The binary powering algorithm

In many algorithms in number theory, we need to find the least nonnegative residue of a^r modulo n where r and n are quite large numbers. We don't do this by first calculating a^r and then taking the remainder by dividing by n. This number may be huge, and quite likely too big for one's computer to deal with. So to simplify things at each intermediate stage of computation we take the remainder modulo n. This means that we need never deal with numbers larger than n^2.

As an example, let's compute 19^5 modulo 29. We could compute in turn

$$19^2 = 361 \equiv 13 \pmod{29},$$

$$19^3 \equiv 19 \times 13 = 247 \equiv 15 \pmod{29},$$
$$19^4 \equiv 19 \times 15 = 285 \equiv 24 \pmod{29},$$
$$19^5 \equiv 19 \times 24 = 456 \equiv 21 \pmod{29}.$$

As you see no number greater than 29^2 needs to be considered.

Alas this method is only practical when r is small as it takes $r-1$ steps. When r is large, say 100 digits long, it is totally impractical. To see a better method let us first consider the case where r is a power of 2, say $r = 2^k$. Instead of computing a^2, a^3, a^4, a^5 modulo n etc., we can compute instead a^2, a^4, $a^8, \ldots, a^{2^k}$ modulo n by squaring each previous term. This cuts the calculation to $k-1$ steps. As an example: what is 2^{32} modulo 101?

$$2^2 = 4,\ 2^4 = 4^2 = 16,\ 2^8 = 16^2 = 256 \equiv 54 \pmod{101},$$

$$2^{16} \equiv 54^2 = 2916 \equiv 88 \pmod{101},$$
$$2^{32} \equiv 88^2 = 7744 \equiv 68 \pmod{101}.$$

But most numbers r are not powers of 2. This makes things just slightly trickier. How could we compute 2^{43} modulo 97? Note that

$$\begin{aligned} 2^{43} &= 2 \times 2^{42} = 2 \times (2^2)^{21} = 2 \times 4^{21} = (2 \times 4) \times (4^2)^{10} = 8 \times (16)^{10} \\ &= 8 \times (16^2)^5 = 8 \times (256)^5 \equiv 8 \times 62^5 \pmod{97} \end{aligned}$$

as $256 \equiv 62 \pmod{97}$. Continuing:

$$\begin{aligned} 2^{43} &\equiv (8 \times 62) \times (62^2)^2 = 496 \times 3844^2 \\ &\equiv 11 \times 61^2 = 11 \times 3721 \equiv 11 \times 35 = 385 \equiv 94 \pmod{97}. \end{aligned}$$

We can make this procedure into a systematic algorithm, here presented in `MAPLE`.

```
> binpow := proc (a,r,n)
> ans := 1:
> b := a:
> s := r:
> while (s > 0) do
> if ((s mod 2) = 1) then
> ans := (ans * b) mod n:
> s := s - 1:
> fi:
> b := (b * b) mod n:
> s := s/2:
> od:
> ans;
> end;
```

To analyse this algorithm, note that it contains three auxiliary variables ans, b and s which are intialized to 1, a and r respectively. There is a loop. The key to the algorithm is that the variables ans, b and s always satisfy $a^r \equiv ans \times b^s \pmod{n}$. The initialization ensures this holds at the start. If s is odd and $a^r \equiv ans \times b^s \pmod{n}$ then $a^r \equiv (ans \times b) \times b^{s-1} \pmod{n}$. We may replace s by $s-1$ (which is even) and ans by $ans \times b$ or its least nonnegative residue modulo n. This is what the `if` clause does. If s is even and $a^r \equiv ans \times b^s \pmod{n}$ then $a^r \equiv ans \times (b^2)^{s/2} \pmod{n}$. We may replace s by $s/2$ (which is even) and b by b^2 or its least nonnegative residue modulo n. This is what the remainder of the loop does.

Each iteration of the loop causes s to be replaced by the integer part of $s/2$. After about $\log_2 r$ steps s reaches zero and the loop is terminated. At this stage $a^r \equiv ans \times b^0 = ans \pmod{n}$ so that the sought answer is ans, which is the value output.

This algorithm is fast. It takes about $\log_2 r$ iterations. Each iteration requires one or two multiplications resulting in numbers $< n^2$. The storage space required is small. Only three numbers are stored, of size $\leq \max(r, n)$ and intermediate stages require only numbers less than n^2.

It is called the binary powering algorithm, as the sequence of values `s mod 2` computed in this `MAPLE` procedure is the sequence of binary digits of r read from right to left.

This algorithm is built in to `MAPLE`. The command `> a &^ r mod n;` gives the same result as `> binpower(a,r,n);` as defined above. The ampersand is important — without it, `MAPLE` will compute a^r before reducing modulo n — not recommended!

RJC 13/11/2003