, W, wMax):
    V1 = V[:]
    W1 = W[:]
    # Tri (par insertion)
    for i in range(1, len(V1)):
        k = i
        v = V1[i]
        w = W1[i]
        while k > 0 and V1[k-1]/W1[k-1] < v/w:
            V1[k], W1[k] = V1[k-1], W1[k-1]
            k -= 1
        V1[k], W1[k] = v, w
    # V1 et W1 sont tries par vi/wi décroissant
    v, w = 0, 0
    for i in range(len(V1)):
        if w + W1[i] <= wMax:
            v += V1[i]
            w += W1[i]
    return v
```

Exemple :

```
In [2]: V = [2, 1, 3, 1, 5]
```

```
In [3]: W = [10, 5, 4, 3, 5]
```

```
In [4]: sacADos(V,W,10)
Out[4]: 8
```

La solution renvoyée est ici optimale.

2) Programmation dynamique de bas en haut : on complète un tableau de taille $(n+1) \times (wMax+1)$, l'élément (i, j) contenant $f(i, j)$.

```
def sacADosDyn1(V,W,wMax):
    n = len(V)
    T = [[0] * (wMax+1) for k in range(n+1)]
    for k in range(n):
        for w in range(wMax+1):
            if W[k] <= w:
                T[k+1][w] = max(V[k] + T[k][w-W[k]], T[k][w])
            else:
                T[k+1][w] = T[k][w]
    return T[n][wMax]
```

3) Programmation dynamique de haut en bas (par récursivité) et avec memoisation.

```
def sacADosDyn2(V,W,wMax):
    dic = {}
    def f(k,w):
        if (k,w) not in dic:
            if k == 0 or w == 0:
                x = 0
            elif W[k-1] <= w:
                x = max(V[k-1]+f(k-1,w-W[k-1]), f(k-1,w))
            else:
                x = f(k-1,w)
            dic[(k,w)] = x
        return dic[(k,w)]
    return f(len(V),wMax)
```

4)

```
def choixOptimal(V,W,wMax):
    # meme code que precedemment pour construire Dic
    dic = {}
    def f(k,w):
        if (k,w) not in dic:
            if k == 0 or w == 0:
                x = 0
            elif W[k-1] <= w:
                x = max(V[k-1]+f(k-1,w-W[k-1]), f(k-1,w))
            else:
                x = f(k-1,w)
            dic[(k,w)] = x
        return dic[(k,w)]
    f(len(V),wMax)
    # Construction solution Optimale
    sac = []
    k, w = len(V), wMax
    while k > 0:
        if dic[(k,w)] > dic[(k-1,w)]:
            sac.append((V[k-1],W[k-1]))
            w -= W[k-1]
        k -= 1
    return sac
```

Exemple, toujours avec les mêmes données :

```
In [2]: choixOptimal(V,W,10)
Out[2]: [(5, 5), (3, 4)]
```