Lecture Notes 8

39 Exponentiation: Speeding up the Computation

In section 38 (lecture notes 7), we described how to control the growth in the lengths of numbers when computing $m^e \bmod n$, for numbers m, e, and n which are 1024 bits long. Nevertheless, there is still a problem with the naive exponentiation algorithm that simply multiplies m by itself a total of $e-1$ times. Since the value of e is roughly 2^{1024}, that many iterations of the main loop would be required, and the computation would run longer than the current age of the universe (which is estimated to be 15 billion years). Assuming one loop iteration could be done in one microsecond (very optimistic seeing as each iteration requires computing a product and remainder of big numbers), only about 30×10^{12} iterations could be performed per year, and only about 450×10^{21} iterations in the lifetime of the universe. But $450 \times 10^{21} \approx 2^{79}$, far less than $e-1$.

The trick here is to use a more efficient exponentiation algorithm based on repeated squaring. To compute $m^e \bmod n$ where $e = 2^k$ is a power of two requires only k squarings, i.e., one computes

$$\begin{aligned} m_0 &= m \\ m_1 &= (m_0 * m_0) \bmod n \\ m_2 &= (m_1 * m_1) \bmod n \\ &\vdots \\ m_k &= (m_{k-1} * m_{k-1}) \bmod n. \end{aligned}$$

Clearly, each $m_i = m^{2^i} \bmod n$. m^e for values of e that are not powers of 2 can be obtained as the product modulo n of certain m_i's. In particular, express e in binary as $e = (b_s b_{s-1} \ldots b_2 b_1 b_0)_2$. Then m_i is included in the final product if and only if $b_i = 1$.

It is not necessary to perform this computation in two phases as described above. Rather, the two phases can be combined together, resulting in a slicker and simpler algorithm that does not require the explicit storage of the m_i's. I will give two versions of the resulting algorithm, a recursive version and an iterative version. I'll write both in C notation, but it should be understood that the C programs only work for numbers smaller than 2^{16}. To handle larger numbers requires the use of big number functions.

```
/* computes m^e mod n recursively */
int modexp( int m, int e, int n)
{
  int r;
  if ( e == 0 ) return 1;          /* m^0 = 1 */
  r = modexp(m*m % n, e/2, n);     /* r = (m^2)^(e/2) mod n */
  if ( (e&1) == 1 ) r = r*m % n;   /* handle case of odd e */
  return r;
}
```

This same idea can be expressed iteratively to achieve even greater efficiency.

```
/* computes m^e mod n iteratively */
int modexp( int m, int e, int n)
{
  int r = 1;
  while ( e > 0 ) {
    if ( (e&1) == 1 ) r = r*m % n;
    e /= 2;
    m = m*m % n;
  }
  return r;
}
```

The loop invariant is $e > 0 \wedge (m_0^{e_0} \bmod n = rm^e \bmod n)$, where m_0 and e_0 are the initial values of m and e, respectively. It is easily checked that this holds at the start of each iteration. If the loop exits, then $e = 0$, so r is the desired result. Termination is ensured since e gets reduced during each iteration.

Note that the last iteration of the loop computes a new value of m that is never used. A slight efficiency improvement results from restructuring the code to eliminate this unnecessary computation. Following is one way of doing so.

```
/* computes m^e mod n iteratively */
int modexp( int m, int e, int n)
{
  int r = ( (e&1) == 1 ) ? m % n : 1;
  e /= 2;
  while ( e > 0 ) {
    m = m*m % n;
    if ( (e&1) == 1 ) r = r*m % n;
    e /= 2;
  }
  return r;
}
```

40 Number Theory Review

We next review some number theory that is needed for understanding RSA. These lecture notes only provide a high-level overview. Further details are contained in course handouts 4–6 and in Chapter 3 of the textbook.

40.1 Divisibility properties

Let a, b be integers and assume $b > 0$. The *division theorem* asserts that there are unique integers q (the *quotient*) and r (the *remainder*) such that $a = bq + r$ and $0 \leq r < b$. In case $r = 0$ we say that b *divides* a (exactly) and write $b \,|\, a$.

Fact If $d \,|\, (a + b)$, then either d divides both a and b, or d divides neither of them.

To see this, suppose $d \,|\, (a + b)$ and $d \,|\, a$. Then by the division theorem, $a + b = dq_1$ and $a = dq_2$ for some integers q_1 and q_2. Subsituting for a and solving for b, we get

$$b = dq_1 - dq_2 = d(q_1 - q_2).$$

But this implies $d \mid b$, again by the division theorem.

40.2 Greatest common divisor

The *greatest common divisor* of two integers a and b, written $\gcd(a, b)$, is the largest integer d such that $d \mid a$ and $d \mid b$. The $\gcd$ is always defined since 1 is a divisor of every integer, and the divisor of a number cannot be larger (in absolute value) than the number itself.

The gcd of a and b is easily found if a and b are already given in factored form. Namely, let p_i be the i^{th} prime and write $a = \prod p_i^{e_i}$ and $b = \prod p_i^{f_i}$. Then $\gcd(a, b) = \prod p_i^{\min(e_i, f_i)}$. However, factoring is believed to be a hard problem, and no polynomial-time factorization algorithm is currently known. Indeed, if it were, then Eve could use it to break RSA, and RSA would be of no interest as a cryptosystem.

Fortunately, $\gcd(a, b)$ can be computed efficiently without the need to factor a and b. Here's a sketch of the ideas that lead to the famous *Euclidean algorithm*.

The $\gcd$ function satisfies several identities. In the following, assume $a \geq b \geq 0$:

$$\gcd(a, b) = \gcd(b, a) \tag{1}$$

$$\gcd(a, 0) = a \tag{2}$$

$$\gcd(a, b) = \gcd(a - b, b) \tag{3}$$

Identity 3 follows from the Fact above. A simple inductive proof shows that identity 3 can be strengthened to

$$\gcd(a, b) = \gcd(a \bmod b, b) \tag{4}$$

where $a \bmod b$ is the remainder of a divided by b. The Euclidean algorithm uses identities 1, 2, and 4 recursively to compute $\gcd(a, b)$ in $O(n)$ stages, where n is the sum of the lengths of a and b when written in binary notation, and each stage requires at most one remainder computation. We will return to this topic in lecture 9.

40.3 Basic definitions and notation

The set

$$\mathbf{Z}_n = \{0, 1, \ldots, n - 1\}$$

contains the non-negative integers less than n. If one defines a binary "addition" operation on $\mathbf{Z}_n$ by

$$a \oplus b \stackrel{\text{df}}{=} (a + b) \bmod n$$

then $\mathbf{Z}_n$ can be regarded as an Abelian group under addition ($\oplus$).

The set

$$\mathbf{Z}_n^* = \{x \in \mathbf{Z}_n \mid \gcd(x, n) = 1\}$$

contains the non-negative integers less than n that are *relatively prime* to n, that is, which do not share any non-trivial common factor with n. If one defines a binary "multiplication" operation on $\mathbf{Z}_n^*$ by

$$a \otimes b \stackrel{\text{df}}{=} (a \cdot b) \bmod n$$

then it can be shown that $\mathbf{Z}_n^*$ is an Abelian group under multiplication ($\otimes$).

Euler's totient (ϕ) function is defined to be the cardinality of $\mathbf{Z}_n^*$:

$$\phi(n) = |\mathbf{Z}_n^*|$$

Properties of $\phi(n)$:

1. If p is prime, then $\phi(p) = p - 1$.

2. More generally, if p is prime and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1} = (p-1)p^{k-1}$.

3. If $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m)\phi(n)$.

These properties enable one to compute $\phi(n)$ for all $n \geq 1$ provided one knows the factorization of n. For example,

$$\phi(126) = \phi(2)\phi(3^2)\phi(7) = (2-1)(3-1)(3^{2-1})(7-1) = 1 \cdot 2 \cdot 3 \cdot 6 = 36.$$

The 36 elements of $\mathbf{Z}^*_{126}$ are: 1, 5, 11, 13, 17, 19, 23, 25, 29, 31, 37, 41, 43, 47, 53, 55, 59, 61, 65, 67, 71, 73, 79, 83, 85, 89, 95, 97, 101, 103, 107, 109, 113, 115, 121, 125.

41 Modular Arithmetic

There are several closely-related notions associated with "mod".

First of all, $\bmod$ is a binary operator. If $a \geq 0$ and $b \geq 1$ are integers, then $a \bmod b$ is the remainder of a divided by b. When either a or b is negative, there is no consensus on the definition of $\bmod$. We are only interested in $\bmod$ for positive b, and we find it convenient in that case to define $(a \bmod b)$ to be the smallest non-negative integer r such that $a = bq + r$ for some integer q. Under this definition, we always have that $r = (a \bmod b) \in \mathbf{Z}_b$. For example $(-5 \bmod 3) = 1 \in \mathbf{Z}_3$ since for $q = -2$, we have $-5 = 3 \cdot (-2) + 1$. Note that in the C programming language, the mod operator % is defined differently, so $a \,\%\, b \neq a \bmod b$ when a is negative and b positive.[1]

Mod is also used to define a relationship on integers:

$$a \equiv b \pmod{n} \quad \text{iff} \quad n \,|\, a - b.$$

That is, a and b have the same remainder when divided by n. An immediate consequence of this definition is that

$$a \equiv b \pmod{n} \quad \text{iff} \quad (a \bmod n) = (b \bmod n).$$

Thus, the two notions of mod aren't so different after all!

When n is fixed, the resulting two-place relationship $\equiv$ is an equivalence relation. Its equivalence classes are called *residue* classes modulo n and are denoted using the square-bracket notation $[b] = \{a \mid a \equiv b \pmod{n}\}$. For example, for $n = 7$, we have $[10] = \{\ldots -11, -4, 3, 10, 17, \ldots\}$. Clearly, $[a] = [b]$ iff $a \equiv b \pmod{n}$. Thus, $[-11]$, $[-4]$, $[3]$, $[10]$, $[17]$ are all names for the same equivalence class. We choose the unique integer in the class that is also in $\mathbf{Z}_n$ to be the *canonical* or preferred name for the class. Thus, the canonical name for the class containing 10 is $[10 \bmod 7] = [3]$.

The relation $\equiv \pmod{n}$ is a *congruence* relation with respect to addition, subtraction, and multiplication of integers. This means that for each of these arithmetic operations $\odot$, if $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then $a \odot b \equiv a' \odot b' \pmod{n}$. This implies that the class containing the result of $a + b$, $a - b$, or $a \times b$ depends only on the classes to which a and b belong and not the

[1] For those of you who are interested, the C standard defines $a \,\%\, b$ to be the number satisfying the equation $(a/b) * b + (a \,\%\, b) = a$. C also defines a/b to be the result of rounding the real number a/b towards zero, so $-5/3 = -1$. Hence, $-5 \,\%\, 3 = -5 - (-5/3) * 3 = -5 + 3 = -2$.

particular representatives chosen. Hence, we can define new addition, subtraction, and multiplication as operations on equivalence classes, or alternatively, regard them as operations directly on $\mathbf{Z}_n$ defined by

$$\begin{aligned} a \oplus b &= (a + b) \bmod n \\ a \ominus b &= (a - b) \bmod n \\ a \otimes b &= (a \times b) \bmod n \end{aligned} \tag{5}$$

We remark that $\otimes$ is defined on all of $\mathbf{Z}_n$, but if a and b are both in $\mathbf{Z}_n^*$, then $a \otimes b$ is also in $\mathbf{Z}_n^*$.

42 Modular Exponentiation and Euler's Theorem

Recall the RSA encryption and decryption functions

$$\begin{aligned} E_e(m) &= m^e \bmod n \\ D_d(c) &= c^d \bmod n \end{aligned}$$

where $n = pq$ is the product of two distinct large primes p and q. We see that both are based on modular exponentiation of large integers, an operation that we now explore in some depth.

We mentioned in section 40.3 that $\mathbf{Z}_n^*$ is an Abelian group under $\otimes$. This means that it satisfies the following properties:

Associativity $\otimes$ is an associative binary operation on $\mathbf{Z}_n^*$. In particular, $\mathbf{Z}_n^*$ is closed under $\otimes$.

Identity 1 is an identity element for $\otimes$ in $\mathbf{Z}_n^*$, that is $1 \cdot x = x \cdot 1 = x$ for all $x \in \mathbf{Z}_n^*$.

Inverses For all $x \in \mathbf{Z}_n^*$, there exists another element $x^{-1} \in \mathbf{Z}_n^*$ such that $x \cdot x^{-1} = x^{-1} \cdot x = 1$.

Commutativity $\otimes$ is commutative. (This is only true for *Abelian* groups.)

Example: Let $n = 26 = 2 \cdot 13$. Then

$$\mathbf{Z}_{26}^* = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$$

and

$$\phi(26) = |\mathbf{Z}_{26}^*| = 12.$$

The inverses of the elements in $\mathbf{Z}_{26}^*$ are given in Table 1. The bottom row of the table gives equiva-

Table 1: Table of inverses in $\mathbf{Z}_{26}^*$.

x	1	3	5	7	9	11	15	17	19	21	23	25
x^{-1}	1	9	21	15	3	19	7	23	11	5	17	25
$=$	1	9	-5	-11	3	-7	7	-3	11	5	-9	-1

lent integers in the range $[-12, \ldots, 13]$. This makes it apparent that $(26 - x)^{-1} = -x^{-1}$. In other words, the last row reads back to front the same as it does from front to back except that all of the signs flip from $+$ to $-$ or $-$ to $+$, so once the inverses for the first six numbers are known, the rest of the table is easily filled in.

It is not obvious from what I have said so far that inverses always exist for members of $\mathbf{Z}_n^*$,, and even showing that $\mathbf{Z}_n^*$ is closed under $\otimes$ takes a bit of work. Nevertheless, both are true. The latter isn't too hard for you to work out for yourself, and the former will become apparent later when we show how to compute the inverse.

Recall Euler's ϕ function which was defined in section 40.3 to be $|\mathbf{Z}_n^*|$, the cardinality of $\mathbf{Z}_n^*$. From the properties given there, one can derive an explicit formula for $\phi(n)$.

Theorem 1 *Write n in factored form, so $n = p_1^{e_1} \cdots p_k^{e_k}$. where $p_1, \ldots, p_k$ are distinct primes and $e_1, \ldots, e_k$ are positive integers.*[2] *Then*

$$\phi(n) = (p_1 - 1) \cdot p_1^{e_1 - 1} \cdots (p_k - 1) \cdot p_k^{e_k - 1}.$$

When p is prime, we have simply $\phi(p) = (p - 1)$, and for the product of two distinct primes, $\phi(pq) = (p-1)(q-1)$. Thus, $\phi(26) = (13-1)(2-1) = 12$, as we have seen.

A general property of finite groups is that if any element x is repeatedly multiplied by itself, the result is eventually 1. That is, 1 appears in the sequence $x, (x \otimes x), (x \otimes x \otimes x), \ldots$, after which the sequence repeats. For example, for $x = 5$ in $\mathbf{Z}_{26}^*$, we get the sequence $5, 25, 21, 1, 5, 25, 21, 1, \ldots$. The result of multiplying x by itself k times can be written x^k. The smallest integer k for which $x^k = 1$ is called the *order* of x, sometimes written $\mathrm{ord}(x)$. It follows from general properties of groups that the order of any element of a group divides the order of the group. For $\mathbf{Z}_n^*$, we therefore have $\mathrm{ord}(x) \,|\, \phi(n)$. From this fact, we immediately get

Theorem 2 (Euler's theorem) $x^{\phi(n)} \equiv 1 \pmod{n}$ *for all $x \in \mathbf{Z}_n^*$.*

As a special case, we have

Theorem 3 (Fermat's theorem) $x^{(p-1)} \equiv 1 \pmod{p}$ *for all x, $1 \le x \le p-1$, where p is prime.*

Corollary 4 *Let $r \equiv s \pmod{\phi(n)}$. Then $a^r \equiv a^s \pmod{n}$ for all $a \in \mathbf{Z}_n^*$.*

Proof: If $r \equiv s \pmod{\phi(n)}$, then $r = s + u\phi(n)$ for some integer u. Then using Euler's theorem, we have

$$a^r \equiv a^{s+u\phi(n)} \equiv a^s \cdot (a^u)^{\phi(n)} \equiv a^s \cdot 1 \equiv a^s \pmod{n},$$

as desired. ∎

The importance of this corollary to RSA is that it gives us a condition on e and d that ensures the resulting cryptosystem works. That is, if we require that

$$ed \equiv 1 \pmod{\phi(n)}, \tag{6}$$

then it follows from Corollary 4 that $D_d(E_e(m)) = m^{ed} \equiv m \pmod{n}$ for all messages $m \in \mathbf{Z}_n^*$, so $D_d()$ really does decrypt messages in $\mathbf{Z}_n^*$ that are encrypted by $E_e()$.

What about the case of messages $m \in \mathbf{Z}_n - \mathbf{Z}_n^*$? There are several answers to this question.

1. For such m, either $p \mid m$ or $q \mid m$ (but not both because $m < pq$). If Alice ever sends such a message and Eve is astute enough to compute $\gcd(m, n)$ (which she can easily do), then Eve will succeed in breaking the cryptosystem. So Alice doesn't really want to send such messages if she can avoid it.

[2] By the fundamental theorem of arithmetic, every integer can be written uniquely in this way up to the ordering of the factors.

2. If Alice sends random messages, her probability of choosing a message not in $\mathbf{Z}_n^*$ is only about $2/\sqrt{n}$. This is because the number of "bad" messages is only $n-\phi(n) = pq-(p-1)(q-1) = p+q-1$ out of a total of $n = pq$ messages altogether. If p and q are both 512 bits long, then the probability of choosing a bad message is only about $2 \cdot 2^{512}/2^{1024} = 1/2^{511}$. Such a small probability event will likely never occur during the lifetime of the universe.

3. For the purists out there, RSA does in fact work for all $m \in \mathbf{Z}_n$, even though Euler's theorem fails for $m \notin \mathbf{Z}_n^*$. For example, if $m = 0$, it is clear that $(0^e)^d \equiv 0 \pmod{n}$, yet Euler's theorem fails since $0^{\phi(n)} \not\equiv 1 \pmod{n}$. We omit the proof of this curiosity.