Basic Number Theory

June 22, 2017

Contents

1 What is mod?

"Find $2^{12312312} + 42$ and print it in base 10".
"Huh, this is pretty trivial, I can write a simple loop in python to run from 1 to 12312312, and multiply by 2 in each step, and add 42 in the end, and done!".

You happily code this up, and then hit run, and then wait.
And wait some more.
And keep waiting.
.
.
.
It turns out that $2^{12312312} + 42$ is a rather large number; its decimal representation has over 3×10^6 digits.
Now, of course, this example is rather contrived, but often enough, in the course of a problem you are solving, you will have to deal with numbers which are very very large.
So instead of computing them exactly (ie. compute every single digit of their decimal representation), we decided to compute the remainder when the number itself is divided by some (pre-decided number).

We call the remainder when a is divided by b, $a \bmod b$. So you have for example, $4 \bmod 3 = 1$, because when you divide 4 by 3, you get the remainder 1. Another

example, you have $2 \bmod 6 = 2$, because when you divide 2 by 6, you get 2 as the remainder.
You're already familiar with how to make your favourite programming language do this; most languages use the `%` symbol to denote mod.

1.1 Addition, subtraction and multiplication

Hmm we talked about computing numbers mod some other number, but how do we do that? One obvious way is to compute the exact integer we want, and then take the remainder in the end.
What if we are feeling lazy, and don't want to work with big numbers at all? Say, for example, we want to find $(300 + 123) \bmod 5$. $300 + 123 = 423$, which leaves a remainder 3 when we divide by 5, so the answer is 3.
Interestingly, if I only add $300 \bmod 5$, and $123 \bmod 5$, I get $0 + 3$, which is also 3.
What if I do another one, say $(441 + 544) \bmod 5$? The hard way, we have $441 + 544 = 985$, which leaves remainder 0.
The easy way, we have $(441 \bmod 5) + (544 \bmod 5)$, which is $1 + 4$, which is 5. Hmm, this is not looking too good; what if we also take remainder after doing this new form of addition? We get $5 \bmod 5$, which is 0.
Try out a few more examples to convince yourself this works, like $(12312 + 12324) \bmod 5$.
Let's try to prove our idea.

Let's say that we have two numbers, x_1 and x_2, and we want to find $(x_1 + x_2) \bmod b$.
Let's say that when we divide x_1 by b, we get the quotient q_1, and the remainder r_1.
And similarly, when we divide x_2 by b, we get the quotient q_2, and the remainder r_2.
What does this mean?

$$\begin{aligned} x_1 &= b \times q_1 + r_1 \\ x_2 &= b \times q_2 + r_2 \end{aligned}$$

(this follows directly from the long division process).
How can we use this? Note that

$$\begin{aligned} x_1 + x_2 &= b \times q_1 + r_1 + b \times q_2 + r_2 \\ &= b \times (q_1 + q_2) + r_1 + r_2 \\ (x_1 + x_2) \bmod b &= (b \times (q_1 + q_2) + r_1 + r_2) \bmod b \\ &= (r_1 + r_2) \bmod b \\ &\text{(because } b \times (q_1 + q_2) \text{ leaves a remainder of 0 when divided by } b\text{)} \end{aligned}$$

Well this is convenient, we can take remainders before adding, and then take remainders again after adding, and we never have to deal with super large numbers, yay!

It turns out this extends to multiplication and subtraction too. In short

$$(x_1 \times x_2) \bmod b = ((x_1 \bmod b) \times (x_2 \bmod b)) \bmod b$$

(proof left as an exercise).
What about subtraction? A similar expression comes to mind.

$$(x_1 - x_2) \bmod b = ((x_1 \bmod b) - (x_2 \bmod b)) \bmod b$$

Which is also true (proof left as an exercise). An interesting case is when $x_1 > x_2$, but $x_1 \bmod b < x_2 \bmod b$;
for example, $x_1 = 6$, $x_2 = 4$, and $b = 5$, gives one such case. We get $(6 - 4) \bmod 5 = 2$, but $((6 \bmod 5) - (4 \bmod 5)) = -3$. In this situation, we add b to the result of the subtraction (5 in this example) before taking the final remainder operation, so we get $(-3 + 5) \bmod 5 = 2$, which is as desired.

Exercises

- Try writing a program to compute the 4 right-most digits of $2^{10105} + 2^{123456}$. Your output should be 3968.

1.2 Division

It turns out that division is a little bit harder to deal with than the other three basic operations; hence we will not cover it in this write-up.

2 Prime numbers

Definition 1 (Prime number)**.** *A prime number is a positive integer that has no divisors other than* 1 *and itself.*

So, for example, 2 is a prime number, and in fact the only even prime.
Now we can observe some interesting things.

Theorem 1 (Fundamental Theorem of Arithmetic)**.** *Every natural number can be uniquely (upto order) factorized into prime powers.*

So, for example, 3072 can be written as $2^{10} \times 3$, or any ordering of the same, but there is no other factorization of 3072. While this result may seem obvious, the proof is actually non-trivial, and beyond the scope of this write-up.
We will, however, use this result while proving another.

Theorem 2. *There are infinitely many prime numbers.*

Proof. Suppose that there are only k primes, where k is some finite natural number, labelled $p_1, p_2, \ldots, p_k$.
Then consider $N = p_1 \times p_2 \times \cdots \times p_k + 1$. By 1, N has a prime factorization, so at least one of $p_1, p_2, \ldots, p_k$ must divide it. But clearly, each of them leaves a remainder of 1 when it divides N. This contradicts 1, so our supposition that there are only k primes must be wrong.
Since this works for any finite k, there must be infinitely many primes. $\square$

Of course, at this point, some of you may be wondering "Well this is all fine, but what's the point of studying prime numbers?"
Alas, the applications are beyond the scope of this write-up, and we will have to satisfy ourselves with only a taste.

Exercises

- Try writing a program that accepts a single natural number N, and prints "YES" if it is prime, and no otherwise. Can you get your program to run in $O\left(\sqrt{N}\right)$ steps?
- Try writing a program that accepts a single natural number N, and prints $(N-1)! \bmod N$ (that is, the factorial of $N-1$, modulo N).

  Given 2, it should print 1, and given 37, it should print 36. Try running the program (once you are certain it is correct), with different prime numbers as input.

  Do you observe anything interesting?

3 Euclid's GCD Algorithm

Finally, after toiling through quite a bit of new theory, we can study a new algorithm. Read this write up, and then try to implement the algorithm it describes in your favourite programming language (the correct answer is C++ $\ddot\smile$)