- Continuing discussion of CRC's, especially looking at two-bit errors

- The definition of **primitive** binary polynomials

- Brute force checking for primitivity

- A theorem giving a better test for primitivity

- Fast modular exponentiation algorithm for integers

- Fast modular exponentiation algorithm for polynomials

Example: Find a 2-bit error that could occur in the message 10100011001100 that would be undetected by the CRC with generating polynomial coefficients (from highest to lowest) 110011.

The specific message 10100011001100 doesn't matter, except that it is long enough to have two bit-errors sufficiently separated so that the CRC fails to detect them. As noted in class and in the book, since the constant coefficient of the generator is 1, a failure to detect two bit-errors depends only upon their distance apart in the message, not their locations. For CRC with generator 110011 to fail to detect a pair of bit errors a distance n apart, it is necessary and sufficient that the generating polynomial $x^5 + x^4 + x + 1$ (made from 110011 by taking coefs in descending order) divide $x^n - 1$ with remainder 0. Test whether 110011 divides $x^t - 1$ with $t = 5, 5 + 1, 5 + 2$. ...

The remainder dividing $x^5 + 1$ by $x^5 + x^4 + x + 1$ is $x^4 + x \neq 0$.

So try $x^6 + 1$: the remainder dividing $x^6 + 1$ by $x^5 + x^4 + x + 1$ is $x^4 + x^2 \neq 0$.

Try $x^7 + 1$: the remainder dividing $x^7 + 1$ by $x^5 + x^4 + x + 1$ is $x^4 + x^3 \neq 0$, so try $x^8 + 1$.

At last, the remainder dividing $x^8 + 1$ by $x^5 + x^4 + x + 1$ is 0.

Thus, any two single-bit errors a distance of 8 apart will *not* be detected by this CRC.

Remark: This *brute force* approach is good for small examples, and is ok if no other approach is available, but does not scale upward well.

For bigger examples use **fast modular exponentiation** (below).

Primitive polynomials

Some CRCs can catch any two bit-errors in very long strings because their generating polynomials are *primitive*, in the sense below.

Keep in mind: with **two bit errors**, at m^{th} and n^{th} positions (with $m < n$), the error is

$$e(x) = x^m + x^n$$

undetected if and only if $g(x)$ divides

$$e(x) = x^m + x^n x^m (1 + x^{n-m})$$

If $g(x)$ has **constant term** 1, then g has no factor of x and then

a two-bit error distance N apart is undetected if and only if $g(x)$ divides $x^N + 1$.

It turns out (!) that the best performance possible is with $N = 2^d - 1$ where $d = \deg g(x)$.

Definition: A polynomial of degree d with coefficients in $\mathbf{F}_2$ is **primitive** if the smallest integer N such that the polynomial divides $x^N - 1$ is $N = 2^d - 1$.

Remark: It is unclear whether there are many such things, how to find them, how to verify the property, etc.

Remark: When used in CRCs, a primitive *cubic* would catch all errors within $2^3 - 1 = 7$ apart. A primitive *quartic* would catch all errors within $2^4 - 1 = 15$ apart. A primitive *quintic* would catch all errors within $2^5 - 1 = 31$ apart. A primitive *sextic* would catch all errors within $2^6 - 1 = 63$ apart. Etc.

Remark: The 16-bit CRC above detects all 3-bit errors in data of 32767 bits or less because it is the product of $x+1$ with a primitive degree 15 polynomial. The primitive degree 15 polynomial detects two-bit errors within distance of 32767 while the $x + 1$ detects all errors consisting of an odd number of bit errors.

Terminology for remainder-upon-divisions:

Definition: the **reduction modulo** $M(x)$ of a polynomial $P(x)$ with respect to a polynomial $M(x)$ is defined to be

$$P(x) \,\%\, M(x)$$
$$= \text{remainder dividing } P(x) \text{ by } M(x)$$

The polynomial $M(x)$ is the **modulus**.

Theorem: A binary polynomial $g(x)$ of degree d is primitive if and only if

- $x^{2^d} \,\%\, g(x) = x$

and, for every prime integer q dividing $2^d - 1$,

- $x^{(2^d-1)/q} \,\%\, g(x) \neq 1$

Remark: Thus, instead of looking at $(x^e - 1) \,\%\, g(x)$ for the possible exponents

$$1 \leq e \leq 2^d - 1$$

we need only examine a much smaller collection.

Example: The binary quartic polynomial $g(x) = x^4 + x + 1$ is primitive if and only if

$$x^{2^4} \,\%\, x^4 + x + 1 = x$$

and for every prime q dividing 15 (thus, by trial division, 3 and 5)

$$x^{(2^4-1)/q} \,\%\, x^4 + x + 1 \neq 1$$

That is, $x^4 + x + 1$ is primitive if and only if

$$x^{2^4} \,\%\, x^4 + x + 1 = x$$

$$x^{(2^4-1)/3} \,\%\, x^4 + x + 1 \neq 1$$

$$x^{(2^4-1)/5} \,\%\, x^4 + x + 1 \neq 1$$

We can be a little clever in doing these computations, after some preparation.

Fast modular exponentiation

Returning for the moment to *ordinary integers* $\mathbf{Z}$ instead of polynomials,

Definition: the **reduction modulo** m of an integer x with respect to an integer m is defined to be

$$\text{reduction of } x \text{ modulo } m =$$

$$x \,\%\, m = \text{remainder upon dividing } x \text{ by } m$$

The integer m is the **modulus**.

For example,

$$\begin{aligned} 5 \,\%\, 3 &= 2 \\ 2 \,\%\, 3 &= 2 \\ 12 \,\%\, 4 &= 0 \\ 13 \,\%\, 4 &= 1 \\ 17 \,\%\, 5 &= 2 \\ 23 \,\%\, 10 &= 3 \\ 119 \,\%\, 10 &= 9 \\ 1000 \,\%\, 11 &= 10 \end{aligned}$$

A type of computation that often arises in practice is something like

$$(5^{65}) \,\%\, 11 = ?$$

Remark: Unless your calculating device has *infinite precision* integer arithmetic, you must **not** attempt to evaluate 5^{65} first and then divide by 11 to find the remainder. Round-off error will give you complete nonsense. Not *partial* nonsense, but a *completely worthless result*.

Instead, there is a hand-executable algorithm **fast modular exponentiation** which scales upward very nicely.

The first observation is that

$$5^{65} = 5^{2^0+2^6} = 5 \cdot ((((((5^2)^2)^2)^2)^2)^2)^2$$

That is, instead of 64 multiplications, there will be 6 squarings and one multiplication.

It is less obvious, but is provably true, and is crucial to this, that in computing $5^{65} \,\%\, 11$, *we can reduce modulo 11 after each step, rather than wait till the end.* That means we do not need to remember any integer larger than 2 digits.

The fast modular exponentiation algorithm to compute $x^e \,\%\, m$ (with positive integer e) says:

Initialize $(X, E, Y) = (x, e, 1)$
While $E > 0$:
  If E odd:
    Replace E by $E - 1$
    Replace Y by $(X \cdot Y) \,\%\, m$
  Else if E even:
    Replace E by $E/$
    Replace X by $(X \cdot X) \,\%\, m$
When $E = 0$, $Y = (x^e) \,\%\, m$.

This algorithm takes at most $2 \log_2 e$ *steps, and uses little memory.*

To compute $5^{65} \,\%\, 11$
Initialize $(X, E, Y) = (5, 65, 1)$
$E = 65$ is odd, so replace $E = 65$ by $E - 1 = 64$
and Y by $(X \cdot Y) \,\%\, m = (5 \cdot 1) \,\%\, 11 = 5$
$E = 64$ is even, so replace $E = 64$ by $E/2 = 32$
and $X = 5$ by $(X^2) \,\%\, m = 25 \,\%\, 11 = 3$
$E = 32$ is even, so replace $E = 32$ by $E/2 = 16$
and $X = 3$ by $(X^2) \,\%\, m = 9 \,\%\, 11 = 9$
$E = 16$ is even, so replace $E = 16$ by $E/2 = 8$
and $X = 9$ by $(X^2) \,\%\, m = 81 \,\%\, 11 = 4$
$E = 8$ is even, so replace $E = 8$ by $E/2 = 4$
and $X = 4$ by $(X^2) \,\%\, m = 16 \,\%\, 11 = 5$
$E = 4$ is even, so replace $E = 4$ by $E/2 = 2$
and $X = 5$ by $(X^2) \,\%\, m = 25 \,\%\, 11 = 3$
$E = 2$ is even, so replace $E = 2$ by $E/2 = 1$
and $X = 3$ by $(X^2) \,\%\, m = 9 \,\%\, 11 = 9$
$E = 1$ is odd, so replace $E = 1$ by $E - 1 = 0$
and Y by $(X \cdot Y) \,\%\, m = (9 \cdot 5) \,\%\, 11 = 45 \,\%\, 11 = 1$
Now $E = 0$ so $5^{65} \,\%\, 11 = 1$, the value of Y

The same algorithm can be applied in many other situations, for example to polynomials. The *reduction* idea is just the *remainder after division* we already discussed in the context of CRCs.

For example, all with coefficients in $\mathbf{F}_2$

$$\begin{array}{lcll} x^2+x+1 & \% & x+1 & = & 1 \\ x^2+x+1 & \% & x^2+1 & = & x \\ x^3+x+1 & \% & x+1 & = & 1 \\ x^3+x+1 & \% & x^2+1 & = & 1 \end{array}$$

Remark: These computations are less familiar than integer computations.

Example: To check whether or not $x^4 + x + 1$ with coefficients in $\mathbf{F}_2$) is primitive, an efficient approach is to invoke the theorem and ehcek that

$$x^{2^4} \,\%\, x^4 + x + 1 = x$$

and that for any prime q dividing $2^4 - 1 = 3 \cdot 5$

$$x^{(2^4-1)/q} \,\%\, x^4 + +x + 1 \neq 1$$

We practice doing exponentiations by fast modular exponentiation. The primes dividing 15 are 3 and 5 (trial division), so we must verify that

$$x^{15/3} \,\%\, x^4 + x + 1 \neq 1$$

and

$$x^{15/5} \,\%\, x^4 + x + 1 \neq 1$$

Initialize $(X, E, Y) = (x, 16, 1)$.

E=16 is even, so square X and reduce mod $m = x^4 + x + 1$ and divide E by 2, giving $(X, E, Y) = (x^2, 8, 1)$

E=8 is even, so square X and reduce mod $m = x^4 + x + 1$ and divide E by 2, giving $(X, E, Y) = (x + 1, 4, 1)$

E=4 is even, so square X and reduce mod $m = x^4 + x + 1$ and divide E by 2, giving $(X, E, Y) = (x^2 + 1, 2, 1)$

E=2 is even, so square X and reduce mod $m = x^4 + x + 1$ and divide E by 2, giving $(X, E, Y) = (x, 1, 1)$

E=1 is odd, so multiply Y by X and reduce mod $m = x^4 + x + 1$, and subtract 1 from E, giving $(X, E, Y) = (x, 0, x)$

Now E is 0, so $Y = x$ is the desired $x^{16} \,\%\, x^4 + x + 1$.

Now the other two conditions:

To compute $x^{15/3} \,\%\, x^4 + x + 1$: initialize $(X, E, Y) = (x, 5, 1)$.

E=5 is odd, so multiply Y by X and reduce mod $m = x^4 + x + 1$, subtract 1 from E, giving

$$(X, E, Y) = (x, 4, x)$$

E=4 is even, so square X and reduce mod $m = x^4 + x + 1$ and divide E by 2, giving

$$(X, E, Y) = (x^2, 2, x)$$

E=2 is even, so square X and reduce mod $m = x^4 + x + 1$ and divide E by 2, giving

$$(X, E, Y) = (x + 1, 1, x)$$

E=1 is odd, so multiply Y by X and reduce mod $m = x^4 + x + 1$, and subtract 1 from E, giving

$$(X, E, Y) = (x + 1, 0, x^2 + x)$$

Now E is 0, so $Y = x^2 + x \neq 1$ is the desired $x^5 \,\%\, x^4 + x + 1$.

More simply, $x^{15/5} \,\%\, x^4 + x + 1 = x^3 \neq 1$, so, $x^4 + x + 1$ is *primitive*.
