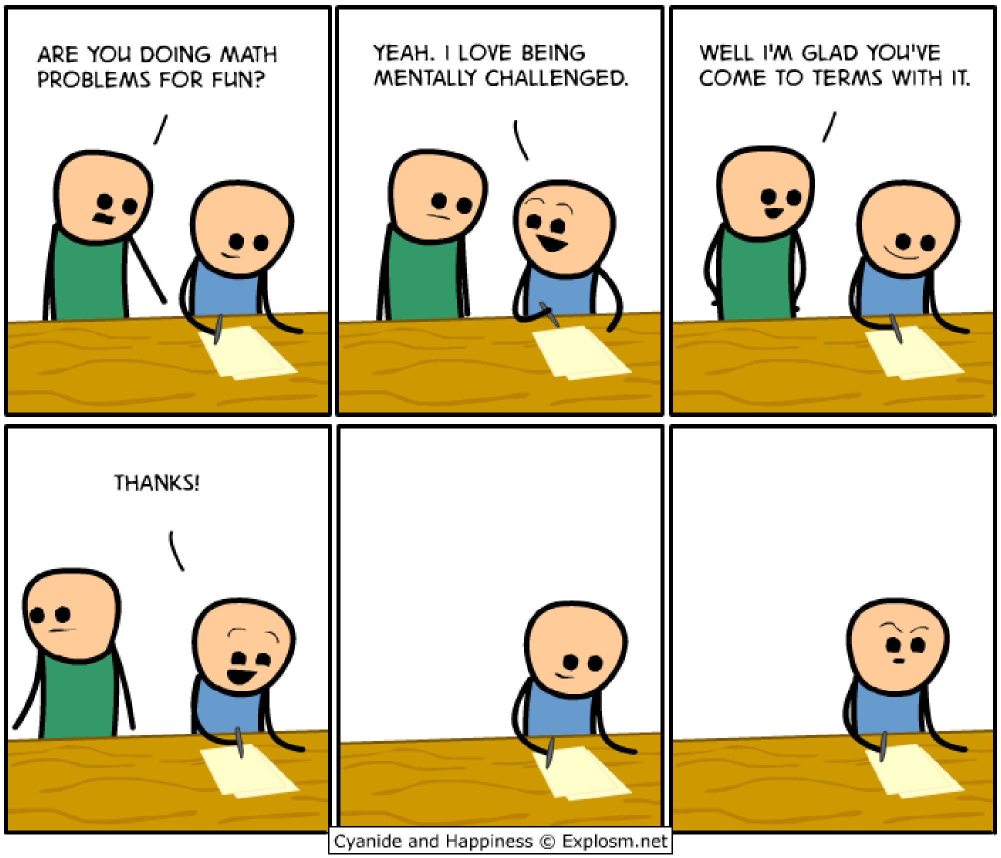
CSE 311: Foundations of Computing

Lecture 13: Modular Inverse, Exponentiation

Pick up
Solutions to HW3
(Please bring ~~extra~~
to the front



Last time: Euclid's Algorithm

gcd(a, b) = gcd(b, a mod b), gcd(a,0)=a.

```
int gcd(int a, int b){ /* a >= b, b >= 0 */
    if (b == 0) {
        return a;
    }
    else {
        return gcd(b, a % b);
    }
```

Last time: Bézout's theorem

If a and b are positive integers, then there exist integers $\boldsymbol{s}$ and $\boldsymbol{t}$ such that

$$\gcd(a,b) = \boldsymbol{s}a + \boldsymbol{t}b.$$

Last time: Extended Euclidean algorithm

- Can use Euclid's Algorithm to find s, t such that

$$\gcd(a, b) = sa + tb$$

Step 3 (Backward Substitute Equations):

$$8 = 35 - 1 * 27$$
$$3 = 27 - 3 * 8$$
$$2 = 8 - 2 * 3$$
$$1 = 3 - 1 * 2$$

Plug in the def of 2

$$1 = 3 - 1 * (8 - 2 * 3)$$
$$= 3 - 8 + 2 * 3$$
$$= (-1) * 8 + 3 * 3$$

Re-arrange into 3's and 8's

Plug in the def of 3

$$= (-1) * 8 + 3 * (27 - 3 * 8)$$
$$= (-1) * 8 + 3 * 27 + (-9) * 8$$
$$= 3 * 27 + (-10) * 8$$

Re-arrange into 8's and 27's

$$= 3 * 27 + (-10) * (35 - 1 * 27)$$
$$= 3 * 27 + (-10) * 35 + 10 * 27$$
$$= 13 * 27 + (-10) * 35$$

Re-arrange into 27's and 35's

Multiplicative inverse $\text{mod } m$

The *multiplication inverse mod* m of $a \text{ mod } m$ is $b \text{ mod } m$ iff $ab \equiv 1 \pmod{m}$.

x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

mod 7

x	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

mod 10

Multiplicative inverse $\bmod m$

Suppose $\mathrm{GCD}(a, m) = 1$

By Bézout's Theorem, there exist integers s and t such that $sa + tm = 1$.

$s \bmod m$ is the multiplicative inverse of $a \bmod m$

$$1 = (sa + tm) \bmod m = sa \bmod m$$

So… we can compute multiplicative inverses with the extended Euclidean algorithm

These inverses let us solve modular equations…

Example

a m

Solve: $7x \equiv 1 \pmod{26}$

gcd(26, 7)

$26 = 3 * 7 + 5$

$7 = 1 * 5 + 2$

$5 = 2 * 2 + 1$

$2 = 2 \cdot 1 + 0$

$5 = 26 - 3 * 7$

$2 = 7 - 1 * 5$

$1 = 5 - 2 * 2$

$1 = 5 - 2 * 2$

$= 5 - 2 * (7 - 1 * 5) = -2 * 7 + 3 * 5$

$= -2 * 7 + 3 * (26 - 3 * 7)$

$= 3 \cdot 26 + (-11) * 7$

$-11 \bmod 26 = 15$

solutions are of the form

Set of all sol'ns

$15 + 26 \cdot k$ for all $k \in \mathbb{Z}$

Example

Solve: $7x \equiv 1 \pmod{26}$

$$\gcd(26,7) = \gcd(7,5) = \gcd(5,2) = \gcd(2,1) = 1$$

$$26 = 3*7 + 5 \qquad 5 = 26 - 3*7$$
$$7 = 1*5 + 2 \qquad 2 = 7 - 1*5$$
$$5 = 2*2 + 1 \qquad 1 = 5 - 2*2$$

$$\begin{aligned} 1 &= 5 - 2*(7 - 1*5) \\ &= (-2)*7 + 3*5 \\ &= (-2)*7 + 3*(26 - 3*7) \\ &= (-11)*7 + 3*26 \end{aligned}$$

Multiplicative inverse of 7 mod 26

Now $(-11) \bmod 26 = 15$. **So,** $x = 15 + 26k$ **for** $k \in \mathbb{Z}$.

Example of a more general equation

Now solve: $7y \equiv 3 \pmod{26}$

We already computed that 15 is the multiplicative inverse of 7 modulo 26:

That is, $7 \cdot 15 \equiv 1 \pmod{26}$

By the multiplicative property of mod we have

$$7 \cdot \underbrace{15 \cdot 3} \equiv 3 \pmod{26}$$

So any $y \equiv 15 \cdot 3 \pmod{26}$ is a solution.

That is, $y = 19 + 26k$ for any integer k is a solution.

Math mod a prime is especially nice

$\gcd(a, m) = 1$ **if** m **is prime and** $0 < a < m$ **so can always solve these equations mod a prime.**

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

x	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

mod 7

Modular Exponentiation mod 7

x	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

a	a^1	a^2	a^3	a^4	a^5	a^6
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4	4	2	1	4	2	1
5	5	4	6	2	3	1
6	6	1	6	1	6	1

Exponentiation

- **Compute** 78365^{81453}
- **Compute** $78365^{81453} \bmod 104729$
- **Output is small**
  - need to keep intermediate results small

Repeated Squaring – small and fast

Since $ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$

we have $a^2 \bmod m = (a \bmod m)^2 \bmod m$

and $a^4 \bmod m = (a^2 \bmod m)^2 \bmod m$

and $a^8 \bmod m = (a^4 \bmod m)^2 \bmod m$

and $a^{16} \bmod m = (a^8 \bmod m)^2 \bmod m$

and $a^{32} \bmod m = (a^{16} \bmod m)^2 \bmod m$

Can compute $a^k \bmod m$ for $k = 2^i$ in only i steps

What if k is not a power of 2?

Fast Exponentiation: $a^k \bmod m$ for all k

$$a^{2j} \bmod m = \left(a^j \bmod m\right)^2 \bmod m$$

$$a^{2j+1} \bmod m \;=\; \left((a \bmod m) \cdot (a^{2j} \bmod m)\right) \bmod m$$

Fast Exponentiation

```
public static long FastModExp(long a, long k, long modulus) {
       long result = 1;
       long temp;

       if (k > 0) {
           if ((k % 2) == 0) {
               temp = FastModExp(a,k/2,modulus);
               result = (temp * temp) % modulus;
           }
           else {
               temp = FastModExp(a,k-1,modulus);
               result = (a * temp) % modulus;
           }
       }
       return result;
   }
```

$$a^{2j} \bmod m = \left(a^{j} \bmod m\right)^{2} \bmod m$$

$$a^{2j+1} \bmod m = \left((a \bmod m) \cdot (a^{2j} \bmod m)\right) \bmod m$$

Fast Exponentiation Algorithm

Another way: 81453 in binary is 10011111000101101

$$81453 = 2^{16} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^{9} + 2^{5} + 2^{3} + 2^{2} + 2^{0}$$

$$a^{81453} = a^{2^{16}} \cdot a^{2^{13}} \cdot a^{2^{12}} \cdot a^{2^{11}} \cdot a^{2^{10}} \cdot a^{2^{9}} \cdot a^{2^{5}} \cdot a^{2^{3}} \cdot a^{2^{2}} \cdot a^{2^{0}}$$

$a^{81453} \bmod m =$

$(\ldots(((((a^{2^{16}} \bmod m \cdot$

$a^{2^{13}} \bmod m\ ) \bmod m \cdot$

$a^{2^{12}} \bmod m) \bmod m \cdot$

$a^{2^{11}} \bmod m) \bmod m \cdot$

$a^{2^{10}} \bmod m) \bmod m \cdot$

$a^{2^{9}} \bmod m) \bmod m \cdot$

$a^{2^{5}} \bmod m) \bmod m \cdot$

$a^{2^{3}} \bmod m) \bmod m \cdot$

$a^{2^{2}} \bmod m) \bmod m \cdot$

$a^{2^{0}} \bmod m) \bmod m$

The fast exponentiation algorithm computes $a^k \bmod m$ using $\leq 2\log k$ multiplications $\bmod m$

Using Fast Modular Exponentiation

- Your e-commerce web transactions use SSL (Secure Socket Layer) based on RSA encryption
- RSA
  - Vendor chooses random 512-bit or 1024-bit primes p, q and 512/1024-bit exponent e. Computes $m = p \cdot q$
  - Vendor broadcasts (m, e)
  - To send a to vendor, you compute $C = a^e \bmod m$ using *fast modular exponentiation* and send C to the vendor.
  - Using secret p, q the vendor computes d that is the *multiplicative inverse* of e mod $(p-1)(q-1)$.
  - Vendor computes $C^d \bmod m$ using *fast modular exponentiation*.
  - Fact: $a = C^d \bmod m$ for $0 < a < m$ unless $p|a$ or $q|a$