

Devoir n°1

Exercice : dynamique unidimensionnelle

On considère la suite de Fibonacci (a_n) définie par $a_0 = a_1 = 1$ et la relation de récurrence

$$\forall n \geq 2, a_n = a_{n-1} + a_{n-2}$$

1. Calculer a_n pour $n = 0, \dots, 10$.
2. Programmation récursive.

(a) La fonction récursive est naturelle à écrire :

```
def a(n):
    if n == 0 or n == 1:
        return 1
    return a(n-1) + a(n-2)
```

Si $n \in \mathbb{N}$, on note $U(n)$ le nombre d'appels nécessaires pour calculer a_n .

On a $U(0) = 1$ et $U(1) = 1$ puisque l'on obtient la valeur de a_0 et a_1 en n'effectuant qu'un seul appel initial, la valeur étant renvoyée dès le premier test d'arrêt.

(b) Montrer que si $n \geq 2$, on a

$$U_n = 1 + U_{n-1} + U_{n-2}$$

On pose $V_n = U_n + 1$. Vérifier que $V_n = V_{n-1} + V_{n-2}$.

Remarques ? En déduire que

$$\forall n \in \mathbb{N}, U(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^{n+1} - \left(\frac{1 - \sqrt{5}}{2} \right)^{n+1} \right] - 1$$

Montrer alors que

$$U(n) \sim \frac{2}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^{n+1}$$

Quelle est la nature de la complexité en appels récursifs ? Conclusion ?

3. Programmation dynamique avec mémorisation partielle.

Pour calculer a_n , nous avons seulement besoin de connaître les deux termes précédent. Compléter (et vérifier le fonctionnement) la fonction suivante qui programme cette idée :

```
def a_dyn(n):
    x, y = 1, 1 # représentent a_0 et a_1
    for i in range(?):
        x, y = y, x + y # affectation en parallèle
    return ?
```

Quelle est la complexité d'une telle fonction ?

À votre avis, peut-on l'améliorer ?

4. Programmation dynamique avec mémorisation totale.

On considère la suite de Catalan définie par $C_0 = 1$ et la relation de récurrence

$$\forall n \in \mathbb{N}^*, C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

(a) Calculer C_1, C_2, C_3 et C_4 .

(b) Sachant que $C_n \sim \frac{4^n}{\sqrt{\pi n^{3/2}}}$, quelle sera la complexité en appels récursifs d'une programmation récursive ?

(c) On propose alors une programmation dynamique avec mémorisation. Compléter alors la fonction suivante qui programme le calcul de C_n de façon dynamique :

```
def C_dyn(n):
    C = [0] * (n+1)
    C[0] = 1
    for i in range(1, n+1):
        s = 0
        for k in range(?):
            s += C[?] * C[?]
        C[?] = ?
    return ?
```

Déterminer la complexité de cette fonction et conclure.

Exercice : dynamique bidimensionnelle

Un élève de CPGE souhaite gagner de l'argent de poche. Il peut y consacrer un maximum de H heures par semaine. Il a repéré n emplois. Mais le salaire offert pour chaque emploi n n'est pas proportionnel au nombre d'heure travaillées. Par exemple, la tableau suivant agrège les salaires $s(h, j)$ de l'emploi j si on travaille h heures ($T = 4, n = 4$) :

emploi : j		1	2	3	4
heures de travail : h					
0		0	0	0	0
1		26	24	16	20
2		38	34	32	34
3		47	42	48	48
4		50	49	64	52

1. La méthode gloutonne est directe (et donc pas très intéressante à programmer ici), elle consiste sur l'exemple à choisir les 4 heures qui payent le plus, soit l'emploi 3. Ce choix est-il pertinent ?

2. On note $P(h, j)$ le salaire maximum perçu possible en travaillant, de façon donc optimale, h heures parmi les emplois de 1 à j .

On pose aussi $P(0, j) = 0$ si $1 \leq j \leq n$.

- Quelle valeur de $P(h, j)$ doit-on calculer pour résoudre le problème ?
- Que vaut $P(h, 1)$ pour $0 \leq h \leq H$?
- Soit $2 \leq j \leq n$. Montrer que si on travaille h heures parmi les emplois de 1 à j de façon optimisée, alors en notant k le nombre d'heures travaillées parmi les emplois parmi 1 à $j - 1$, le salaire de ces k heures parmi les emplois de 1 à $j - 1$ est aussi optimisé.
- En déduire la formule si $2 \leq j \leq n, 0 \leq h \leq H$,

$$P(h, j) = \max_{k=0 \dots h} P(k, j-1) + s(h-k, j)$$

- Oubliant les problèmes dus à la récursivité, on écrit une fonction récursive permettant d'appliquer la formule de récursivité ci-dessus.

- Compléter et commenter la fonction suivante, pour quelle fonctionne avec s défini par

$s = [[0, 0, 0, 0], [26, 24, 16, 20], [38, 34, 32, 34], [47, 42, 48, 48], [50, 49, 64, 52]]$

c'est à dire que les emplois sont numérotés 0 à 3.

```
def p_rec(s, h, j):
    if j == 0:
        return ?
    m = s[0][j] # k=0
    for k in range(1, h+1):
        p = p_rec(s, ?, j-1) + s[?][?]
        if p > m:
            m = ?
    return ?
```

Pour l'exemple, on exécute donc

$p_rec(s, 4, 3)$

pour obtenir le résultat.

- Pour évaluer la complexité de cette fonction récursive, on note $C(h, j+1)$ le nombre d'appels à la fonction récursive pour calculer $P(h, j)$. Nous avons

$$C(h, 1) = 1 \quad C(0, j) = 1$$

Montrer que si $j \geq 2$,

$$C(h, j) = 1 + \sum_{k=1}^h C(k, j-1) = \sum_{k=0}^h C(k, j-1)$$

En déduire que :

$$C(T, 2) = T + 1 \quad C(T, 3) = \frac{(T+2)(T+2)}{2} \quad C(T, 4) = \frac{(T+1)(T+2)(T+3)}{6}$$

On conjecture alors la formule (si $n = 1$, produit vide égal à 1)

$$C(T, n) = \frac{(T+1) \cdots (T+n-1)}{(n-1)!} = \binom{T+n-1}{T} = \binom{T+n-1}{n-1}$$

et une preuve par récurrence sur n conduit, pour prouver l'hérédité, à montrer la formule

$$\sum_{k=0}^T \binom{k+n-1}{n-1} = \binom{T+n}{n}$$

iii. Montrer que la formule ci-dessus est vraie, par récurrence sur T .

- (f) La méthode par récursivité n'est pas efficace. Nous procédons alors par programmation dynamique, à l'aide d'un tableau bi-dimensionnel. Compléter et commenter la fonction suivante, qui renvoie le salaire maximisé :

```
def p_dyn(s):
    H = len(s) - 1
    n = len(s[0])
    p = [[0]*n for _ in range(H+1)]
    for h in range(H+1):
        p[h][0] = ?
        for j in range(1, n):
            for h in range(H+1):
                m = -1
                for k in range(h+1):
                    pp = p[k][j-1] + s[h-k][j]
                    if pp > m:
                        m = ?
                p[h][j] = ?
    return ?
```

- (g) Appliquer la méthode à l'exemple. On obtiendra le tableau p suivant qu'il faudra établir complètement :

heures de travail : h \ emploi : j	1	2	3	4
0	0	0	0	0
1	26	26	26	26
2	?	?	?	?
3	?	?	?	?
4	50	?	?	86

- (h) Déterminer la complexité de la fonction `p_dyn`. On considérera que le corps de la boucle en k est à temps constant.
- (i) Peut-on à l'aide du tableau p déterminer les emplois avec leurs nombres d'heures qui permettent d'optimiser le salaire ?
 Que faut-il ajouter dans la fonction `p_dyn` pour pouvoir effectuer un backtracing ?
 Compléter la fonction pour qu'elle renvoie la liste des emplois avec le nombre d'heures pour optimiser le salaire.
 Appliquer la méthode à l'exemple.

