Lecture Notes 13

68 Jacobi Symbol

The *Jacobi symbol* extends the Legendre symbol to the case where the "denominator" is an arbitrary odd positive number n.

Let n be an odd positive integer with prime factorization $\prod_{i=1}^{k} {p_i}^{e_i}$. We define the *Jacobi symbol* by

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}, \tag{1}$$

where the symbol on the left is the Jacobi symbol, and the symbol on the right is the Legendre symbol. (By convention, this product is 1 when $k = 0$, so $\left(\frac{a}{1}\right) = 1$.) Clearly, when $n = p$ is an odd prime, the Jacobi symbol and Legendre symbols agree, so the Jacobi symbol is a true extension of our earlier notion.

What does the Jacobi symbol mean when n is not prime? If $\left(\frac{a}{n}\right) = -1$ then a is definitely not a quadratic residue modulo n, but if $\left(\frac{a}{n}\right) = 1$, a might or might not be a quadratic residue. Consider the important case of $n = pq$ for p, q distinct odd primes. Then

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

so there are two cases that result in $\left(\frac{a}{n}\right) = 1$: either $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = +1$ or $\left(\frac{a}{p}\right) = \left(\frac{a}{q}\right) = -1$. In the first case, a is a quadratic residue modulo both p and q, so a is a quadratic residue modulo n. In the second case, a is not a quadratic residue modulo either p or q, so it is *not* a quadratic residue modulo n, either. Such numbers a are sometimes called "pseudo-squares" since they have Jacobi symbol 1 but are not quadratic residues.

69 Identities Involving the Jacobi Symbol

The Jacobi symbol is easily computed if the factorization of n is known using Equation 1 above and Theorem 1 of lecture 12, section 67. Similarly, $\gcd(u, v)$ is easily computed given the factorizations of u and v, without resort to the Euclidean algorithm. The remarkable fact about the Euclidean algorithm is that it lets us compute $\gcd(u, v)$ efficiently even without knowing the factors of u and v. A similar algorithm allows the Jacobi symbol $\left(\frac{a}{n}\right)$ to be computed efficiently without knowing the factorization of a or n.

The algorithm is based on identities satisfied by the Jacobi symbol:

1. $\left(\frac{0}{1}\right) = 1$; $\left(\frac{0}{n}\right) = 0$ for $n \neq 1$;
2. $\left(\frac{2}{n}\right) = 1$ if $n \equiv \pm 1 \pmod 8$; $\left(\frac{2}{n}\right) = -1$ if $n \equiv \pm 3 \pmod 8$;
3. $\left(\frac{a_1}{n}\right) = \left(\frac{a_2}{n}\right)$ if $a_1 \equiv a_2 \pmod n$;

4. $\left(\frac{2a}{n}\right) = \left(\frac{2}{n}\right)\left(\frac{a}{n}\right)$;
5. $\left(\frac{a}{n}\right) = -\left(\frac{n}{a}\right)$ if $a \equiv n \equiv 3 \pmod 4$.
6. $\left(\frac{a}{n}\right) = \left(\frac{n}{a}\right)$ if $a \equiv 1 \pmod 4$ or ($a \equiv 3 \pmod 4$ and $n \equiv 1 \pmod 4$);

There are many ways to turn these identities into an algorithm. Below is a straightforward recursive approach. Slightly more efficient iterative implementations are also possible.

```
int jacobi(int a, int n)
/* Precondition: a, n >= 0; n is odd */
{
  if (a == 0)                        /* identity 1 */
    return (n==1) ? 1 : 0;
  if (a == 2) {                      /* identity 2 */
    switch (n%8) {
    case 1:
    case 7:
      return 1;
    case 3:
    case 5:
      return -1;
    }
  }
  if ( a >= n )                      /* identity 3 */
    return jacobi(a%n, n);
  if (a%2 == 0)                      /* identity 4 */
    return jacobi(2,n)*jacobi(a/2, n);
  /* a is odd */                     /* identities 5 and 6 */
  return (a%4 == 3 && n%4 == 3) ? -jacobi(n,a) : jacobi(n,a);
}
```

70 Solovay-Strassen Test of Compositeness

Recall that a test of compositeness for n is a set of predicates $\{\tau_a(n)\}_{a\in\mathbf{Z}_n^*}$ such that if $\tau(n)$ succeeds (is true), then n is composite. The Solovay-Strassen Test is the set of predicates $\{\nu_a(n)\}_{a\in\mathbf{Z}_n^*}$, where

$$\nu_a(n) = \textsf{true iff } \left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \pmod n.$$

If n is prime, the test always fails by Theorem 1 of lecture 12, section 67. Equivalently, if some $\nu_a(n)$ succeeds, then n must be composite. Hence, the test is a valid- test of compositeness.

Let $b = a^{(n-1)/2}$, so $b^2 \equiv a^{n-1}$. There are two possible reasons why the test might succeed. One possibility is that $a^{n-1} \not\equiv 1 \pmod n$ in which case $b \not\equiv \pm 1 \pmod n$. This is just the Fermat test $\zeta_a(n)$ from section 51 of lecture notes 10. A second possibility is that $a^{n-1} \equiv 1 \pmod n$ but nevertheless, $b \not\equiv \left(\frac{a}{n}\right) \pmod n$. In this case, b is a square root of $1 \pmod n$, but it might have the opposite sign from $\left(\frac{a}{n}\right)$, or it might not even be ± 1 since 1 has additional square roots when n is composite. Strassen and Solovay show that at the probability that $\nu_a(n)$ succeeds for a randomly-chosen $a \in \mathbf{Z}_n^*$ is at least $1/2$ when n is composite.[1]

[1] R. Solovay and V. Strassen, "A Fast Monte-Carlo Test for Primality", *SIAM J. Comput.* 6:1 (1977), 84–85.

71 Miller-Rabin Test of Compositeness

The Miller-Rabin Test is more complicated to describe than the Solovay-Strassen Test, but the probability of error (that is, the probability that it fails when n is composite) seems to be lower than for Solovay-Strassen, so that the same degree of confidence can be achieved using fewer iterations of the test. This makes it faster when incorporated into a primality-testing algorithm. It is also closely related to the algorithm presented in section 55.3 (lecture notes 10) for factoring an RSA modulus given the encryption and decryption keys.

71.1 The test

The test $\mu_a(n)$ is based on computing a sequence $b_0, b_1, \ldots, b_k$ of integers in $\mathbf{Z}_n^*$. If n is prime, this sequence ends in 1, and the last non-1 element, if any, is $n-1$ $(\equiv -1 \pmod{n})$. If the observed sequence is *not* of this form, then n is composite, and the Miller-Rabin Test succeeds. Otherwise, the test fails.

The sequence is computed as follows:

1. Write $n-1 = 2^k m$, where m is an odd positive integer. Computationally, k is the number of 0's at the right (low-order) end of the binary expansion of n, and m is the number that results from n when the k low-order 0's are removed.

2. Let $b_0 = a^m \bmod n$.

3. For $i = 1, 2, \ldots, k$, let $b_i = (b_{i-1})^2 \bmod n$.

An easy inductive proof shows that $b_i = a^{2^i m} \bmod n$ for all i, $0 \le i \le k$. In particular, $b_k \equiv a^{2^k m} = a^{n-1} \pmod{n}$.

71.2 Validity

To see that the test is valid, we must show that $\mu_a(p)$ fails for all $a \in \mathbf{Z}_p^*$ when p is prime. By Euler's theorem[2], $a^{p-1} \equiv 1 \pmod{p}$, so we see that $b_k = 1$. Since 1 has only two square roots, 1 and -1, modulo p, and b_{i-1} is a square root of b_i modulo p, the last non-1 element in the sequence (if any) must be $-1 \bmod p$. This is exactly the condition for which the Miller-Rabin test fails. Hence, it fails whenever n is prime, so if it succeeds, n is indeed composite.

71.3 Accuracy

How likely is it to succeed when n is composite? It succeeds whenever $a^{n-1} \not\equiv 1 \pmod{n}$, so it succeeds whenever the Fermat test $\zeta_a(n)$ would succeed. (See section 51 of lecture notes 10.) But even when $a^{n-1} \equiv 1 \pmod{n}$ and the Fermat test fails, the Miller-Rabin test will succeed if the last non-1 element in the sequence of b's is one of the square roots of 1 other than ± 1. It can be proved that $\mu_a(n)$ succeeds for at least $3/4$ of the possible values of a. Empirically, the test almost always succeeds when n is composite, and one has to work to find a such that $\mu_a(n)$ fails.

[2] This is also called Fermat's little theorem.

71.4 Example

For example, take $n = 561 = 3 \cdot 11 \cdot 17$. This number is interesting because it is the first Carmichael number. A *Carmichael number* is an odd composite number n that satisfies $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \mathbf{Z}_n^*$. (See `http://mathworld.wolfram.com/CarmichaelNumber.html`.) These are the numbers that I have been calling "pseudoprimes". Let's go through the steps of computing $\mu_{37}(561)$.

We begin by finding m and k. 561 in binary is 1000110001 (a palindrome!). Then $n - 1 = 560 = (1000110000)_2$, so $k = 4$ and $m = (100011)_2 = 35$. We compute $b_0 = a^m = 37^{35} \bmod 561 = 265$ with the help of the computer. We now compute the sequence of b's, also with the help of the computer. The results are shown in the table below:

i	b_i
0	265
1	100
2	463
3	67
4	1

This sequence ends in 1, but the last non-1 element $b_3 \not\equiv -1 \pmod{561}$, so the test $\mu_{37}(561)$ succeeds. In fact, the test succeeds for every $a \in \mathbf{Z}_{561}^*$ except for $a = 1, 103, 256, 460, 511$. For each of those values, $b_0 = a^m \equiv 1 \pmod{561}$.

71.5 Optimization

In practice, one only wants to compute as many of the b's as necessary to determine whether or not the test succeeds. In particular, one can stop after computing b_i if $b_i \equiv \pm 1 \pmod{n}$. If $b_i \equiv -1 \pmod{n}$ and $i < k$, the test fails. If $b_i \equiv 1 \pmod{n}$ and $i \geq 1$, the test succeeds. This is because we know in this case that $b_{i-1} \not\equiv -1 \pmod{n}$, for if it were, the algorithm would have stopped after computing b_{i-1}.