Exponentiation rapide

Marc Lorenzi

13 août 2024

1 Introduction

1.1 Développement binaire des entiers

Tout entier $n \in \mathbb{N}$ possède un *déceloppement binaire* : il existe une unique suite presque nulle $(b_k)_{k\geq 0}$ d'éléments de $\{0,1\}$ telle que

$$n = \sum_{k=0}^{\infty} b_k 2^k$$

Pour tout $k \in \mathbb{N}$, b_k est le k^{e} *bit* de n. Soit $\mu : \mathbb{N} \longrightarrow \mathbb{N}$ définie par

$$\mu(n) = \min\{m \in \mathbb{N} : \forall k \geq m, b_k = 0\}$$

On a $\mu(0) = 0$. Si $n \geq 1$, on a $b_{\mu(n)-1} = 1$ et pour tout $k \geq \mu(n)$, $b_k = 0$. L'entier $\mu(n)$ est le nombre de chiffres du développement binaire de n. On peut donc aussi écrire

$$n = \sum_{k=0}^{\mu(n)-1} b_k 2^k$$

L'égalité ci-dessus reste vérifiée si $n = 0$ en convenant qu'une somme vide est nulle.

1.2 La fonction mu

Soit $n \in \mathbb{N}^*$. On a

$$2^{\mu(n)-1} \leq n \leq \sum_{k=0}^{\mu(n)-1} 2^k = 2^{\mu(n)} - 1$$

En passant aux logarithmes en base 2 et en ajoutant 1,

$$\mu(n) \leq 1 + \lg n < \mu(n) + 1$$

et donc, puisque $\mu(n) \in \mathbb{N}$,

$$\mu(n) = 1 + \lfloor \lg n \rfloor$$

Voici le graphe de μ sur $[\![0, 128]\!]$.

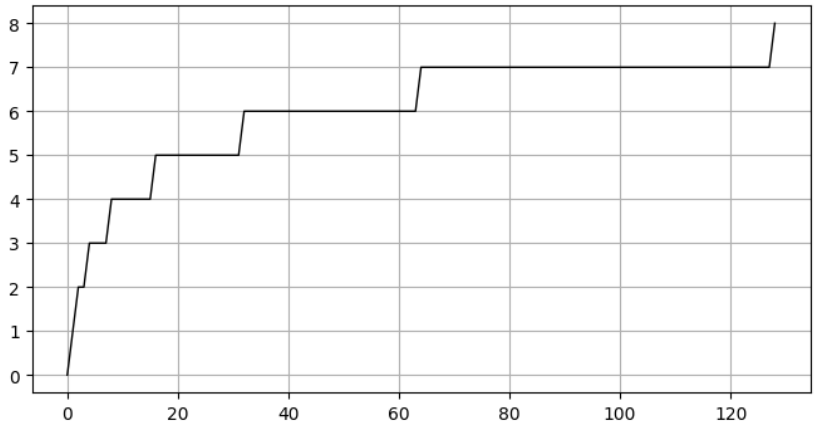

FIGURE 1 – La fonction μ

1.3 La fonction gamma

Soit $\gamma : \mathbb{N} \longrightarrow \mathbb{N}$ définie pour tout $n \in \mathbb{N}$ par

$$\gamma(n) = \sum_{k=0}^{\infty} b_k = \sum_{k=0}^{\mu(n)-1} b_k$$

L'entier $\gamma(n)$ est la somme des bits de n, ou encore le nombre de bits de n qui sont égaux à 1.

On a $\gamma(0) = 0$. Remarquons que pour tout $n \geq 1$, $1 \leq \gamma(n) \leq \mu(n)$. Voici le graphe de γ sur $[\![0, 128]\!]$. En rouge, les graphes de μ et de la fonction constante égale à 1.

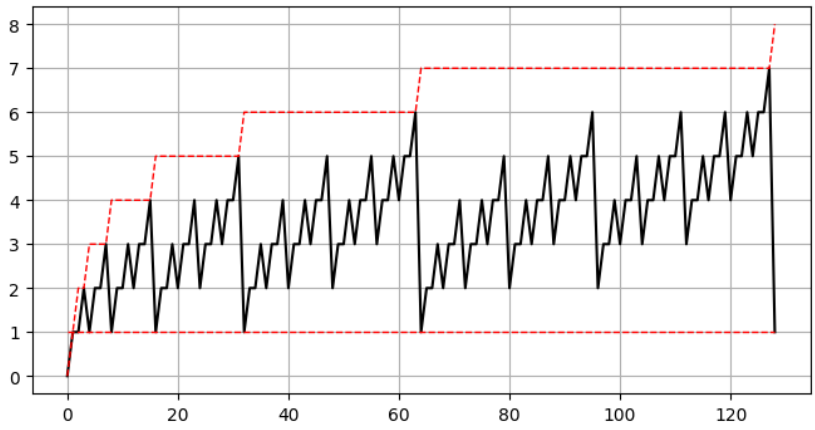

FIGURE 2 – La fonction γ

2 Exponentiation rapide

2.1 L'idée

Soit $(E, \times)$ un monoïde de neutre 1. Pour tout $x \in E$ et tout $n \in \mathbb{N}$, on définit x^n par récurrence sur n en posant $x^0 = 1$ et pour tout $n \in \mathbb{N}^*$, $x^n = x^{n-1} \times x$. Une utilisation naïve de cette définition fournit un algorithme de calcul de x^n qui nécessite d'effectuer n multiplications.

En utilisant le développement binaire de n, on peut calculer x^n avec seulement un nombre de multiplications en $O(\lg n)$. L'idée est la suivante. On a

$$x^n = x^{\sum_{k=0}^{\mu(n)-1} b_k 2^k} = \prod_{k=0}^{\mu(n)-1} \left(x^{2^k}\right)^{b_k}$$

Le produit ci-dessus possède $\mu(n)$ facteurs. Remarquons que

$$x^{2^{k+1}} = \left(x^{2^k}\right)^2$$

On peut donc calculer les puissances x^{2^k} dans une boucle comportant $\mu(n)$ itérations. Il ne reste plus qu'à voir comment obtenir dans cette même boucle les bits b_k de de l'entier n.

2.2 Une fonction Python

Dans la suite nous supposons donnée une classe Python $\mathcal{C}$ dont les éléments sont capables de se multiplier entre-eux via l'opérateur `*`. On suppose également que pour tout objet y de la classe $\mathcal{C}$, `1 * y` a un sens et renvoie y. C'est par exemple le cas pour les classes des entiers, des flottants, des rationnels, des complexes, etc.

La fonction Python `pow` prend deux paramètres $x \in \mathcal{C}$ et $n \in \mathbb{N}$.

```
1 def pow(x, n):
2     y, z = x, 1
3     while n != 0:
4         if n & 1 == 1: z = z * y
5         y = y * y
6         n = n >> 1
7     return z
```

Nous allons montrer que l'évaluation de `pow(x,n)` renvoie x^n.

2.3 Terminaison

Soient $x \in \mathcal{C}$ et $n \in \mathbb{N}$. Soit $N \in \mathbb{N} \cup \{+\infty\}$ le nombre d'évaluations réussies de la ligne 3 lors de l'évaluation de `pow(x,n)`. La valeur $N = +\infty$ signifie que la boucle `while` ne termine pas.

Proposition 1. $N = \mu(n)$.

Dit autrement, la boucle `while` termine après $\mu(n)$ itérations.

Si $n = 0$, la première évaluation de la ligne 3 échoue. On a donc $N = 0 = \mu(0)$. Supposons dorénavant $n \geq 1$.

Numérotons les évaluations de la ligne 3 par les entiers $j = 0, 1, 2, \ldots$ Notons E l'ensemble des valeurs prises par j.

- Si $N < +\infty$, $E = [\![0, N]\!]$. Pour tout $j \in [\![0, N-1]\!]$, la j^{e} évaluation de la ligne 3 réussit, et la N^{e} évaluation échoue.
- Si $N = +\infty$, $E = \mathbb{N}$. Pour tout $j \in \mathbb{N}$, la j^{e} évaluation de la ligne 3 réussit.

Pour tout $j \in E$, notons n_j, y_j et z_j les valeurs des variables n, y et z *avant* la j^{e} évaluation de la ligne 3.

On a $n_0 = n$. Pour tout $j < N$, la j^{e} évaluation de la ligne 3 réussit et la ligne 6 nous dit que $n_{j+1} = n_j \gg 1$.

Lemme 2. *Pour tout $j < \mu(n)$, on a aussi $j < N$ et*

$$n_j = \sum_{k=j}^{\mu(n)-1} b_k 2^{k-j}$$

Démonstration. Montrons-le par récurrence sur n. Tout d'abord, comme $n \geq 1$, l'évaluation de la ligne 3 réussit et donc $0 < N$. De plus,

$$n_0 = n = \sum_{k=0}^{\mu(n)-1} b_k 2^{k-0}$$

La propriété est donc vérifiée pour $j = 0$. Soit $j < \mu(n) - 1$. Supposons que $j < N$ et

$$n_j = \sum_{k=j}^{\mu(n)-1} b_k 2^{k-j}$$

Par l'hypothèse de récurrence, $j < N$ et donc l'évaluation de la ligne 3 réussit. De plus, en décalant les bits de n_j de 1 bit vers la droite,

$$n_{j+1} = n_j \gg 1 = \sum_{k=j+1}^{\mu(n)-1} b_k 2^{k-j-1}$$

Comme $j + 1 \leq \mu(n) - 1$, $n_{j+1} \geq b_{\mu(n)-1} = 1 > 0$ et donc $j + 1 < N$. □

Nous pouvons maintenant montrer la proposition 1. Prenons $j = \mu(n) - 1$ dans le lemme 2. On a $\mu(n) - 1 < N$ et

$$n_{\mu(n)-1} = b_{\mu(n)-1} = 1$$

Une nouvelle itération donne $n_{\mu(n)} = 0$. L'évaluation suivante de la ligne 3 échoue, donc la boucle `while` termine. Le nombre d'itérations effectuées est $N = \mu(n) - 1 + 1 = \mu(n)$.

2.4 Correction

Soient $x \in \mathcal{C}$ et $n \in \mathbb{N}$. Nous allons montrer que l'évaluation de `pow(x,n)` renvoie x^n. Pour tout $j \in \mathbb{N}$, posons

$$S_j(n) = \sum_{k=0}^{j-1} b_k 2^k$$

Les entiers $S_j(n)$ sont les *sommes partielles* du développement binaire de n. On a en particulier $S_0(n) = 0$, pour tout $j \in \mathbb{N}$, $S_{j+1}(n) = S_j(n) + b_j 2^j$ et pour tout $j \geq \mu(n)$, $S_j(n) = n$.

Lemme 3. *Pour tout* $j \in [\![0, \mu(n)]\!]$,

$$y_j = x^{2^j} \text{ et } z_j = x^{S_j(n)}$$

Démonstration. Faisons une récurrence sur j. Tout d'abord, $y_0 = x = x^{2^0}$ et $z_0 = 1 = x^{S_0(n)}$. Soit $j < \mu(n)$. Supposons que $y_j = x^{2^j}$ et $z_j = x^{S_j(n)}$. Comme $j < \mu(n)$, la j^{e} évaluation de la ligne 3 réussit.

Lors de l'évaluation de la la ligne 4, on effectue un test sur la valeur de $n_j \ \& \ 1$, c'est à dire sur le *bit de poids faible* de n_j, qui est égal (lemme 2) à b_j. Si $b_j = 1$ alors $z_{j+1} = z_j y_j$. Sinon, $b_j = 0$ et $z_{j+1} = z_j$. On a donc dans tous les cas

$$z_{j+1} = z_j y_j^{b_j} = x^{S_j(n)} x^{b_j 2^j} = x^{S_j(n) + b_j 2^j} = x^{S_{j+1}(n)}$$

Enfin, l'évaluation de la ligne 5 donne

$$y_{j+1} = y_j^2 = \left(x^{2^j}\right)^2 = x^{2^{j+1}}$$

□

Proposition 4. $z_{\mu(n)} = x^n$.

Démonstration. Prenons $j = \mu(n)$ dans le lemme 3 pour obtenir

$$z_{\mu(n)} = x^{S_{\mu(n)}(n)} = x^n$$

□

Comme l'évaluation de `pow(x,n)` renvoie $z_{\mu(n)}$, la correction de la fonction `pow` est montrée.

2.5 Complexité

Pour tout $x \in \mathcal{C}$ et tout $n \in \mathbb{N}$, notons $T(x,n)$ le nombre de multiplications effectuées lors de l'évaluation de `pow(x,n)`.

Proposition 5. *Pour tout $x \in \mathcal{C}$ et tout $n \in \mathbb{N}$,*

$$T(x,n) = \mu(n) + \gamma(n)$$

Démonstration. Soit $x \in \mathcal{C}$. Soit $n \in \mathbb{N}$. Pour tout $j \in [\![0, \mu(n)-1]\!]$, la j^{e} itération de la boucle `while` effectue

- Une multiplication à la ligne 5 (`y = y * y`).
- b_j multiplication à la ligne 4 (`z = z * y`).

On a donc

$$T(x,n) = \sum_{j=0}^{\mu(n)-1} (1 + b_j) = \sum_{j=0}^{\mu(n)-1} 1 + \sum_{j=0}^{\mu(n)-1} b_j = \mu(n) + \gamma(n)$$

□

L'entier $T(x,n)$ ne dépend donc pas de x, mais seulement de n.

Remarquons que $T(x,n) = \mu(n) + \gamma(n) \leq 2\mu(n)$. On a égalité si et seulement si $\gamma(n) = \mu(n)$, c'est à dire

$$n = \sum_{k=0}^{\mu(n)-1} 2^k = 2^{\mu(n)} - 1$$

Si $n \geq 1$, $T(x,n) \geq \mu(n) + 1$. On a égalité si et seulement si $\gamma(n) = 1$, c'est à dire

$$n = 2^{\mu(n)-1}$$

Voici le graphe de $T(x,n)$ en fonction de n pour $n \in [\![0, 128]\!]$. En rouge, le minorant $\mu(n)+1$ (valable pour $n \neq 0$) et le majorant $2\mu(n)$.

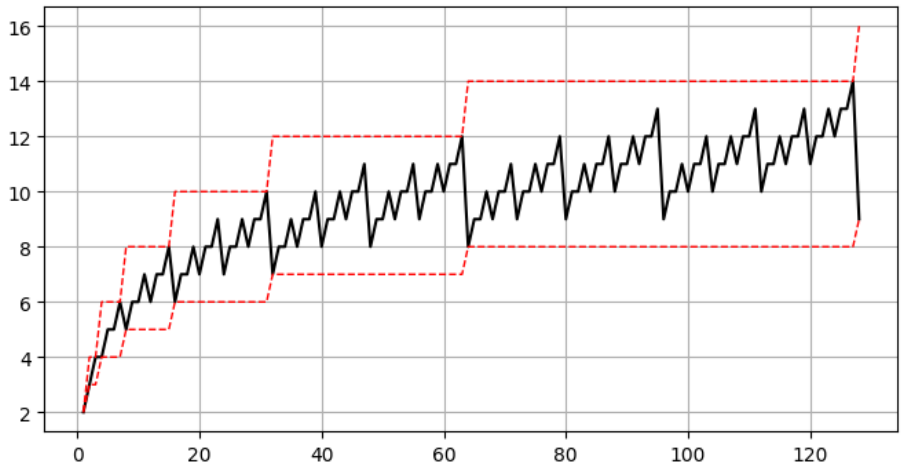

FIGURE 3 – La fonction $n \longmapsto T(x, n)$

3 L'exemple des nombres de Fibonacci

Comme exemple d'utilisation de l'exponentiation rapide, nous allons voir comment il est possible de calculer les nombres de Fibonacci en se se ramenant au calcul des puissances d'un certain élément d'un anneau judicieux.

3.1 La suite de Fibonacci

On définit par récurrence à deux termes sur n la suite $(F_n)_{n\in\mathbb{N}}$ des *nombres de Fibonacci*. Celle-ci est définie par $F_0 = 0$, $F_1 = 1$ et pour tout $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$. Les premiers termes de la suite sont donnés dans le tableau ci-dessous.

n	0	1	2	3	4	5	6	7	8	9	10
F_n	0	1	1	2	3	5	8	13	21	34	55

3.2 L'anneau $\mathbb{Z}[\varphi]$

L'équation $x^2 = x + 1$, d'inconnue $x \in \mathbb{R}$, possède deux solutions qui sont $(1 \pm \sqrt{5})/2$. Notons φ l'une de ces deux solutions. Soit

$$\mathbb{Z}[\varphi] = \{a + b\varphi : (a, b) \in \mathbb{R}^2\}$$

Proposition 6. *$\mathbb{Z}[\varphi]$ est un sous-anneau de $\mathbb{R}$.*

Démonstration. Comme $\varphi \in \mathbb{R}$, $\mathbb{Z}[\varphi] \subseteq \mathbb{R}$. On a $1 = 1 + 0\varphi \in \mathbb{Z}[\varphi]$. Pour tous $a, b, c, d \in \mathbb{R}$,

$$(a + b\varphi) - (c + d\varphi) = (a - c) + (b - d)\varphi$$

On a également

$$(a + b\varphi)(c + d\varphi) = ac + (ad + bc)\varphi + bd\varphi^2$$

En utilisant $\varphi^2 = \varphi + 1$,

$$(a + b\varphi)(c + d\varphi) = (ac + bd) + (ad + bc + bd)\varphi \in \mathbb{Z}[\varphi]$$

□

Proposition 7. *L'application $f : \mathbb{Z}^2 \longrightarrow \mathbb{Z}[\varphi]$ définie par $f(a, b) = a + b\varphi$ est une bijection.*

Démonstration. La surjectivité de f résulte de la définition de $\mathbb{Z}[\varphi]$. Soient (a, b) et (c, d) deux éléments de $\mathbb{Z}^2$. Supposons $f(a, b) = f(c, d)$, c'est à dire $a + b\varphi = c + d\varphi$. Supposons, par l'absurde, que $b \neq d$. On a alors

$$\varphi = \frac{c - a}{b - d} \in \mathbb{Q}$$

Contradiction, donc $b = d$. En reportant, il vient $a = c$. Ainsi, f est injective. □

3.3 Une classe Python

La classe `ZPhi` implémente la multiplication et l'élévation à la puissance n.

```
class ZPhi:

    def __init__(self, a, b):
        self.re = a
        self.im = b

    def __mul__(self, y):
        a = self.re * y.re + self.im * y.im
        b = self.re * y.im + self.im * (y.re + y.im)
        return ZPhi(a, b)

    def __pow__(self, n):
        y = ZPhi(self.re, self.im)
        z = ZPhi(1, 0)
        while n != 0:
            if n & 1 != 0: z = z * y
            y = y * y
            n = n >> 1
        return z
```

La méthode `__pow__` utilise l'algorithme d'exponentiation rapide. Le calcul de x^n demande donc $\mu(n)+\gamma(n)$ multiplications d'éléments de $\mathbb{Z}[\varphi]$. L'examen de la méthode `__mul__` montre que celle-ci nécessite 3 additions et 4 multiplications, c'est à dire 7 opérations sur des entiers. Ainsi, le nombre d'opérations sur des entiers nécessaires au calcul de x^n est

$$T(x,n) = 7(\mu(n) + \gamma(n))$$

3.4 Puissances de φ

Posons $F_{-1} = 1$, de sorte que pour tout $n \geq 1$, $F_n = F_{n-1} + F_{n-2}$.

Proposition 8. *Pour tout $n \in \mathbb{N}$,*

$$\varphi^n = F_{n-1} + F_n\varphi$$

Démonstration. C'est une récurrence facile sur n. Tout d'abord,

$$\varphi^0 = 1 = 1 + 0\varphi = F_{-1} + F_0\varphi$$

Soit $n \in \mathbb{N}$. Supposons que $\varphi^n = F_{n-1} + F_n\varphi$. On a alors

$$\begin{aligned}
\varphi^{n+1} &= \varphi^n\varphi \\
&= (F_{n-1} + F_n\varphi)\varphi \\
&= F_{n-1}\varphi + F_n\varphi^2 \\
&= F_{n-1}\varphi + F_n(1+\varphi) \\
&= F_n + (F_{n-1} + F_n)\varphi \\
&= F_n + F_{n+1}\varphi
\end{aligned}$$

□

En supposant défini l'objet `phi = ZPhi(0, 1)`, on déduit immédiatement de ce résultat une fonction `fib` qui renvoie le n^{e} nombre de Fibonacci.

```
def fib(n): return (phi ** n).im
```

L'évaluation de `fib(n)` renvoie F_n en effectuant $7(\mu(n) + \gamma(n)) = O(\log n)$ opérations sur des entiers.