Number-Theoretic Algorithms

Hengfeng Wei

hfwei@nju.edu.cn

March 31 $\sim$ April 4, 2017

Number-Theoretic Algorithms

Cancellation in modular arithmetic

(TC 31.4.2)

$$ad \equiv bd \pmod{n} \not\Rightarrow a \equiv b \pmod{n}$$

Cancellation in modular arithmetic

(TC 31.4.2)

$$ad \equiv bd \pmod{n} \not\Rightarrow a \equiv b \pmod{n}$$

$$3 \cdot 2 \equiv 5 \cdot 2 \pmod{4} \quad 3 \not\equiv 5 \pmod{4}$$

Cancellation in modular arithmetic

(TC 31.4.2)

$$ad \equiv bd \pmod{n} \not\Longrightarrow a \equiv b \pmod{n}$$

$$ad \equiv bd \pmod{n}, a \perp n \implies a \equiv b \pmod{n}$$

$$3 \cdot 2 \equiv 5 \cdot 2 \pmod{4} \quad 3 \not\equiv 5 \pmod{4}$$

Changing the modulus

$$3 \cdot 2 \equiv 5 \cdot 2 \pmod 4 \quad 3 \not\equiv 5 \pmod 4 \quad 3 \equiv 5 \pmod 2$$

Changing the modulus

$$3 \cdot 2 \equiv 5 \cdot 2 \pmod 4 \quad 3 \not\equiv 5 \pmod 4 \quad 3 \equiv 5 \pmod 2$$

$$ad \equiv bd \pmod{nd} \iff a \equiv b \pmod n \quad (d \neq 0)$$

Changing the modulus

$$3 \cdot 2 \equiv 5 \cdot 2 \pmod 4 \quad 3 \not\equiv 5 \pmod 4 \quad 3 \equiv 5 \pmod 2$$

$$ad \equiv bd \pmod{nd} \iff a \equiv b \pmod n \quad (d \neq 0)$$

$$ad \equiv bd \pmod n \iff a \equiv b \pmod{\frac{n}{(d,n)}}$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

$$a \equiv b \pmod{n} \implies a \equiv b \pmod{n_i}$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

$$a \equiv b \pmod{n} \implies a \equiv b \pmod{n_i}$$

$$a \equiv b \pmod{100} \implies a \equiv b \pmod{20} \implies a \equiv b \pmod{5}$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

$$a \equiv b \pmod{n_1}, a \equiv b \pmod{n_2} \iff a \equiv b \pmod{[n_1, n_2]}$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

$$a \equiv b \pmod{n_1}, a \equiv b \pmod{n_2} \iff a \equiv b \pmod{[n_1, n_2]}$$

$$a \equiv b \pmod{n_1}, a \equiv b \pmod{n_2} \iff a \equiv b \pmod{n_1 n_2}, \text{ if } n_1 \perp n_2$$

Changing the modulus

$$n = n_1 n_2 \cdots n_k$$

$$a \equiv b \pmod{n_1}, a \equiv b \pmod{n_2} \iff a \equiv b \pmod{[n_1, n_2]}$$

$$a \equiv b \pmod{n_1}, a \equiv b \pmod{n_2} \iff a \equiv b \pmod{n_1 n_2}, \text{ if } n_1 \perp n_2$$

$$\forall 1 \le i \le k, a \equiv b \pmod{n_i} \iff a \equiv b \pmod{n}, \text{ if } n_i \perp n_j$$

Number-Theoretic Algorithms

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

1. If $a > b \geq 0$, $\textsc{Euclid}(a, b)$ makes $\leq r \triangleq 1 + \log_\phi b$ recursive calls.

$$a > b \geq 1, b < F_{k+1} \implies r < k.$$

$$r \leq 1 + \log_\phi b \implies k = 2 + \log_\phi b, b < F_{3+\log_\phi b}$$

$$F_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}} = [\frac{\phi^k}{\sqrt{5}}] \geq \frac{\phi^k}{\sqrt{5}}$$

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

1. If $a > b \geq 0$, $\textsc{Euclid}(a, b)$ makes $\leq r \triangleq 1 + \log_\phi b$ recursive calls.

$$a > b \geq 1, b < F_{k+1} \implies r < k.$$

$$r \leq 1 + \log_\phi b \implies k = 2 + \log_\phi b, b < \boxed{?} \leq F_{3+\log_\phi b}$$

$$F_k = \frac{\phi^k - \hat{\phi}^k}{\sqrt{5}} = [\frac{\phi^k}{\sqrt{5}}] \geq \frac{\phi^k}{\sqrt{5}}$$

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

2. Improve this bound to $1 + \log_{\phi}(\frac{b}{(a,b)})$.

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

2. Improve this bound to $1 + \log_\phi(\frac{b}{(a,b)})$.

$$(a, b) = (a, b) \cdot (\frac{a}{(a, b)}, \frac{b}{(a, b)})$$

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

2. Improve this bound to $1 + \log_\phi(\frac{b}{(a,b)})$.

$$(a, b) = (a, b) \cdot (\frac{a}{(a, b)}, \frac{b}{(a, b)})$$

$$\begin{aligned}&(16, 12)\\ &= (12, 4)\\ &= (4, 0)\\ &= 4\end{aligned} \qquad \begin{aligned}&(4, 3)\\ &= (3, 1)\\ &= (1, 0)\\ &= 1\end{aligned}$$

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

2. Improve this bound to $1 + \log_\phi(\frac{b}{(a,b)})$.

$$(a,b) = (a,b) \cdot (\frac{a}{(a,b)}, \frac{b}{(a,b)})$$

$$\begin{aligned} &(16,12) \\ &= (12,4) \\ &= (4,0) \\ &= 4 \end{aligned} \qquad \begin{aligned} &(4,3) \\ &= (3,1) \\ &= (1,0) \\ &= 1 \end{aligned}$$

$$\textsc{Euclid}(a,b) \leftrightarrow \textsc{Euclid}(\frac{a}{(a,b)}, \frac{b}{(a,b)})$$

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

2. Improve this bound to $1 + \log_{\phi}(\frac{b}{(a,b)})$.

$$\textsc{Euclid}(a, b) \leftrightarrow \textsc{Euclid}(\frac{a}{(a, b)}, \frac{b}{(a, b)})$$

$$\textsc{Euclid}(b, a \bmod b) \leftrightarrow \textsc{Euclid}(\frac{b}{(a, b)}, \frac{a}{(a, b)} \bmod \frac{b}{(a, b)})$$

$$\frac{a}{(a, b)} \bmod \frac{b}{(a, b)} = \frac{a \bmod b}{(a, b)}$$

Worst-case analysis of Euclid's algorithm

(TC 31.2–5)

2. Improve this bound to $1 + \log_{\phi}(\frac{b}{(a,b)})$.

Lemma (Generalization of Lemma 31.10)

If $a > b \geq 1, d = (a, b)$ *and* $\textsc{Euclid}(a, b)$ *performs* $k \geq 1$ *recursive calls, then* $a \geq dF_{k+2}$ *and* $b \geq dF_{k+1}$.

Average-case analysis of Euclid's algorithm

$$T(m, 0) = 0; \quad T(m, n) = 1 + T(n, m \bmod n) \; n \geq 1$$

Average-case analysis of Euclid's algorithm

$$T(m, 0) = 0; \quad T(m, n) = 1 + T(n, m \bmod n) \; n \geq 1$$

When m is chosen at random:

$$T_n = \frac{1}{n} \sum_{0 \leq k < n} T(k, n)$$

Average-case analysis of Euclid's algorithm

$$T(m,0) = 0; \quad T(m,n) = 1 + T(n, m \bmod n) \; n \geq 1$$

When m is chosen at random:

$$T_n = \frac{1}{n} \sum_{0 \leq k < n} T(k,n)$$

Assume that, for $0 \leq k < n$, $(n \bmod k)$ is "random":

$$T_n \approx 1 + \frac{1}{n}(T_0 + T_1 + \cdots + T_{n-1})$$

Average-case analysis of Euclid's algorithm

$$T(m, 0) = 0; \quad T(m, n) = 1 + T(n, m \bmod n) \; n \geq 1$$

When m is chosen at random:

$$T_n = \frac{1}{n} \sum_{0 \leq k < n} T(k, n)$$

Assume that, for $0 \leq k < n$, $(n \bmod k)$ is "random":

$$T_n \approx 1 + \frac{1}{n}(T_0 + T_1 + \cdots + T_{n-1}) = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = H_n \approx \ln n + O(1)$$

Average-case analysis of Euclid's algorithm

$$T(m,0) = 0; \quad T(m,n) = 1 + T(n, m \bmod n) \; n \geq 1$$

When m is chosen at random:

$$T_n = \frac{1}{n} \sum_{0 \leq k < n} T(k,n)$$

Assume that, for $0 \leq k < n$, $(n \bmod k)$ is "random":

$$T_n \approx 1 + \frac{1}{n}(T_0 + T_1 + \cdots + T_{n-1}) = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = H_n \approx \ln n + O(1)$$

Reference

"The Art of Computer Programming, Vol 2: Seminumerical Algorithms (Section 4.5.3)" by Donald E. Knuth, 3rd edition.

Number-Theoretic Algorithms

Pairwise relatively prime

(TC 31.2–9)

$$n_1, n_2, n_3, n_4 \text{ are pairwise relatively prime}$$

$$\iff$$

$$\gcd(n_1 n_2, n_3 n_4) = \gcd(n_1 n_3, n_2 n_4) = 1$$

Pairwise relatively prime

(TC 31.2–9)

$n_1, n_2, \ldots, n_k$ are pairwise relatively prime

$$\iff$$

a set of $\lceil \lg k \rceil$ pairs of numbers derived from the n_i are relatively prime.

Pairwise relatively prime

(TC 31.2–9)

$n_1, n_2, \ldots, n_k$ are pairwise relatively prime

$$\Longleftrightarrow$$

a set of $\lceil \lg k \rceil$ pairs of numbers derived from the n_i are relatively prime.

$$\binom{k}{2} = \Theta(k^2) \quad \text{(complete graph)}$$

Pairwise relatively prime

(TC 31.2–9)

$n_1, n_2, \ldots, n_k$ are pairwise relatively prime

$$\iff$$

a set of $\lceil \lg k \rceil$ pairs of numbers derived from the n_i are relatively prime.

$$\binom{k}{2} = \Theta(k^2) \quad \text{(complete graph)}$$

$$\gcd(\boxed{1_L}, \boxed{1_R}) = \gcd(\boxed{2_L}, \boxed{2_R}) = \cdots = \gcd(\boxed{\lceil \lg k \rceil_L}, \boxed{\lceil \lg k \rceil_R}) = 1$$

Pairwise relatively prime

(TC 31.2–9)

$$n_1, n_2, \ldots, n_k \text{ are pairwise relatively prime}$$

$$\iff$$

a set of $\lceil \lg k \rceil$ pairs of numbers derived from the n_i are relatively prime.

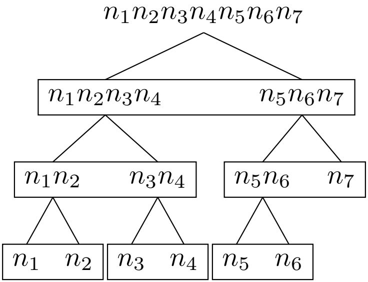
$$\binom{k}{2} = \Theta(k^2) \quad \text{(complete graph)}$$

$$\gcd(\boxed{1_L}, \boxed{1_R}) = \gcd(\boxed{2_L}, \boxed{2_R}) = \cdots = \gcd(\boxed{\lceil \lg k \rceil_L}, \boxed{\lceil \lg k \rceil_R}) = 1$$

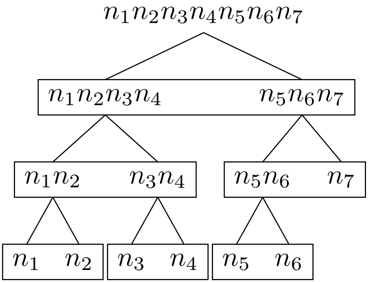
$$k = 3: \quad \gcd(n_1, n_2 n_3) = \gcd(n_2, n_3) = 1$$

$$k = 2: \quad \gcd(n_1, n_2) = 1$$

Pairwise relatively prime: divide-and-conquer

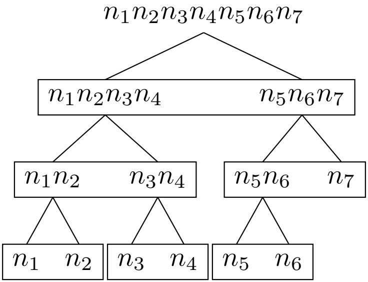


Pairwise relatively prime: divide-and-conquer



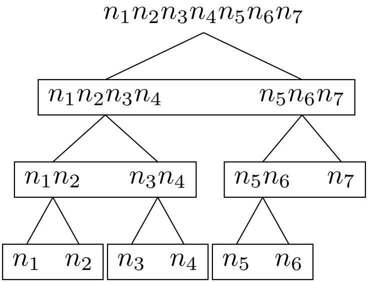
$$\begin{cases} T(1) = 0 \\ T(k) = 2T(\frac{k}{2}) + 1 \end{cases}$$

Pairwise relatively prime: divide-and-conquer



$$\begin{cases} T(1) = 0 \\ T(k) = 2T(\frac{k}{2}) + 1 \end{cases} \implies T(k) = k - 1$$

Pairwise relatively prime: divide-and-conquer



$$\begin{cases} T(1) = 0 \\ T(k) = 2T(\frac{k}{2}) + 1 \end{cases} \implies T(k) = k - 1$$

$$T_k = k - 1 : (n_i, n_{i+1} n_{i+2} \cdots n_k) \quad \forall 1 \leq i < k$$

Pairwise relatively prime: smarter combination

$$\begin{cases} T(1) = 0 \\ T(k) = T(\frac{k}{2}) + 1 \end{cases}$$

Pairwise relatively prime: smarter combination

$$\begin{cases} T(1) = 0 \\ T(k) = T(\frac{k}{2}) + 1 \end{cases} \implies T(k) = \lceil \lg k \rceil$$

Pairwise relatively prime: the dividing pattern

$$k = 7 : \quad n_0, n_1, n_2, \ldots, n_6$$

$$
\begin{array}{c}
000 \\
001 \\
010 \\
011 \\
100 \\
101 \\
110
\end{array}
$$

Pairwise relatively prime: the dividing pattern

$$k = 7 : \quad n_0, n_1, n_2, \ldots, n_6$$

$$\begin{array}{c} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \end{array}$$

$$T(k) = \lceil \lg k \rceil$$

Can we do even better?

$$T(k) \geq \lceil \lg k \rceil$$

Can we do even better?

$$T(k) \ge \lceil \lg k \rceil$$

Prove by (strong) mathematical induction.

Can we do even better?

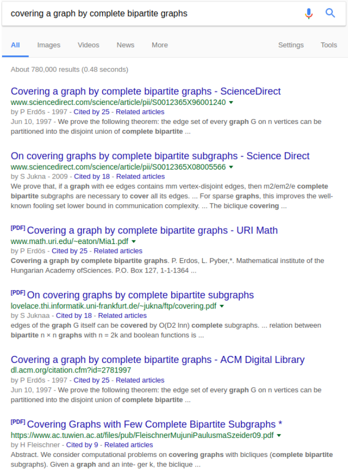
$$T(k) \geq \lceil \lg k \rceil$$

Prove by (strong) mathematical induction.

$$\begin{aligned} T(k) &\geq 1 + T(\lceil \frac{k}{2} \rceil) \\ &\geq 1 + \lceil \lg \lceil \frac{k}{2} \rceil \rceil \\ &= \lceil \lg k \rceil \end{aligned}$$

Biclique covering

Covering a complete graph with few complete bipartite subgraphs.



Biclique covering: rethinking the first divide-and-conquer

$$T(k) = k - 1$$

Biclique covering: rethinking the first divide-and-conquer

$$T(k) = k - 1$$

edge-disjoint biclique partition

Biclique covering: rethinking the first divide-and-conquer

$$T(k) = k - 1$$

edge-disjoint biclique partition

Reference for $T(k) \geq k - 1$

"On the Addressing Problem for Loop Switching" by Graham and Pollak, 1971.

Biclique covering: rethinking the first divide-and-conquer

$$T(k) = k - 1$$

edge-disjoint biclique partition

Reference for $T(k) \geq k - 1$

"On the Addressing Problem for Loop Switching" by Graham and Pollak, 1971.

Reference for *weighted* biclique partition

"Covering a Graph by Complete Bipartite Graphs" by P. Erdős and L. Pyber, 1997.

Number-Theoretic Algorithms

Chinese Remainder Theorem (CRT)

Theorem (CRT)

$$n_1, \ldots, n_k; \quad a_1, \ldots, a_k$$

$$n_i \perp n_j \quad i \neq j, \quad n = n_1 n_2 \cdots n_k$$

$$\exists! a \ (0 \leq a < n) : a \equiv a_i \pmod{n_i}.$$

Proof for uniqueness.

$$a \equiv a' \pmod{n_i} \implies n \mid a - a'.$$

□

History of CRT

Proof of CRT (1)

Nonconstructive proof.

$$f : [0, n) \to \prod_{1 \le i \le k} [0, a_i)$$

$$f : a \mapsto (a \bmod n_1, \ldots, a \bmod n_k)$$

- f is one-to-one.
- f is onto.

$$\exists a : f(a) = (a_1, \ldots, a_k).$$

$\square$

Proof of CRT (2)

Constructive proof by induction.

$$a \equiv a_1 \pmod{n_1} \tag{1}$$
$$a \equiv a_2 \pmod{n_2} \tag{2}$$

Proof of CRT (2)

Constructive proof by induction.

$$a \equiv a_1 \pmod{n_1} \tag{1}$$
$$a \equiv a_2 \pmod{n_2} \tag{2}$$

$$(1) \implies a = a_1 + n_1 y$$

□

Proof of CRT (3)

Constructive proof by induction.

$$a \equiv a_1 \pmod{n_1} \tag{3}$$

$$a \equiv a_2 \pmod{n_2} \tag{4}$$

$$n_1 \perp n_2 \implies n_1 n_1' + n_2 n_2' = 1$$

Proof of CRT (3)

Constructive proof by induction.

$$a \equiv a_1 \pmod{n_1} \tag{3}$$
$$a \equiv a_2 \pmod{n_2} \tag{4}$$

$$n_1 \perp n_2 \implies n_1 n_1' + n_2 n_2' = 1$$

$$x = a_1 n_1 n_1' + a_2 n_2 n_2' \pmod{n_1 n_2}$$

□

Proof of CRT (4)

Constructive proof.

1. $x \equiv 1 \pmod{n_i}, \quad x \equiv 0 \pmod{n_j}\ (i \neq j)$

$$x = M_i(M_i^{-1} \bmod n_i) \implies x = M_i M_i^{-1} \pmod{n}$$

□

Proof of CRT (4)

Constructive proof.

1. $x \equiv 1 \pmod{n_i}, \quad x \equiv 0 \pmod{n_j}\ (i \neq j)$

$$x = M_i(M_i^{-1} \bmod n_i) \implies x = M_i M_i^{-1} \pmod{n}$$

2. $x \equiv a_i \pmod{n_i}, \quad x \equiv 0 \pmod{n_j}\ (i \neq j)$

$$x = a_i M_i M_i^{-1} \pmod{n}$$

Proof of CRT (4)

Constructive proof.

1. $x \equiv 1 \pmod{n_i}, \quad x \equiv 0 \pmod{n_j}\ (i \neq j)$

$$x = M_i(M_i^{-1} \bmod n_i) \implies x = M_i M_i^{-1} \pmod{n}$$

2. $x \equiv a_i \pmod{n_i}, \quad x \equiv 0 \pmod{n_j}\ (i \neq j)$

$$x = a_i M_i M_i^{-1} \pmod{n}$$

3. $a \equiv a_i \pmod{n_i}, \forall 1 \leq i \leq k$

$$a = \sum_{1 \leq i \leq k} a_i M_i M_i^{-1} \pmod{n}$$

□

Proof of CRT (5)

More efficient constructive proof.

Reference

"The Residue Number System" by Garner, 1959.

Reference

"The Art of Computer Programming, Vol 2: Seminumerical Algorithms (Section 4.3.2)" by Donald E. Knuth, 3rd edition.

□

Operations over CRT

$$a \leftrightarrow (a_1, a_2, \ldots, a_n)$$

$$a \pm b \leftrightarrow (a_1 \pm b_1, a_2 \pm b_2, \ldots, a_n \pm b_n)$$
$$a \times b \leftrightarrow (a_1 \times b_1, a_2 \times b_2, \ldots, a_n \times b_n)$$

Operations over CRT

$$a \leftrightarrow (a_1, a_2, \ldots, a_n)$$

$$a \pm b \leftrightarrow (a_1 \pm b_1, a_2 \pm b_2, \ldots, a_n \pm b_n)$$
$$a \times b \leftrightarrow (a_1 \times b_1, a_2 \times b_2, \ldots, a_n \times b_n)$$

TC 31.5–3

$$a \leftrightarrow (a_1, a_2, \ldots, a_n), (a, n) = 1 \implies a^{-1} \leftrightarrow (a_1^{-1}, a_2^{-2}, \ldots, a_n^{-1})$$

Operations over CRT

$$a \leftrightarrow (a_1, a_2, \dots, a_n)$$

$$a \pm b \leftrightarrow (a_1 \pm b_1, a_2 \pm b_2, \dots, a_n \pm b_n)$$
$$a \times b \leftrightarrow (a_1 \times b_1, a_2 \times b_2, \dots, a_n \times b_n)$$

TC 31.5–3

$$a \leftrightarrow (a_1, a_2, \dots, a_n), (a, n) = 1 \implies a^{-1} \leftrightarrow (a_1^{-1}, a_2^{-2}, \dots, a_n^{-1})$$

Proof.

$$a^{-1} \equiv a_i^{-1} \pmod{n_i}$$

Operations over CRT

$$a \leftrightarrow (a_1, a_2, \dots, a_n)$$

$$a \pm b \leftrightarrow (a_1 \pm b_1, a_2 \pm b_2, \dots, a_n \pm b_n)$$
$$a \times b \leftrightarrow (a_1 \times b_1, a_2 \times b_2, \dots, a_n \times b_n)$$

TC 31.5–3

$$a \leftrightarrow (a_1, a_2, \dots, a_n), (a, n) = 1 \implies a^{-1} \leftrightarrow (a_1^{-1}, a_2^{-2}, \dots, a_n^{-1})$$

Proof.

$$a^{-1} \equiv a_i^{-1} \pmod{n_i} \impliedby \begin{cases} a \equiv a_i \pmod{n_i} \\ (a, n) = 1 \end{cases}$$

□

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

$$m \perp n \implies \varphi(mn) = \varphi(m)\varphi(n)$$

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

$$m \perp n \implies \varphi(mn) = \varphi(m)\varphi(n)$$

$$\varphi(n) = \prod_{i=1}^{r} \varphi(p_i^{k_i})$$

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

$$m \perp n \implies \varphi(mn) = \varphi(m)\varphi(n)$$

$$\varphi(n) = \prod_{i=1}^{r} \varphi(p_i^{k_i}) = \prod_{i=1}^{r} (p_i^{k_i} - p_i^{k_i - 1})$$

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

$$m \perp n \implies \varphi(mn) = \varphi(m)\varphi(n)$$

$$\varphi(n) = \prod_{i=1}^{r} \varphi(p_i^{k_i}) = \prod_{i=1}^{r} (p_i^{k_i} - p_i^{k_i - 1}) = \prod_{i=1}^{r} p_i^{k_i} (1 - \frac{1}{p_i})$$

The φ function

Theorem (The φ function)

$$\varphi(p) = p - 1$$
$$\varphi(p^k) = p^k - p^{k-1}$$

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

$$m \perp n \implies \varphi(mn) = \varphi(m)\varphi(n)$$

$$\varphi(n) = \prod_{i=1}^{r} \varphi(p_i^{k_i}) = \prod_{i=1}^{r} (p_i^{k_i} - p_i^{k_i - 1}) = \prod_{i=1}^{r} p_i^{k_i} (1 - \frac{1}{p_i}) = n \prod_{i=1}^{r} (1 - \frac{1}{p_i})$$

The φ function

Theorem (The φ function)

$$m \perp n \implies \varphi(mn) = \varphi(m)\varphi(n)$$

Proof.

$$U_m = \{a \bmod m, (a, m) = 1\}, U_n = \{a \bmod n, (a, n) = 1\},$$
$$U_{mn} = \{c \bmod mn, (c, mn) = 1\}$$

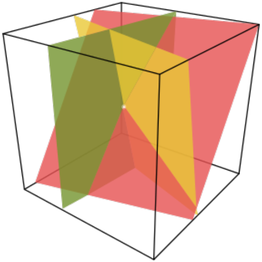
$$f : U_{mn} \to U_m \times U_n$$
$$f(c \bmod mn) = (c \bmod m, c \bmod n).$$

□

Secret sharing using the CRT

Definition ((k, n)-threshold secret sharing scheme)

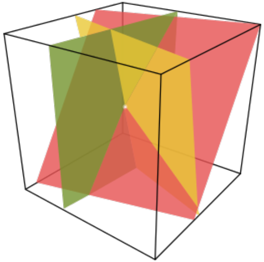
$(2, 3)$-secret sharing:



Secret sharing using the CRT

Definition ((k, n)-threshold secret sharing scheme)

$(2, 3)$-secret sharing:



Reference

"How to Share a Secret" by Mignotte, 1982.

Secret sharing using the CRT

1. Choose m_i:

$$m_1 < m_2 < \cdots < m_n, \quad m_i \perp m_j, \quad \prod_{i=n-k+2}^{n} m_i < \prod_{i=1}^{k} m_i$$

Secret sharing using the CRT

1. Choose m_i:

$$m_1 < m_2 < \cdots < m_n, \quad m_i \perp m_j, \quad \prod_{i=n-k+2}^{n} m_i < \prod_{i=1}^{k} m_i$$

2. Choose the secret S:

$$\prod_{i=n-k+2}^{n} m_i < S < \prod_{i=1}^{k} m_i$$

Secret sharing using the CRT

1. Choose m_i:

$$m_1 < m_2 < \cdots < m_n, \quad m_i \perp m_j, \quad \prod_{i=n-k+2}^{n} m_i < \prod_{i=1}^{k} m_i$$

2. Choose the secret S:

$$\prod_{i=n-k+2}^{n} m_i < S < \prod_{i=1}^{k} m_i$$

3. Compute the shares:

$$s_i = S \bmod m_i$$

Solving the system of congruences

(TC 31.5–2)

$$\begin{cases} x \equiv 1 \pmod 9 \\ x \equiv 2 \pmod 8 \\ x \equiv 3 \pmod 7 \end{cases}$$

Solving the system of congruences

$$19x \equiv 556 \pmod{1155}$$

Solving the system of congruences

CRT with non-pairwise coprime moduli

$$\begin{cases} x \equiv 3 \pmod{8} \\ x \equiv 11 \pmod{20} \\ x \equiv 1 \pmod{15} \end{cases}$$