L3 - Mathématiques pour l'informatique 4

Arithmétique modulaire

Arithmétique élémentaire

L'arithmétique est l'étude des nombres entiers

$$\mathbb{N} = \{0, 1, 2, 3, 4, \ldots\}$$
$$\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$$

Division euclidienne

La propriété fondamentale est la *division euclidienne* : Étant donnés deux entiers $a \in \mathbb{Z}$ et $b > 0$, il existe un unique couple d'entiers (q, r) (quotient et reste) tel que

$$a = bq + r \quad \text{avec } 0 \leq r < b.$$

Lorsque le reste r est nul, on dit que b divise a ou que a est multiple de b, et on écrit $b|a$.

Un entier > 1 qui n'a pas de diviseurs non-triviaux, c'est à dire autres que 1 et lui-même est dit *premier*. Par exemple, $2, 3, 5, 7, 11, 13, 17, 19, 23$ sont premiers.

Le p.g.c.d. de deux entiers a et b est le plus grand entier positif qui divise à la fois a et b. On le note $d = a \wedge b$. Par exemple $(-12) \wedge 18 = 6$.

Le plus petit entier positif divisible à la fois par a et b est appelé p.p.c.m. de a et b. On le note $m = a \vee b$. On a

$$a \vee b = \frac{ab}{a \wedge b}$$

Algorithme d'Euclide

Pour calculer $a \wedge b$, (avec $a \geq b$) on écrit la division euclidienne $a = bq + r$. Alors $r = a - bq$ est divisible par tout entier divisant à la fois a et b donc par le p.g.c.d. de a et b, et tout entier divisant r et b divise aussi a. Donc,

$$a \wedge b = b \wedge r$$

Comme $r < b$, $(b, r) < (a, b)$ et on itère le procédé jusqu'à arriver à un reste nul. Alors, $d \wedge 0 = d$ et on afini.

In [1]:
```
# division euclidienne : q=a//b, r=a%b
print("Pour a=27, b=5, q=%d, r=%d" % (27//5, 27%5))
print("Pour a=-27, b=5, q=%d, r=%d" % (-27//5, -27%5))
```

```
Pour a=27, b=5, q=5, r=2
Pour a=-27, b=5, q=-6, r=3
```

In [2]:
```
# pgcd récursif
def gcd(a,b):
    if b==0: return a
    return gcd(b,a%b)

gcd(192,36)
```

Out[2]: 12

In [3]:
```
gcd(36,192)
```

Out[3]: 12

In [4]:
```
gcd(-36,192)
```

Out[4]: 12

In [5]:
```
gcd(192,-36)
```

Out[5]: -12

On voit qu'il faut faire attention aux signes. On évitera aussi la récursivité, car on va travailler sur des entiers très longs.

In [6]:
```
def gcd(a,b):
    a,b = abs(a), abs(b) # valeur absolue
    r = a%b
    while r:
        a,b=b,r
        r = a%b
    return b
```

Algorithme d'Euclide étendu et théorème de Bézout

Dans la version itérative, on calcule donc une suite de quotients et de restes qu'il est commode de numéroter ainsi. On pose $a_{-1}=a$, $a_0=b$, $q_0=q$, $a_1=r$, et les itérations calculent

$$\begin{aligned} a_{-1} &= a_0q_0 + a_1 \\ a_0 &= a_1q_1 + a_2 \\ \cdots &= \cdots \\ a_{k-1} &= a_kq_k + a_{k+1} \end{aligned}$$

et si le premier reste nul est a_{k+1}, alors le p.g.c.d. est $d=a_k$.

Ceci implique qu'à chaque étape, on peut calculer des entiers u_i, v_i tels que $a_i = u_ia + v_ib$ En effet, au départ, $a_1 = 1a + 0b$, $a_0 = 0a + 1b$, et par récurrence, comme $a_{i-2} = a_{i-1}q_{i-1} + a_i$, on peut écrire

$$a_i = (u_{i-2}a + v_{i-2}b) - q_{i-1}(u_{i-1}a + v_{i-1}b) = (u_{i-2} - q_{i-1}u_{i-1})a + (v_{i-2} - q_{i-1}v_{i-1})b$$

de sorte que

$$\begin{aligned} u_i &= u_{i-2} - q_{i-1}u_{i-1} \\ v_i &= v_{i-2} - q_{i-1}v_{i-1} \end{aligned}$$

Les deux suites u_i et v_i vérifient la même récurrence d'ordre 2, avec des conditions initiales différentes

$$\begin{pmatrix} u_{-1} \\ v_{-1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$\begin{pmatrix} u_0 \\ v_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Si le p.g.c.d. est a_k, on aura donc $d = u_ka + v_kb$.

Cette propriété, connue sous le nom de *théorème de Bézout*, sera très importante dans la suite :

Si $a \wedge b = d$, il existe deux entiers u et v tels que $d = ua + vb$, et l'agorithme ci-dessus, appelé algorithme d'Euclide étendu, permet de les calculer.

Voir ici (https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide_%C3%A9tendu) pour des explications complémentaires.

On programmera cet algorithme en TD. Voilà à quoi devrait ressembler le résultat (le fichier ent3.py sera fourni au TD suivant).

```
In [7]: from ent3 import *

In [8]: xgcd(36,196)
Out[8]: (4, 11, -2)

In [9]: 11*36-2*196
Out[9]: 4

In [10]: xgcd(123456789101112131415161718192021222324 25,3232323232323232323232)
Out[10]: (1, -1328660027339171737607, 5074743117952096996736521329039888644043)
```

Congruences

On dit que a est congru à b modulo n si a et b ont même reste dans la division par n, ou encore si $a-b$ est divisible par n.

On écrit $a \equiv b \mod n$ ou $a \equiv b[n]$.

En python, le test est `a%n == b%n` .

La congruence modulo n est une relation d'équivalence. On peut manipuler les congruences comme des égalités (d'où la notation), car $a \equiv b[n]$ et $a' \equiv b'[n]$ entraîne $a + a' \equiv b + b'[n]$, $aa' \equiv bb'[n]$, etc.

D'où l'idée d'introduire des symboles $\bar{a}$ ($a \in \mathbb{Z}$) qui vérifient

$$\bar{a} = \bar{b} \Leftrightarrow a \equiv b[n]$$

L'addition et la multiplication modulo n possèdent clairement les mêmes propriétés d'associativité et de ditributivité que leurs versions usuelles. On dit que l'ensemble

$$\mathbb{Z}/n\mathbb{Z} = \{\bar{0}, \bar{1}, \ldots, \overline{n-1}\},$$

muni de ces opérations, est un *anneau commutatif unitaire*.

Voici la table d'addition modulo 12, familière grâce aux horloges à cadran.

In [11]:
```
def tabadd(n):
    print(' + |', end=' ')
    for y in range(n):
        print ("%2d "% (y % n),end=' ')
    print()
    print('-'*(4*n+3))
    for x in range(n):
        print('%2d |'%x, end= ' ')
        for y in range(n):
            print ("%2d "% ((x+y) % n),end=' ')
        print()

tabadd(12)
```

```
 + |  0   1   2   3   4   5   6   7   8   9  10  11
-------------------------------------------------
 0 |  0   1   2   3   4   5   6   7   8   9  10  11
 1 |  1   2   3   4   5   6   7   8   9  10  11   0
 2 |  2   3   4   5   6   7   8   9  10  11   0   1
 3 |  3   4   5   6   7   8   9  10  11   0   1   2
 4 |  4   5   6   7   8   9  10  11   0   1   2   3
 5 |  5   6   7   8   9  10  11   0   1   2   3   4
 6 |  6   7   8   9  10  11   0   1   2   3   4   5
 7 |  7   8   9  10  11   0   1   2   3   4   5   6
 8 |  8   9  10  11   0   1   2   3   4   5   6   7
 9 |  9  10  11   0   1   2   3   4   5   6   7   8
10 | 10  11   0   1   2   3   4   5   6   7   8   9
11 | 11   0   1   2   3   4   5   6   7   8   9  10
```

La table de multiplication est peut-être moins familière. Elle possède quelques propriétés étranges.

In [12]:
```
def tabmul(n):
    print(' * |', end=' ')
    for y in range(1,n):
        print ("%2d "% (y % n),end=' ')
    print()
    print('-'*(4*n-1))
    for x in range(1,n):
        print('%2d |'%x, end= ' ')
        for y in range(1,n):
            print ("%2d "% ((x*y) % n),end=' ')
        print()

tabmul(12)
```

```
 * |  1   2   3   4   5   6   7   8   9  10  11
-----------------------------------------------
 1 |  1   2   3   4   5   6   7   8   9  10  11
 2 |  2   4   6   8  10   0   2   4   6   8  10
 3 |  3   6   9   0   3   6   9   0   3   6   9
 4 |  4   8   0   4   8   0   4   8   0   4   8
 5 |  5  10   3   8   1   6  11   4   9   2   7
 6 |  6   0   6   0   6   0   6   0   6   0   6
 7 |  7   2   9   4  11   6   1   8   3  10   5
 8 |  8   4   0   8   4   0   8   4   0   8   4
 9 |  9   6   3   0   9   6   3   0   9   6   3
10 | 10   8   6   4   2   0  10   8   6   4   2
11 | 11  10   9   8   7   6   5   4   3   2   1
```

On constate qu'à la différence de la table d'addition, toutes les lignes ou colonnes ne sont pas des permutations les unes des autres. Dans la table d'addition, chaque ligne contient l'élément neutre 0, $(\mathbb{Z}/n\mathbb{Z},+)$ est un groupe.

Par contre, modulo 12, certains éléments n'ont pas d'inverse, et sont des *diviseurs de zéro*. Par exemple, 2 n'a pas d'inverse, et $\bar{2}\times\bar{6}=\bar{0}$.

En fait, un diviseur de 0 ne peut jamais être inversible : si on suppose $ab=0$ et $a'a=1$, alors $a'ab=(a'a)b=(1)b=b$ par associativité, et d'un autre côté, $a'ab=a'(ab)=a'0=0$. Donc $b=0$, et a n'est pas un diviseur de 0.

Un *corps* est un anneau dans lequel tout élément non nul est inversible. Par exemple, $\mathbb{R}$ est un corps, mais $\mathbb{Z}$ n'en est pas un, car par exemple, 2 n'est pas inversible ($\frac{1}{2}$ n'est pas un entier).

Un anneau unitaire *fini* sans diviseurs de 0 est toujours un corps : si $a_1=1,\ldots,a_n$ sont ses éléments non nuls, multiplions les tous par l'un d'entre eux a_i. Alors, les n éléments $a_1a_i,\ldots,a_na_i$ sont tous différents, car si on avait $a_ja_i=a_ka_i$, on aurait aussi $(a_j-a_k)a_i=0$ et donc des divseurs de zéro. Donc, l'un de ces éléments est égal à 1, et a_i possède donc un inverse.

Dans $\mathbb{Z}n\mathbb{Z}$, un élément $\bar{a}$ est inversible si et seulement si $a\wedge n=1$. En effet, si a et n ont un diviseur commun non trivial d, $a=da'$, $n=dn'$, alors $\bar{a}\bar{n}'=\bar{0}$, bien que $\bar{a}$ et $\bar{n}'$ soient non nuls.

Inversement, si a est premier avec n, $d=a\wedge n=1$, et le théorème de Bézout nous fournit u et v tels que $1=au+vn$. Donc, dans $\mathbb{Z}n\mathbb{Z}$, $\bar{1}=\bar{a}\bar{u}$, et $\bar{a}$ est inversible.

Le théorème de Bézout fournit donc aussi un algorithme pour calculer les inverses.

In [13]:
```
def inversemod(x,n):
    d,u,v = xgcd(x,n)
    if d!=1: raise Exception('Element non inversible')
    return u if u>0 else n+u
```

In [14]:
```
inversemod(5,12)
```

Out[14]:
```
5
```

In [15]:
```
inversemod(6,12)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-15-ec6c8b5f9548> in <module>()
----> 1 inversemod(6,12)

<ipython-input-13-bc211e74865f> in inversemod(x, n)
      1 def inversemod(x,n):
      2     d,u,v = xgcd(x,n)
----> 3     if d!=1: raise Exception('Element non inversible')
      4     return u if u>0 else n+u

Exception: Element non inversible
```

In [16]:
```
# On va devoir calculer modulo de grands nombres premiers
p = 2**607-1
p
```

Out[16]:
```
531137992816767098689588206552468627329593117727031923199444138200403559860852242739162502265229285668889329486246501015346579337652707239409519978766587351943831270835393219031728127
```

In [17]:
```
inversemod(123456,p)
```

Out[17]:
```
513244954713111266948320083892964698633832864870737164187435907164443554632582377543852619821920830842093461149889424828504089515648935779061669784432652696708489171177179763242192120
```

In [18]:
```
(_*123456)%p
```

Out[18]:
```
1
```

On vient donc de démontrer :

"Lorsque p est un nombre premier, $\mathbb{Z}/p\mathbb{Z}$ est un corps."

Cette propriété est fondamentale. Tout ce qu'on apprend en premier cycle sur les polynômes ou les matrices reste vrai pourvu que les coefficients soient pris dans un *corps*. L'existence de corps finis tels que les $\mathbb{Z}/p\mathbb{Z}$ permet d'utiliser ces notions en cryptographie ou pour la construction de codes correcteurs d'erreurs.

Le (petit) théorème de Fermat

Il est donc important d'étudier les nombres premiers, et en particulier d'être capable de les reconnaître. L'observation suivante sera fondamentale pour la suite.

Rappelons la formule du binôme de Newton

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k, \quad \binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{1\cdot 2\cdots k}$$

Regardons le triangle de Pascal modulo divers entiers.

In [19]:
```
def binomial(n,k):
    if n<k: return 0
    if k==0: return 1
    return binomial(n-1,k-1)+binomial(n-1,k)


def triangle(n,r):
    for m in range(n+1):
        print ("%2d" % m, end = ' ')
        for k in range(m+1):
            print ("%3d" % (binomial(m,k) % r),end = ' ')
        print()

# Modulo 2
triangle(15,2)
```

```
 0   1
 1   1   1
 2   1   0   1
 3   1   1   1   1
 4   1   0   0   0   1
 5   1   1   0   0   1   1
 6   1   0   1   0   1   0   1
 7   1   1   1   1   1   1   1   1
 8   1   0   0   0   0   0   0   0   1
 9   1   1   0   0   0   0   0   0   1   1
10   1   0   1   0   0   0   0   0   1   0   1
11   1   1   1   1   0   0   0   0   1   1   1   1
12   1   0   0   0   1   0   0   0   1   0   0   0   1
13   1   1   0   0   1   1   0   0   1   1   0   0   1   1
14   1   0   1   0   1   0   1   0   1   0   1   0   1   0   1
15   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

In [20]:
```
# Modulo 3
triangle(15,3)
```

```
 0   1
 1   1   1
 2   1   2   1
 3   1   0   0   1
 4   1   1   0   1   1
 5   1   2   1   1   2   1
 6   1   0   0   2   0   0   1
 7   1   1   0   2   2   0   1   1
 8   1   2   1   2   1   2   1   2   1
 9   1   0   0   0   0   0   0   0   0   1
10   1   1   0   0   0   0   0   0   0   1   1
11   1   2   1   0   0   0   0   0   0   1   2   1
12   1   0   0   1   0   0   0   0   0   1   0   0   1
13   1   1   0   1   1   0   0   0   0   1   1   0   1   1
14   1   2   1   1   2   1   0   0   0   1   2   1   1   2   1
15   1   0   0   2   0   0   1   0   0   1   0   0   2   0   0   1
```

In [21]:
```
# Modulo 4
triangle(15,4)
```

```
 0   1
 1   1   1
 2   1   2   1
 3   1   3   3   1
 4   1   0   2   0   1
 5   1   1   2   2   1   1
 6   1   2   3   0   3   2   1
 7   1   3   1   3   3   1   3   1
 8   1   0   0   0   2   0   0   0   1
 9   1   1   0   0   2   2   0   0   1   1
10   1   2   1   0   2   0   2   0   1   2   1
11   1   3   3   1   2   2   2   2   1   3   3   1
12   1   0   2   0   3   0   0   0   3   0   2   0   1
13   1   1   2   2   3   3   0   0   3   3   2   2   1   1
14   1   2   3   0   1   2   3   0   3   2   1   0   3   2   1
15   1   3   1   3   1   3   1   3   3   1   3   1   3   1   3   1
```

In [22]:
```
# Modulo 5
triangle(15,5)
```

```
 0   1
 1   1   1
 2   1   2   1
 3   1   3   3   1
 4   1   4   1   4   1
 5   1   0   0   0   0   1
 6   1   1   0   0   0   1   1
 7   1   2   1   0   0   1   2   1
 8   1   3   3   1   0   1   3   3   1
 9   1   4   1   4   1   1   4   1   4   1
10   1   0   0   0   0   2   0   0   0   0   1
11   1   1   0   0   0   2   2   0   0   0   1   1
12   1   2   1   0   0   2   4   2   0   0   1   2   1
13   1   3   3   1   0   2   1   1   2   0   1   3   3   1
14   1   4   1   4   1   2   3   2   3   2   1   4   1   4   1
15   1   0   0   0   0   3   0   0   0   0   3   0   0   0   0   1
```

On observe que lorsque n est premier, apparemment, tous les $\binom{n}{k}$ sont divisbles par n, sauf le premier et le dernier qui valent toujours 1. De fait, la formule avec les factorielles montre que ce nombre, qui est un entier, est le résultat de la simplification d'une fraction dont le numérateur est divisible par n, mais pas le dénominateur si on suppose que n est premier. Donc le résultat est divisible par n.

Pour p premier, la formule du binôme dans $\mathbb{Z}/p\mathbb{Z}$ devient ainsi

$$(\bar{x}+\bar{y})^p = \bar{x}^p + \bar{y}^p.$$

On peut itérer : $(\bar{x}+\bar{y}+\bar{z})^p = \bar{x}^p + (\bar{y}+\bar{z})^p$, etc., de sorte qu'en écrivant un entier quelconque $a = 1+1+\cdots+1$ (a fois), on a

$$\bar{a}^p = (\bar{1}+\bar{1}+\cdots+\bar{1})^p = \bar{1}^p + \bar{1}^p + \cdots + \bar{1}^p = \bar{1}+\bar{1}+\cdots+\bar{1} = \bar{a}$$

Ainsi, tout élément de $\mathbb{Z}/p\mathbb{Z}$ vérifie

$$\bar{a}^p = \bar{a}$$

et s'il est inversible,

$$\bar{a}^{p-1} = \bar{1}.$$

Cet énoncé constitue le *petit théorème de Fermat*, qu'on énonce habituellement avec des congruences :

"Si p est premier, tout entier a vérifie $a^p \equiv a \mod p$, et si de plus a est premier avec p, alors $a^{p-1} \equiv 1 \mod p$."

In [23]:
```
# Par exemple, 2^607-1 est premier
p
```

Out[23]:
```
53113799281676709868958820655246862732959311772703192319944413820040355986085224273916250226522928566888932948624650101534657933765270723940951997876658735194383127083539321903172812 7
```

In [24]:
```
# On écrira une fonction powermod(a,m,p) qui calcule a**m % p
powermod(42,p-1,p)
```

Out[24]:
```
1
```

In [25]:
```
# Celui ci n'est pas premier
n = 2**647-1
n
```

Out[25]:
```
58399239905564098798606996552963728958633324892781567111413664229110722140271070547275683984862353917166621562542008413576815420433605606377634064892444341609625531831811391361060789660756528332 7
```

In [26]:
```
# Ce calcule le prouve
powermod(3,n-1,n)
```

Out[26]:
```
21428223903520356654322755377445578958208675410619789103596222568646209820633205931060590298644292314996507952112174748503760913459299420318597663375422240564022629209691880886344754719021729800
```

Exponentiation modulaire

La fonction `powermod` calcule efficacement $a^m \mod n$ par le procédé suivant (voir ici (https://fr.wikipedia.org/wiki/Exponentiation_modulaire)). On écrit (en pensée) l'exposant m en binaire $m = b_k \cdots b_1 b_0$. Les bits b_i sont calculés au fur est à mesure. Au départ, le résultat intermédiare `res` est 1. On calcule $b_0 = m \mod 2$. Si $b_0 = 1$ on multiplie `res` par a modulo n. On divise m par 2, on remplace a par $a^2 \mod n$, et on recommence. Ceci calcule

$$\bar{a}^m = (\bar{a})^{b_0}(\bar{a}^2)^{b_1})(\bar{a}^{(2^2)})^{b_2} \cdots (\bar{a}^{(2^k)})^{b_k}$$

en remarquant que $(x^{(2^i)})^2 = x^{(2^{i+1})}$ (programme de maths de 4ème ...).

Tests de primalité

Le théorème de Fermat permet de prouver qu'un nombre n *n'est pas* premier en exhibant un a tel que $a^{n-1} \not\equiv 1 \mod n$. C'est le *test de Fermat*. On peut montrer que chaque fois qu'on obtient 1, la probabilité que n ne soit pas premier est *presque* divisée par 4. Malheureusement, il existe des nombres non premiers (les nombres de Carmichael (https://fr.wikipedia.org/wiki/Nombre_de_Carmichael)) qui passent tous les tests de Fermat avec $a \wedge n = 1$. Ils sont très rares, mais suffisent à faire capoter l'entreprise.

In [27]:
```
# en voilà un assez grand
c = 1590231231043178376951698401
for a in range(2,12): print (powermod(3,c-1,c), end= ' ')
```

```
1 1 1 1 1 1 1 1 1 1
```

In [28]:
```
print (factor(c))
```

```
[(17, 1), (19, 1), (23, 1), (29, 1), (31, 1), (37, 1), (41, 1), (43, 1), (61, 1), (67, 1), (71, 1),
(73, 1), (79, 1), (97, 1), (113, 1), (199, 1)]
```

Le *test de Miller-Rabin* permet de remédier à ce problème. C'est une petite modification du calcul de $a^{n-1} \mod n$ dans laquelle on rajoute un test.

L'idée est que dans un corps, l'équation $x^2 - 1 = 0$ ne peut pas avoir plus de deux solutions. Dans les entiers modulo n, on a toujours $\bar{1}$ et $-\bar{1}$, et $x^2 - \bar{1} = (x - \bar{1})(x + \bar{1})$. Si on en trouve une autre $\bar{b}$, alors $(\bar{b} - \bar{1})(b + \bar{1}) = \bar{b}^2 - \bar{1} = \bar{0}$, donc on a des diviseurs de 0 et n n'est pas premier.

Pour cela, on écrit $n - 1 = 2^k m$ avec m impair, et on commence par calculer $b = a^m \mod n$. Si $b = 1$, on renvoie True (probablement premier). Sinon, on calcule les $b^{(2^i)} \mod n$ et la première fois que le résultat vaut 1, on teste si la valeur précédente était $\equiv -1 \mod n$. Si ce n'est pas le cas, on renvoie False (composé). Si on arrive à $b^{(2^k)}$ sans avoir rencontré de racine carrée non triviale de 1, on renvoie True.

In [29]:
```
miller_rabin(c)
```

Out[29]:
```
False
```

Le théorème des restes chinois

Il permet de résoudre les système de conguences

$$\begin{cases} x & \equiv & x_1 & [m_1] \\ x & \equiv & x_2 & [m_2] \\ \vdots & \cdots & \vdots & \\ x & \equiv & x_n & [m_n] \end{cases}$$

lorsque les modules m_i sont deux à deux premiers entre eux.

Pour chaque i, il est facile de trouver a_i qui vérifie $a_i \equiv 0\ [m_j]$ pour tous les $j \neq i$. Il suffit de prendre $a_i = \prod_{j \neq i} m_j$. Ce nombre est premier avec m_i, donc inversible modulo m_i. Calculons les coefficients de Bézout tels que $1 = u_i a_i + v_i m_i$. Alors, $y_i = u_i a_i$ vérifie $y_i \equiv 1\ [m_i]$ et $y_i \equiv 0\ [m_j]$ pour $j \neq i$, ce qui entraîne que

$$X := y_1 x_1 + y_2 x_2 + \cdots y_n x_n$$

est solution du système. Il y en a d'autres : en ajoutant le produit $m = m_1 m_2 \cdots m_n$ à une solution, on obtient une autre solution, et deux solutions quelconques x', x'' diffèrent forcément d'un multiple de de m, puisque leur différence est divisible par tous les m_i.

La plus petite solution positive du système est donc

$$x = y_1 x_1 + y_2 x_2 + \cdots y_n x_n \mod m.$$

Voir ici (https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_des_restes_chinois).

Par exemple, avec $(x_1, x_2, x_3, x_4) = (1, 2, 3, 4)$ et (m_1,m2,m_3,m4)=(1,13,17,19), $ontrouve$ x= 41823$:

```
solve_chinese([1,2,3,4], [11,13,17,19])
41823
```

Quelques résultats de théorie des groupes

On connaît déjà quelques exemples de groupes finis : les $(\mathbb{Z}/n\mathbb{Z})^\times$, les éléments inversibles de $\mathbb{Z}/n\mathbb{Z}$, forment un groupe dont le nombre d'éléments est noté $\varphi(n)$ (indicateur d'Euler).

Le théorème de Lagrange

On appelle *ordre* d'un élément g d'un groupe fini G (noté multiplicativement, d'élement neutre 1), le plus petit entier positif m tel que $g^m = 1$.

Il existe : puisque G est fini, la suite g^k des puissances de g contient forcément deux éléments égaux, et si $g^k = g^l$, ($k < l$), alors $g^{l-k} = 1$.

On appelle *ordre* du groupe G son nombre d'éléments.

Ce choix judicieux des définitions permet d'énoncer élégamment le *théorème de Lagrange* :

"Dans un groupe fini, l'ordre de tout élément est un diviseur de l'ordre du groupe."

En effet, prenons un élément h de G, d'ordre m dans un groupe d'ordre n. Alors, ou bien $m = n$, et c'est fini, ou bien $m < n$. Dans ce cas, on peut trouver un élément g_1 qui n'est pas dans l'ensemble H des puissances de h. L'ensemble $g_1H = \{g_1, g_1h, \ldots, g_1h^{m-1}\}$ a lui aussi m éléments distincts, dont aucun n'est dans H : si on avait $g_1h^i = h^j$ alors, $g_1 = h^{j-i}$ serait dans H. Si on ne peut pas trouver g_2 qui ne soit ni dans H, ni dans g_1H, alors, l'ordre n de G est $n = 2m$, et on a fini. Sinon, on recommence avec g_2, et ainsi de suite, jusqu'à obtenir G comme réunion disjointe d'ensembles $H, g_1H, \ldots, g_{k-1}H$ ayant tous m éléments, et alors, $n = km$.

Fermat

Si $G = (\mathbb{Z}/p\mathbb{Z})^\times$ avec p premier, $n = p - 1$, et donc tout élément vérifie $\bar{a}^{p-1} = \bar{1}$.

Euler

Si $G = (\mathbb{Z}/n\mathbb{Z})^\times$, d'ordre $\varphi(n)$, tout élément vérifie $\bar{a}^{\varphi(n)} = \bar{1}$.

Calcul de $\varphi(n)$

Si $n = p_1^{m_1} \cdots p_r^{m_r}$ est la décomposition de n en facteurs premiers, le théorème des restes chinois montre que

$$\mathbb{Z}/n\mathbb{Z} \simeq \mathbb{Z}/p_1^{m_1}\mathbb{Z} \times \cdots \times \mathbb{Z}/p_r^{m_r}\mathbb{Z}$$

les opérations s'effectuant composante à composante sur les r-uples. Donc, un élément sera inversible si et seulement si ses résidus modulo tous les $p_i^{m_i}$ le sont.

Il suffit donc de savoir calculer $\varphi(p^m)$. Les éléments *non inversibles* de $\mathbb{Z}/p^m\mathbb{Z}$ sont les entiers $0 \leq a < p^m$ qui sont divisibles par p, c'est à dire $0, p, 2p, 3p, \ldots, (p^{m-1} - 1)p$. Il y en a donc p^{m-1}, et $\varphi(p^m) = p^m - p^{m-1}$.

Le produit de ces nombres peut s'écrire

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

(produit sur les diviseurs premiers de n).

On peut exprimer cette formule grace à la *fonction de Möbius* μ

$$\mu(n) = \begin{cases} 1 & \text{si } n = 0 \\ (-1)^r & \text{si } n = p_1 \cdots p_r \text{ est produit de } r \text{ nombres premiers distincts} \\ 0 & \text{sinon} \end{cases}$$

En développant le produit, on a

$$\varphi(n) = \sum_{d|n} \mu(d) \frac{n}{d}.$$

Remarquons que $\varphi(n)$ est aussi le nombre d'élements d'ordre (additif) n dans $\mathbb{Z}/n\mathbb{Z}$. En effet, $kx \equiv 0 \mod n \Rightarrow n|k$ si et seulement si $x \wedge n = 1$. Pour un diviseur d de n, $\varphi(d)$ est donc le nombre d'éléments d'ordre d dans $\mathbb{Z}/n\mathbb{Z}$, et on a

$$n = \sum_{d|n} \varphi(d).$$

L'équivalence des deux expressions est un cas particulier de la formule d'inversion de Möbius (https://fr.wikipedia.org/wiki/Fonction_de_M%C3%B6bius), qui relie deux fonctions f et g sur les entiers positifs

$$g(n) = \sum_{d|n} f(d) \quad \Leftrightarrow \quad f(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right).$$

Groupes cycliques

Un groupe formé des puissances d'un seul élément est dit *cyclique* (pourquoi, à votre avis ?).

Par exemple, le groupe *additif* $(\mathbb{Z}/n\mathbb{Z}, +)$ est cyclique : il est formé des "puissances additives" de $\bar{1}$.

Le théorème de Lagrange entraîne que *tout groupe fini d'ordre premier est cyclique*.

La propriété suivant est moins évidente : *Si p est premier, le groupe multiplicatif $G_p = (\mathbb{Z}/p\mathbb{Z})^\times$ est cyclique.*

Autrement dit, il existe (au moins) un élément g, appelé *générateur* ou *élément primitif* dont les puissances modulo p prennent toutes les valeurs de 1 à $p-1$.

Par exemple, $g = 3$ est primitif modulo $p = 31$:

In [31]:
```
p = 31
for i in range(31): print (pow(3,i,31),end=' ')
```

```
1 3 9 27 19 26 16 17 20 29 25 13 8 24 10 30 28 22 4 12 5 15 14 11 2 6 18 23 7 21 1
```

On peut montrer que d'une manière générale, le groupe multiplicatif d'un corps fini est toujours cyclique.

Dans le cas des entiers modulo p, on peut le voir ainsi. Soit d un diviseur de $n = p-1$. Les éléments de G_p qui vérifient $x^d = 1$ forment un sous-groupe : si $a^d = 1$ et $b^d = 1$, alors $(ab)^d = a^d b^d = 1$. De plus, le polynôme $x^d - 1$ a exactement d racines distinctes modulo p. En effet, d'après Fermat,

$$x^{p-1} - \bar{1} = (x-\bar{1})(x-\bar{2})\cdots(x-\overline{p-1})$$

et $x^d - \bar{1}$ divise ce polynome :

$$x^{p-1} - \bar{1} = (x^d - \bar{1})g(x)$$

avec g de degré $p-1-d$. Comme g ne peut pas avoir plus de racines que son degré, il faut que $x^d - \bar{1}$ en ait exactement d.

Ses racines sont les éléments de G_p dont les ordres sont des diviseurs c de d. Si on note $\psi(c)$ le nombre d'éléments d'ordre c, on a donc

$$d = \sum_{c|d} \psi(c)$$

pour tout diviseur d de n.

Par exemple, pour $p = 13$, $p-1 = 12$ et

- $1 = \psi(1)$, donc $\psi(1) = 1$ (et 1 est l'unique élément d'ordre 1)
- $2 = \psi(1) + \psi(2) = \psi(2) + 1$, donc $\psi(2) = 1$ (-1 est l'unique élément d'ordre 2)
- $3 = \psi(1) + \psi(3) = \psi(3) + 1$, donc $\psi(3) = 2$
- $4 = \psi(1) + \psi(2) + \psi(4) = \psi(4) + 2$, donc $\psi(4) = 2$
- $6 = \psi(1) + \psi(2) + \psi(3) + \psi(6) = \psi(6) + 4$, donc $\psi(6) = 2$
- $12 = \psi(1) + \psi(2) + \psi(3) + \psi(4) + \psi(6) + \psi(12)$ donc $\psi(12) = 12 - 8 = 4$

Ceci entraîne que $\psi(p-1) = \varphi(p-1) \neq 0$, et donc qu'il existe au moins un élément d'ordre $p-1$.

In [ ]:
```
```