Algorithme du Lièvre et de la Tortue

Antoine Dequay

21 septembre 2022

Notes

— Prof : Nicolas Markey.

— Leçon : 927.

— Références :

— Donald Knuth, *The Art of Computer Programming, vol. II : Seminumerical Algorithms*, Addison-Wesley, 1969, p. 7, exercices 6 & p. 453.

— Flageolet-Sedgewick. Analytic Combinatorics (p 465)

— Olivier Bourgain.

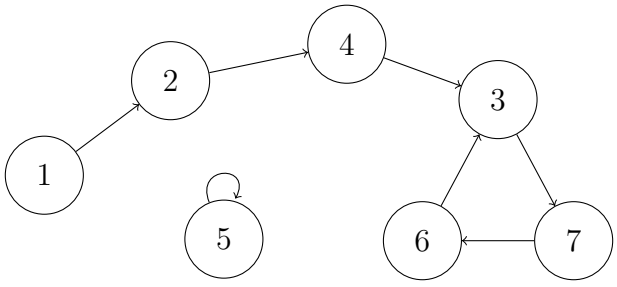
L'algorithme de Floyd, aussi appelé *Algorithme du Lièvre et de la Tortue*, a pour but de trouver la taille de la partie initiale d'une suite récurrente, ainsi que sa période.

Algorithme 1 Soit E un ensemble fini, $\#E$ son cardinal, $f : E \longrightarrow E$ une fonction et $(e_i)_{i\in\mathbb{N}}$ la suite récurrente associée à f.

On définit l'algorithme suivant :

```
entrée : f, e_0
sortie : μ la période et λ la taille de la partie initiale
détection_cycle_Floyd(f, e_0):
    m = 1
    tortue = f(e_0)
    lievre = f(f(e_0))
    while tortue != lievre :
        m = m + 1
        tortue = f(tortue)
        lievre = f(f(lievre))
    λ = 0
    tortue = e_0
    while tortue != lievre :
        λ = λ + 1
        tortue = f(tortue)
        lievre = f(lievre)
    μ = 1
    lievre = f(lievre)
    while tortue != lievre :
        μ = μ + 1
        lievre = f(lievre)
    return (μ,λ)
```

Exemple 2 $E = [\![1, 7]\!]$, f représenté comme tel :



Si $e_0 = 1$, les 3 boucles s'arrêtent sur 3. Si $e_0 = 2$, la première boucle s'arrête sur 7 et les deux autres sur 3.

Théorème 3 L'algorithme de FLOYD est correct et se termine.

Preuve.

— **Terminaison** :

Dans un premier temps, on va vérifier l'existence de i tel que $e_i = e_{2i}$, puis exprimer le plus petit entier naturel vérifiant cette relation, qu'on notera i_0.

L'ensemble E étant de cardinal fini, il est immédiat de voir que la suite admet un cycle au bout d'un certain temps. En effet, un élément de la séquence se répète forcément au plus au bout de $\#E$ itérations. La suite étant récurrente, on a alors forcément un cycle qui se forme. Ainsi, avec les notations précédentes, on a, pour $x \in \mathbb{Z}$:

$$x \in \mathbb{N} \iff e_{\lambda+x} = e_{\lambda+\mu+x}$$

On cherche donc à trouver $x \in \mathbb{N}$ tel que $\lambda + x > 0$ et que $e_{\lambda+x} = e_{2(\lambda+x)} = e_{\lambda+(\lambda+2x)}$. C'est le cas si et seulement si $x \equiv \lambda + 2x \ (mod\ \mu)$, c'est à dire $x \equiv -\lambda \ (mod\ \mu)$

En notant $\lambda = k\mu + r$ la division euclidienne de λ par μ, on a pour $l \in \mathbb{N}^*$:

$$\underbrace{\underbrace{(l\mu - r)}_{=x\geq 0} + \lambda}_{>0} = (k+l)\mu$$

On peut alors remarquer que, par définition de la division euclidienne, on a $i_0 = \lambda + \mu - r$.

Ainsi, la première boucle termine bien, au bout de $\lambda + \mu - r$ itérations.

Il reste à voir que la seconde boucle termine (il est clair que la troisième termine bien).

C'est bien le cas, car $\mu \,|\, \lambda + \mu - r$ par définition de r, donc on a bien $e_\lambda = e_{\lambda+\mu-r+\lambda}$.

— **Correction** :

Comme $e_\lambda = e_{\lambda+\mu-r+\lambda}$ et $e_{\lambda-1} \neq e_{\lambda+\mu-r+\lambda-1}$ par définition de λ, la correction du calcul de λ avérée. Le calcul de μ est clair.

□

Contrairement à un algorithme de recherche classique qui retiendrait tous les états déjà visités (donc complexité spatiale en $O(\lambda + \mu)$), l'algorithme de FLOYD a une complexité spatiale constante. La complexité temporelle pour les comparaisons passe d'un $O((\lambda + \mu)^2)$ à une complexité en $O(\lambda + \mu)$.

' Changer dans la preuve μ par P et λ par L, pour pouvoir parler d'invariants en utilisant les valeurs dans le programme ! ! ! Commencer l'exemple dès le début commencer sur $e_0 = 2$

OU :

Par invariant de boucle :

Dans la première boucle, $m \geq 1$ $l = u_{2m}$ $t = u_m$

Dans la seconde : $\lambda \geq 0$ $l = u_{m+\lambda}$ $t = u_\lambda$ $\forall 0 \leq a \leq \lambda - 1$, $u_a \neq u_{m+a}$

Dans la troisième : $\mu \geq 1$ $l = u_m$ $t = u_{m+\mu}$ $\forall 0 \leq b \leq \mu$, $u_m \neq u_{m+b}$

Lemme 4 La boucle 1 termine, et quand elle termine, m est un multiple de P

Preuve. $\exists k$ tel que $kP > L$

□

Lemme 5 La boucle 2 se termine et $\lambda = L$

Preuve. $\lambda \leq L$ OK Si $\lambda < L$, contradiction.

□

Lemme 6 La boucle 3 termine et $\mu = P$