Notes 8.370/18.435 Fall 2022

Lecture 7 Prof. Peter Shor

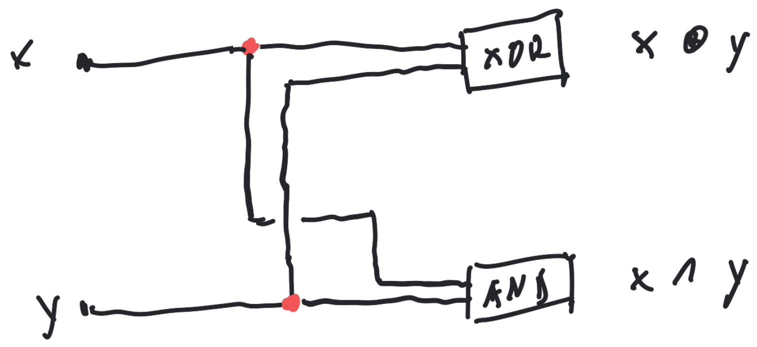
Today's lecture will be on classical Boolean circuits.

There are lots of different models for classical computers—the von Neumann model, the Turing machine, Boolean circuits, and several more. We will be talking about quantum circuits, which are the quantum equivalent of Boolean circuits.

Boolean circuits are one of the most elementary models of computing, but the model that is usually used in the theory of computer science is Turing machines. So why do we use them Boolean circuits and not Turing machines as the model for quantum computers?

When David Deutsch, and later, Umesh Vazirani, started thinking about quantum computers, they indeed started by thinking about quantum Turing machine. However, after I proved the factoring result, I started talking to experimental physicists about quantum Turing machines, and their response was that it is absolutely impossible to design such objects, whereas it's not that difficult to imagine how to build a quantum circuit (building them in practice turns out to be very hard). Thus, the field quickly adapted the quantum circuit model.

The way a classical circuit model works is that there are *gates*. Each gate has one or two inputs, and some small number of outputs. For example, the figure below is a half-adder (the name comes from the fact that you can wire together a bunch of half-adders to make a circuit for adding two binary numbers).



Half-adder circuit

In class, I asked the question: how many gates were in this figure. There were a number of answers. The obvious one is "two", an XOR gate and an AND gate. However, if you look at this figure closely, there are two other possible things you could call gates. Note the two red dots where one wire comes in and two leave. These are FANOUT gates. The place where two wires cross could be called a SWAP gate. And you might need several AND, NOT, and OR gates to implement the XOR gate.

The answer for quantum circuits is "four"; we need to count FANOUT gates, which are the little red dots in the figure, for reasons we will explain later. These gates take in a Boolean value, either 0 or 1, and duplicate it on the two wires leading out. Thus, their truth table is:

in	out
0	00
1	11

.

At this point, I want to discuss the quantum analog of FANOUT gates. These are gates that take

$$|\,0\rangle \rightarrow |\,00\rangle$$
$$|\,1\rangle \rightarrow |\,11\rangle$$

It's clear that the FANOUT gate in this form isn't unitary, because the input space is two-dimensional and the output space is four-dimensional. So how can we implement this on a quantum computer? What we need is a gate that takes a two-dimensional space to a subspace of the four-dimensional space of two qubits. The way we implement them is to add a second qubit in the state $|\,0\rangle$, and then apply a CNOT:

$$|\,0\rangle \rightarrow |\,00\rangle \xrightarrow{\text{CNOT}} |\,00\rangle$$
$$|\,1\rangle \rightarrow |\,10\rangle \xrightarrow{\text{CNOT}} |\,11\rangle$$

How can we reverse this gate? We need to make sure the state is in the subspace generated by $|\,00\rangle$ and $|\,11\rangle$. The easiest way to do this is to first apply a CNOT gate (since CNOT gates are their own inverses) and then measure the second qubit to make sure it is in the state $|\,0\rangle$. If it's not, something has gone wrong, and we would have to report an error.

An important fact about Boolean functions is:

Theorem 1 *Any Boolean function can be computed using AND, OR, and NOT gates, where the inputs are either constant values (1 or 0) or variables ($x_1, x_2, \ldots, x_n$).*

We will prove this by induction. The base case is easy, as the only one-bit Boolean functions are constants, the identity, or NOT. Now, suppose we have a Boolean function $f(x)$ on n variables. We define two functions of $n-1$ variabes,

$$f_0 = f(0, x') \qquad \text{and} \qquad f_1 = f(1, x'),$$

where $x' = x_2, x_3, \ldots, x_n$. By our induction hypothesis, f_0 and f_1 can be built out of AND ($\wedge$), OR ($\vee$), and NOT ($\neg$). But now,

$$f = (x_1 \wedge f(1, x')) \vee (\neg x_1 \wedge f(0, x')) \quad = (x_1 \wedge f_0(x')) \vee (\neg x_1 \wedge f_0(x')).$$

This formula uses two AND gates, one OR gate, and a NOT gate, and two Boolean functions on $n-1$ variables. So if any Boolean function on $n-1$ variables can be computed using $g(n-1)$ gates, a Boolean function on n variables can be solved using

$2g(n-1)+4$ gates. Solving this recurrence shows that $g(n) \propto 2^n$, so any Boolean function on n variables can be computed using around 2^n gates.

Do we really need that many? It turns out the answer is yes, you need exponentially many gates. I'm not going to go into the proof in detail, but the basic idea is that you can count the number of Boolean functions on n variables, and the number of Boolean circuits on m gates, and you choose n and m to make sure that the second quantity is larger than the first quantity.

To do classical circuits on a quantum computer, we may need to use reversible computation.

Unitary gates are reversible, because if $|\,\phi\rangle = U\,|\,\psi\rangle$, then $|\,\psi\rangle = U^\dagger\,|\,\phi\rangle$. Measurement gates are not, so we could perform non-reversible circuits on a quantum computer using measurement gates. However, we will see that measurement tends to destroy quantum interference, and interference is what makes quantum computers more powerful than classical ones. So if we want our quantum algorithms to be more powerful than classical ones, we need to use reversible classical computation on a quantum computer. I will now give a brief overview of reversible classical computation.

It is fairly straightforward to see that there are only two reversible one-bit gates, the identity gate and the NOT gate. What possible reversible two-bit gates are there? First, it's straightforward to see that the SWAP gate is reversible.

We can characterize all two-bit gates by their truth tables. Consider an arbitrary reversible two-bit gate G:

$$G = \begin{array}{c|c} \text{in} & \text{out} \\ \hline 00 & ?? \\ 01 & ?? \\ 10 & ?? \\ 11 & ?? \end{array} \quad .$$

where all four bit strings $??$ are different.

Now, by applying NOT gates after G, we can ensure that 00 goes to 00 (without loss of generality):

$$G' = \begin{array}{c|c} \text{in} & \text{out} \\ \hline 00 & 00 \\ 01 & ?? \\ 10 & ?? \\ 11 & ?? \end{array} \quad .$$

Next, both of 01 and 10 cannot be taken to 11, so by using SWAP gates in front of and behind G' (I'm skipping the details here; you can work them out), we can put it in the form:

$$G'' = \begin{array}{c|c} \text{in} & \text{out} \\ \hline 00 & 00 \\ 01 & 01 \\ 10 & ?? \\ 11 & ?? \end{array} \quad .$$

Now, we need to replace the two $??$'s by 10 and 11. One way of doing this leads to the identity gate. The other way leads to the CNOT gate:

CNOT =

in	out
00	00
01	01
10	11
11	10

.

The formula for the CNOT gate is

$$(a, b) \to (a, a \oplus b)$$

CNOT stands for “controlled NOT”. This gate applies a NOT to the second bit (the target bit) if the first bit (the control bit) is 1 and the identity to the target bit if the control bit is 0.

So can we obtain any two-bit reversible Boolean function by using just SWAP, NOT, and CNOT gates (and in fact, we don’t actually need SWAP gates; later, we will see that a SWAP gate can be buit out of three CNOT gates).

Can we obtain any reversible Boolean function fromm these gates? It turns out that the answer is “‘no”. Why not? If we have time Monday, we will explain it. Otherwise, we may put it on the homework.

Can we obtain any reversible Boolean function with three-bit reversible gates? Yes. it suffices to use NOT gates and Toffoli gates. The Toffoli gate is a controlled controlled NOT, having the following truth table:

Toffoli =

in	out
000	000
001	001
010	010
011	011
100	100
101	101
110	111
111	110

.

Here, a NOT is applied to the third bit if both the first two bits are 1, so the formula is

$$\big(a, b, c\big) \to \big(a, b, c \oplus (a \wedge b)\big).$$

We will explain this in the next lecture.