Note 3: Primality Testing

1 Primality testing

There are many situations in which the primality of a given integer must be determined. For example, fingerprinting requires a supply of prime numbers, as does the RSA cryptosystem (where the primes should typically have hundreds or thousands of bits).

A theoretical breakthrough in 2002, due to Agrawal, Kayal and Saxena [AKS02], has given us a deterministic polynomial time algorithm for primality testing. However, in practice randomized algorithms are more efficient and continue to be used. These algorithms date back to the 1970's and caused a surge in the study of applied number theory.

1.1 A simple deterministic algorithm

Given an odd integer n, we wish to determine whether n is prime or composite. Consider the following deterministic algorithm:

for $a = 2, 3, ..., \lfloor\sqrt{n}\rfloor$ **do**
    if $a|n$ **then** output "composite" and halt
output "prime"

This algorithm is obviously correct. However, because the for-loop has $O(\sqrt{n})$ iterations, the algorithm does not have running time polynomial in the number of input bits (which is $O(\log n)$). (Consider the case where n is an integer with hundreds or even thousands of bits; then $\sqrt{n}$ is an enormous number as well!) Other, more sophisticated algorithms based on prime number sieves are a bit more efficient but still suffer from the same drawback.

1.2 A simple randomized algorithm

The above trivial algorithm can be turned into a randomized, witness-searching algorithm by picking a at random, but this has a potentially huge error probability since n will in general have only few divisors. We need to be more intelligent in our definition of witnesses. In what follows, we use the notation $\mathbb{Z}_n$ to denote the additive group of integers mod n, and $\mathbb{Z}_n^*$ the multiplicative group of integers in $\{1, \ldots, n-1\}$ that are coprime to n (i.e., with $\gcd(a, n) = 1$). We recall also that, when p is prime, the integers mod p form a *field*, usually called $\mathrm{GF}[p]$.

Our first randomized algorithm is based on the following classical theorem:

Theorem 1 (Fermat's Little Theorem) *If p is prime, then $a^{p-1} = 1 \bmod p$ for all $a \in \{1, ..., p-1\}$.*

In particular, for a given integer n, if there exists an $a \in \{1, ..., n-1\}$ such that $a^{n-1} \neq 1 \bmod n$, then surely n is composite. This fact suggests the following algorithm, known as "Fermat's Test":

pick $a \in \{1, ..., n-1\}$ uniformly at random
if $\gcd(a, n) \neq 1$ **then** output "composite" and halt
else if $a^{n-1} \neq 1 \bmod n$ **then** output "composite"
else output "prime"

Computing $\gcd(a, n)$ can be done in time $O(\log n)$ by Euclid's algorithm, and a^{n-1} can be computed in $O(\log^2 n)$ time by repeated squaring, so this algorithm runs in time polynomial in the input size ($\log n$).

Error

Clearly the algorithm is always correct when n is prime. However, when n is composite it may make an error if it fails to find a "witness," i.e., a number $a \in \mathbb{Z}_n^*$ such that $a^{n-1} \neq 1 \bmod n$. Unfortunately, there are composite numbers, known as "Carmichael numbers," that have no witnesses. The first three CN's are 561, 1105, and 1729. (**Exercise**: Prove that 561 is a CN. Hint: $561 = 3 \times 11 \times 17$; use the Chinese Remainder Theorem.) These numbers are guaranteed to fool Fermat's Test.

However, it turns out that CN's are the *only* bad inputs for the algorithm, as we now show.

Theorem 2 *If n is composite and not a Carmichael number, then* $\Pr[\text{Error in the Fermat test}] \leq \frac{1}{2}$.

Proof: Let $S_n = \{a \in \mathbb{Z}_n^* : a^{n-1} = 1 \bmod n\}$, i.e., the set of bad choices for a. Clearly S_n is a subgroup of $\mathbb{Z}_n^*$ (because it contains 1 and is closed under multiplication). Moreover, it is a *proper* subgroup since n is not a CN and therefore there is at least one witness $a \notin S_n$. By Lagrange's Theorem, the size of any subgroup must divide the size of the group so we may conclude that $|S_n| \leq \frac{1}{2}|\mathbb{Z}_n^*|$. ■

Fortunately, CN's are rare: there are only 255 of them less than 10^8, and only a little more than 20 million of them less than 10^{21}. For this reason, Fermat's Test actually performs quite well in practice. Indeed, even the simplified deterministic version which performs the test only with $a = 2$ is sometimes used to produce "industrial grade" primes. This simplified version makes only 22 errors in the first 10,000 integers. It has also been proved for this version that

$$\lim_{b \to \infty} \Pr[\text{Error on random } b\text{-bit number}] \to 0.$$

For values of b of 50 and 100, we get $\Pr[\text{Error}] \leq 10^{-6}$ and $\Pr[\text{Error}] \leq 10^{-13}$ respectively. However, it is much more desirable to have an algorithm that does not have disastrous performance on any input (especially if the numbers we are testing for primality are not random).

In the next section, we will develop a more sophisticated randomized algorithm of similar flavor that can successfully handle all inputs (including Carmichael numbers).

2 A randomized algorithm for primality testing

The algorithm we present in this section is due independently to Miller and Rabin (see the notes at the end of the section for details). It is based on the following simple observation. An integer $x \in \mathbb{Z}_n^*$ is said to be a *square root of* 1 mod n if $x^2 = 1 \bmod n$. Obviously, for any $n > 2$, there are always at least two distinct square roots of 1 mod n, namely ± 1. We call these the *trivial* square roots of 1.

Claim 3 *If p is prime, then 1 has no non-trivial square roots in $\mathbb{Z}_p^*$, i.e., the only square roots of 1 in $\mathbb{Z}_p^*$ are* ± 1.

Proof: When p is prime, the integers mod p form a field $\mathrm{GF}[p]$. The condition $x^2 = 1 \bmod p$ says that x is a root of the degree-2 polynomial $x^2 - 1$ over $\mathrm{GF}[p]$. But any such polynomial can have at most two roots.
■

Note that, when n is composite, there may be non-trivial square roots of 1 mod n: for example, in $\mathbb{Z}_{35}^*$, $6^2 = 1$.

The idea of the algorithm is to search for non-trivial square roots of 1. Specifically, assume that n is odd, and not a prime power. (We can detect perfect powers in polynomial time and exclude them: **Exercise!** HINT: Any perfect power is of the form m^k with $k \leq \log_2 n$.) Then $n-1$ is even, and we can write $n-1 = 2^r R$ with R odd. We search by computing $a^R, a^{2R}, a^{4R}, \cdots, a^{2^r R} = a^{n-1}$ (all mod n). Each term in this sequence is the square of the previous one, and the last term is 1 (otherwise we have failed the Fermat test and n is composite). Thus if the first 1 in the sequence is preceded by a number other than -1, we have found a non-trivial root and can declare that n is composite. More specifically the algorithm works as follows:

if $n > 2$ is even or a perfect power **then** output "composite"
compute r, R s.t. $n - 1 = 2^r \cdot R$ [*Note: R is odd*]
pick $a \in \{1, ..., n-1\}$ uniformly at random
if $\gcd(a, n) \neq 1$ **then** ouput "composite" and halt
compute $b_i = a^{2^i R} \bmod n,\ i = 0, 1, \cdots r$
if $b_r [= a^{n-1}] \neq 1 \bmod n$ **then** output "composite" and halt
else if $b_0 = 1 \bmod n$ **then** output "prime" and halt
else let $j = \max\{i : b_i \neq 1\}$
if $b_j \neq -1$ **then** output "composite"
else output "prime"

For example, for the Carmichael number $n = 561$, we have $n - 1 = 560 = 2^4 \times 35$. If $a = 2$ then the sequence computed by the algorithm is $a^{35} \bmod 561 = 263$, $a^{70} \bmod 561 = 166$, $a^{140} \bmod 561 = 67$, $a^{280} \bmod 561 = 1$, $a^{560} \bmod 561 = 1$. So the algorithm finds that 67 is a non-trivial square root of 1 and therefore concludes that 561 is not prime.

Notice that the output "composite" is always correct. However the algorithm may err when it outputs "prime". It remains to show that the error probability is bounded when n is composite; we will do this next.

The algorithm begins by testing to see if a randomly chosen a passes the test of Fermat's little theorem. If $a^{n-1} \neq 1 \bmod n$, then we know that n is composite, otherwise we continue by searching for a nontrivial square root of 1. We examine the sequence of descending square roots beginning at $a^{n-1} = 1$ until we reach an odd power of a:

$$1 = a^{n-1}, a^{(n-1)/2}, a^{(n-1)/4}, \ldots, a^R$$

There are three cases to consider:

(a) The powers are all equal to 1.

(b) The first power (in descending order) that is not 1 is -1.

(c) The first power (in descending order) that is not 1 is a nontrivial root of 1.

In the first two cases we fail to find a witness for the compositeness of n, so we guess that n is prime. In the third case we have found that some power of a is a nontrivial square root of 1, so a is a witness that n is composite.

2.1 The likelihood of finding a witness

We now show that if n is composite, we are fairly likely to find a witness.

Claim 4 *If n is odd, composite, and not a prime power, then* $\Pr[a \text{ is a witness}] \geq \frac{1}{2}$.

To prove this claim we will use the following definition and lemma.

Definition 5 *Call $s = 2^i R$ a **bad** power if $\exists x \in \mathbb{Z}_n^*$ such that $x^s = -1 \bmod n$.*

Lemma 6 *For any bad power s, $S_n = \{x \in \mathbb{Z}_n^* : x^s = \pm 1 \bmod n\}$ is a proper subgroup of $\mathbb{Z}_n^*$.*

We will first use the lemma to prove claim 4, and then finish by proving the lemma.

Proof of Claim 4: Let $s^* = 2^{i^*} R$ be the largest bad power in the sequence $R, 2R, 2^2 R, \ldots, 2^r R$. (We know s^* exists because R is odd, so $(-1)^R = -1$ and hence R at least is bad.)

Let S_n be the proper subgroup corresponding to s^*, as given by Lemma 6. Consider any non-witness a. One of the following cases must hold:

(i) $a^R = a^{2R} = a^{4R} = \ldots = a^{n-1} = 1 \bmod n$;

(ii) $a^{2^i R} = -1 \bmod n$, $a^{2^{i+1} R} = \ldots = a^{n-1} = 1 \bmod n$ (for some i).

In either case, we claim that $a \in S_n$. In case (i), $a^{s^*} = 1 \bmod n$, so $a \in S_n$. In case (ii), we know that $2^i R$ is a bad power, and since s^* is the largest bad power then $s^* \geq 2^i R$, implying $a^{s^*} = \pm 1 \bmod n$ and so $a \in S_n$.

Therefore, all non-witnesses must be elements of the proper subgroup S_n. Using Lagrange's Theorem just as we did in the analysis of the Fermat Test, we see that

$$\Pr[a \text{ is not a witness}] \leq \frac{|S_n|}{|\mathbb{Z}_n^*|} \leq \frac{1}{2}.$$

■

We now go back and provide the missing proof of the lemma.

Proof of Lemma 6: S_n is clearly closed under multiplication and hence a subgroup, so we must only show that it is proper, i.e., that there is some element in $\mathbb{Z}_n^*$ but not in S_n. Since s is a bad power, we can fix an $x \in \mathbb{Z}_n^*$ such that $x^s = -1$. Since n is odd, composite, and not a prime power, we can find n_1 and n_2 such that n_1 and n_2 are odd, coprime, and $n = n_1 \cdot n_2$.

Since n_1 and n_2 are coprime, the Chinese Remainder theorem implies that there exists a unique $y \in \mathbb{Z}_n$ such that

$$\begin{aligned} y &= x \bmod n_1; \\ y &= 1 \bmod n_2. \end{aligned}$$

We claim that $y \in \mathbb{Z}_n^* \setminus S_n$.

Since $y = x \bmod n_1$ and $\gcd(x, n) = 1$, we know $\gcd(y, n_1) = \gcd(x, n_1) = 1$. Also, $\gcd(y, n_2) = 1$. Together these give $\gcd(y, n) = 1$. Therefore $y \in \mathbb{Z}_n^*$.

We also know that

$$\begin{aligned} y^s &= x^s \bmod n_1 \\ &= -1 \bmod n_1 \quad (*) \\ y^s &= 1 \bmod n_2 \quad (**) \end{aligned}$$

Suppose $y \in S_n$. Then by definition, $y^s = \pm 1 \bmod n$.

If $y^s = 1 \bmod n$, then $y^s = 1 \bmod n_1$ which contradicts $(*)$.

If $y^s = -1 \bmod n$, then $y^s = -1 \bmod n_2$ which contradicts $(**)$.

Therefore, y cannot be an element of S_n, so S_n must be a proper subgroup of $\mathbb{Z}_n^*$. ■

2.2 Notes and some background on primality testing

The above ideas are generally attributed to both Miller [M76] and Rabin [R76]. More accurately, the randomized algorithm is due to Rabin, while Miller gave a deterministic version that runs in polynomial time assuming the Extended Riemann Hypothesis (ERH): specifically, Miller proved under the ERH that a witness a of the type used in the algorithm is guaranteed to exist within the first $O((\log n)^2)$ values of a. Of course, proving the ERH would require a major breakthrough in Mathematics.

A tighter analysis of the above algorithm shows that the probability of finding a witness for a composite number is at least $\frac{3}{4}$, which is asymptotically tight.

Another famous primality testing algorithm, with essentially the same high-level properties and relying crucially on the subgroup trick but with a rather different type of witness, was developed by Solovay and Strassen around the same time as the Miller/Rabin algorithm [SS77]. Both of these algorithms, and variants on them, are used routinely today to certify massive primes having thousands of bits (required in applications such as the RSA cryptosystem).

In 2002, Agrawal, Kayal, and Saxena made a theoretical breakthrough by giving a deterministic polynomial time algorithm for primality testing [AKS02]. However, it is much less efficient in practice than the above randomized algorithms. This algorithm was inspired by yet another randomized polynomial time algorithm due to Agrawal and Biswas in 1999 [AB99], which uses a generalization of the Fermat test to polynomials.

The algorithm we have seen has a one-sided error: for prime n, the probability of error is 0, while for composite n the probability of error is at most $\frac{1}{2}$. Adleman and Huang [AH87] came up with a polynomial time randomized algorithm with one-sided error in the opposite direction, i.e., it is always correct on composites, but may err on primes with probability at most $\frac{1}{2}$.[1] Although the Adleman-Huang algorithm is not very efficient in practice, it is of theoretical interest to note that it can be combined with the Miller-Rabin algorithm above to create a stronger *Las Vegas* algorithm for primality testing:

repeat forever
    run Miller-Rabin on n
    if Miller-Rabin outputs "composite" **then** output "composite" and halt
    run Adleman-Huang on n
    if Adleman-Huang outputs "prime" **then** output "prime" and halt

If this algorithm ever terminates it must be correct since Miller-Rabin never makes an error when it outputs "composite", and likewise for Adleman-Huang and "prime". Also, since the probability of error in each component is at most $\frac{1}{2}$, the probability of iterating t times without terminating decreases exponentially with t.

[1] More practically useful certificates of primality were developed by Goldwasser and Kilian [GK86]; to guarantee their existence one needs to appeal to an unproven assumption.

References

[AH87] L.M. ADLEMAN and A.M.-D. HUANG, "Recognizing primes in random polynomial time," *Proceedings of the 19th ACM STOC*, 1987, pp. 462–469.

[AB99] M. AGRAWAL and S. BISWAS, "Primality and identity testing via Chinese remaindering," *Proceedings of IEEE FOCS* 1999, pp. 202–209. Full version appeared in *Journal of the ACM* **50** (2003), pp. 429–443.

[AKS02] M. AGRAWAL, N. KAYAL and N. SAXENA, "PRIMES is in P," *Annals of Mathematics* **160** (2004), pp. 781–793.

[BM77] R.S. BOYER and J.S. MOORE, "A fast string-searching algorithm," *Communications of the ACM*, 20(10):762–772, 1977.

[Fre77] R. FREIVALDS, "Probabilistic Machines Can Use Less Running Time," *IFIP Congress* 1977, pp. 839–842.

[GK86] S. GOLDWASSER and J. KILIAN, "Almost all primes can be quickly certified," *Proceedings of the 18th ACM STOC*, 1986, pp. 316–329.

[KR81] R. KARP and M. RABIN, *Efficient randomized pattern-matching algorithms*, Technical Report TR-31-81, Aiken Computation Laboratory, Harvard University, 1981.

[KMP77] D. KNUTH, J. MORRIS and V. PRATT, "Fast pattern matching in strings," *SIAM Journal on Computing*, 6(2):323-350, 1977.

[M76] G.L. MILLER, "Riemann's hypothesis and tests for primality," *Journal of Computer and Systems Sciences* **13** (1976), pp. 300–317.

[R76] M.O. RABIN, "Probabilistic algorithms," in J.F. Traub (ed.), *Algorithms and Complexity, Recent Results and New Directions*, Academic Press, New York, 1976.

[SS77] R. SOLOVAY and V. STRASSEN, "A fast Monte Carlo test for primality," *SIAM Journal on Computing* **6** (1977), pp. 84–85. See also *SIAM Journal on Computing* **7** (1978), p. 118.