CMSC 657: Introduction to Quantum Information Processing
Lecture 11

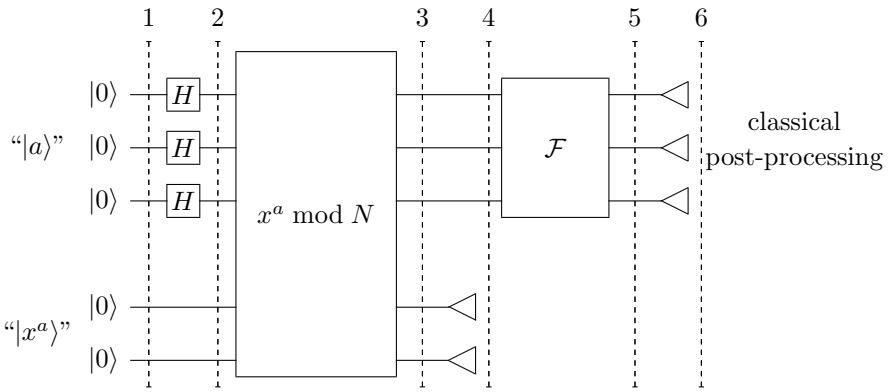
Instructor: Daniel Gottesman

Fall 2024

1 Shor's Algorithm

1.1 Shor's Algorithm overview

Shor's algorithm solves the period finding problem. I will describe it for the function $f(a) = x^a \bmod N$, but it works in the same way for any periodic function with period r that does not otherwise repeat values.



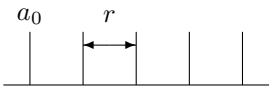
There are two registers, each with multiple qubits. The first register has n qubits and the second register has $\lceil \log N \rceil$ qubits. Note that the labels on the left are simply to remind you what the registers represent. The qubits don't start in that state (although they do get there later in the circuit). The modular exponentiation calculates x^a, where $|a\rangle$ is the value of the first register at this point and x is a fixed value hardcoded into the circuit, and adds it to the second register. That is, this section does $|a\rangle|c\rangle \mapsto |a\rangle|c \oplus x^a\rangle$. The $\mathcal{F}$ is the Fourier transform, specifically the *discrete* Fourier transform over $\mathbb{Z}_{2^n}$:

$$\mathcal{F}|a\rangle = \sum_b \omega^{ab}|b\rangle. \tag{1}$$

Here $\omega = \exp(2\pi i/2^n)$, a 2^n-th root of unity.

To understand this algorithm, let us first imagine an idealized case when $r|2^n$ (recall r is the period of $x^a \bmod N$), and track the state through the algorithm:

1. $|0\rangle|0\rangle$
2. $\sum_a |a\rangle|0\rangle$
3. $\sum_a |a\rangle|x^a\rangle$
4. Measurement result is $z = x^a$. However, $x^{a+jr} = x^a$, so the state is $\sum_j |a_0 + jr\rangle$.



5. After the Fourier transform, we have

$$\sum_j \sum_b \omega^{(a_0+jr)b}|b\rangle = \sum_b \omega^{a_0 b}\left(\sum_j e^{2\pi i j b/(2^n/r)}\right)|b\rangle \tag{2}$$

$$= \sum_c \omega^{a_0 c 2^n/r}|c2^n/r\rangle. \tag{3}$$

   To get the last line, note that if $b = c\alpha$, then $\sum_j \exp(2\pi i j b/\alpha) = \alpha$ (the number of values of j), whereas if $b \neq c\alpha$ for integer c, then $\sum_j \exp(2\pi i j b/\alpha) = 0$ as the phases cancel out around the unit circle. Letting $\alpha = 2^n/r$, we get the last line above. (The factor α gets absorbed into the normalization.)

6. The measurement output will be $c(2^n/r)$ for random c

Post-processing: Repeat the procedure a few times. We get $c_1\alpha$, $c_2\alpha$, $c_3\alpha$, ... Find the gcd of these results to get $\alpha = 2^n/r$, which then gives us r.

In the realistic case, it will not generally be true that $r|2^n$. We also need to make sure there are efficient quantum circuits for the big unitaries calculating $x^a \bmod N$ (the modular exponentiation) and doing the Fourier transform.

Shor's algorithm is useful for breaking other crypto systems and finding periods of other kinds of functions as well.

One other thing to note: We don't actually need to measure the second register, as we don't ever use the outcome of that measurement. It is merely helpful when thinking about the algorithm to put the measurement in, but the algorithm works exactly the same with or without the measurement.

1.2 Fourier transform

How can we implement the Fourier transform efficiently? Recall we are trying to perform the unitary $\mathcal{F}|a\rangle = \sum_b \omega^{ab}|b\rangle$

- Input $a = a_0 2^{n-1} + a_1 2^{n-2} + a_2 2^{n-3} + \ldots + a_{n-1}$
- Output $b = b_0 2^{n-1} + b_1 2^{n-2} + b_2 2^{n-3} + \ldots + b_{n-1}$

We then calculate

$$ab = 2^n(\ldots) + 2^{n-1}(a_0 b_{n-1} + a_1 b_{n-2} + \ldots + a_{n-1}b_0) + 2^{n-2}(a_1 b_{n-1} + \ldots + a_{n-1}b_1) + \ldots + 2^0(a_{n-1}b_{n-1}). \tag{4}$$

The coefficient of 2^n does not matter since ab only appears in ω^{ab} and $\omega^{2^n} = 1$.

It is convenient to write b in reverse order and then separate out the powers of each bit of b in the expression, so that the output of the unitary on basis state input $|a\rangle$ is

$$\sum_b \omega^{ab}|b\rangle = \sum_b \omega^{ab}|b_{n-1}\rangle|b_{n-2}\rangle\cdots|b_0\rangle \tag{5}$$

$$= \left[\sum_{b_{n-1}} \omega^{(a_0 2^{n-1} + a_1 2^{n-2} + \ldots + a_{n-1}2^0)b_{n-1}}|b_{n-1}\rangle\right]\left[\sum_{b_{n-2}} \omega^{(a_1 2^{n-1} + a_2 2^{n-2} + \ldots + a_{n-1}2^1)b_{n-2}}|b_{n-2}\rangle\right]\cdots$$

$$\cdot \left[\sum_{b_0} \omega^{(a_{n-1}2^{n-1})b_0}|b_0\rangle\right]. \tag{6}$$

Let us focus on one of these factors, the b_{n-1-k} term. The sum goes over two values $b_{n-1-k} = 0$ and $b_{n-1-k} = 1$, i.e.

$$|0\rangle + \omega^{p_k}|1\rangle. \tag{7}$$

The power of ω that appears is

$$p_k = \left(a_k 2^{n-1} + \sum_{j>k} a_j 2^{n-1-(j-k)} \right), \tag{8}$$

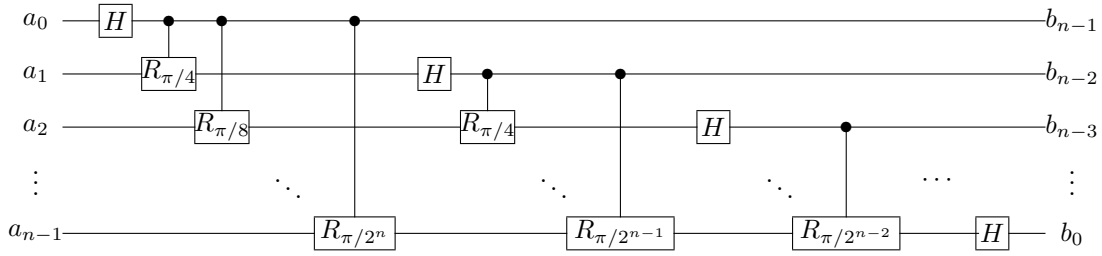
so the phase for $|1\rangle$ is

$$\omega^{a_k 2^{n-1}} \prod_j \omega^{a_j 2^{n-1-(j-k)}} = (-1)^{a_k} \prod_j e^{a_j \pi i / 2^{j-k}}. \tag{9}$$

(And $|0\rangle$ has phase $+1$.)

Recall that we are writing the output in reverse order, so the output bit b_{n-1-k} is the same physical qubit as input qubit a_k. The first phase term $(-1)^{a_i b_{n-1-k}}$ is thus what would be achieved via a Hadamard on the qubit that began as a_k (the kth qubit in the a register before the Fourier transform).

The other phase terms can be achieved via a controlled-$R_{\pi/2^{j-k+1}}$ between the b_{n-1-k} qubit and the a_j qubit. If we first do the Hadamard for qubit k (so it now holds b_{n-1-k}) and then do the phase shifts between the kth qubit and the $(j+1)$th qubit (which still holds a_j since $j \geq k$), we get the desired phase. We do this whole procedure sequentially for each k.



The total complexity of this Fourier transform circuit is $n + (n-1) + (n-2) + \ldots + 1 = O(n^2)$

1.3 Modular exponentiation

This is basically the same as the classical modular exponentiation, but we need to do it in superposition. x is fixed. We can pre-calculate $x^2, x^4, \ldots, x^{2^{n-1}}$ by repeated squaring, as before. This can be done classically because no superposition is needed.

In the quantum part of the algorithm, we decompose $a = \sum_k a_k 2^{n-1-k}$, its binary representation. Using controlled additions, we map

$$|a\rangle|0\rangle \mapsto |a\rangle|x^{a_{n-1}}\rangle|(x^2)^{a_{n-2}}\rangle|(x^4)^{a_{n-3}}\rangle \cdots |(x^{2^{n-1}})^{a_0}\rangle. \tag{10}$$

The control in each case is the ith qubit of the first register. Then we multiply all these powers of x together using a quantum (i.e. reversible) version of some multiplication circuit and then uncompute the intermediate powers of x, getting

$$|a\rangle|x^{a_{n-1}+2a_{n-2}+4a_{n-3}+\ldots+2^{n-1}a_0}\rangle = |a\rangle|x^a\rangle. \tag{11}$$

Now let us calculate the complexity of this procedure:

- The pre-calculation involves $n-1$ multiplications since each power of 2 can be formed by multiplying the previous power by itself.

- There are n controlled-addition steps in the intermediate calculation, and we do it twice (to uncompute), so $2n$ controlled-additions.
- We multiply together n values, so there are $n-1$ multiplications. Many of the values being multiplied are likely to be 1, but we are working in superposition, so we can't see how many are actually 1 and should do all $n-1$ multiplications regardless.
- Each multiplication takes time $O(\log^2 N)$ via long multiplication. Additions take less time.
- The total time for this part of the algorithm is thus $O(n \log^2 N)$. If $n = O(\log N)$, as we will want it to be, we have a time $O(n^3)$.
- The multiplications are time-consuming, but faster classical algorithms for multiplication are known. The total time for modular exponentiation using the best fast multiplication algorithms is asymptotically $O(n^2 \log n \log\log n)$. However, note that you have to be using really really big numbers before the fast multiplication algorithms kick in and become better.

Shor's algorithm as a whole uses $O(n + \log N)$ single-qubit operations (state preparations, Hadamards, and measurements), plus the Fourier transform, which uses $O(n^2)$ operations, as discussed, and the modular exponentiation. The modular exponentiation therefore dominates the complexity of the algorithm.