Escapade algorithmique avec Fibonacci

Présentation ADS - TIPE

Lucas RODRIGUEZ

21 Mars 2019 - 16h30

Problématique générale

Comment calculer les nombres de Fibonacci de manière performante ?

Sommaire

Formule de récurrence

Définition

La suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par : $F_0 = 0$, $F_1 = 1$ et :

$$\forall n \geq 2,\ F_{n+2} = F_{n+1} + F_n$$

Formule de récurrence

Avec tableau

```
def fibo_table(n):
    f = [0, 1]
    for k in range(2,n+1):
        f.append(f[k-1] + f[k-2])
    return f[n]
```

Avantage

$n-1$ additions réalisées

Inconvénient

$n+2$ variables affectées

Formule de récurrence

Sans tableau

```
def fibo(n):
    if n == 0 or n == 1:
        return n
    u = 0
    v = 1
    for i in range(2, n+1):
        s = u + v
        u = v
        v = s
    return s
```

Avantages

- $n-1$ additions réalisées
- 3 variables affectées $+$ 1 compteur

Formule explicite

Démonstration

$$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$$

On reconnaît une suite récurrente d'ordre 2.

Démonstration.

Polynôme caractéristique : $P = r^2 - r - 1$ avec $r_1 = \phi$ et $r_2 = \overline{\phi}$
$\exists (\alpha, \beta) \in \mathbb{K}^2$ tq :

$$\forall n \in \mathbb{N}, F_n = \alpha\phi^n + \beta\overline{\phi}^n$$

Avec les conditions initiales $F_0 = 1$ et $F_1 = 1$, on obtient :
$\alpha = -\beta = \frac{1}{\sqrt{5}}$ □

Formule explicite

Démonstration

$$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$$

On reconnaît une suite récurrente d'ordre 2.

Démonstration.

Polynôme caractéristique : $P = r^2 - r - 1$ avec $r_1 = \phi$ et $r_2 = \overline{\phi}$
$\exists (\alpha, \beta) \in \mathbb{K}^2$ tq :

$$\forall n \in \mathbb{N}, F_n = \alpha\phi^n + \beta\overline{\phi}^n$$

Avec les conditions initiales $F_0 = 1$ et $F_1 = 1$, on obtient :
$\alpha = -\beta = \frac{1}{\sqrt{5}}$ □

Formule de Binet

$$\forall n \in \mathbb{N}, F_n = \frac{1}{\sqrt{5}}\left(\phi^n - \overline{\phi}^n\right)$$

Formule explicite

Implémentation

```
import math

def fibo_binet(n):
    phi, phib = (1 + math.sqrt(5))/2, (1 - math.sqrt(5))/2
    f = (phi**n - phib**n)//math.sqrt(5)
    return int(f)
```

Avantage

Permet de calculer F_n sans connaître les termes précédents

Inconvénient

Moins efficace que les 2 précédents algorithmes

- **Cas de base** : $F_0 = 0$ et $F_1 = 1$
- **Formule de propagation** : $F_n = F_{n-1} + F_{n-2}$

- **Cas de base** : $F_0 = 0$ et $F_1 = 1$
- **Formule de propagation** : $F_n = F_{n-1} + F_{n-2}$

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

Avantage

Meilleure écriture et lisibilité

Inconvénient

Pas performant car **beaucoup de calculs sont inutilement répétés**

Amélioration possible

Garder en mémoire les anciens termes pour éviter de les recalculer

Inconvénient persistant

Complexité spatiale très mauvaise

Comparaison des 2 méthodes

- Algorithme itératif
  - $n-1$ additions
  - $2+3(n-1)$ affectations

$\Longrightarrow$ Complexité linéaire

Comparaison des 2 méthodes

- Algorithme itératif
  - $n-1$ additions
  - $2+3(n-1)$ affectations

  $\Longrightarrow$ Complexité linéaire
- Algorithme récursif
  - s_n : nombre d'additions $s_n > F_n \sim \frac{\phi^n}{\sqrt{5}}$

  $\Longrightarrow$ Complexité exponentielle

Comparaison des 2 méthodes

- Algorithme itératif
  - $n-1$ additions
  - $2+3(n-1)$ affectations

  $\Longrightarrow$ Complexité linéaire
- Algorithme récursif
  - s_n : nombre d'additions $s_n > F_n \sim \frac{\phi^n}{\sqrt{5}}$

  $\Longrightarrow$ Complexité exponentielle

Conclusion

La récursivité est donc à éviter pour calculer les nombres de Fibonacci. (comportement chaotique)

Ecriture matricielle

Présentation

$$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$$

Ecriture matricielle

Présentation

$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$

Démonstration.

On pose : $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$.

Ecriture matricielle

Présentation

$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$

Démonstration.

On pose : $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. On obtient directement :

$$\forall n \in \mathbb{N}, X_{n+1} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = A * X_n$$

Ecriture matricielle

Présentation

$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$

Démonstration.

On pose : $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. On obtient directement :

$$\forall n \in \mathbb{N}, X_{n+1} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = A * X_n$$

D'où : $\forall n \in \mathbb{N}, \boxed{X_n = A^n * X_0}$. □

Ecriture matricielle

Présentation

$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$

Démonstration.

On pose : $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. On obtient directement :

$$\forall n \in \mathbb{N}, X_{n+1} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = A * X_n$$

D'où : $\forall n \in \mathbb{N}, \boxed{X_n = A^n * X_0}$. □

Remarque

Le nombre de Fibonacci F_n est donc dans la matrice A^n :

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

Ecriture matricielle

Présentation

$\forall n \in \mathbb{N},\ F_{n+2} = F_{n+1} + F_n$

Démonstration.

On pose : $X_n = \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. On obtient directement :

$$\forall n \in \mathbb{N}, X_{n+1} = \begin{pmatrix} F_{n+1} \\ F_{n+2} \end{pmatrix} = A * X_n$$

D'où : $\forall n \in \mathbb{N}, \boxed{X_n = A^n * X_0}$. □

Remarque

Le nombre de Fibonacci F_n est donc dans la matrice A^n :

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

Utilisons la méthode de l'exponentiation rapide.

Ecriture matricielle

Méthode d'exponentiation rapide

Permet de diminuer considérablement le nombre de multiplications réalisées

Ecriture matricielle

Méthode d'exponentiation rapide

Permet de diminuer considérablement le nombre de multiplications réalisées

- **Cas de base** : $a^0 = 1$
- **Formule de propagation** : $\begin{cases} a^n = (a^2)^{\frac{n}{2}} & \text{si } n \text{ pair} \\ a^n = a * (a^2)^{\frac{n-1}{2}} & \text{sinon} \end{cases}$

En l'implémentant par récursivité, on obtient une complexité logarithmique.
$2(\log_2(n) + 1)$ multiplications effectuées pour calculer a^n

Ecriture matricielle

Méthode d'exponentiation rapide

Permet de diminuer considérablement le nombre de multiplications réalisées

- **Cas de base** : $a^0 = 1$
- **Formule de propagation** : $\begin{cases} a^n = (a^2)^{\frac{n}{2}} & \text{si } n \text{ pair} \\ a^n = a * (a^2)^{\frac{n-1}{2}} & \text{sinon} \end{cases}$

En l'implémentant par récursivité, on obtient une complexité logarithmique.
$2(\log_2(n) + 1)$ multiplications effectuées pour calculer a^n

Implémentons-le maintenant en tenant compte du produit matriciel !

Ecriture matricielle

Implémentation

Soit `proMat(A,B)`, une fonction renvoyant le produit matriciel de A et B.

```
A = [[0, 1], [1, 1]]
def fibo_exporap(A, n):
    if n == 0:
        return [[1, 0], [0, 1]] # retourner I_2
    elif n % 2 == 0:
        return fibo_exporap(proMat(A, A), n/2)
    else:
        return proMat(A, fibo_exporap(proMat(A, A), (n-1)/2))
```

Ecriture matricielle

Implémentation

Soit `proMat(A,B)`, une fonction renvoyant le produit matriciel de A et B.

```
A = [[0, 1], [1, 1]]
def fibo_exporap(A, n):
    if n == 0:
        return [[1, 0], [0, 1]] # retourner I_2
    elif n % 2 == 0:
        return fibo_exporap(proMat(A, A), n/2)
    else:
        return proMat(A, fibo_exporap(proMat(A, A), (n-1)/2))
```

On exécute :

```
>>> fibo_exporap(A, 50)[0][1]
12586269025
```

Ecriture matricielle

Etude de complexité

- Algorithme récursif utilisant l'exponentiation rapide
  - $2(\log_2(n)+1)$ multiplications matricielles
  - $16(\log_2(n)+1)$ multiplications d'entiers
  - $8(\log_2(n)+1)$ additions d'entiers

$\Longrightarrow$ Complexité logarithmique

Réponse au problème

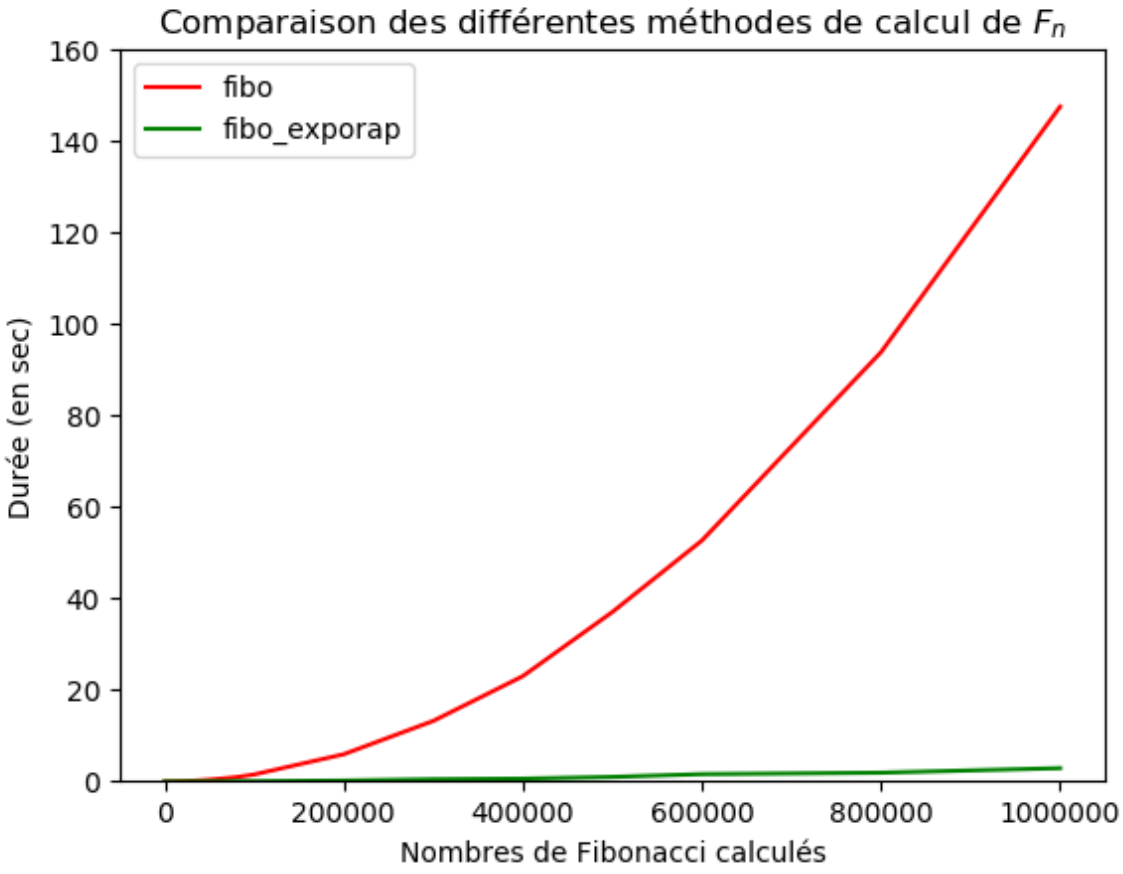
Etude comparative

3 programmes pour calculer les nombres de la suite de Fibonacci :

- `fibo`
- `fiborec` (on l'abandonne car ne satisfait pas les contraintes imposées)
- `fibo_exporap`

Réponse au problème

Etude comparative



Réponse au problème

Etude comparative

3 programmes pour calculer les nombres de la suite de Fibonacci :

- `fibo`
- `fiborec` (on l'abandonne car ne satisfait pas les contraintes imposées)
- `fibo_exporap`

Conclusion

La fonction `fibo_exporap` (utilisant l'écriture matricielle) est la plus performante dans le calcul des nombres de Fibonacci.

Réponse au problème

Etude comparative

3 programmes pour calculer les nombres de la suite de Fibonacci :

- `fibo`
- `fiborec` (on l'abandonne car ne satisfait pas les contraintes imposées)
- `fibo_exporap`

Conclusion

La fonction `fibo_exporap` (utilisant l'écriture matricielle) est la plus performante dans le calcul des nombres de Fibonacci.

Intéressons-nous maintenant à une application de la récursivité et des nombres de Fibonacci...

Application de Fibonacci : l'algorithme d'Euclide

Définition & Récursivité

L'algorithme d'Euclide permet de calculer efficacement le PGCD de 2 nombres entiers a et b.

```
def euclide(a, b):
    u, v = a, b
    while v != 0:
        r = u % v
        u = v
        v = r
    return u
```

```
def euclideрec(a, b):
    if b = 0:
        return a
    else:
        return euclideрec(b, a%b)
```

- **Cas de base** : $b = 0 \Rightarrow a \wedge b = a$
- **Formule de propagation** : $a \wedge b = b \wedge r$

Application de Fibonacci : l'algorithme d'Euclide

Théorème & Etude de complexité

Le théorème de Lamé nous dit que le **nombre de divisions euclidiennes est inférieur ou égal à 5 fois le nombre de chiffre du plus petit nombre entre** a **et** b.

Application de Fibonacci : l'algorithme d'Euclide

Théorème & Etude de complexité

Le théorème de Lamé nous dit que le **nombre de divisions euclidiennes est inférieur ou égal à 5 fois le nombre de chiffre du plus petit nombre entre** a **et** b.
De plus, pour calculer $F_{n+2} \wedge F_{n+1}$, `euclide(a,b)` va effectuer exactement n divisions et $F_{n+2} \wedge F_{n+1} = 1$.

Application de Fibonacci : l'algorithme d'Euclide

Théorème & Etude de complexité

Le théorème de Lamé nous dit que le **nombre de divisions euclidiennes est inférieur ou égal à 5 fois le nombre de chiffre du plus petit nombre entre** a **et** b.
De plus, pour calculer $F_{n+2} \wedge F_{n+1}$, `euclide(a,b)` va effectuer exactement n divisions et $F_{n+2} \wedge F_{n+1} = 1$.

Conclusion

Les nombres de Fibonacci servent de véritables indicateurs dans l'analyse de la complexité de l'algorithme d'Euclide.
Plus précisément, ils permettent d'obtenir une majoration du nombre de divisions euclidiennes et donc de la complexité de ce dernier.

Fin de l'exposé