Módulo de Algoritmos

teoría de números y temas varios

Luis J. Domínguez Pérez

ldominguez@tamps.cinvestav.mx

Marzo de 2014

Contenido

Los seis fundamentales

Hay seis conceptos elementales que necesitan enfatizarse en las matemáticas de la criptografía. Sea S un conjunto de elementos, y $+$, $\times$, $\odot$ operadores binarios en S:

- Cerradura: S está cerrado sobre $\odot$ si para todo $a, b \in S$, $a \odot b \in S$
- Asociatividad: S es asociativo sobre $\odot$ si para todo $a, b, c \in S$:

$$a \odot (b \odot c) = (a \odot b) \odot c.$$

los elementos en $\mathbb{R}$ son asociativos sobre $+$, y $\times$.

Cinvestav

. . . los seis fundamentales

- Conmutabilidad: S es conmutativa sobre $\odot$ si para todo $a, b, c \in S$:

$$a \odot b = b \odot a$$

los elementos en $\mathbb{R}$ son conmutativos sobre $+$, y $\times$.

- Distributiva: S es distributivo sobre $+$ si para todo $a, b, c \in S$:

$$a \times (b + c) = (a \times b) + (a \times c)$$

los elementos en $\mathbb{R}$ son distributivos sobre la suma

Cinvestav

. . . los seis fundamentales

- Identidad: El elemento $I \in S$ es una identidad sobre $+$ si para todo $a \in S$:

$$a + I = I + a = a$$

los elementos en $\mathbb{R}$ tienen el 0 como elemento identidad para la suma, y el 1 para la multiplicación

- Inverso: Sean $0, 1 \in S$ las identidades aditivas y multiplicativas, respectivamente, de S. Un elemento $a \in S$ es el inverso aditivo de $b \in S$ si $a + b = b + a = 0$. Es el inverso multiplicativo si $a \times b = b \times a = 1$. Por ejemplo, 2 es el inverso aditivo de -2 (y viceversa), mientras que 0.5 es su inverso multiplicativo.

Cinvestav

Estructuras algebraicas

- Las estructuras algebraicas son el corazón de la mayoría de los criptosistemas y de los ataques criptoanalíticos.

- Sea G un conjunto de elementos, y $+, \times, \odot$ operadores binarios mapeando G a G, recordando las propiedades básicas discutidas en el inicio, tenemos que...

Cinvestav

Objetos básicos

Los objetos matemáticos básicos son:

- Semigrupo: $\langle G, \odot \rangle$ es un semigrupo si G es cerrado y asociativo bajo $\odot$
- Monoide: $\langle G, \odot \rangle$ es un monoide si es un semigrupo, y existe un elemento identidad $e \in G$
- Grupo: $\langle G, \odot \rangle$ es un grupo si es un monoide, y existe un inverso para todo $a \in G$.
- Grupo abeliano: $\langle G, \odot \rangle$ es un grupo abeliano si es un grupo, y si $\odot$ es conmutativo
- Anillo: $\langle G, +, \times \rangle$ es un anillo si $\langle G, + \rangle$ es un grupo abeliano con identidad 0, $\langle G - \{0\}, \times \rangle$ es un monoide con identidad 1, y mantiene la propiedad distributiva bajo $+$
- Campo: $\langle G, +, \times \rangle$ es un campo si es un anillo, y $\langle G - \{0\}, \times \rangle$ es un grupo abeliano.

Cinvestav

Ejemplos de estructuras algebraicas

Estructura	Monoide	Grupo	G. Abeliano	Anillo	A. Conmutativo	Campo
$\langle \mathbb{Q}^{n\times n}, \times \rangle$	✓	×	×	×	×	×
$\langle \mathbb{Q}^{n\times n}(inv), \times \rangle$	✓	✓	×	×	×	×
$\langle \mathbb{Z}, + \rangle$	✓	✓	✓	×	×	×
$\langle \mathbb{Q}^{n\times n}, +, \times \rangle$	–	–	–	✓	×	×
$\langle \mathbb{Z}/(15)\mathbb{Z}, +, \times \rangle$	–	–	–	✓	✓	×
$\langle \mathbb{Z}/(17)\mathbb{Z}, +, \times \rangle$	–	–	–	✓	✓	✓

- Las estructuras más utilizadas son los campos infinitos: $\mathbb{Q}, \mathbb{R}, \mathbb{C}$.
- En criptografía, las estructuras más utilizadas son las estructuras finitas, principalmente los grupos abelianos y los campos.

Cinvestav

Observaciones sobre las estructuras

- En el caso de la criptografía, los campos utilizados son los *Campos Finitos* (también conocidos como *Campos de Galois*).
- Los campos finitos son los enteros módulo un primo p, o una potencia $q = p^m$, denotados como $\mathbb{F}_q$, o $\mathbb{Z}/q\,\mathbb{Z}$. Con $m \in \mathbb{Z}$, pudiendo ser un número compuesto.
- En el caso de ser una potencia prima, se les conoce como *extensión de campo*

Cinvestav

observaciones

- Utilizar la q es una generalización para describir el campo finito. Algunos autores prefieren utilizar p, o la potencia explícita según les convenga.
- Un caso interesante es cuando el primo es 2, o 3. Adicionalmente, la potencia m compuesta por $2^i 3^j$, con $i, j \in \mathbb{Z}$, sin ser ambos cero, es popular.

- *Recientemente* se incluyen como estructuras algebraicas comunes a los grupos abelianos de puntos en curvas elípticas sobre un campo finito (o su extensión): $E(\mathbb{F}_p)$, $E_1(\mathbb{F}_{2^{1971}})$, $E'(\mathbb{F}_{p^d})$, ...

Cinvestav

Subestructuras

Un subgrupo/campo es un subconjunto del conjunto original, el cual es cerrado bajo ciertas operaciones, y tiene las mismas propiedades que el original.

- Por ejemplo, $\mathbb{Z}/(15)\mathbb{Z}$ es un anillo, el subconjunto $\{0, 3, 6, 9, 12\}$ es cerrado, asociativo, y conmutativo bajo la adición, además, ya que tiene un elemento identidad, también es un subgrupo abeliano de $\mathbb{Z}/(15)\mathbb{Z}$ bajo la adición.

Cinvestav

Orden

- Sea $\langle G, \odot \rangle$ un grupo con un elemento identidad e. El orden de G, escrito como ord(G), u $|G|$ es el número de elementos en G. Si G es infinito, también lo es su orden.
- Otro tipo de orden existe para elementos en G. Si $a \in G$, entonces el orden de a es el entero positivo más chico $n > 0$, tal que:

$$\overbrace{a \odot a \odot \ldots \odot a}^{n \text{ times}} = 1$$

Si no existe n, entonces ord$(a) = \infty$.

Ejemplos

G	$\mathrm{ord}(G)$	a	$\mathrm{ord}(a)$
$\langle \mathbb{F}_{19}, \times \rangle$	18	7	3
$\langle \mathbb{F}_{19}, + \rangle$	19	7	19
$\langle \mathbb{F}_{17}, \times \rangle$	16	2	8
$\langle \{1,3,5,9,13\} \subset \mathbb{Z}/(14)\mathbb{Z}, \times \rangle$	6	11	3
$\langle \mathbb{Q}, \times \rangle$	∞	-1	2

Contenido de la sección 2

Divisibilidad

- Un concepto central en la teoría de números es la *divisibilidad*.
- Sean $a, b \in \mathbb{Z}$, se dice que a **divide** b (denotado como: $a|b$) si $az = b$ para algún $z \in \mathbb{Z}$. Se dice que a es un **divisor** de b, que b es un **múltiplo** de a, o que b es **divisible por** a. Si a no divide b, entonces se escribe como: $a \nmid b$

Cinvestav

Sobre la divisibilidad

Para todo $a, b, c \in \mathbb{Z}$, tenemos que:

- $a|a$, $1|b$, y $a|0$
- $0|b$ sí y solo si $b = 0$
- $a|b$ sí y solo si $-a|b$, sí y solo sí $a| - b$
- $a|b$ y $a|c$, implica que $a|(b + c)$
- $a|b$ y $b|c$, implica que $a|c$.

Observación: si $a|b$ y $b \nmid 0$, entonces $q \leq |a| \leq |b|$. De hecho, si $az = b \nmid 0$ para algún entero z, entonces $a \nmid 0$ y $z \nmid 0$; por lo que $|a| \geq 1$, $z \geq 1$, y $|a| \leq |a||z| = |b|$.

Sobre la divisibilidad

Teorema

Para todo $a, b \in \mathbb{Z}$, se tiene que $a|b$ y $b|a$ sí y sólo si $a = \pm b$.
En particular, para todo $a \in \mathbb{Z}$, tenemos que $a|1$ sí y sólo si $a = \pm 1$

Sobre la divisibilidad

Proof.
Si $a = \pm b$, entonces $a|b$ y $b|a$. Asumamos que $a|b$ y que $b|a$ para probar que $a + \pm b$. Si a o b son cero, entonces el otro debe ser cero también. Asumamos que ninguno es cero. $a|b$ implica que $|a| \leq |b|$, y $b|a$ implica que $|b| \leq |a|$; por lo que $|a| = |b|$, entonces $a = \pm b$. Esto prueba la primera parte. La segunda parte viene de poner a $b = 1$, entonces $1|a$ □

Cinvestav

Números Primos

Sea n un número positivo y entero. Sabemos que 1 y n dividen n. Si $n > 1$ y ningún otro número además de 1 y n lo dividen, decimos que n es **primo**. Si $n > 1$ pero n no es primo, entonces decimos que n es **compuesto**. Nota: el número 1 no se considera ni primo, ni compuesto.

n es compuesto si y sólo si $n = ab$ para algún entero a, b con $1 < a < n$, y $1 < b < n$.

Normalmente, al hablar de un número primo o compuesto, nos referimos a un número entero positivo.

Cinvestav

Teorema fundamental de la aritmética

Teorema

Todo entero n distinto de cero puede expresarse como:

$$n = \pm p_1^{e_1} \cdots p_r^{e_r}$$

donde $p_1, \ldots, p_r$ son primos distintos, y $e_1, \ldots, e_r$ son enteros positivos.

Cinvestav

teorema fundamental de la aritmética (proof)

Proof

Por unicidad, la expresión que describe un número entero es única después de reordenar los primos:

- Asuma que $s > 1$ es el producto de números primos escrita de dos maneras diferentes:

$$\begin{aligned} s &= p_1 \cdot p_2 \cdots p_m \\ &= q_1 \cdot q_2 \cdots q_n. \end{aligned}$$

- Debemos demostrar que $m = n$ and that the q_j are a rearrangement of the p_i.

teorema fundamental de la aritmética (proof)

- Por el lema de Euclides (divisibilidad), p_1 debe dividir a uno de los q_j; reetiquetando los q_j de ser necesario, digamos que p_1 divide q_1. Dado que q_1 es primo, sus únicos divisores son el 1 y sí mismo, por lo que $p_1 = q_1$, entonces

$$\begin{aligned}\frac{s}{p_1} &= p_2 \cdots p_m \\ &= q_2 \cdots q_n.\end{aligned}$$

- Siguiendo el mismo razonamiento, p_2 debe de ser igual a alguno de los q_j restantes. Reetiquetamos otra vez de ser necesario, por ejemplo $p_2 = q_2$. Entonces,

$$\begin{aligned}\frac{s}{p_1 \cdot p_2} &= p_3 \cdots p_m \\ &= q_3 \cdots q_n.\end{aligned}$$

teorema fundamental de la aritmética (proof)

- Esto puede hacerse para todo m de p_i, demostrando que $m \leq n$. Si hubiera otra q_j tendríamos que

$$\frac{s}{p_1 \cdot p_2 \cdots p_m} = 1$$
$$= q_k \cdots q_n,$$

lo cual es imposible, dado que el producto de números mayores a 1 no puede ser igual a 1, por lo que $m = n$, y cada q_j es un p_i.

□

Cinvestav

Conceptos

Teorema - Propiedad de la división con residuo

Sea $a, b \in \mathbb{Z}$ con $b > 0$. Existen $q, r \in \mathbb{Z}$ únicos tales que $a = bq + r$, con $0 \leq r \leq b$.

Número liso

Si un número entero positivo es divisible solamente por números primos "pequeños", se dice que es un número *liso* (smooth).

Los números lisos son muy utilizados en el criptoanálisis para verificar un sistema (romperlo). Por otro lado, los números que solamente se pueden factorizar por dos números primos muy grandes son esenciales para la criptografía de clave pública.

Cinvestav

Divisor

Máximo común divisor

Dados dos números $a, b \in \mathbb{Z}$, distintos a cero, el Máximo común divisor (MCD), denotado como MCD(a, b), o en ocasiones símplemente como (a, b), es un número entero d que es el más grande que divide tanto a a como a b.

Mínimo común múltiplo

Dados dos números $a, b \in \mathbb{Z}$, distintos a cero, el mínimo común múltiplo (mcm), denotado como mcm(a, b) es el número entero positivo más pequeño al cual a y b dividen.

Cinvestav

Algoritmo de Euclides

El algoritmo de Euclides es una manera rápida de encontrar el MCD(a, b) aún cuando se desconozcan los factores primos de a y b.

El algoritmo funciona así:

- Reordene para que $a > b$
- Divida a sobre b, y guarde el cociente q_1, y el residuo $r_1 : a = q_1 b + r_1$
- Reordene para que $a > b$: b es el nuevo a, y r_1 es el nuevo b
- Divida b sobre r_1 y guarde q_2 y $r_2 : b = q_2 r_1 + r_2$
- Reordene para que $a > b \ldots$
- Se detiene el algorimto cuando el último residuo divide al anterior: $r_n | r_{n-1}$

Ejemplo:

Encontrar el MCD(1547, 560):

$$\begin{aligned} 1547 &= 2 \cdot 560 + 427 \\ 560 &= 1 \cdot 427 + 133 \\ 427 &= 3 \cdot 133 + 28 \\ 133 &= 4 \cdot 28 + 21 \\ 28 &= 1 \cdot 21 + 7 \end{aligned}$$

Dado que $7|21$ hemos terminado: $\text{MCD}(1547, 560) = 7$.

Ejercicio

- Genere un programa en Maple para calcular el MCD.

Propiedades

- Por definición $\text{MCD}(0,0) = 0$
- $\text{MCD}(a,b) = \text{MCD}(b,a)$
- $\text{MCD}(a,b) = \text{MCD}(-a,b)$
- $\text{MCD}(-a,0) = |a|$
- $\text{MCD}(a,b) \cdot c = \text{MCD}(ac,bc)$, si $c \geq 0$
- $\text{mcm}(a,b) \cdot c = \text{mcm}(ac,bc)$, si $c \geq 0$
- $ab = \text{MCD}(a,b)\text{mcm}(a,b)$, si $a,b \geq 0$
- $\text{MCD}(\text{mcm}(a,b), \text{mcm}(a,c)) = \text{mcm}(a, \text{MCD}(b,c))$
- $\text{mcm}(\text{MCD}(a,b), \text{MCD}(a,c)) = \text{MCD}(a, \text{mcm}(b,c))$

Cinvestav

Propiedades binarias

- Si a, b son pares, entonces:
  - $\text{MCD}(a, b) = 2 \cdot \text{MCD}(a/2, b/2)$
- Si a es par, y b es impar, entonces:
  - $\text{MCD}(a, b) = \text{MCM}(a/2, b)$
  - $\text{MCD}(a, b) = \text{MCM}(a - b, b)$
- Si a, b son impares, entonces:
  - $a - b$ es par
  - $|a - b| < \max(a, b)$

Cinvestav

Ejercicio

- Genere un programa en Maple para calcular los números primos menores a 10000.

- Si contamos a 2 como el primer número primo, 3 el segundo, ¿cuál es el 100mo. primo?

Contenido de la sección 3

MCD revisitado

Hemos visto anteriorme cómo calcular el máximo común divisor de un número. Si $a, b \in \mathbb{Z}$, con $0 < b \leq a$, entonces:

- del algoritmo tradicional de la división sabemos que existen $q, r \in \mathbb{Z}$, con $r < b$, y $a = bq + r$.
- si $g \in \mathbb{Z}$, y $g|a$ y $g|b$, entonces:

$$g|(a - bq) \Rightarrow g|r$$

MCD revisitado

Ahora bien:

- Si $\text{MCD}(a, b) = g$, entonces implica que $g|r$, donde r es el residuo que resulta de dividir a por b
- Si $r = 0$, entonces $b|a$, y el máximo común divisor de a, b es b.
- Si $r \neq 0$, entonces $\text{MCD}(a, b) = \text{MCD}(b, r)$, la cual es una operación más económica. Repitiendo este proceso hasa que $r = 0$ nos da el algoritmo MCD.

Cinvestav

Algoritmo MCD

Algoritmo simple de MCD:

Input: Integers $0 < b \leq a$
Ouput: MCD(a, b)

$n = a$; $d = b$
$r = n - (d \times \lfloor \frac{n}{d} \rfloor)$
while $r \neq 0$ **do**
$n = d$
$d = r$
$r = n - (d \times \lfloor \frac{n}{d} \rfloor)$
end while
return MCD$(a, b) = d$

MCD extendido

El algoritmo MCD extendido es idéntico al algoritmo estándar, pero además carga con información adicional. Si $\text{MCD}(a, b) = g$, entonces sabemos que existen $x, y \in \mathbb{Z}$ tal que:

$$ax + by = g$$

y es la mínima combinación lineal positiva para a y b.

MCD extendido

- Note que si el MCD es 1, entonces esta ecuación nos da los inversos para $a \bmod b$, y $b \bmod a$: $x \equiv a^{-1} \bmod b$, y $y \equiv b^{-1} \bmod a$.
- Si el MCD no es 1, entonces, y dado que es la combinación linear más pequeña, se dice que no existen inversos para $a \bmod b$ o $b \bmod a$.
- Extendiendo el algoritmo de MCD para calcular este dato extra, nos da un algoritmo eficiente para calcular los inversos multiplicativos.

Cómo extender el algoritmo MCD

Para extender el algoritmo MCD, la parte de la ecuación debe de agregarse a las iteraciones. Los valores iniciales de la ecuación son:

$$a(1) + b(0) = a$$
$$a(0) + b(1) = b$$

Algoritmo extendido de MCD

Input: Integers $0 < b \leq a$

Ouput: $x, y, \text{MCD}(a, b)$ tal que $ax + by = \text{MCD}(a, b)$

$v_0 = a;\ v_1 = b$

$x_0 = 1;\ y_0 = 0$

$x_1 = 0;\ y_1 = 1$

$i = 1$ {puntero al valor v más pequeño}

while $v_i \neq 0$ **do**

$i = i + 1 \bmod 2$

$q = \left\lfloor \frac{v_i}{v_{i+1 \bmod 2}} \right\rfloor$

$v_i = v_i - (q \times v_{i+1 \bmod 2})$

$x_i = x_i - (q \times x_{i+1 \bmod 2})$

$y_i = y_i - (q \times y_{i+1 \bmod 2})$

end while

$i = i + 1 \bmod 2$

return $x_i, y_i, \text{MCD}(a, b) = v_i$

Cinvestav

Ejercicio

- Implemente el algoritmo extendido de Euclides (MCD)

- Calcule el inverso multiplicativo de $101 \bmod 1999$, y el inverso de $1999 \bmod 101$ utilizando el algoritmo.

Contenido de la sección 4

Congruencias

Propiedades básicas

Dados tres enteros a, b, y m, decimos que a *es congruente a* b *módulo* m, denotado: $a \equiv b \bmod m$, si la diferencia $a - b$ es divisible por m.

A m se le conoce como el *módulo* de la congruencia.

Propiedades

Propiedades de la congruencia:

1.
   - $a \equiv a \bmod m$
   - $a \equiv b \bmod m$ sí y solo sí $b \equiv a \bmod m$
   - Si $a \equiv b \bmod m$, y $b \equiv c \bmod m$, entonces $a \equiv c \bmod m$

   Para una m fija, esto significa que la congruencia módulo m es una *relación de equivalencia*.

Propiedades. . .

2.
- Para una m fija, cada *clase de equivalencia* con respecto a un módulo m tiene 1 y sólo 1 representante entre 0 y $m-1$.
- El conjunto de clases de equivalencia (*clases residuales*) se denota como $\mathbb{Z}/m\mathbb{Z}$
- Cualquier conjunto de representantes para las clases residuales es llamado *conjunto completo de residuos módulo* m.

Propiedades...

3. Si $a \equiv b \bmod m$ and $c \equiv d \bmod m$, then $a \pm c \equiv b \pm d \bmod m$ and $ac \equiv bd \bmod m$.

En otras palabras, las congruencias (con el mismo módulo) se pueden sumar, restar, o multiplicar.

- El conjunto de clases de equivalencia $\mathbb{Z}/m\mathbb{Z}$ es un anillo conmutativo (lo veremos en la siguiente unidad). Esencialmente, las clases residuales se pueden sumar, restar, o multiplicar, y los axiomas básicos aplican (asociatividad, conmutabilidad, inversos aditivos, etc.)

Propiedades. . .

4. Si $a \equiv b \bmod m$, entonces $a \equiv b \bmod d$ para cualquier $d|m$
5. Si $a \equiv b \bmod m$, $a \equiv \bmod n$, y m y n son primos relativos, entonces $a \equiv b \bmod mn$

Pequeño teorema de Fermat

Teorema

Sea p un primo. Cualquier entero a satisface $a^p \equiv a \bmod p$, y cualquier entero a no divisible por p satisface $a^{p-1} \equiv 1 \bmod p$

Pequeño teorema de Fermat (proof)

- Suponga que $p \nmid a$.
- Tenemos que los enteros $0a, 1a, 2a, \ldots, (p-1)a$ son el conjunto de residuos módulo p.
- Observe que 2 elementos, ia y ja tendrían que estar en la misma clase residual $ia \equiv ja \bmod p$; sin embargo, esto significaría que $p|(i-j)a$, como a no es divisible por p, tendríamos que $p|(i-j)$, pero como $i, j < p$ se requiere que $i = j$ para que esto sucediera
- Concluímos que los enteros $a, 2a, \ldots, (p-1)$ son un reacomodo de $a, 2, \ldots, p-1$ cuando se considera el módulo p ...

Pequeño teorema de Fermat (proof)

- Ahora tenemos que los números de la primera secuencia son congruentes módulo p a los números de la segunda secuencia: $a^{(p-1)}(p-1)! \equiv (p-1)! \bmod p$. Entonces, $p|((p-1)!(a^{p-1}-1))$
- Dado que $(p-1)!$ no es divisible por p, nos quedamos con $p|(a^{p-1}-1)$.
- Finalmente, si multiplicamos ambos lados de la congruencia $a^{p-1} \equiv 1 \bmod p$, tenemos la primer congruencia del teorema cuando a no es divisible por p
- Si a es divisible por p, entonces ambos lados se hacen $\equiv 0 \bmod p$.

□

Cinvestav

Corolario sobre el pequeño teorema de Fermat

Corolario

Si a no es divisible por p y si $n \equiv m \bmod (p-1)$, entonces $a^n \equiv a^m \bmod p$.

Corolario sobre el pequeño teorema de Fermat (proof)

Proof

- Suponga que $n > m$
- dado que $p - 1 | n - m$, tenemos que $n = m + c(p - 1)$, para un entero positivo c.
- multiplicando la congruencia $a^{p-1} \equiv 1 \bmod p$ por sí misma c veces
- y después por $a^m \equiv a^m \bmod p$, obtenemos $a^n \equiv a^m \bmod p$.

Chinese Remainder Theorem (CRT)

Definición

Sea R un anillo, I es un *ideal* de R si es un subconjunto no vacío de R tal que:

- I es un subgrupo de R con respecto a la ley de $+$
- para todo $x \in \mathbb{Z}$ y todo $y \in R$, $xy \in I$ y $yx \in I$.

- El ideal $I \subsetneq R$ es *primo* si para todo $x, y\ in R$ con $xy \in I$ se obtiene $x \in I$ o $y \in I$.
- El ideal $I \subsetneq R$ es *máximal* si para todo ideal J de R la inclusión $I \subset J$ implica que $J = I$ o $J = R$.
- Dos ideales I y J de R son *coprimos* si $I + J = \{i = j | i \in I y j \in J\}$ es igual a R.

Cinvestav

Chinese Remainder Theorem (CRT)

Definición

Un ideal I de un anillo R está *finitamente generado* si existen elementos $a_1, \ldots, a_n$ tal que para cada $x \in I$ se pueda escribir $x = x_1 a_1 + \cdots + x_n a_n$ con $x_1, \ldots, x_n \in R$.

El ideal I es *principal* si $I = aR$ y R es un *dominio de ideals principales (PID)*, si es un dominio integral, y si cada ideal de R es principal. Ejemplos:

- El anillo de los enteros $\mathbb{Z}$
- El anillo de polinomios $\mathcal{K}[X]$, en donde $\mathcal{K}$ es un campo.

Cinvestav

Chinese Remainder Theorem (CRT)

Teorema

Sean $I_1, \ldots, I_k$ ideales coprimos en parejas de R, entonces:

$$R/\prod_{i=1}^{k} I_i \simeq \prod_{i=1}^{k} R/I_i$$

Corolario

Sea $n_1, \ldots, n_k$ enteros coprimos, e.g. $(n_i, n_j) = 1$ para $i \neq j$. Entonces, existe una solución simultánea a x:

$$\begin{cases} x & \equiv x_1 \bmod n_1 \\ x & \equiv x_2 \bmod n_2 \\ & \vdots \\ x & \equiv x_k \bmod n_k \end{cases}$$

congruentes entre si módulo $N = \prod_{i=1}^{k} n_i$.

Chinese Remainder Theorem (CRT) - proof

- Unicidad módulo N: Suponga que x' y x'' son dos soluciones. Si $x = x' - x''$, entonces x debe de ser congruente a 0 módulo n_i, y por lo mismo, a módulo N.
- Definimos $N_i = N/n_i$ como el producto de los módulos *excepto el i-nésimo*. El $(n_i, n_j) = 1$, y existe un entero M_i tal que $N_i M_i = 1 \bmod n_i$
- Si tenemos que $x = \sum_i a_i N_i M_i$, para cada i los términos en la suma son divisibles por n_i, ya que $n_i | N_i$ para valores de $i \neq j$, así que para cada i tenemos que $x \equiv a_i N_i M_i \equiv a_i \bmod n_i$. □

Cinvestav

Chinese Remainder Theorem (CRT)

Sabiendo que $x = \sum_i a_i N_i M_i$, ¿cómo se soluciona el siguiente sistema de ecuaciones?

$$\begin{cases} x & \equiv 1 \bmod 3 \\ x & \equiv 2 \bmod 5 \\ x & \equiv 4 \bmod 7 \\ x & \equiv 5 \bmod 11 \\ x & \equiv 9 \bmod 13 \end{cases}$$

Algoritmo CRT

Input: Enteros coprimos $n_1, \ldots, n_k$ y enteros x_1 para $1 \leq i \leq k$.

Ouput: Entero x tal que $x \equiv x_i \bmod n_i$, para todo $1 \leq i \leq k$

$N \leftarrow n_1$; $x = x_1$

for $i = 2$ **to** k **do**

Calcule u y v tal que $un_i + vN = 1$ [use el GCD extendido]

$x \leftarrow un_i x + vNx_i$

$N \leftarrow Nn_i$

$x \leftarrow x \bmod N$

end for

return x

Algoritmo CRT

i	n_i	x_i	N	u	v	x	$x \bmod n_i$
1	3	1	3	-	-	1	1
2	5	2	15	-1	2	7	2
3	7	4	105	-2	1	67	4
4	11	5	1155	-19	2	907	5
5	13	9	15015	-533	6	8992	9

Función Euler phi φ

Definición

La función φ de Euler, también llamada *totient*, es la función definida por la siguiente regla:

$$\varphi(m) = \#\mathbb{Z}/m\mathbb{Z} = \#\{0 \leq a < m : \mathbf{MCM}(a, m) = 1\}$$

Podemos calcular el valor con:

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

Ejemplo

Calcular para 100:

- Factores: $2, 5$
- $\varphi(100) = 100 \times \left(1 - \frac{1}{2}\right) \times \left(1 - \frac{1}{5}\right) = 40$

- En el caso de números primos, $\varphi(p) = p - 1$.
- En el caso de potencias de primos, $\varphi(p^\alpha) = p^\alpha(1 - \frac{1}{p})$

Ejercicio

- Genere un programa en Maple para calcular la función φ de Euler.

Observaciones

Corolario

La función Euler phi-function is *multiplicativa*, esto es $\varphi(mn) = \varphi(m)\varphi(n)$, siempre y cuando $(m, n) = 1$.

Observaciones

Proof

- Se deberán de contar los enteros entre 0 y $mn-1$ que no tienen factor común con mn.
- Para todo elemento (j) en el rango, sea j_1 el residuo módulo m no-negativo más chico, y j_2 el de módulo n.
- Del Chinese Remainder Theorem tenemos que por cada j_1, j_2 hay solamente una $0 \leq j < mn-1$ que cumple con $j \equiv j_1 \bmod m$, $j \equiv j_2 \bmod n$.
- Note que j no tiene factor común con $m : (j_1, m) = 1$, ni con $n : (j_2, n) = 1$, por lo que las j's que vamos a contar tienen correspondencia $1-\text{to}-1$ con los pares j_1, j_2 correspondientes.
- El número posible de j_1's son $\varphi(m)$, y el número de j_2's es $\varphi(n)$, por lo que el número de parejas es $\varphi(m)\varphi(n)$.

□

Cinvestav

Observaciones

Proposición

Si $(a, m) = 1$, entonces $a^{\varphi(m)} \equiv 1 \bmod m$

Observaciones - proof

Proof

m es una potencia prima ($m = p^\alpha$).

- Si $\alpha = 1$, es el teorema pequeño de Fermat
- Si $\alpha \geq 2$, y la fórmula aplica para la $(\alpha - 1)-$ava potencia de p, entonces $a^{p^{\alpha-1}-p^{\alpha-2}} = +p^{\alpha-1}b$, para un entero b.
- Elevando ambos lados de la ecuación a la p potencia (y tomando en cuenta que los coeficientes del binomio en $(1+x)^p$ son divisibles por p), tenemos que $a^{p^\alpha - p^{\alpha-1}}$ es igual a 1 más la suma de cada término divisible por P^α, esto es $a^{\varphi(p^\alpha)} - 1$ es divisible por p^α.
- Dada la multiplicabilidad de φ, $a^{\varphi(m)} \equiv 1 \bmod p^\alpha$, y ya que $p^\alpha | m$, y que estas potencias primas no tienen factores comunes, tenemos que $a^{\varphi(m) \equiv 1 \bmod m}$.

Contenido de la sección 5

Enteros largos

- Sin software o hardware especial, las computadoras solamente pueden trabajar con números enteros pequeños (del tamaño de la palabra del procesador).
- para poder considerar cantidades mayores, primeramente hay que recordar ciertos aspectos elementales de los enteros.

Base

- Debemos representar los enteros grandes como vectores de dígitos en alguna base B, junto con un bit que indique el signo.
- Esto es, para $a \in \mathbb{Z}$, si escribimos

$$a = \pm \sum_{i=0}^{k-1} a_i B^i = \pm(a_{k-1} \cdot a_1 a_0)_B,$$

en donde $0 \leq a_i < B$ para $i = 0, \ldots, k-1$, entonces a será representado en memoria como una estructura de datos que consiste en el vector de base$-B$ con dígitos $a_0, \ldots, a_{k-1}$, junto con un "bit de signo" para indicar el signo de a.

Base

- Para asegurar una representación única, si a no es cero, entonces su bit más representativo a_{k-1} debera no ser cero.
- Usualmente, escogemos una base B constante, y preferentemente una potencia de 2, lo cual es conveniente en términos técnicos para la computadora.

Para fines prácticos, utilizaremos QuoRem(x, y) para denotar la pareja $(\lfloor x/y \rfloor, x \bmod y)$

Adición

Sea $a = (a_{k-1} \cdots a_0)_B$, y $b = (b_{k-1} \cdots b_0)_B$ dos enteros sin signo, y asumiendo que $k \geq \ell \geq 1$ (se pueden intercambiar de ser necesario)

- la suma $c := a + b$ es de la forma $c = (c_k c_{k-1} \cdots c_0)_B$
- el método de "escuela" es:

Adición - Algoritmo

carry $\leftarrow 0$
for $i \leftarrow 0$ **to** $\ell - 1$ **do**
  tmp $\leftarrow a_i + b_i$+carry, (carry, c_i) $\leftarrow$ QuoRem(tmp, B)
end for
for $i \leftarrow \ell$ **to** $k - 1$ **do**
  tmp $\leftarrow a_i$+carry, (carry, c_i) $\leftarrow$ QuoRem(tmp, B)
end for
$c_k \leftarrow$ carry

Note que el carry tiene valores 0 o 1, y que la variable tmp $\in [0 \ldots 2B - 1]$

Cómputo en $\mathbb{Z}_n$

Sea $n > 1$

- Para cada $\alpha \in \mathbb{Z}$, existe un entero único $a \in \{0, \ldots, n-1\}$ tal que $\alpha = [a]_n$
- Este entero a se le conoce como *representante canónico* de α, y se denota como $\text{rep}(\alpha)$.
- Para fines computacionales, se representan los elementos de $\mathbb{Z}_n$ con sus representantes canónicos.

Cómputo en $\mathbb{Z}_n$

En la práctica, no se hace distinción, y simplemente se escriben los elementos tal cual. Si se necesita hacer una conversión, entonces se utiliza:

- $\alpha \leftarrow [a]_n$, si $a \in \{0, \ldots, n-1\}$
- $a \leftarrow \text{rep}(\alpha)$

(no se realiza ningún cómputo)

Sin embargo, note que:

- $a \bmod n$ es un elemento en el conjunto $\{0, \ldots, n-1\}$
- $[a]_n$ es un elemento en $\mathbb{Z}_n$

Adición modular - Algoritmo

Input: módulo p, enteros $a, b \in [0, \ldots, p-1]$
Ouput: $c := a + b \bmod p$

$c_0 = \text{Add}(a_0, b_0)$
for $i \leftarrow 1$ **to** $k-1$ **do** $c_i = \text{AddCC}(a_i, b_i)$ **end for**
if carry **then** $c \leftarrow c - p$ **end if**
if $c \geq p$ **then** $c \leftarrow c - p$ **end if**
return $c = (c_{k-1}, \ldots, c_0)$

En donde

- Add - Adición sin acarreo de entrada
- AddCC - Adición con acarreo

Sustracción

Sea $a = (a_{k-1} \cdots a_0)_B$, y $b = (b_{k-1} \cdots b_0)_B$ dos enteros sin signo, y asumiendo que $k \geq \ell \geq 1$

- La resta $c := a - b$ puede utilizar el mismo algoritmo anterior (pero restando)
- En cada iteración, el carry toma valores de 0 o -1, y $-B \leq \text{tmp} \leq B - 1$.
- Si $a \geq b$ $c_k = 0$, sino, $c_k = -1$ (y $b - a = B^k - (c_{k-1} \cdots c_0)_B$), cuyo resultado se puede obtener con otra resta.

¿Cuál es la complejidad del algoritmo de suma?

Sustracción modular - Algoritmo

Input: módulo p, enteros $a, b \in [0, \ldots, p-1]$
Ouput: $c := a - b \bmod p$
$c_0 = \text{Sub}(a_0, b_0)$
for $i \leftarrow 1$ **to** $k-1$ **do** $c_i = \text{SubCC}(a_i, b_i)$ **end for**
if carry **then** $c \leftarrow c + p$ **end if**
return $c = (c_{k-1}, \ldots, c_0)$

En donde

- Sub - Sustracción sin acarreo de entrada
- SubCC - Sustracción con acarreo (préstamo)

Multiplicación

Sea $a = (a_{k-1} \cdots a_0)_B$, y $b = (b_{k-1} \cdots b_0)_B$ dos enteros sin signo, y asumiendo que $k \geq \ell \geq 1$

- El producto $c := a \cdot b$ es de la forma $(c_{k+\ell-1} \cdots c_0)_B$, y se puede calcular como sigue:

Multiplicación

				a_3	a_2	a_1	a_0
				b_3	b_2	b_1	b_0
				$t_{0,3}$	$t_{0,2}$	$t_{0,1}$	$t_{0,0}$
			$t_{1,3}$	$t_{1,2}$	$t_{1,1}$	$t_{1,0}$	
		$t_{2,3}$	$t_{2,2}$	$t_{2,1}$	$t_{2,0}$		
	$t_{3,3}$	$t_{3,2}$	$t_{3,1}$	$t_{3,0}$			
t_7	t_6	t_5	t_4	t_3	t_2	t_1	t_0

Multiplicación - Algoritmo

for $i \leftarrow 0$ **to** $k + \ell - 1$ **do** $c_i \leftarrow 0$ **end for**
for $i \leftarrow 0$ **to** $k - 1$ **do**
  carry $\leftarrow 0$
  for $j \leftarrow 0$ **to** $\ell - 1$ **do**
    tmp $\leftarrow a_i b_j + c_{i+j}$+carry
    (carry, c_{i+j}) $\leftarrow$ QuoRem(tmp, B)
  end for
  $c_{i+j} \leftarrow$ carry
end for

Note que el valor carry $\in [0 \ldots B - 1]$, y tmp $\in [0 \ldots B^2 - 1]$.

¿Cuál es la complejidad del algoritmo? ¿Cuál es el número de multiplicaciones?

Multiplicación Karatsuba-Offman

Sea $R = B^n$, $d = 2n$, y $u = (u_{d-1}, \ldots, u_0)_B$ y $v = (v_{d-1}, \ldots, v_0)_B$ son dos enteros de d-palabras.

- El método de Karatsuba se basa en que al separar u y v en dos partes: $u = U_1 R + U_0$, y $v = V_1 R + V_0$, podemos obtener:

$$uv = U_1 V_1 R^2 + ((U_0 + U_1)(V_0 + V_1) - U_1 V_1 - U_0 V_0) R + U_0 V_0$$

¿Cuál es el número de multiplicaciones?

Multiplicación Karatsuba-Offman - Algoritmo

Input: $u = (u_{n-1}, \ldots, u_0)_B$ y $v = (v_{m-1}, \ldots, v_0)_B$,
$d = \max(m, n)$, valor d_0 máximo

Ouput: $w := uv$, $w = (w_{(m+n)-1}, \ldots, w_0)_B$

if $d \leq d_0$ **then return** uv **end if**

$p \leftarrow \lfloor d/2 \rfloor$, $q \leftarrow \lceil d/2 \rceil$

$U_0 \leftarrow (u_{q-1}, \ldots, u_0)_B$, $V_0 \leftarrow (v_{q-1}, \ldots, v_0)_B$

$U_1 \leftarrow (u_{p+q-1}, \ldots, u_q)_B$, $V_1 \leftarrow (v_{p+q-1}, \ldots, v_q)_B$

$U_s \leftarrow U_0 + U_1$, $V_s \leftarrow V_0 + V_1$

Calcule recursivamente $U_o V_o$, U_1, V_1, $U_s V_s$

return $U_1 V_1 B^{2q} + (U_s V_s - U_1 V_1 - U_o V_0)^q + U_0 V_0$

¿Cuál es la complejidad del algoritmo?

Cuadrado

Una modificación directa al algoritmo de multiplicación da el siguiente algoritmo para los cuadrados. Dándonos la mitad de las operaciones de multiplicación.

Cuadrado - Algoritmo

Input: $u = (u_{\ell-1}, \ldots u_0)_B$
Ouput: $w := u^2$, $w = (w_{2\ell-1}, \ldots, w_0)_B$
  for $i \leftarrow 0$ **to** $2\ell - 1$ **do** $w_i \leftarrow 0$ **end for**
  for $i \leftarrow 0$ **to** $\ell - 1$ **do**
    $t \leftarrow u_i^2 + w_{2i}$
    $w_{2i} \leftarrow t \bmod B$, $k \leftarrow \lfloor t/B \rfloor$
    for $j \leftarrow i + 1$ **to** $\ell - 1$ **do**
      $t \leftarrow 2u_i u_j + w_{i+j} + k$
      $w_{i+j} \leftarrow t \bmod B$, $k \leftarrow \lfloor t/B \rfloor$
    end for
    $w_{i+\ell} \leftarrow k$
  end for
  return $(w_{2\ell-1}, \ldots, w_0)_B$

Cuadrado

Escriba la fórmula de cuadrados utilizando la idea de Karatsuba

Cuadrado

Algoritmo alternativo [Guajardo and Paar]

Input: $u = (u_{\ell-1}, \ldots u_0)_B$
Ouput: $w := u^2$, $w = (w_{2\ell-1}, \ldots, w_0)_B$
  for $i \leftarrow 0$ **to** $2\ell - 1$ **do** $w_i \leftarrow 0$ **end for**
  for $i \leftarrow 0$ **to** $\ell - 1$ **do**
    $(k, t) \leftarrow w_{2i} + u_i^2$
    $w_{2i} \leftarrow t$, $c_1 \leftarrow k$, $c_2 \leftarrow 0$
    for $j \leftarrow i + 1$ **to** $\ell - 1$ **do**
      $(k, t) \leftarrow w_{i+j} + u_i u_j + c_1$, $c_1 \leftarrow k$
      $(k, t) \leftarrow t + w_i w_j + c_2$, $w_{i+j} \leftarrow t$, $c_2 \leftarrow k$
    end for
    $(k, t) \leftarrow c_1 + c_2$, $c_2 \leftarrow k$
    $(k, t) \leftarrow w_{i+\ell} + t$, $w_{i+\ell} \leftarrow t$
    $w_{i+\ell+1} \leftarrow c_2 + k$
  end for
  return $w = (w_{2\ell-1}, \ldots, w_0)_B$

División con residuo

Sea $a = (a_{k-1} \cdots a_0)_B$, y $b = (b_{k-1} \cdots b_0)_B$ dos enteros sin signo, $k \geq 1$, $\ell \geq 1$, $b_{\ell-1} \neq 0$

- Se quiere calcular q y r tal que $a = bq + r$, y $0 \leq r < b$
- Si $k < \ell$, entonces $q \leftarrow 0$, $r \leftarrow a$.
- El cociente q tiene a lo más $m := k - \ell + 1$ base$-B$ dígitos.
- $q = (q_{m-1} \cdots q_0)_B$

División con residuo - Algoritmo

El algoritmo genérico es el siguiente:

$r \leftarrow a$
for $i \leftarrow m-1$ down **to** 0 **do**
  $q_i \leftarrow \lfloor r/B^i b \rfloor$
  $r \leftarrow r - B^i \cdot q_i b$
end for

Al inicio de cada ciclo $0 \leq r < B^{i+1}b$, y $0 \leq q_i \leq B-1$

- Note que es necesaria una manera eficiente de realizar $\lfloor r/B^i b \rfloor$.

División con residuo

Caso especial $\ell = 1$

Theorem

Sea x y y enteros tales que

$$0 \leq x = x'2^n + s \text{ y } 0 < y = y'2^n$$

para enteros n, s, x', y', con $n \geq 0$ y $0 \leq s < 2^n$, por lo que $\lfloor x/y \rfloor = \lfloor x'/y' \rfloor$

División con residuo (proof)

Proof.

- Tenemos que
$$\frac{x}{y} = \frac{x'}{y'} + \frac{s}{y'2^n} \geq \frac{x'}{y'}$$
por lo que $\lfloor x/y \rfloor \geq \lfloor x'/y' \rfloor$.
- También tenemos que
$$\frac{x}{y} = \frac{x'}{y'} + \frac{s}{y'2^n} < \frac{x'}{y'} + \frac{1}{y'} \leq \left(\lfloor \frac{x'}{y'} \rfloor + \frac{y'-1}{y'} \right) + \frac{1}{y'}$$
Así que tenemos $x/y < \lfloor x'/y' \rfloor + 1$, y entonces $\lfloor x/y \rfloor \leq \lfloor x'/y' \rfloor$.

División con residuo - Algoritmo

De este teorema, se deriva el algoritmo para el caso de $\ell = 1$

$r \leftarrow 0$
for $i \leftarrow k-1$ down **to** 0 **do**
  tmp $\leftarrow r \cdot B + a_i$
  $(q_i, r) \leftarrow$ QuoRem(tmp, b_0)
end for
return $q = (q_{k-1} \cdots q_0)$, r

Note que en cada iteración el valor de r está entre 0 y $b_0 < B-1$, y que *tmp* está entre 0 y $B \cdot b_0 + (B-1) \leq B^2 - 1$.

División con residuo

Caso general $\ell > 1$

Theorem

Sea x y y enteros tales que

$$0 \leq x = x'2^n + s \text{ y } 0 < y = y'2^n + t$$

para enteros n, s, t, x', y', con $n \geq 0$, $0 \leq s < 2^n$ y $0 \leq t < 2^n$, suponiendo que $2y' \geq x/y$, entonces

$$\lfloor x/y \rfloor \leq \lfloor x'/y' \rfloor \leq \lfloor x/y \rfloor + 2.$$

División con residuo (proof)

Proof.

- Tenemos que $x/y \leq x'/y'2^n$, así que $\lfloor x/y \rfloor \leq \lfloor x'/y'2^n \rfloor$, y por el teorema anterior tenemos que $\lfloor x/y'2^n \rfloor = \lfloor x'/y' \rfloor$. Con esto se prueba la primer inequalidad
- Tenemos que $x/y \geq x'/(y'+1)$, lo que implica que $x'y - xy' - x \leq 0$. Dado que $2y' \leq x/y$, y que $2yy' - x \geq 0$, tenemos que $2yy' - x \geq 0 \geq x'y - xy' - x$, lo que implica que $x/y \geq x'/y' - 2$, y que $\lfloor x/y \rfloor \geq \lfloor x'/y' \rfloor - 2$

□

División con residuo - Algoritmo

Se deriva el algoritmo para el caso de $\ell \geq 1$

for $i \leftarrow 0$ **to** $k-1$ **do** $r_i \leftarrow a_i$ **end for**
$r_k \leftarrow 0$
for $i \leftarrow k-\ell$ down **to** 0 **do**
  $q_i \leftarrow \lfloor (r_{i+\ell}B + r_{i+\ell-1})/b_{\ell-1} \rfloor$
  if $q_i \geq B$ **then** $q_i \leftarrow B-1$ **end if**
  carry $\leftarrow 0$
  for $j \leftarrow 0$ **to** $\ell - 1$ **do**
    tmp $\leftarrow r_{i+j} - q_i b_j +$ carry
    (carry, r_{i+j}) $\leftarrow$ QuoRem(tmp, B)
  end for
  $r_{i+\ell} \leftarrow r_{i+\ell} +$ carry
  while $r_{i+\ell} < 0$ **do**
    carry $\leftarrow 0$
    for $j \leftarrow 0$ **to** $\ell - 1$ **do**
      tmp $\leftarrow r_{i+j} + b_i +$ carry
      (carry, r_{i+j}) $\leftarrow$ QuoRem(tmp, B)
    end for
    $r_{i+\ell} \leftarrow r_{i+\ell} +$ carry
    $q_i \leftarrow q_i - 1$
  end while
end for
return $q = (q_{k-1} \cdots q_0)_B$, $r = (r_{\ell-1}, \cdots r_0)_B$

Cinvestav

División con residuo - Nota

El algoritmo anterior supone una b "normalizada", esto es $b_{\ell-1} > 2^{w-1}$, con $B = 2^w$. En caso contrario:

- Multiplicamos a y b por $2^{w'}$, con $0 \leq w' < w$, obteniendo $a' := a2^{w'}$, $b' := b2^{w'}$, dejando b' normalizada (también se puede hacer un "left-shift").
- Después se calcula q y r tal que $a' := b' + r'$, tomando en cuenta que $q = \lfloor a'/b' \rfloor = \lfloor a/b \rfloor$, y que $r' = r2^{w'}$
- Para recuperar r, dividimos entre $2^{w'}$ por división entera, o con un "right shift".

Reducción

En muchos casos, solamente el residuo de una división es lo requerido, por lo que podemos simplificar el algoritmo de la división para estos casos.

A continuación se describen algunos métodos generales

Restoring Division - Algoritmo

Input: Enteros t, n
Ouput: $R \leftarrow t \bmod n$

$R_0 \leftarrow t$
$n \leftarrow 2^k n$
for $i \leftarrow 1$ **to** k **do**
  $R_i \leftarrow R_{i-1} - n$
  if $R_i < 0$ **then** $R_i \leftarrow R_{i-1}$ **end if**
  $n \leftarrow n/2$
end for
return R_k

Restoring Division - Ejemplo

- $t = 3019 = (101111001011)_2$
- $n = 53 = (110101)_2$

R_0	101111	001011	t
n	110101	000000	Sustracción
	-000110		Negativo
R_1	101111	001011	Restaurar
$n/2$	11010	100000	Sustracción
	$+10100$		Positivo
R_2	10100	101011	No restaurar
$n/2$	1101	010000	Sustraer
	$+0111$		Positivo
R_3	0111	011011	No restaurar
$n/2$	110	101000	Sustraer

Restoring Division - Ejemplo

	$+000$		Positivo
R_4	$+000$	001011	No restaurar
$n/2$	11	010100	
$n/2$	1	101010	
$n/2$	0	110101	Sustraer
		-000010	Negativo
R_5		110011	Restore
R		110011	Final

Non-restoring division

La división con non-restoring permite residuos negativos en la operación:

- Suponga que $R_i := R_{i-1} - n < 0$, en el caso del *restoring algorithm* $R_i \leftarrow R_{i-1}$, y se ejecuta la sustracción con la n rotada a la derecha, obteniendo $R_{i+1} := R_i - n/2 = R_{i-1} - n/2$
- Sin embargo, en este algoritmo, la R_i permanece negativa y le agrega el valor de n rotado en la siguiente iteración:

$$R_{i+1} := R_i + n/2 = (R_{i-1} - n) + n/2 = R_{i-1} - n/2$$

Non-restoring Division - Algoritmo

Input: Enteros t, n
Ouput: $R \leftarrow t \bmod n$

$R_0 \leftarrow t$
$n \leftarrow 2^k n$
for $i \leftarrow 1$ **to** k **do**
  if $R_{i-1} < 0$ **then**
    $R_i \leftarrow R_{i-1} - n$
  else
    $R_i \leftarrow R_{i-1} + n$
  end if
  $n \leftarrow n/2$
end for
return R_k

Nota: puede ser necesario corregir el último R_i si queda negativo. Además, hay que tomar en cuenta el almacenamiento de números negativos (complemento a 2)

Non-restoring Division - Ejemplo

- $t = 3019 = (101111001011)_2$
- $n = 53 = (110101)_2$

R_0	101111	001011	t
n	110101	000000	Sustracción
	1111010		Negativo
$n/2$	011010	100000	Adición
R_2	010100	100000	Positivo
$n/2$	1101	010000	Sustracción
R_3	0111	010000	Positivo
$n/2$	110	101000	Sustracción
R_4	0000	000110	Positivo
$n/2$	011	010100	
$n/2$	01	101010	

Non-restoring Division - Ejemplo

$n/2$	0	110101	Sustracción
R_5	1	101010	Negativo!
n	0	110101	Adición (restore)
R		110011	Fin

Reducción de Barret

Si u y N son números reales, existe otro método para dividir U por N.

- Calcule el inverso de N a una precisión suficiente (método de Newton)
- Multiplíquelo por u

Tal precómputo de N^{-1} es útil si se tienen que hacer varias reducciones.

Existe un método similar para los enteros

Cinvestav

Reducción de Barret

Sea N un entero de n-palabra. Se define el *entero recíproco* de N como

$$R(N) = \lfloor b^{2n}/N \rfloor$$

Si u es un entero de $2n$-palabra, tenemos que

$$q = \left\lfloor \frac{u}{N} \right\rfloor = \left\lfloor \frac{\frac{u}{b^{n-1}} \frac{b^{2n}}{N}}{b^{n+1}} \right\rfloor$$

Lo cual puede ser aproximado a:

$$\hat{q} = \left\lfloor \frac{\lfloor \frac{u}{b^{n-1}} \rfloor R}{b^{n+1}} \right\rfloor$$

Reducción de Barret - Nota

- Adicionalmente $q - 2 \leq \hat{q} \leq q$, y $\hat{q} = q$ en el 90% de los casos.
- Para una reducción modular esta aproximación es suficiente, ya que $\hat{u} = u - \hat{q}N$ está en la misma clase residual módulo N ya que u será mínima si $\hat{q} = q$. Si $\hat{u} > N$, entonces se necesitan a lo más dos reducciones módulo N para obtener el mínimo representativo.

Reducción de Barret - Algoritmo

Input: $u = (u_{2n-1}, \ldots, u_0)$, $N = (N_{n-1}, \ldots, N_0)$.
$R = \lfloor b^{2n}/N \rfloor$ precalculado
Ouput: $r = (r_{n-1}, \ldots r_0)$, s.t. $u \equiv r \bmod N$
$\hat{q} = \left\lfloor \lfloor (u/b^{n-1}) \rfloor R/b^{n+1} \right\rfloor$
$r_1 \leftarrow u \bmod b^{n+1}$, $r_2 \leftarrow (\hat{q}N \bmod b^{n+1}$, $r \leftarrow r_1 - r_2$
if $r < 0$ **then** $r \leftarrow r + b^{n+1}$ **end if**
while $r \geq N$ **do** $r \leftarrow r - N$ **end while**
return r

Cinvestav

Reducción de Barret - Ejemplo

- Sea $B = 4$, $k = 3$, $x = (313221)_B$, y $p = (233)_B$ ($x = 3561$, $p = 47$)
- $\mu = \lfloor 46/p \rfloor = 87 = (1113)_B$
- $\lfloor x/b^{k-1} \rfloor = \lfloor (313221)_B/4^2 \rfloor = (3132)_B$
- $\lfloor x/b^{k-1} \rfloor \cdot \mu = (3132)_B \cdot (1113)_B = (10231302)_B$
- $q = (1023)_B$
- $r_1 = (3221)_B$
- $r_2 = (1023)_B \cdot (233)_B \bmod B^4 = (3011)_B$
- $r = r_1 - r_2 = (210)_B$
- Por lo que $x \bmod p = (210)_B = 36$

Square-and-Multiply

- La manera más sencilla de realizar (implementar) al exponenciación de α^e es multiplicando α repetidamente.
- Para valores pequeños de e esto es válido; sin embargo, un método más rápido es el de *repeated square-and-multiply*.
- El método se basa en la siguiente observación: Sea $e = (b_{\ell-1}, \ldots, b_0)$, para $i \in \{0, \ldots, \ell\}$ definimos $e_i \leftarrow \lfloor e/2^i \rfloor$, donde $e_i = (b_{\ell-1}, \ldots, b_i)_2$. También definimos $\beta_i \leftarrow \alpha^{e_i}$ para $i = \{0, \ldots, \ell)$, con $\beta_\ell = 1$ y $\beta_0 = \alpha^e$, así que tenemos:

$$e_i = 2e_{i+1} + b_i, \text{y} \beta_i = \beta_{i+2}^2 \alpha^{b_i} \text{para} i = \{0, \ldots, \ell - 1\}$$

Square-and-Multiply - Algoritmo

Input: $\alpha \in \mathbb{Z}_n$, e entero no negativo
Ouput: $\beta = \alpha^e$

$\beta \leftarrow [1]_n$
for $i \leftarrow \ell - 1$ down to 0 **do**
  $\beta \leftarrow \beta^2$
  if $b_i = 1$ **then**
    $\beta \leftarrow \beta \cdot \alpha$
  end if
end for
return β

Square-and-Multiply - Ejemplo

- Suponga $e = 37 = (100101)_2$

$$
\begin{array}{ll}
\beta \leftarrow [1] & \\
\beta \leftarrow \beta^2, \beta \leftarrow \beta \cdot \alpha & 1 \\
\beta \leftarrow \beta^2 & 10 \\
\beta \leftarrow \beta^2 & 100 \\
\beta \leftarrow \beta^2, \beta \leftarrow \beta \cdot \alpha & 1001 \\
\beta \leftarrow \beta^2 & 10010 \\
\beta \leftarrow \beta^2, \beta \leftarrow \beta \cdot \alpha & 100101
\end{array}
$$

Ejercicio

- El mismo método se puede utilizar para la multiplicación, modifique el algoritmo.

Método de Montgomery

- Al igual que en la reducción de Barrett, la estrategia en el método de Montgomery es la de reemplazar la división en la reducción clásica con operaciones menos costosas.
- El método no es eficiente cuando se desea realizar una sola multiplicación, pero es eficiente cuando se utiliza varias veces, por ejemplo, en la exponenciación.

Método de Montgomery

- Se reemplazan las divisiones de n por divisiones de $r = 2^k$. Si n es un entero de k-bits, entonces $2^{k-1} < n < 2^k$.
- Asignamos $r = 2^k$, y se hace un mapeo de $a \in [0, n-1]$ a $\tilde{a} \in [0, n-1]$ utilizando el mapa 1-a-1: $\tilde{a} := a \cdot r \bmod n$
- Llamamos $\tilde{a}$ el n-residuo de a.

Método de Montgomery - Multiplicación de Montgomery

- Definimos el *Producto de Montgomery* de dos n-residuos como:

$$\text{MonPro}(\tilde{a}, \tilde{b}) = \tilde{a} \cdot \tilde{b} \cdot r^{-1} \bmod n$$

- Además, se resuelven la siguiente ecuación utilizando el algoritmo extendido de Euclides:

$$r \cdot r^{-1} - n \cdot n' = 1$$

Exponenciación de Montgomery - Algoritmo

Input: $\tilde{a}, \tilde{b}, r, n'$
Ouput: $\tilde{a} \cdot \tilde{b} \cdot r^{-1} \bmod n$

$t \leftarrow \tilde{a} \cdot \tilde{b}$
$m \leftarrow t \cdot n' \bmod r$
$u \leftarrow (t + m \cdot n)/r$
if $u \geq n$ **then**
  return $(u - n)$
else
  return u
end if

Cinvestav

Exponenciación de Montgomery - Algoritmo

Input: Base M, exponente e, módulo n, $r = 2^k$, n' en $rr^{-1} - nn' = 1$

Ouput: $M^e \bmod n$

$\tilde{M} \leftarrow M \cdot r \bmod n$

$\tilde{c} \leftarrow 1 \cdot r \bmod n$

for $i \leftarrow k-1$ down to 0 **do**

  $\tilde{c} \leftarrow \text{MonPro}(\tilde{c}, \tilde{c})$

  if $e_i = 1$ **then** $\tilde{c} \leftarrow \text{MonPro}(\tilde{c}, \tilde{M})$ **end if**

end for

return $\text{MonPro}(\tilde{c}, 1)$

Exponenciación de Montgomery - Ejemplo

Calcule $7^{10} \bmod 13$

- $e = (1010)_2$
- $r = 2^k = 16$
- $16 \cdot 9 - 13 \cdot 11 = 1$:
  - $r^{-1} = 9$
  - $n' = 11$
- $M = 7$, así que $\tilde{M} = C \cdot r \bmod n = 7 \cdot 16 \bmod 13 = 8$
- $c = 1$, así que $\tilde{M} = C \cdot r \bmod n = 1 \cdot 16 \bmod 13 = 3$
- $\tilde{M} = 8$
- $\tilde{c} = 3$

Exponenciación de Montgomery - Ejemplo

e_i	Square	Multiply
1	MonPro$(3,3) = 3$	MonPro$(8,3) = 8$
0	MonPro$(8,8) = 4$	
1	MonPro$(4,4) = 1$	MonPro$(8,1) = 7$
0	MonPro$(7,7) = 12$	
	Recover	ModPro$(12,1) = 4$

Exponenciación de Montgomery - Ejemplo

- Cálculo de MonPro$(3, 3)$
  - $t \leftarrow 3 \cdot 3 = 9$
  - $m \leftarrow 9 \cdot 11 \bmod 16 = 3$
  - $u \leftarrow (9 + 3 \cdot 13)/16 = 48/16 = 3$

- Cálculo de MonPro$(8, 1)$
  - $t \leftarrow 8 \cdot 1 = 8$
  - $m \leftarrow 8 \cdot 11 \bmod 16 = 8$
  - $u \leftarrow (8 + 8 \cdot 13)/16 = 112/16 = 7$

Cinvestav

Inversión simultánea (Montgomery)

Si se requiere obtener el inverso de varios números, por ejemplo: x, y, ¿cómo obtener un ahorro?

- Basándonos en la observación de que: $\frac{1}{x} = y\frac{1}{xy}$
- Podemos calcular $\frac{1}{x}$ y $\frac{1}{y}$ utilizando solamente 1 inversión
- Se puede generalizar el algoritmo a n elementos, siendo su costo: 1 inversión y $3(n-1)$ multiplicaciones.

Inversión simultánea (Montgomery) - Ejercicio

- Desarrolle el algoritmo