Fast modular exponentiation

(also called *square-and-multiply*)

Computing things like

$$x^e \,\%\, m$$

for x, e, m all $\sim 10^{100}$ or larger, in ~ 300 steps, not $\sim 10^{100}$ multiplications, is essential in public-key crypto.

Repeated squaring reduces the total number of operations:

$$x^{69} = x^{2^6+2^2+2^0} = ((((((x^2)^2)^2)^2)^2)^2)^2 \cdot (x^2)^2 \cdot x$$

This is an important short cut with or without reduction modulo m.

The other point is that it is very much worthwhile to *frequently reduce modulo* m, rather than compute a huge number and only reduce at the end.

Further, rather than spend time and/or energy worrying about *when* to reduce modulo m in the course of this calculation, it is more efficient to simply do so on every possible occasion.

The fast modular exponentiation algorithm:

To compute $x^e \,\%\, m$

initialize $(X, E, Y) = (x, e, 1)$
while $E > 0$
    if E is even
        replace X by $X^2 \,\%\, m$
        replace E by $E/2$
    elsif E is odd
        replace Y by $X \cdot Y \,\%\, m$
        replace E by $E - 1$

The final value of Y is $x^e \,\%\, m$.

For example, to compute $5^{17} \,\%\, 101$,
Initialize $(X, E, Y) = (5, 17, 1)$

$(X, E, Y) = (5, 17, 1)$, $E = 17$ is odd

replace $E = 17$ by $17 - 1 = 16$

replace $Y = 1$ by $X * Y \,\%\, 101 = 5$

$(X, E, Y) = (5, 16, 5)$, $E = 16$ is even

replace $E = 16$ by $16/2 = 8$

replace $X = 5$ by $X * X \,\%\, 101 = 25$

$(X, E, Y) = (25, 8, 5)$, $E = 8$ is even

replace $E = 8$ by $8/2 = 4$

replace $X = 25$ by $X * X \,\%\, 101 = 19$

$(X, E, Y) = (19, 4, 5)$, $E = 4$ is even

replace $E = 4$ by $4/2 = 2$

replace $X = 19$ by $X * X \,\%\, 101 = 58$

$(X, E, Y) = (58, 2, 5)$, $E = 2$ is even

replace $E = 2$ by $2/2 = 1$

replace $X = 58$ by $X * X \,\%\, 101 = 31$

$(X, E, Y) = (31, 1, 5)$, $E = 1$ is odd

replace $E = 1$ by $1 - 1 = 0$

replace $Y = 5$ by $X * Y \,\%\, 101 = 54$

Now $(X, E, Y) = (31, 0, 54)$, $E = 0$, so

$5^{17} \,\%\, 101 =$ current value $Y = 54$

Run-time estimate

For fast modular exponentiation, instead of performing roughly e multiplications (and reductions) to evaluate $x^e \,\%\, m$, there are at most $\log_2 e$ squarings (and reductions) and at most $\log_2 e$ multiplications (and reductions).

Thus, instead of e (admittedly slightly simpler) operations, we have at worst $2 \log_2 e$ operations.

For example, with $e \sim 10^{100}$, instead of about

$$10^{100} \text{ multiplications}$$

we have at worst

$$2 \log_2(10^{100}) \sim 665$$

square-or-multiply operations.

Square roots modulo $p = 3 \bmod 4$

Fermat's Little Theorem gives us some interesting formulas (below), which become *practical* due to the efficiency of Fast Modular Exponentiation.

Definition: An integer b is a **square root** of a modulo m if

$$b^2 = a \bmod m$$

If a has a square root modulo m, then say a **is a square modulo** m.

Much as *inverses* modulo m don't have much connection with inverses in the rational or real numbers, square roots modulo m do not have much direct connection with square roots that are real numbers.

For example, 3 is a square root of 2 modulo 7 because

$$3^2 = 2 \bmod 7$$

Remark: At this point, there is no assurance in advance that numbers have (or don't have) square roots modulo m.

At worst (which is pretty bad for large moduli) we could try to find a square root of a modulo m by just trying all the possibilities from 0 up to $m-1$.

For example, to *try* to find a square root of 2 modulo 17:
$0^2 = 0 \neq 2 \bmod 17$: 0 is not the square root
$1^2 = 1 \neq 2 \bmod 17$: 1 is not the square root
$2^2 = 4 \neq 2 \bmod 17$: 2 is not the square root
$3^2 = 9 \neq 2 \bmod 17$: 3 is not the square root
$4^2 = 16 \neq 2 \bmod 17$: 4 is not the square root
$5^2 = 25 \neq 2 \bmod 17$: 5 is not the square root
$6^2 = 36 = 2 \bmod 17$: 6 **is** the square root

Parallel to proving primality by *failing* to find a factor, we can prove that something has *no* square root mod m by *failing* to find a square root:

For example, to *try* to find a square root of 3 modulo 7:
$0^2 = 0 \neq 3 \bmod 7$: 0 is not the square root
$1^2 = 1 \neq 3 \bmod 7$: 1 is not the square root
$2^2 = 4 \neq 3 \bmod 7$: 2 is not the square root
$3^2 = 9 \neq 3 \bmod 7$: 3 is not the square root
$4^2 = 16 \neq 3 \bmod 7$: 4 is not the square root
$5^2 = 25 \neq 3 \bmod 7$: 5 is not the square root
$6^2 = 36 \neq 3 \bmod 7$: 6 is not the square root

We need look only in the range $0, 1, \ldots, m - 1$, since by well-definedness of operations modulo m, if b were a square root of a modulo m, then $b \,\%\, m$ is also a square root of a modulo m

As usual, brute force is very suboptimal.

Theorem: Let p be a prime with $p = 3 \bmod 4$. *If* a is a square modulo p, then

$$a^{\frac{p+1}{4}} \text{ (and/or its reduction mod } m\text{)}$$

is a square root of a modulo p. (If a is *not* a square modulo m, then obviously this expression does not yield a square root of a modulo m.)

Positive real numbers have positive and negative square roots (and negative real numbers have *no* real square roots) and we tend to think of the positive square root as being more important.

Similarly, if a is a square modulo p as above, the square root $b = a^{(p+1)/4} \bmod p$ given by that formula is called the **principal square root** of a, and $-b$ is the *other* square root of a modulo p.

Proof: The fact that $p = 3 \bmod 4$ assures that $(p+1)/4$ really is an integer. Otherwise the formula makes no sense in the first place.

Suppose that $a = b^2 \bmod p$. Then *all modulo p*

$$\left(a^{\frac{p+1}{4}}\right)^2 = \left((b^2)^{\frac{p+1}{4}}\right)^2 = b^{p+1} \bmod p$$

By Fermat's Little Theorem, $b^p = b \bmod p$, so this becomes

$$= b \cdot b = a \bmod p$$

since by hypothesis $b^2 = a \bmod p$. ///

Ok, once told the formula we could verify that it is correct. *But how would a person get the idea of such a formula?*

Remark: For cryptographic purposes, it is important to realize that we have or will have formulas for square roots (and other roots) **modulo primes**, but *not* modulo composite numbers **unless we can factor them**.

In fact, if $n = p \cdot q$ with secret primes $p = q$, any **oracle** that can compute square roots modulo n gives a *probabilistic* algorithm to *factor* n.

We'll look at this later in the context of **Sun-Ze's theorem** (sometimes called the Chinese Remainder Theorem).

Remark: In computations, we may not have any idea in advance as to whether a is or is not a square modulo p, so we do not know whether the formula is producing the principal square root or just garbage.

One must square the result and see whether or not the result is the thing whose square root we want!

That is, in a computational setting, usually we have to *check* whether or not the output of the formula is or is not a square root of the given thing.

If it *does* square to the original thing, then certainly the original thing is a square.

If it does *not* square to the original, then the original was *not* a square ... since if it *were* a square the formula would have produced its (principal) square root!

For example, we *try* to find a square root of 2 modulo 31. We do not know in advance whether or not 2 is a square modulo 31.

First, note that (by trial division) 31 is indeed prime, and $31 = 3 \bmod 4$. Thus, the formula applies: *if* 2 is a square modulo 31, then its (principal) square root is

$$2^{\frac{31+1}{4}} \bmod 31$$

The numbers are so small here that there's not mandate to use Fast Modular Exponention:

$$2^{\frac{31+1}{4}} = 2^8 = 256 = 8 \bmod 31$$

Thus, *if* 2 is a square mod 31 *then* its (principal) square root is 8. How do we test? Square 8 and see if we get 2 modulo 31: all modulo 31

$$8^2 = 64 = 64 \,\%\, 31 = 64 - 2 \cdot 31 = 2 \bmod 31$$

So, yes, 8 is the (principal) square root of 2 modulo 31. ///

For example, we *try* to find a square root of 3 modulo 31. We do not know in advance whether or not 3 is a square modulo 31.

First, (by trial division) 31 is indeed prime, and $31 = 3 \bmod 4$. Thus, the formula applies: *if* 3 is a square modulo 31, then its (principal) square root is

$$3^{\frac{31+1}{4}} \bmod 31$$

$$3^{\frac{31+1}{4}} = 3^8 = 6561 = 20 \bmod 31$$

If 3 is a square mod 31 *then* its (principal) square root is 20. We must test. Square 20 and see if we get 3 modulo 31: all modulo 31

$$20^2 = 400 = 400 \,\%\, 31 = 28 \neq 3 \bmod 31$$

So, **no**, 20 is **not** a square root of 3 modulo 31, and in fact 3 is not a square mod 31. (If it were, we would have found its square root by the above process!) ///

Remark: Actually, if a is not a square mod p, then the formula above gives the principal square root of $-a$ modulo p instead.

This is not easy to verify, but examples would suggest this fact.

Remark: In larger examples it would become wise or even necessary to use the Fast Modular Exponentiation algorithm to do the relevant computations.