MATH2068/2988 Week 8 Lecture 3

Anthony Henderson (substitute for today)

Yesterday: algorithms to test primality (fast)

Problem: given a large number n known to be composite, find a nontrivial factor of n.

e.g. if n is an RSA modulus, then

$n = pq$ where p, q large primes.

No known polynomial-time algorithm.

Method 1 (trial division): successively test all the primes p from 2 up to $\sqrt{n}$ to see whether p divides n. How long could it take?

At worst, might have to go up to $\sqrt{n}$.

Prime Number Theorem: (number of primes $\leq N$) $\sim \dfrac{N}{\ln N}$

ratio tends to 1 as $N \to \infty$

So time for Method 1 is about $\dfrac{\sqrt{n}}{\ln \sqrt{n}} = \dfrac{n^{1/2}}{\frac{1}{2}\ln n}$

Not polynomial in $k = \log_2(n)$. $(n = 2^k,\ n^{1/2} = 2^{k/2})$

<u>Method 2</u> successively choose elements $a \in \{1, \dots, n-1\}$ and test whether $\gcd(a, n) = 1$.

↳ fast because of the Euclidean Algorithm

If $\gcd(a,n) > 1$, then $\gcd(a,n)$ is a nontrivial divisor of n.

Worst case: $n = pq$, p, q primes $p < q$.
To be lucky, need to choose a which is a multiple of p or q. Probability on each choice is $\frac{1}{p} + \frac{1}{q}$
If p, q are both about $\sqrt{n}$,
the probability is about $\frac{2}{\sqrt{n}}$.

So the number of times you expect to have to choose before you are lucky is constant $\times \sqrt{n}$.
No better than Method 1, if you choose randomly.

Pollard's Rho Algorithm (1975)

↑ Greek letter ρ

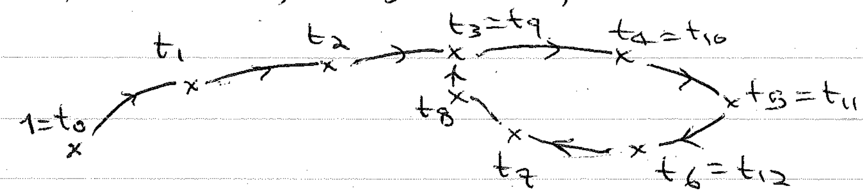
Define a sequence $t_0, t_1, t_2, t_3, \ldots$ by

$$t_0 = 1$$

$$t_i = t_{i-1}^2 + 1 \quad \text{reduced mod } n.$$

Example $n = 55$

$t_0 = 1$, $t_1 = 2$, $t_2 = 5$, $t_3 = 26$, $t_4 = 17$,
$t_5 = 15$, $t_6 = 6$, $t_7 = 37$, $t_8 = 50$, $t_9 = 26$, $t_{10} = 17$,
$t_{11} = 15$, $t_{12} = 6$, $t_{13} = 37$, ...



Every element of the sequence is between 0 and $n-1$, so the sequence has to repeat within at most n steps. When do you expect it to repeat for the first time?

Birthday Problem: how many people do you need to have before the probability that two share the same birthday is $\geq 50\%$?

Answer: 23.

If you imagine choosing birthdays at random (let $N = 365$), the probability that the first m choices are all different is: $\frac{N-1}{N} \frac{N-2}{N} \frac{N-3}{N} \cdots \frac{N-m+1}{N} = \frac{(N-1)!}{(N-m)!\,N^{m-1}}$

Have to minimize m such that $\frac{(N-1)!}{(N-m)!\,N^{m-1}} < \frac{1}{2}$.

For general N, the answer to the Birthday Problem is $O(\sqrt{N})$.

So if we treat the elements of the sequence $t_0, t_1, t_2, \cdots$ as random selections from $\{0, \cdots, n-1\}$, we expect the first repetition

$$t_i = t_j \qquad i < j$$

to happen when $j = \text{constant} \times \sqrt{n}$.

If p is a nontrivial divisor of n, we can think of sequence of residues of $t_0, t_1, \cdots$ modulo p.

We expect the first repetition

$$t_i \equiv t_j \pmod{p} \qquad i < j$$

to happen when $j = \text{constant} \times \sqrt{p}$.

It's likely that $t_i \equiv t_j \pmod{p}$ happens <u>before</u> $t_i = t_j$.

So the numbers $t_i - t_j$ are a good collection of numbers to test $\gcd(t_i - t_j, n) = 1$ or not.

<u>Example</u> (continued) $p = 11$

$t_0 \bmod 11 = 1$

$t_1 \bmod 11 = 2$ $\qquad \gcd(t_3 - t_5, 55) = 11.$

$t_2 \bmod 11 = 5$

$t_3 \bmod 11 = 4$

$t_4 \bmod 11 = 6$

$t_5 \bmod 11 = 4$

Problem: if you actually have to calculate
$\gcd(t_i - t_j, n)$ for $i, j < \sqrt{p} \approx n^{1/4}$
Then you are calculating $n^{1/2}$ gcd's.
No better than trial division!

<u>"Cycle-finding"</u>: If $t_i \equiv t_j \pmod{p}$, $i<j$,
then there is some l such that $i \leq l < j$
and $t_l \equiv t_{2l} \pmod{p}$.

<u>Proof:</u> Let $m = j - i$.
The set $\{i, i+1, \ldots, j-1\}$ consists of
m consecutive integers, so exactly one of
them is a multiple of m, call that one l.
Now $t_i \equiv t_j \pmod{p}$
so $t_{i+1} \equiv t_{j+1} \pmod{p}$
$t_{i+2} \equiv t_{j+2} \pmod{p}$ etc.
i.e. $t_k \equiv t_{k+m} \pmod{p}$ for all $k \geq i$.
So $t_l \equiv t_{l+m} \equiv t_{l+2m} \equiv \cdots \equiv t_{2l} \pmod{p}$. $\square$

Algorithm: Start with $l=0$, $t_0=1$.

Increase l by 1, compute

t_l = residue of t_{l-1}^2+1 mod n

t_{2l} = residue of $(t_{2(l-1)}^2+1)^2+1$ mod n

Check whether $\gcd(t_l - t_{2l}, n) > 1$; if so, stop.

~~you've found a nontrivial divisor of n~~.

Expect to stop within about $n^{1/4}$ steps.

When you stop, if $t_l = t_{2l}$, bad luck

(try again with different t_0, or replace x^2+1 by some other operation).

Otherwise, if $t_l \neq t_{2l}$, you've found a nontrivial divisor of n.

Monte Carlo Algorithm — not guaranteed to work.