Getting to understand deterministic primality testing

Sharat Chandran

August 21, 2002

Overview

- Background
- The naive algorithm
- The competition: randomized primality testing in logarithmic time
- The (slower) deterministic algorithm

Introduction to deterministic primality testing

- Many application on the Internet using a scheme to generate *large* prime numbers. (The popular `https` protocol does).
- Given an integer p, we want to `isPrime(p)` to return 1 if p is prime, and 0 if p is composite.
- If the number turns out NOT to be prime, then we simply take the next random number and try again. The number of primes is plenty.
- The input p is given as a string of $\lg p$ bits and so ideally we would like our algorithm to take $\Theta(\lg p)$ time, or $\Theta(\lg^2 p)$ time or, in general $\Theta(\lg^k p)$ time where k does not depend on p

A naive algorithm

- (Trivial) Claim: If p is divisible by any odd number i between 3 and $\sqrt{p}$ (p assumed to be greater than 2), then p is composite

```
public static boolean isPrime (long n) {
for (int i=3;  i*i <=n; i+=2)
  if (n%i == 0) return false
return true;
}
```

- If any one of the odd numbers reports divisibility (i.e., $i \mid p$), we immediately return 0
- Notice that we used the test for p being composite, and we return 1 when we are convinced that p is NOT composite.
- The algorithm is exponential in the size of the input. In order to avoid exponential performance, we need to prevent intermediate numbers being too large. Therefore use *modular* arithmetic.

Using Fermat's Little Theorem

- Claim: For any a, $0 < a < p$, if $a^{p-1} \neq 1 \pmod{p}$, then p is composite.
- Consider the randomized algorithm below

```
Random a = new Random();
if (modularExponent(a, p-1, p) != 1) return false;
else return true;
}
```

- Since `modularExponent()` can be computed in $\lg p$ time, the algorithm is fast
- However, even if we try all $p - 1$ integers between 0 and p, and modularExponent() returns 1 every single time, we cannot be sure that p is prime, it may be prime or composite.
- Some numbers just don't have good witnesses to their compositeness if we use the Fermat little theorem.
- Despite this, for many numbers, the test is sufficient.

Better Witnesses For Being Composite

- Claim: If $x \neq \|1\| \pmod{p}$ AND $x^2 = 1 \pmod{p}$, then p is composite.
- It turns out that using both tests mentioned TOGETHER, the number of witnesses to p's compositeness is at least $\frac{p-1}{2}$. Thus the chance of our random a being a witness being a witness is independent of which number p is given

```
Random a = new Random();
if (modularExponent(a, p-1, p) != 1) return false;
for (l =0; l < log (p-1); if++)
 l (a^l != 1 && a^l != -1 && a^2l == 1) return false;
return true;
}
```

- This algorithm takes $\log^2 p$ time
- Since computing a^{p-1} requires $a^{\frac{p-1}{2}}$, we can reduce the complexity
- For a random n the probability of wrongly classifying n to be prime is about 0.25

Rabin-Miller and The RSA primality test

- We can improve the error rate by simply running the test several times

```
Random a = new Random();
for (int counter=0; counter < TRIALS; counter++)
if (witness (a, n)) return false;
return true;
}
```

- If we the number of trials is 20, the error rate is about one in a million million.
- In fact, using only 5 trials seem to be more than enough, and the algorithm is extremely fast.
- https uses this test!

Deterministic Testing

- Claim: For any a, a coprime to p, $(x-a)^p \neq (x^p - a) \pmod{p}$ implies p is composite, and vice-versa.
- This claim is bi-directional
- We could test the equivalence for a random a, so unlike the Rabin-Miller algorithm, we don't have to try various random numbers!
- Computing the left hand side requires the computation of a polynomial (symbolically). It has $p+1$ terms many of which we must verify are zero (since the right hand side has only terms in x^p and x^0).
- Not only should the coefficients not get very large, the powers should not get very large either, otherwise it will take too long to compute.

Deterministic Testing: Key result

- We now have an inferior claim
- Claim:
  - If there exists a, r, where $0 < a < p$, and $1 < r < n$, and $(x - a)^p \neq (x^p - a) \pmod{(x^r - 1)} \pmod{p}$, p is composite.
  - If we try all (a, r) pairs, and don't find the inequality p is prime. (Notice that a statement similar to the last one was absent for Fermat's little theorem.)
- Further, if p is composite, r is prime, q is a prime factor of $r - 1$, $q \geq 4\sqrt{r} \lg n$, and $n^{\frac{r-1}{q}} \neq 1 \pmod{p}$, then there exists an $a \leq 2\sqrt{r} \lg p$ such that the $(x - a)^p \neq (x^p - a) \pmod{(x^r - 1)} \pmod{p}$.
- An r for which the above is true is *nice*.
- A nice r can be obtained in $\Theta(\log^9 p)$ time

The Algorithm Skeleton

```
#define PRIMES = {2,3,5,7,9,13, ...}
i = 3;
while (!isNice(PRIMES[i]) i++;
r = PRIMES[i];
for (int a=1; a < 2 * sqrt(r)* log p; a++)
 if (witness (a, n)) return false;
return true;
}
```

- The while loop will find a nice r in $\log^9 p$ time, and r itself is at most $\log^6 p$.
- One iteration of the for loop takes $r \log^2 p$ time
- The overall algorithm is $\Theta(\log^{12} p$

Concluding Remarks

- Some ways to go before the randomized algorithm can be beaten
- Ironical that to test if a number is prime, we need a prime.
- Details of how to write the witness function has been skipped.