Capítulo 3

Práctica 3. Aritmética entera y modular.

3.1. Aritmética entera.

Dados dos números enteros, a, b, con $b \neq 0$, podemos realizar la división de a entre b, y eso nos da un cociente c y un resto r, satisfaciendo $a = b \cdot c + r$ y $0 \leq r < |b|$. Por ejemplo, si dividimos 27 entre 5, nos da de cociente 5 y resto 2. Maxima calcula el cociente y el resto mediante las instrucciones `quotient` y `mod`.

```
(%ixx) quotient(27,5);
(%oxx) 5
(%ixx) mod(27,5);
(%oxx) 2
(%ixx) quotient(605,31);
(%oxx) 19
(%ixx) mod(605,31);
(%oxx) 16
```

pues $605 = 31 \cdot 19 + 16$.

Cuando el dividendo es negativo, calcula el resto positivo. Por ejemplo:

```
(%ixx) mod(23,7); mod(-23,7);
(%oxx) 2
(%oxx) 5
```

ya que $23 = 7 \cdot 3 + 2$ y $-23 = 7 \cdot (-4) + 5$.

Sin embargo, tal y como Maxima trae implementadas las funciones `quotient` y `mod` no es cierto, si a es negativo, y b es positivo, que `a=b*quotient(a,b) + mod(a,b)`.

```
(%ixx) quotient(-23,7);
(%oxx) -3
```

Cuando debería darnos -4.

Cuando el divisor es negativo, el resto nos lo devuelve negativo.

```
(%ixx) mod(23,-7); mod(-23,-7);
(%oxx) -5
(%oxx) -2
```

Para evitar esto, podemos definirnos nosotros las funciones `cociente` y `modulo` de forma que el resto de una división siempre sea positivo o cero, y se satisfaga la igualdad $a = b \cdot \text{cociente}(a, b) + \text{modulo}(a, b)$. Para esto vamos a usar el operador condicional `if` y el operador lógico `or`

```
(%ixx) modulo(x,y):=if (y>0 or mod(x,y)=0) then mod(x,y) else mod(x,y)-y$
```

Después de `if` viene una expresión lógica. Si se evalúa como verdadera, entonces se ejecuta lo que va después de `then` Si se evalúa como falsa, se ejecuta lo que va después de `else` En este caso, la expresión lógica es la disyunción de dos expresiones más simples. Por tanto, van separadas por `or`. Para que se evalúe como verdadera hace falta que una de las dos (o las dos) expresiones (`y>0` ó `mod(x,y)=0`) se evalúe como verdadera.

```
(%ixx) cociente(x,y):=quotient(x-modulo(x,y),y)$
(%ixx) cociente(23,-5); modulo(23,-5); cociente(-23,-5); modulo(-23,-5);
(%oxx) -4
(%oxx) 3
(%oxx) -5
(%oxx) 2
```

Las funciones `gcd` y `lcm` nos calculan el máximo común divisor y el mínimo común múltiplo respectivamente de dos números (o polinomios).

```
(%ixx) gcd(45,123); lcm(45,123);
(%oxx) 3
(%oxx) 1845
```

La función `lcm` admite más argumentos, no así la función `gcd`

```
(%ixx) lcm(4,6,8);
(%oxx) 24
```

mientras que si introdujéramos `gcd(4,6,8);` nos daría un mensaje de error.
Sabemos que si $d = mcd(a, b)$ entonces podemos encontrar u, v enteros tales que $d = a \cdot u + b \cdot v$. Estos coeficientes, Maxima los calcula con la función `gcdex`

```
(%ixx) gcdex(45,123);
(%oxx) [11,-4,3]
```

Lo que nos dice que el máximo común divisor de 45 y 123 vale 3 (el último de la lista) y los coeficientes u y v son respectivamente 11 y -4, es decir, $3 = 45 \cdot 11 - 4 \cdot 123$.
Dada una ecuación de la forma $ax + by = c$ sabemos que tiene solución si, y sólo si, $mcd(a, b)|c$. En tal caso, para encontrar una solución podemos resolver $ax + by = mcd(a, b)$ y luego multiplicar la solución que nos haya dado por $\frac{c}{mcd(a,b)}$. Tenemos de esta forma una solución de la ecuación. Por ejemplo, vamos a buscar una solución de la ecuación $2475x + 7548y = 57$.
En primer lugar vemos si la ecuación tiene solución.

```
(%ixx) is(mod(57,gcd(2475,7548))=0);
(%oxx) true
```

Como tiene solución, calculamos una.

```
(%ixx) gcdex(2475,7548);
(%oxx) [-491,161,3]
```

Luego una solución es $x = -491 \cdot \frac{57}{3} = 9329$, $y = 161 \cdot \frac{57}{3} = 3059$.

```
(%ixx) x:-491*57/3; y:161*57/3;
(%oxx) -9329
(%oxx) 3059
```

Podemos automatizar todo este proceso, con una función que vamos a llamar `diofantica`, y que tendrá tres argumentos.

```
(%ixx) diofantica(a,b,c):=if mod(c,gcd(a,b))=0 then rest(gcdex(a,b),-1)*c/gcd(a,b)$
 else print("No tiene solucion");
```

Aquí hemos usado la función `rest` Esta función tiene dos argumentos: una lista y un número. `rest(lista,n)` devuelve la lista que resulta de eliminar los n primeros elementos `lista` si n es positivo, y la lista que resulta de eliminar los $-n$ últimos elementos de `lista` si n es negativo. En nuestro caso, al hacer `rest(gcdex(a,b),-1)` lo que hacemos es de la lista `gcdex(a,b)` eliminamos el último elemento, que es el máximo común divisor de a y b, y nos quedamos con los coeficientes u y v.
Lo primero que la función `diofantica` hace es comprobar si la ecuación $ax + by = c$ tiene solución. Para esto, comprueba si el resto de la división de c entre el máximo común divisor de a y b es cero. En caso afirmativo, da una solución, multiplicando los coeficientes u y v por $\frac{c}{mcd(a,b)}$. En caso negativo, nos dice que no tiene solución.

```
(%ixx) diofantica(2475,7548,57);
(%oxx) [-9329,3059]
(%ixx) diofantica(2465,7548,49);
No tiene solucion
(%oxx) No tiene solucion
```

Una vez encontrada una solución (x_0, y_0) de la ecuación $ax+by = c$, todas las soluciones pueden calcularse mediante la expresión $x = x_0 + k \cdot \frac{b}{d}$; $y = y_0 - k \cdot \frac{a}{d}$.

Si quisiéramos encontrar una solución a una ecuación de la forma $ax + by + cz = e$, el criterio para ver si tiene o no solución es el mismo. Si $mcd(a, b, c)|e$ tiene solución y en caso contrario no la tiene. Una forma de resolverla sería llamar d al máximo común divisor de a y b, resolver $du + cz = e$, y una vez hecho esto, resolver $ax + by = du$ (en lugar de haber tomado a y b podríamos haber tomado a y c, o b y c). Por ejemplo, vamos a encontrar una solución de $6x + 10y + 15z = 7$. Puesto que $mcd(6, 10) = 2$, resolvemos $2u + 15z = 7$.

```
(%ixx) diofantica(2,15,7);
(%oxx) [-49,7]
```

Y ahora resolvemos $6x + 10y = 2 \cdot (-49)$

```
(%ixx) diofantica(6,10,2*(-49));
(%oxx) [-98,49]
```

Luego una solución es $x = -98$, $y = 49$, $z = 7$.

Ejercicio.

Calcula, si es posible, dos soluciones enteras de la ecuación $1270500x + 7110675y + 19145672z = 1$.

Maxima también tiene implementados algunos aspectos relacionados con los números primos. Ya vimos en la práctica anterior como el comando `primep` nos decía si un número es o no primo.

Para factorizar un número como producto de primos tenemos la función `factor`

(%ixx) factor(24);

(%oxx) $2^3 3$

(%ixx) factor(60984);

(%oxx) $2^3 3^2 7\ 11^2$

También podemos usar la función `ifactors`. Esta función, aplicada a un número nos devuelve una lista, en la que cada elemento de la lista vuelve a ser una lista con dos entradas. Un primo, divisor del número, y el exponente al que aparece elevado ese primo en la descomposición.

```
(%ixx) ifactors(60984);
(%oxx) [[2,3],[3,2],[7,1],[11,2]]
```

Lo que significa que el número 60984 se factoriza como $2^3 \cdot 3^2 \cdot 7 \cdot 11^2$.

Para obtener los divisores de un número tenemos la función `divisors`

```
(%ixx) divisors(24);
(%oxx) {1,2,3,4,6,8,12,24}
```

Si ahora quisiéramos obtener una lista con los divisores primos de un número, podríamos definir la función `div_primos`.

```
(%ixx) div_primos(n):=makelist(ifactors(n)[i][1],i,1,length(ifactors(n)))$
(%ixx) div_primos(60984);
(%oxx) [2,3,7,11]
```

También podríamos obtenerlos con la siguiente función.

```
(%ixx) div_primos2(n):=subset(divisors(n),primep)$
(%ixx) div_primos2(60984);
(%oxx) {2,3,7,11}
```

Por último, mencionamos dos comandos que nos permiten obtener un número primo. Estos son `next_prime` y `prev_prime`, que nos dan el siguiente número primo (positivo), y el anterior número primo de un número dado.

```
(%ixx) next_prime(12345); prev_prime(12345);
(%oxx) 12347
(%oxx) 12343
```

3.2. Aritmética modular.

Vimos que la función `mod` nos calcula el resto de una división entera. O si queremos, podemos usar la función `modulo` que definimos previamente.

Entonces, para calcular sumas, restas y productos módulo un número, no tenemos más que introducir la expresión como el primer argumento de la función `mod`.

```
(%ixx) mod(6*9-4*(5-2),11);
(%oxx) 9
```

Pues el resultado de la operación $6 \cdot 9 - 4 \cdot (5-2)$ es 42, que al dividirlo por 11 da resto 9.
También podemos calcular potencias

```
(%ixx) mod(12^31,47);
(%oxx) 34
```

aunque Maxima dispone de un comando específico para el cálculo de potencias modulares. Este es `power_mod`.

```
(%ixx) power_mod(12,31,47);
(%oxx) 34
```

Y es conveniente calcularlas usando esta función. Sobretodo si trabajamos con números grandes. Por ejemplo, podemos ejecutar

```
(%ixx) power_mod(13579,123456789,987654321);
(%oxx) 691505902
```

Pero si quisiéramos ejecutar `mod(13579^123456789,987654321)`, Maxima probablemente se bloquearía, pues en primer lugar intenta calcular el número entero $13579^{123456789}$ y luego trataría de reducirlo módulo 987654321. Pero el número anterior es un número de más de 510 millones de cifras.
Prueba a escribir al azar tres números x, y, z de aproximadamente 200 cifras, y calcula `power_mod(x,y,z)`. Comprobarás como el cálculo es inmediato.
El comando `power_mod` puede usarse también con exponentes negativos.

```
(%ixx) power_mod(5,-2,9);
(%oxx) 4
```

Tomando como exponente -1 nos calcula el inverso de un número módulo otro.

```
(%ixx) power_mod(5,-1,9);
(%oxx) 2
```

Y es que, $2 \cdot 5 = 1$ módulo 9, luego $5^{-1} = 2$. Por tanto, $5^{-2} = (5^{-1})^2 = 2^2 = 4$, como nos ha calculado antes.
Si quisiéramos calcular el inverso de a módulo m, y tal inverso no existe (pues $mcd(a, m) \neq 1$), Maxima nos devuelve `false`

```
(%ixx) power_mod(6,-1,9);
(%oxx) false
```

El exponente debe ser un número entero (positivo o negativo). No admite, por ejemplo, como exponente $\frac{1}{2}$.
Para el cálculo de inversos modulares, Maxima dispone de una función para ello. Esta es `inv_mod`.

```
(%ixx) inv_mod(11,23);
(%oxx) 21
```

Maxima trae implementada la aritmética modular. Para ello dispone de una variable global, `modulus`, cuyo valor por defecto es `false`, pero que podemos asignarle cualquier valor natural mayor que cero (también admite el valor cero, pero luego da error al realizar los cálculos). Si introducimos un valor para `modulus` que no sea primo, Maxima nos da un aviso, pero admite ese valor.
Sin embargo, para que nos muestre el resultado como queremos, tenemos que decirle que nos simplifique la expresión, mediante el comando `rat`

```
(%ixx) modulus:11$
(%ixx) 6+7;
(%oxx) 13
(%ixx) rat(6+7);
(%oxx)/R/ 2
(%ixx) rat(7*8);
(%oxx)/R/ 1
```

La representación que trae Maxima de los enteros módulo m, comprende los números desde $\frac{-m+1}{2}$ hasta $\frac{m-1}{2}$, si m es impar, y desde $\frac{-m}{2}+1$ hasta $\frac{m}{2}$ si m es par.

```
(%ixx) rat(8+9);
(%oxx)/R/ -5
(%ixx) makelist(rat(i),i,0,20);
(%oxx)/R/ [0,1,2,3,4,5,-5,-4,-3,-2,-1,0,1,2,3,4,5,-5,-4,-3,-2]
(%ixx) modulus:10$
warning: assigning 10, a non-prime, to 'modulus'
```

```
(%ixx) makelist(rat(i),i,0,20);
(%oxx)/R/ [0,1,2,3,4,5,-4,-3,-2,-1,0,1,2,3,4,5,-4,-3,-2,-1,0]
```

La función `rat` lo que hace es simplificar expresiones racionales. Por ejemplo:

```
(%ixx) (x^2-1)/(x+1);
```

(%oxx) $\frac{x^2-1}{x+1}$

Si ahora introducimos la misma expresión, pero precedida del comando `rat`, vemos como la simplifica.

```
(%ixx) rat((x^2-1)/(x+1));
(%oxx) x-1
```

Cuando dentro de la función `rat` introducimos una expersión numérica (por ejemplo, $4 * 5$), lo que entiende Maxima por simplificar es reducirla módulo el valor que tenga en ese momento la variable `modulus`.

Cuando la variable `modulus` toma un valor distinto de `false`, Maxima realiza sus cálculos módulo ese valor. De esta forma, puede ser que obtengamos resultados no esperados al llamar a ciertas funciones.

```
(%ixx) gcd(6,27);
(%oxx) 1
```

Si le damos a `modulus` su valor por defecto, entonces realiza el cálculo correctamente.

```
(%ixx) modulus:false$ gcd(6,27);
(%oxx) 3
```

Dado un número natural n, definimos el conjunto $U(\mathbb{Z}_n)$ como el conjunto de todos los números menores que n que tienen inverso módulo n. Este conjunto se denomina el conjunto de las unidades de $\mathbb{Z}_n$. Por ejemplo, para $n = 28$, este conjunto es $\{1, 3, 5, 9, 11, 13, 15, 17, 19, 23, 25, 27\}$. Vamos a ver cómo podemos obtenerlo con Maxima.

En primer lugar tomamos el conjunto de todos los elementos menores que 28.

```
(%ixx) Z28:setify(makelist(i,i,0,27));
(%oxx) {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27}
```

Y ahora nos quedamos con el subconjunto de los que son primos relativos con 28. Para introducir la condición de que el máximo común divisor del elemento y 28 valga 1, vamos a emplear la notación λ.

```
(%ixx) UZ28:subset(Z28, lambda([x],is(gcd(x,28)=1)));
(%oxx) {1,3,5,9,11,13,17,19,23,25,27}
```

Donde `lambda([x],is(gcd(x,28)))` es la función que para cada elemento x pregunta si $mcd(x, 28)$ vale 1 (y devuelve `true` si la respuesta es afirmativa, y `false` si la respuesta es negativa).

Y ahora vamos a calcular los inversos de cada uno de ellos.

```
(%ixx) uz28:listify(UZ28);
(%oxx) [1,3,5,9,11,13,17,19,23,25,27]
(%ixx) inversos28:makelist(inv_mod(uz28[i],28),i,1,length(uz28));
(%oxx) [1,19,17,25,23,13,15,5,3,11,9,27]
```

Y vemos que son exactamente los mismos elementos, pero cambiando el orden.

```
(%ixx) is(setify(inversos28)=UZ28);
(%oxx) true
```

Si volvemos a hacer los inversos a la lista de inversos, vemos que nos sale la lista inicial

```
(%ixx) makelist(inv_mod(inversos28[i],28),i,1,length(uz28));
(%oxx) [1,3,5,9,11,13,17,19,23,25,27]
```

Lo que nos dice que el inverso del inverso de un elemento es el propio elemento.

Con lo visto, podemos definir una función que para cada número natural n nos calcule el conjunto $U(\mathbb{Z}_n)$.

```
(%ixx) UZ(n):=subset(setify(makelist(i,i,1,n-1)),lambda([x],is(gcd(x,n)=1)))$
(%ixx) UZ(28);
(%oxx) {1,3,5,9,11,13,17,19,23,25,27}
(%ixx) UZ(30);
(%oxx) {1,7,11,13,17,19,23,29}
```

La función φ de Euler, aplicada a un número natural n, nos calcula el número de elementos menores que n que son primos relativos con n. Maxima emplea para ello el comando `totient`

```
(%ixx) totient(28); totient(30);
```

```
(%oxx) 12
(%oxx) 8
```
que son justamente los cardinales de los conjunto $U(\mathbb{Z}_{28})$ y $U(\mathbb{Z}_{30})$ respectivamente.
```
(%ixx) is(totient(500)=cardinality(UZ(500)));
(%oxx) true
```
Sabemos, por el teorema de Fermat, que si $mcd(a,m) = 1$ entonces $a^{\varphi(m)} \equiv 1 \pmod{m}$. Vamos a hacer algunas comprobaciones con Maxima.
```
(%ixx) makelist(power_mod(uz28[i],12,28),i,1,length(uz28));
(%oxx) [1,1,1,1,1,1,1,1,1,1,1,1]
```
Es decir, hemos elevado cada elemento de $U(\mathbb{Z}_{28})$ a 12, y en todos los casos nos ha salido 1.
Vamos a calcular la décimosegunda potencia de todos los elementos de $\mathbb{Z}_{28}$ (no sólo de las unidades).
```
(%ixx) makelist(power_mod(i,12,28),i,0,27);
(%oxx) [0,1,8,1,8,1,8,21,8,1,8,1,8,1,0,1,8,1,8,1,8,21,8,1,8,1,8,1]
```
Y vemos que sale 1 únicamente en los lugares que se corresponden con las unidades.
Hacemos lo mismo con otro número, por ejemplo, 31.
```
(%ixx) makelist(power_mod(i,30,31),i,0,30);
(%oxx) [0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
```
Lo cual era de esperar, pues 31 es un número primo, luego todos los elementos, salvo el cero, son unidades en $\mathbb{Z}_{31}$.
```
(%ixx) makelist(power_mod(i,34,35),i,0,34);
(%oxx) [0,1,9,4,11,30,1,14,29,16,25,11,9,29,21,15,16,4,4,16,15,21,29,9,11,25,16,29,14,1,
30,11,4,9,1]
```
Al no ser 35 un número primo, nos han salido muchos valores diferentes.
```
(%ixx) cardinality(setify(%));
(%oxx) 12
```
Por ejemplo, si hacemos
```
(%ixx) cardinality(setify(makelist(power_mod(i,756,757),i,1,756)));
(%oxx) 1
```
lo que nos dice que 757 es un número primo, mientras que
```
(%ixx) cardinality(setify(makelist(power_mod(i,656,657),i,1,656)));
(%oxx) 40
```
nos dice que 657 no es un número primo. De hecho, $657 = 3^2 \cdot 73$. El comando `ifactors` nos da la factorización de un número como producto de números primos.
```
(%ixx) ifactors(657);
(%oxx) [[3,2],[73,1]]
```
Lo que significa que 657 es el producto del primo 3, con exponente 2, y el primo 73 con exponente 1.

Ejercicio.

Dado un elemento $a \in U(\mathbb{Z}_n)$, el orden de a es el menor exponente positivo al que hay que elevar a para que de 1. Este número coincide con el número de potencias distintas de a.

- Define una función `orden` que dados dos números a y n calcule el orden de a en $\mathbb{Z}_n$ (esta función no tiene que comprobar si a es o no una unidad en $\mathbb{Z}_n$).
- Elige al azar un número entre 300 y 500, y calcula el siguiente número primo (esto puedes hacerlo con el comando `next_prime`). Llama a este número p. Construye un conjunto que contenga a todos los elementos de la forma $[a,b]$ donde a toma los valores desde 1 hasta $p-1$, y b es el orden de a en $\mathbb{Z}_p$.
- Un elemento $a \in \mathbb{Z}_p$ se dice primitivo si las diferentes potencias de a dan lugar a todos los elementos de $\mathbb{Z}_p$ salvo el cero. Calcula un elemento primitivo de $\mathbb{Z}_{359}$. Toma un número primo p de cuatro cifras, y calcula un elemento primitivo de $\mathbb{Z}_p$.

Se tiene que un elemento $a \in \mathbb{Z}_p$ es primitivo si, y sólo si, $a^m \neq 1$ para cualquier m divisor maximal de $p-1$. Usando esto, define una función `primitivo`, que aplicada a dos números a y p devuelva `true` o `false` según a sea un elemento primitivo en $\mathbb{Z}_p$ o no lo sea (no es necesario comprobar si p es un número primo. Puedes usar la función `div_max` definida anteriormente).

3.3. Sistemas de congruencias.

Vamos a ver aquí como resolver sistemas de congruencias lineales con Maxima. Nos planteamos, en primer lugar, cómo resolver una congruencia de la forma

$$ax \equiv b(\text{mód } m)$$

Sabemos que esa congruencia tiene solución si, y sólo si, $mcd(a, m)$ es un divisor de b. En caso afirmativo, si d es el máximo común divisor de a y m, la congruencia anterior es equivalente a

$$a'x \equiv b'(\text{mód } m') \quad \text{donde } a' = \frac{a}{d},\ b' = \frac{b}{d} \ \text{ y } m' = \frac{m}{d}.$$

Y ahora, como $mcd(a', m') = 1$ podemos calcular el inverso de a' módulo m'. Si llamamos a este inverso u, la congruencia es equivalente a

$$x \equiv u \cdot b'(\text{mód } m')$$

cuyas soluciones son todas de la forma $x = ub' + k \cdot m'$, con $k \in \mathbb{Z}$.
Entonces, dada una congruencia $ax \equiv b(\text{mód } m)$, lo que tenemos que hacer es, en primer lugar, estudiar si tiene o no solución, y en caso afirmativo, encontrar los números $u \cdot b'$ y m', que nos dicen como es la solución de la congruencia. Vamos a hacer un ejemplo con Maxima. Tomamos la congruencia $963x \equiv 291(\text{mód } 1578)$.
Comprobamos si tiene solución:

```
(%ixx) is(mod(291,gcd(963,1578))=0);
(%oxx) true
```

Como sí tiene solución, dividimos por 3, luego nos queda la congruencia $321x \equiv 97(\text{mód } 526)$. Calculamos el inverso de 321 en $\mathbb{Z}_{526}$.

```
(%ixx) inv_mod(321,526);
(%oxx) 195
```

Por tanto, la congruencia es equivalente a $x \equiv 97 \cdot 195(\text{mód } 526)$. Calculamos $97 \cdot 195$ módulo 526.

```
(%ixx) mod(97*195,526);
(%oxx) 505
```

Luego todas las soluciones son de la forma $x = 505 + 526 \cdot k$, con k un número entero.
A continuación vamos a definir una función, `cong`, que nos va a dar la solución de una congruencia como la que hemos resuelto, de forma que si introducimos `cong(963,291,1578)` nos devuelve la lista `[505,526]`, lo que significa que la solución es $x = 505 + 526 \cdot k;\ k \in \mathbb{Z}$.
Nuestra función, lo primero que debe hacer es ver si tiene o no solución. Es decir, deberá ser de la forma

```
cong(a,b,m):=if mod(b,gcd(a,m))=0 then xxxxxx else "No tiene solución"
```

Y ahora debemos ver que tiene que hacer para que nos resuelva la congruencia en caso de que tenga solución. Si analizamos los pasos que hemos seguido, vemos que la primera parte de la solución es $\frac{b}{mcd(a,m)} \cdot \left(\frac{a}{mcd(a,m)}\right)^{-1}$, donde el inverso está tomado módulo $\frac{m}{mcd(a,m)}$, pero esto último habría que reducirlo módulo $\frac{m}{d}$.. La segunda parte es $\frac{m}{d}$.
Como vemos que $mcd(a, m)$ aparece muchas veces, para que no lo calcule cada vez que lo llamamos, lo vamos a guardar en una variable d.
Tendríamos entonces:

```
(%ixx) cong(a,b,m):=block(d:gcd(a,m),
                    if mod(b,d)=0
                       then [mod(b/d*inv_mod(a/d,m/d),m/d), m/d]
                       else "No tiene solución"
                     )$
```

Vemos que también calcula varias veces $\frac{m}{d}$, así que podríamos guardarlo en otra variable. Lo que hemos escrito antes, podemos hacerlo en una sola línea. El hacerlo así es para que se vea un poco más claro.

Ahora podemos decirle que nos resuelva la anterior congruencia.

```
(%ixx) cong(963,291,1578);
(%oxx) [505,526]
```

Mientras que si cambiamos 291 por 290 por ejemplo,

```
(%ixx) cong(963,290,1578);
(%oxx) No tiene solución
```

Ahora lo que vamos a hacer es resolver un sistema de congruencias. Por ejemplo, tomamos el sistema

$$\left\{\begin{array}{rcrl} 675x & \equiv & 485 & \text{mód } 1085 \\ 1211x & \equiv & 2156 & \text{mód } 2247 \end{array}\right.$$

Para resolverlo, seguimos los siguientes pasos:

1. Resolvemos la primera congruencia.

```
(%ixx) cong(675,485,1085);
(%oxx) [192,217]
```

Es decir, la solución es $x = 192 + 217 \cdot k$.

2. Sustituimos en la segunda congruencia, y tenemos $1211(192 + 217k) \equiv 2156(\text{mód } 2247)$, lo que nos da una congruencia en la que la incógnita es k.

3. Agrupamos los términos:

```
(%ixx) 1211*217; 2156-1211*192;
(%oxx) 262787
(%oxx) -230356
```

4. Luego tenemos que resolver la congruencia $262787k \equiv -230356(\text{mód } 2247)$.

```
(%ixx) cong(262787,-230356,2247);
(%oxx) [211,321]
```

Por tanto, la solución es $k = 211 + 321 \cdot k'$.

5. Sustituimos en la solución de la primera congruencia $x = 192 + 217 \cdot (211 + 321 \cdot k')$.

```
(%ixx) [192+217*211,217*321];
(%oxx) [45979,69657]
```

Y la solución del sistema es $x = 45979 + 69657 \cdot k'$.

Ejercicio.

Lo que vamos a hacer ahora es definir una función `Cong`. Esta función va a tener 4 argumentos: tres números y una lista de longitud 2. Supongamos que introducimos `Cong(a,b,m,l)`, donde l es la lista `[l1,l2]`. Entonces, nos debe devolver la solución de la congruencia $ax \equiv b(\text{mód } m)$, donde $x = l1 + l2 \cdot k$.

Así, por ejemplo, si introducimos `Cong(1211,2156,2247,[192,217])` nos devolvería `[45979,69657]`. Esta misma respuesta debe dar si introducimos `Cong(1211,2156,2247,cong(675,485,1085))`.

De esta forma, vemos como podemos resolver sistemas de dos o más congruencias. Caso de que el sistema no tenga solución, nos devolverá un mensaje de error.

Por ejemplo, si quisiéramos resolver el sistema

$$\left\{\begin{array}{rcrr} x & \equiv & 30 & \text{mód } 100 \\ x & \equiv & 25 & \text{mód } 99 \\ x & \equiv & 13 & \text{mód } 97 \end{array}\right.$$

podemos escribir

```
(%ixx) Cong(1,30,100,Cong(1,25,99,cong(1,13,97)));
```

y lo que nos debe devolver es

```
(%oxx) [316330,960300]
```

lo que significa que la solución del sistema es $x = 316330 + 960300 \cdot k$.

Caso de que el sistema no tenga solución, nos dará un mensaje de error

```
(%ixx) Cong(1,2,6,cong(1,6,9));
MARRAY-TYPE-UNKNOWN: array type "No tiene solución" not recognized.
#0: Cong(a=1,b=2,m=6,l=[6,9])
 -- an error. To debug this try: debugmode(true);
```

Para evitar esto, podemos decir en la función `cong` que cuando no tenga solución nos devuelva `[0,0]`, es decir,

```
(%ixx) cong(a,b,m):=block(d:gcd(a,m),
```

```
                    if mod(b,d)=0
                       then [mod(b/d*inv_mod(a/d,m/d),m/d), m/d]
                       else [0,0]
                     )$
```

Y ahora, al ejecutar la instrucción anterior

```
(%ixx) Cong(1,2,6,cong(1,6,9));
```

nos devuelve [6,0]. El hecho de que nos haya dado 0 en la segunda parte significa que no tiene solución.

Ejercicio:

Resuelve los siguientes sistemas de congruencias:

$$\left\{\begin{array}{rcrr} 5679523045x & \equiv & 238165236 & \text{mód } 291342571 \\ 57892345987x & \equiv & 231293175 & \text{mód } 714897237 \\ 74202349823x & \equiv & 2813407650 & \text{mód } 57492734832 \\ 758298734023x & \equiv & 47366104086 & \text{mód } 65929875234 \end{array}\right.$$

$$\left\{\begin{array}{rcrr} 8342347923x & \equiv & 436782834 & \text{mód } 291342571 \\ 47234987239x & \equiv & 47392348723 & \text{mód } 714897237 \\ 23479234983x & \equiv & 83239487973 & \text{mód } 57492734832 \\ 4322349823492x & \equiv & 23209872349 & \text{mód } 65929875234 \end{array}\right.$$