Taller de análisis y diseño de algoritmos

Algoritmos para problemas aritméticos

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

Trimestre 2021 Invierno

Contenido

Basado en *Primes and Programming. Peter Giblin. Cambridge University Press. 1993.*

Enteros

Los enteros son $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$. Si queremos representar un entero x, entonces usaremos `signed char` si $-2^7 \leq x < 2^7$, `short` si $-2^{15} \leq x < 2^{15}$, `int` si $-2^{31} \leq x < 2^{31}$ o `long long` si $-2^{-63} \leq x < 2^{63}$. Nota: En gcc también hay enteros de 128 bits `__int128`.

Naturales

Los naturales son $\mathbb{N} = \{0, 1, 2, \ldots\}$. Si queremos representar un natural x, entonces usaremos `unsigned char` si $0 \leq x < 2^8$, `unsigned short` si $0 \leq x < 2^{16}$, `unsigned int` si $0 \leq x < 2^{32}$ o `unsigned long long` si $0 \leq x < 2^{64}$.

Notas

Vale la pena tener en mente que $10^3 \approx 2^{10}$ para poder estimar, por ejemplo, que el `int` más grande vale $\approx 2 \times 10^9$, mientras que el `unsigned int` más grande vale $\approx 4 \times 10^9$.

Divisibilidad

Diremos que un entero a divide a un entero b si $b = ac$ para algún entero c. Esto se escribe $a|b$ y también se lee como a es un divisor de b o como a es un factor de b. Observa que si $a = 0$, entonces $b = 0$. Por otro lado, si $a \neq 0$, entonces tanto a como $-a$ dividen a b, por lo que podemos concentrarnos en los divisores positivos. En este caso, saber si $a|b$ se puede hacer como `b%a==0`.

Primos y compuestos

Un primo es un entero $p > 1$ cuyos únicos divisores positivos son 1 y p. Un compuesto es un entero $b > 1$ tal que $b = ac$ para algunos $a > 1$ y $c > 1$. Observa que el 1 no es ni primo ni compuesto. El conjunto de los primos será $\mathbb{P} = \{2, 3, 5, 7, \ldots\}$. Una pregunta fundamental de la aritmética es: Dado un entero $n > 0$, ¿es n primo?

Primalidad

Un primer algoritmo

Leyendo la definición, n no será primo si $n \leq 1$, ni tampoco lo será si tiene algún divisor $1 < a < n$, mientras que sí lo será en cualquier otro caso.

```
bool esprimo(int n) {
  if (n <= 1) return 0;
  for (int a = 2; a < n; a++)
    if (n%a == 0) return 0;
  return 1;
}
```

En el peor de los casos (cuando n sí sea primo), la cantidad de iteraciones será $\Theta(n)$. Esto es bastante malo, pues el tamaño de la entrada (la cantidad de bits de n) es $t \approx \log_2 n$, así que la cantidad de iteraciones es $\Theta(2^t)$, es decir, exponencial. Como ejemplo, si $n = 2^{31} - 1$, esta función se tarda más de 5 segundos en determinar que n sí es primo.

Primalidad

Un segundo algoritmo

Si $n > 1$ es compuesto, entonces tiene algún divisor $a \leq \sqrt{n}$ (si $a > \sqrt{n}$ y $\frac{n}{a} > \sqrt{n}$, entonces $n = a \cdot \frac{n}{a} > n$). Usemos además que el único primo par es el 2.

```
bool esprimo(int n) {
  if (n == 2) return 1;
  if (n <= 1 || n%2 == 0) return 0;
  int raizn = raiz(n);
  for (int a = 3; a <= raizn; a += 2)
    if (n%a == 0) return 0;
  return 1;
}
```

En el peor de los casos la cantidad de iteraciones será $\Theta(\sqrt{n}) = \Theta(2^{t/2})$: sigue siendo exponencial. Se debe implementar `raiz` con cuidado para que sirva en el rango de `int`.

Raíz cuadrada entera

Con el método de Newton

Aunque baste hacer `raizn = ceil(sqrt(n))`, es bueno conocer otros métodos. El método de Newton hace ≤ 19 iteraciones para cualquier `int` no negativo. Si se usa `long long`, entonces son ≤ 35 iteraciones. Se puede adaptar fácilmente para encontrar otras raíces.

```
int raiz(int n) {
  int x0 = n >> 1;
  if (x0 == 0) return n;
  int x1 = (x0 + n/x0) >> 1;
  while (x1 < x0) {
    x0 = x1;
    x1 = (x0+n/x0) >> 1;
  }
  return x0;
}
```

Raíz cuadrada entera

Con el método de bit por bit

Para cualquier `int` no negativo hace 17 iteraciones.

```
int raiz(int n) {
  unsigned int r = 0;
  unsigned int u = 1 << 30;
  while (u > n)
    u >>= 2;
  while (u > 0) {
    if (n >= r+u) {
      n -= r+u;
      r += u << 1;
    }
    r >>= 1;
    u >>= 2;
  }
  return r;
}
```

Teorema (Factorización única en potencias de primos)

Todo entero $n > 1$ tiene exactamente una factorización $n = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ con $k \geq 1$, $p_1 < p_2 < \cdots < p_k$ primos y $n_1, n_2, \ldots, n_k$ enteros positivos.

Teorema (Primos y productos)

1. *Si $p \neq q$ son primos, $p|n$ y $q|n$, entonces $pq|n$.*
2. *Si p es primo y $p|ab$, entonces $p|a$ o $p|b$ (o ambos).*

Teorema (Infinitud de los primos)

1. *Hay una cantidad infinita de primos.*
2. *Hay una cantidad infinita de primos de las formas $4k \pm 1$ (todos excepto 2).*
3. *Hay una cantidad infinita de primos de las formas $6k \pm 1$ (todos excepto 2 y 3).*

Sean a y b dos enteros positivos.

Divisor común

Un divisor común d de a y b es un entero tal que $d|a$ y $d|b$. Si $p_1, \ldots, p_k$ son primos distintos, $a = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, $b = p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}$ y $a_i, b_i \geq 0$ para toda $1 \leq i \leq k$, entonces $d|a$ y $d|b$ si y solo si $d = p_1^{d_1} p_2^{d_2} \cdots p_k^{d_k}$ con $d_i \leq \min(a_i, b_i)$ para toda $1 \leq i \leq k$.

Máximo común divisor

El entero d tal que $d_i = \min(a_i, b_i)$ para toda $1 \leq i \leq k$ se llama el máximo común divisor de a y b. Se denota por $d = \mathrm{mcd}(a, b)$.

Primos relativos

Si $\mathrm{mcd}(a, b) = 1$, decimos que a y b son primos relativos.

Teorema (Máximo común divisor)

1. *Si p es primo, entonces* $\text{mcd}(p, a) = p$ *cuando* $p|a$ *y* $\text{mcd}(p, a) = 1$ *en otro caso.*
2. *Si p es primo y no divide a a, entonces* $\text{mcd}(p^n, a) = 1$ *para toda* $n \geq 1$
3. *Si* $\text{mcd}(a, b) = d$*, entonces* $\text{mcd}(a/d, b/d) = 1$.
4. *Si* $a|bc$ *y* $\text{mcd}(a, b) = 1$*, entonces* $a|c$.
5. *Si p es primo,* $p^n|ab$ *y p no divide a* $\text{mcd}(a, b)$*, entonces* $p^n|a$ *o* $p^n|b$.

Teorema (Rejilla rectangular)

Considere un rectángulo de lados enteros a y b dividido en una rejilla de ab cuadrados de lado 1*. Entonces cualquier diagonal del rectángulo (a) pasa por* $1 + \text{mcd}(a, b)$ *vértices de los cuadrados y (b) cruza* $a + b - \text{mcd}(a, b)$ *cuadrados.*

Sean a y b dos enteros positivos.

Multiplo común

Un múltiplo común d de a y b es un entero tal que $a|d$ y $b|d$. Si $p_1, \ldots, p_k$ son primos distintos, $a = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, $b = p_1^{b_1} p_2^{b_2} \cdots p_k^{b_k}$ y $a_i, b_i \geq 0$ para toda $1 \leq i \leq k$, entonces $a|d$ y $b|d$ si y solo si $d = p_1^{d_1} p_2^{d_2} \cdots p_k^{d_k}$ con $d_i \geq \max(a_i, b_i)$ para toda $1 \leq i \leq k$.

Mínimo común múltiplo

El entero d tal que $d_i = \max(a_i, b_i)$ para toda $1 \leq i \leq k$ se llama el mínimo común múltiplo de a y b. Se denota por $d = \mathrm{mcm}(a, b)$.

Teorema

Para toda a y b distintas de 0, $\mathrm{mcd}(a, b)\,\mathrm{mcm}(a, b) = |ab|$.

El algoritmo de Euclides (300 AE)

De acuerdo a la definición, para calcular $\mathrm{mcd}(a, b)$ se requiere primero factorizar a y b en primos. Como esto es muy lento, resulta afortunado que posiblemente el primer algoritmo de la historia nos dice cómo hacerlo usando sólo los residuos de la división.

Cociente y residuo de la división

Si $a \geq 0$ y $b > 0$, entonces existen enteros únicos q y r tales que $a = bq + r$ con $b > r \geq 0$. Si `a` y `b` son `int` positivos, entonces el cociente `q` y el residuo `r` se pueden calcular como `q = a/b` y `r = a%b`, respectivamente.

El algoritmo de Euclides

Una versión recursiva está basada en que, por un lado, $\mathrm{mcd}(a, 0) = a$ para toda $a > 0$ y, por otro lado, $\mathrm{mcd}(a, b) = \mathrm{mcd}(b, r)$ donde r es el residuo de la división descrito arriba. Note que el segundo parámetro decrece y, por lo tanto, eventualmente llega a 0.

Cualquiera de estas dos implementaciones hace $O(\log b)$ pasos.

Recursiva

```
int mcdr(int a, int b) {
  if (b == 0) return a;
  return mcdr(b, a%b);
}
```

Iterativa

```
int mcd(int a, int b) {
  int r;
  while (b != 0)
    r = a%b, a = b, b = r;
  return a;
}
```

Sean a y b dos enteros. Cualquier entero $c = as + bt$ con s y t enteros se llama una combinación lineal de a y b.

Teorema (Divisores y combinaciones lineales)

Si $d|a$ y $d|b$, entonces $d|c$. En particular $\operatorname{mcd}(a, b)|c$.

Teorema (Máximo común divisor como combinación lineal)

Existen s, t tales que $\operatorname{mcd}(a, b) = as + bt$. En particular, a y b son primos relativos si y sólo si existen s, t tales que $as + bt = 1$.

Teorema (Ecuación diofantina)

La ecuación $as + bt = c$ tiene soluciones enteras s, t si y sólo si $\operatorname{mcd}(a, b)|c$.

Cálculo de la combinación lineal

Lo siguiente se obtiene de analizar con cuidado el algoritmo de Euclides.

```
void combinacion(int a, int b, int *s, int *t) {
  int s1=0, t1=1, s2=1, s1=0, q, r;
  while (b > 0) {
    q = a/b;
    r = a%b;
    *s = s2 - q*s1;
    *t = t2 - q*t1;
    a = b, b = r;
    s2 = s1, s1 = *s;
    t2 = t1, t1 = *t;
  }
}
```

Ecuaciones diofantinas

Como $ax + by = c$ tiene soluciones enteras x, y si y sólo si $\operatorname{mcd}(a, b) | c$, entonces lo primero que se hace es dividir los tres coeficientes a, b, c entre $d = \operatorname{mcd}(a, b)$. Con eso, ahora podemos suponer que a y b son primos relativos.

Ahora encuentra una combinación lineal $as + bt = 1$. Entonces $x_0 = cs$, $y_0 = ct$ es una solución a la ecuación $ax + by = c$. El conjunto completo de soluciones enteras estará dado por $x_k = x_0 + kb$, $y_k = y_0 - ka$ para toda k entera.

Si sólo se quieren las soluciones con $x, y \geq 0$, entonces basta considerar el rango de k correspondiente. En particular, si $a, b > 0$, entonces $x_k \geq 0$ implica $k \geq -\frac{x_0}{b}$ y $y_k \geq 0$ implica $\frac{y_0}{a} \geq k$. En otras palabras, el rango es $\frac{y_0}{a} \geq k \geq -\frac{x_0}{b}$.

Factorizaciones especiales

Factoriales y binomiales

Recuerda que el factorial de n es el producto $n! = 1 \cdot 2 \cdots n$. ¿Cuál es la máxima potencia 2^{n_2} que divide al factorial de n? Podemos observar que uno de cada 2 factores es divisible entre 2, uno de cada 2^2 factores es divisible entre 2^2, etc. Por lo tanto

$$n_2 = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{2^2} \right\rfloor + \left\lfloor \frac{n}{2^3} \right\rfloor + \cdots$$

De la misma manera, para cualquier primo p, la máxima potencia p^{n_p} que divide a $n!$ es

$$n_p = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \cdots$$

Finalmente, como $c = \binom{n}{m} = \frac{n!}{m!(n-m)!}$, la máxima potencia p^{c_p} que divide a c es

$$c_p = n_p - m_p - (n-m)_p.$$

Contenido

Basado en *Primes and Programming. Peter Giblin. Cambridge University Press. 1993.*

Listando primos usando divisiones

Dividiendo entre los números impares

El 2 es primo. A partir de allí, sólo se revisan los divisores impares de los impares.

```
void listaprimos(int max) {
  int n, q, raizn;
  printf("2\n");
  for (n = 3; n <= max; n += 2) {
    raizn = raiz(n);
    for (q = 3; q <= raizn; q += 2)
      if (n%q == 0)
        break;
    if (q > raizn)
      printf("%d\n", n);
  }
}
```

Listando primos usando divisiones

Dividiendo entre los primos ya encontrados

El 2 y el 3 son primos. A partir de allí se revisan los divisores primos de los impares.

```
void arregloprimos(int max, int p[]) {
  int n = 3, raizn, primo;
  p[0] = 2, p[1] = 3;
  for (int i = 2; i < max; i++) {
    do {
      n += 2;
      raizn = raiz(n);
      primo = 1;
      for (int j = 1; primo && p[j] <= raizn; j++)
        if (n%p[j] == 0)
          primo = 0;
    } while (!primo);
    p[i] = n;
  }
}
```

Listando primos usando divisiones

Aplicaciones

Si ya se tiene un arreglo `p` de `max` primos, entonces:

1. Se puede saber (dividiendo) si cualquier entero menor o igual que el último primo de ese arreglo elevado al cuadrado es primo o no.
2. Se puede saber (usando búsqueda binaria) si cualquier entero menor o igual que el último primo de ese arreglo es primo o no.
3. Se puede saber (usando búsqueda binaria) para cualquier entero menor o igual que el último primo de ese arreglo cuántos primos menores hay.
4. Se puede factorizar (dividiendo) cualquier entero cuyos factores primos menores o iguales que el último primo de ese arreglo.

Sea r un par positivo y sea n un entero positivo. Queremos listar los primos impares p tales que $r < p \leq r + 2n - 1$. Alternativamente, queremos saber cuáles compuestos ij (con $i, j > 1$ e impares) son de la forma $r + 2k - 1$ para alguna $1 \leq k \leq n$.

Para esto, sea $c = r + 2n - 1$. De $r + 1 \leq ij \leq c$ se obtiene $\frac{r+1}{i} \leq j \leq \frac{c}{i}$. El impar más pequeño $\geq \frac{r+1}{i}$ es $s = 2\lfloor \frac{r+i}{2i} \rfloor + 1$. Este es el menor valor posible de j (si $s = 1$ lo reemplazamos por $s = 3$). El rango de valores de i es $3 \leq i \leq m$, donde $m = \lfloor \sqrt{c} \rfloor$.

Para cada i, probamos los impares j de s a $\lfloor \frac{c}{i} \rfloor$, calculamos ij y $k = \frac{1}{2}(ij - r + 1)$. Si $1 \leq k \leq n$, entonces eliminamos k. Las k nunca eliminadas corresponden con primos.

Listando primos usando productos

Primos en un intervalo (implementación)

En este código `a` es un arreglo de tamaño `n+1` y no se usa `a[0]`.

```
void intervalo(int r, int n, bool a[]) {
  for (int k = 1; k <= n; k++)
    a[k] = 1;
  int c = r + 2*n - 1;
  int m = raiz(c);
  for (int i = 3; i <= m; i += 2) {
    int lj = c/i;
    int sj = 2*((r + i)/(2*i)) + 1;
    if (sj < 3) sj = 3;
    for (int j = sj; j <= lj; j += 2)
      a[(i*j - r + 1)/2] = 0;
  }
}
```

Considere todos los enteros del 2 al n. El 2 es primo, pero sus múltiplos no, así que podemos eliminarlos. El primero no eliminado es el 3 que es primo, pero sus múltiplos no, así que podemos eliminarlos. El siguiente no eliminado es el 5 que es primo, pero sus múltiplos no, así que podemos eliminarlos. Si continuamos de esta manera, sólamente quedarán los números primos del 2 al n.

Es obvio que es suficiente llevar a cabo este proceso hasta que se encuentren todos los primos $\leq \sqrt{n}$: Si se está considerando el primo impar p, entonces el primer múltiplo a eliminar será p^2. Más aún, como los múltiplos $p(p+1), p(p+3), p(p+5), \ldots$ son pares, en realidad sólo debemos eliminar los múltiplos $p^2, p(p+2), p(p+4), \ldots$.

Listando primos usando una criba

La criba de Eratóstenes

En este código a es un arreglo de tamaño n+1. Si hacemos **int** a[] y a[i] = i, podemos poner en a[j] un divisor de j con a[j] = p (p será primo si a[p] == p). También podemos ignorar al 2 y concentrarnos en los impares, de modo que a sólamente mida la mitad.

```
void criba(int n, bool a[]) {
  a[0] = a[1] = 0;
  for (int i = 2; i <= n; i++)
    a[i] = 1;
  p = 2;
  while (p <= n) {
    for (int j = p*p; j <= n; j += p)
      a[j] = 0;
    do p++ while (p <= n && a[p] == 0);
  }
}
```

Para x real, definamos la función $\pi(x)$ como la cantidad de primos $\leq x$.

Ejemplo

$\pi(1) = 0$, $\pi(2) = 1$, $\pi(e) = 1$, $\pi(\sqrt{10}) = 3$, $\pi(10) = 4$.

Uno de los resultados más importantes acerca de la distribución de los números primos afirma que la función $\pi(x)$ se puede aproximar cuando $x \to \infty$.

Teorema (Aproximación de $\pi(x)$)

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1.$$

Es decir, $\pi(x) \approx \frac{x}{\ln x}$, o bien, aproximadamente 1 de cada $\ln x$ enteros $\leq x$ es primo.

Contenido

Basado en *Primes and Programming. Peter Giblin. Cambridge University Press. 1993.*

Congruencias y módulos

Sean a, b, m enteros con $m > 0$. La notación de congruencia $a \equiv b \pmod{m}$ significa que $m | a - b$. Dados a, m con $m > 0$, se puede escribir $a = qm + b$ para una única $0 \leq b < m$. A m se le llama el módulo y $a \equiv b \pmod{m}$. Si $a \geq 0$, entonces b se calcula como `a%m`.

Teorema (Propiedades básicas de la congruencia)

1. *La congruencia* $a \equiv b \pmod{1}$ *siempre es cierta.*
2. *La congruencia* $a \equiv a \pmod{m}$ *siempre es cierta.*
3. *Si* $a \equiv b \pmod{m}$ *y* $b \equiv c \pmod{m}$, *entonces* $a \equiv c \pmod{m}$.
4. *Si* $a \equiv b \pmod{m}$ *y* $c \equiv d \pmod{m}$, *entonces* $a + c \equiv b + d \pmod{m}$.
5. *Si* $a \equiv b \pmod{m}$ *y* $c \equiv d \pmod{m}$, *entonces* $ac \equiv bd \pmod{m}$.
6. *Si* $a \equiv b \pmod{m}$, *entonces* $a^k \equiv b^k \pmod{m}$ *para todo entero* $k \geq 0$.
7. *Sea* $c > 0$. *Entonces* $a \equiv b \pmod{m}$ *si y sólo si* $ac \equiv bc \pmod{mc}$.

Teorema (Cancelación de factores)

Si $ac \equiv bc \pmod{m}$ *y* $d = mcd(c, m)$, *entonces* $a \equiv b \pmod{m/d}$. *En particular, si* c *y* m *son primos relativos* ($d = 1$), *entonces* $a \equiv b \pmod{m}$.

Teorema (Congruencia simultánea)

Si $a \equiv b \pmod{m}$ *y* $a \equiv b \pmod{n}$, *entonces* $a \equiv b \pmod{\operatorname{mcm}(m, n)}$.

Teorema (Congruencias simultáneas con módulos primos relativos)

Si $a \equiv b \pmod{m_i}$ *para* $1 \leq i \leq r$, *donde* $\operatorname{mcd}(m_i, m_j) = 1$ *para toda* $i \neq j$, *entonces* $a \equiv b \pmod{m_1 m_2 \cdots m_r}$.

Congruencias y módulos

Propiedades útiles

1. Para todo impar a se cumple $a^2 \equiv 1 \pmod{4}$, $a^2 \equiv 1 \pmod{8}$ y $a^4 \equiv 1 \pmod{16}$.
2. Si p es primo, entonces $x^2 \equiv y^2 \pmod{p}$ implica que $x \equiv \pm y \pmod{p}$.
3. Si $n = 10^k a_k + 10^{k-1} a_{k-1} + \cdots + 10^1 a_1 + a_0$ (es decir, n en decimal), entonces $n \equiv a_k + a_{k-1} + \cdots + a_1 + a_0 \pmod{3}$ y $\pmod{9}$.
4. Bajo la misma suposición, $n \equiv a_0 - a_1 + a_2 - \cdots \pmod{11}$.

Inversos y ecuaciones en congruencias

Sean a, m enteros con $m > 0$. Existe un entero b tal que $ab \equiv 1 \pmod{m}$ si y sólo si a y m son primos relativos. Cuando b existe, es único $\pmod{m}$ y es primo relativo con m. En este caso, a b se le llama el inverso de $a \pmod{m}$ y se denota por $b \equiv a^{-1} \pmod{m}$.

Si uno puede encontrar inversos, entonces uno puede resolver ecuaciones en congruencias de la forma $ax \equiv c \pmod{m}$. Sea $h = \text{mcd}(a, m)$. Si h no divide a c, entonces no hay soluciones. Suponiendo que $h|c$, se puede escribir $a = ha'$, $m = hm'$, $c = hc'$, de donde $a'x \equiv c' \pmod{m'}$ con $\text{mcd}(a', m') = 1$. Ahora sea $b' \equiv a'^{-1} \pmod{m'}$ y multiplique para obtener la única solución $x \equiv b'c' \pmod{m'}$. Las soluciones de la ecuación original serán $x, x + m', x + 2m', \ldots, x + (h-1)m'$.

Ecuaciones simultáneas en congruencias

Si tenemos algunas ecuaciones lineales simultáneas en congruencias, entonces se pueden resolver por sustitución repetida.

Ejemplo (Resolver $6x \equiv 8 \pmod{10}$ y $4x \equiv 7 \pmod{9}$)

La ecuación $6x \equiv 8 \pmod{10}$ implica que $3x \equiv 4 \pmod{5}$. El inverso de $3 \pmod{5}$ es 2 ($2 \cdot 3 = 6 \equiv 1 \pmod{5}$). Por lo tanto $2 \cdot 3x \equiv 2 \cdot 4 \pmod{5}$, es decir $6x \equiv 8 \pmod{5}$, o bien $x \equiv 3 \pmod{5}$. Así, podemos escribir $x = 5k + 3$ para k entera.

Sustituyendo en la ecuación $4x \equiv 7 \pmod{9}$ obtenemos $20k + 12 \equiv 7 \pmod{9}$, es decir $2k \equiv 4 \pmod{9}$. El inverso de $2 \pmod{9}$ es 5 ($5 \cdot 2 = 10 \equiv 1 \pmod{9}$). Por lo tanto $5 \cdot 2k \equiv 5 \cdot 4 \pmod{9}$, es decir, $10k \equiv 20 \pmod{9}$, o bien $k \equiv 2 \pmod{9}$. Así, podemos escribir $k = 9j + 2$ para j entera.

Finalmente, $x = 5k + 3 = 5(9j + 2) + 3 = 45j + 13$, y la solución es $x \equiv 13 \pmod{45}$.

Si tenemos ecuaciones simultáneas $a_i x \equiv b_i \pmod{m_i}$ para $1 \leq i \leq r$, entonces cada una de ellas no tiene solución o se puede transformar en una ecuación en la que primero $\operatorname{mcd}(a_i, m_i) = 1$ y luego $a_i = 1$. A partir de allí, hay un caso de especial importancia.

Teorema (Chino del residuo)

Sean $m_1, \ldots, m_r$ enteros positivos y primos relativos a pares (es decir, $\operatorname{mcd}(m_i, m_j) = 1$ cuando $i \neq j$). Entonces el sistema de ecuaciones simultáneas $x \equiv c_i \pmod{m_i}$ para $1 \leq i \leq r$ tiene una solución única $x \pmod{m_1 m_2 \cdots m_r}$.

Además, x se obtiene de la siguiente forma: Sea $M = m_1 m_2 \cdots m_r$, sea $k_i = M/m_i$ para cada $1 \leq i \leq r$ y sea $n_i \equiv k_i^{-1} \pmod{m_i}$ para cada $1 \leq i \leq r$. La solución buscada es $x = c_1 k_1 n_1 + c_2 k_2 n_2 + \cdots + c_r k_r n_r \pmod{m_1 m_2 \cdots m_r}$.