EXPONENTIATION MODULAIRE RAPIDE : ALGORITHMES

Étant donnés une base *a*, un exposant *e* et un entier *n,* on souhaite calculer *c* tel que :

$$c \equiv a^e \ [n].$$

L'exponentiation modulaire rapide permet de le faire.

I. Algorithme classique

```
def expmodrap(a, e, n) :
    p=1
    while e>0:
        if e % 2 == 1:
            p = (p*a)%n
        a=(a*a)%n
        e=e//2
    return p
```

II. Algorithme astucieux (plus rapide)

```
def lpowmod(x, y, n) :
    result = 1
    while y>0:
        if y&1>0:
            result = (result*x)%n
        y >>= 1
        x = (x*x)%n
    return result
```

Explications :

>>=1 est l'opérateur de décalage à droite
On pousse tous les bits d'un cran (ou de plusieurs si on remplace 1 par n) vers la droite (on perd donc des données de ce côté), on complète par des zéros sur la gauche.
On obtient le quotient par une puissance de 2 sans faire appel à une routine coûteuse de division.

Un & logique avec 1 donne le bit de poids faible qui est le reste de la division par 2.

Ces opérations (>> et &) correspondent à des instructions du processeur, donc de la logique câblée extrêmement rapide.

III. Fonction *pow* de Python

Python intègre une fonction qui permet de faire une exponentiation modulaire rapide : pow(a,e,n).

IV. Rapidité de chaque algorithme

Pour calculer 4^{13} modulo 497, sur mon PC, j'obtiens les temps suivants[1] :

Avec expmodrap : $2{,}4550\times10^{-6}$
Avec lpowmod : $2{,}2413\times10^{-6}$
Avec pow : $1{,}0076\times10^{-6}$

Pour calculer $65127^{12111985}$ modulo 81629, j'obtiens les temps suivants[2] :

Avec expmodrap : $1{,}5196\times10^{-5}$
Avec lpowmod : $1{,}4385\times10^{-5}$
Avec pow : $0{,}6733\times10^{-5}$

Il semble donc que la fonction pow implémentée dans Python soit deux fois plus rapide que nos algorithmes !

Algorithmes utilisés pour le calcul de rapidité de chaque méthode :

```
import time
sum_a, sum_b, sum_c = 0,0,0
nbfois = 500000
for i in range(0,nbfois):
    a = time.clock()
    expmodrap(4,13,497)
    b = time.clock()
    sum_a = sum_a + b - a
    b = time.clock()
    lpowmod(4,13,497)
    c = time.clock()
    sum_b = sum_b + c - b
    c = time.clock()
    pow(4,13,497)
    d = time.clock()
    sum_c = sum_c + d - c
print("Avec expmodrap : ",sum_a/nbfois)
print("Avec lpowmod : ",sum_b/nbfois)
print("Avec pow : ",sum_c/nbfois)
```

1 Moyenne des temps sur 500 000 essais consécutifs. Temps exprimé en secondes.
2 Idem