Exponentiation and Primality

1 Exponentiation

We are asked to compute $x^y \bmod z$, where x, y, z are given integers. The obvious iterative algorithm (multiply by x y times) is too slow, it takes time exponential in the number of bits of the given integers. The right way to do it is this:

Repeatedly square x $\lfloor \log y \rfloor$ times to compute $x^2 \bmod z, x^4 \bmod z, x^8 \bmod z, \ldots, x^{2^{\lfloor \log y \rfloor}} \bmod z$.

Compute $x^y \bmod z$ from these numbers by multiplying together, always modulo z, the powers that correspond to ones in the binary representation of y.

For example, to compute $23^{20} \bmod 10$ we would compute $23^2 \bmod 10, 23^4 \bmod 10, 23^8 \bmod 10, 23^16 \bmod 10$, and then we would multiply $23^4 \cdot 23^16 \bmod 10$ to obtain the result.

This takes at most $2 \log y$ arithmetic operations of integers modulo z, and is therefore efficient.

Another way to view the same algorithm: Remember the multiplication algorithm in the first lecture (recursive version):

```
mult(x,y)
if y = 1 then return(x)
else return(mult(x+x,⌊ y/2 ⌋)+odd(y)·x)
```

The same "control structure" exponentiates, just change + to *:

```
exp(x,y,z)
if y = 1 then return(x mod z)
else return(exp(x*x,⌊ y/2 ⌋)*x ↑ odd(y) mod z)
```

2 Primality

We shall study the complexity of two very fundamental, and intimately related, computational problems:

PRIMALITY Given an integer n, is it a prime?

FACTORING Given an integer n, what are its prime factors?

Obviously, PRIMALITY cannot be harder than FACTORING, since, if we knew how to factor, we would definitely know how to test for primality. *What is surprising and fundamental —and the basis of modern cryptography— is that* PRIMALITY *is easy while* FACTORING *is hard!*

PRIMALITY can be trivially solved in $O(n)$ time —in fact, $O(\sqrt{n})$ is easy, we need only test factors up to $\sqrt{n}$. But these are exponential algorithms —in the number of bits of n. In fact, pursuing this line will get us nowhere: Since FACTORING is hard, our only hope for finding a fast PRIMALITY algorithm is to look for an algorithm that decides whether n is prime without discovering a factor of n in case the answer is "no."

We describe such an algorithm next. We start with four facts from number theory on which this algorithm —and also the RSA cryptosystem— is based. We shall only prove facts 1 and 4.

Fact 1. *(Fermat's Little Theorem.)* If p is prime, then for all $a \neq 0 \bmod p$ $a^{p-1} = 1 \bmod p$.

Fact 2. If p is *not* a prime, *and if p is not a Carmichael number,* then for *most* $a \neq 0 \bmod p$ $a^{p-1} \neq 1 \bmod p$.

Fact 3. The density of primes around n is about $\frac{1}{\ln n}$. In other words, among all numbers with D decimal digits, the chances of being prime is one in (a little more than) $2D$. "One in about twenty social security numbers are prime."

Fact 4. If p and q are primes, then for all $a \neq 0 \bmod p, q$ $a^{(p-1)\cdot(q-1)} = 1 \bmod p \cdot q$.

Let us prove Fact 1, by example. Take $p = 7$, and consider the nonzero numbers modulo 7, $\{1, 2, 3, 4, 5, 6\}$. Pick an a in this set, and multiply all these numbers by a, modulo 7, we get (say, $a = 5$) $\{5, 3, 1, 6, 4, 2\}$ —*the same numbers!* We should expect this: If $a \cdot i = a \cdot j \bmod p$, then, by premultiplying by a^{-1} —in this case 3— we get $i = j$.

So, by multiplying this set by a we get the same set.

Now multiply together the numbers in the two sets (which we know are equal) $\{i : 0 < i < p\}$ and $\{a \cdot i \bmod p : 0 < i < p\}$. We must get the same product. So,

$$1 \cdot 2 \cdot 3 \cdot \cdots \cdot (p-1) = a \cdot 1 \cdot a \cdot 2 \cdot a \cdot 3 \cdot \cdots \cdot a \cdot (p-1) \bmod p,$$

and if we multiply both sides by the inverses of all numbers $1^{-1}, 2^{-1}, \ldots, (p-1)^{-1}$ we get that

$$a^{p-1} = 1 \bmod p,$$

which is Fact 1.

Fact 2 is a weak converse of Fact 1: It says that *if* p is composite, and *if* p does not happen to be among a set of extremely rare exceptions called Carmichael numbers,[1] then the primality test suggested by Fermat's Little Theorem will fail with probability at leat 50%.

Fermat's Little Theorem suggests the following *randomized algorithm* for primality:

```
primality(p)
repeat
{ pick an integer $a$ between 0 and $p$ at random
if $a^{p-1} \neq 1 \bmod p$ then return($p$ ``is not a prime'') }
until satisfied
return($p$ ``is a almost certainly a prime'')
```

By repeating the test 100 times, we establish primality to a degree of confidence (.999... up to thirty nines) that surpasses all other aspects of life and computation. The test takes $O(\log^3 p)$ steps —fewer if fast multiplication is used.

Incidentally Fact 4's proof is exactly the same as Fact 1's, except that we consider the set of all numbers between 0 and $p \cdot q$ *that are relatively prime to $p \cdot q$. Notice that there are $(p-1)\cdot(q-1)$ such numbers —check it out, one in every p of the numbers between 0 and $p \cdot q$ are divisible by p, and one every q by q.*

[1]A number c is a Carmichael number if it is not a prime, and still $d|c$ implies $d-1|c-1$. The smallest Carmichael number is 561.