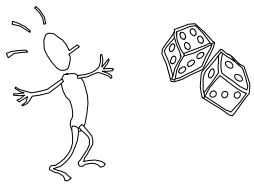
Great Theoretical Ideas In Computer Science			
(Steven Rudich) John Lafferty		CS 15-251	Fall 2005
Lecture 22	Nov. 9, 2005	Carnegie Mellon University	

Randomness and Computation: Some Prime Examples



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 $O(n^2)$ work by elementary school algorithms

-- or O(n log n) work 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?

Idea: Evaluate f on a *random* input z.
If we get f(z) = 0, this is 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. Choose sample space $S=\{z_1, z_2,..., z_m\}$ with arbitrary points z_i, for $m=2n/\delta$.
2. Select z from S with probability 1/m
3. Evaluate p(z) q(z) – r(z) = f(z)
4. If f(z) = 0, output "equal" otherwise output "not equal"

Equality checking by random evaluation

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

Let A = {z | z is a root of f}. Then $|A| \leq$ 2n. Therefore:

$P(A) \leq 2n/m = \delta$. We take δ to be small.

Equality checking by random evaluation

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

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

with probability no bigger than

$P(A) \leq (2n)^k/m^k = \delta^k$

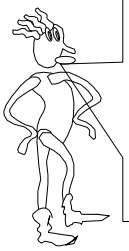
Wow! That idea could be used for testing equality of lots of different types of "functions"!

Yes! It's a very powerful technique.

For example, a matrix is just a special kind of function. Suppose we do a matrix multiplication of two nxn matrices:

$$AB = C$$

The idea of random evaluation can be used to efficiently check the calculation.



Just evaluate the "function" C on a random input vector r. In fact, we can take r to be a *random bit vector*: $r=(1,0,0,1,\ldots,0)^T$

So to test if $AB = C$ we compute

$x = Br$, $y = Ax$, and $z = Cr$

If $y = z$, we take this as evidence that the calculation was correct. The amount of work is only $O(n^2)$.

Claim: If $AB \neq C$ and r is a random n-bit vector, then $\Pr(ABr = Cr) \leq \frac{1}{2}$.

So, if a complicated, fancy algorithm is used to compute AB in time $O(n^{2+\omega})$, it can be efficiently checked with only $O(n^2)$ extra work, using randomness!

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.

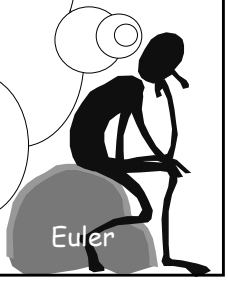
EARTH: X

MOON: Y

Let $\pi(n)$ be the number of primes between 1 and n. I wonder how fast $\pi(n)$ grows?

Conjecture [1750]:

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



Hadamard [1900]

The Prime Density Theorem:

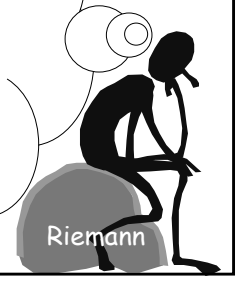
$$\lim_{n\to\infty}\frac{\pi(n)}{n/\ln n}=1$$

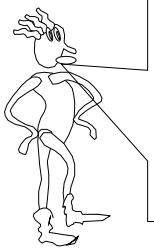
The Prime Density Theorem

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]

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





For EC on Homework:

The Theta Version Of The Prime Density Theorem:

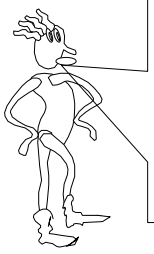
$\pi(n) = \theta(n / \ln n)$

$\pi(n)/n = \theta(1/ \ln n)$

The probability that a randomly selected n-bit number is prime is $\theta(1/n)$. Explicitly, $\pi(n) / n \geq 1/2\log n$

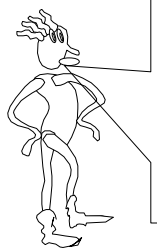
Random $\log n$ bit number is a random number from 1..n

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

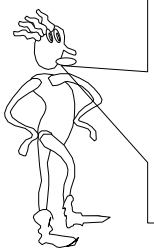


Random k bit number is a random number from $1..2^k$

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



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



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

Hence, if we pick 2k random k bit numbers the
expected number of primes on the list is ≥ 1

Picking A Random Prime

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

"Pick a uniformly chosen n-bit prime."

How can this be done efficiently?

Picking A Random Prime

"Pick a uniformly chosen n-bit prime."

Strategy:

1) Generate random n-bit numbers
2) Test each one for primality [more on this to later in the lecture]

Picking A Random Prime

"Pick a uniformly chosen n-bit prime."

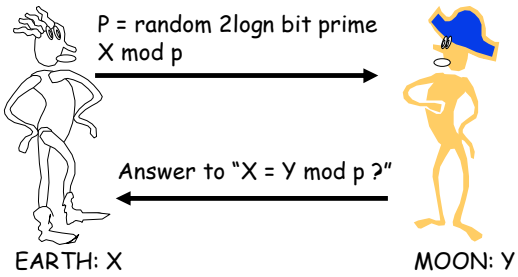
1) Generate kn random n-bit numbers
   Each trial has a $\geq 1/2n$ chance of being prime. Probability that we get all composites
   $\leq (1-1/2n)^{kn} = (1-1/2n)^{2n * k/2} \leq 1/e^{k/2}$

Theorem:
Let X and Y be n-bit numbers. If $X \neq Y$ then at least half the 2logn bit primes p satisfy $X \neq Y \bmod p$.

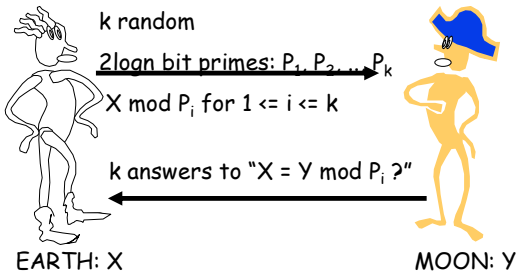
Theorem: Let X and Y be distinct, n-bit numbers. Let p be a random 2logn-bit prime:

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

Are X and Y the same n-bit numbers?



Are X and Y the same n-bit numbers?



If X=Y, Earth and Moon will always accept. If $X \neq Y$ then Earth and Moon have a $\geq \frac{1}{2}$ chance of catching it, for each of k iterations.

If $X \neq Y$,

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

Picking A Random Prime

"Pick a uniformly chosen n-bit prime."

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

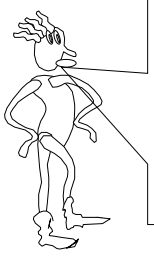
How can we test primality efficiently?

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"

$O(\sqrt{n}\,(\log n)^2)$ time if division is $O((\log n)^2)$



On input n, trial division uses $O(\sqrt{n}\,(\log n)^2)$ time. Is that efficient?

No! The input length is logn. Let k = log n. In terms of k, we are using $2^{k/2}\,k^2$ time.

The time is EXPONENTIAL in the input length.

Do the primes have a polynomial-time decision algorithm?

Euclid gave us a fast GCD algorithm. Surely, he tried to give a faster primality test than trial division. Euclid, Euler, and Gauss all failed!

In 2002, Agrawal, Saxena, and Kayal (AKS) gave a deterministic primality test that runs in time $O(n^{12})$.

This was the first *deterministic* polynomial-time algorithm that didn't depend on some *unproven conjecture*, like the Riemann Hypothesis!

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?

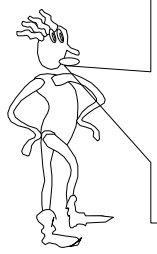
There are fast *randomized* algorithms to do primality testing.

Strangely, by allowing our computational model an extra instruction for flipping a fair coin, we seem to be able to compute some things faster!

If n is composite, what would be a certificate of compositeness for n?

A non-trivial factor of n.

Even using randomness, no one knows how to find a factor quickly. We will use a *different* certificate of compositeness that does not require factoring.



When working modulo a prime p, for any $a \neq 0$, $a^{(p-1)/2} = \pm 1$.

Fermat: $a^{p-1} = 1 \bmod p$.

$X^2 = 1 \bmod p$ has at most 2 roots.

1 and -1 are roots, so it has no others.

"Euler Certificate" Of Compositeness

When working modulo a prime p, for any $a \neq 0$, $a^{(p-1)/2} = \pm 1$.

We say that a is a certificate of compositeness for n, if $a \neq 0$, $a^{(n-1)/2} \neq \pm 1$.

Clearly, if we find a certificate of compositeness for n, we know that n is composite.

"Euler Certificates" Of Compositeness

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

$NOT\text{-}EC_n = \{ a \in Z^*_n \mid a^{(n-1)/2} = \pm 1 \}$

if $NOT\text{-}EC_n \neq Z^*_n$ then EC_n is at least half of Z^*_n

Suppose a is in EC_n and a_i are in $NOT\text{-}EC_n$. Then $(a\,a_i)^{n-1} = a^{n-1}\,a_i^{n-1} = a^{n-1} \neq 1 \pmod n$

So, $a\,a_i$ is in EC_n.

"Euler Certificates" Of Compositeness

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

$NOT\text{-}EC_n = \{ a \in Z^*_n \mid a^{(n-1)/2} = \pm 1 \}$

Fancier agrument: $NOT\text{-}EC_n$ is a subgroup of Z^*_n

Hence, by Lagrange's theorem, if $NOT\text{-}EC_n \neq Z^*_n$, then the size of $NOT\text{-}EC_n$ is no more than half the size of Z^*_n.

"Euler Certificates" Of Compositeness

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

$NOT\text{-}EC_n = \{ a \in Z^*_n \mid a^{(n-1)/2} = \pm 1 \}$

Hence, $EC_n = \emptyset$, or EC_n contains at least half the elements of Z^*_n.

"Euler Certificates" Of Compositeness

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

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

This makes it easy to find one using random bits. . . . . .

Randomized Primality Test

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

Loop i = 1 to k:

Pick random $a_i \in [2 .. n-1]$;

If $GCD(a_i,n) \neq 1$, Halt with "Composite";

If $a_i^{(n-1)/2} = \pm 1$, Halt with "Composite";

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

n prime means half of a's satisfy

$a^{(n-1)/2} = -1 \bmod n$

If n is prime, then Z_n^* has a generator g. This means that a random $a \in Z_n^*$ is given by g^r for uniformly distributed r. Half the time, r is odd:

$$(g^r)^{(n-1)/2} = -1 \bmod n$$

Randomized Primality Test

Loop i = 1 to k:

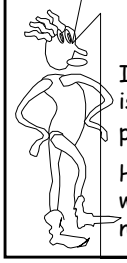
Pick random $a_i \in [2 .. n-1]$;

If $GCD(a_i,n) \neq 1$, Halt with "Composite";

If $a_i^{(n-1)/2} \neq \pm 1$, Halt with "Composite";

If all the $a_i^{(n-1)/2}$ s = 1, Halt with "I think n is composite";

if n were prime, half of the a's satisfy $a^{(n-1)/2} = -1$



Randomized Primality Test

Test to see if n is even, or a power of a number.

Loop i = 1 to k:

Pick random $a_i \in [2 .. n-1]$;

If $GCD(a_i,n) \neq 1$, Halt with "Composite";

If $a_i^{(n-1)/2} = \pm 1$, Halt with "Composite";

If all the $a_i^{(n-1)/2}$ s = 1, halt with "I think n is composite. I am only wrong with $pr \leq (\frac{1}{2})^k$";

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

Test to see if n is even, or a power of a number.

Loop i = 1 to k:

Pick random $a_i \in [2 .. n-1]$;

If $GCD(a_i,n) \neq 1$, Halt with "Composite";

If $a_i^{(n-1)/2} = \pm 1$, Halt with "Composite";

If all the a_i's = 1, Halt with "I think n is composite. I am only wrong with $pr \leq (\frac{1}{2})^k$";

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

Can prove that if n is an odd composite, not a power, and there is some a such that $a^{(n-1)/2} = -1$, then $EC_n \neq \emptyset$. Hence, EC_n is at least a half fraction of Z^*_n.

Carmichael Numbers

Certain numbers *masquerade* as primes.

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

Example: n=561 (the smallest Carmichael number)

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

Randomized Primality Tests

The test we gave is a fast randomized algorithm, that makes 2-sided error. This means that sometimes we are mistaken when we think n is prime, and sometimes we are mistaken when we think n is composite. We can err in both directions.

There is a different, one-sided algorithm that never makes a mistake when it thinks n is composite.

There is yet another one-sided algorithm that never makes a mistake when it thinks n is prime.

Picking A Random Prime

"Pick a uniformly chosen n-bit prime."

Strategy:

1) Generate random n-bit numbers
2) Do a fast randomized primality check on each one

Primality Testing Versus Factoring

Primality has a fast randomized algorithm.

Factoring is not known to have a fast algorithm. In fact, after thousands of years of research, the fastest randomized algorithm takes $\exp(O(n \log n \log n)^{1/3})$ operations on numbers of length n. With great effort, we can currently factor 200 digit numbers.

RSA-200, May 9, 2005: 2799783391 1221327870 8294676387 2260162107 0446786955 4285375600 9992932612 8400107609 3456710529 5536085606 1822351910 9513657886 3710595448 2006576775 0985805576 1357909873 4950144178 8631789462 9518723786 9221823983

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	open
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.