Notes 8.370/18.435 Fall 2022

Lectures 21 Prof. Peter Shor

1 The Factoring Algorithm

The first quantum computer algorithm that really caught people's attention was the quantum factoring algorithm. This was in part because the security of many applications on the internet depended on the hardness of factoring. In particular, the RSA (Rivest-Shamir-Adleman) public key cryptosystem is based on the hardness of factoring. The receiver multiplies two large primes P and Q and tells everybody the number N. To encode a message, you only need N, but to decode it, you need to know P and Q. Thus, two people can communicate securely over a public channel without having any shared secrets in advance. In our lecture today, we will assume that we know a product of two primes: $N = PQ$, and want to factor it. The factoring algorithm works just as well on a product of more than two primes, so this restriction is only for pedagogical reasons.

How did I discover the factoring algorithm? It was a lot more convoluted than I am going to make it sound, but the basic idea is that Simon's algorithm uses period-finding over $\mathbb{Z}_2^n$—it finds a c such that $f(x) = f(x \oplus c)$. It turns out that period finding over $\mathbb{Z}$ is a key ingredient in the factoring algorithm—find a c such that $f(x) = f(x + c)$. SImon's algorithm uses the Hadamard transform to do it, but the Hadamard transform is the Fourier transform over $\mathbf{Z}_2^n$. To find periods over $\mathbb{Z}$, we can use the Fourier transform over $\mathbb{Z}_{2^n}$.

Today, we will start on the factoring algorithm. How do factoring algorithms work?. They use many different methods, but one technique used by both the quantum factoring algorithm and the quadratic sieve (the second best classical factoring algorithm), is to find two numbers such that $A^2 \equiv B^2 \pmod N$ but where $A \neq \pm B \pmod N$. If we have this, then we have $(A - B)(A + B) \equiv 0 \pmod N$ which means

$$(A - B)(A + B) \text{ is a multiple of } N$$

However, N doesn't divide either $A-B$ or $A+B$, so one of P or Q must divide $A-B$ and the other one must divide $A + B$. This means we can recover the two primes P and Q.

How do we recover P (or Q) from $A - B$? We take the greatest common divisor of N and $P - Q$ using the Euclidean algorithm, which will be in the following lecture notes. (In order not to keep interrupting our description of the factoring algorithm, we are postponing all the number theory to the following lecture notes.)

But how do we find the two numbers A and B? This is where we use the period-finding algorithm.

Let's look at the function $f(x) = a^x \pmod N$. This gives

$$1, a, a^2 \pmod N, a^3 \pmod N, \ldots, a^k \pmod N.$$

Eventually, because there are only a finite number of residues modulo N, this sequence will start repeating. We will get $a^k \equiv a^{k+r} \pmod N$. If a is relatively prime to N,

then we can divide both sides by a^k and get $a^r \equiv 1 \pmod N$. Thus, r is the period of this sequence.

Now we have $a^r \equiv 1$. If r is even, this gives

$$\left(a^{r/2}\right)^2 \equiv 1^2 \pmod N,$$

and we can hope $a^{r/2} \neq -1 \pmod N$. If it isn't then we have a potential factor. And if it is, we were unlucky and can try again. It is possible to show that repeating this method for different values of a will eventually give a factor within a polynomial number of trials. Again, we will postpone this discussion to the next lecture notes.

Let's do an example. Let's try to factor $N = 33$. You probably already know this factorization, $P = 11$ and $Q = 3$. But we'll see how the algorithm does it.

First, let's choose $a = 2$. We get, for $a^k \pmod N$:

a^0	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}
1	2	4	8	16	32	31	29	25	17	1

So we have $a^{10} = 1 \pmod{33}$ so $(a^5 + 1)(a^5 - 1) = 0 \pmod{33}$ Unfortunately, it doesn't work this time because we chose the wrong value of a: since $2^5 + 1 = 33$, so we don't get a factor.

Let's try $a = 5$:

a^0	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}
1	5	25	26	31	23	16	14	4	20	1

This time, again $a^{10} = 1$. Now $a^5 = 23$, and $(23-1)(23+1) = 0 \pmod{33}$. However, now $23 - 1$ is a multiple of $P = 11$ and $23 + 1$ is a multiple of $Q = 3$.

So how do we find the period of a sequence? One way we can do this is to use a unitary transformation that takes us from one element of the sequence to the next. In this case, the function is simple

$$U_a \left|y \pmod N\right\rangle = \left|ay \pmod N\right\rangle.$$

We need to implement this function reversibly on a quantum computer. For that, we need $a^{-1} \pmod N$. Here, a^{-1} is the residue modulo N such that $a^{-1}a \equiv 1 \pmod N$. Recall that to implement a reversible computation that took the input to the output without leaving any extra non-constant bits around, we needed to be able to compute both the function and its inverse.

Now, we can find classical circuits for computing

$$V_a \left|y \pmod N\right\rangle \left|0\right\rangle = \left|y \pmod N\right\rangle \left|ay \pmod N\right\rangle$$

and its inverse,

$$V_{a^{-1}} \left|ay \pmod N\right\rangle \left|0\right\rangle = \left|ay \pmod N\right\rangle \left|y \pmod N\right\rangle,$$

so we can combine them to get the unitary transform U_a above.

Recall that to apply the phase estimation algorithm, we need to be able to perform the unitaries U_a^2, U_a^4, U_a^8, and so forth. We can do this: U_a^2 is just $U_{a^2 \pmod N}$, U_a^4 is just $U_{a^4 \pmod N}$ and in general, $U_a^{2^k}$ is just $U_{a^{2^k} \pmod N}$. We can find $a^{2^k} \pmod N$ by repeatedly squaring a to get a^2, a^4, a^8, $\ldots \pmod N$. We can thus implement $U_{a^{2^k}}$, and so can do phase estimation.

What are the eigenvectors and eigenvalues of U_a? Consider the quantum state (leaving out the (mod N)'s to save space)

$$|\zeta_k\rangle = \frac{1}{\sqrt{r}} \left(|1\rangle + e^{2\pi i k/r} |a\rangle + e^{4\pi i k/r} |a^2\rangle + \ldots e^{2\pi(r-2)k/r} |a^{r-2}\rangle + e^{2\pi(r-1)k/r} |a^{r-1}\rangle \right)$$

What happens when we apply U_a? We get

$$U_a |\zeta_k\rangle = \frac{1}{\sqrt{r}} \left(|a\rangle + e^{2\pi i k/r} |a^2\rangle + e^{4\pi i k/r} |a^3\rangle + \ldots e^{2\pi(r-2)k/r} |a^{r-q}\rangle + e^{2\pi(r-1)k/r} |a^r\rangle \right),$$

and because $|a^r\rangle = |1\rangle$, this $U_a |\zeta_k\rangle$ is $e^{-2\pi k/r} |\zeta_k\rangle$. So we have found r eigenvectors of U_a. If we had one of these eigenvectors, we could use the phase estimation algorithm to approximate its eigenvalue, which would give us an approximation to k/r. This would, hopefully, give us r. But we don't actually have one of these eigenvectors. What do we do?

What we do is use the phase estimation algorithm anyway. If we are trying to factor an L-bit number, we will use the phase estimation algorithm and the quantum Fourier transform with $2L$ qubits. This will actually measure the eigenvector, along with its eigenvalue, and once it is measured, we will get a state very close to $|\zeta_k\rangle$ for some k, and the phase estimation algorithm will give us a good approximation of k/r in the form of $d/2^{2L}$ for some d. From this, we will be able to find r. More specifically, the phase estimation algorithm will give us $d/2^{2L}$ which is very close to k/r.

Why do we need to approximate the phase to $2L$ bits? We are trying to find k/r, and each of k and r is at most L bits, since N is at most L bits. Thus, we need $2L$ bits to have enough information to determine k and r. We can now use a classical number theory algorithm, called continued fractions, to round $d/2^{2L}$ ti k/r (in loweat terms). This number theory algorithm will also be described in more detail in the next lecture notes.

Exactly how does this work? We start the algorithm in the state $|1\rangle$. Note that

$$|1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |\zeta_k\rangle .$$

This means that when we apply the phase estimation algorithm, we will get a random $|\zeta_k\rangle$, and the eigenvalue $e^{-2\pi i k/r}$. The phase estimation algorithm returns a fraction $d/2^{2L}$ close to k/r, and much of the time this fraction can be used to find r.