Diviser pour régner

Fibonacci et multiplication de Karatsuba

Exercice 1 *La suite de Fibonacci*

La suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$ pour $n \geq 2$. Cette définition suggère une résolution « diviser pour régner » : pour calculer F_n, on résout les deux sous-problèmes F_{n-1} et F_{n-2}, puis on combine les résultats par une addition.

```
from time import perf_counter


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


debut = perf_counter()
for i in range(38):
    print(f"{i} {fibo(i)}")
fin = perf_counter()
print(f"Durée : {fin - debut} s")
```

1. Exécuter ce programme. Que remarque-t-on ?
2. Dessiner l'arbre des appels provoqués par `fibo(5)`. Combien d'appels à `fibo` compte-t-on ? Combien de fois `fibo(2)` est-il calculé ?
3. En déduire pourquoi l'approche « diviser pour régner » est mal adaptée à ce problème.
4. Écrire une version itérative `fibo_iteratif(n)` qui ne conserve que les deux derniers termes, puis la chronométrer sur les 50 premiers termes.
5. Écrire une version récursive rapide `fibo_rapide(n, a=0, b=1)` : `a` et `b` représentent deux termes consécutifs de la suite, chaque appel avance d'un rang et il n'y a qu'un seul appel récursif. La chronométrer de la même façon.

Exercice 2 *Multiplication de grands nombres (Karatsuba)*

Pour multiplier deux entiers de n chiffres, la méthode apprise à l'école multiplie chaque chiffre du premier nombre par chaque chiffre du second, puis additionne les résultats décalés : n^2 produits de chiffres, soit un coût en $O(n^2)$.

Coupons chaque nombre en deux moitiés de k chiffres, $x = a \times 10^k + b$ et $y = c \times 10^k + d$. Alors

$$x \times y = ac \times 10^{2k} + (ad + bc) \times 10^k + bd,$$

ce qui demande quatre produits de nombres deux fois plus courts : ac, ad, bc et bd. Diviser ainsi ne fait rien gagner : quatre sous-problèmes de taille moitié coûtent toujours $O(n^2)$.

L'idée de Karatsuba (1960) est de n'utiliser que **trois** produits, en remarquant que

$$ad + bc = ac + bd - (a - b)(c - d).$$

Les additions, les soustractions et les multiplications par une puissance de 10 (de simples décalages) sont peu coûteuses par rapport aux multiplications.

Exemple. Calcul de 1237×2587 avec $a = 12$, $b = 37$, $c = 25$, $d = 87$ et $k = 2$.

— On pose $a_0 = ac = 12 \times 25$, $a_2 = bd = 37 \times 87$ et $a_1 = (a - b)(c - d) = (12 - 37) \times (25 - 87) = 25 \times 62$; alors $1237 \times 2587 = a_0 \times 10^4 + (a_0 + a_2 - a_1) \times 10^2 + a_2$.

— Chaque produit est calculé de la même façon. Pour 12×25 : $b_0 = 1 \times 2 = 2$, $b_2 = 2 \times 5 = 10$, $b_1 = (1 - 2) \times (2 - 5) = 3$, d'où $12 \times 25 = 2 \times 10^2 + (2 + 10 - 3) \times 10 + 10 = 300$. Les produits de chiffres ne sont pas redécomposés.

— De même $a_1 = 25 \times 62 = 1550$ et $a_2 = 37 \times 87 = 3219$.

— Finalement $1237 \times 2587 = 300 \times 10^4 + (300 + 3219 - 1550) \times 10^2 + 3219 = 3\,000\,000 + 196\,900 + 3\,219 = 3\,200\,119$.

6. Écrire la fonction `karatsuba(x, y)` qui multiplie deux entiers selon ce principe. On coupera les nombres avec `divmod(x, 10 ** k)`, où `k` est la moitié du nombre de chiffres du plus long des deux ; les produits où l'un des facteurs n'a qu'un chiffre sont calculés directement. Vérifier la fonction avec `doctest`.

```
import doctest


def karatsuba(x, y):
    """
    >>> for i in range(100):
    ...     for j in range(100):
    ...         assert karatsuba(i, j) == i * j
    >>> a = 1298065705673133242342 34
    >>> karatsuba(a, a)
    1684974576244689379013199208845711432249756 6756
    """
    ...


doctest.testmod()
```

7. Un produit de n chiffres demande trois produits de $n/2$ chiffres et des additions en $O(n)$. Combien de produits de chiffres effectue-t-on au total pour $n = 2^p$? En déduire que la complexité est en $O(n^{\log_2 3}) \approx O(n^{1,58})$, meilleure que $O(n^2)$. On admet que $3^{\log_2 n} = n^{\log_2 3}$.