Lecture 5
Linear Congruences, Chinese Remainder Theorem, Algorithms

Recap - linear congruence $ax \equiv b \mod m$ has solution if and only if $g = (a, m)$ divides b. How do we find these solutions?

Case 1: $g = (a, m) = 1$. Then invert $a \mod m$ to get $x \equiv a^{-1}b \mod m$. Algorithmically, find $ax_0 + my_0 = 1$ with Euclidean Algorithm, then $ax_0 \equiv 1 \mod m$ so $x_0 = a^{-1}$, so $x \equiv x_0 b = a^{-1}b$ solves the congruence. ($ax \equiv a(x_0 b) \equiv (ax_0)b \equiv b \mod m$). Conclusion: There is a unique solution mod m.

Case 2: $g = (a, m) > 1$. If $g \nmid b$, there are no solutions. If $g|b$, write $a = a'g, b = b'g, m = m'g$ so that $ax \equiv b \mod m \Rightarrow a'x = b' \mod m'$ so that (a', m') is now 1. The unique solution (found by Case 1) $x \mod m'$ also satisfied $ax \equiv b \mod m$ so that we have one solution mod m. We know any solution $\tilde{x} \mod m$ must be congruent to $x \mod m'$, so $\tilde{x}$ must have form $x + m'k$ for some k. As k goes from 0 through $g - 1$ we get the g distinct integers mod m: $x, x + m', x + 2m' \ldots x + (g-1)m'$, which all satisfy $a\tilde{x} \equiv b \mod m$ because

$$\begin{aligned} a(x + km') &= ax + akm' \\ &= ax + a'gkm' \\ &= ax + m(a'k) \\ &\equiv ax \pmod m \\ &\equiv b \pmod m \end{aligned}$$

Conclusion: this congruence has $g = (a, m)$ solutions mod m.

Eg.,

$$35x \equiv 14 \pmod{28}$$

$(35, 28) = g = 7$. To solve, first divide through by 7 to get $5x \equiv 2 \mod 4$. Solution of $x \equiv 2 \mod 4$ is $x = 2$, which will also satisfy original congruence. $m' = \frac{28}{7} = 4 \Rightarrow$ all solutions mod $28 \equiv 2, 6, 10, 14, 18, 22, 26$.

Simultaneous System of Congruences to Different Moduli: Given

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k} \end{aligned}$$

Does this system have a common solution? (Not always, eg., $x \equiv 3 \mod 8$ and $x \equiv 1 \mod 12$) In general, need some compatibility conditions.

Theorem 25 (Chinese Remainder Theorem). *If the moduli are coprime in pairs (ie., $(m_i, m_j) = 1$ for $i \neq j$), then the system has a unique solution mod $m_1 m_2 \dots m_k$.*

Proof of Uniqueness. Suppose there are two solutions $x \equiv y \equiv a_1 \mod m_1$, $x \equiv y \equiv a_2 \mod m_2$, etc. Then $m_1 | (x - y)$, $m_2 | (x - y)$, etc. Since m's are relatively prime in pairs, their product $m_1 m_2 \dots m_k$ divides $x - y$ as well, so $x \equiv y \mod m_1 m_2 \dots m_k$. So solution, if exists, must be unique mod $m_1 m_2 \dots m_k$. ■

Proof of Existence. Write solution as a linear combination of a_i

$$A_1 a_1 + A_2 a_2 + \cdots + A_k a_k$$

Want to arrange so that mod a_i all the A_j for $j \neq i$ are $\equiv 0 \mod m$, and $A_i \equiv 1 \mod m_i$. Let

$$\begin{aligned} N_1 &= m_2 m_3 \dots m_k \\ N_2 &= m_1 m_3 \dots m_k \\ &\vdots \\ N_i &= m_1 m_2 \dots m_{i-1} m_{i+1} \dots m_k \end{aligned}$$

So $(N_i, m_i) = 1$, since all the other m are coprime to m_i. Let H_i equal the multiplicative inverse of $N_i \mod m_i$, and let $A_i = H_i N_i$. Then, $A_i \equiv 0 \mod m_j$ for $j \neq i$ and $A_i \equiv 1 \mod m_i$. So now let

$$\begin{aligned} a &= A_1 a_1 + A_2 a_2 + \cdots + A_k a_k \\ &= H_1 N_1 a_1 + H_2 N_2 a_2 + \cdots + H_k N_k a_k \end{aligned}$$

Then if we take mod m_i all the terms except ith term will vanish (since $m_i | N_j$ for $j \neq i$). So

$$\begin{aligned} a &\equiv H_i N_i a_i \pmod{m_i} \\ &\equiv a_i \pmod{m_i} \end{aligned}$$

■

Eg.

$$\begin{aligned} &x \equiv 2 \mod 3, && N_1 = 5 \cdot 7 = 35 \equiv 2 \mod 3, && H_1 = 2 \\ &x \equiv 3 \mod 5, && N_2 = 3 \cdot 7 = 21 \equiv 1 \mod 5, && H_2 = 1 \\ &x \equiv 5 \mod 7, && N_3 = 3 \cdot 5 = 15 \equiv 1 \mod 7, && H_3 = 1 \end{aligned}$$

$$
\begin{aligned}
x &= H_1N_1a_1 + N_2H_2a_2 + N_3H_3a_3 \pmod{m_1m_2m_3}\\
&= 2\cdot 35\cdot 2 + 1\cdot 21\cdot 3 + 1\cdot 15\cdot 5 \pmod{105}\\
&= 278 \pmod{105}\\
&\equiv 68 \pmod{105}
\end{aligned}
$$

Note: Assuming we have $m_1, m_2 \dots m_k$ that are relatively prime, the Chinese Remainder Theorem says that any choice of $a_1 \mod m_1$, $a_2 \mod m_2$, etc. gives rise to particular $x(a_1, a_2, \dots a_k, m_1, \dots m_k) \mod m_1m_2\dots m_k$. Number of choices that we have is $m_1m_2\dots m_k$, which agrees with number of integers mod $m_1m_2\dots m_k$.

Note: Now note that $x(a_1, a_2, \dots a_k, m_1, \dots m_k)$ is coprime to $m_1m_2\dots m_k$ if and only if $(a_i, m_i) = 1$.

- If x is coprime to $\prod m_i$ then it is relatively coprime to each of them, so since $x \equiv a_i \mod m_i$ we'll also have $(a_i, m_i) = 1$.
- Conversely if $(a_i, m_i) = 1$ for all i, then since $x \equiv a_i \mod m_i$, this implies that $(x_i, m_i) = 1$ holds for all i, so $(x, \prod m_i) = 1$ as well.

What is the number of x coprime to $\prod m_i$? (by definition this is $\phi(m_1m_2\dots m_k)$)

$$
\underbrace{(\text{\# of choices of } a_1)}_{\phi(m_1)}\underbrace{(\text{\# of choices of } a_2)}_{\phi(m_1)}\dots
$$

with each a_i coprime to m_i. This gives corollary that if m_i coprime in pairs, $\phi(\prod m_i) = \prod \phi(m_i)$. We can use this to understand $\phi(n)$ for any n. With m_i coprime in pairs,

$$
\begin{aligned}
n &= p_1^{e_1}p_2^{e_2}\dots p_k^{e_k}\\
m_1 &= p_1^{e_1}, \quad m_2 = p_2^{e_2}\dots \quad m_k = p_k^{e_k}\\
\phi(n) &= \phi(p_1^{e_1})\phi(p_2^{e_2})\dots\phi(p_k^{e_k})
\end{aligned}
$$

All we need, then, is how to find $\phi(p^e)$.

$$
\begin{aligned}
\phi(p^e) &= \text{\# of } \{x | 1 \le x \le p^e \text{ and } (x,p) = 1 \text{ and so } (x,p^e) = 1\} \\
&= p^e - p^{e-1} \\
&= p^{e-1}(p-1) \\
&= p^e \left(1 - \frac{1}{p}\right)
\end{aligned}
$$

and so

$$
\begin{aligned}
\phi(n) &= p_1^{e_1-1}(p_1-1)p_2^{e_2-1}(p_2-1)\dots p_k^{e_k-1}(p_k-1) \\
&= p_1^{e_1}p_2^{e_2}\dots p_k^{e_k}\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\dots\left(1-\frac{1}{p_k}\right) \\
&= n\prod_{p|n}\left(1-\frac{1}{p}\right)
\end{aligned}
$$

Numerical Calculations for Algorithms

Want to do arithmetic modulo N (some large number). Benchmark = time to write down N, which is roughly the number of digits of $N = c\log N$ for some constant c.

Addition is $\log N$ steps/time
Multiplication is $\log^2 N$ steps/time in the simplest way

Karatsuba Multiplication This is a faster algorithm for multiplication (see http://en.wikipedia.org/wiki/Karatsuba_algorithm#Algorithm); reduces time to $(\log N)^{\log 3/\log 2}$

Multiplication can be further improved by using Fast Fourier Transforms to $\log N\text{poly}(\log\log n)$.

Exponentiation - we want to compute $a^b \mod N$, with a at most N and b is also small ($\sim N$). Most obvious way would be repeated multiplication for $N\log^2 N$, but better to use repeated squaring. Write b in binary as

$$
\begin{aligned}
b &= b_r b_{r-1}\dots b_0 \\
&= 2^r b_r + 2^{r-1}b_{r-1} + \dots + b_0
\end{aligned}
$$

then compute $a^{2^0}, a^{2^1}, \dots a^{2^r} \mod N$ by repeatedly squaring the previous one (at most $\log^2 N$ for each). Then take

$$
\left(a^{2^0}\right)^{b_0}\left(a^{2^1}\right)^{b_1}\left(a^{2^2}\right)^{b_2}\dots\left(a^{2^r}\right)^{b_r}
$$

for a total of $\log b\log^2 N \sim \log^3 N$ steps.

18.781 Theory of Numbers
Spring 2012