Scribe Notes for *Algorithmic Number Theory*
Class 27—June 24, 1998

Scribes: Scott A. Guyer, Duxing Cai, and Degong Song

Abstract

Today's lecture continues with another algorithm for testing primality based on elliptic curves. We also begin talking about the principles of integer factoring and present a few algorithms.

1 Primeness (continued)

Define $\#E_{A,B}(\mathbb{Z}/(p))$ to be the order of the abelian group formed by the elliptic curve (A, B). The following theorem suggests a method for testing primality.

THEOREM 1 *If p is a prime and (A, B) is an elliptic curve, then*

$$(p+1) - 2\sqrt{p} \leq \#E_{A,B}(\mathbb{Z}/(p)) \leq (p+1) + 2\sqrt{p}.$$

The algorithm suggested by this theorem is sketched below. Its only input is an integer $n \in \mathbb{Z}^+$.

EC-Prime-Test

1. Select a random elliptic curve over $\mathbb{Z}/(n)$.
2. Compute $o = \#E_{A,B}(\mathbb{Z}/(p))$.
3. Test whether $o = 2q$ for some prime q (Use one of the previous tests for primality to get the desired confidence that q is prime). If not, return to step 1. If certain that q is a prime, then return the list of q's.
4. Select a point on $E_{A,B}(\mathbb{Z}/(n))$ with order q.
5. Start the algorithm again with input q (recursively).

The sequence of q's represent a certificate of primeness for n.

2 Factoring Integers

We now move on to the more difficult task of determining the prime factorization of a composite number. The algorithms we will investigate solve the following problem:

FACTOR INTEGER

INSTANCE: An odd integer $n \geq 3$.

SOLUTION: A non-trivial factor of n. In other words, an integer d such that $d \mid n$ and $1 < d < n$.

We assume that we already know the integer we wish to factor is indeed composite and that it is not a perfect prime power because those cases are already solved (factoring primes is easy and we can handle perfect prime powers using root techniques). Finally, we introduce the notion of smoothness. An integer m is *B-smooth* if all prime factors of m are less than or equal to B.

Example 2.1 Both 60 and $1,000,000$ are 5-smooth.

The time complexity for most of the algorithms we will discuss has the following form

$$L_n(\alpha, c) = O(\exp((c + o(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha})),$$

where $\alpha, c \in \mathbb{Z}^+$. But this is roughly equivalent to

$$C^{(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}}.$$

Notice that when $\alpha = 1$, the algorithm is exponential in the size of the input. However, when $0 < \alpha < 1$, the complexity is sub-exponential.

2.1 Trial Division

The obvious approach to factoring an integer is to run through all the primes less than or equal to $\sqrt{n}$. Unfortunately this is very expensive when n is a product of large primes. We can make a slight modification to this approach by running through all the primes up to some bound B where B is $O((\lg n)^{O(1)})$. The time complexity of trial division in this case is satisfactory when n is B-smooth.

2.2 Pollard's Rho Algorithm

This algorithm is well suited for finding small prime factors of a composite integer n. This approach is based on Floyd's cycle finding algorithm and so we present it first.

Cycle Finding. Floyd developed an algorithm to find cycles in a random function $f : S \longrightarrow S$ where S is a finite set of cardinality n. The idea is to apply f successively on an initial element of S until a cycle is found.

$$x_0 \xrightarrow{f} x_1 \xrightarrow{f} \cdots \xrightarrow{f} x_\lambda \xrightarrow{f} \cdots \xrightarrow{f} x_{\lambda+\mu} = x_\lambda$$

Floyd's algorithm determines λ and x_λ.

CYCLE-FINDING(f, x_0)
1 $x_1 \leftarrow f(x_0)$
2 $x_2 \leftarrow f(x_1)$
3 **while** $x_i \neq x_{2i}$
4 **do** $x_{i+1} \leftarrow f(x_i)$
5 $x_{2i+2} \leftarrow f(f(x_{2i}))$
6 $i \leftarrow i + 1$
7 **return** (i, x_i)

Pollard's Rho algorithm for factoring integers follows this algorithm, except it defines its random function $f : \mathbb{Z}/(n) \longrightarrow \mathbb{Z}/(n)$ to be

$$f(x) = x^2 + 1 \pmod{n}.$$

```
POLLARD-RHO(n)
 1  a ← 2
 2  b ← 2
 3  for i ← 1 to ∞
 4      do a ← a^2 + 1   (mod n)
 5         b ← b^2 + 1   (mod n)
 6         b ← b^2 + 1   (mod n)
 7         d ← gcd(a − b, n)
 8         if 1 < d < n
 9           then return d
10         if d = n
11           then fail
```

This algorithm has expected running time $O(n^{1/4})$. Equivalently, its expected running time is $O(p^{1/2})$ which illustrates the fact that this algorithm is well suited for integers that have small prime factors.

2.3 Pollard's $p-1$ Algorithm

Pollard's next algorithm attempts to find a prime factor p such that $p-1$ is B-smooth. First, we define Q as follows:

$$\begin{aligned} Q &= \prod_{q \le B,\, q^i \le n < q^{i+1}} q^i \\ &= \prod_{q \le B} q^{\lfloor \log_q n \rfloor}. \end{aligned}$$

If $p-1$ is B-smooth, then $(p-1) \mid Q$. By Fermat's theorem, $a^Q \equiv 1 \pmod{p}$ for any integer a such that $\gcd(a, n) = 1$. Hence, $p \mid \gcd(a^Q - 1, n)$.

The time complexity of this approach is dominated by the modular multiplications of a. Hence, the time complexity is $O(B \log_B n)$.

2.4 Random Square Factoring

This approach to integer factoring is based on the following theorem and its corollary.

THEOREM 2 *Let $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$. Let $a \neq 0$ be a quadratic residue modulo n. Then*

$$x^2 \equiv a \pmod{n}$$

has precisely 2^k solutions modulo n.

Proof: Recall that $(\mathbb{Z}/(n))^* \cong (\mathbb{Z}/(p_1^{e_1}))^* \times \cdots \times (\mathbb{Z}/(p_k^{e_k}))^*$. So a maps to a k-tuple $(a_1, a_2, \ldots, a_k)$. Each a_i is a quadratic residue modulo $p_i^{e_i}$ (precisely two square roots). Every k-tuple where the ith entry is a square root of a_i in $\mathbb{Z}/(p_i^{e_i}))^*$ is a solution modulo n, hence, there are 2^k solutions modulo n. □

COROLLARY 3 *If n has $k \geq 2$ distinct factors and $x, y \in (\mathbb{Z}/(n))^*$ are chosen at random such that*

$$x^2 \equiv y^2 \pmod{n},$$

then $x \not\equiv \pm y \pmod{n}$ with probability

$$\begin{aligned} \frac{2^k - 2}{2^k} &= 1 - \frac{1}{2^{k-1}} \\ &\geq \frac{1}{2}. \end{aligned}$$

The next and final class will cover two algorithms based on random square factoring which have been used for factoring large primes over the World Wide Web.