Notes 8.370/18.435 Fall 2022

Lecture 23 Prof. Peter Shor

Today, we are talking about the discrete log algorithm. This is a problem that is very similar in some respects to the factoring problem. While there is no formal reduction between these problems, they both can be used for public key cryptosystems, and every time somebody has found a better algorithm for one of them, this discovery has been followed by the discovery of an analogous algorithm for the other one.

What is the discrete log problem? We will assume we have a prime P (the discrete log problem is also defined for other numbers, but the algorithm is easier for a prime). The multiplicative group modulo a prime P always has a generator g, where any non-zero number h modulo P can be represented as $g^x \equiv h \pmod{P}$. For example, 2 is a generator modulo 11 because he sequence of powers of $2 \pmod{11}$ is:

$$2^1 = 2, 2^2 = 4, 2^3 = 8, 2^4 = 5, 2^5 = 10, 2^6 = 9, 2^7 = 7, 2^8 = 3, 2^9 = 6, 2^{10} = 1,$$

and covers all ten non-zero numbers modulo 11. The number 3 is not a generator, because $3^5 = 243 \equiv 1 \pmod{11}$, and thus only five numbers are powers of 3 modulo 11.

Given a prime P, and a potential generator g, there is an efficient quantum algorithm for testing whether g is a generator for the multiplicative group P – you compute the periodicity of the sequence

$$1, g, g^2 \pmod{P}, g^3 \pmod{P}, g^4 \pmod{P}, \ldots$$

and see whether it is $P - 1$. There's no efficient classical algorithm to test whether something is a generator g for a prime P, but in fact, for the classical application we will discuss (Diffie-Hellman key exchange), anything that looks like a generator will do.

The discrete logarithm problem is important for classical cryptography. One cryptographic protocol that relies on the hardness of discrete logarithms is fhe Diffie-Hellman key exchange protocol.

Suppose that Alice and Bob are two people who would like to communicate securely, but only have access to a channel that they believe an eavesdropper has access to. If they share a secret that nobody else knows, they can use this to communicate securely, by using it as the key for some secret-key cryptosystem.

Suppose they don't have a secret, what can they do? Key exchange protocols generate secrets that only the two participants know. The Diffie-Hellman key exchange protocol is one of the oldest, and will let Alice and Bob generate a secret that only they know (assuming that discrete log is hard).

How does Diffie-Hellman work? Alice and Bob agree on a large prime P and a generator g for it. They then each choose a random number between 2 and $P-2$. Let's say that Alice chooses x and Bob chooses y. Alice sends Bob $g^x \pmod{P}$, and Bob sends Alice $g^y \pmod{P}$. Alice then raises g^x to the y'th power, and Bob raises g^y to the x'th power (mod P), so they both obtain $g^{xy} \pmod{P}$. This is their shared secret.

What does an eavesdropper know about their secret? She knows g, $g^x \pmod{P}$, and $g^y \pmod{P}$. If the eavesdropper could find discrete logs, she could use g^x

$\pmod{P}$ and g to find x, and then use this to obtain $g^{xy} \pmod{P}$. However, we assume that discrete logs are hard. While we cannot formally prove that finding $g^{xy} \pmod{P}$ from the three values g, $g^x \pmod{P}$, and $g^y \pmod{P}$ is as hard as finding discrete logs modulo P, nobody knows how to do it, and most cryptographers are convinced that any method for breaking Diffie-Hellman will also serve to find discrete logs.

How do you find $g^x \pmod{P}$? One way is to find $g^2 \pmod{P}$, $g^4 \pmod{P}$, $g^8 \pmod{P}$, and so forth. You can compute each of these quantities by squaring the previous one mod P, so you can find all of these efficiently. You then write out x in binary: $x = x_{L-1}x_{L-2}\ldots x_1x_0$.

$$\begin{array}{lllllll} 1 & g & g^2 \,(\mathrm{mod}\, P) & g^4 \,(\mathrm{mod}\, P) & g^8 \,(\mathrm{mod}\, P) & \ldots & g^L \,(\mathrm{mod}\, P) \\ x_0 & x_1 & x_2 & x_3 & x_4 & \ldots & x_L \end{array}$$

Now multiplying the powers g^{2^i} where $x_i = 1$ will give $g^{\sum 2^i x_i} = g^x$. Essentially the same idea was used for the phase estimation algorithm.

How does the quantum discrete log algorithm work? Our presentation of it is based on phase estimation. This isn't the original version of the discrete log algorithm, and it isn't the version in Nielsen and Chuang, either, but I believe it is simpler than these.

We will be applying the phase estimation algorithm for two unitaries. One of these is:

$$U_g : |\, y \,(\mathrm{mod}\, P)\rangle \to |\, gy(\bmod P)\rangle \ .$$

Recall that to make a reversible circuit that doesn't keep the input around, we need to have both a circuit for the function and a circuit for the inverse. We can do this. The number g has an inverse $g^{-1} \pmod{P}$, and $gg^{-1} = 1$. For example, if P is 31 and g is 3. then g^{-1} is 21, because $3 \cdot 21 = 63 \equiv 1 \pmod{31}$. We explained how to compute g^{-1} later in a previous set of lecture notes. The inverse of the unitary U_g is:

$$U_g^{-1} : |\, y \,(\mathrm{mod}\, P)\rangle \to \left|\, g^{-1}y(\bmod P)\right\rangle \ .$$

Recall also that for the phase estimation algorithm, if we want an accurate approximation of the phase, we need to be able to compute $U_g^{2^k}$. This can be done efficiently:

$$U_g^{2^k} = U_{g^{2^k}} : |\, y \,(\mathrm{mod}\, P)\rangle \to \left|\, g^{2^k} y(\, 9\bmod P)\right\rangle \ ,$$

and we can compute $g^{2^k} \pmod{P}$ classically, and use the result to make a quantum circuit for $U_{g^{2^k}}$.

So what are the eigenvectors of U_g? I claim that they are

$$|\, v_k\rangle = \frac{1}{\sqrt{P-1}} \Big(|\, 1\rangle + e^{2\pi k i \frac{1}{P-1}} \,|\, g\rangle + e^{2\pi i k \frac{2}{P-1}} \left|\, g^2\right\rangle + e^{2\pi i k \frac{3}{P-1}} \left|\, g^3\right\rangle + \ldots e^{2\pi i k \frac{P-2}{P-1}} \left|\, g^{P-2}\right\rangle \Big) \ .$$

where all the integers in the kets are taken mod P.

What happens when you apply U_g to this state? We get

$$\begin{aligned} U_g \,|\, v_k\rangle &= \frac{1}{\sqrt{P-1}} \Big(|\, g\rangle + e^{2\pi i k \frac{1}{P-1}} \left|\, g^2\right\rangle + e^{2\pi i k \frac{2}{P-1}} \left|\, g^3\right\rangle + e^{2\pi i k \frac{3}{P-1}} \left|\, g^4\right\rangle + \ldots e^{2\pi i k \frac{P-2}{P-1}} \,|\, 1\rangle \Big) \\ &= e^{-2\pi i k/(P-1)} \,|\, v_k\rangle \ . \end{aligned}$$

Here the last term is $|\,1\rangle$ because $g^{P-1} = 1$. This is true for any generator of the multiplicative group (mod P) because 1, g, g^2, . . ., g^{P-2} (mod P) must be the numbers 1, 2, . . ., $P-1$ in some order.

Now, suppose we take the state $|\,1\rangle$, apply the phase estimation algorithm, and measure the second register. We start with

$$|\,1\rangle = \frac{1}{\sqrt{P-1}} \sum_{k=0}^{P-2} |\,v_k\rangle \ .$$

This is because the amplitude on $|\,1\rangle$ is just $(P-1)\left(\frac{1}{\sqrt{P-1}}\right)^2$, and the amplitude on $\left|\,g^\ell\right\rangle$ for $\ell \neq 0$ is $\frac{1}{P-1}\sum_{\ell=0}^{P-1} e^{2\pi i k\ell/(P-1)}$, which is a geometric sum that sums to 0.

The phase estimation algorithm estimates the eigenvalue of v_k, which is $e^{-2\pi i k/(P-1)}$. Let's say the estimate is θ_k. The phase estimation takes

$$|\,v_k\rangle \to |\,v_k\rangle\,|\,\theta_k\rangle \ ,$$

where θ_k is an estimate of $-\frac{k}{P-1}$ of the form $\frac{d}{2^L}$.[1]

Because of round-off error, each $|\,v_k\rangle$ will give us several different values of $\theta_k = \frac{d}{2^L}$:

$$|\,v_k\rangle \to |\,v_k\rangle \sum_{\theta_k^{(j)}} \alpha_j \left|\,\theta_k^{(j)}\right\rangle \ .$$

How far off are these estimates? The analysis of the phase estimation algorithm says that they will typically be off by $\frac{a}{2^L}$, where a is a small integer. Thus, in order to be able to find the right value of $\frac{k}{P-1}$, we should take 2^L to be at least a small factor grater than P ($2^L \approx 2P$ probably isn't good enough, but $2^L \approx 20P$ should be).

Let us apply the phase estimation algorithm to the state $|\,1\rangle = \sum_{k=0}^{P-2} |\,v_k\rangle$. We get

$$\frac{1}{\sqrt{P-1}} \sum_{k=0}^{P-2} |\,v_k\rangle \sum_{\theta_k^{(j)}} \alpha_j \left|\,\theta_k^{(j)}\right\rangle$$

Now, if we measure $|\,\theta_k\rangle$, we get each possible value of k between 0 and $P-2$ with equal probability, along with an estimate for $\frac{k}{P-1}$, and the first register remains in the state $|\,v_k\rangle$. Thus, we can assume that we have determined k and also have the eigenvector $|\,v_k\rangle$.

Recall that $h = g^x \pmod{P}$, and that we want to find x.

[1] Strictly speaking, it's an estimate of $1 - \frac{k}{P-1}$, but we can always subtract 1 from it.

We claim that $|\, v_k \rangle$ is also an eigenvalue of U_h. Why?

$$\begin{aligned} U_h \,|\, v_k \rangle &= U_h \,\frac{1}{\sqrt{P-1}} \sum_{\ell=0}^{P-2} e^{2\pi i k \ell/(P-1)} \,\big|\, g^\ell \big\rangle \\ &= \frac{1}{\sqrt{P-1}} \sum_{\ell=0}^{P-2} e^{2\pi i \ell k/(P-1)} \,\big|\, h g^k \big\rangle \\ &= \frac{1}{\sqrt{P-1}} e^{-2\pi i k x/(P-1)} \sum_{\ell=0}^{P-2} e^{2\pi i k x/(P-1)} e^{2\pi i k \ell/(P-1)} \,\big|\, g^x g^\ell \big\rangle \\ &= \frac{1}{\sqrt{P-1}} e^{-2\pi i k x/(P-1)} \sum_{\ell=0}^{P-2} e^{2\pi i k (x+\ell)/(P-1)} \,\big|\, g^{x+\ell} \big\rangle \ , \end{aligned}$$

and this last term is just $e^{-2\pi i k x/(P-1)} \,|\, v_k \rangle$.

So what we do is we apply the phase estimation algorithm to the state $|\, 1 \rangle$ to get a random eigenvector $|\, v_k \rangle$ and an approximation θ to the phase $\frac{k}{P-1}$. We then take the eigenvedctor $|\, v_k \rangle$ that the algorithm gave us, and apply the phase estimation algorithm to this eigenvector with the unitary U_h. This gives os $\frac{kx}{P-1}$. But now,

Now we know $k \pmod{P-1}$ and $kx \pmod{P-1}$, and we want to find x. How can we do this? If k and $P-1$ are relatively prime, we can find $k^{-1} \pmod{P-1}$ and multiply it by $kx \pmod{P-1}$ to get $x \pmod{P-1}$. For example, suppose we use the generator 3 for the multiplicative group (mod 31) and are trying to find the discrete log of $16 \pmod{31}$. We get the numbers $k = 7 \pmod{30}$ and $kx = 12 \pmod{30}$. $7^{-1} \pmod{30} = 13$ because $7 \cdot 13 = 91 = 1 \pmod{30}$. Now, we multiply k^{-1} by kx, namely $13 \cdot 12 = 156 = 6 \pmod{30}$, and we have our discrete log. One can check that $3^6 = 729 \equiv 16 \pmod{31}$.

What happens when we don't find a k relatively prime to $P-1$? For example, suppose we got $k = 5$ and $kx = 0$. In this case, we can't divide—for example, if $x = 6$ and $P - 1 = 30$, the numbers $x = 6$, 12, 18, 24, all give $kx = 0 \pmod{30}$. Here, maybe the simplest thing to do is try again until we find a k that is relatively prime to $P-1$. There are cleverer things to do which will result in our needing to run the quantum part of the algorithm fewer times ... I'll leave finding these as an exercise.