Modular Exponentiation of Integers

(Handout November 12, 2008)

> "Space is big. Really big. You just won't believe how vastly hugely mind-bogglingly big it is. I mean, you may think it's a long way down the road to the chemist, but that's just peanuts to space."
>
> —Douglas Adams, *The Hitchhiker's Guide to the Galaxy*.

> "I believe there are 15,747,724,136,275,002,577,605,653,961,181,555,468,044,717,914,527,116,709,366,231,425,076,185,631,031,296 protons in the universe and the same number of electrons."
>
> —Sir Arthur Eddington, *The Nature of the Physical World*, 1928.

Big Numbers

Let's warm up by mentioning some large numbers. For starters we have Eddington's number $N_{\text{Edd}} \approx 1.5\times10^{79}$ given above. On mostly philosophical grounds, Eddington proposed this as the number of protons (and electrons) in the universe. (No, he didn't claim to have counted them all.) We immediately face the issue: for numbers of this magnitude, can the exact value ever really matter? It is quite conceivable that the observable properties of the universe would remain pretty much the same with a slightly larger number

$$N_{\text{Edd}}+118=157477241362750025776056539611815554680447179145271167093662314250761856310314 14$$

of protons instead of N_{Edd} protons.

On the other hand, exact values are sometimes important. For example if a large number is a password for my preferred wireless internet connection, then without the *exact* sequence of digits, my laptop won't be able to access the internet (even approximately)! More generally for our interest, numbers (as sequences of digits or bits) often represent some encoded information, rather than counting any particular set of objects. And the *exact* sequence of digits (or bits) is often vitally important.

Current estimates for the total number of elementary particles in the universe (protons, neutrons, electrons, muons, neutrinos, etc.) range from 10^{72} to 10^{87}. Let's take this number to be

$$10^{80}=100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.$$

This number is peanuts next to the kind of numbers we will routinely consider. Let's move up to a googol:

$$\begin{aligned} 1 \text{ googol } &= 1 \text{ followed by 100 zeroes} \\ &= 10^{100} \\ &= 10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000. \end{aligned}$$

We're still just getting started! Consider next the googolplex:

$$\begin{aligned} 1 \text{ googolplex } &= 10^{10^{100}} = 1 \text{ followed by a googol zeroes} \\ &= 1000000000000000000000000000000000000000000000000\ldots \end{aligned}$$

This number is so large, we cannot possibly print out all its decimal digits. Even if every electron in the universe were a sheet of paper, there would not be enough paper in the universe to print out this number in decimal notation!

But we are about to encounter numbers much bigger even than this.

Modular Exponentiation

Let x, m and n be positive integers (typically many digits long). A practical concern in implementing many cryptographic protocols is the computation of

$$x^m \mod n,$$

by which we mean finding the unique integer in the range $\{0, 1, 2, \ldots, n-1\}$ congruent to x^m modulo n (i.e. the remainder when x^m is divided by n). A first observation is that if x and m are hundreds of digits long (as is typically the case in our intended applications), it is *impossible* to first evaluate x^m and then reduce modulo n. This is because x^m has googols of digits! No physical computer has either the time or space resources to compute this number. Even if a computer could imagine this number, there is no way it could print it out! Even if every elementary particle in the universe could serve as a sheet of paper, it would take a million million million million million million million million of our universes to print out just the first googol digits of x^m. And if x and m are 200 digits each, the number of digits in x^m *vastly* exceeds a googol. Moreover the computation of x^m would require an execution time vastly exceeding the lifetime of the universe, even with the fastest computer that could theoretically be built. This is why a MAPLE command like

```
> x:=105619323236521942515637474091265961025712465498125 0;

                x := 105619323236521942515637474091265961025712465498125 0

> m:=435724089518651052301657460126517160147234615623298571;

                m := 435724089518651052301657460126517160147234615623298571

> n:=105715612365102560152365217652104165129865312 36;

                n := 105715612365102560152365217652104165129865312 36

> x^m mod n;

Error, numeric exception: overflow

>
```

is doomed to fail (the computer indicates a numeric overflow).

On the other hand, if n is only 200 digits long, then $x^m \mod n$ cannot have more than 200 digits; and it is possible to compute this value very easily! The MAPLE syntax for this computation is as follows:

```
> x&^m mod n;

        189566205862495862910357889328655462031050236 0

>
```

How does MAPLE accomplish such a prodigious calculation? We'll illustrate with a smaller example to get the idea: consider

$$16243^{77} \mod 622.$$

We first compute the binary expansion of 77 using repeated division by 2:

$$\begin{array}{r|l} 2 & 77 \\ \hline 2 & 38 \text{ r } 1 \\ \hline 2 & 19 \text{ r } 0 \\ \hline 2 & 9 \text{ r } 1 \\ \hline 2 & 4 \text{ r } 1 \\ \hline 2 & 2 \text{ r } 0 \\ \hline 2 & 1 \text{ r } 0 \\ \hline & 0 \text{ r } 1 \end{array}$$

so that the decimal number 77 is written in binary as 1001101 (notice that we *reverse* the list of remainders). So

$$77 = 1\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$

or simply

$$77 = 64 + 8 + 4 + 1.$$

Now start with $x = 16243 \equiv 71 \mod 622$ and square six times modulo n (six being one less than seven, the bit-length of 1001101):

$$\begin{aligned} x &\equiv 71 \mod 622 \\ x^2 &\equiv 65 \\ x^4 &\equiv 493 \\ x^8 &\equiv 469 \\ x^{16} &\equiv 395 \\ x^{32} &\equiv 525 \\ x^{64} &\equiv 79 \end{aligned}$$

Now refer to the 1's in the binary expansion of 77, and multiply together the corresponding powers of x modulo n, thus:

$$\begin{aligned} x^{64+8} &\equiv 79{\times}469 \equiv 353 \\ x^{64+8+4} &\equiv 353{\times}493 \equiv 491 \\ x^{77} = x^{64+8+4+1} &\equiv 491{\times}71 \equiv 29 \mod 622 \end{aligned}$$

We refer to this algorithm, using the binary representation of m in the computation of $x^m \mod n$, as *binary exponentiation*.

What if x, m and n are each about 200 digits long? Then the binary representation of m has about 670 bits, so we must perform 670 squaring operations (modulo n). Each one generates in a 400-digit number, which (after reduction mod n) results in a 200-digit number. If half the bits in the binary representation of m are 1's, then we need to perform another 335 multiplications mod n, each time saving the computed power as a 200-digit number. Altogether we have about 1000 multiplication operations to perform, each of which might take a microsecond to perform; and we have our final answer for $x^m \mod n$ in about a millisecond. I still find it impressive that such a computation is possible!

But other methods are sometimes available, and sometimes even faster.

Euler's Formula

For any positive integer n, we define $\phi(n)$ to be the number of integers in the range $1, 2, 3, \ldots, n$ which are relatively prime to n, i.e.

$$\phi(n) = \big|\{k \in \mathbb{Z} \ : \ 1 \le k \le n, \ \gcd(k, n) = 1\}\big|.$$

This is known as *Euler's totient function*. To compute $\phi(n)$ for small values of n, we may list the numbers from 1 to n, and then cross out those having a factor bigger than 1 in common with n. Thus for example $\phi(12) = 4$ since four numbers remain in the list

$$1, \quad \cancel{2}, \quad \cancel{3}, \quad \cancel{4}, \quad 5, \quad \cancel{6}, \quad 7, \quad \cancel{8}, \quad \cancel{9}, \quad \cancel{10}, \quad 11, \quad \cancel{12}.$$

In particular it is easy to see that $\phi(p) = p - 1$ whenever p is prime. More generally, if one knows the prime factorization of n, then the value of $\phi(n)$ is found directly as follows: if

$$n = p_1^{r_1} p_2^{r_2} \cdots p_k^{r_k}$$

where $p_1, p_2, \ldots, p_k$ are the distinct prime divisors of n, and if all the exponents $r_i \geq 1$, then we have

$$\begin{aligned} \phi(n) &= n\big(1 - \tfrac{1}{p_1}\big)\big(1 - \tfrac{1}{p_2}\big) \cdots \big(1 - \tfrac{1}{p_k}\big) \\ &= p_1^{r_1 - 1}(p_1 - 1) \times p_2^{r_2 - 1}(p_2 - 1) \times \cdots \times p_k^{r_k - 1}(p_k - 1). \end{aligned}$$

Thus, for example,

$$\phi(12) = \phi(2^2 \times 3) = 12\big(1 - \tfrac{1}{2}\big)\big(1 - \tfrac{1}{3}\big) = 12\big(1 - \tfrac{1}{2}\big)\big(1 - \tfrac{1}{3}\big) = 12 \times \tfrac{1}{2} \times \tfrac{2}{3} = 4.$$

Theorem (Euler). If x and n are relatively prime positive integers, then

$$x^{\phi(n)} \equiv 1 \mod n.$$

We will prove this result in a later section; but first observe that this gives Fermat's Little Theorem in the special case when n is prime.

Example. Find the last two digits of 4567^{123}.

Solution. We are really asking for the value of $4567^{123} \mod 100$. Observe that

$$gcd(4567, 100) = 1 \quad \text{and} \quad \phi(100) = 100\big(1 - \tfrac{1}{2}\big)\big(1 - \tfrac{1}{5}\big) = 100 \times \tfrac{1}{2} \times \tfrac{4}{5} = 40.$$

So by Euler's Theorem,

$$4567^{40} \equiv 67^{40} \equiv 1 \mod 100.$$

Now

$$4567^{123} \equiv 67^{123} \equiv 67^{3 \times 40 + 3} \equiv (67^{40})^3 \times 67^3 \equiv 1 \times 67^3 \equiv 67^3 \equiv 63 \mod 100.$$

The last computation follows directly using a calculator: $67^3 = 300763$. If the numbers in question were larger, we might have resorted at this point to binary exponentiation; but the numbers are much smaller than in the original problem, so our use of Euler's Theorem has significantly reduced the computational effort required.

The preceding example illustrates an alternative approach to evaluating $x^m \mod n$, which applies in many important cases: If $gcd(x, n) = 1$, we may first simplify the problem by reducing m modulo $\phi(n)$. (We also reduce x modulo n in all cases. Note the different modulus here! i.e. $\phi(n)$ for the exponent, versus n for the base.)

Solving for x

Now consider a variant of the preceding problems, one that typically arises when considering how to break encrypted messages. We are given integers m, n and a, and we are required to solve for x in the congruence relation $x^m \equiv a \mod n$. For example, we ask: Find an integer x such that

$$x^{9108163} \equiv 3208718 \mod 130355971.$$

In principle one might try all x-values in the range 0,1,2,...,130355970 to see which ones satisfy this condition; but that is likely to require far too much execution time. Certainly if n were replaced by a much larger number, say a 200-digit number, this approach would be out of the question.

To get an idea of how to proceed, consider the analogous problem over $\mathbb{R}$. Here we would be asked to solve the equation

$$x^{9108163} = 3208718$$

for $x \in \mathbb{R}$. You'll recognize that this requires us to take 9108163^{th} roots of both sides:

$$(x^{9108163})^{1/9108163} = 3208718^{1/9108163}$$
$$x = x^1 \approx 1.000001645$$

This cannot be the answer in the integers modulo 130355971; however, the idea is a good one. Starting with

$$x^{9108163} \equiv 3208718 \mod 130355971$$

the problem is essentially one of raising both sides to the k^{th} power, where k is the inverse of 9108163 mod $\phi(130355971)$; remember (from the previous section) that the exponent is defined modulo $\phi(130355971)$. In order to simplify computations, let's use MAPLE to do all this:

```
> with(numtheory):
Warning, the protected name order has been redefined and unprotected
> n:=130355971:
> phi(n);
                                  130096312
> m:=9108163:
> k:=1/m mod phi(n);
                                  k := 19904051
> x:=3208718&^k mod n;
                                  x := 98725341
```

Check that this is in fact a solution:

```
> x&^m mod n;
                                  3208718
```

Good!

Note that if n has 200 or more decimal digits, we will be unable to factor n, and hence unable to compute $\phi(n)$ (our formula for $\phi(n)$ requires knowing the factorization of n). In this case our method will not work. Is there a way to solve for x in this case? Nobody knows. Likely not! Or if there is, this would undermine the security of some of the most popular public key cryptosystems in use today (details about this in the next handout).

Solving for m

Here is another variation on the theme above. This is another question that a cryptanalyst (code-breaker) might face. We are given integers x, n and a, and we are required to solve for m in the relation $x^m \equiv a \mod n$. For example, we ask: Find an integer m such that

$$542276103^m \equiv 478354870 \mod 1651057907.$$

Here at least you can check easily that $n = 1651057907$ is prime. (Recall that although factorization is hard, checking to see if a number is prime can be done efficiently, even for large numbers having thousands of digits.) But this doesn't help much. One idea is to try all values of $m = 0, 1, 2, \ldots$ etc. Some MAPLE code for this approach is shown:

```
> x:=542276103;
                                  x := 542276103
> n:=1651057907;
                                  n := 1651057907
```

We check that n is prime:

```
> ifactor(n);
                                  (1651057907)
> m:=0;
                                  m := 0
> while (x&^m mod n)<>478354870 do
>   m:=m+1:
>   od:
```

Given enough time (hours maybe?) the last loop will eventually terminate and then we can print out the smallest value of m that solves the required congruence. However I didn't have the patience to wait for an answer; my screen showed the hourglass symbol while MAPLE chugged away for several minutes looking for a solution. Of this much I will assure you: that there is a solution. There are more efficient ways to compute m than the 'while' loop I have shown; but no known method works for really large values of x, n and a (i.e. hundreds of digits long).

For comparison, consider the analogous problem over $\mathbb{R}$: here we are asked to find $m \in \mathbb{R}$ such that

$$542276103^m = 478354870.$$

Taking the logarithm of both sides to base 542276103, this says simply that

$$m = \log_{542276103}(478354870) \approx 0.9937635803\ .$$

Unlike the previous section where we solved for x (essentially by taking appropriate powers of both sides), this situation offers no suggestion for how to proceed in solving the original congruence modulo n. However it does offer an appropriate name for our problem: the value of m is called the *discrete logarithm* of 478354870 to base 542276103. It is, after all, the power to which 542276103 must be raised (modulo n) to give 478354870. The computation of discrete logarithms (of large numbers) is generally believed to be a very hard problem, in fact of the same difficulty as the factorization of large integers.

Summary

We consider four (typically large) non-negative integers x, m, a and n related by

$$x^m \equiv a \mod n.$$

If we are asked to solve for $a \in \{0, 1, 2, 3, \ldots, n-1\}$ given the other quantities (the problem of *modular exponentiation*) then in general the method of *binary exponentiation* may be used: work out the binary representation of m, repeatedly square x mod n, and multiply the required powers of x together mod n to obtain the result. In special cases, however, the problem is significantly reduced by observing several observations. We may in every case first reduce the value of x mod n. If $\gcd(x, n) = 1$ (as we check by Euclid's Algorithm) and $\phi(n)$ is known or can be determined from the prime factorization of n, then we may further reduce m mod $\phi(n)$. This observation is of little value for large composite values of n whose factorization is not known, since in this case we don't know how to determine $\phi(n)$.

If we are asked to solve for x given m, a and n, and if $\phi(n)$ is known (or computable, say, from the prime factorization of n), then we should first check whether $\gcd(m, \phi(n)) =$

1. If so then there exists a positive integer k such that $km \equiv 1 \mod \phi(n)$. (The value of k may be computed by the Extended Euclidean Algorithm.) Raising both sides to the power k yields the solution $x \equiv a^k \mod n$.

If we are asked to solve for m given x, a and n, then we are faced with the *discrete logarithm problem.* If n is small enough, we follow the naive approach of trying all possible values of $x \in \{0, 1, 2, \ldots, n-1\}$ until we find a solution (or all solutions, if required). Some more efficient algorithms are known, but no practical solution is known in general for large values of n. The difficulty of solving the discrete logarithm problem for large integers appears to be comparable to the problem of integer factorization, which is also seemingly intractable.

We actually take n to be a fixed positive integer throughout. (There is neither any challenge nor any interest in solving for n given the other quantities x, m and a.)

HW#3 Due Fri Nov 21, 2008

Instructions: Let w be your W-number. (If your number is W02912077 then you should take $w = 2912077$ since, by our definition, the leading digit 0 doesn't affect the value of w.)

In Question 1 you should work by hand, with the help of a claculator, and show your work. You may use MAPLE to check your answers. In Questions 2, 3 and 4 you may freely use MAPLE to perform any computations necessary, and check your answers whenever possible. And be careful in copying the large integers from this assignment!

1. Find the last two digits of 1234567^w. (Work by hand.)

2. Find the remainder when $w^{17762091}$ is divided by 123456789. (Use MAPLE.)

3. Find an integer m satisfying $4444^m \equiv w \mod 16187$. (Use MAPLE.)

4. Find an integer x such that $x^{314159} \equiv w \mod 863116403$. (Use MAPLE.)

5. Find an integer x such that $x^{2718281} \equiv w \mod 130024553$. (Use MAPLE.)