The Baby Step, Giant Step Method

The following prime is not so easy to deal with.

\[ p = 633073699 \]

\[ 633073699 \]

\[
\alpha = \text{numlib::primroot}(p)
\]

\[ 3 \]

We first find \( N \) for the "giant steps".

\[ N = \text{ceil}(\sqrt{p - 1}) \]

\[ 25161 \]

Say we want to find \( L(15679625) \). We need to generate two lists.

\[
b = 15679625:
\]

\[
\text{firstlist} := \text{matrix}(\array{1..N, [\text{powermod}(\alpha, j, p) \; \text{\$} \; j=0..N-1]})
\]

Now we compute the size of the "giant steps".

\[
c = \text{powermod}(\alpha, -N, p):
\]

\[
\text{secondlist} := \text{matrix}(\array{1..N, [\text{mod}(b\times\text{powermod}(c, j, p), p) \; \text{\$} \; j=0..N-1]})
\]

Now we look for a coincidence between the two lists. The obvious method takes \( N^2 \) comparisons, which is slow. So let's do something that takes only a single loop.

The following is the concatenation of the two lists:

\[
u := \text{coerce(firstlist, DOM_LIST)}.\text{coerce(secondlist, DOM_LIST)}:
\]

Now we search for duplications.

\[
v := \text{listlib::removeDuplicates}(u, \text{KeepOrder}):
\]

Now we see how lists \( u \) and \( v \) differ.

\[
\text{for } j \text{ from } 1 \text{ to } 2*N \text{ do}
\]

\[
\text{if } v[j] \neq u[j] \text{ then print}(j); \text{ break; end_if; end_for}
\]

\[ 48697 \]

This was the only slow step; it took 117 seconds of CPU time. OK, that means that at the entry number 48697 of list \( u \), there was a duplication of an earlier entry. So this happened at entry number 48697-N of the second list.

\[
j = 48697-N
\]

\[ 23536 \]

\[
t := \text{secondlist}[23536]:
\]

\[
\text{for } k \text{ from } 1 \text{ to } N \text{ do}
\]

\[
\text{if } \text{firstlist}[k] = t \text{ then print}(k); \text{ break; end_if; end_for}
\]

\[ 12815 \]

We can assemble this to get the discrete log, \( x \).

\[
x := 12814 + 23535*N
\]

\[ 592176949 \]

Check:

\[
\text{powermod}(\alpha, x, p)
\]

\[ 15679625 \]

\[ \% = b \]
15679625 = 15679625

Yep, it works!