Discrete Log Problem

A normal exponential statement is something like $2^5 = 32$.

The inverse function in real numbers for an exponential function is the log function. Thus, the corresponding log statement to the exponential statement above is:

$$\log_2 32 = 5$$

Essentially, $\log_b a$ asks the question, "To what exponent do I have to raise b to, in order to obtain a?"

In real numbers, the log function with bases above 1 is monotonically increasing. Namely, we know that if $b > 1$ and $c < d$, then, $\log_b c < \log_b d$. Thus, knowing information about one log can help you find another one.

Imagine trying to make a similar function, but under mod, where our mod value is a prime. Let's list a table of the power 2 raised to each power from 0 to 10 mod 11:

Exp	0	1	2	3	4	5	6	7	8	9	10	11
Value	1	2	4	8	5	10	9	7	3	6	1	2

Compare this with the same table, but without the mod:

Exp	0	1	2	3	4	5	6	7	8	9	10	11
Value	1	2	4	8	16	32	64	128	256	512	1024	2048

If we want to evaluate $\log_2 256$, and we observe on the table that $\log_2 32 = 5$, then we know that our own answer has to be greater than 5. Namely, this table has a pattern (the values are sorted) that we can exploit to avoid looking at the whole table to find a particular entry on the second row.

But, if we want to find which exponent makes the statement $2^x \equiv 3 \pmod{11}$ true and we observe that $2^5 \equiv 10 \pmod{11}$, this doesn't help us narrow down which column to find the value 3, because the values on the second row of the table calculated mod 11 doesn't seem to have any obvious pattern.

This is the Discrete Log Problem:

Given values a, b and p, find the value x which satisfies the equation:

$$b^x \equiv a \pmod{p}$$

As we can see from the illustration above, the regular log problem is "easy" to solve. We can fairly quickly compute the value of a regular log (in fact, most calculators have log buttons!) But, as it turns out, no one has solved the discrete log problem, generally speaking, in an efficient manner.

In fact, the security of several public key cryptography systems relies on our belief that the discrete log problem is difficult and can not be efficiently solved. If someone were to do so, then those cryptosystems would be rendered immediately insecure.

If we are making calculations mod 11, let us consider a different base than 2, say 10. Here is the corresponding table for 10 raised to the exponents 0 through 11:

Exp	0	1	2	3	4	5	6	7	8	9	10	11
Value	1	10	1	10	1	10	1	10	1	10	1	10

A few things to notice here:

1) this table is quite a bit less interesting than the one using 2 as the base.

2) this table only has 2 unique entries (1, 10), while the other one has all possible non-zero mods, mod 11.

Thus, unlike the regular logarithm problem, where the choice of base doesn't seem to change the nature of the problem that much (if we use log base 2 or log base 10, there's an answer for any positive real number of which we are taking the log), here it almost seems as if some bases shouldn't even be allowed, because they don't produce all possible "answers."

Let's quickly generate a table using all possible bases (1 through 10), raised to all possible exponents (0 through 10), stating the results mod 11 and let's try to make some observations about this table:

B\E	0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	5	10	9	7	3	6	1
3	1	3	9	5	4	1	3	9	5	4	1
4	1	4	5	9	3	1	4	5	9	3	1
5	1	5	3	4	9	1	5	3	4	9	1
6	1	6	3	7	9	10	5	8	4	2	1
7	1	7	5	2	3	10	4	6	9	8	1
8	1	8	9	6	4	10	3	2	5	7	1
9	1	9	4	3	5	1	9	4	3	5	1
10	1	10	1	10	1	10	1	10	1	10	1

The first repeat of 1 is noted on each row with a yellow highlight. Notice that for four rows (bases 2, 6, 7 and 8), the first repeat occurrence of 1 is at exponent 10, and that these four rows roughly represent random permutations of the values {1, 2, 3, …, 10}, all non-zero remainders when dividing by 11. On all of the rest of the rows though, not all possible remainders are seen. In 4 more rows (bases 3, 4, 5 and 9), only half of the 10 possible non-zero remainders appear (and the ones that appear are those values and 1), in one row two remainders appear and in another only 1 remainder appears.

The bases for which each possible non-zero remainder is generated have a special name: they are called either **generators** or **primitive roots** mod p.

The proof of why there exists at least one primitive root mod each prime number greater than 2 is quite detailed. It uses the factorization of the expression $x^{p-1} - 1$, and then looks at the number of roots mod p. This video discusses the proof in detail:

https://www.youtube.com/watch?v=kHrNBwsM3lY

Once we know there is one primitive root though, it's fairly easy to calculate the total number of primitive roots a prime number will have. Let's look at our table again:

B\E	0	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	5	10	9	7	3	6	1
3	1	3	9	5	4	1	3	9	5	4	1
4	1	4	5	9	3	1	4	5	9	3	1
5	1	5	3	4	9	1	5	3	4	9	1
6	1	6	3	7	9	10	5	8	4	2	1
7	1	7	5	2	3	10	4	6	9	8	1
8	1	8	9	6	4	10	3	2	5	7	1
9	1	9	4	3	5	1	9	4	3	5	1
10	1	10	1	10	1	10	1	10	1	10	1

Let's just analyzed rows 2 and 4:

2	1	2	4	8	5	10	9	7	3	6	1
4	1	4	5	9	3	1	4	5	9	3	1

noting that $4 = 2^2$. In some sense, the row for 4 is really "skipping" every other value on the row for 2. (Notice the highlights in blue on the row for 2.) This makes perfect sense if you think about it. $4^1 = 2^2$, $4^2 = 2^4$, $4^3 = 2^6$, $4^4 = 2^8$ and $4^5 = 2^{10}$. We can think of modular exponentiation as running around a track with markers. If we have a primitive root like 2, then we can label our markers around the track as 1, 2, 4, 8, 5, 10, 9, 7, 3, 6, and 1. When we exponentiate any other number, such as 4, then instead of moving by one marker each time we multiply, we move by 2 markers. If we are exponentiating 9 (which is 2^6 mod 11), then that is as if we move 6 markers each time we multiply.

It's sort of like the problem: 10 people start at the starting line of the track. These people move 1, 2, 3, …, 10 markers, respectively, per second. For each person, mark the first integer time they return to their starting point. If anyone moves a distance that shares a common factor with the number of markers (so in our case, 2, 4, 6, 8, 10, and 5), then these people will coincide with the starting line in fewer than 10 steps. (In fact, since we know EVERYONE returns to the starting point in 10 steps, we know that the amount of time each of these people will take will divide evenly into 10.) So, now, what we find out is, is that the number of people who first return to the starting

line at time 10 is the set of people whose numbers don't share a common factor with 10. But we have a definition for that number, it's nothing but $\varphi(10) = 4$.

It follows that, given one primitive root, x mod p, then there are $\varphi(p - 1)$ primitive roots total. Furthermore, we know what these roots are, the are:

$$x^{a_1}, x^{a_2}, \dots, x^{a_{\phi(p-1)}}$$

Where the values $a_1, a_2, \dots, a_{\phi(p-1)}$, are the unique values mod p – 1 that do NOT share a common factor with p – 1.

In our specific example, if we use x = 2 and p = 11, then these four primitive roots are:

$2^1, 2^3, 2^7, 2^9 \pmod{11}$

Simplifying mod 11 we get: 2, 8, 7, and 6, respectively.

Thus, for the discrete log problem, we limit ourselves to the use of bases, b, that are primitive roots. (As noted above, these are also called generators.) This is so there will always be an answer, to the equation $b^x \equiv a \pmod{p}$, no matter what value of a is chosen from the set {1, 2, …, p – 1}.

To determine if a base, b, is a primitive root or not, what we need to do is find each unique prime factor y, of p – 1, and then calculate b^y mod p. If any of these are equal to 1, then the base is NOT a primitive root (since we found a smaller exponent to raise b to, to obtain 1), but if NONE of these are equal to 1, we are guaranteed that b is a primitive root.