Corrigé

Programmation dynamique

Exercice 1. rendu de monnaie

```
def glouton(n, c):
    p = len(c) - 1
    s = []
    while n > 0:
        while c[p] > n:
            p -= 1
        s.append(c[p])
        n -= c[p]
    return s
```

Si $n < c_p$ une décomposition de n ne peut utiliser la valeur c_p donc $f(n,p) = f(n,p-1)$.
Si $n \geqslant c_p$ il y a deux types de décompositions possibles de n :

- celles qui n'utilisent pas de pièces c_p et dont la valeur minimale est $f(n,p-1)$;
- celles qui utilisent au moins une pièce c_p et dont la valeur minimale est $f(n-c_p,p)$.

On a donc bien $f(n,p) = \min(f(n,p-1), f(n-c_p,p))$.
La fonction qui suit calcule le nombre d'unités monétaires minimal à utiliser :

```
def dynamique1(n, c):
    p = len(c) - 1
    t = [[None for j in range(p + 1)] for i in range(n + 1)]
    for j in range(0, p + 1):
        t[0][j] = 0
    for i in range(1, n + 1):
        t[i][0] = i
    for j in range(1, p + 1):
        for i in range(1, n + 1):
            if i < c[j]:
                t[i][j] = t[i][j - 1]
            else:
                t[i][j] = min(t[i][j - 1], 1 + t[i - c[j]][j])
    return t[n][p]
```

On modifie cette fonction pour retourner la décomposition :

```
def dynamique2(n, c):
    p = len(c) - 1
    t = [[None for j in range(p + 1)] for i in range(n + 1)]
    for j in range(0, p + 1):
        t[0][j] = 0, []
    for i in range(1, n + 1):
        t[i][0] = i, [c[0]] * i
    for j in range(1, p + 1):
        for i in range(1, n + 1):
            if i >= c[j] and 1 + t[i - c[j]][j][0] < t[i][j-1][0]:
                t[i][j] = 1 + t[i - c[j]][j][0], [c[j]] + t[i-c[j]][j][1]
            else:
                t[i][j] = t[i][j - 1]
    return t[n][p][1]
```

Enfin, la version avec tableau uni-dimensionnel :

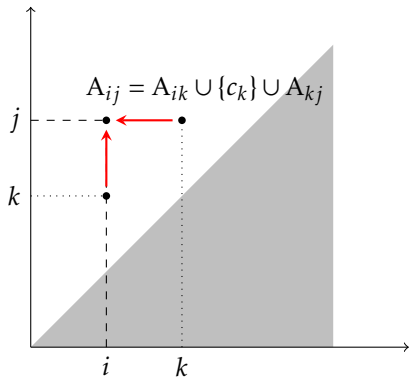
```python
def dynamique3(n, c):
    p = len(c) - 1
    t = [None  for i in range(n + 1)]
    t[0] = 0, []
    for i in range(1, n + 1):
        t[i] = i, [c[0]] * i
    for j in range(1, p + 1):
        for i in range(1, n + 1):
            if i >= c[j] and 1 + t[i - c[j]][0] < t[i][0]:
                t[i] = 1 + t[i - c[j]][0], [c[j]] + t[i-c[j]][1]
    return t[n][1]
```

Exercice 2. Programmation de festival

On peut assister au concert c_k entre les dates f_i et d_j si et seulement si $f_i \leqslant d_k < f_k \leqslant d_j$.
Mais si $i \geqslant j$ on a $f_j \leqslant f_i$ et $S_{ij} = \emptyset$.
Si $i < j$, supposons qu'il existe un concert c_k auquel on peut assister entre les dates f_i et d_j (notons au passage que ceci implique $i < k < j$). Le nombre maximal de concerts auquel il est possible d'assister entre les dates f_i et d_j *et parmi lesquels figure* c_k est alors égal à $m_{ik} + 1 + m_{kj}$. On a donc bien dans ce cas :

$$m_{ij} = \max\left\{m_{ik} + m_{kj} + 1 \;\middle|\; c_k \in S_{ij}\right\}$$

Nous allons donc créer un tableau A de taille $(n+2)\times(n+2)$ destiné à contenir des listes de concerts de S_{ij} pour lesquels $|A_{ij}| = m_{ij}$. Ce tableau sera rempli suivant le diagramme des dépendances suivant :



en le parcourant de bas en haut et de la droite vers le gauche (seul le demi-tableau non grisé est rempli) et le résultat final retourné sera $A_{0,n+1}$.

```python
def festival1(d, f):
    n = len(d)
    d = [-float('inf')] + d + [float('inf')]
    f = [-float('inf')] + f + [float('inf')]
    A = [[[] for j in range(n+2)] for i in range(n+2)]
    for j in range(1, n + 2):
        for i in range(j - 1, -1, -1):
            for k in range(i + 1, j):
                if d[k] >= f[i] and f[k] <= d[j]:
                    if len(A[i][k]) + 1 + len(A[k][j]) > len(A[i][j]):
                        A[i][j] = A[i][k] + [k] + A[k][j]
    return A[0][n+1]
```

Considérons maintenant un ensemble A de concerts compatibles entre eux et de cardinal maximal, et $c_m \in A$ un concert dont la date de fin f_m est minimale. Le concert c_m est nécessairement le premier auquel on assiste car s'il existait $c_i \in A$ tel que $d_i < d_m$ on aurait $f_i \leqslant d_m < f_m$ pour que les deux concerts ne se chevauchent pas. Ainsi tous les autres concerts c_i de A vérifient $d_i \geqslant f_m$. Or $f_1 \leqslant f_m$ donc il est possible de remplacer c_m par c_1 dans A, et ainsi on a montré qu'il existe une série de concerts de cardinal maximal qui contient c_1.

On peut donc trouver une série de concerts de cardinal maximal en choisissant c_1 puis en réitérant le procédé à partir de la date f_1.

```
def festival2(d, f):
    n = len(d)
    i, a = 1, [1]
    for k in range(2, n + 1):
        if d[k-1] >= f[i-1]:
            a.append(k)
            i = k
    return a
```

Exercice 3. Impression équilibrée

La stratégie gloutonne est la suivante :

```
def glouton(m, w):
    s = 0
    e = m - w[0]
    for j in range(1, len(w)):
        if w[j] + 1 <= e:
            e -= w[j] + 1
        else:
            s += e**3
            e = m - w[j]
    return s
```

Cependant, elle ne fournit pas nécessairement une solution optimale. Par exemple, si $m = 5$ et $w = [2, 2, 1, 4]$ l'algorithme glouton fournit le découpage indiqué à gauche ci-dessous pour un déséquilibre total de $4^3 = 64$ alors que le découpage de droite n'a qu'un déséquilibre de $3^3 + 1^3 = 28$.

```
xx xx
x␣␣␣␣
xxxx
```

```
xx␣␣␣
xx x␣
xxxx
```

Considérons le formatage du paragraphe contenant les mots $w_i, \ldots, w_n$. Si la première ligne contient les mots $w_i, \ldots, w_j$ avec $j < n$, le coût du déséquilibre associé est égal à $d(j+1) + f(i, j)$. On en déduit la formule :

$$d(i) = \begin{cases} 0 & \text{si } \alpha(i) = n \\ \min\limits_{i \leqslant j \leqslant \alpha(i)} \big(d(j+1) + f(i,j)\big) & \text{sinon} \end{cases}$$

(le premier cas correspond au cas où les mots restants peuvent être tous positionnés sur la dernière ligne).
On peut donc remplir le tableau d de la droite vers la gauche, la première case étant le résultat du déséquilibre total.

```
def dynamique(m, w):
    n = len(w)
    d = [0 for k in range(n)]
    for i in range(n-2, -1, -1):
        e = m - w[i]
        x = d[i+1] + e**3
        j = i + 1
        while j < n and w[j] + 1 <= e:
            e -= w[j] + 1
            x = min(x, d[j+1] + e**3)
        d[i] = x
    return d[0]
```

Si on prend maintenant pour mesure du déséquilibre la fonction

$$f(i, j) = m - j + i - \sum_{k=i}^{j} w_k$$

nous allins montrer que l'algorithme glouton fournit la solution optimale.
En effet, si d_g désigne le déséquilibre obtenu avec l'algorithme glouton on dispose de la relation :

$$d_g(i) = \begin{cases} 0 & \text{si } \alpha(i) = n \\ d_g\big(\alpha(i)+1\big) + f\big(i, \alpha(i)\big) & \text{sinon.} \end{cases}$$

On prouve alors que pour une telle mesure du déséquilibre, $d(i) = d_g(i)$, en démontrant que dans ce cas la valeur du minimum est toujours atteinte pour $j = \alpha(i)$.

- C'est clair lorsque $\alpha(i) = n$ car dans ce cas $d_g(i) = d(i) = 0$.
- Lorsque $\alpha(i) < n$ nous avons $d(i) = \min_{i \leqslant j \leqslant \alpha(i)} \Big(d(j+1) + f(i,j)\Big)$ et il reste à observer que l'application $\varphi : j \mapsto d(j+1) + f(i,j)$ est décroissante pour conclure. En effet, $\varphi(j) - \varphi(j-1) = d(j+1) - d(j) - w_j - 1$. Or si on considère le découpage optimal de $w_j, \ldots, w_n$ et qu'on élimine le mot w_j, on augmente le déséquilibre de $w_j + 1$ et donc $d(j+1) \leqslant d(j) + w_j + 1$. Ainsi $\varphi(j) \leqslant \varphi(j-1)$, ce qui prouve que le minimum de φ est atteint pour $j = \alpha(i)$.