Les entiers et l'ordinateur

La représentation des nombres

L'écriture des nombres

On utilise un système de représentation dit de position (Gerbert d'Aurillac (940-1003), pape Sylvestre II)

On choisit une base $b \in \mathbb{N} \setminus \{0, 1\}$.

Alors $\forall n \in \mathbb{N}, \exists!(n_k, ..., n_0) \in \{0, 1, 2, ...b-1\}^{k+1}$ tels que

$$n = n_k \ldots n_3 n_2 n_1 n_0 \, [b] = \sum_{i=0}^{k} n_i b^i$$

Exemple de base

- Base dix et $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$:
  $9281 = 9 \times 10^3 + 2 \times 10^2 + 8 \times 10 + 1$
- Base deux et $\{0, 1\}$: $10010001 = 1 \times 2^7 + 1 \times 2^4 + 1$
- Base hexadécimale (16) et
  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$:
  $D3C2 = 13 \times 16^3 + 3 \times 16^2 + 12 \times 16 + 2$

Les entiers de l'ordinateur

Au sein de la machine, les nombres sont représentés en base 2, un chiffre est appelé "**bit**" (binary digit, Claude Shannon, 1940) .

Un entier est codé sur un nombre fini k de bits ($k = 8$ à 64)

De ce fait, seuls les nombres strictement inférieurs à 2^k sont représentables.

Soit $n = 39011 = 2^{15} + 2^{12} + 2^{11} + 2^6 + 2^5 + 2^1 + 2^0$ sur 16 bits, se code en

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

L'arithmétique est donc modulaire, modulo 2^k.

Passage de la base 10 à la base b

On utilise la division euclidienne :

$$\forall (a,b) \in {\mathbb{N}^*}^2, \exists! (q,r) \in \mathbb{N}^2 \text{ tels que } a = bq + r \text{ avec } 0 \leqslant r < b$$

On note $q = a/b$ et $r = a\%b$

Algorithme :

$i \leftarrow 0$

Tant que $n > 0$ **faire**

$n_i \leftarrow n\%b$

$n \leftarrow n/b$

$i \leftarrow i + 1$

$i \leftarrow 0$

Tant que $n > 0$ **faire**

$n_i \leftarrow n\%b$

$n \leftarrow n/b$

$i \leftarrow i + 1$

Exemple : 1766 en base 7

Cas particulier de la base 2

On pourrra avoir intérêt à commencer par les puissances élevées :

$$\begin{array}{cccccccccccccccc}
2, & 4, & 8, & 16, & 32, & 64, & 128, & 256, & 512, & 1024, & 2048, & 4096, & 8192, & 16384, & 32768, & 65536 \\
1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 9, & 10, & 11, & 12, & 13, & 14, & 15, & 16
\end{array}$$

Exemple : 2321

Cas particulier de la base 2 vers la base 16 et réciproquement

De la base 2 vers la base 16 : on groupe les bits par 4

$1011001101011000100100100\,[2] =$

$101\ 1001\ 1010\ 1100\ 0100\ 1001\ 0100\,[2] = 59AC494\,[16]$

De la base 16 vers la base 2, on développe chaque chiffre hexadécimal.

$7A5B1F0F\,[16] = 0111\ 1010\ 0101\ 1011\ 0001\ 1111\ 0000\ 1111\,[2]$

Fin

Les entiers non-signés en C

En C, les entiers `int` et `long int` sont signés par défaut
On déclare un entier non-signé comme `unsigned int`
Un entier C non-signé codé sur k bits

- peut représenter les valeurs entre 0 et $2^k - 1$,
- respecte une arithmétique modulaire $\mod 2^k$

Sauf qu'on ne connaît pas k pour `unsigned int`.
Alors:

- mettre `#include <stdint.h>` et
- utiliser `uint8_t`, `uint16_t`, `uint32_t` ou `uint64_t`

Attention: les constantes entières sont signées par défaut !
$\Rightarrow$ Utiliser `2019u`.

Effets de bord

Soit le programme C

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

void main()
{
  uint16_t n=1;
  int i;
  for(i=1; i<10; i++)
    {
      n *= 10;
      printf("i = %2d, n = %10d \n",i,n);
    }
}
```

Effets de bord

i = 1, n = 10
i = 2, n = 100
i = 3, n = 1000
i = 4, n = 10000
i = 5, n = 34464
i = 6, n = 16960
i = 7, n = 38528
i = 8, n = 57600
i = 9, n = 51712

car $2^{16} = 65536$

Effets de bord

De même, avec `uint32_t`:

$$\begin{array}{rcr} 2^{30} & \rightarrow & 1073741824 \\ 2^{31} & \rightarrow & 2147483648 \\ 2^{32} & \rightarrow & 0 \\ 2^{33} & \rightarrow & 0 \end{array}$$

car $\mathbf{2^{31} \times 2 = 0}$!!

Les entiers non-signés en C

Dans la machine, les entiers sont représentés en binaire c-à-d pour 42, on a bien l'octet 00101010 quelque part.

Mais: on programme en C, donc on a le droit d'écrire 42 en décimal et c'est le compilateur qui se charge de la conversion.

Pour accéder au codage binaire, il faut utiliser les opérations de masque et décalage.

Les masques

En C, pour deux valeurs `uint32_t a, b`

- `a & b` est le ET logique de tous les bits de `a` et de `b`

$$\begin{array}{rcccc} \texttt{a} & = & 42 & = & 00101010 \\ \texttt{b} & = & 22 & = & 00010110 \\ \hline \texttt{a \& b} & = & 2 & = & 00000010 \end{array}$$

- `a | b` est le OU logique de tous les bits de `a` et de `b`

$$\begin{array}{rcccc} \texttt{a} & = & 42 & = & 00101010 \\ \texttt{b} & = & 22 & = & 00010110 \\ \hline \texttt{a | b} & = & 62 & = & 00111110 \end{array}$$

- Avec un tel masque & on peut donc accéder aux différents bits d'un entier en C:
  - `a & 1u` est non-nul ssi `a` a son bit à droite à 1
  - `a & 2u` est non-nul ssi `a` a son bit de poids 1 à 1
  - `a & 4u` est non-nul ssi `a` a son bit de poids 2 à 1
  - ...

Les décalages

En C, pour une valeur `uint32_t` `a` et un entier `b`

- `a << b` a les mêmes bits que `a` décalés de b bits à gauche

$$\begin{array}{rcccc} \texttt{a} & = & 42 & = & 00101010 \\ \hline \texttt{a << 2} & = & 168 & = & 10101000 \end{array}$$

- `a >> b` a les mêmes bits que `a` décalés de b bits à droite

$$\begin{array}{rcccc} \texttt{a} & = & 68 & = & 01000100 \\ \hline \texttt{a >> 2} & = & 17 & = & 00010001 \end{array}$$

- Les bits *en trop* à gauche ou à droite *disparaissent*.

- `1u << k` a un seul bit à 1 à la k-ième position

$\Rightarrow$ `a & (1u << k)` est non-nul ssi `a` a son bit de poids k à 1.

Les décalages et les masques

- De la même façon qu'on peut abréger `a = a + b` en `a += b`,
  on peut abréger `a = a & b` en `a &= b`,
  `a = a | b` en `a |= b`,
  `a = a >> b` en `a >>= b` et
  `a = a << b` en `a <<= b`.

Fin

Les entiers signés en C

Pour représenter des entiers signés (i.e. les entiers relatif de $\mathbb{Z}$, on utilise le **complément à deux**.

Plus précisement, sur k bits

1 - si $n \in \left[0; 2^{k-1} - 1\right]$, n est codé tel quel sur $k - 1$ bits,

2 - $n \in \left[-2^{k-1}; -1\right]$,on code $2^k + n \in \left[2^{k-1}; 2^k - 1\right]$ donc le bit de gauche vaut 1.

- Accès au signe facile: prendre le bit à gauche
- Une représentation unique
- Rien n'est à changer dans les algorithmes pour $+, -, <$.

C'est la représentation utilisée dans les processeurs actuels.

Pour passer du codage de $n > 0$ au codage de $-n$ (et inversement) sur k bits, on va au premier bit non nul par la droite. On ne touche pas à ce bit puis on inverse tous les autres bits sur la gauche..
Soit $n = 0a_{k-2}...a_{k_0}10...0\,[2]$.

Quelques exemples d'opérations.
On travaille sur 8 bit.
Soit $a = 57 = 0011\ 1001\,[2]$ et $b = 44 = 0010\ 1100\,[2]$

$2^8 - a = 199 = 1100\ 0111\,[2]$ et $2^8 - b = 212 = 1101\ 0100\,[2]$

$$\begin{array}{rcl} 57 & = & 0011\ 1001 \\ 44 & = & 0010\ 1100 \\ \hline 101 & = & 0110\ 0101 \end{array} \qquad \begin{array}{rcl} 2^8 - 57 = 199 & = & 1100\ 0111 \\ 2^8 - 44 = 212 & = & 1101\ 0100 \\ \hline 2^8 - 101 = 155 & = & 1001\ 1011 \end{array}$$

$$\begin{array}{rcl} 57 & = & 0011\ 1001 \\ 2^8 - 44 = 212 & = & 1101\ 0100 \\ \hline 13 & = & 00001101 \end{array} \qquad \begin{array}{rcl} 2^8 - 57 = 199 & = & 1100\ 0111 \\ 44 & = & 0010\ 1100 \\ \hline 2^8 - 13 = 243 & = & 1111\ 0011 \end{array}$$

Les entiers signés de l'ordinateur

Pour passer à la ligne suivante, on ajoute 1 sur le premier bit et on propage la rétenue.

écritures binaire	valeurs absolues	valeurs signées
0000000000000000	0	0
0000000000000001	1	1
0000000000000010	2	2
0000000000000011	3	3
...	...	...
0111111111111110	$2^{15}-2$	$2^{15}-2$
0111111111111111	$2^{15}-1$	$2^{15}-1$
1000000000000000	$2^{15}=2^{16}-2^{15}$	-2^{15}
1000000000000001	$2^{15}+1$	$-(2^{15}-1)$
...	...	...
1111111111111110	$2^{16}-2$	-2
1111111111111111	$2^{16}-1$	-1

Les entiers signés en C

En C, les entiers `int` et `long int` sont signés par défaut
Un entier C signé

- peut représenter les valeurs entre -2^{k-1} et $2^{k-1} - 1$,
- respecte une arithmétique modulaire $\mod 2^k$

Pour autant, il est préférable d'éviter d'utiliser `int` et `long int` pour garantir une portabilité optimale du code.
Alors:

- mettre `#include <stdint.h>` et
- utiliser `int8_t`, `int16_t`, `int32_t` ou `int64_t`

Rappel: les constantes entières sont signées par défaut.

Second effet de bord

Soit le programme C

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

void main()
{
  int16_t n=1;
  int i;
  for(i=1; i<10; i++)
    {
      n *= 10;
      printf("i = %2d, n = %10d \n",i,n);
    }
}
```

Second effet de bord

i = 1, n = 10
i = 2, n = 100
i = 3, n = 1000
i = 4, n = 10000
i = 5, n = -31072
i = 6, n = 16960
i = 7, n = -27008
i = 8, n = -7936
i = 9, n = -13824

car $2^{16} = 65536$

Fin

L'algorithme d'Euclide et le théorème chinois des restes

Rappels de maths et PGCD

Rappels de math

Division Euclidienne :
$\forall (a,b) \in \mathbb{Z} \times \mathbb{N}^*, \exists ! (q,r) \in \mathbb{Z} \times \mathbb{N}$ tels que $a = bq + r$ avec $0 \leq r < b$

$\mathbb{Z}$ est un anneau principal : les seuls idéaux de $\mathbb{Z}$ sont les $n\mathbb{Z} = \{nq, q \in \mathbb{Z}\}$

On dit que a est congru à c modulo b et on écrit

$$a \equiv c \pmod{b} \Leftrightarrow (a - c) \in b\mathbb{Z}$$

la relation $\equiv$ est stable par addition et multiplication

$$a_1 \equiv c_1 \pmod{b} \text{ et } a_2 \equiv c_2 \pmod{b} \Rightarrow a_1 + a_2 \equiv c_1 + c_2 \pmod{b}$$
$$a_1 \equiv c_1 \pmod{b} \text{ et } a_2 \equiv c_2 \pmod{b} \Rightarrow a_1 \times a_2 \equiv c_1 \times c_2 \pmod{b}$$

la relation $\equiv$ est une relation d'équivalence

$\mathbb{Z}/n\mathbb{Z}$ est un anneau commutatif

Rappels de math

On dit que b divise a ssi $a \in b\mathbb{Z}$

Le plus grand commun diviseur d de a et b, noté $\mathrm{pgcd}(a, b)$, est défini par

$$a\mathbb{Z} + b\mathbb{Z} = d\mathbb{Z}$$

donc si $d = \mathrm{pgcd}(a, b)$ alors (identité de Bézout)

$$\exists (u, v) \in \mathbb{Z}^2 \text{ tels que } au + bv = d$$

a et b sont dit premier entre eux ssi $\mathrm{pgcd}(a, b) = 1$.

$\mathbb{Z}/n\mathbb{Z}$ est un corps ssi n est premier.

Calcul du PGCD à base de soustraction

L'algorithme est basé sur la remarque suivante :
si $a > b$ alors $\text{pgcd}(a, b) = \text{pgcd}(a - b, b)$

Entrées A et B

Sortie U le pgcd de A et de B

Corps

$U \leftarrow A$

$V \leftarrow B$

Tant que $V > 0$ **faire**

  Si $V > U$ alors

    $T \leftarrow V$, $V \leftarrow U$, $U \leftarrow T$

  $U \leftarrow U - V$

Invariant de boucle : $U \geq V$ et $\text{pgcd}(U, V) = \text{pgcd}(U - V, V)$
Terminaison : si $V = 0$ alors $U = \text{pgcd}(U, V)$

Exemple

U	V	U - V
1479	699	780
780	699	81
699	81	618
618	81	537
...	...	...
213	81	132
132	81	51
81	51	30
51	30	21
30	21	9
21	9	12
12	9	3
9	3	6
6	3	3
3	3	0
3	0	

Calcul du PGCD à base de division

Entrées A et B

Sortie U le pgcd de A et de B

Corps

$U \leftarrow A$

$V \leftarrow B$

Tant que $V > 0$ **faire**

$T \leftarrow U\%V$

$U \leftarrow V$

$V \leftarrow T$

Le PGCD

Algoritme à base de division : exemple

U	V	U % V
1479	699	81
699	81	51
81	51	30
51	30	21
30	21	9
21	9	3
9	3	0
3	0	

Fin

l'algorithme d'Euclide étendu

Rappels de math

Identité de Bézout : Soient a et b deux entiers, il existe deux entiers u et v tels que : $a \times u + b \times v = \text{pgcd}(a, b)$

Remarque 1 : a et b premiers entre eux $\Leftrightarrow a \times u + b \times v = 1$

Remarque 2 : $a \times u + b \times v = 1$
$\Rightarrow a^{-1} \equiv u \pmod{b}$ et $b^{-1} \equiv v \pmod{a}$

L'algorithme d'Euclide étendu

Le but est de calculer les coefficients de Bézout :

$$u_1 \times a + u_2 \times b = u_3 = \mathrm{pgcd}(a, b)$$

Entrées : a et b

Sortie : $u_1 \times a + u_2 \times b = u_3 = \mathrm{pgcd}(a, b)$

Initialisation :
$$\begin{array}{lcl} (u_1, u_2, u_3) & \leftarrow & (1, 0, a) \\ (v_1, v_2, v_3) & \leftarrow & (0, 1, b) \end{array}$$

Itération : tant que $v_3 \neq 0$

$$\begin{array}{lcl} q = u_3 / v_3 & & \\ (t_1, t_2, t_3) & \leftarrow & (u_1, u_2, u_3) - q \times (v_1, v_2, v_3) \\ (u_1, u_2, u_3) & \leftarrow & (v_1, v_2, v_3) \\ (v_1, v_2, v_3) & \leftarrow & (t_1, t_2, t_3) \end{array}$$

Exemple

On veut calculer le pgcd et les coefficients de Bézout pour $a = 391$ et $b = 276$.

u_1	u_2	u_3	v_1	v_2	v_3	q	t_1	t_2	t_3
1	0	391	0	1	276	1	1	-1	115

Exemple

On veut calculer le pgcd et les coefficients de Bézout pour $a = 391$ et $b = 276$.

u_1	u_2	u_3	v_1	v_2	v_3	q	t_1	t_2	t_3
1	0	391	0	1	276	1	1	-1	115
0	1	276	1	-1	115	2	-2	3	46

Exemple

On veut calculer le pgcd et les coefficients de Bézout pour $a = 391$ et $b = 276$.

u_1	u_2	u_3	v_1	v_2	v_3	q	t_1	t_2	t_3
1	0	391	0	1	276	1	1	-1	115
0	1	276	1	-1	115	2	-2	3	46
1	-1	115	-2	3	46	2	5	-7	23

Exemple

On veut calculer le pgcd et les coefficients de Bézout pour $a = 391$ et $b = 276$.

u_1	u_2	u_3	v_1	v_2	v_3	q	t_1	t_2	t_3
1	0	391	0	1	276	1	1	-1	115
0	1	276	1	-1	115	2	-2	3	46
1	-1	115	-2	3	46	2	5	-7	23
-2	3	46	5	-7	23	2	-12	17	0

Exemple

On veut calculer le pgcd et les coefficients de Bézout pour $a = 391$ et $b = 276$.

u_1	u_2	u_3	v_1	v_2	v_3	q	t_1	t_2	t_3
1	0	391	0	1	276	1	1	-1	115
0	1	276	1	-1	115	2	-2	3	46
1	-1	115	-2	3	46	2	5	-7	23
-2	3	46	5	-7	23	2	-12	17	0
5	-7	23	-12	17	0				

donc l'identité de Bézout s'écrit

$$5 * 391 - 7 * 276 = 23 \text{ et } \operatorname{pgcd}(391, 276) = 23.$$

Fin

Le théorème chinois des reste

Il apparaitrait pour la première fois dans le livre de Sun Zi, le Sunzi suanjing, datant du IIIe siècle

Théorème chinois des restes

Étant donné des entiers (*moduli*) $m_1, \ldots, m_n$ premiers entre eux deux à deux et des restes associés $a_1, \ldots, a_n$, on cherche à résoudre le système de congruences

$$\left\{\begin{array}{lllll} x & \equiv & a_1 & & \mod m_1 \\ x & \equiv & a_2 & & \mod m_2 \\ \vdots & & & & \\ x & \equiv & a_n & & \mod m_n \end{array}\right.$$

où x est l'inconnue.

Les outils mathématiques :

Si a et premier avec b et premier avec c alors a est premier avec bc.

Les outils mathématiques :

Si a est premier avec b et premier avec c alors a est premier avec bc.

Si m est premier, $\forall a, \exists b$ tel que $ab \equiv 1 \pmod{m}$

Les outils mathématiques :

Si a est premier avec b et premier avec c alors a est premier avec bc.

Si m est premier, $\forall a, \exists b$ tel que $ab \equiv 1 \pmod{m}$

Si m_1 et m_2 sont premiers entre eux et que a est un multiple de m_1 et de m_2 alors a est un multiple de $m_1 \times m_2$.

Étude Théorique

$$\begin{cases} x & \equiv & a_1 & \mod m_1 \\ x & \equiv & a_2 & \mod m_2 \\ \vdots & & & \\ x & \equiv & a_n & \mod m_n \end{cases}$$

Théoreme : Soit $M = \prod_{i=1}^{n} m_i$ et $M_i = \frac{M}{m_i}$, soit $y_i = M_i^{-1} \pmod{m_i}$ alors

$$y = \sum_{i=1}^{n} a_i \times y_i \times M_i$$

est une solution du système modulaire et l'ensemble des solutions est $S = \{y + k \times M, k \in \mathbb{Z}\}$

Démonstration

Vérifions que $\forall 1 \leq i \leq n, y \equiv a_i \pmod{m_i}$

D'abord, si $i \neq j, M_j \equiv 0 \pmod{m_i}$ car $M_j = \prod_{k=1,k\neq j}^{n} m_k$ est un multiple de m_i.

Donc, soit $1 \leq i_0 \leq n$,

$$\sum_{k=1}^{n} a_k \times y_k \times M_k \equiv a_{i_0} \times y_{i_0} \times M_{i_0} \pmod{m_{i_0}}$$

Par définition,

$$y_{i_0} \times M_{i_0} \equiv 1 \pmod{m_{i_0}}$$

donc

$$y \equiv a_{i_0} \pmod{m_{i_0}}$$

Démonstration suite

Supposons que y et z soient deux solutions du systèmes modulaires. On a

$$\begin{array}{rcll} y & \equiv & a_1 & \mod m_1 \\ y & \equiv & a_2 & \mod m_2 \\ \vdots & & & \\ y & \equiv & a_n & \mod m_n \end{array} \quad \text{et} \quad \begin{array}{rcll} z & \equiv & a_1 & \mod m_1 \\ z & \equiv & a_2 & \mod m_2 \\ \vdots & & & \\ z & \equiv & a_n & \mod m_n \end{array}$$

Et donc
$$\begin{array}{rcll} y-z & \equiv & 0 & \mod m_1 \\ y-z & \equiv & 0 & \mod m_2 \\ & \vdots & & \\ y-z & \equiv & 0 & \mod m_n \end{array}$$

Donc $y - z$ est un multiple de $m_i, \forall i$
donc $y - z$ est un multiple de M.

Algorithme

On résout le système de proche en proche.

Entrées $m_1, \ldots, m_n$ des moduli premiers entre eux
$a_1, \ldots, a_n$ des entiers restes

Sortie x resolvant le système de congruences

Corps

$M \leftarrow m_1$

$x \leftarrow a_1$

Pour $i = 2, \ldots, n$ **faire**

Euclide étendu: calculer (u, v) tq.

$u\, m_i + v\, M = 1$

$x \leftarrow u\, m_i\, x + v\, M\, a_i$

$M \leftarrow m_i\, M$

$x \leftarrow x \mod M$

Application

Une jeune fille portait un panier rempli d'œufs. Un chevalier passa avec son cheval et toucha le panier, qui tomba par terre et tous les œufs se cassèrent. Il voulut dédommager la fille et il lui demanda combien d'œufs elle avait eu.
Elle ne sut plus dire leur nombre mais elle se rappela que quand elle les avait comptés par paires, un œuf était resté seul, que quand elle avait compté par triplets, deux étaient restés et quand elle les avait arrangés par groupes de cinq, quatre étaient restés. Finalement, quand elle les avait comptés par groupes de sept, aucun œuf n'était resté à côté.
Alors le chevalier répondit: maintenant je sais combien d'œufs il y avait.

Brahmagupta, VIIe siècle

$$\begin{aligned} x &\equiv 1 \quad \mod 2 \\ x &\equiv 2 \quad \mod 3 \\ x &\equiv 4 \quad \mod 5 \\ x &\equiv 0 \quad \mod 7 \end{aligned}$$

$$\begin{array}{lcll} x & \equiv & 1 & \mod 2 \\ x & \equiv & 2 & \mod 3 \\ x & \equiv & 4 & \mod 5 \\ x & \equiv & 0 & \mod 7 \end{array} \qquad x \leftarrow u\, m_i\, x + v\, M\, a_i$$

On organise les calculs dans un tableau comme suit:

i	m_i	a_i	M	u	v	x	$x \mod M$
1	2	1	2			1	1
2							
3							
4							

$$
\begin{aligned}
x &\equiv 1 \mod 2\\
x &\equiv 2 \mod 3\\
x &\equiv 4 \mod 5\\
x &\equiv 0 \mod 7
\end{aligned}
\qquad x \leftarrow u\, m_i\, x + v\, M\, a_i
$$

i	m_i	a_i	M	u	v	x	$x \mod M$
1	2	1	2			1	1
2	3	2	6	1	-1	-1	-1
3							
4							

$$\begin{array}{llll} x & \equiv & 1 & \bmod 2 \\ x & \equiv & 2 & \bmod 3 \\ x & \equiv & 4 & \bmod 5 \\ x & \equiv & 0 & \bmod 7 \end{array}$$

$x \leftarrow u\, m_i\, x + v\, M\, a_i$

i	m_i	a_i	M	u	v	x	$x \bmod M$
1	2	1	2			1	1
2	3	2	6	1	-1	-1	-1
3	5	4	30	-1	1	29	-1
4							

$$
\begin{aligned}
x &\equiv 1 \mod 2\\
x &\equiv 2 \mod 3\\
x &\equiv 4 \mod 5\\
x &\equiv 0 \mod 7
\end{aligned}
\qquad x \leftarrow u\, m_i\, x + v\, M\, a_i
$$

i	m_i	a_i	M	u	v	x	$x \mod M$
1	2	1	2			1	1
2	3	2	6	1	-1	-1	-1
3	5	4	30	-1	1	29	-1
4	7	0	210	-17	4		119

Fin