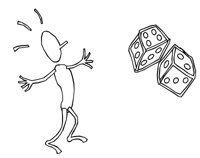
15-251

Great Theoretical Ideas in Computer Science

Randomness and Computation

Lecture 18 (October 25, 2007)



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 are long, this 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?

Great Idea: Evaluating on Random Inputs

Let f(x) = p(x) q(x) – r(x). Is f zero everywhere?
Idea: Evaluate f on a *random* input z.

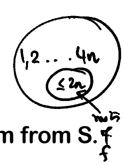
If we get nonzero f(z), clearly f is not zero.

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

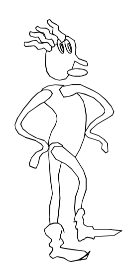
In fact: If f(x) is a degree 2n polynomial, it can only have 2n roots. We're unlikely to guess one of these by chance!

Equality checking by random evaluation

1. Fix a sample space $S=\{z_1, z_2,\ldots, z_m\}$ with arbitrary points z_i, for m=4n.
2. Select random 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!

f(z)=0

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

Equality checking by random evaluation

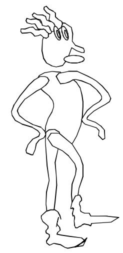
What is the probability the algorithm outputs "maybe equal" when in fact $f \neq 0$?

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

Recall that $|A| \leq$ degree of $f \leq 2n$.

Therefore: $P(A) \leq 2n/m = 2n/4n = 1/2$

Equality checking by random evaluation



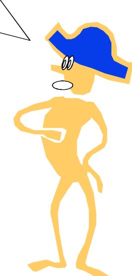
By repeating this procedure k times, we are "fooled" by the event

$f(z_1) = f(z_2) = \ldots = f(z_k) = 0$
when actually $f(x) \neq 0$

with probability no bigger than

$P(A) \leq (2n/m)^k = 2^{-k}$

Wow! That idea could be used for testing equality of lots of different types of "functions"!

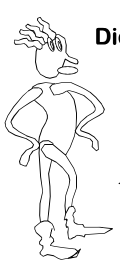


"Random Fingerprinting"

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!

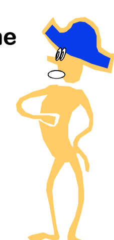
Earth has huge file X that she transferred to Moon. Moon gets Y.



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

Uh, yeah....

I guess....

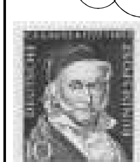


Earth: X

How do we quickly check for accuracy? More soon...

Moon: Y

Legendre



Gauss

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

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

De la Vallée Poussin

J-S Hadamard

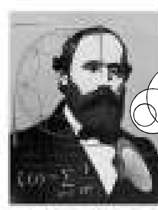
Two <u>independent</u> proofs of the Prime Density Theorem [1896]:

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

The Prime Density Theorem

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

In fact, an <u>even sharper conjecture</u> remains one of the great open problems of mathematics!



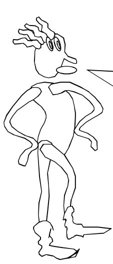
The Riemann Hypothesis [1859]:

$$\lim_{n\to\infty} \frac{\pi(n) - n/\ln n}{\sqrt{n}} = 0$$

still unproven!

The Prime Density Theorem

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



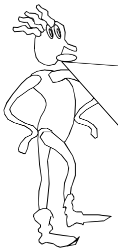
Slightly easier to show $\pi(n)/n \geq 1/(2 \log n)$.

Handout $\frac{\pi(n)}{n} \geq \frac{\log 2}{4} \cdot \frac{1}{\log n}$

Random log n bit number is a random number from 1..n

$\pi(n) / n \geq 1/2\log n$

means that a random logn-bit number has at least a 1/(2logn) chance of being prime.



Random k bit number is a random number from $1..2^k$

$\pi(2^k) / 2^k \geq 1/(2k)$ means that a random k-bit number has at least a 1/(2k) chance of being prime.

Really useful fact

A random k-bit number has at least a 1/2k chance of being prime.

So if we pick 2k random k-bit numbers the expected number of primes on the list is at least 1

Picking A Random Prime

Many modern cryptosystems (e.g., RSA) include the instructions:

"Pick a random n-bit prime."

How can this be done efficiently?

Picking A Random Prime

"Pick a random n-bit prime."

Strategy:

1) Generate random n-bit numbers
2) Test each one for primality
   [more on this later in the lecture]
3) Repeat until you find a prime.

Picking A Random Prime

"Pick a random n-bit prime."

1) Generate kn random n-bit numbers

Each trial has a ≥ 1/2n chance of being prime.

Pr[all kn trials yield composites]

$\leq (1-1/2n)^{kn} = (1-1/2n)^{2n * k/2} \leq 1/e^{k/2}$

Picking A Random Prime

"Pick a random n-bit prime."

Strategy:

1) Generate random n-bit numbers
2) Test each one for primality

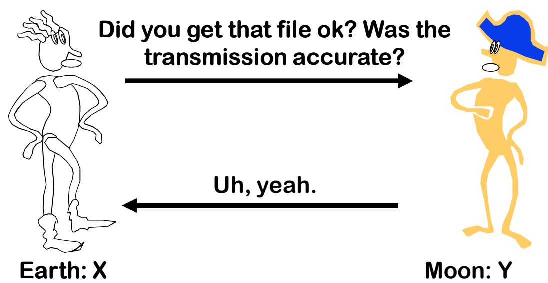
If we try out 10000 random 1000-bit numbers, chance of not getting any 1000-bit primes $\leq e^{-5}$

Moral of the story

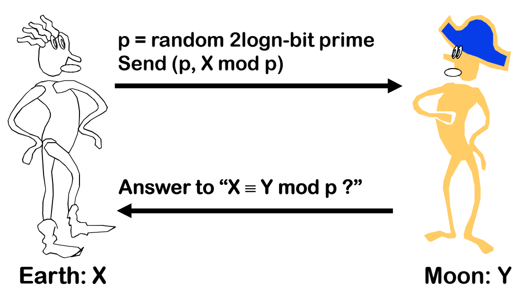
Picking a random prime is
"almost as easy as"
picking a random number.

(Provided we can check for primality.
More on this later.)

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



Are X and Y the same n-bit numbers?



Why is this any good?

Easy case:

If X = Y, then X ≡ Y (mod p)

Why is this any good?

X≠Y   X mod p = Y mod p

Harder case:

What if X ≠ Y? We mess up if p | (X-Y).

Define Z = (X-Y). To mess up, p must divide Z.

Z is an n-bit number.
⇒ Z is at most 2^n.

But each prime ≥ 2.
Hence Z has at most n prime divisors.

Almost there…

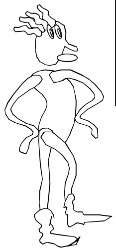
Z has at most n prime divisors.

How many 2logn-bit primes?   1, n^2, $2^{2\log n}$ #primes ≥ $\frac{n^2}{\log(n^2)} = \frac{n^2}{2\log n} >>$

A random k-bit number has at least a 1/2k 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.

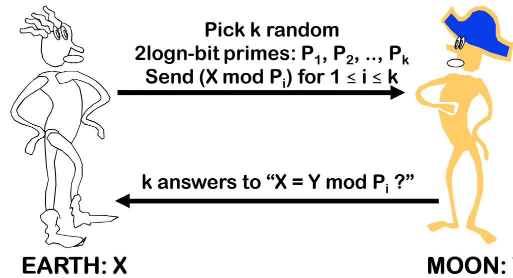


Theorem:
Let X and Y be distinct n-bit numbers. Let p be a random 2logn-bit prime.

Then
Prob [X = Y mod p] < 1/2

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

Boosting the success probability



Exponentially smaller error probability

If X=Y, always accept.

If $X \neq Y$,
Prob [X = Y mod P_i for all i] $\le (1/2)^k$

Picking A Random Prime

"Pick a random n-bit prime."

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

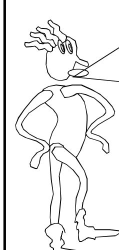
How do we test for primality?

Primality Testing: Trial Division On Input n

Trial division up to $\sqrt{n}$

for $k = 2$ to $\sqrt{n}$ do
  if $k | n$ then
  return "n is not prime"
  otherwise return "n is prime"

about $\sqrt{n}$ divisions



Trial division performs $\sqrt{n}$ divisions on input n.

Is that efficient?

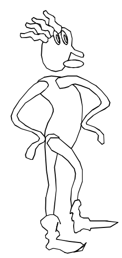
For a 1000-bit number, this will take about 2^{500} operations.

That's not very efficient at all!!!

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

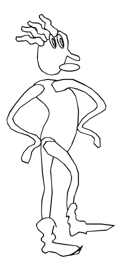
But so many cryptosystems, like RSA and PGP, use fast primality testing as part of their subroutine to generate a random n-bit prime!

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

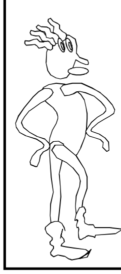


If n is composite, how would you show it?

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.



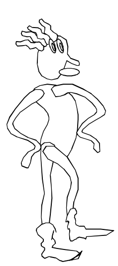
Recall that for prime p, $a \neq 0 \bmod p$:
Fermat Little Thm: $a^{p-1} = 1 \bmod p$.

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

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

$\Rightarrow$ p must not be prime.

Is n prime?



want to find $a \in Good_n$

$Good_n = \{ a \in Z^*_n \mid a^{(n-1)/2} \neq \pm 1 \}$
(these prove that n is not prime)

$Useless_n = \{ a \in Z^*_n \mid a^{(n-1)/2} = \pm 1 \}$
(these don't prove anything)

Theorem:
if $Good_n$ is not empty, then $Good_n$ contains at least half of Z_n^*.

Proof

$Good_n = \{ a \in Z^*_n \mid a^{(n-1)/2} \neq \pm 1 \}$
$Useless_n = \{ a \in Z^*_n \mid a^{(n-1)/2} = \pm 1 \}$

Fact 1: $Useless_n$ is a subgroup of Z_n^*

Fact 2: If H is a subgroup of G then |H| divides |G|. (Lagrange's Thm)

$\Rightarrow$ If Good is not empty, then $|Useless| \leq |Z_n^*| / 2$

$\Rightarrow |Good| \geq |Z_n^*| / 2$

Randomized Primality Test

Let's suppose that $Good_n = \{ a \in Z^*_n \mid a^{(n-1)/2} \neq \pm 1 \}$ contains at least half the elements of Z^*_n.

Randomized Test:

For i = 1 to k:

- Pick random $a_i \in [2 .. n-1]$;
- If GCD(a_i, n) $\neq$ 1, Halt with "Composite";
- If $a_i^{(n-1)/2} \neq \pm 1$, 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 $Good_n$ non-empty for all primes n?

Recall: $Good_n = \{ a \in Z^*_n \mid a^{(n-1)/2} \neq \pm 1 \}$

Good_n may be empty even if n is not a prime.

A Carmichael number is a number n such that $a^{(n-1)/2} = 1 \pmod n$ for all numbers a with gcd(a,n)=1.

Example: n = 561 =3*11*17 (the smallest Carmichael number)

1105 = 5*13*17

1729 = 7*13*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

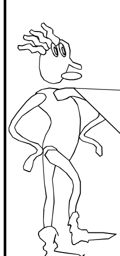
Let's suppose that $Good_n$ contains at least half the elements of Z^*_n.

Randomized Test:

For i = 1 to k:

- Pick random $a_i \in [2 .. n-1]$;
- If GCD(a_i, n) $\neq$ 1, Halt with "Composite";
- If $a_i^{(n-1)/2} \neq \pm 1$, Halt with "Composite";
- If n is Carmichael, 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."



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.

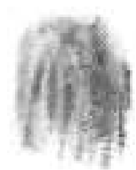
number	digits	prize	factored
RSA-100	100		Apr. 1991
RSA-110	110		Apr. 1992
RSA-120	120		Jun. 1993
RSA-129	129	$100	Apr. 1994
RSA-130	130		Apr. 10, 1996
RSA-140	140		Feb. 2, 1999
RSA-150	150		Apr. 16, 2004
RSA-155	155		Aug. 22, 1999
RSA-160	160		Apr. 1, 2003
RSA-200	200		May 9, 2005
RSA-576	174	$10,000	Dec. 3, 2003
RSA-640	193	$20,000	Nov 2, 2005
RSA-704	212	$30,000	open
RSA-768	232	$50,000	open
RSA-896	270	$75,000	open
RSA-1024	309	$100,000	open
RSA-1536	463	$150,000	open
RSA-2048	617	$200,000	open

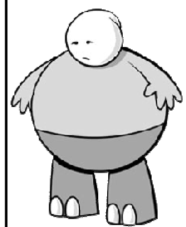
Google: RSA Challenge Numbers

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.





Here's What You Need to Know…

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