Super-simple and powerful idea

Randomness and Computation
Lecture 17, October 20, 2009

Drawing balls at random
You have a bucket with n balls
there are n/100 green balls (good)
the remaining are red (bad)

What is the probability of drawing a good ball
if you draw a random ball from the bucket?

Now if you draw balls from the bucket at random
(with replacement), how many draws until you
draw a good ball?

Expected number of draws until a good ball
= n/k.

Drawing balls at random
You have a bucket with n balls
there are k green balls (good)
the remaining are red (bad)

Probability of getting a good ball
= k/n.

even simpler idea...
Repeated experiments

Suppose you run a random experiment that fails with probability $\frac{1}{2}$ independent of the past.

What is the probability that you succeed in $k$ steps?
- $1 - \text{probability you fail in all } k \text{ steps}$
- $1 - (\frac{1}{2})^k$

If probability of failure was at most $\delta$, then
probability of success at least once in $k$ steps is at least $1 - \delta$

the following (trivial) question

Representing numbers

Question:
Given two numbers $a$ and $b$, both $\leq n$,
how long does it take to add them together?
- a) $n$
- b) $\sqrt{n}$
- c) $\log n$
- d) $2^n$

Representing the number $n$ takes $\log n$ bits

Representing numbers

Suppose I want to sell you (for $1M)
an algorithm
that takes as input a number $n$, and factors them in $\approx \sqrt{n}$ time,
should you accept my offer?

Factoring fast ⇒ breaking RSA!

The Fundamental theorem of Algebra

A root of a polynomial $p(x)$ is a value $r$, such that $p(r) = 0$.

If $p(x)$ is a polynomial of degree $d$, how many roots can it have?

At most $d$. 

Finally, remember this bit of algebra
How to check your work...

Checking Our Work

Suppose we want to check \( p(x) q(x) = r(x) \), where \( p, q \) and \( r \) are three polynomials.
\[ (x-1)(x^3+x^2+x+1) = x^4-1 \]

If the polynomials have degree \( n \), requires \( n^2 \) mults by elementary school algorithms -- or can do faster with fancy techniques like the Fast Fourier transform.

Can we check if \( p(x) q(x) = r(x) \) more efficiently?

Idea: Evaluate on Random Inputs

Let \( f(x) = p(x) q(x) - r(x) \). Is \( f \) zero everywhere?

Idea: Evaluate \( f \) on a random input \( z \).

If we get nonzero \( f(z) \), clearly \( f \) is not zero.

If we get \( f(z) = 0 \), this is (weak) evidence that \( f \) is zero everywhere.

If \( f(x) \) is a degree \( 2n \) polynomial, it can only have \( 2n \) roots. We’re unlikely to guess one of these by chance!

Equality checking by random evaluation

1. Say \( S = \{1, 2, ..., 4n\} \)
2. Select value \( z \) uniformly at random from \( S \).
3. Evaluate \( f(z) = p(z) q(z) - r(z) \)
4. If \( f(z) = 0 \), output “possibly equal” otherwise output “not equal”

Equality checking by random evaluation

What is the probability the algorithm outputs “not equal” when in fact \( f = 0 \)?

Zero!

If \( p(x)q(x) = r(x) \), always correct!

Equality checking by random evaluation

What is the probability the algorithm outputs “maybe equal” when in fact \( f \neq 0 \)?

Let \( A = \{z \mid z \text{ is a root of } f \} \).

Recall that \( |A| \leq \text{degree of } f \leq 2n \).

Therefore: \( P(\text{picked a root}) \leq \frac{2n}{4n} = \frac{1}{2} \)
Equality checking by random evaluation

By repeating this procedure k times, we are “fooled” by the event

\[ f(z_1) = f(z_2) = \ldots = f(z_k) = 0 \]

when actually \( f(x) \neq 0 \)

with probability no bigger than

\[ P(\text{picked root k times}) \leq \left(\frac{1}{2}\right)^k \]

This idea can be used for testing equality of lots of different types of “functions”!

“Random Fingerprinting”

Find a small random “fingerprint” of a large object: e.g., the value \( f(z) \) of a polynomial at a point \( z \).

This fingerprint captures the essential information about the larger object: if two large objects are different, their fingerprints are usually different!

Earth has huge file \( X \) that she transferred to Moon. Moon gets \( Y \).

Did you get that file ok? Was the transmission accurate?

Uh, yeah….

I guess….

Earth: \( X \)

How do we quickly check for accuracy? More soon…

Moon: \( Y \)

Picking A Random Prime

How do you pick a random 1000-bit prime?

“Pick a random 1000-bit prime.”

Strategy:
1) Generate random 1000-bit number
2) Test for primality
   [more on this later in the lecture]
3) Repeat until you find a prime.
How many retries until we succeed?

Recall the balls-from-bucket experiment?

If \( n = \) number of 1000-bit numbers = \( 2^{1000} \)
and \( k = \) number of primes in \( 0 \ldots 2^{1000} - 1 \)
then \( E[\text{number of rounds}] = \frac{n}{k} \).

Question:

How many primes are there between 1 and \( n \)?

(approximately…)

Let \( \pi(n) \) be the number of primes between 1 and \( n \).

I wonder how fast \( \pi(n) \) grows?

Conjecture [1790s]:

\[
\lim_{n \to \infty} \frac{\pi(n)}{n / \ln n} = 1
\]

Their estimates

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \pi(x) )</th>
<th>Gauss' Li</th>
<th>Legendre</th>
<th>( x/(\log x - 1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>168</td>
<td>178</td>
<td>172</td>
<td>169</td>
</tr>
<tr>
<td>10000</td>
<td>1229</td>
<td>1246</td>
<td>1231</td>
<td>1218</td>
</tr>
<tr>
<td>100000</td>
<td>9592</td>
<td>9630</td>
<td>9588</td>
<td>9512</td>
</tr>
<tr>
<td>1000000</td>
<td>78498</td>
<td>78628</td>
<td>78534</td>
<td>78030</td>
</tr>
<tr>
<td>10000000</td>
<td>664579</td>
<td>664918</td>
<td>665138</td>
<td>661459</td>
</tr>
<tr>
<td>100000000</td>
<td>5761455</td>
<td>5762209</td>
<td>5769341</td>
<td>5740304</td>
</tr>
<tr>
<td>1000000000</td>
<td>50847534</td>
<td>50849235</td>
<td>50917519</td>
<td>50701542</td>
</tr>
<tr>
<td>10000000000</td>
<td>455052511</td>
<td>455055614</td>
<td>455743004</td>
<td>454011971</td>
</tr>
</tbody>
</table>

Two independent proofs of the Prime Density Theorem [1896]:

\[
\lim_{n \to \infty} \frac{\pi(n)}{n / \ln n} = 1
\]

The Prime Density Theorem

This theorem remains one of the celebrated achievements of number theory.

In fact, an even sharper conjecture remains one of the great open problems of mathematics!
The Riemann Hypothesis [1859]:
\[
\lim_{n \to \infty} \frac{\pi(n) - n/\ln n}{\sqrt{n}} = 0
\]
still unproven!

The Prime Density Theorem
\[
\lim_{n \to \infty} \frac{\pi(n)}{n/\ln n} = 1
\]
Slightly easier to show \( \pi(n)/n \geq 1/(2 \log n) \).
In other words, at least \((1/2B)\) of all B-bit numbers are prime.

So, for this algo...
“Pick a random 1000-bit prime.”

Strategy:
1) Generate random 1000-bit number
2) Test for primality
   [more on this later in the lecture]
3) Repeat until you find a prime.

These are the facts:
If we’re picking 1000-bit numbers,
- number of numbers is \( n = 2^{1000} \)
- number of primes is \( k \geq n/(2 \log n) \)
Hence, expected number of trials before we get a prime number is \( n/k \leq 2 \log n = 2000 \).

Moral of the story
Picking a random B-bit prime is “almost as easy as” picking B random B-bit numbers.

Need to try at most 2 B times, in expectation.

Earth has huge file X that she transferred to Moon. Moon gets Y.

Did you get that file ok? Was the transmission accurate?

Uh, yeah.

Earth: X

Moon: Y
Are X and Y the same N-bit numbers?

\[ p = \text{random } 2\log N\text{-bit prime} \]
\[ \text{Send } (p, X \mod p) \]
\[ \text{Answer to } "X \equiv Y \mod p?" \]

Earth: X  
Moon: Y

Why is this any good?

Easy case:
If \( X = Y \), then \( X \equiv Y \pmod{p} \)

Why is this any good?

Harder case:
What if \( X \neq Y \)? We mess up if \( p \mid (X-Y) \).
Define \( Z = (X-Y) \). To mess up, \( p \) must divide \( Z \).

\( Z \) is an \( N \)-bit number.
\( \Rightarrow \) \( Z \) is at most \( 2^N \).

But each prime \( \geq 2 \).
Hence \( Z \) has at most \( N \) prime divisors.

Almost there...

\( Z = (X-Y) \) has at most \( N \) prime divisors.

How many \( 2\log N \)-bit primes?

A random \( B \)-bit number has at least a \( 1/2B \) chance of being prime.

at least \( 2^{2\log N}/(2^2\log N) = N^2/(4\log N) >> 2N \) primes.

Only (at most) half of them divide \( Z \).

Boosting the success probability

Pick \( t \) random \( 2\log N \)-bit primes: \( P_1, P_2, \ldots, P_t \)
\[ \text{Send } (X \mod P_i) \text{ for } 1 \leq i \leq t \]

k answers to \( "X \equiv Y \mod P_i?" \)

Earth-Moon protocol makes mistake with probability at most \( 1/2! \)

Theorem:
Let \( X \) and \( Y \) be distinct \( N \)-bit numbers. Let \( p \) be a random \( 2\log N \)-bit prime.

Then
\[ \text{Prob } [X = Y \pmod{p}] < 1/2 \]
Exponentially smaller error probability

If X=Y, always accept.

If X ≠ Y,

\[ \text{Prob}[X = Y \mod P, \text{for all } i] \leq (1/2)^i \]

Picking A Random Prime

"Pick a random B-bit prime."

Strategy:
1) Generate random B-bit numbers
2) Test each one for primality

How do we test if a number \( n \) is prime?

Primality Testing: Trial Division On Input \( n \)

Trial division up to \( \sqrt{n} \)

for \( k = 2 \) to \( \sqrt{n} \) do

if \( k \mid n \) then

return "\( n \) is not prime"

otherwise return "\( n \) is prime"

about \( \sqrt{n} \) divisions

Is that efficient?

For a 1000-bit number, this will take about \( 2^{500} \) operations.

That's not very efficient at all!!!

More on efficiency and run-times in a future lecture...

But so many cryptosystems, like RSA and PGP, use fast primality testing as part of their subroutine to generate a random \( n \)-bit prime!

What is the fast primality testing algorithm that they use?

There are fast randomized algorithms to do primality testing.

Miller-Rabin test Solovay-Strassen test
If n is composite, how would you show it?

Give a non-trivial factor of n.

But, we don’t know how to factor numbers fast.

We will use a different certificate of compositeness that does not require factoring.

Recall that for prime p, a ≠ 0 mod p:
Fermat Little Thm: \( a^{p-1} = 1 \mod p \).

Hence, \( a^{(p-1)/2} = \pm 1 \).

So if we could find some a ≠ 0 mod p such that \( a^{(p-1)/2} \neq \pm 1 \)

⇒ p must not be prime.

simple idea #1

Good \( n = \{ a \mod n \mid a^{(n-1)/2} \neq \infty \} \)

(these prove that n is not prime)

Useless \( n = \{ a \mod n \mid a^{(n-1)/2} = \infty \} \)

(these don’t prove anything)

Theorem:
if Good \( n \) is not empty, then Good \( n \) contains at least half of \( Z_n^* \).

Proof

Good \( n = \{ a \mod n \mid a^{(n-1)/2} \neq \infty \} \)

Useless \( n = \{ a \mod n \mid a^{(n-1)/2} = \infty \} \)

Fact 1: Useless \( n \) is a subgroup of \( Z_n^* \)

Fact 2: If H is a subgroup of G then |H| divides |G|.

⇒ If Good is not empty, then |Useless| ≤ |\( Z_n^* \)| / 2

⇒ |Good| ≥ |\( Z_n^* \)| / 2

simple idea #2

Remember Lagrange’s theorem:
If G is a group, and U is a subgroup then |U| divides |G|.

In particular, if \( U \neq G \) then |U| ≤ |G|/2.

Randomized Primality Test

Let’s suppose that Good \( n \) = \( \{ a \mod n \mid a^{(n-1)/2} \neq \infty \} \)

contains at least half the elements of \( Z_n^* \).

Randomized Test:

For i = 1 to k:
Pick random a \( _i \mod \{ 2 \ldots n-1 \} \);
If GCD(a, n) = 1, Halt with “Composite”;
If a^{(n-1)/2} = \infty \mod 1, Halt with “Composite”;

Halt with “I think n is prime. I am only wrong (½)^k fraction of times I think that n is prime.”
Is $\text{Good}_n$ non-empty for all primes $n$?

Recall: $\text{Good}_n = \{ a \in \mathbb{Z}_n^* : a^{(n-1)/2} \not\equiv 1 \} $

$\text{Good}_n$ may be empty even if $n$ is not a prime.

A Carmichael number is a number $n$ such that $a^{n-1} \equiv 1 \pmod n$ for all numbers $a$ with $\gcd(a, n) = 1$.

Example: $n = 561 = 3 \cdot 11 \cdot 17$ (the smallest Carmichael number)

$1105 = 5 \cdot 13 \cdot 17$

$1729 = 7 \cdot 13 \cdot 19$

And there are many of them. For sufficiently large $m$, there are at least $m^{2/7}$ Carmichael numbers between 1 and $m$.

The saving grace

The randomized test fails only for Carmichael numbers.

But, there is an efficient way to test for Carmichael numbers.

Which gives an efficient algorithm for primality.

Randomized Primality Test

Let's suppose that $\text{Good}_n$ contains at least half the elements of $\mathbb{Z}_n^*$.

Randomized Test:

If $n$ is Carmichael, Halt with "Composite"

For $i = 1$ to $k$:

Pick random $a_i \not\equiv 1 \pmod n$;

If $\gcd(a_i, n) = 1$, Halt with "Composite";

If $a_i^{n-1} \not\equiv 1 \pmod n$, Halt with "Composite";

Halt with "I think $n$ is prime. I am only wrong $(1/2)^k$ fraction of times I think that $n$ is prime."

Primality Versus Factoring

Primality has a fast randomized algorithm.

Factoring is not known to have a fast algorithm. The fastest randomized algorithm currently known takes $\exp(O(n \log n \log n)^{1/3})$ operations on $n$-bit numbers.

Google: RSA Challenge Numbers

<table>
<thead>
<tr>
<th>number</th>
<th>digits</th>
<th>price</th>
<th>factored</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-100</td>
<td>100</td>
<td>Apr. 1991</td>
<td></td>
</tr>
<tr>
<td>RSA-110</td>
<td>110</td>
<td>Apr. 1993</td>
<td></td>
</tr>
<tr>
<td>RSA-120</td>
<td>120</td>
<td>Jun. 1993</td>
<td></td>
</tr>
<tr>
<td>RSA-129</td>
<td>129</td>
<td>$100 Apr. 1994</td>
<td></td>
</tr>
<tr>
<td>RSA-130</td>
<td>130</td>
<td>Apr. 15, 1996</td>
<td></td>
</tr>
<tr>
<td>RSA-140</td>
<td>140</td>
<td>Feb. 2, 1999</td>
<td></td>
</tr>
<tr>
<td>RSA-150</td>
<td>150</td>
<td>Apr. 16, 2004</td>
<td></td>
</tr>
<tr>
<td>RSA-155</td>
<td>155</td>
<td>Aug. 22, 1999</td>
<td></td>
</tr>
<tr>
<td>RSA-160</td>
<td>160</td>
<td>Apr. 1, 2003</td>
<td></td>
</tr>
<tr>
<td>RSA-200</td>
<td>200</td>
<td>May 9, 2005</td>
<td></td>
</tr>
<tr>
<td>RSA-576</td>
<td>174</td>
<td>$10,000 Dec. 3, 2003</td>
<td></td>
</tr>
<tr>
<td>RSA-640</td>
<td>193</td>
<td>$20,000 Nov 2, 2005</td>
<td></td>
</tr>
<tr>
<td>RSA-704</td>
<td>212</td>
<td>$30,000 open</td>
<td></td>
</tr>
<tr>
<td>RSA-768</td>
<td>232</td>
<td>$50,000 open</td>
<td></td>
</tr>
<tr>
<td>RSA-896</td>
<td>270</td>
<td>$75,000 open</td>
<td></td>
</tr>
<tr>
<td>RSA-1024</td>
<td>309</td>
<td>$100,000 open</td>
<td></td>
</tr>
<tr>
<td>RSA-1536</td>
<td>463</td>
<td>$150,000 open</td>
<td></td>
</tr>
<tr>
<td>RSA-2048</td>
<td>617</td>
<td>$200,000 open</td>
<td></td>
</tr>
</tbody>
</table>

The techniques we’ve been discussing today are sometimes called “fingerprinting.”

The idea is that a large object such as a string (or document, or function, or data structure…) is represented by a much smaller “fingerprint” using randomness.

If two objects have identical sets of fingerprints, they’re likely the same object.
Primes
Prime number theorem
How to pick random primes

Fingerprinting
How to check if a polynomial of degree d is zero
How to check if two n-bit strings are identical

Primality
Fermat’s Little Theorem
Algorithm for testing primality

Here’s What You Need to Know...