EXPONENTIATION MODULAIRE

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

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

Nous prendrons pour exemple $b=4$, $e=13$ et $n=497$.

I. Méthode directe

Si on peut calculer b^e , alors il suffit de trouver le reste de b^e par n.

A. Exemple

$4^{13} = \ldots$

donc $4^{13} \equiv \ldots [497]$

B. Cas général : quelles limites de calcul ?

Les informations sont traitées dans des circuits ne pouvant avoir que deux états (« le courant passe » et « le courant ne passe pas ») : les données sont donc stockées sous formes de séquences de ***bits*** (0 et 1).
Dans la mémoire des ordinateurs, ces circuits sont souvent groupés par huit : les octets.
On utilise souvent des nombres exprimés en notation binaire sur un, deux, quatre ou huit octets, soit 8, 16, 32 ou 64 bits. Sur un octet, on peut représenter les nombres de $\overline{0000\,000}^2$ à $\overline{1111\,1111}^2$, *ie* de 0 à 255. Sur deux octets, de 0 à $\overline{1111\,1111\,1111\,1111}^2 = 65\,535$. Sur quatre octets, de 0 à 4 294 967 295.

En Python, les nombres entiers (relatifs) sont représentés par des objets qui appartiennent au type *int* (pour *integer*). Les valeurs possibles pour un objet de ce type ne sont limitées que par les capacités de l'ordinateur. Les calculs réalisés sont réalisés de manière exacte ou n'aboutissent pas.
Si le processeur d'un ordinateur peut exécuter du 64 bits[1], alors la valeur maximale représentable est :

$$\overline{1111111111111111111111111111111111111111111111111111111111111111}^2$$

c'est-à-dire $2^0+2^1+2^2+\ldots+2^{63}$ *ie* $2^{64}-1$ *ie* 18 446 744 073 709 551 615.
La plupart des ordinateurs personnels intègrent un processeur en 32 bits seulement, ce qui donne une valeur maximale représentable de $2^{32}-1=4\,294\,967\,295$.

Puisqu'une opération mathématique peut produire un résultat plus grand que la valeur maximale représentable, une impossibilité d'enregistrer le résultat de l'opération peut survenir : c'est ce qu'on appelle un ***dépassement d'entier***. Cette condition d'erreur résulte en un message d'erreur ou en la troncature du résultat qui est alors erroné.
Le dépassement d'entier le plus célèbre de ces vingt dernières années est très probablement celui qui causa la ***destruction de la fusée Ariane 5, lors de son vol inaugural, le 4 juin 1996***.
La fusée a explosé à une altitude de 4 000 mètres au-dessus du centre spatial de Kourou, en Guyane française. Il n'y a eu aucune victime, les débris étant retombés relativement près du pas de tir et le vol étant inhabité.
L'incident, dû à un dépassement d'entier dans les registres mémoire des calculateurs électroniques utilisés par le pilote automatique, a provoqué la panne du système de navigation de la fusée, causant de fait sa destruction ainsi que celle de la charge utile. Cette charge utile était constituée des quatre satellites de la mission Cluster, d'une valeur totale de 370 millions de dollars.

1 Et si le système d'exploitation, les pilotes et le logiciel peuvent exécuter du 64 bits…

II. Méthode « par produits modulaires itérés »

Pour calculer b^e modulo n, on peut procéder par e étapes en utilisant le produit des congruences…

A. Exemple

Calculons 4^{13} [497] :

- $4^1 \equiv \ldots$ [497]
- .
- .
- .
- .
- .
- .
- .
- .
- .
- .
- .

B. Analyse de la méthode

1. Combien de calculs sont nécessaires pour effectuer cette méthode dans le cas général (modulo *n*) ?

2. a) Quel est l'avantage principal de cette méthode ?

b) Quel est l'inconvénient principal de cette méthode ?

III. Méthode d'exponentiation modulaire rapide (« square-and-multiply »)

Une troisième méthode réduit drastiquement à la fois le nombre d'opérations et la place en mémoire nécessaires à l'exécution de l'exponentiation modulaire. C'est une combinaison de la méthode précédente et d'un principe plus général appelé exponentiation rapide.

A. Du décimal au binaire

Un nombre décimal $\overline{a_n a_{n-1} \dots a_2 a_1 a_0}^{10}$ s'écrit sous la forme $\sum_{k=0}^{n} a_k 10^k$.

Un nombre binaire $\overline{a_n a_{n-1} \dots a_2 a_1 a_0}^{2}$ s'écrit sous la forme $\sum_{k=0}^{n} a_k 2^k$.

Pour convertir un nombre décimal en binaire, il faut donc le décomposer en puissances de 2.
Par exemple, pour convertir le nombre décimal 324 :

$$\begin{aligned}
324 &= 2 \times 162 + \mathbf{0} \\
162 &= 2 \times 81 + \mathbf{0} \\
81 &= 2 \times 40 + \mathbf{1} \\
40 &= 2 \times 20 + \mathbf{0} \\
20 &= 2 \times 10 + \mathbf{0} \\
10 &= 2 \times 5 + \mathbf{0} \\
5 &= 2 \times 2 + \mathbf{1} \\
2 &= 2 \times 1 + \mathbf{0} \\
1 &= 2 \times 0 + \mathbf{1}
\end{aligned}$$

Donc 324 est 101000100 en binaire. Vérification : $2^2+2^6+2^8=324$.

1. Convertir le nombre décimal 5357 en binaire.

2. Voici une fonction Python qui construit une chaîne binaire à partir d'un nombre :

```
def bin(n) :
   q = -1
   res = ''
   while q != 0 :
      q = n // 2
      r = n % 2
      res = str(r) + res
      n = q
   return res
```

Expliquer cet algorithme.

B. Exemple d'exponentiation rapide

Un exemple : imaginons que l'on veuille calculer a^{17}, a fixé. Nous pouvons :

– soit calculer les a^k par une boucle for en s'arrêtant à 17 en initialisant à 1 et en multipliant à chaque étape par a, ce qui fait 17 multiplications

– soit calculer $a^2=a\times a$, $a^4=a^2\times a^2$ puis $a^8=a^4\times a^4$ puis $a^{16}=a^8\times a^8$ et $a^{17}=a^{16}\times a$.
Ce qui fait 5 multiplications au total et qui est beaucoup plus rapide. C'est « ***l'exponentiation rapide*** ».

C. Exemple d'exponentiation modulaire rapide

13 est 1101 en binaire, donc $4^{13}=4^{1\times2^0+0\times2^1+1\times2^2+1\times2^3}=(4^1)^{\mathbf{1}}\times(4^2)^{\mathbf{0}}\times(4^4)^{\mathbf{1}}\times(4^8)^{\mathbf{1}}$.
Or, la suite 4^1, 4^2, 4^4 et 4^8 s'obtient par une suite d'élévations au carré.
Il suffit donc de calculer ces nombres modulo $n = 497$ au fur et à mesure (de façon itérative) et de ne garder que les puissances de 4 associées à un bit égal à 1.

La fonction Python suivante permet donc de calculer a^e modulo n :

```
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
```

Expliquer cet algorithme.

Remarque : Python intègre déjà une fonction `pow(a, e, n)` qui permet de faire la même chose.