Lecture 9: Modular Exponentiation; Primality Testing

Problem: Compute $a^m \pmod n$ efficiently.

Many applications:

- Solve problems like: "what are the last 5 digits of 3^{5211}?"
- Many primality tests; e.g. n prime $\Rightarrow 2^{n-1} \equiv 1 \pmod n$ (often)
- Public-key cryptography: - Diffie-Hellman key exchange
  - RSA cryptosystem

Dumbest algorithm:

(1) $\underbrace{a \times a \times a \times \cdots \times a}_{m \text{ times}}$. (huge number!) ~ m digits so if $m \sim 10^{100}$, in big trouble!

(2) reduce this huge number mod n by long division

Dumb algorithm:

(1) $b = (a \times a) \bmod n$

$(b \times a) \bmod n$

multiply m times, reducing mod m each time. No big numbers.

But still takes m steps.

Hmm.

Example:

$$a^{2^k} \pmod n$$

$$= \underbrace{a \times a \times a \times a \times \cdots \times a}_{2^k \text{ factors}} \pmod n$$

$$\left(\left(\left(\left(a^2\right)^2\right)^2\right)^{\cdots}\right)^2 \quad \text{— } k \text{ squarings} \quad \leftarrow \text{vastly better.}$$

k much smaller than 2^k!

15 versus $2^{15} = 32768$

how many steps would you rather do?

This special case generalizes:

Algorithm: To compute $a^m \pmod n$

(1) Write $m = \sum_{i=0}^{k} \varepsilon_i 2^i$ with $\varepsilon_i \in \{0,1\}$

i.e. in binary.

(2) Then

$$a^m \equiv a^{\sum_{i=0}^{k} \varepsilon_i 2^i} \equiv \prod_{\varepsilon_i \neq 0} a^{2^i}, \pmod n$$

Where we compute $a^{2^i} = \left(\left(\left(a^2\right)^2\right)\cdots\right)^2$ by i squarings.

Algorithm: Write m in binary.

As long as m is nonzero:

if m is odd: output a "1" digit

if m is even: output a "0" digit

$m = \lfloor \frac{m}{2} \rfloor$. ← floor division

Complete example: $7^{25} \pmod{100}$

$a \equiv b \pmod{100}$ means last two digits the same.

① Write 25 in binary

i	m	digit c_i
0	25	1
1	12	0
2	6	0
3	3	1
4	1	1

Indeed $25 = 1 + 8 + 16$

Answer.

② $7^{25} = 7^{1+8+16} \equiv 7 \cdot 7^{2^3} \cdot 7^{2^4} \equiv 7 \cdot 43 \cdot 7 \equiv 1 \cdot 7 \equiv \boxed{7}$

k	7^{2^k}
0	7
1	49
2	43
3	1
4	7

$49 \cdot 7 = 343$

$43 \cdot 7 = 301$

Applications:

- Primality test: If n prime then $a^{n-1} \equiv 1 \pmod{n}$.

  Will show computer examples.

  Amazing: Can tell n not prime, but factoring infeasible.

- Miller-Rabin: A better primality test.

Miller - Rabin:

Random algorithm

Input: $n \geq 5$

OUTPUT: True or false

⇓ n probably prime

⇓ n definitely composite

Algorithm: I

① Write $n-1 = 2^k \cdot m$ with m odd

② Choose random a with $1 < a < n$

③ Set $b = a^m \pmod n$

If $b \equiv \pm 1 \pmod n$ output "n is probably prime" and terminate.

④ If $b^{2^r} \equiv -1 \pmod n$ for some r with $1 \leq r \leq k-1$, output "n prob. prime"

⑤ Output "n definitely composite.

[To understand requires stuff we haven't done yet — good project idea.]

Probability of true correctness goes up a lot on each successive call.

"If n is composite then M-R called t times declares n prime with probability $\leq 4^{-t}$."

By making t big, e.g. around 1000, then prob. of error much less than weird hardware failures due to cosmic radiation.

Agrawal, Kayal, Saxena (2002) Gave first ever deterministic polynomial time primality test. Also uses exponentiation, but in "cyclotomic rings".