The Pollard \( \rho \) algorithm is a method for solving discrete logarithm problems that works in time \( O(\sqrt{\text{group size}}) \). We will use it for \((\mathbb{Z}/p\mathbb{Z})^*\), though it doesn't use any specific features of \((\mathbb{Z}/p\mathbb{Z})^*\). As a result, it is a "general purpose" algorithm that is (in a provable sense) the best possible.

Here's how it works. We are given \( p \) and a generator \( g \) of \((\mathbb{Z}/p\mathbb{Z})^*\). Given an element \( h \in (\mathbb{Z}/p\mathbb{Z})^* \), the discrete logarithm problem (DLOG) asks us to find some integer \( y \) such that

\[
g^y \equiv h \pmod{p}.
\]

Much like the "Pollard \( \rho \) for factoring" algorithm we studied earlier, this involves taking something that looks like a random walk. Namely, we need a random-looking rule that tells us where to "walk" from a given vertex. We do this first by partitioning the group \((\mathbb{Z}/p\mathbb{Z})^*\) into 3 random subsets...
of roughly equal size:

\[ S_1, S_2, S_3. \]

You can think of this as randomly coloring each vertex one of 3 colors. Pollard suggested a particular rule which is easy to implement. Note that storing a list of \( S_1, S_2, S_3 \) in general would take an absurd amount of memory, so in practice tricks are found to get around that (this is not a significant detail for us).

Now that \( \mathbb{Z}/p\mathbb{Z} \) has been divided up into \( S_1, S_2, S_3 \), we can define our iterating function:

\[
 f(x) = \begin{cases} 
 g(x)(\text{mod} p) & \text{if } x \in S_1, \\
 h(x)(\text{mod} p) & \text{if } x \in S_2, \\
 x^2(\text{mod} p) & \text{if } x \in S_3.
\end{cases}
\]

This behaves pretty randomly, but what is crucial is that it is determined by a rule.
Like the Pollard $\rho$ for factoring algorithm, we have an iteration:

Let $X_0 = 1$ (more generally, you can take $X_0 = g^{t_0} h^{r_0}$, for random powers, which must then keep track of).

Now let

\[ X_1 = f(X_0) \]
\[ X_2 = f^2(X_0) \]
\[ X_3 = f^3(X_0) \]
\[ \vdots \]
\[ X_{k+1} = f^k(X_k) \]

Eventually, one must have $X_k = X_{k'}$ since \((\mathbb{Z}/N^\times)^*\)

From then on, the iteration goes in a loop,

The collision probably happens for

\[ k \cdot l = O(\sqrt{P}) \]

with very high probability. This is one of the few times one can prove how long an algorithm
We detect the collision using Floyd's cycle detection algorithm exactly like we did for the Pollard's factorizing algorithm: we look to see when

\[ x_k = x_{2k}, \]

This means far fewer checks, far less storage.

What to do with a collision?

An important difference with the Pollard's factorizing algorithm is that we actually detect the collision. How does it help us solve discrete logs?

Let's analyze the above algorithm more carefully. Lemma: At each stage,

\[ x_k = g^{ak+bk}, \]

with exponents \( ak+bk \) that we can explicitly compute,
Proof \ we \ start \ with \n
\[ X_0 = l = g^0 h^0, \text{ so} \]
\[ (a_0, b_0) = (0, 0). \]

We will argue by induction, and will show how to compute the exponents in

\[ X_{k+1} = g^{a_k+1} h^{b_k+1} \]

knowing the exponents in \( X_k = g^{a_k} h^{b_k} \).

We break into 3 cases depending on which set \( S_1, S_2, \text{ or } S_3 \) \( X_k \) belongs to:

\[ X_k \in S_1 \]

Then \( f(X_k) = gX_k = g^{a_k+1} h^{b_k} \),

Thus

\[ X_{k+1} = f(X_k) = g^{a_k+1} h^{b_k+1} \]

Where

\[ (a_{k+1}, b_{k+1}) = (l + a_k, b_k) \]

\[ X_k \in S_2 \]

Then \( f(X_k) = hX_k = g^{a_k} h^{b_k+1} \),

\[ X_{k+1} = g^{a_k+1} h^{b_k+1} \]

Where

\[ (a_{k+1}, b_{k+1}) = (a_k, l + b_k) \]

\[ X_k \in S_3 \]

Now

\[ X_{k+1} = f(X_k) = X_k^2 = g^{2a_k} h^{2b_k}, \text{ so} \]
\[ (a_{k+1}, b_{k+1}) = (2a_k, 2b_k). \]
Thus we can write the exponents explicitly in each of the 3 cases:

\[ X_k = X_e, \]

so

\[ a_k h_k = a_e h_e. \]

Let's put this together. At a collision,

\[ h = g, \]

let's write \( h = g \) and \( y \) unknown. See what these equal expressions reflect about \( y, \) \( t, \) \( a_e \) and \( a_k \).

\[ g_a t a_k y \] (nearly)

which means the exponents satisfy

\[ a_k t a_k y = a_e t a_k y \text{ (mod p)} \]

we need to solve for \( y \). Rearrange:

\[ (b_k - b_e) y = a_e - a_k \text{ (mod p)} \]

If we can list all solutions, the...
will be among them. In particular, if
\[ \gcd(k - b_e, p - 1) = 1 \]
then \( k - b_e \) has a modular inverse \( (k - b_e)^{-1} \) (mod \( p - 1 \)),

\[ X = (k - b_e)^{-1} (a_e - ak) \]
is the discrete logarithm of \( h \).

**Example** Pollard suggests
\[ S_1 = \{ 0 < x < \sqrt{3} \} \]
false \[ S_2 = \{ \sqrt{3} < x < 2\sqrt{3} \} \]
true \[ S_3 = \{ 2\sqrt{3} < x < 3 \} \]
Try \( p = 31 \), \( g = 3 \), \( h = 7 \).

\[ X_0 = 1 \]
\[ X_1 = f(X_0) = gX_0 = 3 \quad g^1 h^0 \]
\[ X_2 = f(X_1) = gX_1 = 9 \quad g^2 h^0 \]
\[ X_3 = f(X_2) = gX_2 = 27 \quad g^0 h^0 \]
\[ X_4 = 27^2 \equiv 16 \pmod{31} \quad g^6 h^0 \]
\[ X_5 = 7 \times 16 = 112 \equiv 19 \pmod{31} \quad g^6 h^1 \]
\[ X_6 = 7 \times 19 = 133 \equiv 9 \pmod{31} \quad g^6 h^2 \]
So, \( x_6 = x_2 \), and:

\[ g^6 h^2 \equiv g^2 h^0 \pmod{p} \]

\[ g^6 z^2 \equiv g^2 \pmod{p} \]

\[ \equiv 6 + 2y \equiv 2 \pmod{p-1=30} \]

\[ 2y \equiv -1 \pmod{31} \]

So, \( y = -2 \quad \text{or} \quad 13 \)

Check:

\[ 3^{-2} = \frac{1}{9} \]

(\( \equiv 28 \))

(\( \text{and} \quad 3^{-1} = -10, \quad 100 \equiv 7 \pmod{31} \)).