Scribe Notes for *Algorithmic Number Theory*
Class 10—June 1, 1998

Scribes: Lynn Jones, Nick Loehr, and Hussein Suleman

Abstract

This class began our discussion of arithmetic in the ring $\mathbb{Z}/(n)$. After recalling Fermat's Theorem and stating Euler's Theorem, we analyzed the bit complexities of algorithms for addition, subtraction, multiplication, and exponentiation in $\mathbb{Z}/(n)$. More generally, we considered two versions of the efficient exponentiation algorithm that work in arbitrary monoids.

1 Homework Assignment 2

We started class by going over solutions to the second homework assignment. John presented the solution to problem 1, and Lynn presented the solution to problem 2.

2 The Ring $\mathbb{Z}/(n)$

Recall that $\mathbb{Z}/(n)$ denotes the set of integers modulo n, which consists of the n equivalence classes

$$\{\overline{0}, \overline{1}, \ldots, \overline{n-1}\}.$$

The integers $0, 1, \ldots, n-1$ are the *canonical* representatives of these equivalence classes. $\mathbb{Z}/(n)$ (with the usual definitions of addition and multiplication modulo n) is always a ring. $\mathbb{Z}/(n)$ is also a field if and only if n is a prime. If n is composite, say $n = ab$ where $a > 1$ and $b > 1$, then $\overline{a}$ and $\overline{b}$ are zero divisors in $\mathbb{Z}/(n)$ since $\overline{a} \cdot \overline{b} = \overline{n} = \overline{0}$. Thus, $\mathbb{Z}/(n)$ cannot be a field. Conversely, suppose p is prime. Recall the following theorem, due to Fermat.

Theorem 2.1. *(Fermat's Theorem — cf. Theorem 2.1.3) Suppose p is a prime and a is an integer with* $\gcd(a, p) = 1$. *Then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

From this theorem, it follows that $(\overline{a})^{-1} = \overline{a^{p-2}}$ for any nonzero element $\overline{a} \in \mathbb{Z}/(p)$. Thus, all nonzero elements of $\mathbb{Z}/(p)$ have multiplicative inverses, and $\mathbb{Z}/(p)$ is a field.

More generally, for any $n \geq 1$, define

$$(\mathbb{Z}/(n))^* = \{\overline{a} \in \mathbb{Z}/(n) : \overline{a} \cdot \overline{b} = \overline{1} \text{ for some } \overline{b} \in \mathbb{Z}/(n)\}.$$

$(\mathbb{Z}/(n))^*$ is the set of elements in $\mathbb{Z}/(n)$ that have multiplicative inverses; it is easy to check that this set forms a group under multiplication. It is also easy to check that

$$(\mathbb{Z}/(n))^* = \{\overline{a} : \gcd(a, n) = 1\}.$$

Thus, $(\mathbb{Z}/(n))^*$ consists of $\phi(n)$ distinct elements, since $\phi(n)$ is the number of canonical representatives that are relatively prime to n.

In fact, if we take any $\overline{a} \in (\mathbb{Z}/(n))^*$ and apply the extended Euclidean algorithm to the pair (n, a), we obtain a pair of integers (x, y) such that

$$nx + ay = 1.$$

Thus, $ay \equiv 1 \pmod{n}$, so $\overline{ay} = \overline{1}$ in $\mathbb{Z}/(n)$. Thus, $(\overline{a})^{-1} = \overline{y}$ in $\mathbb{Z}/(n)$. This observation leads to an efficient algorithm for computing multiplicative inverses in $(\mathbb{Z}/(n))^*$ (cf. Section 3.3).

Another method of obtaining the multiplicative inverse of $\overline{a}$ is provided by Euler's Theorem, which is a generalization of Fermat's Theorem.

Theorem 2.2. *(Euler's Theorem — cf. Theorem 2.1.4 and Theorem 5.1.1) Let* $\overline{a} \in (\mathbb{Z}/(n))^*$, *so that* $\gcd(a, n) = 1$. *Then*

$$a^{\phi(n)} \equiv 1 \pmod{n}.$$

Equivalently, $\overline{a}^{\phi(n)} = \overline{1}$ *in* $(\mathbb{Z}/(n))^*$.

If n is prime, we obtain Fermat's Theorem by noting that $(\mathbb{Z}/(n))^* = \mathbb{Z}/(n) - \{0\}$ and $\phi(n) = n - 1$. As an immediate corollary to Euler's Theorem, we see that

$$(\overline{a})^{-1} = \overline{a}^{\phi(n)-1}$$

for $\overline{a} \in (\mathbb{Z}/(n))^*$.

3 Basic Arithmetic in $\mathbb{Z}/(n)$

3.1 Addition and Subtraction

Using the canonical representatives $0, 1, \ldots, n-1$ for $\mathbb{Z}/(n)$ allows us to represent any element $\overline{a} \in \mathbb{Z}/(n)$ using $\lg n$ bits. Thus, to compute $\overline{c} = \overline{a} + \overline{b}$ in $\mathbb{Z}/(n)$:

1. Find $a + b$, the ordinary integer sum of integers a and b. This addition requires $O(\lg n)$ bit operations.

2. Take the sum to be $\overline{c} = (a + b) \bmod n$. Since $0 \leq a + b < 2n$, this reduction can be computed quickly by subtracting n one time if $a + b$ exceeds $n - 1$. This subtraction requires $O(\lg n)$ bit operations.

Hence, the bit complexity for addition in $\mathbb{Z}/(n)$ is $O(\lg n)$. Subtraction is performed in an analogous fashion and also has bit complexity $O(\lg n)$.

3.2 Multiplication

To compute $\overline{c} = \overline{a} \cdot \overline{b}$ in $\mathbb{Z}/(n)$:

1. Find $a \cdot b$, the ordinary integer product of the integers a and b. This multiplication requires $O((\lg n)^2)$ bit operations.

2. Divide the product ab by n to obtain an integer quotient and remainder, viz.

$$ab = qn + r \text{ with } 0 \leq r < n.$$

   This division also requires $O((\lg n)^2)$ bit operations. The product $\overline{c}$ is now given by $\overline{r}$.

Hence, the bit complexity for multiplication in $\mathbb{Z}/(n)$ is $O((\lg n)^2)$.

3.3 Finding Multiplicative Inverses

Since $\mathbb{Z}/(n)$ is not necessarily a field, we cannot, in general, define a division operation. However, we can find a multiplicative inverse for every element in $\mathbb{Z}/(n)$ whose canonical representative is relatively prime to n.

Let $\overline{a} \in (\mathbb{Z}/(n))^*$ so that a is relatively prime to n. Then apply the Extended Euclidean algorithm to (n, a). Assume that this takes m steps. The algorithm will generate a set of coefficients, (x, y), such that $nx + ay = 1$. The integer y is a representative of the multiplicative inverse of $\overline{a}$. To find the canonical representative, we then need to calculate $y \bmod n$.

Now, using the previously derived formula for continuants we know that

$$\begin{bmatrix} n \\ a \end{bmatrix} = \begin{bmatrix} Q[0, m-1] & Q[0, m-2] \\ Q[1, m-1] & Q[1, m-2] \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Bigg|_{X_i = a_i.}$$

The continuants in the above equation are evaluated at the a_i's generated by the Extended Euclidean algorithm.

Then, by inverting the matrix, we get

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = (-1)^m \begin{bmatrix} Q[1, m-2] & -Q[0, m-2] \\ -Q[1, m-1] & Q[0, m-1] \end{bmatrix} \begin{bmatrix} n \\ a \end{bmatrix} \Bigg|_{X_i = a_i.}$$

So the multiplicative inverse of $\overline{a}$ is $y = (-1)^{m+1} Q[0, m-2]|_{X_i = a_i}$.

Now, to achieve a bound on the time complexity, it is desirable that y not be too large.

It can be observed that the highest order term in $Q[0, m-2]$ is $X_0 X_1 \ldots X_{m-2}$. By Exercise 4.5, it has been proven that $|a_0 a_1 \ldots a_{m-2}| < n$. Thus, asymptotically, the calculation of y is dominated by the highest order term, which is $O(n)$. Thus, y can be represented in $O(\lg n)$ bits.

Now, to calculate $y \bmod n$, we can perform the division by repeated subtraction, exploiting the fact that y is $O(n)$. Since the number of subtractions will be bounded by a constant, the computation is $O(\lg n)$.

Corollary 3.1. *If $\overline{a} \in \mathbb{Z}/(n)$ with $\gcd(a, n) = 1$, then we can compute $(\overline{a})^{-1}$ in $O((\lg a)(\lg n))$ bit operations.*

Thus, using this construction, we can compute multiplicative inverses efficiently.

4 Exponentiation

4.1 An Efficient Algorithm for Exponentiation

Assume $(S, \cdot)$ is a semigroup. This implies that an associative binary operation is defined on S. Assume also that a multiplicative identity, usually denoted 1, exists for this semigroup (this is necessary to define $a^0 = 1$ for any $a \in S$). If S does not have an identity, we can easily add one to S to create a monoid.

Denote the exponentiation operation as a^e for $e \geq 0$ and $a \in S$.

There exist many different approaches for evaluating this expression. The naive approach simply computes the product of e a's. This can, however, be very time-consuming if e is large.

An alternative approach is to exploit the observation that any power can be expressed by the following recursive definition:

$$a^e = \begin{cases} a \cdot a^{e-1} & \text{if } e \text{ is odd;} \\ \left(a^{e/2}\right)^2 & \text{if } e > 0 \text{ and } e \text{ is even;} \\ 1 & \text{if } e = 0. \end{cases}$$

This definition can be recast as a recursive algorithm to calculate powers as follows:

```
Power (a, e)
(1)  if e = 0 then return 1;
(2)  else if e mod 2 = 0 then
(3)     return (Power (a, e/2)^2);
     else
(4)     return a · Power (a, e − 1);
```

Example 4.1. For this example, we will use 2×2 matrices of integers. Note that matrix multiplication is associative, and has an identity, namely the 2×2 identity matrix I. We trace a sample execution of the algorithm in the following table; the recursive calls are listed going down and the resulting operations are listed from the bottom up.

First, define matrix A:

$$\begin{pmatrix} -1 & 3 \\ 2 & 1 \end{pmatrix}$$

We wish to find A^9.

$Power(A, 9)$			AA^8	$\begin{pmatrix} -2401 & 7203 \\ 4802 & 2401 \end{pmatrix}$
A	$Power(A, 8)$		$(A^4)^2$	$\begin{pmatrix} 2401 & 0 \\ 0 & 2401 \end{pmatrix}$
	$Power(A, 4)$		$(A^2)^2$	$\begin{pmatrix} 49 & 0 \\ 0 & 49 \end{pmatrix}$
	$Power(A, 2)$		A^2	$\begin{pmatrix} 7 & 0 \\ 0 & 7 \end{pmatrix}$
	$Power(A, 1)$		AI	$\begin{pmatrix} -1 & 3 \\ 2 & 1 \end{pmatrix}$
	A	I		

Table 1: Recursive calls in power algorithm to find A^9.

You can see from the table that this recursive call requires three squarings and two multiplications by the matrix A, whereas the naive approach would have required eight multiplications by A.

Complexity Analysis

Suppose `Power` is called with a nonzero exponent. This exponent is either even or odd. In the former case, the recursive call in line (3) executes immediately and cuts the size of the exponent in half. In the latter case, the exponent is reduced by one in line (4), but the next recursive call is on an even exponent. Hence, after at most two recursive calls, the size of the exponent is always halved. Thus, the number of recursive function calls is $O(\lg e)$. The time complexity is thus $O(s \lg e)$, where s is the complexity of multiplication in the semigroup $(S, \cdot)$. To particularize this algorithm to $\mathbb{Z}/(n)$, in which we are computing $a^e \bmod n$ for $0 \leq a \leq n$ and $e \geq 0$, this algorithm requires $O((\lg n)^2(\lg e))$ bit operations (Theorem 5.4.1).

4.2 An Alternative Algorithm for Exponentiation

The exponent, e, in its binary representation, can be expressed as a sum of powers of 2:

$$e = b_k 2^k + b_{k-1} 2^{k-1} + \cdots + b_1 2 + b_0.$$

Then, for $e > 0$, $b_k = 1$ and $b_i \in \{0, 1\}$ for $0 \leq i < k$. So we have

$$a^e = a^{\sum_{i=0}^{k} b_i 2^i} = \prod_{i=0}^{k} a^{b_i 2^i}.$$

Each a_i in the product will be either 1 (when $b_i = 0$) or will be a^{2^i}. The steps to the alternative algorithm are to create a table of a^{2^i}, and use the table to look up the factors in

$$a^e = \prod_{\substack{0 \leq i \leq k \\ b_i = 1}} a^{2^i}.$$

Complexity Analysis

Building the table of powers requires $O(\lg e)$ squarings and $O(\lg e)$ multiplications. For this algorithm in particular, we must also examine its space requirements. (Although the recursive *Power* algorithm has space requirements for the stack, it could be rewritten iteratively. In this case, storage of the table is an explicit part of the algorithm.) The table lookup algorithm requires storage space for $\Theta(\lg e)$ elements of S.

This alternative algorithm has the same time complexity and potentially greater space complexity than the `Power` algorithm, but it is useful when the computations may be repeated and particularly when something is known about the exponent. Notice that there are $O(\lg e)$ multiplications for this algorithm. The exact number of multiplications depends not only on the size of e, but also on how many bits in e's binary representation are 1 (i.e., how many $b_i = 1$). If, for example, e is some power of 2, this algorithm has much better time complexity than the previous one.

5 Next Class

In the next class, we will conclude the discussion of the exponentiation algorithms and begin talking about the Chinese Remainder Theorem.

References

[1] E. Bach and J. Shallit, *Algorithmic Number Theory*, The MIT Press, Cambridge, Massachusetts, 1996.