Taller de Álgebra I

Clase 10 - Sistemas lineales de congruencia

Primer cuatrimestre 2022

Objetivo:

El objetivo principal de esta clase es obtener un código que permita resolver sistemas lineales de ecuaciones de congruencia sobre los enteros del siguiente tipo:

$$\begin{cases} a_1 \cdot X & \equiv & b_1 & (\mathrm{mod}\ m_1) \\ a_2 \cdot X & \equiv & b_2 & (\mathrm{mod}\ m_2) \\ & \vdots & & \\ a_\ell \cdot X & \equiv & b_\ell & (\mathrm{mod}\ m_\ell) \end{cases}$$

implementando los métodos aprendidos en Álgebra I (suponemos siempre $a_1, a_2, \ldots, a_\ell$, $b_1, b_2, \ldots, b_\ell \in \mathbb{Z}$, $m_1, m_2 \ldots, m_\ell \in \mathbb{N}$).

¿Qué entendemos por resolver estos sistemas?

Resolver es describir de alguna manera preacordada todas las soluciones.

El caso más simple: sistemas formados por una ecuación sola

Consideramos la ecuación:

$$a \cdot X \quad \equiv \quad b \pmod{m}.$$

Vale lo siguiente:

Sea $d = (a : m)$. Si d no divide a b, la ecuación no tiene solución. Si d divide a b, la ecuación es equivalente a

$$\frac{a}{d} \cdot X \quad \equiv \quad \frac{b}{d} \quad \left(\mathrm{mod}\ \frac{m}{d}\right).$$

La última ecuación tiene la propiedad adicional (con respecto a la primera) de que $\frac{a}{d}$ y $\frac{m}{d}$ son coprimos.

Consideramos ahora la ecuación

$$a \cdot X \equiv b \pmod{m}$$

con a y m coprimos. Entonces existen $s, t \in \mathbb{Z}$ tales que

$$s \cdot a + t \cdot m = 1.$$

Vale lo siguiente:

La ecuación es equivalente a la ecuación

$$X \equiv s \cdot b \pmod{m}.$$

Además, si r es el resto de $s \cdot b$ en la división por m, la ecuación anterior es equivalente a

$$X \equiv r \pmod{m}.$$

¿Como representamos estos datos en Haskell?

- Ecuación --→ `(Int, Int, Int)`
- $a \cdot X \equiv b \pmod{m}$ --→ (a, b, m)

- Solución (clase de congruencia) --→ `(Int, Int)`
- $X \equiv r \pmod{m}$ --→ (r, m) (tomando siempre $0 \leq r < m$).
- ¿Qué hacemos cuando una ecuación no tiene solución? --→ `undefined` , $(0, 0)$, ...

Ejercicio

Programar la función `solucionEc :: (Int, Int, Int) -> (Int, Int)` que resuelve una ecuación lineal de congruencia.

Sistemas de ecuaciones

Repitiendo lo que hicimos anteriormente para cada ecuacion por separado, tenemos que cada sistema del tipo

$$\begin{cases} a_1 \cdot X & \equiv & b_1 & (\mathrm{mod}\ m_1) \\ a_2 \cdot X & \equiv & b_2 & (\mathrm{mod}\ m_2) \\ & \vdots & & \\ a_\ell \cdot X & \equiv & b_\ell & (\mathrm{mod}\ m_\ell) \end{cases}$$

es equivalente a un sistema simplificado del tipo

$$\begin{cases} X & \equiv & r_1 & (\mathrm{mod}\ m'_1) \\ X & \equiv & r_2 & (\mathrm{mod}\ m'_2) \\ & \vdots & & \\ X & \equiv & r_\ell & (\mathrm{mod}\ m'_\ell) \end{cases}$$

¿Como representamos estos datos en Haskell? II

- Sistemas de ecuaciones `-->` `[(Int, Int, Int)]`
- Sistemas simplificados de ecuaciones `-->` `[(Int, Int)]`

Ejercicio

Programar la función `sistemaSimplifEquiv :: [(Int, Int, Int)] -> [(Int, Int)]`, que dado un sistema lineal de ecuaciones de congruencia, devuelve un sistema simplificado equivalente, como se explicó en la diapositiva anterior.

Sistemas simplificados

Consideramos ahora sistemas el tipo

$$\begin{cases} X & \equiv & r_1 & (\mathrm{mod}\ m_1) \\ X & \equiv & r_2 & (\mathrm{mod}\ m_2) \\ & \vdots & & \\ X & \equiv & r_n & (\mathrm{mod}\ m_n) \end{cases}$$

Lamentablemente, no podemos contar con que $m_1, m_2, \ldots, m_n$ sean coprimos dos a dos, así que por ahora no podemos aplicar directamente el ultrapoderoso Teorema Chino del Resto... pero tratemos de ponernos en esa sitaución.

Primos malos

Decimos que un primo p es malo para el sistema

$$\begin{cases} X & \equiv & r_1 & (\mathrm{mod}\ m_1) \\ X & \equiv & r_2 & (\mathrm{mod}\ m_2) \\ & \vdots & & \\ X & \equiv & r_\ell & (\mathrm{mod}\ m_\ell) \end{cases}$$

si p divide al menos a dos de los módulos $m_1, m_2, \ldots, m_\ell$.

Si $m_1, m_2, \ldots, m_\ell$ no son coprimos dos a dos, es porque hay al menos un primo malo para el sistema.

Lo primero que vamos a hacer es encontrar todos los primos malos para un sistema simplificado.

Ejercicio

Programar la función `todosLosPrimosMalos :: [(Int, Int)] -> [Int]` que devuelve una lista con todos los primos malos para un sistema simplificado.

Solucionando el problema con respecto a un primo malo

Si encontramos un primo malo, lo que hacemos es, en algún sentido, extraer ese primo de todas las ecuaciones para encontrar un sistema equivalente.

Supongamos que p es un primo que divide a m, y la mayor potencia de p que divide a m es k.

Por el Teorema Chino del Resto, vale lo siguiente:

La ecuación

$$X \equiv r \pmod{m}$$

es equivalente al sistema

$$\begin{cases} X \equiv r \pmod{p^k} \\ X \equiv r \pmod{m/p^k} \end{cases}$$

Solucionando el problema con respecto a un primo malo

- En la primera ecuación, puede ocurrir que $r \geq p^k$; por lo tanto, reemplazamos r por su resto en la división por p^k. Procedemos similarmente con la segunda ecuación.

- En el caso en que m es una potencia de p (o sea $m = p^k$), la última ecuación queda

$$X \equiv 0 \pmod{1}$$

y podemos eliminarla.

Ejemplo: p primo malo

Supongamos que p divide a m_1, a m_3 y a m_5 pero m_1 y m_5 no son potencia de p y m_3 sí lo es, y que p no divide a m_2 ni a m_4. Entonces:

$$\begin{cases} X \equiv r_1 \pmod{m_1} \\ X \equiv r_2 \pmod{m_2} \\ X \equiv r_3 \pmod{m_3} \\ X \equiv r_4 \pmod{m_4} \\ X \equiv r_5 \pmod{m_5} \end{cases} \dashrightarrow \begin{cases} X \equiv r_1 \pmod{p^{k_1}} \\ X \equiv r_1 \pmod{m_1/p^{k_1}} \\ X \equiv r_2 \pmod{m_2} \\ X \equiv r_3 \pmod{p^{k_3}} \\ X \equiv r_4 \pmod{m_4} \\ X \equiv r_5 \pmod{p^{k_5}} \\ X \equiv r_5 \pmod{m_5/p^{k_5}} \end{cases}$$

1ra parte:

$$\begin{cases} X \equiv r_1 \pmod{p^{k_1}} \\ X \equiv r_3 \pmod{p^{k_3}} \\ X \equiv r_5 \pmod{p^{k_5}} \end{cases}$$

$\dashrightarrow$

2da parte:

$$\begin{cases} X \equiv r_1 \pmod{m_1/p^{k_1}} \\ X \equiv r_2 \pmod{m_2} \\ X \equiv r_4 \pmod{m_4} \\ X \equiv r_5 \pmod{m_5/p^{k_5}} \end{cases}$$

Sistemas en los que todos los módulos son potencias de un primo p

Nos detenemos ahora en la primera parte del sistema, que tiene todos sus módulos dados por potencias de p.

Podemos resumir estas ecuaciones en una única ecuación. Para eso, vamos mirando de a dos ecuaciones a la vez y eliminando en cada paso una de ellas.

Vale lo siguiente:

Consideramos las ecuaciones

$$\begin{cases} X & \equiv & r_1 & (\mathrm{mod}\ p^{k_1}) \\ X & \equiv & r_2 & (\mathrm{mod}\ p^{k_2}) \end{cases}$$

y supongamos $k_1 \leq k_2$.

Si $r_1 \equiv r_2 \mod p^{k_1}$, entonces todas las soluciones de la segunda ecuación son soluciones de la primera ecuación (y por lo tanto la primera ecuación puede eliminarse).
Si $r_1 \not\equiv r_2 \mod p^{k_1}$, entonces ambas ecuaciones no tienen ninguna solución en común.

Ejercicio

Programar la funcion `solucSistemaPotenciasPrimo :: [(Int, Int)] -> (Int, Int)` que resuelve un sistema en el que todos los módulos son potencias de un mismo primo.

¡Esta es la parte mas difícil de la clase!

Ahora la idea es ir mirando los primos malos de a uno a la vez.

Para cada primo malo p, hacemos lo siguiente:

- Desdoblamos las ecuaciones en las que el módulo es múltiplo del p.

- Usamos la funcion `solucSistemaPotenciasPrimo` para o bien resumir en una sola ecuación todas las ecuaciones cuyo módulo es una potencia de p o bien decidir ya que el sistema no tiene solución.

Luego pasamos al primo siguiente, mirando solo la segunda parte del sistema (en la que ya sabemos que todos los módulos son coprimos con p).

Continuación del ejemplo: p primo malo

1ra parte:

$$\begin{cases} X \equiv r_1 \pmod{p^{k_1}} \\ X \equiv r_3 \pmod{p^{k_3}} \\ X \equiv r_5 \pmod{p^{k_5}} \end{cases}$$

2da parte:

$$\begin{cases} X \equiv r_1 \pmod{m_1/p^{k_1}} \\ X \equiv r_2 \pmod{m_2} \\ X \equiv r_4 \pmod{m_4} \\ X \equiv r_5 \pmod{m_5/p^{k_5}} \end{cases} \dashrightarrow \begin{cases} X \equiv r_{\ldots} \pmod{p^{k_{\cdots}}} \\ X \equiv r_1 \pmod{m_1/p^{k_1}} \\ X \equiv r_2 \pmod{m_2} \\ X \equiv r_4 \pmod{m_4} \\ X \equiv r_5 \pmod{m_5/p^{k_5}} \end{cases}$$

Ejercicio

Programar la funcion `desdoblarSistemaEnFcionPrimo :: [(Int, Int)] -> Int -> ([(Int, Int)], [(Int, Int)])` que dado un sistema y un primo p, devuelve los dos sistemas que surgen al desdoblar cada ecuación del sistema original según el primo p como se explicó anteriormente.

Consejo: Utilizar la función `quePotenciaLoDivide :: Int -> Int -> Int` de la Clase Adicional (si no la hiciste, programala ahora!!!), que dados m y p, calcula la mayor potencia de p que divide a m.

Ejercicio

Programar la funcion `sistemaEquivSinPrimosMalos :: [(Int, Int)] -> [(Int, Int)]` que dado un sistema, devuelve un sistema equivalente sin primos malos.

Pasó la tormenta...

Ahora sí, como le gusta a los docentes de Álgebra, tenemos un sistema lineal de ecuaciones cuyos módulos son coprimos dos a dos, y podemos usar el Teorema Chino del Resto: un sistema así siempre tiene solución y está dada por una clase de congruencia módulo el producto de los módulos del sistema original.

Para obtener esa solución, vamos juntando las ecuaciones de a dos, hasta llegar a tener una sola ecuación:

Vale que:

Si $(m_1 : m_2) = 1$ y $1 = s \cdot m_1 + t \cdot m_2$, entonces el sistema

$$\begin{cases} X \equiv r_1 \pmod{m_1} \\ X \equiv r_2 \pmod{m_2} \end{cases}$$

es equivalente a la ecuación

$$X \equiv r \pmod{m_1 \cdot m_2}$$

con r igual al resto de $r_1 \cdot t \cdot m_2 + r_2 \cdot s \cdot m_1$ en la división por $m_1 \cdot m_2$.

Ejercicio

Programar la funcion `solucSistemaModCoprimos :: [(Int, Int)] -> (Int, Int)` que resuelve un sistema simplificado en el que los módulos son coprimos dos a dos.

Ya hicimos toda la jugada, solamente falta hacer el gol.

Ejercicio

Programar la funcion `solucSistema :: [(Int, Int, Int)] -> (Int, Int)` que resuelve un sistema general.

Ejercicios

1. Programar la función `cadaEcTieneSoluc :: [(Int, Int, Int)] -> Bool` que, dado un sistema general, decide si cada una de sus ecuaciones vista independientemente de las otras, tiene solución.
2. Programar la función `tieneSolucionSimplif :: [(Int, Int)] -> Bool` que, dado un sistema simplificado, decide si tiene solución.
3. Programar la función `tieneSolucion :: [(Int, Int, Int)] -> Bool` que, dado un sistema general, decide si tiene solución.
4. Programar la función `dirichlet :: Int -> Int -> Int` que dados dos números coprimos r y m con $1 \leq r < m$, encuentra un número primo en la clase de congruencia $X \equiv r \pmod{m}$.