Chemin dans une matrice

Étant donnée une matrice d'entiers $M = (a_{i,j})$ de taille $n \times k$, on veut connaître un chemin (n'utilisant que des déplacements $\rightarrow$ ou $\downarrow$) de la case en haut à gauche (de coordonnées $(0,0)$) à la case en bas à droite (de coordonnées $(n-1, k-1)$) maximisant la somme des entiers rencontrés (le **poids** du chemin).

1. Quelle serait la complexité d'un algorithme de recherche exhaustive, énumérant tous les chemins possibles de $(0,0)$ à $(n-1, n-1)$? (on suppose pour simplifier que $n = k$, dans cette question)

   Solution : Un chemin de $(0,0)$ à $(n-1, n-1)$ doit effectuer $n-1$ déplacements vers le bas ($\rightarrow$) et $n-1$ vers la droite ($\downarrow$).
   Parmi ces $2n-2$ déplacements, il suffit, pour déterminer le chemin, de choisir ceux qui sont $\rightarrow$ (les $\downarrow$ sont alors déterminés). Il y a donc $\boxed{\binom{2n-2}{n-1}}$ choix possibles.
   En posant $p = n-1$ et en utilisant la formule de Stirling :

   $$\binom{2n-2}{n-1} = \binom{2p}{p} = \frac{(2p)!}{(p!)^2} \sim ... \sim \frac{2^{2p}}{\sqrt{p\pi}}$$

   La complexité d'un tel algorithme serait donc exponentielle...

2. Supposons qu'un chemin C de poids maximum de $(0,0)$ à $(n-1, k-1)$ passe par la case (i,j). Montrer que le sous-chemin de C de $(0,0)$ à (i,j) est de poids maximum (c'est une propriété de **sous-optimalité**).

   Solution : Supposons par l'absurde qu'il existe un chemin de $(0,0)$ à (i,j) de poids strictement supérieur. Alors en concaténant ce chemin avec la partie de C de (i,j) à $(n-1, k-1)$, on contredirait la maximalité de C : c'est absurde.

3. Soit $p_{i,j}$ le poids maximum d'un chemin de $(0,0)$ à (i,j). Donner, en la prouvant, une formule de récurrence sur $p_{i,j}$ pour $i > 0$ et $j > 0$.

   Solution : Un chemin de poids maximum jusqu'à $(i,j) \neq (0,0)$ passe nécessairement par $(i-1,j)$ ou $(i,j-1)$: d'après la question 2, son poids est donc soit $p_{i-1,j} + a_{i,j}$, soit $p_{i,j-1} + a_{i,j}$. D'où :

   $$\boxed{p_{i,j} = \max(p_{i-1,j}, p_{i,j-1}) + a_{i,j}}$$

4. En déduire une fonction récursive simple `poids_max` tel que `poids_max(m, i, j)` renvoie le poids maximum d'un chemin de $(0,0)$ vers (i,j) dans la matrice `m`. Que dire de sa complexité ?

   Solution : La complexité de cette fonction est exponentielle, comme expliqué en question 1.

```python
def poids_max(m, i, j):
    if i == 0 and j == 0: return m[0][0]
    if i == 0: return poids_max(m, 0, j-1) + m[0][j]
    if j == 0: return poids_max(m, i-1, 0) + m[i][0]
    return max(poids_max(m, i-1, j), poids_max(m, i, j-1)) + m[i][j]
```

5. Écrire une fonction `poids_max_dp(m)` donnant le poids maximum d'un chemin de la case en haut à gauche à la case en bas à droite dans la matrice `m`, en utilisant une méthode par programmation dynamique. Comparer sa complexité avec la méthode précédente.

   Solution : La complexité de la fonction suivante est bien meilleure : $\boxed{\mathrm{O}(nk)}$.

```python
def poids_max_dp(m):
    n, k = len(m), len(m[0])
    p = [[0 for _ in range(k)] for _ in range(n)] # matrice n*k remplie de 0
    p[0][0] = m[0][0]
    for i in range(1, n):
        p[i][0] = p[i - 1][0] + m[i][0]
    for j in range(1, k):
        p[0][j] = p[0][j - 1] + m[0][j]
    for i in range(1, n):
        for j in range(1, k):
            p[i][j] = max(p[i - 1][j], p[i][j - 1]) + m[i][j]
    return p[n - 1][k - 1]
```

Remarque : Attention à traiter les cas où $i = 0$ et $j \neq 0$ (et cas symétrique), pour ne pas dépasser de la matrice. Autre solution, par mémoïsation (où on utilise un autre cas de base pour éviter de dépasser de la matrice) :

```python
def poids_max_memo(m):
    n, k = len(m), len(m[0])
    p = {}
    def aux(i, j):
        if (i, j) == (0, 0): return m[0][0]
        if i == -1 or j == -1: return 0
        if not (i, j) in p:
            p[(i, j)] = max(aux(i - 1, j), aux(i, j - 1)) + m[i][j]
        return p[(i, j)]
    return aux(n - 1, k - 1)
```

6. La fonction précédente ne donne que le poids maximum d'un chemin... Expliquer comment faire pour trouver un chemin de poids maximum.

Solution : On peut utiliser une matrice `pere` de même taille que `m` pour stocker telle que `pere[i][j]` soit le couple (i', j') de la case précédent la case (i, j) dans un chemin de poids maximum de $(0, 0)$ à (i, j). On peut alors reconstruire le chemin en remontant la matrice `pere` depuis la case en bas à droite.

7. (à faire seulement si vous avez fini tout le reste) Écrire une fonction `chemin_max_dp(m)` renvoyant la liste des cases d'un chemin de poids maximum de $(0,0)$ à $(n-1, k-1)$ dans la matrice `m`.

Solution :

```python
def chemin_max_dp(m):
    n, k = len(m), len(m[0])
    p = [[0 for _ in range(k)] for _ in range(n)]
    pere = [[(0, 0) for _ in range(k)] for _ in range(n)]
    p[0][0] = m[0][0]
    for i in range(1, n):
        p[i][0] = p[i-1][0] + m[i][0]
        pere[i][0] = (i - 1, 0)
    for j in range(1, k):
        p[0][j] = p[0][j - 1] + m[0][j]
        pere[0][j] = (0, j - 1)
    for i in range(1, n):
        for j in range(1, k):
            if p[i-1][j] > p[i][j - 1]:
                p[i][j] = p[i - 1][j] + m[i][j]
                pere[i][j] = (i-1, j)
            else:
                p[i][j] = p[i][j - 1] + m[i][j]
                pere[i][j] = (i, j - 1)

    # on reconstruit ensuite le chemin en partant de la fin
    chemin = [(n - 1, k - 1)]
    i, j = n-1, k-1
    while (i, j) != (0, 0):
        i, j = pere[i][j]
        chemin.append((i, j))
    return chemin[::-1] # inverse la liste
```