Complexité
TD 4 - Correction

Master 1 Informatique (2022-2023)

Serge Miguet, Guillaume Metzler, Tess Masclef
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
serge.miguet@univ-lyon2.fr guillaume.metzler@univ-lyon2.fr
tess.masclef@univ-lyon2.fr

La suite de Fibonacci

Une expression récursive et étude de la complexité. D'après la définition qui nous est donnée, on obtient le n-ème terme de la suite de Fibonacci en additionant les deux termes précédents, *i.e.* les $n-1$-ème et $n-2$-ème termes. Si on note F_n le n-ème terme de la suite de Fibonacci, nous avons alors la relation de récurrence suivante :

$$F_{n+1} = F_n + F_{n-1}.$$

Le processus de récursivité est présenté en par l'algorithme 1.

Regardons maintenant la complexité de cet algorithme et évaluons le nombre d'appel de Fibo(n) nécessaire au calcul du n-ème terme de la suite de Fibonacci, notons A_n cette valeur. Nous avons alors :

$$\begin{aligned} A_0 &= 1, \\ A_1 &= 1. \end{aligned}$$

En effet, dans les deux premiers cas, on appelle une seule fois la fonction Fibo. De même, nous avons

Algorithm 1: Une approche récursive

Paramètre : Un entier $n \geq 0$
Sortie : Valeur de F_n
Fibo(n)
1: **if** $n \leq 1$ **then**
2: retourner n
3: **else**
4: retourner Fibo($n-1$) + Fibo($n-2$)
5: **end if**=0

$$\begin{aligned} A_2 &= 3 = A_1 + A_0 + 1, \\ A_3 &= 1 = A_2 + A_1 + 1, \end{aligned}$$

où le terme +1 correspond à l'appel de la fonction Fibo pour initier le calcul et qui nécessite de alors autant d'opérations que pour le calcul des valeurs de la suite pour les deux termes précédents. Plus généralement, on a donc

$$A_{n+1} = A_n + A_{n-1} + 1. \tag{1}$$

On pourra également noter que le nombre d'appels à la fonction correspond au nombre de sommes effectuées pour la calcul de la $n+1$-ème valeur de la suite.

Regardons maintenant un équivalent asymptotique de cette suite $(A_n)_{n\mathbb{N}}$. Pour cela on va déterminer une expression de la suite $(A_n)_{n\mathbb{N}}$.

On s'intéresse pour cela à l'équation homogène

$$A_{n+1} - A_n - A_{n-1} = 0.$$

On considère le polynôme caractéristique associé à cette équation homogène

$$x^2 - x - 1$$

dont les racines sont $\dfrac{1+\sqrt{5}}{2}$ (ce nombre est communément appelé φ et s'appelle le nombre d'or) et $\dfrac{1-\sqrt{5}}{2}$. On a donc l'expression suivante de la suite A_n

$$C_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + C_2 \left(\frac{1-\sqrt{5}}{2}\right)^n + p_n,$$

où C_1 et C_2 sont des constantes réelles à déterminer et p_n est une solution particulière de l'équation (1).

L'équation non homogène fait intervenir un terme constant (+1),on va donc chercher cette solution sous la forme d'une constante c qui vérifie l'équation (1), soit

$$c = 2c + 1 \implies c = -1 \implies p_n = 1, \quad \forall n \in \mathbb{N}.$$

Donc pour tout $n \geq 1$, nous avons

$$C_1 \left(\frac{1+\sqrt{5}}{2} \right)^n + C_2 \left(\frac{1-\sqrt{5}}{2} \right)^n - 1.$$

On déterminera les constantes à l'aide des conditions initiales de notre problème, mais cela importe peu ici.

On a plus précisément :

$$\begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \begin{pmatrix} \frac{1+\sqrt{5}}{2} \times \frac{2}{\sqrt{5}} \\ -\frac{1-\sqrt{5}}{2} \times \frac{2}{\sqrt{5}} \end{pmatrix}$$

Ce qui nous intéresse, c'est le comportement asymptotique de la suite $(A_n)_{n\in\mathbb{N}}$. Notons que $\left| \frac{1-\sqrt{5}}{2} \right|$ est un nombre plus petit que 1, on a donc

$$A_n = \mathcal{O}\left(\frac{2}{\sqrt{5}} \varphi^{n+1} \right).$$

Ce qui montre que la complexité de ce problème est bien exponentielle.

Remarque : ce développement, et plus particulièrement l'équation homogène, permet également de déterminer une expression analytique des termes de la suite de Fibonacci. De plus, pour trouver une expression de la suite $(A_n)_{n\in\mathbb{N}}$ nous aurions également pu passer par la suite auxiliaire définie par $B_n = A_n + 1$.

Nous aurions également plus procéder plus simplement en déterminant une borne inférieure sur la complexité, à l'aide de la relation de récurrence. Cela nous donne

$$A_n = A_{n-1} + A_{n-2} + 1,$$
$$\downarrow \ (A_n)_{n\in\mathbb{N}} \text{ est croissante}$$

$$
\begin{aligned}
&\geq 2A_{n-2} + 1 \\
&= 2(A_{n-3} + A_{n-4} + 1) + 1, \\
&\quad \downarrow \ (A_n)_{n\in\mathbb{N}} \text{ est croissante} \\
&\geq 4A_{n-4} + 3, \\
&= 4(A_{n-5} + A_{n-6} + 1) + 3, \\
&\quad \downarrow \ (F_n)_{n\in\mathbb{N}} \text{ est croissante} \\
&\geq 8A_{n-6} + 7, \\
&\geq \dots, \\
&\geq 2^k A_{n-2k} + 2^k - 1.
\end{aligned}
$$

On peut continuer le processus jusqu'à ce que k atteigne la valeur $n/2$, auquel cas $n - 2k = 0$ et on retombe sur le premier terme de la suite de Fibonacci. Ainsi, pour cette valeur, nous avons

$$
\begin{aligned}
A_n &\geq 2^{n/2} A_0 + 2^{n/2} - 1, \\
&\quad \downarrow \text{ or } A_0 = 1 \\
A_n &= 2^{n/2} + 2^{n/2} - 1, \\
&\quad \downarrow \text{ on s'affranchit du dernier terme} \\
&\geq \sqrt{2}^n.
\end{aligned}
$$

Ce qui permet d'en déduire que le processus récursif de calcul des termes de la suite de Fibonacci est au moins exponentiel, *i.e.* $\mathcal{O}(\sqrt{2}^n)$.

Cette version n'est pas optimale car elle nécessite de recalculer plusieurs fois le même terme de cette suite, comme on peut le voir en Figure **??**

On comprend alors que l'idéal serait de pouvoir conserver les termes déjà calculés pour réduire le temps d'éxécution, ce qui va nécessité un brin de mémoire.

Une version par recensement L'approche par recensement est similaire à l'approche récursive précédement étudiée. On va cependant tirer profit des calculs qui sont déjà effectués pour éviter de répéter des calculs qui sont déjà effectués. Cela va nécessiter de stocker des résultats intermédiaires dans une table de taille n. Cette nouvelle procédure aura alors une complexité en mémoire en $\mathcal{O}(n)$.
Sa complexité sera également linéaire en ce qui concerne le nombre d'opérations effectuées car nous pourrons faire appel aux valeurs stockées dans une table pour éviter de calculer les différents termes. Il faudra simplement remplir un tableau avec un décalage à chaque fois et ensuite parcourir les éléments dans un tableau mais dans un ordre donné à chaque itération (coût global de n).

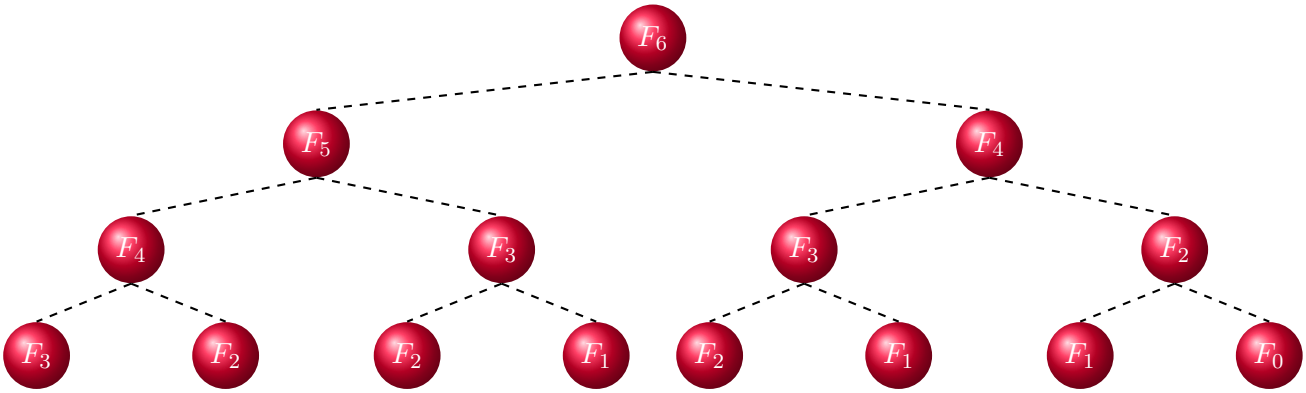

Figure 1: Représentation sous forme d'arbre des appels à la fonction *Fibo*.

D'un point de vue algorithmique, on peut éditer un pseudo-code très proche. de ce qui précède comme celui présenté par l'Algorithme 2

Algorithm 2: Une approche par recensement

Paramètre: Un entier $n \geq 0$
Sortie : Valeur de F_n
Table de Stockage : T de taille n Fibo(n)

```
1: if n ≤ 1 then
2:     retourner n
3: else
4:     retourner Fibo(n − 1) + Fibo(n − 2) T[n − 1] = Fibo(n − 1)
5: end if=0
```

Une version linéaire On peut aisément créer une version linéaire de cet algorithme en gardant en mémoire les valeurs précédentes. Ainsi, le nombre d'affectations évoluera linéairement au cours du temps (même si plusieurs affections seront à effectuer au cours du temps). Le processus est décrit en Algorithme 3

Cet algorithme effetue une addition et 3 affectations par passage dans la boucle (complexité linéaire).

Calcul d'une puissance de matrice On considère la matrice X de taille 2×2 définie par

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

On peut alors calculer le carré et le cube de cette matrice qui sont respectivement données par

Algorithm 3: Une approche itérative

Paramètre : Un entier $n \geq 0$
Sortie : Valeur de F_n
Fibo(n)
1: **if** $n \leq 1$ **then**
2: retourner n
3: **else**
4: u = 1
5: v =1
6: **for** $k = 2$ à n **do**
7: T prend la valeur $u + v$
8: v prend la valeur T
9: u prend la valeur v
10: **end for**
11: retourner la valeur de T
12: **end if**=0

$$X^2 = XX = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$$

et

$$X^3 = XX^2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}.$$

On peut se demande maintenant pourquoi on s'intéresse aux puissances de cette matrice. Pour cela remarquons que, pour tout $n \geq 1$, l'on peut adopter l'écriture matricielle de notre relation de récurrence :

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_n \end{pmatrix}.$$

Par exemple, pour $n = 1$ et $n = 2$ nous avons respectivement

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} = X \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

et

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = X \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = X^2 \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}.$$

On montre alors par récurrence que pour tout $n \geq 1$, nous avons :

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = X^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}.$$

Cette dernière formulation est beaucoup plus pratique car elle ne nécessite de ne calculer qu'une puissance de matrice pour déterminer les valeurs successives de la suite $(F_n)_{n\in\mathbb{N}}$. Quant au calcul des puissances de cette matrice ... penser à la diagonalisation de matrice.

Enfin, elle permet également de mettre en évidence la relation suivante

$$X^n = \begin{pmatrix} F_{n-1} & F_n \\ F_n & F_{n+1} \end{pmatrix}.$$

En revanche, cela ne sera toujours pas la solution la plus optimale et on pourrait réduire le nombre d'opérations à effectuer en utilisant la parité de n.

Une version logarithmique. Il s'agit ici de calculer les puissances de X en limitant le nombre d'opérations, plus précisément le nombre de multiplications. On pourra ainsi calculer X^n de la façon suivante :

$$\forall n \geq 2,\ X^n = \begin{cases} (XX)^{n/2} & \text{si } n \text{ est pair} \\ X(XX)^{n/2} & \text{si } n \text{ est impair} \end{cases},$$

où $n/2$ désigne le quotient de division euclidienne de n par 2. Cette approche nécessite de précalculer les valeurs de la matrice XX.

Il reste à étudier la complexité de cet algorithme en montrant qu'elle est bien logarithmique, mais cela se voit directement avec la relation précédente.