SRQR: Binary Exponentiation, Representing Numbers (April 7, 2021)

> In example 4.5, how did it go from $5^2 \equiv 25 \pmod{131}$ to $5^4 \equiv 101 \pmod{131}$?

> In example 4.5, I realize the $5^8 - 114$ is a multiple of 131, but how was the remainder of 114 found for this problem?

If $5^2 \equiv 25 \pmod{131}$, that means that

$$5^4 = (5^2)^2 \equiv 25^2 = 625 \equiv 101 \bmod 131.$$

I personally can't "instantly" see the remainder when 625 is divided by 131. The numbers are small enough that the long division wouldn't take that long, but in any case, I would probably have used a computer to find that remainder. Then, for the next step,

$$5^8 = (5^4)^2 \equiv 101^2 = 10201 \equiv 114.$$

Again, I can't "instantly" see the square of 101, nor the remainder when 10201 is divided by 114. Neither that squaring nor the division would take very long to do by hand, but I would still have used a computer for these. The important thing to note, though, is that neither you nor a computer ever actually has to calculate 5^8!

> In example 4.5, where it says, $105 \cdot 74 \cdot 114 \cdot 101 \cdot 25 \equiv 60 \pmod{131}$, I'm wondering where the 60 is coming from.

Again, I would probably use a calculator, but if you do the multiplications and reductions iteratively, you never have to deal with very big numbers. For example,

$$105 \cdot 74 = 7770 \equiv 41 \pmod{131}.$$

Then

$$(105 \cdot 74) \cdot 114 \equiv 41 \cdot 114 = 4674 \equiv 89 \pmod{131}.$$

Keep going like this until you've dealt with all 5 terms.

> What is significant about the use of binary in the binary exponential algorithm? In example 4.5, it looks like the process should work for any set of numbers that add to 110 — does it matter that we use binary specifically, or is that just one way to obtain such a set?

Theoretically speaking, there is no significance to using binary. The point is just to compute a high power of a number using a small number of steps, and binary is not the only way of doing that. You can do it using other bases equally well, or you can do the hybrid approach that Burton takes at the end of example 4.5.

Practically speaking, there is a significance to binary: if you give a computer a, k, n and tell it to compute $a^k \pmod{n}$, the computer *already knows* the binary expansion for k! That means you can jump right into the exponentiation. If you wanted to use some other base, the computer would have to first calculate the representation of k in that base, and then do the exponentiation. So using binary saves the computer a little bit of work.

How does one get (on the top of page 70) $-b < a_k - c_k < b$ from $0 \leqslant a_k < b$ and $0 \leqslant c_k < b$?

Let's think about when $a_k - c_k$ is as *big* as possible. That happens when a_k is as big as possible and c_k is as small as possible. The biggest possible value for a_k is $b - 1$, and the smallest possible value for c_k is 0, so the maximum possible value for $a_k - c_k$ is $b - 1$. In other words, $a_k - c_k < b$.

You can argue similarly to see that $-b < a_k - c_k$ (think about when $a_k - c_k$ is as *small* as possible).

I'm kind of confused about what theorem 4.4 means.

Theorem 4.4 is saying that if you have a polynomial $P(x)$ with integer coefficients, and you have two numbers a and b which are congruent mod some integer n, then the evaluations $P(a)$ and $P(b)$ are also congruent mod n.

For example, say $P(x) = x^2 + 1$ and notice that $1 \equiv 4 \bmod 3$. We can evaluate $P(x)$ at $x = 1$ to get

$$P(1) = 1^2 + 1 = 2,$$

and we can evaluate $P(x)$ at $x = 2$ to get

$$P(4) = 4^2 + 1 = 17,$$

and notice that 17 is in fact congruent to 2 mod 3 (the difference is 15, which is divisible by 3). Theorem 4.4 says this will always happen, no matter what polynomial with integer coefficients $P(x)$ you start with and no matter what congruent integers a and b you plug in.

Could you use the method in theorem 4.5 to determine divisibility of any base n system for $n - 1$?

Yep!

Would you mind talking about how to prove CC 2(d)?

The nth triangular number t_n can be expressed as $t_n = n(n+1)/2$, as we know from a previous comprehension check. To find the units digit, just reduce mod 10 and see what happens! The easiest thing to do is to consider 10 cases separately:

- Case 1: Suppose $n \equiv 0 \pmod{10}$. Then $n = 10q$ for some q, and $n+1 = 10q+1$. Thus
$$t_n = \frac{10q(10q+1)}{2} = 5q(10q+1) \equiv 5q \pmod{10}$$
(where the last step is because $10q \equiv 0 \bmod 10$ so we can just "cancel" that term out!) This means that t_n is congruent to either 0 or 5, depending on whether q happens to be even or odd.

- Case 2: Suppose $n \equiv 1 \pmod{10}$. Then $n = 10q+1$ for some n, and $n+1 = 10q+2$. Thus
$$t_n = \frac{(10q+1)(10q+2)}{2} = (10q+1)(5q+1) \equiv 5q+1 \pmod{10}.$$
Then $5q$ is congruent to either 0 or 5, as we noted in the previous case, so $5q+1$ is congruent to either 1 or 6 mod 10.

- Case 3: Suppose $n \equiv 2 \pmod{10}$. Then $n = 10q+2$ for some n, and $n+1 = 10q+3$. Thus
$$t_n = \frac{(10q+2)(10q+3)}{2} = (5q+1)(10q+3) \equiv (5q+1)\cdot 3 \equiv 5q+3 \pmod{10}.$$
Then $5q$ is congruent to either 0 or 5, so $5q+3$ is congruent to either 3 or 8 mod 10.

I'll let you think about the remaining 7 cases! They're all pretty similar. ☺