

Devoir surveillé n°2 - Corrigé

Exercice 1 : 1. On considère $L=[6,9,4,8,12]$, donc $\text{len}(L)=5$.

Appel de `mystere(L,2)` :

- $i = 2 \neq (5 - 1)$
 $L[2] \leq L[3]$, donc **appel à `mystere(L,3)`**
- $i = 3 \neq (5 - 1)$
 $L[3] \leq L[4]$, donc **appel à `mystere(L,4)`**
- $i = 4 = (5 - 1)$ donc **le programme s'arrête : `return True`**

Appel de `mystere(L,0)` :

- $i = 0 \neq (5 - 1)$
 $L[0] \leq L[1]$, donc **appel à `mystere(L,1)`**
- $i = 1 \neq (5 - 1)$
 $L[1] > L[2]$, donc **le programme s'arrête : `return False`**

2. Cette fonction détermine si la liste est **triée dans l'ordre croissant à partir de l'indice p** .

3. Voici une fonction non récursive qui réalise la même chose :

```
1 def croissant(L,p):
2     assert p>=0 and p<=len(L)-1
3     if p==len(L)-1:
4         return True
5     for i in range(p, len(L)-1):
6         if L[i]>L[i+1]:
7             return False
8     return True
```

Exercice 2 : 1. (a) Nous créons un dictionnaire vide, et au fur et à mesure du parcours de la liste, nous créons une entrée dans le dictionnaire, ou bien nous actualisons la valeur correspondant à une clef.

```
1 def occurrences(L):
2     d={}
3     for x in L:
4         if x in d:
5             d[x]+=1
6         else:
7             d[x]=1
8     return d
```

(b) Complexité :

.....

.....

.....

.....

.....

2. (a) Comparaison de deux listes de même longueur en utilisant des dictionnaires :

```

1 def compare(L1,L2):
2     d1=occurrences(L1) #on crée les dictionnaires une fois pour toutes
3     d2=occurrences(L2)
4     for cle in d1:
5         if cle not in d2:
6             return False
7         else:
8             if d1[cle]!=d2[cle]:
9                 return False
10    return True

```

(b) Complexité :

.....

.....

.....

.....

.....

.....

Exercice 3 : 1. On considère la matrice

$$M = \begin{pmatrix} 5 & 3 & 2 \\ 1 & 9 & 1 \\ 0 & 2 & 8 \end{pmatrix}.$$

Tous les chemins de $(0,0)$ à $(2,2)$ (on effectue exactement 2 déplacements à droite et 2 déplacements en bas, soit 4 déplacements; l'ordre des déplacements détermine le chemin). Il y a $\binom{4}{2} = 6$ chemins. Les chemins (en notant R pour droite et D pour bas) et leurs poids :

- $RRDD$ ($\rightarrow \rightarrow \downarrow \downarrow$) : cases parcourues 5, 3, 2, 1, 8
somme = $5 + 3 + 2 + 1 + 8 = 19$.
- $RDRD$ ($\rightarrow \downarrow \rightarrow \downarrow$) : cases 5, 3, 9, 1, 8
somme = $5 + 3 + 9 + 1 + 8 = 26$.
- $RDDR$ ($\rightarrow \downarrow \downarrow \rightarrow$) : cases 5, 3, 9, 2, 8
somme = $5 + 3 + 9 + 2 + 8 = 27$.
- $DRRD$ ($\downarrow \rightarrow \rightarrow \downarrow$) : cases 5, 1, 9, 1, 8
somme = $5 + 1 + 9 + 1 + 8 = 24$.
- $DRDR$ ($\downarrow \rightarrow \downarrow \rightarrow$) : cases 5, 1, 9, 2, 8
somme = $5 + 1 + 9 + 2 + 8 = 25$.
- $DDRR$ ($\downarrow \downarrow \rightarrow \rightarrow$) : cases 5, 1, 0, 2, 8
somme = $5 + 1 + 0 + 2 + 8 = 16$.

Donc le poids maximal obtenu est 27, atteint par le chemin $RDDR$ ($\rightarrow \downarrow \downarrow \rightarrow$).

2. (a) Pour aller de $(0,0)$ à $(n-1, n-1)$ on doit effectuer exactement $(n-1)$ déplacements vers la droite et $(n-1)$ déplacements vers le bas, soit au total $2(n-1)$ déplacements.

Choisir un chemin revient donc à choisir où placer les $(n-1)$ déplacements vers le bas parmi les $2(n-1)$ déplacements à faire. (les autres déplacements seront alors automatiquement des déplacements vers la droite). Le nombre de chemins possibles est donc

$$\binom{2(n-1)}{n-1}.$$

(b) Posons $k = n - 1$. On obtient :

$$\binom{2k}{k} = \frac{(2k)!}{k! k!} \sim \frac{\sqrt{2\pi(2k)} (2k)^{2k} e^{-2k}}{(\sqrt{2\pi k} k^k e^{-k})^2} = \frac{\sqrt{4\pi k} (2k)^{2k} e^{-2k}}{2\pi k k^{2k} e^{-2k}} = \frac{(2k)^{2k}}{k^{2k}} \cdot \frac{\sqrt{4\pi k}}{2\pi k}.$$

Simplifions :

$$\frac{(2k)^{2k}}{k^{2k}} = (2^2)^k = 4^k, \quad \frac{\sqrt{4\pi k}}{2\pi k} = \frac{2\sqrt{\pi k}}{2\pi k} = \frac{1}{\sqrt{\pi k}}.$$

Donc $\binom{2k}{k} \sim \frac{4^k}{\sqrt{\pi k}}$, et en remplaçant $k = n - 1$:

$$\boxed{\binom{2(n-1)}{n-1} \sim \frac{4^{n-1}}{\sqrt{\pi(n-1)}}}.$$

Complexité de l'approche exhaustive :

Un algorithme qui énumère tous les chemins et calcule la somme pour chacun aura une complexité proportionnelle au nombre de chemins fois le coût par chemin. Chaque chemin comporte $2n - 1$ cases visitées (ou $2(n-1)$ déplacements), donc un coût $O(n)$ par chemin. Ainsi la complexité temporelle est $O\left(n \cdot \frac{4^{n-1}}{\sqrt{\pi(n-1)}}\right)$ qui est *exponentielle* en n .

On peut résumer en disant que l'approche exhaustive a une complexité exponentielle en n .

3. (a) Une implémentation simple en Python :

```
1 def max2(a, b):
2     if a >= b:
3         return a
4     else:
5         return b
```

(b) **Cas terminal** : si $(i, j) = (n - 1, n - 1)$ alors `poids_max(n-1,n-1)=M[n-1][n-1]`.

(c) **Bord droit** : si $i = n - 1$ et $j \neq n - 1$ (dernière ligne, on ne peut que faire des déplacements à droite), alors `poids_max(i,j)=M[i][j]+poids_max(i,j+1)`.

(On ajoute la valeur courante et on continue uniquement vers la droite.)

(d) **Bord bas** : si $j = n - 1$ et $i \neq n - 1$ (dernière colonne, on ne peut que descendre), alors `poids_max(i,j)=M[i][j]+F(i+1,j)`.

(e) **Cas général** : si $i \neq n - 1$ et $j \neq n - 1$, on a le choix entre aller à droite ou aller en bas, donc `poids_max(i,j)=M[i][j] + max2(poids_max(i,j+1),poids_max(i+1,j))`.

(f) Voici un script Python qui implémente `poids_max(M,i,j)` de façon récursive et qui utilise un dictionnaire `D` pour mémoriser les sous-résultats.

```
1 D={} #dictionnaire pour la mémorisation;
2     #clef = (i,j), valeur = poids_max à partir de (i,j)
3 def poids_max(M,i,j):
4     if (i,j) in D:
5         return D[(i,j)]
6     n=len(M)
7     if i==n-1 and j==n-1:
8         D[(i,j)]=M[i][j]
9         return D[(i,j)]
10    if i==n-1 and j!=n-1:
11        D[(i,j)]=M[i][j]+poids_max(M, i, j+1)
12        return D[(i,j)]
13    if i!=n-1 and j==n-1:
14        D[(i,j)]=M[i][j]+poids_max(M, i+1, j)
15        return D[(i,j)]
16    D[(i,j)]=M[i][j]+max2(poids_max(M, i+1, j),poids_max(M, i, j+1))
17    return D[(i,j)]
```

(g) Pour obtenir le poids maximal global, appeler `poids_max(M,0,0,)`.

Remarque : Grâce à la mémoïsation, chaque sous-problème (i, j) est calculé au plus une fois. Il y a n^2 sous-problèmes pour une matrice $n \times n$. La complexité temporelle est donc $\Theta(n^2)$ (chaque calcul faisant un nombre constant d'opérations outre les appels mémorisés). La mémoire utilisée est aussi $\Theta(n^2)$ pour stocker le dictionnaire `memo` dans le pire cas. Si on compare avec l'approche exhaustive, c'est une amélioration massive