Notes 8.370/18.435 Fall 2022

Lecture 1: Introduction and History Prof. Peter Shor

Quantum mechanics is a decidedly weird theory. I start with a few quotes about this. On the web, there are lots of quotes attributed to famous physicists saying how strange quantum theory is. Unfortunately, it seems like most of them are apocryphal. Here are some which seem to be verified:

> "Those who are not shocked when they first come across quantum theory cannot possibly have understood it." — Niels Bohr.

> "I admit, of course, that there is a considerable amount of validity in the statistical approach which you were the first to recognize clearly as necessary given the framework of the existing formalism. I cannot seriously believe in it because the theory cannot be reconciled with the idea that physics should represent a reality in time and space, free from spooky actions at a distance." — Albert Einstein

> "If you will simply admit that maybe [nature] does behave like this, you will find her a delightful, entrancing thing. Do not keep saying to yourself, if you can possible avoid it, 'But how can it be like that?' because you will get 'down the drain', into a blind alley from which nobody has escaped."— Richard Feynman

In the first lecture, I'm going to explain the mechanics of the course (see the syllabus for this) and I'm going to give a very abbreviated history of quantum mechanics. Quantum mechanics is a very non-intuitive theory, and in this lecture, I'm going to concentrate not on the usual parts of quantum mechanics, but the non-intuitive parts, so maybe this should be called a history of the discovery of the weirdness of quantum mechanics.

Quantum mechanics was formulated over the early part of the 20th century, but it really came together as a coherent theory in 1925 when several physicists, most notably Werner Heisenberg, Max Born, and Erwin Schrödinger, put together the matrix mechanics formulation of quantum mechanics and the wave-function formulation of quantum mechanics and showed that they were equivalent.

Einstein wasn't particularly happy with the theory they came up with. After a few earlier attempts trying to explain why he didn't like it, he wrote a paper in 1935 with Boris Podolsky and Nathan Rosen (generally called EPR after the initials of the authors) explaining why he thought the then-current formulation of quantum mechanics couldn't be complete. This paper sparked a lot of discussion, with Schrödinger agreeing with Einstein (and coming up with his famous cat to explain what was wrong with quantum mechanics), while Heisenberg, Born, and Wolfgang Pauli took Bohr's side, arguing that the then-current formulation of quantum mechanics was completely satisfactory, and did not need to be changed.

What was the gist of the argument? The Heisenberg uncertainty principle says that you cannot simultaneously measure the position and the momentum of a particle with a

high degree or precision for both measurements. One possible reason for this might be that a particle cannot have both an exact position and an exact momentum; in fact, this is indeed one reason for this. However, Einstein came up with a thought experiment which he thought showed that a particle must necessarily possess both, and from this he concluded that the then-current quantum mechanics was incomplete,

What was Einstein's argument? You can create a state of two quantum particles where, if you measure both of their positions, they will be opposite, and if you measure both of their momenta, they will also be opposite. Take these particles, make sure they're well separated, and measure them simultaneously. The EPR argument assumes that information cannot travel faster than light. And since information about which measurement you chose for the leftmost particle cannot get to the rightmost one faster than light, the rightmost particle must "know" what result it will give for both of these measurements, even though you can only carry out one of these measurements. Thus, quantum mechanics cannot describe the entire state of the rightmost particle, so it is incomplete.

In 1964, John Bell took the ideas of the EPR paper, and formalized them to prove that quantum mechanics violates either realism or locality. (Of course, this theorem depends on your exact definitions of "realism" and "locality".) We will go over his proof later in the course. This theorem involves showing that two particles (that are now called an EPR pair and said to be *entangled*) have probabilities of experimental outcomes that cannot be explained by standard probability theory; i.e., that quantum probability is fundamentally different from classical probability. We will explain Bell's theorem in detail later in the course.

In 1981, Alain Aspect performed the experiment proposed in Bell's paper and showed that the results agreed with the predictions of quantum mechanics, and thus that the quantum "paradoxes" were real. This experiment has been performed many times since. In 2022, three physicists, Alain Aspect, John Clauser, and Anton Zeilinger, were given the Nobel Prize for similar experiments.

Why does physics work like this, and what is the underlying mechanism of the universe? Some physicists have been arguing for decades about what the real structure of the universe might be that would give rise to these rules. They have been regularly coming up with new proposals. These theories of the real structure of the universe are called "interpretations" of quantum mechanics, because they all give the same experimental predictions. However, there's no consensus about which of them are reasonable, or even workable. Furthermore, many of the theories they have come up with are relatively useless for doing actual physics calculations.

One of these interpretations, and possibly the most popular, is the "Copenhagen interpretation." Niels Bohr, one of the physicists who contributed to the invention of quantum mechanics, founded and was director of the Institute for Physics at the University of Copenhagen, where much of the theory of quantum mechanics was discovered. Bohr also wrote several papers on the philosophy of quantum mechanics, some of which are very difficult to understand. The Copenhagen interpretation of quantum mechanics is suposedly the interpretation that Niels Bohr developed there.

There is no consensus as to the exact details of the Copenhagen interpretation; however, it often involves something called "the collapse of the wave function." However, it appears that Bohr did not believe that this collapse was real — one view is that

nothing on the quantum scale is real, but that quantum mechanics is a mechanism for doing calculations so as to predict the outcomes of experiments. Some things Bohr said support the idea that this was his view, for instance, he said "Everything we call real is made of things that cannot be regarded as real." As we have gotten better and better at imaging quantum systems, this view has become increasingly untenable, but the alternative interpretations all have other problems.

When David Mermin (who had generally been dismissive of the Copenhagen interpretation) started teaching quantum computing to computer scientists, he found himself reinventing the Copenhagen interpretation, as this is possibly the interpretation that is easiest to explain. In this course, I will explain the rules of quantum mechanics using a variant of the Copenhagen interpretation, for exactly this reason. And maybe this was one of the reasons that physicists at the Institute of Physics settled on the Copenhagen interpretation during the 1920s and 1930s — they were teaching quantum mechanics to physicists who didn't know it, and the Copenhagen interpretation is the one they naturally came up with.

In 1968, a grad student, Stephen Wiesner, started wondering whether the weirdness of quantum mechanics could be used for something. One thing he came up with was something he called "quantum money". He wrote up the idea, submitted the paper, and it was rejected, at which point he gave up. It wasn't actually published until 1983, when Charlie Bennett sent it to a theoretical computer science newsletter.

Stephen Wiesner's quantum money scheme actually has several flaws in it, but Charlie Bennett and Gilles Brassard took Wiesner's basic idea and came up with a quantum key distribution protocol, now called BB84 (after the proposers and the year it was published) which was sound. This protocol lets two people who have never communicated in the past and who are connected by a (possibly noisy) quantum channel, that an eavesdropper can listen to, establish an absolutely secure secret key. Classically, you can do this only if you rely on the eavesdropper not being able to solve some problem that you believe is computationally hard. Several companies are selling quantum key distribution systems today. If we have time, we will cover this at the end of the course.

In the early 1980s, Feynman started thinking about the complexity of simulating quantum physics on a classical computer. It appeared to require exponential time in the worst case (it still appears to), so he wondered about using quantum computers to simulate quantum systems[1]. He published a paper about this in 1982, and another in 1985, and simulating quantum systems is still believed to be one of the most likely applications of quantum computers, if we ever manage to build them.

Feynman's paper started computer scientists and others thinking about quantum computing. David Deutsch asked the question: "if quantum computers are faster for simulating quantum mechanics, might they be faster for other problems?" David Deutsch and Richard Jozsa found the first algorithm that was significantly faster than a classical algorithm for the same problem (although this speed-up was rather unimpressive). This was followed by an algorithm of Dan Simon which gave much stronger evidence that quantum computers could speed up solving classical problems. Looking

[1] R.P. Poplavskii and Yu. Manin also observed this earlier in the Soviet Union, but they each wrote a few paragraphs about it rather than two full papers.

at Simon's algorithm, In 1994 I found an algorithm that could factor large integers into primes efficiently. Again, we will explain this algorithm later in the course.

Lov Grover, around a year later, found a quantum algorithm that sped up exhaustive search. We will be covering his algorithm as well.

Digital computers are built out of bits, which are systems that can take on two states. Quantum computers are built out of quantum bits (called "qubits" for short) rather than classical bits. On Friday, I will tell you what a qubit is, and we will start explaining how qubits behave.

Notes 8.370/18.435 Fall 2022

Lecture 2 Prof. Peter Shor

Today, I'll start explaining the quantum mechanics you will need for this course. I'll start with a statement of what physicists call the *Superposition Principle* or the *Linearity Principle* of quantum mechanics. The following statement is expressed close to the way you'll see it physics literature.

The Superposition Principle
Suppose $|A\rangle$ *and* $|B\rangle$ *are two perfectly distinguishable quantum states of a quantum system, and* $\alpha, \beta \in \mathbb{C}$ *with* $|\alpha^2 + \beta^2| = 1$. *Then*

$$\alpha |A\rangle + \beta |B\rangle$$

is a valid state of the quantum system.

Here $|A\rangle$ and $|B\rangle$ are notation which physicists use for quantum states. It simply means that A and B can be described by complex column vectors.

What *perfectly distinguishable* means here is that there is some experiment that can distinguish between these two states, at least theoretically (all experiments will have some amount of noise in practice).

There is a second part of this principle—if the experiment distinguishing $|A\rangle$ from $|B\rangle$ is applied to the state $\alpha |A\rangle + \beta |B\rangle$, then the probability of getting the answer $|A\rangle$ is $|\alpha|^2$ and the probability of $|B\rangle$ is $|\beta^2|$.

How can we rewrite this principle using mathematics?

First, the fact that you can add states $\alpha |A\rangle + \beta |B\rangle$ means that the quantum state space is a complex vector space. What also follows, but is harder to see, is that a quantum state is a unit vector in the vector space. Two states are orthogonal if and only if they are perfectly distiguishable.

In this course, we will be talking about quantum computation. Let us recall that classical computers work on bits, which are systems that are in one of two states. Traditionally, we call the values of these bits 0 and 1, independent of the details of their physical representation. For example, on an integrated circuit, a higher voltage might represent a 1 and a low voltage might represent a 0. Or in a magnetic memory, a region where the magnetic north is oriented upward might be a 0 and where it's oriented downward might be a 1.

The same thing is true of qubits. A qubit is a quantum system that can only take on two distinguishable values. But these values might be the polarization of a photon, or the spin of an electron, or the ground state and an excited state of an ion, or a clockwise and a counterclockwise rotation of a current in a superconductor. We will give two examples in this lecture; the first is the polarization of a photon, and the second is the spin of a spin-$\frac{1}{2}$ particle (such as an electron, a proton, or a neutron). Polarization is probably the example most people are familiar with, so we start with that.

You may be familiar with polarized sunglasses. A polarizing filter only lets vertically polarized (or horizontally polarized) photons pass through. Because sun glare tends to be horizontally polarized, if you use vertical polarizing filters as lenses, they screen out most the glare, but let through most of the rest of the light, and so reduce

glare. Of course, you can turn these sunglasses 45°, so they only let right diagonally polarized light through. How do we explain this in terms of the quantum state space?

It is the case that horizontal and vertical polarization are two distinguishable vectors. This means that other polarizations are linear combinations (called superpositions) of these.

Now, we need to introduce a little physics notation. Physicists like to represent iquantum states using a funny notation they call "kets", so a quantum state v would be represented as $|\, v\rangle$. In mathematics notations, kets are column vectors. So $|\leftrightarrow\rangle$ and $|\updownarrow\rangle$ are the horizontal and vertical polarizations of light.

Because we're doing quantum computation and working with qubits, we will represent these abstractly as $|\, 0\rangle$ and $|\, 1\rangle$, using the convention that $|\, 0\rangle = |\leftrightarrow\rangle$ and $|\, 1\rangle = |\updownarrow\rangle$. For example, if we have the vector

$$v = \begin{pmatrix} 1/\sqrt{2} \\ i/\sqrt{2} \end{pmatrix},$$

then

$$|\, v\rangle = \frac{1}{\sqrt{2}}\,|\, 0\rangle + \frac{i}{\sqrt{2}}\,|\, 1\rangle = \frac{1}{\sqrt{2}}\,(|\leftrightarrow\rangle + i\,|\updownarrow\rangle)$$

Other polarizations are linear combinations (called superpositions) of $|\leftrightarrow\rangle$ and $|\updownarrow\rangle$. For example,

$$\begin{aligned} |\nearrow\rangle &= \tfrac{1}{\sqrt{2}}\,(|\leftrightarrow\rangle + |\updownarrow\rangle) = \tfrac{1}{\sqrt{2}}\,(|\, 0\rangle + |\, 1\rangle)\,, \\ |\nwarrow\rangle &= \tfrac{1}{\sqrt{2}}\,(|\leftrightarrow\rangle - |\updownarrow\rangle) = \tfrac{1}{\sqrt{2}}\,(|\, 0\rangle - |\, 1\rangle)\,. \end{aligned}$$

We will be using the states $\frac{1}{\sqrt{2}}\,(|\, 0\rangle + |\, 1\rangle)$ and $\frac{1}{\sqrt{2}}\,(|\, 0\rangle - |\, 1\rangle)$ often enough that it helps to have special names for these; we call these $|+\rangle$ and $|-\rangle$ respectively.

Since the quantum state space is a complex vector space, we can also use complex coefficients in our superposition. What does this give us? It turns out that these give right and left circularly polarized light, $|\circlearrowright\rangle$ and $|\circlearrowleft\rangle$:

$$\begin{aligned} |\circlearrowright\rangle &= \tfrac{1}{\sqrt{2}}\,(|\leftrightarrow\rangle + i\,|\updownarrow\rangle)\,, \\ |\circlearrowleft\rangle &= \tfrac{1}{\sqrt{2}}\,(|\leftrightarrow\rangle - i\,|\updownarrow\rangle)\,. \end{aligned}$$

And if you use unequal coefficients in the above equations, you get elliptically polarized light.

Also note that the choice of whether $\frac{1}{\sqrt{2}}\,(|\leftrightarrow\rangle + |\updownarrow\rangle)$ is $|\nearrow\rangle$ or $|\nwarrow\rangle$ is purely a convention; I'll try to stick with the conventions that the textbook uses in these notes.

There is another half of the ket notation: a *row* vector is represented by a *bra:* $\langle\cdot\,|$. By convention, $\langle v\,|$ is the conjugate transpose of $|\, v\rangle$, so if

$$|\, v\rangle \;\text{ is the column vector }\; \begin{pmatrix} 1/\sqrt{2} \\ i/\sqrt{2} \end{pmatrix},$$

then

$$\langle v\,| \;\text{ is the row vector }\; \left(1/\sqrt{2}, -i/\sqrt{2}\right),$$

Together, the "bracket" $\langle v|w\rangle$ is the inner product of $\langle v \,|$ and $|\, w\rangle$. Since quantum states are unit vectors, and going from the ket to the bra involves a complex conjugation, we get

$$\langle \psi|\psi\rangle = 1$$

for any quantum state $|\,\psi\rangle$. The notation $\langle\phi|\psi\rangle$ represents the inner product of two states, and is called a *bracket* (and here you see the wordplay that Dirac used to come up with "bra" and "ket".

How can we change a quantum state? We will be doing quantum computation, so we need to know how to change the state of our quantum computer. Physics says that the evolution of an isolated quantum system is linear, and if we apply a linear transformation to a vector, we have a matrix.

Suppose we have a matrix M that operates on a quantum state $|\,\Psi\rangle$. If this transformation is something that can be implemented physically, the result of applying M to a quantum state $|\,\psi\rangle$, which is $M\,|\,\psi\rangle$, has to be a unit vector, so we need

$$\langle \psi \,|\, M^\dagger M \,|\, \psi\rangle = 1 \quad \text{for all quantum states } |\,\psi\rangle\,.$$

This implies that $M^\dagger M = I$, the identity matrix. This is the condition for M being a *unitary* matrix. Unitary matrices are essentially rotation matrices in complex space.

There are other conditions for a matrix U being unitary besides $U^\dagger U = I$. A matrix is unitary if and only if any of the following conditions hold:

- $U^\dagger U = I$,
- $UU^\dagger = I$,
- the columns of U are orthonormal vectors,
- the rows of U are orthonormal vectors.

There are three unitary matrices that operate on qubits (and thus are 2×2 which are very useful for quantum computing (and also for quantum physics in general). These are

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

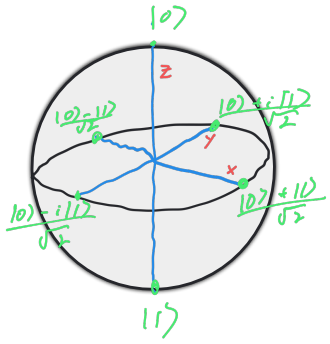
These are called the *Pauli matrices*. A silly mnemonic for remembering which entry has $-i$ in σ_y is that $-i$ is lighter than i, and thus floats to the top.

How do these matrices act on a qubit. Why are they labeled x, y, and z? To answer this, we will use a different representation of a qubit, namely, the spin of a spin $\frac{1}{2}$ particle. A spin $\frac{1}{2}$ particle, such as an electron, only has two completely distinguishable states of its spin, which we will take to be up and down. Of course, because of three-dimensional symmetry, you can prepare a spin $\frac{1}{2}$ particle with its spin axis pointing in any direction. However, these other directions are linear combinations of spin up and spin down. Namely, using the most common convention, Let us take up and down to be the z-axis, and left and right to be the x-axis, so the y-axis goes into and out of the

board. Then we will let $|0\rangle = |\uparrow\rangle$ and $|1\rangle = |\downarrow\rangle$. The standard convention is:

$$
\begin{aligned}
|\rightarrow\rangle &= \tfrac{1}{\sqrt{2}}\left(|\uparrow\rangle + |\downarrow\rangle\right), \\
|\leftarrow\rangle &= \tfrac{1}{\sqrt{2}}\left(|\uparrow\rangle - |\downarrow\rangle\right), \\
|\text{in}\rangle &= \tfrac{1}{\sqrt{2}}\left(|\uparrow\rangle + i\,|\downarrow\rangle\right), \\
|\text{out}\rangle &= \tfrac{1}{\sqrt{2}}\left(|\uparrow\rangle - i\,|\downarrow\rangle\right),
\end{aligned}
$$

We can represent these states on the Bloch sphere (See Figure.) Here, each point on the Bloch sphere represents a quantum state of a spin $\frac{1}{2}$ particle that that has spin in the corresponding direction.



Bloch Sphere

What happens when we apply $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$? One can see that it takes the vector $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ to $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and vice versa. It also takes the vector $|+\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ to itself, and $|-\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ to $\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix} = -|-\rangle$. Except for the fact that $|-\rangle$ is taken to $-|-\rangle$, this is a 180° rotation around the x-axis.

In fact, in some sense this really does represent a 180° rotation around the x-axis. This is because multiplying like a unit complex vector like -1, i, or $e^{i\theta}$ does not change the essential physical state — two quantum states $|\psi\rangle$ and $e^{i\theta}|\psi\rangle$ are indistinguishable by any quantum measurement. We will see this when we get to measurements next week. So the matrix σ_x represents a 180° rotation of the Bloch sphere around the x axis. Similarly, the matrices σ_y and σ_z represent rotations of the Bloch sphere around the y- and z-axes, respectively.

We will come back to the Bloch sphere later and tell you how to find the quantum state pointing in an arbitrary direction along the Bloch sphere, as well as the unitary matrix corresponding to an arbitrary rotation of the Bloch sphere.

Notes 8.370/18.435 Fall 2022

Lecture 3 Prof. Peter Shor

Last time, we started talking about quantum mechanics. We mentioned the principle that: *Isolated quantum systems evolve unitarily.* That is, for an isolated system (one that does not interact with its environment), there is a unitary matrix U_t such that if we let the system alone for time t, $|\,\Psi_t\rangle = U_t\,|\,\Psi_0\rangle$, where the initial state of the system is $|\,\Psi_0\rangle$ and ehe final state of the system is $|\,\Psi_t\rangle$.

Unitary matrices U are the complex analog of rotation matrices, also called orthogonal matrices; they take unit length complex vectors to unit length complex vectors. A matrix is unitary if and only if $U^\dagger U = I$. (An equivalent condition is $UU^\dagger = I$.) Here $U^\dagger$ is the complex conjugate transpose of U (also called the *Hermitian transpose*). We mentioned three specific unitary matrices last time, the Pauli matrices:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

We discussed the representation of spin-$\frac{1}{2}$ states on the Bloch sphere, and we looked at the actions of the Pauli matrices on the Bloch sphere, these being 180° rotations around the x-, y-, and z-axes, respectively. .

Today, we'll be talking about how unitary matrices arise in quantum mechanics, and then talk more about the Bloch sphere and rotations on the Bloch sphere.

So suppose you want to build a quantum computer, and you want to implement a unitary matrix U. What do you do? There's no magic incantation that takes the state directly from $|\,\psi\rangle$ to $U\,|\,\psi\rangle$. In fact, quantum unitary evolution can only changes quantum states continuously, and not in discrete jumps.

To explain how to implement a unitary gate, first I need to say something about quantum mechanics. Quantum mechanics assumes that isolated systems evolve according to Schödinger's equation,

$$i\hbar \frac{d}{dt}\,|\,\psi(t)\rangle = H\,|\,\psi(t)\rangle\,.$$

Here $\hbar$ is a very small physical constant, $|\,\psi(t)\rangle$ is the quantum state at time t, and H is a Hamiltonian—a Hermitian[1] matrix, to be exact, which can be associated with a Hermitian quadratic form that takes the quantum state as input and outputs its energy.

If we assume that H is constant, we can solve this equuation:

$$|\,\psi(t)\rangle = e^{-iHt/\hbar}\,|\,\psi(0)\rangle$$

Let's assume that the eigenvectors of H are $|\,\xi_1\rangle$, $|\,\xi_2\rangle$, ..., $|\,\xi_d\rangle$, with λ_i be the eigenvalue corresponding to $|\,\xi_i\rangle$. What the above equation means is that if the intial state $|\,\psi(0)\rangle$ is $\sum_i \alpha_i\,|\,\xi_i\rangle$, then

$$|\,\psi(t)\rangle = \sum_i e^{-\lambda_i t/\hbar}\alpha_i\,|\,\xi_i\rangle\,.$$

[1] A Hermitian matrix is one that satisfies $M = M^\dagger$.

We're not actually going to be using Schrödinger's equation until much later in the course. I'm introducing it now to give you some idea as to how you might implement unitary gates in practice, and to motivate the next thing I'll be talking about, which is three one-qubit gates which are rotations of the Bloch sphere by an angle θ. These are

$$\begin{aligned} R_x(\theta) &= e^{-i\theta\sigma_x/2}, \\ R_y(\theta) &= e^{-i\theta\sigma_y/2}, \\ R_z(\theta) &= e^{-i\theta\sigma_z/2}, \end{aligned}$$

What are these rotations? We have $R_z(\theta) = e^{-i\theta\sigma_z/2}$. To exponentiate a diagonal matrix, we can simply exponentiate each of the elements along the diagonal This gives

$$R_z(\theta) = e^{-i\theta\sigma_z/2} = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} = e^{-i\theta/2} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

You can check that if $\theta = \pi/2$, this is a rotation of a $\pi/2$ angle around the z-axis (multiplied by a global phase) and if $\theta = \pi$, this is a rotation of π around the z-axis (again multiplied by a global phase). So this gives us rotations around the z axis of an arbitrary angle.

To compute $R_y(\theta)$, we could diagonalize it to get $D = U^\dagger R_y(\theta) U$, and then exponentiate D to get $R_y(\theta) = U e^D U^\dagger$. We will compute it using a different method, to show how this method works. What we do is use a Taylor expansion.

$$e^{-i\theta\sigma_y/2} = I - i\frac{\theta}{2}\sigma_y - \frac{1}{2}\left(\frac{\theta}{2}\right)^2 \sigma_y^2 + i\frac{1}{3!}\left(\frac{\theta}{2}\right)^3 \sigma_y^3 + \frac{1}{4!}\left(\frac{\theta}{2}\right)^4 \sigma_y^4 - i\frac{1}{5!}\left(\frac{\theta}{2}\right)^5 \sigma_y^5 + \dots$$

Since $\sigma_y^2 = I$, we have

$$\begin{aligned} e^{-i\theta\sigma_y/2} &= I - i\frac{\theta}{2}\sigma_y - \frac{1}{2}\left(\frac{\theta}{2}\right)^2 I + i\frac{1}{3!}\left(\frac{\theta}{2}\right)^3 \sigma_y + \frac{1}{4!}\left(\frac{\theta}{2}\right)^4 I \dots \\ &= I\left(1 - \frac{1}{2}\left(\frac{\theta}{2}\right)^2 + \frac{1}{4!}\left(\frac{\theta}{2}\right)^4 - \dots\right) - i\sigma_y\left(\frac{\theta}{2} - \frac{1}{3!}\left(\frac{\theta}{2}\right)^3 + \frac{1}{5!}\left(\frac{\theta}{2}\right)^5 - \dots\right) \\ &= I\cos\frac{\theta}{2} - i\sigma_y \sin\frac{\theta}{2} \end{aligned}$$

But $-i\sigma_y = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, so we see that

$$e^{-i\theta\sigma_y/2} = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix},$$

which we see is a rotation in the $x - z$ plane. Looking at its action on the states of the Bloch sphere, it rotates the Bloch sphere around the y-axis by an angle of θ.

We can similarly see that

$$e^{-i\theta\sigma_x/2} = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix},$$

Now that we have $R_y(\theta)$ and $R_z(\theta)$, we can perform any one-qubit unitary by applying

$$R_z(\theta_3)R_y(\theta_2)R_z(\theta_1).$$

To see this, let us visualize what this does to the Bloch sphere. We first rotate the Bloch sphere by an angle of θ_1 around the north pole. The second rotation moves the north pole down to an arbitrary longitude. Finally, by applying $R_z(\theta_3)$, we can move the north pole to end up at an arbitrary latitude. Combined, these give an arbitrary rotation of the Bloch sphere. See the figure

Figure 1: Performing an arbitrary rotation by using three rotations, around the z-axis, the y-axis, and the z-axis.

We now explain one way to find the quantum state $|\phi_p\rangle$ corresponding to a point p on the Bloch sphere. There are other ways to do this, which lead to simpler expressions; we may revisit this question later in the term and explain them.

Suppose we have a point $p = (p_x, p_y, p_z)$ on the Bloch sphere $p_i \in \mathbb{R}$. Since it's on a unit sphere, we must have $p_x^2 + p_y^2 + p_z^2 = 1$. Let us consider the 2×2 matrix

$$M_p = p_x\sigma_x + p_y\sigma_y + p_z\sigma_z,$$

Since σ_x, σ_y, σ_z are Hermitian matrices, M_p must also be Hermitian, that is $M_p = M_p^\dagger$. Now,

$$M_p^2 = (p_x^2 + p_y^2 + p_z^2)I + 2p_xp_y(\sigma_x\sigma_y + \sigma_y\sigma_x) + 2p_yp_z(\sigma_y\sigma_y + \sigma_z\sigma_y) + 2p_xp_z(\sigma_x\sigma_z + \sigma_z\sigma_x) \quad = I.$$

where the last equality follows from the facts that the vector p has length 1 and $\sigma_a\sigma_b = -\sigma_b\sigma_z$ if $a \neq b$.

We can also show that

$$\mathrm{Tr}M_p = p_x\mathrm{Tr}\sigma_x + p_y\mathrm{Tr}\sigma_y + p_z\mathrm{Tr}\sigma_z = 0,$$

since the Pauii matrices all have trace 0. Since $M_p^2 = I$, its eigenvalues have to be ± 1. And since its trace is 0, one of its eigenvalues has to be 1 and the other has to be -1. Now, we will let the eigenvector with eigenvalue $+1$ be the corresponding quantum state $|\psi_p\rangle$ to point p on the unit sphere. We already know this holds for the

unit vectors on the x, y, and z axes. Note also that since M_p is a Hermitian matrix its two eigenvectors are orthogonal, and since $M_p = -M_{-p}$, the -1 eigenvector of M_p is the point antipodal to $|\,\phi_p\rangle$.

Let's illustrate this by an example. Let $p = \left(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right)$. Then

$$M_p = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Now, M_p has eigenvalues 1 and -1, and its eigenvectors are

$$\frac{\sqrt{2+\sqrt{2}}}{2}\,|\,0\rangle + \frac{\sqrt{2-\sqrt{2}}}{2}\,|\,1\rangle \quad \text{and} \quad -\frac{\sqrt{2-\sqrt{2}}}{2}\,|\,0\rangle + \frac{\sqrt{2+\sqrt{2}}}{2}\,|\,1\rangle,$$

where the first eigenvector has eigenvalue $+1$ and the second -1.

These vectors are in fact just

$$\cos\frac{\pi}{8}\,|\,0\rangle + \sin\frac{\pi}{8}\,|\,1\rangle \quad \text{and} \quad -\sin\frac{\pi}{8}\,|\,-\rangle + \cos\frac{\pi}{8}\,|\,1\rangle.$$

This makes sence, because the point $p = \frac{1}{\sqrt{2}}(1,0,1)$ is halfway between $p = (1,0,0)$ and $p = (0,0,1)$, and these points correspond to $\cos\frac{\pi}{4}\,|\,0\rangle + \sin\frac{\pi}{4}\,|\,1\rangle$ and $\cos 0\,|\,0\rangle + \sin 0\,|\,1\rangle$, respectively.

Notes 8.370/18.435 Fall 2022

Lecture 4 Prof. Peter Shor

Last time, we started talking about quantum mechanics. We mentioned the principle that *Isolated quantum systems evolve unitarily.* That is, for an isolated system, there is some unitary matrix U_t that takes the state of the system at time $-$, $|\,\phi(0)\rangle$ to the state of the system at time t, ie. $|\,\psi(t)\rangle = U_t\,|\,\Psi(0)\rangle$.

There is another operation we need to know for quantum computation, *measurement*. We have already seem an example of a von Neumann measurement (also called a *projective measurement*). Recall that we said if we have a quantum state $\alpha\,|\,A\rangle + \beta\,|\,B\rangle$, where the $|\,A\rangle$ and $|\,B\rangle$ are completely distinguishable states, then if we apply the experiment that distinguishes $|\,A\rangle$ from $|\,B\rangle$, then we see $|\,A\rangle$ with probability $|\alpha|^2$ and $|\,B\rangle$ with probability $|\beta|^2$. This is probably the simplest example of the following definition:

Definition:

A *complete projective measurement* corresponds to a basis of the system. Suppose $|\,v_1\rangle$, $|\,v_2\rangle$, $\ldots$ $|\,v_d\rangle$ is an orthonormal basis for a quantum system. Then, if the system is in state $|\,\psi\rangle$, the von Neumann measurement corresponding to this basis yields $|\,v_i\rangle$ with probability $|\langle v_i|\psi\rangle|^2$, where $\langle v|w\rangle$ is the inner product of the row vector $\langle v\,|$ and the column vector $|\,w\rangle$. Remember also that when you go from $|\,v\rangle$ to $\langle v\,|$, you take the complex conjugate of v, so for a unit vector $|\,v\rangle$, $\langle v|v\rangle = 1$. this is called a *projective measurement* because the quantum state $|\,\psi\rangle$ is projected onto one of the basis vectors $|\,v_i\rangle$: theoretically, after the measurement, the quantum state is $|\,v_i\rangle$. (In practice, sometimes the measurement destroys or alters the quantum state.)

We now show that the probabilities of all the outcomes of a measurement sum to 1. The sum of the probabilities of the outcomes is

$$\sum_i |\langle v_i|\psi\rangle|^2 = \sum_i \langle\psi|v_i\rangle\langle v_i|\psi\rangle.$$

This equality follows from the fact that $\langle\psi|v_i\rangle = \langle v_i|\psi\rangle^*$, so when you multiply them together, you get the real value $|\langle v_i|\psi\rangle|^2$. Now,

$$\sum_i \langle\psi|v_i\rangle\langle v_i|\psi\rangle = \langle\psi\,|\left(\sum_i |\,v_i\rangle\langle v_i\,|\right)|\,\psi\rangle$$

because we can move the sum inside the bracket. Now, since $\{|\,v_i\rangle\}$ forms an orthonormal basis, $\sum_i |\,v_i\rangle\langle v_i\,| = I$ (you will prove this on the homework). So we have

$$\langle\psi\,|\left(\sum_i |\,v_i\rangle\langle v_i\,|\right)|\,\psi\rangle = \langle\psi|\psi\rangle = 1.$$

Maybe I should say a little more about $|\,v\rangle\langle v\,|$. This is just the column vector $|\,v\rangle$ multiplied by the row vector $\langle v\,|$. So if $|\,v\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, then

$$|\,v\rangle\langle v\,| = \begin{pmatrix} 1 \\ 0 \end{pmatrix}(1,0) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}.$$

Let's now give an example. Suppose we have the quantum state

$$|\,\psi\rangle = \frac{4}{5}\,|\,0\rangle + \frac{3}{5}\,|\,1\rangle\,.$$

What is the probability of observing each of the outcomes if we measure it in the basis $\{|\,+\rangle, |\,-\rangle\}$?

The probability of seeing $|\,+\rangle$ is

$$\begin{aligned}|\langle +|\psi\rangle|^2 &= \left|\tfrac{1}{\sqrt{2}}(\langle 0\,| + \langle 1\,|)(\tfrac{4}{5}\,|\,0\rangle + \tfrac{3}{5}\,|\,1\rangle)\right|^2 \\ &= \left|\tfrac{4}{5\sqrt{2}} + \tfrac{3}{5\sqrt{2}}\right|^2 = \tfrac{49}{50}\end{aligned}$$

and similarly, the probability of seeing $|\,-\rangle$ is

$$\begin{aligned}|\langle -|\psi\rangle|^2 &= \left|\tfrac{1}{\sqrt{2}}(\langle 0\,| - \langle 1\,|)(\tfrac{4}{5}\,|\,0\rangle - \tfrac{3}{5}\,|\,1\rangle)\right|^2 \\ &= \left|\tfrac{4}{5\sqrt{2}} - \tfrac{3}{5\sqrt{2}}\right|^2 = \tfrac{1}{50},\end{aligned}$$

So the chance of the result $|\,+\rangle$ is $\frac{49}{50}$ while the chance of the result $|\,-\rangle$ is $\frac{1}{50}$.

This shouldn't be surprising; the vector $(0.8, 0.6)$ is nearly parallel with the vector $|\,+\rangle = \frac{1}{\sqrt{2}}(1,1)$, so when you project $|\,\psi\rangle$ onto $|\,+\rangle$, you get a large projection, while for $|\,-\rangle$, you get a small projection.

There is a more general kind of projective measurement, where the outcomes correspond to subspaces rather than quantum states. To specify this kind of measurement, you need to give a set of subspaces S_1, S_2, ..., S_k. Let Π_i be the projection matrix onto the ith subspace. In other words, if S_i has $v_i^{(1)}$, $v_i^{(2)}$ ... $v_i^{(\ell_i)}$ as an orthogonal basis, then

$$\Pi_i = \sum_{j=1}^{\ell_j} \left|\, v_i^{(j)} \right\rangle\!\!\left\langle v_i^{(j)} \,\right|.$$

This set of subspaces must satisfy the conditions:

$$\Pi_i \Pi_j = 0 \quad \text{if} \quad i \neq j,$$

i.e., the subspaces must be orthogonal, and

$$\sum_i \Pi_i = I,$$

i.e., the subspaces must span the entire quantum state space.

When we apply this measurement to a quantum state $|\,\psi\rangle$, the probability of seeing the i'th outcome is

$$P(i) = \langle \psi\,|\,\Pi_i\,|\,\psi\rangle\,,$$

and the state of the system after the measurement is

$$\frac{\Pi_i\,|\,\psi\rangle}{\langle \psi\,|\,\Pi_i\,|\,\psi\rangle^{1/2}}\,,$$

where we have normalized the projection onto S_i so as to make it a unit vector.

We now give an example. Let's take a four-dimensional quantum system with basis states $|\,0\rangle$, $|\,1\rangle$, $|\,2\rangle$ and $|\,3\rangle$. We want the measurement that asks the question: is the state $|\,0\rangle$ or $|\,1\rangle$, as opposed to being $|\,2\rangle$ or $|\,3\rangle$. That is, we want to the subspace corresponding to the projection matirx $\Pi_A = |\,0\rangle\langle 0\,| + |\,1\rangle\langle 1\,|$ or the projection matrix $\Pi_B = |\,2\rangle\langle 2\,| + |\,3\rangle\langle 3\,|$.

We have

$$\Pi_A + \Pi_B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = I$$

and

$$\Pi_A \Pi_B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = 0,$$

so these projection matrices satisfy the conditions needed for them to be a projective measurement.

Let's measure the state $|\,\psi\rangle = \frac{5}{10}\,|\,0\rangle + \frac{7}{10}\,|\,1\rangle + \frac{5}{10}\,|\,2\rangle + \frac{1}{10}\,|\,3\rangle$ with this measurement.

We have

$$\Pi_A\,|\,\psi\rangle = \tfrac{5}{10}\,|\,0\rangle + \tfrac{7}{10}\,|\,1\rangle$$

This gives a probability of $.5^2 + .7^2 = .74$ for this outcome, and after this outcome, the state

$$\frac{1}{\sqrt{.74}}\left(\tfrac{5}{10}\,|\,0\rangle + \tfrac{7}{10}\,|\,1\rangle\right)$$

Similarly,

$$\Pi_B\,|\,\psi\rangle = \tfrac{5}{10}\,|\,2\rangle + \tfrac{1}{10}\,|\,3\rangle.$$

This gives a probability of $.5^2 + .1^2 = .26$ for this outcome, and after this outcome, the state

$$\frac{1}{\sqrt{.26}}\left(\tfrac{5}{10}\,|\,2\rangle + \tfrac{1}{10}\,|\,3\rangle\right)$$

I should probably tell you that there are types of measurements that are more general than these projective measurements, called POVM measurements. We will not discuss them in this class (except maybe in a homework problem or two), because we actually won't need them in this course. You can learn about them by reading the textbook, or taking the follow-up course 8.371/18.436.

There is another way of describing mesurement in quantum mechanics, which is to use *observables*.

Recall from linear algebra that a symmetric matrix M is one for which $M^{\mathrm{T}} = M$, where M^{T} is the transpose of M. A *Hermitian* matrix is one for which $M^\dagger = M$, where $M^\dagger$ is the Hermitian transpose of M. The *The Hermitian transpose* is the

conjugate transpose: you take the transpose of a matrix and you take the complex conjugates of each of its elements.

You can aways diagonalize a Hermitian matrix; if M is Hermitian, then there is a unitary matrix U such that $U^\dagger M U = D$, where D is a diagonal matrix with real entries.

An *observable* is a Hermitian matrix M on the state space of the quantum system. Each of the eigenvalues of M will be associated with a subspace of eigenvectors with that eigenvalue. The subspaces are orthogonal, and span the state space, and thus can be associated with a measurement. For this observable, when the eigenspace associated with the eigenvaue λ_i of M is observed, we say that we have observable takes the value λ_i.

Before we go into the theory more, let's do an example. We will use the observable

$$M = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

for a qubit. This matrix has two eigenvectors,

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \text{ with eigenvalue 3, \quad and \quad } \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \text{ with eigenvalue 1.}$$

This means we can express M as a linear combination of projectors:

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = 3 \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} + \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{pmatrix} = 3 \left| + \right\rangle\!\left\langle + \right| + \left| - \right\rangle\!\left\langle - \right|.$$

One of the useful properties of the observable formulation is that we can easily calculate the expectation value of an observable applied to a quantum state.

Theorem 1 *Given a state $|\,\psi\rangle$ and an observable H, the expectation value of the measurement applied to H is*

$$\text{Expectation} = \langle \psi \,|\, H \,|\, \psi \rangle .$$

Before we prove this theorem, let's do an example. Suppose we have the state $|\,\psi\rangle = \frac{3}{5}\,|\,0\rangle + \frac{4}{5}\,|\,1\rangle$. The observable $M = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ measures it in the basis $\{|\,+\rangle, |\,-\rangle\}$, and returns the value 3 for the outcome $|\,+\rangle$ and the value 1 for the outcome $|\,-\rangle$.

The theorem gives

$$\langle \psi \,|\, M \,|\, \psi \rangle = \left(\frac{3}{5}, \frac{4}{5}\right) \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \frac{3}{5} \\ \frac{4}{5} \end{pmatrix} = \frac{74}{25}$$

for the expectation vaue of the observable.

Now, the probability of the outcome $|\,+\rangle$ is $|\langle + | \psi \rangle|^2 = \frac{49}{50}$ and the probability of the outcome $|\,-\rangle$ is $|\langle - | \psi \rangle|^2 = \frac{1}{50}$. Thus, the expected value of the observable is

$$\frac{49}{50} \cdot 3 + \frac{1}{50} \cdot 1 = \frac{74}{25}.$$

We now prove the theorem. Suppose the eigenvalues of an observable H are $\lambda_1, \lambda_2, \ldots, \lambda_j$, and the corresponding eigenspaces are $S_1, S_2, \ldots, S_j$. Suppose further that Π_i is the orthogonal projector onto S_i. Let us now apply this observable to a quantum state $|\,\psi\rangle$. The outcome of the measurement is i with probability $\langle\psi\,|\,\Pi_i\,|\,\psi\rangle$. Thus, the expectation of the observable is

$$\begin{aligned}\sum_i \langle\psi\,|\,\Pi_i\,|\,\psi\rangle\,\lambda_i &= \langle\psi\,|\left(\sum_i \lambda_i \Pi_i\right)|\,\psi\rangle \\ &= \langle\psi\,|\,H\,|\,\psi\rangle\,,\end{aligned}$$

where the first equality holds because matrix multiplication is linear, and the second because $H = \sum_i \lambda_i \Pi_i$. This proves the theorem.

Notes 8.370/18.435 Fall 2022

Lecture 5 Prof. Peter Shor

In the previous few lectures, we saw how to operate on one qubit—or more generally, one d-dimensional quantum system. We studied how to transform quantum systems with a unitary matrix, and how to measure them via a complete set of projection matrices. In this lecture, we explain how the state space of a joint quantum system, composed of two individual quantum systems, is constructed.

Suppose we have two quantum systems. Each of these has a state space which is a complex vector space of dimensions d_1 and d_2, respectively. When we consider these two quantum systems together, we get a state space which is the tensor product of their individual state spaces, and which has dimension $d_1 d_2$.

Some of you probably haven't see tensor products before. We will show how it works by exammple. If you have two qubits, each of which has a state space with basis $\{|\,0\rangle\,, |\,1\rangle\}$, then the state space of the joint system has basis

$$\{|\,0\rangle \otimes |\,0\rangle\,, |\,0\rangle \otimes |\,1\rangle\,, |\,1\rangle \otimes |\,0\rangle\,, |\,1\rangle \otimes |\,1\rangle\}.$$

The convention is to write the basis in lexicographical order. Thus, if you have two qubits in state $\alpha_0 |\,0\rangle + \alpha_1 |\,1\rangle$ and $\beta_0 |\,0\rangle + \beta_1 |\,1\rangle$, the system of both qubits is in state

$$\alpha_0\beta_0 |\,0\rangle \otimes |\,0\rangle + \alpha_0\beta_1 |\,0\rangle \otimes |\,1\rangle + \alpha_1\beta_0 |\,1\rangle \otimes |\,0\rangle + \alpha_1\beta_1 |\,1\rangle \otimes |\,1\rangle\,,$$

so the standard distributed law applies to tensor products. Often, rather than writing $|\,0\rangle \otimes |\,1\rangle$, we will write this as $|\,01\rangle$.

To illustrate tensor products using a more usual vector notation,

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix}$$

What if you have n qubits? The state space has dimension 2^n, and has basis $|\,00\ldots00\rangle$, $|\,00\ldots01\rangle$, $|\,00\ldots10\rangle$, ... (Recall that by notation, $|\,00\ldots00\rangle = |\,0\rangle \otimes |\,0\rangle \otimes \ldots \otimes |\,0\rangle \otimes |\,0\rangle$.) As a convention, we list the basis states in lexicographical order.

Now let us count up (real) degrees of freedom. A qubit has a state $\alpha |\,0\rangle + \beta |\,1\rangle$, and since α and β are complex numbers, this would give four real degrees of freedom. However, it also satisfies $i|\alpha|^2 + |\beta|^2 = 1$, which means that it only has three real degrees of freedom (if we use the fact that multiplying by a global phase leaves the state essentially unchanged, there are only two degrees of freedom). However, the joint state of two qubits has four complex coefficients, leaving 7 (or 6) degrees of freedom. Thus, because $7 > 2 \cdot 3$, there are some states of the joint system which cannot be the tensor product of states of the individual systems. These states are called *entangled*. In fact, because the number of degrees of freedom of tensor product states is less than that of entangled states, the tensor product states form a lower-dimensional manifold in the entangled states, and we see that most quantum states are entangled.

For example, the state

$$\frac{1}{\sqrt{3}}\,|\,00\rangle+\frac{1}{\sqrt{3}}\,|\,01\rangle+\frac{1}{\sqrt{6}}\,|\,10\rangle+\frac{1}{\sqrt{6}}\,|\,11\rangle=\left(\frac{\sqrt{2}}{\sqrt{3}}\,|\,0\rangle+\frac{1}{\sqrt{3}}\,|\,1\rangle\right)\left(\frac{1}{\sqrt{2}}\,|\,0\rangle+\frac{1}{\sqrt{2}}\,|\,1\rangle\right)$$

is a tensor product state, while the state

$$\frac{1}{\sqrt{3}}\,|\,00\rangle+\frac{1}{\sqrt{3}}\,|\,01\rangle+\frac{1}{\sqrt{3}}\,|\,10\rangle$$

is entangled You can see that the second one is not a tensor product state, and thus entangled, because if we could represent it as a tensor product state

$$\alpha_0\beta_0\,|\,00\rangle+\alpha_0\beta_1\,|\,01\rangle+\alpha_1\beta_0\,|\,10\rangle+\alpha_1\beta_1\,|\,11\rangle\,,$$

then we would need $\alpha_1\beta_1 = 0$. But if that holds, either α_1 or β_1 is 0, so one of $\alpha_0\beta_1$ and $\alpha_1\beta_0$ must be 0, which is not the case.

What about unitary transformations on a joint state space. Suppose we have two unitary matrices U and V, of dimensions k and ℓ. The tensor product of them is the $k\ell \times k\ell$ matrix

$$U\otimes V=\begin{pmatrix} u_{11}V & u_{12}V & \dots & u_{1k}V \\ u_{21}V & u_{22}V & \dots & u_{2k}V \\ \vdots & \vdots & & \vdots \\ u_{k1}V & u_{k2}V & \dots & u_{kk}V \end{pmatrix},$$

where $u_{ij}V$ is the (i,j) entry of U multiplied by V.

Let's do an example and figure out what $\sigma_x \otimes \sigma_z$ is. We have

$$\begin{aligned}\sigma_x\otimes\sigma_z &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\otimes\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\ &= \left(\begin{array}{cc|cc} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ \hline 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{array}\right)\end{aligned}$$

If the input is in a tensor product state $|\,\phi_1\rangle\otimes|\,\phi_2\rangle$, and you apply a tensor product unitary $U_1\otimes U_2$, then the output is also in a tensor product state, namely $U_1\,|\,\phi_1\rangle\otimes U_2\,|\,\phi_2\rangle$. So to get an entangled output from a tensor product input, you need to apply a unitary that is not a tensor product. One non-tensor-product unitary that we will be using extensively in this class is the CNOT.

We have

$$\begin{aligned}CNOT &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ &= |\,0\rangle\langle 0\,|\otimes I + |\,1\rangle\langle 1\,|\otimes\sigma_x\end{aligned}$$

The last description says that if the first qubit is a $|0\rangle$, then we apply an identity to the second qubit, and if the first qubit is a $|1\rangle$, we apply a σ_x (or NOT gate) to the second qubit. This is why it's called a controlled NOT — depending on the state of the first qubit, we apply a NOT gate to the second one (or we don't).

Note that we don't measure the first qubit — we don't get a classical record of the state of the first qubit, and a state that was in a superposition of the first qubit being $|0\rangle$ and$|1\rangle$ remains in a superposition. Let's do an example. Suppose we start with the state $|+\rangle|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$. When we apply the CNOT gate, the $|00\rangle$ remains unchanged, because the first qubit is $|0\rangle$, but the $|10\rangle$ becomes $|11\rangle$. This means

$$\text{CNOT}\,|+\rangle|0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$

so we have used the CNOT gate to entangle two qubits.

Notes 8.370/18.435 Fall 2022

Lecture 6 Prof. Peter Shor

In the last class, you saw how the tensor product of quantum states and the tensor product of unitary operators worked. Today, we'll talk about the tensor product of measurements.

Recall that a von Neumann measurement on a quantum state space A was a set of Hermitian projectors $\Pi_1, \Pi_2, \ldots, \Pi_k$ such that $\Pi_i \Pi_j = 0$ if $i \neq j$ and $\sum_{i=1}^k \Pi_i = I_A$.

Suppose you have two individual von Neumann measurements on systems A and B, $\{\Pi_1^A, \Pi_2^A, \ldots, \Pi_k^A\}$ and $\{\Pi_1^B, \Pi_2^B, \ldots, \Pi_\ell^B\}$. Then there is a tensor product measurement on $A \otimes B$ given by

$$\{\Pi_i^A \otimes \Pi_j^B | 1 \leqslant i \leqslant k, 1 \leqslant j \leqslant \ell\}.$$

It is not difficult to show that this set of projectors satisfies the conditions to be a von Neumann measurement.

One very common tensor product measurement is when we just make a measurement on system A and leave system B unchanged. This operation can be expressed in the above formulation by letting the measurement on B be the single projector I_B (which does nothing to the quantum state on B). Thus if we have a basis for a qubit $|\, v\rangle$, $|\, \bar{v}\rangle$ on systeam A, measurement operators for system AB are $|\, v\rangle_A \langle v\,| \otimes I_B$ and $|\, \bar{v}\rangle_A \langle \bar{v}\,| \otimes I_B$.

Let's do an example. Suppose we have two qubits in the joint state

$$|\, \psi\rangle = \alpha\, |\, 00\rangle + \beta\, |\, 01\rangle + \gamma\, |\, 10\rangle$$

and we measure the first qubit in the $\{0, 1\}$ basis. What happens?

The projection matrices associated with this measurement are

$$|\, 0\rangle\langle 0\,| \otimes I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad |\, 1\rangle\langle 1\,| \otimes I = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

What happens? The probability of getting the first outcome is

$$\left|\left(|\, 0\rangle\langle 0\,| \otimes I\right) |\, \psi\rangle\right|^2 = \left| \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ 0 \end{pmatrix} \right|^2 = \left| \begin{pmatrix} \alpha \\ \beta \\ 0 \\ 0 \end{pmatrix} \right|^2 = |\alpha|^2 + |\beta|^2$$

and the resulting state is $\frac{1}{\sqrt{|\alpha|^2+|\beta|^2}}(\alpha\, |\, 00\rangle + \beta\, |\, 01\rangle)$. Since we measured the first qubit, we know the state of the first qubit, so sometimes (by an abuse of notation or something) we just take the resulting state to be the state on the second qubit, $\frac{1}{\sqrt{|\alpha|^2+|\beta|^2}}(\alpha\, |\, 0\rangle + \beta\, |\, 1\rangle)$. Similarly, the probability of getting the second outcome is $|\gamma|^2$, and the resuting state of the second qubit is $|\, 0\rangle$.

There is a different way of doing these calculations. Here, we need to be careful about which qubits the kets and bras apply to, so I will write them explicitly in the following.

We can think of measuring the first qubit as applying either the bra (row vector) $\langle 0 \,|$ or $\langle 1 \,|$ to the first qubit. We will make this explicit by labeling the qubits A and B. Thus, when we measure the first qubit, we get one of the following two (unnormalized) states:

$$
\begin{aligned}
{}_A\langle 0 \,| \left(\alpha \,|\, 00 \rangle_{AB} + \beta \,|\, 01 \rangle_{AB} + \gamma \,|\, 00 \rangle_{AB} \right) &= \alpha \,|\, 0 \rangle_B + \beta \,|\, 1 \rangle_B \\
{}_A\langle 1 \,| \left(\alpha \,|\, 00 \rangle_{AB} + \beta \,|\, 01 \rangle_{AB} + \gamma \,|\, 00 \rangle_{AB} \right) &= \gamma \,|\, 0 \rangle_B
\end{aligned}
$$

How do these calculations work? We have ${}_A\langle 0|00\rangle_{AB} = |\, 0\rangle_B$ and ${}_A\langle 0|10\rangle_{AB} = 0$. So applying ${}_A\langle \cdot \,|$ to $|\, \cdot \rangle_{AB}$ results in an unnormalized quantum state vector on qubit B. The square of length of this vector gives the probability of that measurement outcome, and the normalized vector gives the resulting quantum state.

We now talk about the tensor product of observables. Suppose we have two observables M_A on quantum system A and M_B on quantum system B. Recall that if we have an observable M applied to a quantum state $|\, \psi \rangle$, it returns a value, and that $\langle \psi \,|\, M \,|\, \psi \rangle$ is the expectation of the value.

How do we get the observable for the product of the values? We use $M_A \otimes M_B$. How about for the sum of the values? We use $M_A \otimes I_B + I_A \otimes M_B$, where I_A and I_B are the identity matrices on systems A and B, respectively.

Let's take an example. The angular momentum observable for a spin-$\frac{1}{2}$ particle is $\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{-1}{2} \end{pmatrix}$. Suppose we have two particles, A and B. Then

$$
J_A \otimes J_B = \begin{pmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & -\frac{1}{4} & 0 & 0 \\ 0 & 0 & -\frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{pmatrix}
$$

$$
J_A \otimes I_b + I_A \otimes J_B = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix} + \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}
$$

Now, let's look at the state $\frac{1}{\sqrt{2}}(|\, 01\rangle - |\, 10\rangle)$. I claim that if you measure both qubits in the same basis, no matter which basis it is, the two qubits will be different, i.e., they will be orthogonal. What this means physically is that if you have the state $\frac{1}{\sqrt{2}}(|\uparrow\rangle|\downarrow\rangle - |\downarrow\rangle|\uparrow\rangle)$, and you measure them along any axis, you will find that they are always spinning in opposite directions; a physical way of understanding this is that the total spin of the state $\frac{1}{\sqrt{2}}(|\uparrow\rangle|\downarrow\rangle - |\downarrow\rangle|\uparrow\rangle)$ is 0, so when you meaaure each qubit, you end up with a total spin of 0.

Let's do the computation. Suppose we measure the first particle in the basis

$$
\{\cos(\theta) \,|\, 0\rangle + e^{i\phi} \sin(\theta) \,|\, 1\rangle \,, \sin(\theta) \,|\, 0\rangle + e^{-i\phi} \cos(\theta) \,|\, 1\rangle\}.
$$

(It is straightforward to check that these two states are orthogonal and thus form a

basis.) Then

$$\frac{1}{\sqrt{2}}\Big(\cos(\theta)\langle 0\,| + e^{-i\phi}\sin(\theta)\langle 1\,|\Big)(|\,01\rangle - |\,10\rangle) = \frac{1}{\sqrt{2}}\cos(\theta)\,|\,1\rangle - e^{-i\phi}\sin(\theta)\,|\,0\rangle\,.$$

This is indeed the orthogonal state, so we have that if the two qubits are measured along the same axis, the two states are orthogonal.

Now, we know enough to explain a variation of the EPR thought experiment. Suppose Alice and Bob have an entangled pair of qubits in the state $\frac{1}{\sqrt{2}}(|\,01\rangle - |\,10\rangle)$ (called an *EPR pair*). They can separate them far enough so that they measure them simultaneously, so that the speed of light prevents Alice's particle from communicating the basis its being measured in to Bob's particle.

Now, Alice and Bob measure their particle in one of the four following bases:

$$\begin{aligned}
&\{ \quad |\,f_1\rangle = |\,0\rangle, && |\,f_2\rangle = |\,1\rangle && \} \\
&\{ \quad |\,g_1\rangle = \cos\frac{\pi}{8}\,|\,0\rangle + \sin\frac{\pi}{8}\,|\,1\rangle, && |\,g_2\rangle = -\sin\frac{\pi}{8}\,|\,0\rangle + \cos\frac{\pi}{8}\,|\,1\rangle && \} \\
&\{ \quad |\,h_1\rangle = \cos\frac{\pi}{4}\,|\,0\rangle + \sin\frac{\pi}{4}\,|\,1\rangle, && |\,h_2\rangle = -\sin\frac{\pi}{4}\,|\,0\rangle + \cos\frac{\pi}{4}\,|\,1\rangle && \} \\
&\{ \quad |\,j_1\rangle = \cos\frac{3\pi}{8}\,|\,0\rangle + \sin\frac{3\pi}{8}\,|\,1\rangle, && |\,j_2\rangle = -\sin\frac{3\pi}{8}\,|\,0\rangle + \cos\frac{3\pi}{8}\,|\,1\rangle && \}
\end{aligned}$$

Let's assume that each of Alice and Bob's EPR pairs has a table of outcomes that it will yield for each particle for all these three measurements. We are assuming a deterministic process, but you can show that the same argument applies to probilistic processes. The table will have to look something like this.

Basis F		Basis G		Basis H		Basis J	
A	B	A	B	A	B	A	B
f_1	f_2	g_1	g_2	h_2	h_1	j_2	j_1
f_1	f_2	g_1	g_2	h_1	h_2	j_1	j_2
f_2	f_1	g_2	g_1	h_2	h_1	j_2	j_1
f_2	f_1	g_2	g_1	h_2	h_1	j_1	j_2
f_1	f_2	g_2	g_1	h_2	h_1	j_2	j_1
f_1	f_2	g_1	g_2	h_1	h_2	j_1	j_2

When Alice and Bob measure in the same basis, they must always get opposite outcomes. If Alice measures f_1 in basis F, then Bob must always measure f_2 in basis F. Thus, we can assume that Bob's state is f_2. Now, suppose Bob measures in basis G, then because $\langle f_2|g_2\rangle^2 = \cos^2\frac{\pi}{8} \approx 0.85$, Bob must get g_2 with probability 0.85, so the first two columns must match (i.e., have (f_1, f_2, g_1, g_2) or (f_2, f_1, g_2, g_1)) in around 0.85 of their entries. Similarly, the second and third columns must match in around 0.85 of their entries, and the third and fourth columns must match in around 0.85 of their entries. This means that the first and fourth columns must match in at least $1 - 3 * 0.15 \approx 0.55$ of their entries. However, $\langle f_1|j_2\rangle^2 = \sin^2\frac{\pi}{8} \approx 0.15$, meaning that the first and fourth columns can only match in 0.15 of their entries, a contradiction.

So what does this mean for physical reality? Physicists (and philosophers) have been arguing about this for decades. Is the universe non-local (so the decision as to which basis Alice measures in transmitted faster than light to Bob), or is something

else going on? I'm going to leave that question to philosophers. One thing that is true is that even if some information is transmitted faster than light, it's not very useful information. You cannot use entanglement to transmit a message faster than light (although you can use it to do some non-intuitive things, as we will see later in the course).

Notes 8.370/18.435 Fall 2022

Lecture 7 Prof. Peter Shor

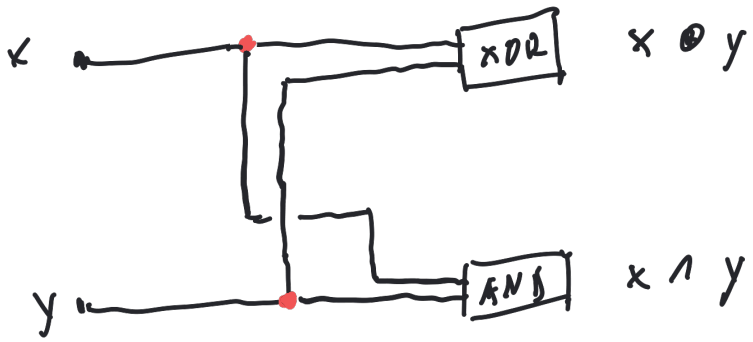
Today's lecture will be on classical Boolean circuits.

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:

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

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:

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

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):

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

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:

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

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:

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

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:

$$\text{Toffoli} = \begin{array}{c|c} \text{in} & \text{out} \\ \hline 000 & 000 \\ 001 & 001 \\ 010 & 010 \\ 011 & 011 \\ 100 & 100 \\ 101 & 101 \\ 110 & 111 \\ 111 & 110 \end{array} .$$

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.

Notes 8.370/18.435 Fall 2022

Lecture 8 Prof. Peter Shor

Today we continue discussing classical Boolean circuits.

Quantum mechanics is reversible. Because of this, we will study quantum circuits that are made up from reversible quantum gates. We thus first need to study classical reversible circuits. Reversible circuits are made from reversible gates, both to shed light on reversible quantum circuits, and because we will be using reversible classical circuits as subroutines in our quantum algorithms. Last time we saw that any two-bit reversible gates can be made from NOT, CNOT, and SWAP gates (in fact, we don't need the SWAP gate). While two-bit classical gates are sufficient for universal classical computation, two-bit reversible classical gates are not. We will see this later in this class.

There are, however, three-bit gates that are universal. One of these is the Toffoli gate (discovered by Tom Toffoli, who worked at MIT at the time). This gate negates the third bit if the first two are both 1. It can thus be described as

$$\begin{aligned} x' &= x \\ y' &= y \\ z' &= z \oplus (x \wedge y), \end{aligned}$$

where $\oplus$ is binary addition, or XOR. Its truth table is

variables	
x, y (in)	x', y' (out)
0,0,0	0,0,0
0,1,0	0,1,0
1,0,0	1,0,0
1,1,0	1,1,1
0,0,1	0,0,1
0,1,1	0,1,1
1,0,1	1,0,1
1,1,1	1,1,0

We show that together with the NOT gate, the Toffoli gate is universal for reversible computation. How do we prove this? What we do is that we show that the Toffoli gate can simulate AND, and FANOUT gates. More specifically,

$$\begin{aligned} \text{AND}: \quad T(x, y, 0) &= (0, 0, x \wedge y) \\ \text{FANOUT}: \quad T(1, x, 0) &= (1, x, x) \end{aligned}$$

The OR gate can be composed of AND and NOT gates. You can check that $x \vee y = \neg(\neg x \wedge \neg y)$. Thus, since Toffoli gates can generate AND, OR and FANOUT gates, together with NOT gates, they can be used to perform any computation.

In fact, if there is a source of bits that are set to 1, Toffoli gates can simuate NOT gates as well

$$T(1, 1, x) = T(1, 1, \neg x).$$

But if you don't start with any 1 bits, or with just one 1 bit, Toffoli gates cannot create any new 1 bits. So all we really need the NOT gates for was to generate two 1 bits.

Note that using Toffoli gates as we did above for arbitrary computations generates a lot of extra bits which are not part of the output. For example, to compute an AND, we start with one extra bit initialized to the 0 value, and end up with two extra copies of our input bits that aren't generated by a classical AND gate. Thus, reversible computation is generally less space-efficient that classical computation. What is worse for quantum comptuation, however, is that we have extra "garbage" bits lying around. You don't have enough background to understand this now, but these will destroy interference interactions that are necessary for quantum computation to work. There is a theorem, however, that if you keep the input around, you can reset all these workbits to their initial values (either 0 or 1, say). We now prove this theorem.

Theorem 1 *If there is a classical circuit with a gates taking x_{in} to x_{out}, there is a reversible circuit with $O(a + |x_{out}|)$ gates consisting of NOT gates, CNOT gates, and Toffoli gates taking $(x_{in}, \bar{0}, \bar{0})$ to $(x_{in}, x_{out}, \bar{0})$. Here $\bar{0}$ means a string of 0s of the appropriate legnth.*

Proof:

First, to turn Toffoli gates into AND and NOT gates, we need to initialize some bits to 0 and 1. However, since we are allowing NOT gates, we can use a NOT gate to turn a 0 into a 1, and initialize all our extra workbits to 0.

Now, we can take our classical circuit and, with extra bits initialized to 0, turn all the OR and AND gates into Toffoli gates. This circuit C_1 takes

$$(x_{in}, \bar{0}, \bar{0}) \rightarrow (x_{garbage}, x_{out}, x'_{garbage}),$$

where $x_{garbage}$ and $x'_{garbage}$ are the output bits from the Toffoli gates that we don't use.

Now, let's take the previous circuit, and add a bunch of extra bits initialized to 0 to it. These extra bits should have just enough room to hold the desired output. We can now copy the output to these extra bits. This circuit $C_2 = C_{copy} \circ C_1$ takes

$$(x_{in}, \bar{0}, \bar{0}, \bar{0}) \rightarrow (x_{garbage}, x_{out}, x'_{garbage}, x_{out})$$

But our circuit C_1 is reversible, which means that we can run it backwards and take the output to the input. Doing this on the first three registers of the above state takes $(x_{garbage}, x_{out}, x'_{garbage})$ to $(x_{in}, \bar{0}, \bar{0}, \bar{0})$ Thus, the circuit $C_3 = C_1^{-1} C_{copy} C_1$ takes

$$(x_{in}, \bar{0}, \bar{0}, \bar{0}) \rightarrow (x_{in}, \bar{0}, \bar{0}, x_{out}).$$

We need $|x_{\rm out}|$ FANOUT gates to copy the output, and for each gate in the original circuit, we use a constant number of gates in the reversible circuit. Thus, we have proved our theorem.

The next thing that I did was to show

Theorem 2 *If there is a classical circuit with a gates taking x_{in} to x_{out}, and a classical circuit with b gates taking x_{out} to x_{in}, then there is a reversible circuit with $O(a + b + |x_{in}| + |x_{out}|$ gates that takes*

$$(x_{in}, \bar{0}) \rightarrow (x_{out}, \bar{0})$$

Proof:

This follows from the previous theorem. We know that we have a reversible circuit taking

$$(x_{in}, \bar{0}, \bar{0}) \to (x_{in}, x_{out}, \bar{0})$$

and one taking

$$(x_{out}, \bar{0}, \bar{0}) \to (x_{out}, x_{in}, \bar{0}).$$

To get the desired circuit, just apply the first circuit, swap x_{out} and x_{in}, and apply the reverse of the second circuit.

Also note that if you have a reversible circuit taking $(x_{in}, \bar{0}) \to (x_{out}, \bar{0})$, then this gives a classical circuit taking $x_{in} \to x_{out}$ and one taking $x_{out} \to x_{in}$, so we need both parts of our hypothesis.

The last thing I did in lecture was to show that CNOT SWAP, and NOT gates cannot do universal reversible computation. To do this, I proved the theorem

Theorem 3 *Suppose you have a circuit made of CNOT, SWAP, and NOT gates that takes x_1, x_2, x_3, x_4, ..., x_n to y_1, y_2, y_3, y_4, ..., y_n. Then for all y_j, either*

$$y_j = x_{i_1} \oplus x_{i_2} \oplus x_{i_3} \ldots \oplus x_{i_k}$$

or

$$y_j = x_{i_1} \oplus x_{i_2} \oplus x_{i_3} \ldots \oplus x_{i_k} \oplus 1$$

Proof Sketch: We prove the theorem by induction. It is easy to see it is true for the CNOT, SWAP, and NOT gates. We thus assume that it is correct for any circuit of at most n gates. Let us consider a circuit containing $n + 1$ gates. The last gate will have input that is of this form. It is fairly straightforward to see that the output is also of this form. For example,

$$NOT(x_{i_1} \oplus x_{i_2} \oplus x_{i_3} \ldots \oplus x_{i_k}) = x_{i_1} \oplus x_{i_2} \oplus x_{i_3} \ldots \oplus x_{i_k} \oplus 1$$

and if you take the XOR of two expressions of this form (as you do in a CNOT gate), and variables that are duplicated cancel, and you are left with something of this form. For example,

$$\Big(x_{i_3} \oplus x_{i_5} \oplus x_{i_7} \oplus x_{i_8} \oplus 1\Big) \oplus \Big(x_{i_1} \oplus x_{i_3} \oplus x_{i_7} \oplus 1\Big) = x_{i_1} \oplus x_{i_3} \oplus x_{i_8}.$$

We leave the remainder of the proof, namely, the case of SWAP to the reader.

These functions (which can be composed of CNOT, NOT, and SWAP gates) are called Boolean linear functions, since they can be expressed as

$$y = Mx + b \pmod 2$$

where M is a binary matrix and b is a binary vector.

Notes 8.370/18.435 Fall 2022

Lecture 9 Prof. Peter Shor

For the last couple of lectures, we've been talking about classical gates and classical circuits. Today, we'll start talking about quantum gates.

Recall that we can build any Boolean circuit out of AND, OR, and NOT gates. These are one- and two-bit gates. We will show that the same thing is true for quantum circuits——we can build quantum circuits out of one-qubit gates and a small set of two-qubit gates. A one-qubit gate is a 2×2 unitary matrix, and a two-qubit gate is a 4×4 unitary matrix.

We've discussed a nubmer of specific quantum gates before. For example, we've seen the Pauli matrices

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

These satisfy the commutation relations:

$$\sigma_x^2 = \sigma_y^2 = \sigma_z^2 = I$$
$$\sigma_z \sigma_x = -\sigma_x \sigma_z = i\sigma_y$$

as well as the relations obtained from this one by cyclic permutations of x, y, and z.

$$\sigma_x \sigma_y = -\sigma_y \sigma_x = i\sigma_z$$
$$\sigma_y \sigma_z = -\sigma_z \sigma_y = i\sigma_x.$$

These are often abbreviated by X, Y, and Z, especially in quantum circuit diagrams.

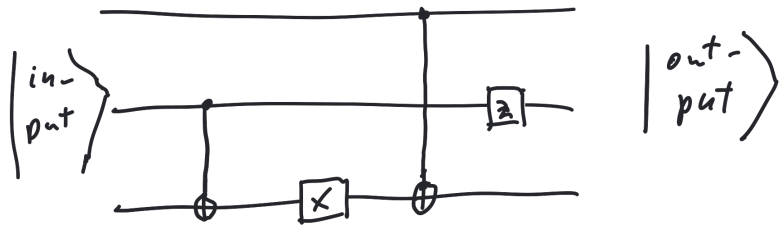
We also saw the rotations of the Bloch sphere areound the x-, y-, and z-axes by an angle 2θ:

$$R_x(2\theta) = \begin{pmatrix} \cos\theta & -i\sin\theta \\ -i\sin\theta & \cos\theta \end{pmatrix},$$
$$R_y(2\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix},$$
$$R_z(2\theta) = \begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

We have $R_z(\pi) = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} = -i\sigma_z$ and similarly $R_x(\pi) = -i\sigma_x$ and $R_y(\pi) = -i\sigma_y$. Recall that a global phase does not affect the actual quantum state, so $R_w(\pi)$ is essentially the same transformation as σ_w, where w is x, y, or z.

Finally, there is the Hadamard gate, $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, which rotates the Bloch sphere around the point $(\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}})$, switching the x- and z-axes. So we have $HXH = Z$ and $HYH = -Y$.

A quantum circuit is a set of quantum gates connecting qubits. We represent each qubit as a horizontal wire, and we draw the gates connecting two qubits as a line between two wires. Time proceeds from left to right. For example, a simple quantum circuit is given below.



Quantum Circuit

Here, there are two CNOT gates (represented by a dot on the control qubit and an XOR symbol $\oplus$ on the target qubit, a σ_x gate (represented by X and a σ_z gate (represented by a Z. The CNOT gate, for example, is the 4×4 matrix

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

But the state space of three qubits is an 8-dimensional state space. How do we apply a 4×4 matrix to it? We take the tensor product with the other qubit. For example, in the state space of three qubits, a CNOT with qubit 2 as the control and qubit 1 as the target is:

$$I_1 \otimes \mathrm{CNOT}_{2\to 3} = \begin{pmatrix} 1 & 0 & 0 & 0 & & & & \\ 0 & 1 & 0 & 0 & & & & \\ 0 & 0 & 0 & 1 & & & & \\ 0 & 0 & 1 & 0 & & & & \\ & & & & 1 & 0 & 0 & 0 \\ & & & & 0 & 1 & 0 & 0 \\ & & & & 0 & 0 & 0 & 1 \\ & & & & 0 & 0 & 1 & 0 \end{pmatrix},$$

and a CNOT with qubit 2 as the control and qubit 3 as the target would be:

$$\mathrm{CNOT}_{1\to 2}\otimes I_3 = \begin{pmatrix} 1 & 0 & & & & & & \\ 0 & 1 & & & & & & \\ & & 1 & 0 & & & & \\ & & 0 & 1 & & & & \\ & & & & & & 1 & 0 \\ & & & & & & 0 & 1 \\ & & & & 1 & 0 & & \\ & & & & 0 & 1 & & \end{pmatrix}.$$

But how do we handle a CNOT from qubit 1 to qubit 3? What we need to do is tensor $\mathrm{CNOT}_{1\to 3}$ with I_2. This isn't exactly representable in the standard Kronecker form that you've seen for $A\otimes B$, because the qubits the CNOT acts on aren't consecutive. So what do we do? One thing we can do is write down the formula for $\mathrm{CNOT}_{1\to 2}$ and then apply a change of basis matrix that interchanges qubits 2 and 3. This gives the formula:

$$\mathrm{CNOT}_{1\to 3} = \big(I_1\otimes \mathrm{SWAP}_{2,3}\big)\big(\mathrm{CNOT}_{1\to 2}\otimes I_3\big)\big(I_1\otimes \mathrm{SWAP}_{2,3}\big)$$

where

$$\mathrm{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This gives the correct answer, as does

$$\mathrm{CNOT}_{1\to 3} = \big(\mathrm{SWAP}_{1,2}\otimes I_3\big)\big(I_1\otimes \mathrm{CNOT}_{2\to 3}\big)\big(\mathrm{SWAP}_{1,2}\otimes I_3\big)$$

but it's rather cumbersome.

An easier way to do it is to break the CNOT into 2×2 blocks and tensor the identity on qubit 2 with each of the 2×2 blocks . This is shown below:

$$\mathrm{CNOT} = \left(\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}\right) \qquad \mathrm{CNOT}_{1\to 3} = \left(\begin{array}{cccc|cccc} 1 & 0 & & & 0 & 0 & & \\ 0 & 1 & & & 0 & 0 & & \\ & & 1 & 0 & & & 0 & 0 \\ & & 0 & 1 & & & 0 & 0 \\ \hline 0 & 0 & & & 0 & 1 & & \\ 0 & 0 & & & 1 & 0 & & \\ & & 0 & 0 & & & 0 & 1 \\ & & 0 & 0 & & & 1 & 0 \end{array}\right)$$

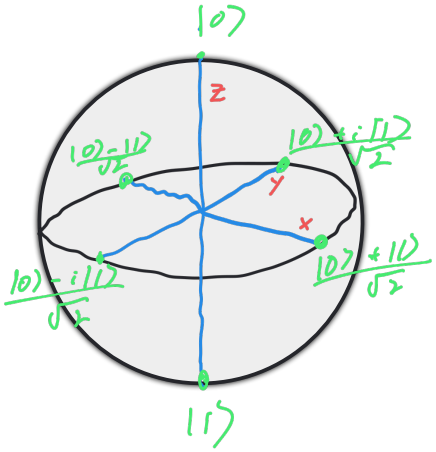
Since we aren't going to be representing gates on n qubits as $2^n\times 2^n$ matrices (for obvious reasons), we won't be seeing this kind of representation much. But it's good to know what's going on conceptually.

Recall that we showed that AND, OR, and NOT can produce any Boolean function, and TOFFOLI gates and NOT Gates can produce any reversible Boolean function.

So we can ask: is there a finite set of quantum gates that will produce any unitary transformation?

The answer to this question is "no", for a fairly straightforward reason: there are uncountably many unitary transformations, and only a countable number of quantum circuits using a finite gate set. The resolution to this is that we can approximate any unitary transformation arbitrarily well using a finite gate set, and that in fact we don't even need an unreasonable number of gates to do so. However, this is the Solovay-Kitaev theorem, and the proof of this theorem is rather involved. (It's in an appendix to the textbook, if you want to look at it.) So what we will show is that any unitary transformation can be produced using CNOT gates and one-qubit gates (in particular, a limited set of one-qubit gates).

The first thing we will do is recall the Bloch sphere. Recall that $R_y(\theta)$ was a



Bloch Sphere

rotation of θ around the y axis, and similarly for $R_z(\theta)$. Further recall that we can use $R_y(\theta)$ and $R_z(\theta)$, to perform an arbitrary rotation of the Bloch sphere by applying

$$R_z(\theta_3)R_y(\theta_2)R_z(\theta_1),$$

for some angles θ_1, θ_2, and θ_3. This isn't quite an arbitrary unitary because $R_z(\theta)$ and $R_y(\theta)$ have determinant 1, so we only get determinant-1 unitaries this way. However, we can multilply any unitary by a global phase to get a determinant-1 unitary, and global phases have no effect on a quantum state, so this gives us an arbitrary quantum transformation on a one-qubit state.

Our next goal will be to show that we can get controlled $R_z(\theta)$ and controlled $R_y(\theta)$ gates. These are

$$\text{C}{-}R_y(\theta) = \begin{pmatrix} I_2 & 0 \\ 0 & R_y(\theta) \end{pmatrix}, \qquad \text{C}{-}R_z(\theta) = \begin{pmatrix} I_2 & 0 \\ 0 & R_z(\theta) \end{pmatrix}.$$

where I_2 is the 2×2 identity matrix.

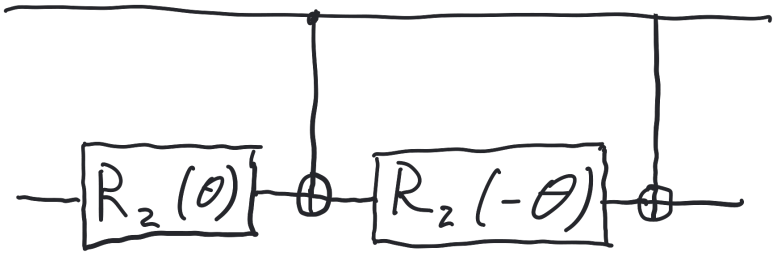
We first show how to make controlled rotations C-$R_z(\theta)$ and C-$R_y(\theta)$. First, however, let's do a straightforward matrix calculation:

$$\sigma_x R_z(\theta)\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} e^{i\theta/2} & 0 \\ 0 & e^{-i\theta/2} \end{pmatrix} = R_z(-\theta)$$

Similarly,

$$\sigma_x R_y(\theta)\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \cos\theta/2 & -\sin\theta/2 \\ \sin\theta/2 & \cos\theta/2 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} \cos\theta/2 & \sin\theta/2 \\ -\sin\theta/2 & \cos\theta/2 \end{pmatrix} = R_y(-\theta)$$

Now, consider the following circuit. If the first qubit is a $|\,0\rangle$, then we have $R_z(-\theta)R_z(\theta)$



applied to the second qubit, and these two operations cancel each other out. Now, suppose the first qubit is a $|\,1\rangle$. Recall that a CNOT is also a controlled σ_x. So if the first qubit is $|\,1\rangle$, we have $R_z(\theta)$ applied to the second qubit, followed by $\sigma_x R_z(-\theta)\sigma_x$. This is $R_z(\theta)$, which when multiplied by the first $R_z(\theta)$ gives $R_z(2\theta)$. We thus have a circuit for a C-$R_z(2\theta)$.

The same circuit with $R_z(\theta)$ replaced by $R_y(\theta)$ gives the C-$R_y(2\theta)$.

We now show how to make an arbitrary controlled unitary on two qubits. A C-U gate applies the identity to the target qubit if the control qubit is $|\,0\rangle$ and applies U to the target qubit if the control qubit is $|\,1\rangle$. That is, it is the matrix

$$\begin{pmatrix} & & 0 & 0 \\ & I_2 & 0 & 0 \\ 0 & 0 & & \\ 0 & 0 & & U \end{pmatrix}$$

where the control qubit is the first qubit and the target qubit is the second one, and I_2 is the 2×2 identity matrix.

Now, how can we apply C-U for an arbitrary one-qubit unitary U. Recall that we can express U as $R_z(\theta_3)R_y(\theta_2)R_z(\theta_1)$ for appropriately chosen θ_1, θ_2, and θ_3. Similarly

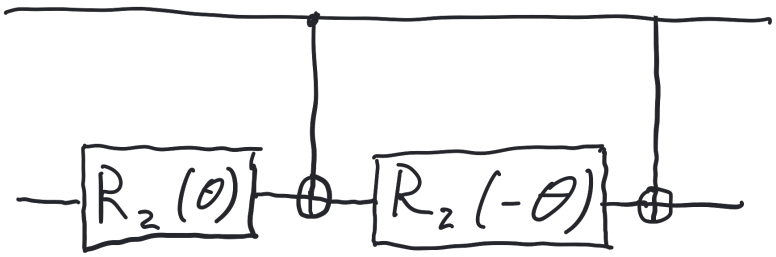
$$\text{C-}U = \text{C-}R_z(\theta_3)\,\text{C-}R_y(\theta_2)\,\text{C-}R_z(\theta_1).$$

This gives the implementation of C-U for an arbitrary one-qubit U.

Notes 8.370/18.435 Fall 2022

Lecture 10 Prof. Peter Shor

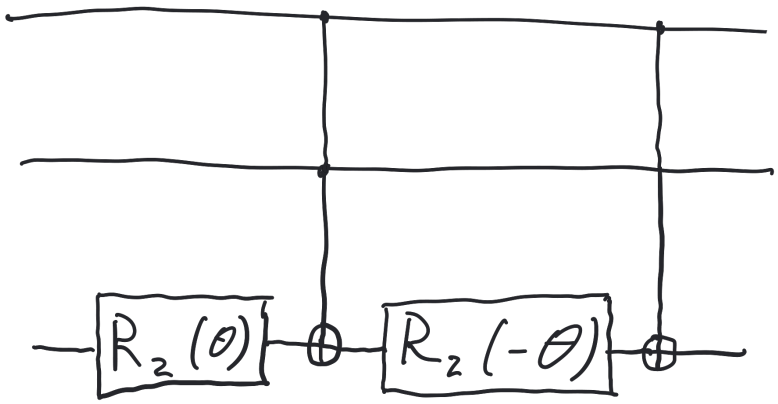
Today, we continue our discussion of gates permitting universal quantum computation. Recall that last time we showed how to make an arbitrary controlled R_y or R_z rotation, using the following circuit.



If the first qubit is a $|0\rangle$, then we have $R_z(-\theta)R_z(\theta)$ applied to the second qubit, and these two operations cancel each other out. If the first qubit is $|1\rangle$, we have $R_z(\theta)$ applied to the second qubit, followed by $\sigma_x R_z(-\theta)\sigma_x$. This is $R_z(\theta)$, which when multiplied by the first $R_z(\theta)$ gives $R_z(2\theta)$. We thus have a circuit for a C-$R_z(2\theta)$.

The same circuit with $R_z(\theta)$ replaced by $R_y(\theta)$ gives the C-$R_y(2\theta)$.

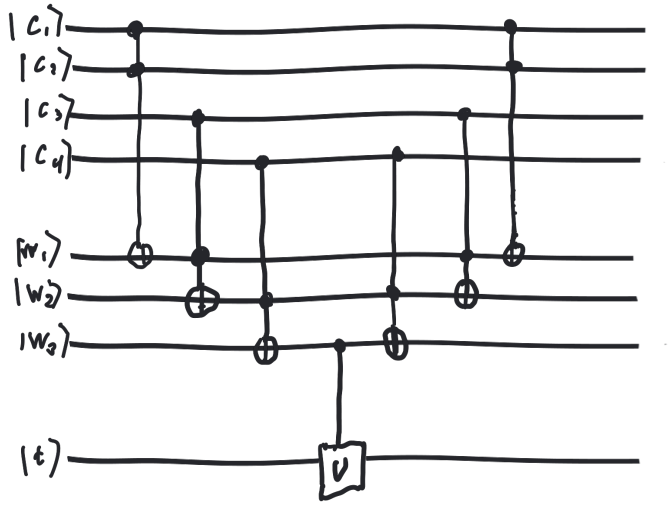
The next thing we want to do is to show how to do a doubly controlled U gate, where we apply a U if both the first two qubits are $|1\rangle$ and an identify otherwise. If we can do a Toffoli gate, we can use the same technique we used for contructing for a controlled NOT to do this:



We will not show how to make a Toffoli gate in this lecture; this will be a homework assignment.

We can construct a doubly controlled unitary CC-U using the same technique we used to construct a C-U: Find angles α, β, and γ so that $U = R_z(\gamma)R_y(\beta)R_z(\alpha)$; we then have $\mathrm{CC}{-}U = \mathrm{CC}{-}R_z(\gamma)\mathrm{CC}{-}R_y(\beta)\mathrm{CC}{-}R_z(\alpha)$.

We now will construct a C^k-U, a circuit that applies a U gate to the target qubit if the k control qubits are in the state $|\,1\rangle$, and applies the identity otherwise. This is accomplished by the following circuit:



Here, we use $k-1$ extra work qubits which start in the state $|\,0\rangle$. If all the control bits are $|\,1\rangle$, then the first $k-1$ Toffoli gates set all the work qubits to $|\,1\rangle$, and a controlled-U gate applied to the last work bit applies a U to the target qubit. Otherwise, the last work qubit remains in the state $|\,0\rangle$, and an identity is applied to the target qubit. The

last $k-1$ Toffoli gates set all the work bits back to $|0\rangle$; this step will be necessary in quantum computation to make the interference work properly.

We now have constructed enough gates to show how to make an arbitrary unitary from $R_z(\theta)$, $R_y(\theta)$, CNOT, and Toffoli gates. For this, we will need what the textbook calls a two-level gate. This is a gate on n qubits which is diagonal except for two rows and two columns, which contain a 2×2 unitary matrix in them. An example of such a gate is:

$$\begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & a & & b & & & \\ & & & 1 & & & & \\ & & c & & d & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{pmatrix},$$

where $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is a small unitary matrix.

To show that we can construct an arbitrary matirx U from two-level matrices, we need to show that we can find two-level matrices T_i so that

$$T_1^\dagger T_2^\dagger T_3^\dagger \dots T_m^\dagger = U.$$

We will work backwards, showing that there are two-level matrices $T_i^\dagger$ such that

$$U T_m T_{m-1} \dots T_1 = I.$$

Since the conjugate transpose of T_i (which is also the inverse of T_i, because T_i is unitary) is a two-level matrix, this will show that U is a product of two-level matrices.

When you multiply U by a two-level matrix, it will only affect two columns and two rows of U, since a $r \times r$ two-level matrix is an identity in $r-2$ of its rows and columns. Now, for an arbitrary $r \times r$ matrix U, there is a two-level matrix T_1 (which only affects the first and last columns) that sets the lower left entry of UT, $UT(r,1)$, to 0. Now that UT_1 is 0, we can find another two-level matrix T_2 that sets the $(r-1,1)$ entry of UT_1T_2 to 0. If we continue setting entries of $UT_1T_2 \dots T_k$ to 0, and work from the lower right up, when we apply a new two-level matrix, we will never undo one of the zeroes, because we have already set all the entries to the left and below the entry we are working on to 0. Thus, doing this, we can set all the entries below the diagonal to 0. Because the length of all rows and columns in a unitary matrix are 1, this means that all the entries above the diagonal are also 0, and that the diagonal contains unit complex numbers. These can all be set to 1 by applying another sequence of two-level matrices.

So how can we construct an arbitrary two-level matrix? We have already constructed one class of two-level matrices. If we have n qubits, then the multiply con-

trolled gate with $n-1$ control qubits and one target qubit looks like:

$$\begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & a & b \\ & & & & & & c & d \end{pmatrix},$$

We only need to show that we can move the unitary in the last two rows and columns to an arbitrary pair of rows and columns. We can do this with an appropriate sequence of NOTs and CNOTS, which we will describe next.

Actually, for ease of exposition, it's better to put the unitary matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ at the top left of the matrix rather than the bottom right. We can do that by applying a NOT gate to all but the last qubit, giving this matrix.

$$M_{0,1} = \begin{pmatrix} a & b & & & & & & \\ c & d & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{pmatrix},$$

Now, suppose we want to construct the matrix $M_{2,4}$ where the unitary is in rows and columns 2 and 4, rather than 0 and 1.

$$M_{2,4} = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & a & & b & & & \\ & & & 1 & & & & \\ & & c & & d & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{pmatrix},$$

What we can do is construct a permutation matrix P that takes the basis vector $|\, e_2\rangle = |\, 010\rangle$ to the basis vector $|\, e_0\rangle = |\, 000\rangle$ and the basis vector $|\, e_4\rangle = |\, 100\rangle$ to basis vector $|\, e_1\rangle = |\, 001\rangle$, and then use P to construct $M_{2,4}$ by applying $P^{-1}M_{0,1}P$. The matrix $P^{-1}M_{0,1}P$ only affects the $|\, 010\rangle$ and the $|\, 100\rangle$ coordinates of a vector v, because the P^1 undoes everything that $M_{0,1}$ does not affect.

So how do we move two arbitrary basis vectors to $|\, 000\rangle$ and $|\, 001\rangle$? We will give a proof by example; once you've seen how it works for two specific basis vectors, extending the procedure to an arbitrary two basis vectors is straightforward. What we

want to do is to move the basis vectors $|\,010\rangle$ and $|\,100\rangle$ to the basis vectors $|\,000\rangle$ and $|\,001\rangle$. We use NOTs and CNOTs for this. First, we can move the coordinate $|\,010\rangle$ to $|\,000\rangle$ by applying a NOT gate to the second qubit. Applying this NOT gate takes $|\,100\rangle$ to $|\,110\rangle$. We next take $|\,110\rangle$ to $|\,001\rangle$ by applying CNOT gates. These CNOT gates do not affect $|\,000\rangle$, because it contains all 0s. We can do this as follows:

$$\begin{aligned}
\mathrm{CNOT}_{1,3}\,|\,110\rangle &= |\,111\rangle \\
\mathrm{CNOT}_{1,2}\,|\,111\rangle &= |\,101\rangle \\
\mathrm{CNOT}_{3,1}\,|\,101\rangle &= |\,001\rangle\,.
\end{aligned}$$

we thus have constructed P, and this lets us produce $M_{2,4}$. So we are done.

Notes 8.370/18.435 Fall 2022

Lecture 11 Prof. Peter Shor

Today, we will be talking about quantum teleportation.

Note: This lecture took a little more than a class period,

What is quantum teleportation? There is a theorem (which we will prove shortly) that you cannot send quantum information over a classical channel. More specifically, you cannot send an unknown quantum state $|\psi\rangle$ from Alice to Bob if the only communication channels connecting them are classical.

However, if Alice and Bob share an EPR pair, Alice can send Bob an unknown quantum state $|\psi\rangle$ using an EPR pair and two bits communicated on a classical channel. This process is known as quantum teleportation, and destroys the EPR pair and Alice's copy of $|\psi\rangle$.

Theorem 1 *If Alice and Bob have a classical channel between them, but do not share any entanglement, then Alice cannot send an unknown quantum state to Bob.*

We want to show that you cannot send an unknown quantum state over a classical channel. (If you have a known quantum state, you can send an arbitrarily precise description of it, just by sending a message like "$0.809017\,|0\rangle + 0.587785\,|1\rangle$".)

We will prove the theorem by contradiction. Let's say that Alice receives an unknown quantum state $|\psi\rangle$, and can encode it in a classical channel, which she then sends to Bob, who can reconstruct $|\psi\rangle$. There is nothing preventing Alice from duplicating the classical information going down the channel, so she could then send the same information to Carol, who could also reconstruct $|\psi\rangle$, using the same recipe Bob uses. We have thus cloned $|\psi\rangle$, a contradiction,so we have proved the theorem.

However, you can send a quantum state over a classical channel if Alice and Bob have an entangled state. More precisely, we will show if they have an EPR pair of qubits in the state $\frac{1}{\sqrt{2}}(|00\rangle_{AB} + |11\rangle_{AB})$, with Alice holding one of these qubits and Bob the other, then Alice can send Bob the state of an unknown qubit by using two classical bits.

This does not allow us to clone, because while a third party, Carol, could copy the two classical bits sent, she does not share the EPR pair that was used to teleport the qubit. Without the EPR pair, the classical bits used to teleport are completely random, so that she learns nothing from them. Further, the unknown qubit that Alice is sending over the channel is measured during the teleportation operation, so Alice transmits the state of the unknown qubit, but does not clone it. Thus, quantum teleportation doesn't contradict our no-go theorem.

Before I describe teleportation, we need a little background. The *Bell basis* for a two qubit state space is

$$\frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big), \quad \frac{1}{\sqrt{2}}\big(\,|00\rangle - |11\rangle\,\big), \quad \frac{1}{\sqrt{2}}\big(\,|01\rangle + |10\rangle\,\big), \quad \frac{1}{\sqrt{2}}\big(\,|01\rangle - |10\rangle\,\big).$$

These form an orthonormal basis for the set of two qubits. Each of these is called a *Bell state*, and each of Alice and Bob can switch from one Bell state to another by applying

a Pauli matrix to their qubit:

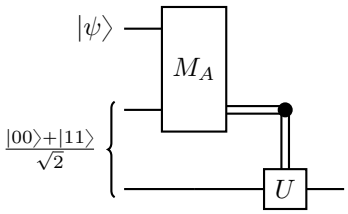
$$\frac{1}{\sqrt{2}}\big(\,|00\rangle - |11\rangle\,\big) = \big(\sigma_z \otimes I\big)\frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big)$$
$$\frac{1}{\sqrt{2}}\big(\,|01\rangle + |10\rangle\,\big) = \big(\sigma_x \otimes I\big)\frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big)$$
$$\frac{1}{\sqrt{2}}\big(\,|01\rangle - |10\rangle\,\big) = i\big(\sigma_y \otimes I\big)\frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big).$$

How does it work? In this lecture, I will first give the teleportation protocol, and show that it works for one of Alice's measurement results by explicit calculation. I will then use a clever manipulations of formulas to show that it works in the remaining three cases. Finally, i will derive the quantum circuit for teleportation.

So how does teleportation work? Alice holds a qubit in a state she does not know, which we will call $|\psi\rangle = \alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}$. There are also two qubits in the state $\frac{1}{\sqrt{2}}\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\,\big)$, of which Alice holds the first and Bob the second. Thus, the state is

$$\frac{1}{\sqrt{2}}\big(\alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}\,\big)\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\,\big)$$

What is the teleportation protocol? Alice measures the first two qubits of her state using the Bell basis, and then sends the results of the measurement (which can be encoded in two bits, since there are four outcomes) to Bob. Bob then applies a unitary to his state depending on the measurement. So in broad outline, this looks like:



One way to show that this works is do four computations, one for each of the possible measurement results that Alice gets. This isn't hard, but you'll learn more from my showing you a cleverer way. This way involves doing the computation explicitly for one of the four measurement outcomes, and deriving the other three from this one.

First, let's do the case where Alice gets $\frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\,\big)$ as the result of her measurement:

$$\tfrac{1}{2}\big({}_{A_1A_2}\langle 00| + {}_{A_1A_2}\langle 11|\,\big)\big(\alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}\,\big)\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\,\big) = \tfrac{1}{2}\big(\alpha\,|0\rangle_B + \beta\,|1\rangle_B\,\big)$$

The ${}_{A_1A_2}\langle 00|$ picks out the term $\alpha\,|000\rangle_{A_1A_2B}$ and, after the inner product, leaves $\alpha\,|0\rangle_B$. Similarly, the term ${}_{A_1A_2}\langle 11|$ picks out the term $\beta\,|111\rangle_{A_1A_2B}$ and leaves $\beta\,|1\rangle_B$. Thus, the measurement outcome occurs with probability $\big(\frac{1}{2}\big)^2 = \frac{1}{4}$, and Bob gets Alice's original unknown state $\alpha\,|0\rangle + \beta\,|1\rangle = |\psi\rangle$.

For our next calculation, we need some identities on the Bell states:

$$\begin{aligned}
\Big(\sigma_x \otimes I\Big)\frac{|00\rangle + |11\rangle}{\sqrt{2}} &= \Big(I \otimes \sigma_x\Big)\frac{|00\rangle + |11\rangle}{\sqrt{2}} = \frac{|19\rangle + |01\rangle}{\sqrt{2}} \\
\Big(\sigma_z \otimes I\Big)\frac{|00\rangle + |11\rangle}{\sqrt{2}} &= \Big(I \otimes \sigma_z\Big)\frac{|00\rangle - |11\rangle}{\sqrt{2}} = \frac{|00\rangle - |11\rangle}{\sqrt{2}} \\
\Big(\sigma_y \otimes I\Big)\frac{|00\rangle + |11\rangle}{\sqrt{2}} &= -\Big(I \otimes \sigma_y\Big)\frac{|00\rangle + |11\rangle}{\sqrt{2}} = i\frac{|10\rangle - |01\rangle}{\sqrt{2}}
\end{aligned}$$

These properties are fairly easy to prove, so I won't do it in these notes.

Suppose Alice gets the result $\frac{1}{\sqrt{2}}(I \otimes \sigma_w)(|00\rangle + |11\rangle)$ where $w = x$ or z. Then we have that the state of the system is

$$\tfrac{1}{2}\big({}_{A_1A_2}\langle 00| + {}_{A_1A_2}\langle 11|\big)\big(I_{A_1} \otimes \sigma_{w(A_2)}\big)\big(\alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}\big)\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\big)$$

But now, $\sigma_{w(A_2)}$ doesn't affect qubit A_1, so we can move this Pauli and get

$$\tfrac{1}{2}\big({}_{A_1A_2}\langle 00| + {}_{A_1A_2}\langle 11|\big)\big(\alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}\big)\big(\sigma_{w(A_2)} \otimes I_B\big)\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\big)$$

Now, by the above identities on the Bell basis, this is the same as

$$\tfrac{1}{2}\big({}_{A_1A_2}\langle 00| + {}_{A_1A_2}\langle 11|\big)\big(\alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}\big)\big(I_{A_2} \otimes \sigma_{w(B)}\big)\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\big)$$

(this is where we use $w = x$ or z). But because $\sigma_{w(B)}$ only affects Bob's qubit, we can move it all the way to the left, across Alice's qubits and operations, to get

$$\tfrac{1}{2}\sigma_{w(B)}\big({}_{A_1A_2}\langle 00| + {}_{A_1A_2}\langle 11|\big)\big(\alpha\,|0\rangle_{A_1} + \beta\,|1\rangle_{A_1}\big)\big(\,|00\rangle_{A_2B} + |11\rangle_{A_2B}\big)$$

However, except for the $\sigma_{w(B)}$, this is exactly what we get when Alice measures her first outcome, and we've computed this before. So we are left with

$$\sigma_{w(B)}\big(\alpha\,|0\rangle_B + \beta\,|1\rangle_B\big) = \sigma_w\,|\psi\rangle\,,$$

and if Bob applies σ_w (he knows what w is because Alice tells him the result of her measurement), he gets $|\psi\rangle$, the state Alice wanted to teleport. The third case, when Alice measures $\frac{1}{\sqrt{2}}\big(\,|01\rangle - |10\rangle\big)$, works exactly the same way except for the phases, which don't matter because they are global phases.

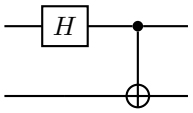
How can we figure out a quantum circuit for teleportation? There are two elements to this: first, we need to figure out how to measure in the Bell basis, and second, we need how to apply the correct Pauli matrix to correct the state to $|\psi\rangle$.

How do we figure out how to measure in the Bell basis? The easiest way to do it is to work backwards. We want to find a circuit where you input one of the elements of the Bell basis, and where we output 00, 01, 10, 11. after a measurement. Let's start by figuring out a circuit where we input $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, and output a member of the Bell basis. This is actually fairly easy to do. First, we need to make a superposition of states at some point, and then we need to entangle the two qubits. To make a superposition of states, we use the Hadamard gate, $H = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$, and to entangle the

qubits we use the CNOT gate:

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

So the quantum circuit we will try is:

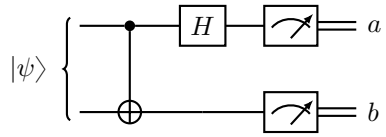


What happens with this circuit? Here is a table of the inputs, the intermediate states, and the outputs:

input	after H	output
$\|00\rangle$	$\frac{1}{\sqrt{2}}\big(\|0\rangle + \|1\rangle\big)\|0\rangle$	$\frac{1}{\sqrt{2}}\big(\|00\rangle + \|11\rangle\big)$
$\|01\rangle$	$\frac{1}{\sqrt{2}}\big(\|0\rangle + \|1\rangle\big)\|1\rangle$	$\frac{1}{\sqrt{2}}\big(\|01\rangle + \|10\rangle\big)$
$\|10\rangle$	$\frac{1}{\sqrt{2}}\big(\|0\rangle - \|1\rangle\big)\|0\rangle$	$\frac{1}{\sqrt{2}}\big(\|00\rangle - \|11\rangle\big)$
$\|11\rangle$	$\frac{1}{\sqrt{2}}\big(\|0\rangle - \|1\rangle\big)\|1\rangle$	$\frac{1}{\sqrt{2}}\big(\|01\rangle - \|10\rangle\big)$

So this does what we want it to do.

Now, let's run the circuit in reverse.

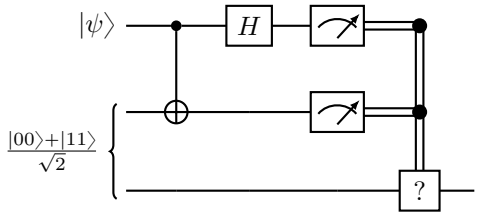


The probability of getting output a, b is

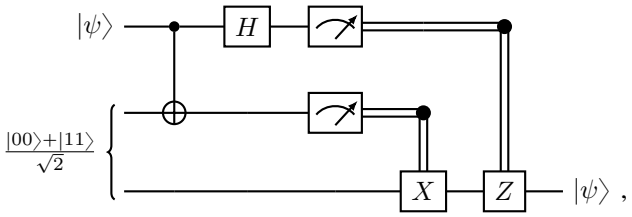
$$|\langle ab|\, H_1 \mathrm{CNOT}_{1\to 2}\, |\psi\rangle|^2,$$

so this measures the input state in the basis $\langle ab|\, H_1 \mathrm{CNOT}_{1\to 2}$, where a and b are either 0 or 1, which gives a measurement in the Bell basis.

So now, our quantum circuit looks like:



And all we need to do is figure out which unitary we need to use to make the correction. Recall that we showed that we need to make the correction σ_w when Alice measures the state $(\sigma_w \otimes I)\frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big)$. By the table above, if the first measurement result is $|1\rangle$, we need to apply a σ_z and if the second measurement bit is $|1\rangle$, we need to apply a σ_x. This gives the circuit



so we have derived a quantum circuit for teleportation.

We can represent quantum teleportation schematically as follows:

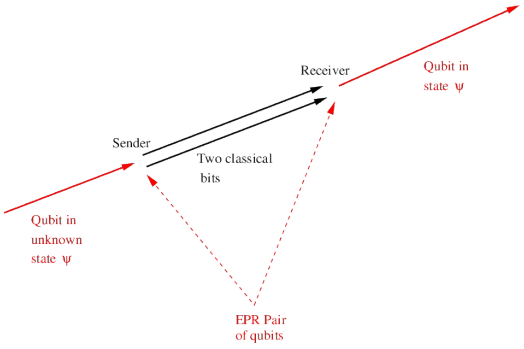


Figure 1: teleportation

Time goes up in this figure, so the first thing that happens is that the sender and receiver share an EPR pair. Then, the sender encodes her unknown qubit and sends it to the receiver, who decodes it with the help of his half of the EPR pair.

There is a converse process to teleortation: superdense coding. Here, if they share an EPR pair, the sender and receiver can send two classical bits using one quantum bit. A schematic representation of this process is:

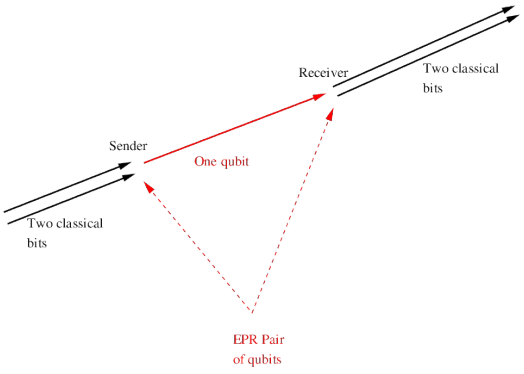


Figure 2: superdense coding

As we will see, for superdense coding, the sender encodes using the same process the receiver uses to decode in teleportation, and the receiver decodes using the process the sender uses to encode in teleportation.

How does the process work? Recall the Bell basis is four entangled states, and can be obtained from the state $|00\rangle + |11\rangle$ by applying a Pauli matrix. We have:

$$\begin{aligned}
&\frac{1}{\sqrt{2}}\big(\,|00\rangle + |11\rangle\big) \\
&\frac{1}{\sqrt{2}}\big(\,|00\rangle - |11\rangle\big) = \frac{1}{\sqrt{2}}(\sigma_z \otimes I)\big(\,|00\rangle + |11\rangle\big) \\
&\frac{1}{\sqrt{2}}\big(\,|01\rangle + |10\rangle\big) = \frac{1}{\sqrt{2}}(\sigma_x \otimes I)\big(\,|00\rangle + |11\rangle\big) \\
&\frac{1}{\sqrt{2}}\big(\,|01\rangle - |10\rangle\big) = \frac{1}{\sqrt{2}}(\sigma_z\sigma_x \otimes I)\big(\,|00\rangle + |11\rangle\big)
\end{aligned}$$

Thus, Alice can convert the EPR pair she shares with Bob to any one of the Bell basis states. When she sends her qubit to Bob, he measures in the Bell basis, and gets one of four values, giving him two classical bits. So superdense coding works.

You can also use superdense coding to show that teleportation is optimal. Even with an arbitrarily large number of Bell pairs, you cannot send a qubit using fewer than two classical bits. Why not? Suppose you could find a protocol that sent a qubit using 1.9 classical bits on average. Then, encoding two classical bits using superdense coding, and encoding the resulting qubit with the improved teleportation protocol, you could send 2 classical bits using entanglement and 1.9 classical bits on average. Repeating this k times, for large n you could send n classical bits using $(0.95)^k n$ classical bits on average. While we won't prove it in class, Shannon's channel capacity theorem shows that this in turn would let you send classical information faster than the speed of light using just entanglement, which we assume is impossible from Einstein's theory of relativity.

Notes 8.370/18.435 Fall 2022

Lecture 12 Prof. Peter Shor

Today we started on density matrices.

Suppose we have a probabilistic mixture of quantum states. That is, suppose somebody prepares the state $|\,v_1\rangle$ with probability p_1, the state $|\,v_2\rangle$ with probability p_2, the state $|\,v_3\rangle$ with probability p_3, and so on. How can we describe it? It turns out that a very good way to describe it is with something called a *density matrix*. If we have a system that is in state $|\,v_i\rangle$ with probability p_i, then the density matrix is

$$\rho = \sum_i p_i \,|\,v_i\rangle\langle v_i\,|\,.$$

Density matrices are customarily denoted by the variable ρ.

The density matrix ρ is a Hermitian positive semi-definite trace 1 matrix. Here, *trace 1* means that $\mathrm{Tr}\,\rho = 1$. It is easy to see that it is trace 1 because

$$\mathrm{Tr}\sum_i p_i \,|\,v_i\rangle\langle v_i\,| = \sum_i p_i \mathrm{Tr}\;|\,v_i\rangle\langle v_i\,| = \sum_i p_i = 1$$

The word $positive$ means that it has all non-negative eigenvalues. This is equivalent to the condition $\langle v\,|\,\rho\,|\,v\rangle \geqslant 0$ for all $|\,v\rangle$. It is also easy to see that it is positive because for any $|\,w\rangle$, we have $\langle w|v_i\rangle\langle v_i|w\rangle \geqslant 0$. This means that

$$\langle w\,|\,\rho\,|\,w\rangle = \sum_i p_i \langle w|v_i\rangle\langle v_i|w\rangle \geqslant 0 \;\forall\,|\,w\rangle\,.$$

In fact, any Herititian positive semidefinite trace 1 matrix ρ is a density matrix for some probabilistic mixture of quantum states, One way to see this is to use the eigenvectors of ρ for the quantum states and the eigenvalues for the probabilities.

The difference between a *superposition* of states and a *mixture* of states is important. A superposition of states is something like

$$\frac{1}{\sqrt{2}}\,|\,0\rangle + \frac{1}{\sqrt{2}}\,|\,1\rangle\,,$$

while a mixed state is

$$|\,0\rangle\langle 0\,| \text{ with probability } \frac{1}{2} \quad \text{and} \quad |\,1\rangle\langle 1\,| \text{ with probability } \frac{1}{2},$$

which would be represented by the density matrix

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2}. \end{pmatrix}$$

For these two states, if you measure either one in the basis $\{|\,0\rangle\,, |\,1\rangle\}$, you get $|\,0\rangle$ with probability $\frac{1}{2}$ and $|\,1\rangle$ with probability $\frac{1}{2}$. But they don't behave the same. Consider what happens if you apply the Hadamard gate $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and then

measure them in the basis $\{|\,0\rangle, |\,1\rangle\}$. (This is equivalent to measuring in the basis $\{|+\rangle, |-\rangle\}$.) The first one is taken to the state $|\,0\rangle$, and you observe the outcome $|\,0\rangle$ with probabiity 1. The second is taken to the state

$$|+\rangle \text{ with probability } \frac{1}{2} \quad \text{and} \quad |-\rangle \text{ with probability } \frac{1}{2},$$

and you observe the outcomes $|\,0\rangle$ and $|\,1\rangle$. with probabiity $\frac{1}{2}$ each.

There is another way of computing this result. To apply a unitary U to a density matrix ρ, you take $U\rho U^\dagger$. Thus, for the calculation above, we have

$$H \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} H^\dagger = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} HH^\dagger = \begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix},$$

since H commutes with the identity matrix.

One thing to notice is that two different probabilistic ensembles of quantum states can give the same density matrix. For example, suppose you have the ensembles

(a) $\frac{4}{5}|\,0\rangle + \frac{3}{5}|\,1\rangle$ with probability $\frac{1}{2}$ and $\frac{4}{5}|\,0\rangle - \frac{3}{5}|\,1\rangle$ with probability $\frac{1}{2}$,

(b) $|\,0\rangle$ with probability $\frac{16}{25}$ and $|\,1\rangle$ with probability $\frac{9}{25}$.

These both give the density matrix

$$\begin{pmatrix} \frac{16}{25} & 0 \\ 0 & \frac{9}{25} \end{pmatrix}.$$

So why use density matrices if they don't give a complete description of the quantum state? The reason is that if you know the density matrix, this is sufficient information to tell you the outcome of any experiment on the quantum state. Let's now give a demonstration of this fact.

Suppose we use the basis $\left\{\frac{2}{\sqrt{5}}|\,0\rangle + \frac{1}{\sqrt{5}}|\,1\rangle, -\frac{1}{\sqrt{5}}|\,0\rangle + \frac{2}{\sqrt{5}}|\,1\rangle\right\}$ to measure the probabilistic ensemble (a) above. The probability of observing the first basis state is

$$\frac{1}{2}\left|\left(\tfrac{2}{\sqrt{5}}\langle 0\,| + \tfrac{1}{\sqrt{5}}\langle 1\,|\right)\left(\tfrac{4}{5}|\,0\rangle + \tfrac{3}{5}|\,1\rangle\right)\right|^2 + \frac{1}{2}\left|\left(\tfrac{2}{\sqrt{5}}\langle 0\,| + \tfrac{1}{\sqrt{5}}\langle 1\,|\right)\left(\tfrac{4}{5}|\,0\rangle - \tfrac{3}{5}|\,1\rangle\right)\right|^2,$$

which is

$$\frac{1}{2}\left(\frac{11}{5\sqrt{5}}\right)^2 + \frac{1}{2}\left(\frac{5}{5\sqrt{5}}\right)^2 = \frac{73}{125}$$

If we do this for the probabilistic ensemble (b), we get

$$\frac{16}{25}\left|\left(\tfrac{2}{\sqrt{5}}\langle 0\,| + \tfrac{1}{\sqrt{5}}\langle 1\,|\right)|\,0\rangle\right|^2 + \frac{9}{25}\left|\left(\tfrac{2}{\sqrt{5}}\langle 0\,| + \tfrac{1}{\sqrt{5}}\langle 1\,|\right)\,|\,1\rangle\right|^2 = \frac{64}{25} + \frac{9}{25} = \frac{73}{125}$$

We will see that the formula for the probability of obtaining $|\,v\rangle$ in a von Neumann measurement on state ρ is $\langle v\,|\,\rho\,|\,v\rangle$. For the example above,

$$\langle v\,|\,\rho\,|\,v\rangle = \left(\tfrac{2}{\sqrt{5}}, \tfrac{1}{\sqrt{5}}\right)\begin{pmatrix} \frac{16}{25} & 0 \\ 0 & \frac{9}{25} \end{pmatrix}\begin{pmatrix} \frac{2}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} \end{pmatrix} = \frac{73}{125}.$$

To show that the outcome of any experiment depends only on the density matrix, we need to prove that if you apply a unitary U to a mixture of quantum states with density matrix ρ, you get a mixture of quantum states with density matrix $U\rho U^\dagger$, and that if you perform a measurement on a mixture of quantum states, the density matrix is enough to predict the outcome. We now show these two facts.

Suppose you have a mixture of quantum states, where $|\, v_i\rangle$ has probability p_i. Applying U gives the mixture where $U\,|\, v_i\rangle$ has probability p_i. Computing the density matrix of this mixture gives

$$\sum_i p_i U\,|\, v_i\rangle\langle v_i\,|\, U^\dagger = U\left(\sum_i p_i\,|\, v_i\rangle\langle v_i\,|\right)U^\dagger = U\rho U^\dagger,$$

as we wanted.

Now, suppose we have a mixture of quantum states with density matrix ρ, and we measure it with the von Neumann measurement corresponding to the basis $\{|\, w_i\rangle\}$. The probability that we see $|\, w_j\rangle$ if we started with state $|\, v_i\rangle$ is $|\langle w_j|v_i\rangle|^2 = \langle w_j|v_i\rangle\langle v_i|w_j\rangle$, so the total probability that we see $|\, w_j\rangle$

$$\begin{aligned}\sum_i p_i\langle w_j|v_i\rangle\langle v_i|w_j\rangle &= \langle w_j\,|\left(\sum_i p_i\,|\, v_i\rangle\langle v_i\,|\right)|\, w_j\rangle \\ &= \langle w_j\,|\,\rho\,|\, w_j\rangle\,,\end{aligned}$$

which depends only on ρ and not the p_i's and $|\, v_i\rangle$s.

Do these calculations show that density matrices are sufficient to predict all experimental outcomes from mixed quantum states? Not quite—we haven't shown that results of the more general type of von Neumann measurements, with projections onto subspaces of rank greater than 1, can be predicted from the density matrix. Let's do that now.

Recall that a von Neumann measurement has Hermitian projection matrices Π_1, Π_2, …, Π_r with $\sum_i \Pi_r = I$. When applied to a quantum state $|\,\phi\rangle$, we observe the rth outcome with probability

$$|\Pi_r\,|\,\psi\rangle\,|^2 = \langle\psi\,|\,\Pi_r\,|\,\psi\rangle$$

and the state after the measurement is

$$\frac{1}{|\Pi_r\,|\,\psi\rangle\,|}\Pi_r\,|\,\psi\rangle\,.$$

So what is the probability of observing the rth outcome if we have a probabilistic ensemble of quantum states where $|\, v_i\rangle$ appears with probabiity p_i. It's just the weighted

sum of the probability of each outcome:

$$\begin{aligned}\sum_i |\Pi_r \mid v_i\rangle|^2 &= \sum_i p_i \langle v_i \mid \Pi_r \mid v_i\rangle \\ &= \sum_i p_i \mathrm{Tr}\ \langle v_i \mid \Pi_r \mid v_i\rangle \\ &= \sum_i p_i \mathrm{Tr}\, \Pi_r \mid v_i\rangle\langle v_i \mid \\ &= \mathrm{Tr}\, \Pi_r \left(\sum_i p_i \mid v_i\rangle\langle v_i \mid\right) \\ &= \mathrm{Tr}\, \Pi_r \rho,\end{aligned}$$

which depends only on the density matrix. For the second and third steps of this calculation, we used the fact that the trace of a scalar, i.e., a 1×1 matrix, is just the scalar, and the cyclic property of the trace:

$$\mathrm{Tr}\, ABC = \mathrm{Tr}\, BCA = \mathrm{Tr}\, CAB$$

(which is actually a straightforward consequence of $\mathrm{Tr}\, AB = \mathrm{Tr}\, BA$).

What we will do now is compute the residual state we get if we start with v_i and observe the rth outcome, and then compute the conditional probability of having started with v_i, given that we observe the rth outcome. These last two quantities will let us compute the density matrix, conditional on having observed the rth outcome.

First, the residual state is

$$\frac{\Pi_r \mid v_i\rangle}{|\Pi_r \mid v_i\rangle|} = \frac{\Pi_r \mid v_i\rangle}{\langle v_r \mid \Pi_r \mid v_i\rangle^{1/2}}.$$

The conditional probability that we started with $\mid v_i\rangle$, given that we observe the rth outcome, can be calculated using Bayes' rule. This gives:

$$\frac{p_i \langle v_i \mid \Pi_r \mid v_i\rangle}{\sum_i p_i \langle v_i \mid \Pi_r \mid v_i\rangle} = \frac{p_i \langle v_i \mid \Pi_r \mid v_i\rangle}{\mathrm{Tr}\ (\Pi_r \rho)}$$

The conditional density matrix, given that we observe the rth outcome, is thus.

$$\begin{aligned}\rho_r &= \sum_i \frac{p_i \langle v_i \mid \Pi_r \mid v_i\rangle}{\mathrm{Tr}\ (\Pi_r \rho)} \cdot \frac{\Pi_r \mid v_i\rangle}{\langle v_r \mid \Pi_r \mid v_i\rangle^{1/2}} \frac{\langle v_i \mid \Pi_r}{\langle v_r \mid \Pi_r \mid v_i\rangle^{1/2}} \\ &= \frac{\Pi_r \left(\sum_i p_i \mid v_i\rangle\langle v_i \mid\right) \Pi_r}{\mathrm{Tr}\ (\Pi_r \rho)} \\ &= \frac{\Pi_r \rho \Pi_r}{\mathrm{Tr}\, (\Pi_r \rho)},\end{aligned}$$

which depends only on ρ, so we are done. Note that $\mathrm{Tr}\ (\Pi_r \rho \Pi_r) = \mathrm{Tr}\ (\Pi_r \rho)$, so this indeed has trace 1, and so is a valid density matrix.

Notes 8.370/18.435 Fall 2022

Lecture 13 Prof. Peter Shor

Today we talked more about density matrices.

There are two ways of obtaining density matrices. One is by having a probabilistic mixture of states, and the other is by starting with an entangled state and disregarding (or throwing away) part of it. We explained the first case on Friday. Today, we will explain the second case.

Let's do an example before we explain things in general. Suppose we have the state

$$\frac{2}{\sqrt{5}}\,|\,00\rangle_{AB} + \frac{1}{\sqrt{5}}\,|\,11\rangle_{AB}$$

shared between Alice and Bob. Now, suppose Alice measures her qubit but doesn't tell Bob the result. Bob will have a probabilistic mixture of quantum states, i.e., a density matrix.

FIrst, let's assume Alice measures her qubit in the $\{|\,0\rangle, |\,1\rangle\}$ basis. She gets the outcome $|\,0\rangle$ with probability $\frac{4}{5}$, in which case Bob also has the state $|\,0\rangle$, and she gets the state $|\,1\rangle$ with probability $\frac{1}{5}$, in which case Bob also has the state $|\,1\rangle$. Thus, Bob's density matrix (assuming he doesn't learn Alice's measurement result) is

$$\frac{4}{5}\,|\,0\rangle\langle 0\,| + \frac{1}{5}\,|\,1\rangle\langle 1\,| = \frac{4}{5}\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{5}\begin{pmatrix} 0 & 0 \\ 010 & \end{pmatrix} = \begin{pmatrix} \frac{4}{5} & 0 \\ 0 & \frac{1}{5} \end{pmatrix}.$$

Now, let's assume that Alice measures in the $\{|\,+\rangle, |\,-\rangle\}$ basis. If Alice gets the result $|\,+\rangle$, we have

$$\frac{1}{\sqrt{2}}\Big(\langle 0\,| + \langle 1\,|\Big)\Big(\tfrac{2}{\sqrt{5}}\,|\,00\rangle_{AB} + \tfrac{1}{\sqrt{5}}\,|\,11\rangle_{AB}\Big) = \frac{2}{\sqrt{10}}\,|\,0\rangle + \frac{1}{\sqrt{10}}\,|\,1\rangle,$$

so Bob's state is $\frac{1}{5}\big(2\,|\,0\rangle + |\,1\rangle\big)$ with probability $\frac{4}{10} + \frac{1}{10} = \frac{1}{2}$.

A similar calculation shows that if Alice gets $|\,-\rangle$, then Bob's state is $\frac{1}{5}\big(2\,|\,0\rangle - |\,1\rangle\big)$ with probability $\frac{1}{2}$.

Bob's density matrix in this case is

$$\frac{1}{2}\cdot\frac{1}{5}\begin{pmatrix} 4 & 2 \\ 2 & 1 \end{pmatrix} + \frac{1}{2}\cdot\frac{1}{5}\begin{pmatrix} 4 & -2 \\ -2 & 1 \end{pmatrix} = \begin{pmatrix} \frac{4}{5} & 0 \\ 0 & \frac{1}{5} \end{pmatrix}.$$

Thus, in this case Bob's density matrix doesn't depend on the basis that Alice used for her measurement. We will later show that this is true in general.

The operation on matrices that gets a density matrix on a state space B (or A) from a density matrix on the joint state space of A and B. is called a *partial trace*. Suppose we have a density matrix ρ_{AB} on a joint system AB. If A is a qubit, we can express it as

$$\rho_{AB} = \begin{pmatrix} P & Q \\ R & S \end{pmatrix},$$

where P, Q, R, S, are matrices on the quantum system B. Now, the partial trace over A is

$$\mathrm{Tr}_A \begin{pmatrix} P & Q \\ R & S \end{pmatrix} = P + S \tag{1}$$

and the partial trace over B is

$$\mathrm{Tr}_B \begin{pmatrix} P & Q \\ R & S \end{pmatrix} = \begin{pmatrix} \mathrm{Tr}\, P & \mathrm{Tr}\, Q \\ \mathrm{Tr}\, R & \mathrm{Tr}\, S \end{pmatrix}.$$

This generalizes in a straightforward way to the case where A is not a qubit. If A has dimension j and B has dimension k, then ρ_{AB} is a $j \times j$ array of $k \times k$ matrices. $\mathrm{Tr}_A \rho_{AB}$ is just the sum of the matrices along the diagonal, and to get $\mathrm{Tr}_B \rho_{AB}$, you take the trace of each of the j^2 matrices.

The reason that this is called a *partial trace* is that if we take the partial trace of a tensor product matrix, say $M_A \otimes M_B$, then

$$\mathrm{Tr}_A(M_A \otimes M_B) = (\mathrm{Tr}\, M_A) M_B$$
$$\mathrm{Tr}_B(M_A \otimes M_B) = (\mathrm{Tr}\, M_B) M_A.$$

Let's take the partial trace for the example state we had earlier,

$$\frac{2}{\sqrt{5}} |\, 00\rangle_{AB} + \frac{1}{\sqrt{5}} |\, 11\rangle_{AB}$$

The density matrix is

$$\rho_{AB} = \frac{1}{5} \left(\begin{array}{cc|cc} 4 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 \end{array} \right)$$

To get $\mathrm{Tr}_A \rho_{AB}$, we add up the 2×2 matrices along the diagonal, which gives

$$\mathrm{Tr}_A \rho_{AB} = \frac{1}{5} \begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix} + \frac{1}{5} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}$$

To get $\mathrm{Tr}_B \rho_{AB}$, we take the trace of each of the 2×2 matrices in each quadrant. This gives

$$\mathrm{Tr}_B\, \rho AB = \frac{1}{5} \left(\begin{array}{c|c} \mathrm{Tr} \begin{pmatrix} 4 & 0 \\ 0 & 0 \end{pmatrix} & \mathrm{Tr} \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix} \\ \hline \mathrm{Tr} \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix} & \mathrm{Tr} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \right) = \frac{1}{5} \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}$$

For this case, this turns out to be the same density matrix as we obtained when we took the partial trace over A's system, because the original state $\frac{2}{\sqrt{5}} |\, 00\rangle + \frac{1}{\sqrt{5}} |\, 11\rangle$ is symmetric in A and B.

We now give another formula for the partial trace of a quantum state. This formula can be generalized to take the partial trace over a system that is a tensor product of more than two subsystems. Suppose you have a basis $\{|\, e_i\rangle\}$ for system A. Then

$$\mathrm{Tr}_A \rho = \sum_{i=0}^{d-1} \langle e_i \,|\, \rho \,|\, e_i \rangle \tag{2}$$

Why does this represent the same matrix as the first formula? What we mean by $\langle e_0 \,|$ if the systems are qubits is

$$\langle e_0 \,| = (1,0) \otimes I_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

And

$$\langle e_1 \,| = (0,1) \otimes I_2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Thus, if

$$M = \begin{pmatrix} P & Q \\ R & S \end{pmatrix},$$

where P, Q, R, S, are 2×2 matrices, we have

$$\langle e_0 \,|\, M \,|\, e_0 \rangle = P$$

and

$$\langle e_1 \,|\, M \,|\, e_1 \rangle = S,$$

so summing them gives the formula of Eq. (1).

There is another way of seeing this. If $\{|\, e_i\rangle\}$ is the standard basis $\{|\, 0\rangle, |\, 1\rangle, \ldots, |\, d-1\rangle\}$, over system A, then $\langle e_i \,|\, \rho \,|\, e_i \rangle$ just picks out the entries with i in the first coordinate of the matrix ρ_{AB}, and summing these gives the sum of the matrices along the diagonal.

Now, we will show that it doesn't matter which basis you use in the formula (2) for partial trace, you get the same result for $\mathrm{Tr}_A\, \rho$. Suppose we have two orthonormal bases for system A, $\{|\, e_j\rangle\}$ and $\{|\, f_i\rangle\}$. We can express one basis in terms of the other:

$$|\, f_i\rangle = \sum \alpha_{ij} \,|\, e_j\rangle\,.$$

Since both bases are orthonormal, the length of a vector expressed in the basis $\{|\, e_j\rangle\}$ must be the same as the length in $\{|\, f_i\rangle\}$. This means that the matrix (α_{ij}) is unitary, since this change of basis preserves the lengths of vectors. Now, we do some algebra:

$$\begin{aligned} \sum_i \langle f_i \,|\, \rho \,|\, f_i \rangle &= \sum_i \sum_{j'} \sum_j \alpha^*_{ij'} \langle e_{j'} \,|\, \rho \,|\, e_j \rangle \alpha_{ij} \\ &= \sum_{j'} \sum_j \langle e_{j'} \,|\, \rho \,|\, e_j \rangle \left(\sum_i \alpha^*_{ij'} \alpha_{ij} \right) \end{aligned}$$

The term $\sum_i \alpha^*_{ij'}\alpha_{ij}$ is just the inner product of the columns j and j', which is $\delta(j, j')$, i.e., it is 1 if $j = j'$ and 0 otherwise. Thus, we have

$$\sum_i \langle f_i \,|\, \rho \,|\, f_i \rangle = \sum_j \langle e_j \,|\, \rho \,|\, e_j \rangle,$$

which is the formula we wanted to prove.

So this shows that no matter what measurement Alice makes, Bob has the same density matrix.

Notes 8.370/18.435 Fall 2022

Lecture 14 Prof. Peter Shor

The Nobel Prize in Physics was given out this year to Alain Aspect, John Clauser, and Anton Zeilinger. In honor of that, we are going to explain the GHZ paradox, one of the many things that Anton Zeilinger won the Nobel Prize for on Tuesday. This paradox is named for its discoverers, Greenberger, Horne, and Zeilinger. In the next set of lecture notes, we will describe an experiment that Zeilinger did demonstrating the GHZ paradox.

These lecture notes cover something like half a class period, but I'm writing them up as a single lecture because it's easier that way (they were combined with a quite different subject).

The GHZ paradox can be viewed as a game. It wasn't at first, but computer scientists love putting things in terms of games, because this has been a very fruitful way of thinking about several aspects of computer science. So when computer scientists started looking at quantum paradoxes, this was a natural way to view them.

This game is played by three players, Alice, Bob, and Carol, who are all cooperating, and a referee who plays randomly. Alice, Bob, and Carol cannot communicate once the game starts, but they can meet beforehand and agree on a strategy. The referee gives Alice, Bob, and Carol each a bit, which is either an X or a Y. He always gives an odd number of Xs, so either there are three Xs or one X and two Ys. More specifically, he gives the three players XXX, YYX, YXY, and XYY with equal probabilities.

When the players get these bits, they have to give a bit back to the referee, which will either be a $+$ or a $-$. The referee looks at the bits he receives, and decides whether the players win. The rule is that if all three players get an X, they must return an odd number of $+$s. However, if two players get a Y and one an X, the must return an even number of $+$s.

Can the players win this game with a classical strategy? The answer is "no", they can only win $3/4$ of the time. One way they can do this is to always return an even number of Xs, so for instance they could all return Ys. Then they win unless the referee gives them XXX, which only happens $1/4$ of the time.

Why can't the players do better? Let's consider what happens when they use a deterministic strategy. It's fairly easy to use probability theory to show that a probabilistic strategy cannot do better than a deterministic one, but we won't include the proof in these notes.

So what is a deterministic strategy? It's a table telling what each of Alice, Bob, and Carol will return if the referee gives them an X or a Y. For example, this might be the strategy:

$player$	X	Y
A	$+$	$-$
B	$+$	$-$
C	$+$	$+$

In this strategy, if they get XXX, they return $+++$, so they win. If they get XYY they return $+-+$, so they win. Similarly, if they get YXY, they return $-++$, so they win.

But if they get YYX, they return $--+$, so they lose.

So let's consider the strategy table.

$player$	X	Y
A	a	d
B	b	e
C	c	f

There are only four possible challenges the referee can give the players, XXX YYX, YXY, XYY. The players' responses to these are abc, dec, dbf, aef, respectively. They must return an odd number of $+$'s in one case, and an even number of $+$'s in the other three. Thus, the set of responses

$$\{abc, dec, dbf, aef\}$$

must contain an odd number of $+$'s altogether. However, this is impossible, since each letter appears exactly twice in the sequence abc, dec, dbf, aef, so no matter how you assign $+$ and $-$ to the letters, the total number of $+$'s must be even.

Now, how can they win with probability 1 by using quantum mechanics? What they do is share what is called a GHZ state before the game. This is the state $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. Then each of the players measures the state in either the x or the y basis, depending on whether they were given on X or a Y.

Suppose they were given three X's. Then the probability they measure $|+++\rangle$ is

$$\frac{1}{\sqrt{2}^4}(\langle 0| + \langle 1|)(\langle 0| + \langle 1|)(\langle 0| + \langle 1|)(|000\rangle + |111)\rangle = \frac{1}{4}(1+1)$$

You can see that there are only two terms that appear in the final sum: $\langle 0|\langle 0|\langle 0|000\rangle$ and $\langle 1|\langle 1|\langle 1|111\rangle$, and they both contribute $\frac{1}{4}$ to the amplitude. The amplitude is thus $\frac{1}{2}$, and the probability of getting this oucome is its square, $\frac{1}{4}$. Similarly, the probability of getting the outcome $|+--\rangle$ is

$$\frac{1}{\sqrt{2}^4}(\langle 0| + \langle 1|)(\langle 0| - \langle 1|)(\langle 0| - \langle 1|)(|000\rangle + |111)\rangle = \frac{1}{4}(1+1),$$

because the two -1 coefficients from the two $|-\rangle$ multiply to give $+1$. However, if we have just one $-$, as in $|+-+\rangle$, then the amplitude coming from the term $|111\rangle$ would be -1, and the amplitude from the term $|000\rangle$ would still be 1. Adding these gives 0 meaning that the probability of seeing exactly one $-$ is 0.

It is fairly easy to see that if the players were given XYY, then the chance of seeing $+++$ is

$$\frac{1}{\sqrt{2}^4}(\langle 0| + \langle 1|)(\langle 0| + i\langle 1|)(\langle 0| + i\langle 1|)(|000\rangle + |111)\rangle = \frac{1}{4}(1-1),$$

as the two i's multiply to give -1. Thus the probability of seeing $|+++\rangle$ is 0. You need an even number of $+$s in an outcome to get $1+1$, and thus a chance of seeing that outcome.

Thus, if they are allowed to share a GHZ state, the players can measure it and win with probability 1.

Notes 8.370/18.435 Fall 2022

Lecture 15 Prof. Peter Shor

This year (2022), three experimentalists, Alain Aspect, John Clauser and Anton Zeilinger, were awarded the Nobel Prize for experiments showing that quantum mechanics really was non-local. They did their experiments using quantum optics. I want to describe one of these experiments, the GHZ experiment, performed by Zeilinger. However, first I need to explain some basic facts about quantum optics.

What we will do first is explain interferometers, and then explain the Elitzur-Vaidman bomb test. This is a very non-intuitive thought experiment (which has actually been carried out, although not with bombs). It was discovered in 1993, but it only uses quantum mechanics that was known 60 years earlier.

The premise of the test is that you have a lot of bombs. Their fuses are so sensitive that if a single photon strikes them, the bomb goes off. However, some bombs are missing fuses. . .if you shine a photon on the place where the fuse should be, the photon simply passes by unchanged. Your task is to identify some bombs which have intact fuses without exploding them (it's okay if you end up exploding some others).

Classically, it seems absolutely impossible. If you look to see whether there is a fuse there, the bomb explodes. However, if you don't look, you can't tell whether the bomb has a fuse or not. Quantum mechanically, however, it can be done. This is the *Elitzur-Vaidman bomb test experiment*, and it is one of a class of experiments called *interaction-free measurement.*

The key ingredient in this test is an interferometer. An interferometer is an experiment with (in the simplest case) two beam splitters, two mirrors, and two photodetectors. (See Figure.) The beam splitter is like a unitary transform. Assuming there's only

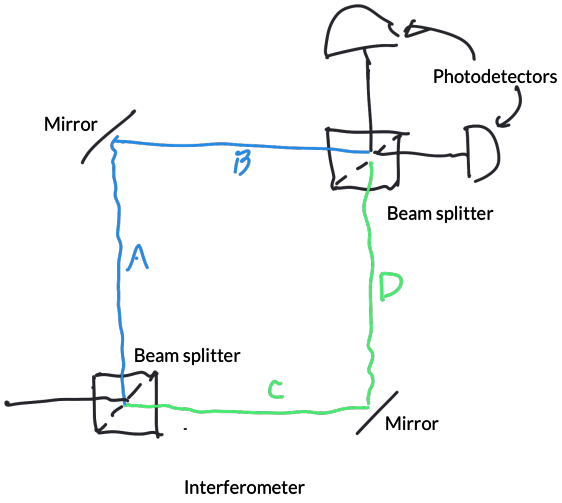


Figure 1: Elitzur Vaidman bomb detector experiment

one photon, this photon can be in a superposition of the horizontal and vertical inputs, and gets transformed into a superposition of the horizontal and vertical outputs. We will take the horizontal input and output to be the state $|0\rangle$ and the vertical input and output to be the state $|1\rangle$. The matrix that gives the action of the beam splitter is

$$M = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix},$$

This isn't the only way that a beam splitter can be described. By changing the bases, you can make it look much more like a Hadamard gate (do this as an exercise). However, this representation has the advantage of treating the vertical and horizontal paths on an equal footing.

Thus, suppose we put a photon in the horizontal input. After it goes through the first beam splitter, the state of the system is

$$M \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ i \end{pmatrix}.$$

The photon now travels along the superposition of paths AB and CD and enters the top right beam splitter. If the two paths are both an integer multiple of the wavelength of the light, the state of the photon going into the second beam splitter is the same as the state of the photon that came out of the first beam splitter, except the horizontal and vertical labels have been switched. Thus, the photon going into the second beam splitter is in the state $\frac{1}{\sqrt{2}} \begin{pmatrix} i \\ 1 \end{pmatrix}$. Multiplying by M again gives the vector $\begin{pmatrix} i \\ 0 \end{pmatrix}$, so it comes out horizontally and triggers the rightmost photodetector in the drawing.

You can similarly check that if the photon entered the first beam splitter on the vertical path, it would come out the second beam splitter vertically.

How can we use interferometers? Suppose one of the paths applies a phase $e^{i\theta}$ to the photon that the other does not. One way this can happen is if one of the paths is longer than the other. If the difference between the paths is d, then the longer one will acquire an extra phase of $e^{-id/\lambda}$, where λ is the wavelength of the photon. Michelson and Morley used an interferometer to try to measure the speed of the Earth, because under Newtonian mechanics, the path length would be different for photons moving perpendicular and parallel to the Earth's motion. However, because of Einstein's theory of relativity, the path lengths were exactly the same, and the experiment yielded a value of 0 for the Earth's motion. More recently, LIGO is using an interferometer to detect gravity waves—here the path length varies because gravity waves stretch and shrink space-time.

So what is the Elitzur-Vaidman bomb detection experiment? Suppose you have a factory that makes bombs. Each of these bombs has a fuse that is so sensitive that the bomb explodes if a single photon hits it. However, some of these bombs are defective—they are missing fuses. Your mission is to find some bombs that are guaranteed not to be defective. With classical physics, this is clearly impossible; if you shine a photon on a fuse to see whether it's there, the bomb explodes (unless you "cheat" by doing something like weighing the bombs). However, quantum mechanically, you can solve this problem.

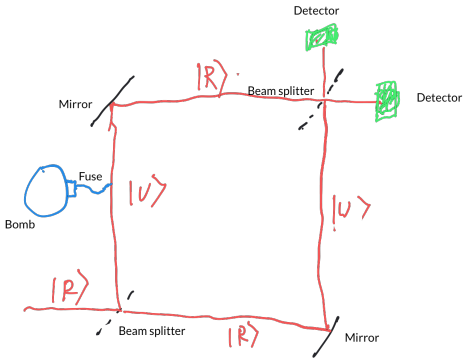


Figure 2: GHZ experiment. The R implements the quantum gate $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$.

What you can do is position a bomb so that the fuse overlaps the 'A' arm of the interferometer. Suppose the bomb is defective. Then, the photon behaves as it would when both the AB and CD paths are unobstructed, so it always triggers right photodetector.

Now, suppose you have a bomb. By exploding, the bomb measures the path the photon took. The photon is in a superposition of taking the AB path and the CD path. So if the bomb doesn't explode, then the state collapses and the photon is on the CD path. When it hits the second beam splitter, it comes out in an equal superposition of the vertical and horizontal exists, so with probability $\frac{1}{2}$, it triggers each of the photodetectors.

Thus, if the fuse was defective, the rightmost photodetector is always triggered, while if the fuse was good, the bomb explodes $\frac{1}{2}$ of the time, the top photodetector is triggered $\frac{1}{4}$ of the time, and the rightmost photodetector is triggered $\frac{1}{4}$ of the time.

So when the top photodetector is triggered, you know that the fuse is not defective.

This experiment has actually been carried out, although not with real bombs.

There is a variation on this experiment where you can make the bomb will explode an arbitrarily small fraction of the time (not 0) and still get a good yield of bombs guaranteed to be non-defective. It's quite a bit more complicated, so we won't go into it.

There are some more quantum optical elements we need to discuss before we can explain the GHZ experiment. One of them is a polarizing beam splitter. This is a device that transmits horizontally polarized photons and reflects vertically polarized photons.

Another one of them is a parametric downconverter. This is a crystal which, when you send a beam of light with frequency ν, you get as output, in addition to a slightly attenuated beam of light with frequency ν, two beams of light with frequency $\nu/2$. What is happening (to give a slightly simplified explanation) is that some of the photons of frequency ν are splitting into pairs of photons of frequency $\nu/2$. Recall that a photon's energy is proportional to its frequency, so energy is conserved. Further, these two photons are entangled, being in the EPR state $\frac{1}{\sqrt{2}}(|\leftrightarrow\updownarrow\rangle - |\updownarrow\leftrightarrow\rangle)$.

With just a parametric downconverter, beam splitters, and polarizing beam splitters, it is fairly straightforward to set up an experiment that shows violations of Bell's inequality. I'll let you figure out how this works.

The experiment becomes much more difficult to implement if you want to close all the loopholes. You have to prove beyond all doubt that the detectors aren't communicating somehow; the way to do that is to make sure the settings of the detectors are changed fast enough that the information from one about the setting of one can't reach the other in time, even if it's transmitted at the speed of light. This requires quite a bit more sophistication on the experimenter's part.

However, for the GHZ experiment, we need three entangled photons, and parametric downconverters only produce two. How can we possibly entangle three photons with just a parametric downconverter and the optical elements we know?

What Pan, Bouwmeester, Daniell, Weinfurter and Anton Zeilinger did was to use the apparatus in the figure below. You need to illuminate the parametric downconverter at a high enough intensity so that you will occasinally get two simultaneous downconversions (creating four photons in two entangled pairs), but not a high enough intensity that there are many simultaneous downconversions of three photons.

Now, you only count instances where four detectors (at the end of each of the four paths) register photons. There are two entangled photons, and each of these entangled pairs sends one photon down path a and the other down path b.

For the Detector 1, you can only detect horizontal photons, because only those pass through the polarizing filter. This means that the other photon that went down path a was a vertical one, as otherwise it would have also gone through the polarizing beam splitter and into Detector 1. This photon goes through the gate R and turns into $\frac{1}{\sqrt{2}}(|\leftrightarrow\rangle + |\updownarrow\rangle)$. Now, before the photon goes through the gate R, there must be an equal number of horizontally and vertically polarized photons (because they form two EPR pairs). This means the photons must be in the state $|\leftrightarrow\updownarrow\leftrightarrow\updownarrow\rangle$ or $|\leftrightarrow\updownarrow\updownarrow\leftrightarrow\rangle$ And in fact, it is easy to show that both of these states have the same phase.

Now, there must be two photons that approach the central polarizing beam splitter, and these two photons must have the same polarization (otherwise they would both end up at only one of the two central detectors). Thus, the first three photons must either be in the state $|\leftrightarrow\leftrightarrow\leftrightarrow\rangle$ or $|\leftrightarrow\updownarrow\updownarrow\rangle$. Since we know that the second photon was originally $|\updownarrow\rangle$ before it went through the gate R, and there were two of each horizontal and vertical polarizations, the only two possibilities are. $|\leftrightarrow\leftrightarrow\leftrightarrow\updownarrow\rangle$ or $|\leftrightarrow\updownarrow\updownarrow\leftrightarrow\rangle$. Thus, the state of the last three photons are

$$\frac{1}{\sqrt{2}}(|\leftrightarrow\leftrightarrow\updownarrow\rangle + |\updownarrow\updownarrow\leftrightarrow\rangle),$$

which is essentially a GHZ state. If we apply a NOT gate to the last of these photons,

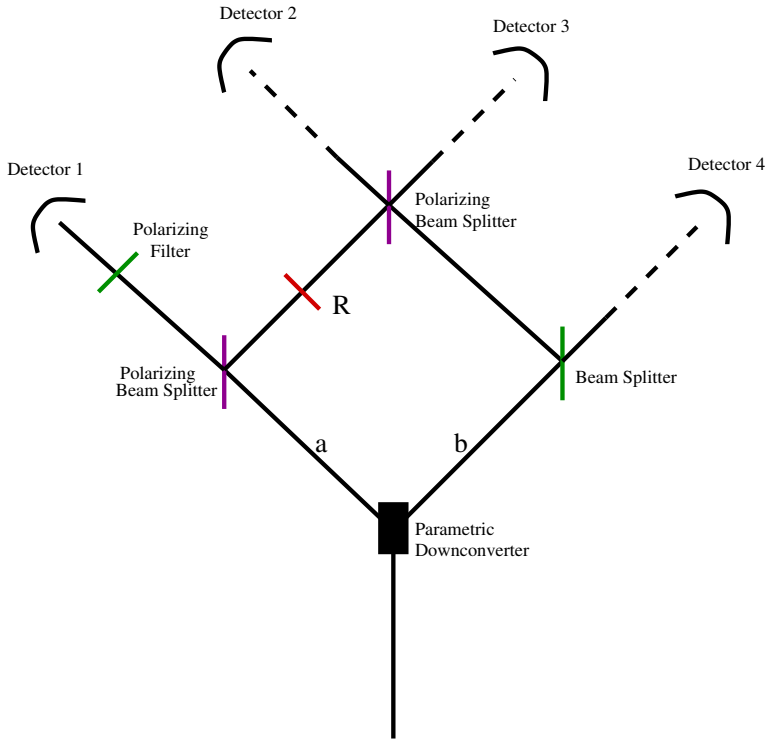


Figure 3: Mach-Zehnder Interferometer

we get

$$\frac{1}{\sqrt{2}}(|\leftrightarrow\leftrightarrow\leftrightarrow\rangle + |\updownarrow\updownarrow\updownarrow\rangle),$$

We still need to add some quantum optics elements before the detectors, in the space where the dashed lines are, to measure the GHZ states in the correct basis. Note that this means we don't actually have to apply a NOT gate to the photon which will hit Detector D, since we can adjust these elements to compensate for the lack of a NOT gate.

Notes 8.370/18.435 Fall 2022

Lecture 16 Prof. Peter Shor

Today, we covered the Deutsch-Jozsa algorithm.

Before talking about the Deutsch-Jozsa algorithm, I'm going to explain the Hadamard transform, which is a necessary ingredient both for the Deutsch-Jozsa algorithm and Simon's algorithm, which we will do next. Recall the Hadamard gate $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. The Hadamard transform is simply $H^{\otimes n}$. That is, you do a Hadamard gate on each of n qubits.

What does $H^{\otimes n}$ do to a quantum state?

First, let's do an easy example

$$\begin{aligned} H^{\otimes n} |0^n\rangle &= \frac{1}{2^{n/2}} \big(|0\rangle + |1\rangle \big) \big(|0\rangle + |1\rangle \big) \dots \big(|0\rangle + |1\rangle \big) \\ &= \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle . \end{aligned}$$

Here, $|j\rangle$ means the bitstring in $\{0,1\}^n$ that represents the integer j in binary. For example, if $n = 4$, then $|12\rangle = |1100\rangle$ and $|7\rangle = |0111\rangle$. When we apply the distributive law to $\big(|0\rangle + |1\rangle \big) \big(|0\rangle + |1\rangle \big) \dots \big(|0\rangle + |1\rangle \big)$, we get every n-bit binary string exactly once, and so we get every integer $|j\rangle$ from $|0\rangle$ to $|2^n - 1\rangle$.

Now, what happens when we apply the Hadamard transform to an arbitrary bit string? For example,

$$\begin{aligned} H^{\otimes 5} |01011\rangle &= \frac{1}{\sqrt{32}} \big(|0\rangle + |1\rangle \big) \big(|0\rangle - |1\rangle \big) \big(|0\rangle + |1\rangle \big) \big(|0\rangle - |1\rangle \big) \big(|0\rangle - |1\rangle \big) \\ &= \frac{1}{\sqrt{32}} \big(|00000\rangle - |00001\rangle - |00010\rangle + |00011\rangle + |00100\rangle - \dots - |11111\rangle \big) \end{aligned}$$

Here, for example, we see that $|11111\rangle$ has a minus sign in the expansion because there are three $-$'s in the five $|1\rangle$ terms we tensor together to get $|11111\rangle$.

We see from the above that each binary string representing integers k between 0 and 31 appears in the sum with amplitude $2^{-n/2}$. The question is; what is the sign on $|k\rangle$. Let $j \cdot k$ be the dot product of j and k, i.e., the number of places where 1s appear in both j and k. I claim that

$$H^{\otimes n} |j\rangle = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} (-1)^{j \cdot k} |k\rangle .$$

Why do we get the phase $(-1)^{j \cdot k}$ in the sum above. Let's look at our example above. What would the phase on $|11001\rangle$ be in $H^{\otimes 5} |01011\rangle$? To get $|11001\rangle$, we take the $|1\rangle$ from the first, second, and fifth terms, and the $|0\rangle$ from the second and third terms. This gives

$$\frac{1}{\sqrt{32}} \big(|1\rangle \big) \big(-|1\rangle \big) \big(|0\rangle \big) \big(|0\rangle \big) \big(-|1\rangle \big) = \frac{1}{\sqrt{32}} |11001\rangle$$

We get a minus sign only when we choose a $|1\rangle$ (which means there is a 1 bit in that spot in $|k\rangle$) and when there is a $-$ sign on the $|1\rangle$ (which means there is a $|1\rangle$ in that position in $|j\rangle$). So the number of $-$ signs is just the number of places which have a 1 in both k and j. This is $j \cdot k$. Further, multiplying all the $-$ signs together gives the term $(-1)^{j \cdot k}$ given in our formula.

Next, we describe the Deutsch-Jozsa algorithm.

The Deutsch-Jozsa algorithm operates on a function mapping binary strings of length n into bits. First, let's give a couple of definitions.

Definition 1 *A function is* constant *when* $f(x) = f(y)$ *for all* x, y *in* $\mathrm{Domain}(f)$.

Definition 2 *A function is* balanced *when there are an equal number of inputs that produce a* 0 *and that produce a* 1.

For example, the function on two bits:

$$f(00) = 0; \quad f(01) = 1 \quad f(10) = 1 \quad f(11) = 0$$

is balanced.

The Deutsch-Jozsa algorithm takes a function which is either constant or balanced, and tells which it is.

How many queries to the function does it take a classical computer to solve this problem? We will require that it solve it deterministically. This is equivalent to the problem: you have 2^n balls in an urn, and the balls are either black or white. You know that either all the balls are the same color, or half of them are each color. How many balls do you need to take to be know for certain which case holds?

The answer is that you need to take $2^{n-1} + 1$, or one more than half the balls. Suppose you start drawing balls and you draw 2^{n-1} of them that are black. You can't know for certain that all the other balls in the urn are white, and that you were very unlucky in which balls you chose. Thus, classically, it may take evaluating the function on $2^{n-1} + 1$ different inputs to be sure you have the right answer.

Now, we will assume that the function f is given to us in the form of some circuit or black box (we will call it an *oracle*) that takes

$$O_f |x\rangle = \begin{cases} |x\rangle & \text{if } f(x) = 0 \\ -|x\rangle & \text{if } f(x) = 1 \end{cases}$$

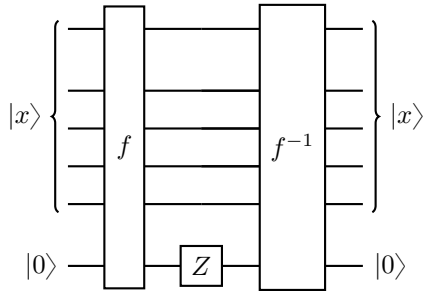
This is a *phase oracle* because it encodes the information about the function in the phase. A *bit oracle* computes the result of the function in a separate register.

$$O_f |x\rangle |0\rangle = |x\rangle |f(x)\rangle \ .$$

Here the first register has n qubits and the second has one qubit.

We will show that if you have a bit oracle for a function f, you can construct a phase oracle for this function. It actually turns out that these two kinds of oracles are equivalent, but we will not prove that now.

How can we construct a phase oracle if we are given a bit oracle? What we do is first compute $|f(x)\rangle$ in a second register. We then apply a Z gate to this register to get $(-1)^{f(x)}$, and then uncompute $|f(x)\rangle$. The quantum circuit is as below:



After the first gate, we have $|x\rangle\,|f(x)\rangle$. After the Z gate, we get $(-1)^{f(x)}\,|x\rangle\,|f(x)\rangle$. The last gate uncomputes f, so we get $(-1)^{f(x)}\,|x\rangle\,|0\rangle$, which is the phase oracle (aside from the work bit that ends up in the same state as it started in, and thus can be disregarded).

The Deutsch algorithm is fairly straightforward:

Step 1: Start with $|0^n\rangle$,

Step 2: Apply the Hadamard transform $H^{\otimes n}$,

Step 3: Apply the phase oracle O_f,

Step 4: Apply the Hadamard transform $H^{\otimes n}$,

Step 5: Measure the quantum state.

Now let's see what happens in this algorithm. We've already computed much of what we need for these manipulations in our discussion of the Hadamard transform.

$$H^{\otimes n}\,|0\rangle = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle$$

$$O_f H^{\otimes n}\,|0\rangle = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} (-1)^{f(j)}\,|j\rangle$$

$$H^{\otimes n} O_f H^{\otimes n}\,|0\rangle = \frac{1}{2^n} \sum_{j=0}^{2^n-1} \sum_{k=0}^{2^n-1} (-1)^{f(j)} (-1)^{j\cdot k}\,|k\rangle$$

Now, let's compute the probability of seeing the state $|0^n\rangle$ (i.e., $k = 0$) after the measurement. This probability is just the square of the amplitude on the $|0^n\rangle$ state.

This amplitude is just

$$\frac{1}{2^n} \sum_{j=0}^{2^n-1} (-1)^{f(j)}$$

because $j \cdot 0 = 0$.

If f is constant, then $f(j) = 0$ for all j or $f(j) = 1$ for all j. The sum is then either $+1$ or -1, so the probability of seeing $|0^n\rangle$ is 1.

If the function is balanced, then an equal number of $+1$ and -1 terms appear in the sum $\sum_{j=0}^{2^n-1}(-1)^{f(j)}$, so this sum is 0, and the probability of seeing $|0^n\rangle$ is 0.

So if the output is $|0^n\rangle$, we conclude that the function is constant, and if the output is anything else we conclude that the function is balanced. This computation takes one call to the oracle and a linear number of gates in n, while the best deterministic classical algorithm takes around $2^{n/2}$ gates. Thus, for this problem, the best deterministic quantum algorithm is exponentially faster.

The Deutsch-Jozsa algorithm was not a very convincing argument that quantum computation was more powerful than classical computation. While you need around $2^{n/2}$ queries to tell with absolute certainty whether a function is balanced or constant, if you make 20 or 30 queries, the chances that you will be wrong are very, very small. And randomized algorithms are a well-studied area of computer science, and are generally considered almost as good as deterministic algorithms. The reason that the Deutsch-Jozsa algorithm is important is that it led to Simon's algorithm, which makes a much more convincing argument that quantum computation is more powerful.

Notes 8.370/18.435 Fall 2022

Lecture 17 Prof. Peter Shor

Today, I'm going to talk about classical complexity theory, and then I'll talk about oracles. I'll explain some ways in which oracles are used in classical theoretical computer science, and then talk about how quantum oracles work.

Theoretical computer scientists loosely consider an algorithm "efficient" if it runs in time that is a polynomial function of the length n of its input and "inefficient" if it runs in time super-polynomial in the length of its input. Here, what running time means is simply the number of elementary steps the program takes. This is clearly wrong, in terms of actual intuitive notions of efficient and inefficient. An algorithm that takes time n^{24} will only run in a reasonable time for very small input sizes n, and there are several algorithms, like the simplex algorithm, which have a worst-case running time that is exponential in the input size, but which actually run fast nearly all the time.

So how did this state of affairs develop?

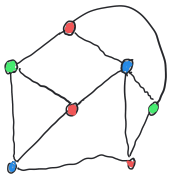
Back in the early days of computing, there were many different kinds of models of computational machines. The first mathematical model of a computing machine developed was put forward by Alan Turing, in 1936, a decade or so before the advent of actual electronic computers. This is a *Turing machine*, and it is still one of the main models used in classes on complexity theory. Another model is a random-access machine (RAM), which is closer to what a real-life computer looks like. When you're talking about abstract models of computation, you'd like to define complexity classes that are independent of the exact details of machine you're running on. But an algorithm that runs in n time on a RAM might take n^2 time on a Turing machine. So several computer scientists, most notably Alan Cobham, decided to define the complexity class P of problems which could be solved with an algorithm that ran in polynomial time. The difference between the different computational models was in all the cases that these computer scientists considered only a polynomial factor, so this difference was not relevant for the question of whether a problem is in P.

There is a larger class than P, namely BPP.[1] This is the set of problems that can be solved with high probability with a randomized algorithm that runs in polynomial time. For quantum complexity classes, the most relevant complexity class is BQP, the class of problems that can be solved on a quantum computer with high probability in polynomial time. Here, because quantum computation is inherently probabilistic, the most natural class is randomized; while people have defined deterministic quantum complexity classes, these classes appear to depend on the exact details of the definition of a quantum computer, and are thus less natural.

We now want to talk about the complexity class NP. This is the set of problems that can be easily verified. An example is three-colorability. A graph is three-colorable if its vertices can be colored so that no edge connects two vertices of the same color (see the figure below for an example).

[1] although it has not been proved larger yet, and some theoretical computer scientists think it's the same

Figure: A 3-colorable graph.

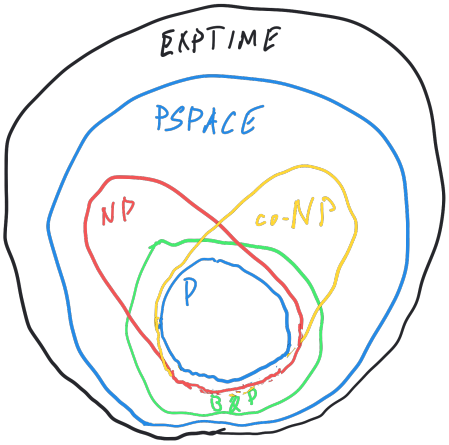


If you're given a graph, there is no polynomial time algorithm known that will tell whether this graph is three-colorable. However, if you're given a graph and a coloring, it is easy to check whether the coloring is a three-coloring of the graph.

Formally, a problem is in NP if there is a program that runs in polynomial time, which takes as input the problem instance and another input called the "witness", and outputs "accept" or "reject". For a problem instance that does not have the property (being three-colorable in our example), this program must always output "reject", and for one that does, there must always be some witness that will make the program output "accept". For instance, for three-colorability, you would input the graph (the problem instance) and a three-coloring of the graph (the witness); the program would check that there are no edges with both endpoints colored by the same color, and output "accept" if this holds.

Some problems in NP, (three-colorability is one) are NP-complete. We can prove that if you can solve one of these problems in polynomial time, you can solve all problems in NP in polynomial time. How can you do that? First, you need to find a master problem that is NP-complete. The most commonly used problem for this is 3-SAT. I'm not going into the details, but you can express any computation as a Boolean formula, and you can turn any Boolean formula into a 3-SAT problem. So this shows that any problem in NP can be turned into a 3-SAT problem.

Now, to show that 3-colorability is NP-complete, you need to show that for any 3-SAT formula, you can find a graph that's only larger by a polynomial factor, and for which the graph is 3-colorable if and only if the 3-SAT formula is satisfiable.



So where does BQP fall with respect to the complexity class NP? It is not believed that BQP is contained in NP, but we also do not believe that quantum computers can solve NP-complete problems in polynomial time. Thus, the general belief among theoretical computer scientists is that they are incomparable. This can be seen in the following Venn diagram. Here, co-NP is the complement language of NP: problems for which a negative solution can be verified. PSPACE is the set of problems that have algorithms which will solve them with polynomial space, or memory, rather than time. And EXPTIME is the set of problems that can be solved in time exponential in their input size. It turns out that quantum PSPACE is the same as PSPACE, so quantum computing cannot reduce the amount of space needed to solve a problem by more than a polynomial factor. The only two complexity classes in the diagram which theoretical computer scientists can prove are different are P and EXPTIME, which shows how hard it is to prove that two complexity classes are different.

The rest of this lecture was spent introducing the quantum Fourier transform, and I will write this part up in the lecture notes for Wednesday's lecture.

Notes 8.370/18.435 Fall 2022

Lecture 18 Prof. Peter Shor

On Friday, we covered Simon's algorithm. Simon's algorithm is a quantum algorithm that solves Simon's problem. This isn't a problem that arose from any practical application, but a problem that was invented to show that quantum computers could be faster than classical computers.

What is Simon's problem? We are given a function f that maps length-n binary strings to length-n binary strings. This function has the property that it is 2-to-1, and that there is some binary string c such that

$$f(x) = f(x \oplus c)$$

where $\oplus$ is bitwise exclusive or. For example, such a function might be

$$\begin{array}{ll} f(000) = 101 & f(100) = 011 \\ f(001) = 010 & f(101) = 100 \\ f(010) = 011 & f(110) = 101 \\ f(011) = 100 & f(111) = 010 \end{array}$$

Simon's problem is to find c, which is 110 in the above example.

How can we find c? First, let's look at classical algorithms for the problem. If we find two function inputs x_1 and x_2 so $f(x_1) = f(x_2)$, then $c = x_1 \oplus x_2$. How can we do this? The obvious way is to try inputs at random until we find such a pair.

Let's try an example. By symmetry, it doesn't matter which input we test first, so let's try 000. And let's try 001 for the second input. We've discovered two function values 101 and 010, and because these are different, can conclude that $c \neq 001$.

Now, let's try $f(010)$. This gives another distinct output, and now we can see that f is not

$$\begin{aligned} 000 \oplus 001 &= 001 \\ 000 \oplus 010 &= 010 \\ 001 \oplus 010 &= 011 \end{aligned}$$

Let's try one more input value. if we choose 011, we won't eliminate any possible values of c. but if we try 101, we will eliminate three more values of c,

$$\begin{aligned} 000 \oplus 101 &= 101 \\ 001 \oplus 101 &= 100 \\ 010 \oplus 101 &= 111 \end{aligned}$$

So the most number of values we can eliminate on the jth call to the function is $j - 1$. This means after t function evaluations, we have eliminated at most

$$\sum_{j=1}^{t} (j-1) = \frac{t(t-1)}{2}$$

possible valued of c.

This says to be sure of finding c, we need $t(t-1)/2 \geqslant 2^n - 1$, or $t \approx 2^{(n+1)/2}$.

Now, let's think about randomized algorithms. Suppose that f is a random function with Simon's property. Before we've found c, the value of c is equally likely to be any of the values that we have not yet eliminated. Thus, for us to have a fifty percent chance of finding c, we need to eliminate roughly half the possible values of c, which gives $t^2 \approx 2^{n-1}$. So the median running time of the best randomized classical algorithm for this problem is somewhere around $2^{n/2}$.

What about a quantum algorithm for this problem? Before we can give the quantum algorithm, we need to say how the function is given to the quantum computer. We will use an oracle O_f that behaves as

$$O_f |x\rangle |z\rangle = |x\rangle |z \oplus f(x)\rangle .$$

If $z = 0$, then the function computes $f(x)$ in the second register. We arrange the oracle this way so that it will be reversible, because functions on quantum computers have to be reversible.

Also recall that the Hadamard transform $H^{\otimes n}$ takes a binary string $|j\rangle$ of length n to a superposition of all binary strings of length n:

$$H^{\otimes n} |j\rangle = \sum_{k=0}^{2^n-1} (-1)^{j \cdot k} |k\rangle$$

We can now give Simon's algorithm. We start with two registers of n qubits each, initialized in the state $|0^n\rangle$:

$$|0^n\rangle |0^n\rangle .$$

We now apply a Hadamard transform to the first register:

$$\frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle |0^n\rangle ,$$

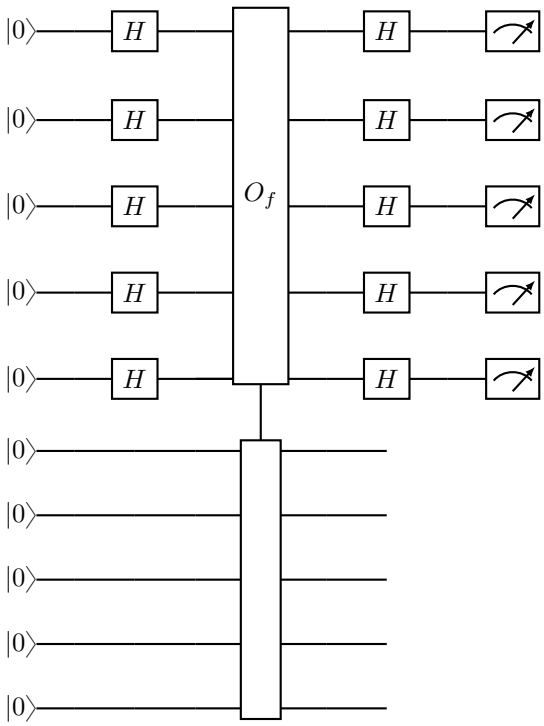
where $|j\rangle$ is the representation of the integer j in binary. We next apply the function O_f:

$$\frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle |f(j)\rangle .$$

Now, we apply another. Hadamard transform to the first register:

$$\frac{1}{2^n} \sum_{j-0}^{2^n-1} \left(\sum_{k=0}^{2^n-1} (-1)^{j \cdot k} |k\rangle \right) |f(j)\rangle = \frac{1}{2^n} \sum_{k=0}^{2^n-1} \left(\sum_{j=0}^{2^n-1} (-1)^{j \cdot k} \right) |k\rangle |f(j)\rangle$$

And finally, we measure the registers (although we will only use the result from the first register).



To analyse the algorithm, we need to compute the probability of seeing a pair of values $|k\rangle|f(j)\rangle$ in the registers. This will be equal to the square of the amplitude on this state. First, note that there are only two values of j that will produce $f(j)$; these are j and $j \oplus c$. So there are only two terms in the sum over j that contribute to this amplitude, and the amplitude is:

$$\frac{1}{2^n}\left((-1)^{j\cdot k} + (-1)^{(j\oplus c)\cdot k}\right).$$

We now use the distributive law $(r \oplus s)\cdot t = r\cdot t \oplus s\cdot t$, and we get

$$\begin{aligned}\frac{1}{2^n}\left((-1)^{j\cdot k} + (-1)^{(j\oplus c)\cdot k}\right) &= \frac{1}{2^n}\left((-1)^{j\cdot k} + (-1)^{j\cdot k \oplus c\cdot k}\right)\\ &= \frac{1}{2^n}\left((-1)^{j\cdot k} + (-1)^{j\cdot k}(-1)^{c\cdot k}\right)\\ &= \frac{1}{2^n}(-1)^{j\cdot k}\left(1 + (-1)^{c\cdot k}\right)\end{aligned}$$

But if $c\cdot k = 1$, this expression is 0, and if $c\cdot k = 0$, this expression is $\pm\frac{2}{2^n}$. We thus have that the probability of seeing a value k is 0 if $c\cdot k = 0$, that is, if k is not perpendicular to c, and $\frac{1}{2^{2n-2}}$ if k is perpendicular to c.

This lets us use linear algebra over $\mathbb{Z}_2$ to find c. Suppose we repeat the process above to find $n-1$ linearly independent vectors perpendicular to c. If you have $n-1$

vectors in an n-dimensional space, there is only one vector perpendicular to them. This is c; all we need to do is find it.

Linear algebra over a finite field (like $\mathbb{Z}_2$) is different in several ways from linear algebra over $\mathbb{R}$ or $\mathbb{C}$. The most significant difference is that a vector can be perpendicular to itself; for example, $1010011 \cdot 1010011 = 4 = 0 \pmod 2$. However, many of the techniques and theorems of linear algebra over $\mathbb{Z}_2$ are the same as those over $\mathbb{C}$. Rather than explaining the differences in detail, I'll just demonstrate how we find c for an example. It should be straightforward to deduce the general algorithm from the example, and you should be able to see why it always works.

Suppose we have some vectors perpendicular to c. First, we need to check that they are all linearly independent (or eliminate the linearly dependent ones, if there are any). We use Gaussian elimination. Let's say we have the vectors

$$\begin{aligned} k_1 &= 11011 \\ k_2 &= 01011 \\ k_3 &= 01111 \\ k_4 &= 11010\,. \end{aligned}$$

Our first step will be to make the first column all 0's, except for the top row. We can do this by subtracting (i.e., XORing) the first row with any row that begins with 1. This gives $k_4' = k_4 \oplus k_1$, and we have:

$$\begin{aligned} k_1 &= 11011 \\ k_2 &= 01011 \\ k_3 &= 01111 \\ k_4' &= 00001\,. \end{aligned}$$

Now, we do the same for the second column, for all rows below the 2nd row. This gives $k_3' = k_3 \oplus k_2$, and we have:

$$\begin{aligned} k_1 &= 11011 \\ k_2 &= 01011 \\ k_3' &= 00100 \\ k_4' &= 00001\,. \end{aligned}$$

Now we're done, because this matrix is in row-echelon form, and we see that the original four vectors were linearly independent. However, in general we might have to use this procedure on all the columns.

What we do now is figure out which vector is perpendicular to all these rows. Looking at the last row, k_4', we see that the last coordinate of c must be 0, so $c = ???0$. We don't have any rows of our matrix that start $0001?$, which means that the fourth coordinate isn't constrained—it could be either 0 or 1. if we take it to be 0, we will end up with the 0 vector, so we take it to be 1, and we have $c = ??10$. Now, the fact that c must be perpendicular to k_3' means that the third coordinate is 0, so we get $c = ?010$. Making c perpendicular to the second row, k_2, gives $c = 1010$, and making

it perpendicular to the first row gives $c = 01010$. And indeed, you can check that $c = 01010$ is perpendicular to every row in our first matrix.

Note that in general, if you have $n-1$ linearly independent vectors perpendicular to c, and you carry out this procedure, all but one of the coordinates of c will be determined by a row in the row-echelon matrix, and this procedure will give you a unique non-zero vector c.

How long does this algorithm take? That is, how many vectors will we have to sample to find $n-1$ vectors that are linearly independent. For the first step, there is only one vector that is linearly dependent ($000\ldots0$), and there are 2^{n-1} possible vectors perpendicular to c. Our calculations showed that we find each of them with equal probability, so the probability of the first vector not being linearly independent is $\frac{1}{2^{n-1}}$. On the second step, there are two vectors that are not linearly independent (k_1 and 0), so the probability of the second vector not being linearly independent is $\frac{1}{2^{n-2}}$. And in general, the probability that the ith vector is linearly dependent on the first $i-1$ is $\frac{1}{2^{n-i}}$. All of these probabilities are less than $\frac{1}{2}$, so the expected number of tries it takes to get a linearly independent vector on the ith step is at most $\frac{1}{2}$, and we see that the expected number of steps is less than $2n$. Since each step takes $O(n)$ time, the total time taken is $O(n^2)$. This contrasts with the expected running time of at least $2^{n/2}$ for the classical algorithm, giving an exponential advantage for quantum computation.

In fact, with a more clever analysis you can show that the expected number of steps is always less than $n+2$.

Now, let's look at the circuit for Simon's algorithm more closely. You will notice that we first do a Hadamard transform on the first register. We then computer take the value of $|x\rangle$ in the first register and compute $|f(x)\rangle$ in the second register. However, we never look at the second register again. I don't know how many of you know about optimizing compilers, but a classical optimizing compiler would look at this circuit, and ask: why are you computing $f(x)$ in the second register if you never look at the value again? (And in fact, this was one of the questions that was asked in class on Friday.) So why is the computation of the oracle necessary for the algorithm?

It is easy to see that it is necessary; without the oracle, you would just be applying an H gate to each wire in the first register, and then applying another H gate to these wires. Since $H^2 = I$, we would have a circuit that did nothing. But how can computing the value of $|f(x)\rangle$ in the second register change the values of the first register. This is an example of a process called *back action*.

Back action is a fundamental principal of quantum mechanics. It says that any time two systems interact, if the first system has an effect on the second, then the second also has an effect on the first. Maybe for this course, the best illustration of back action is the CNOT gate.

Recall that the Hadamard gate has the following properties:

$$\begin{aligned} H^2 &= I \\ H\sigma_z H &= \sigma_x \\ H\sigma_x H &= \sigma_z \end{aligned}$$

Now, we will deal with an identity involving CNOT and Hadamard gates. First,

let's consider the CNOT gate, and input the state $|-\rangle|-\rangle$. What happens?

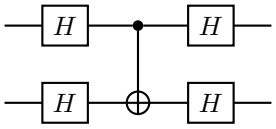
$$\begin{aligned}|-\rangle|-\rangle &= \frac{1}{2}\big(|0\rangle - |1\rangle\big)\big(|0\rangle - |1\rangle\big) \\ &= \frac{1}{2}\big(|00\rangle - |01\rangle - |10\rangle + |11\rangle\big)\end{aligned}$$

So because CNOT$|10\rangle = |11\rangle$ and CNOT$|11\rangle = |10\rangle$, we get

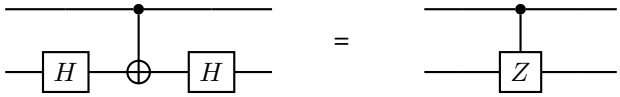
$$\begin{aligned}CNOT\frac{1}{2}\big(|00\rangle - |01\rangle - |10\rangle + |11\rangle\big) &= \frac{1}{2}\big(|00\rangle - |01\rangle - |11\rangle + |10\rangle\big) \\ &= \frac{1}{2}\big(|0\rangle + |1\rangle\big)\big(|0\rangle - |1\rangle\big) = |+\rangle|-\rangle.\end{aligned}$$

Here, the target qubit has stayed the same and the control bit has changed. This is the opposite of what you might think should happen. Classically, when you apply an "if" statement on variable A, and use the results to modify variable B, then variable A doesn't change. Here, a CNOT looks like an "if" statement, but the value of variable A has changed. We'll say a little more about this phenomenon at the end of these lecture notes.

Let's look at the quantum circuit

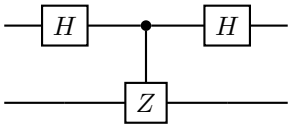


What does this do? First, let's look at a piece of this circuit: We will show that



If the top wire starts in $|0\rangle$, the CNOT acts as identity on the bottom wire, and we get $HH = I$. If the top wire starts in $|1\rangle$, the CNOT acts as a σ_x on the bottom wire, and we get $H\sigma_x H = \sigma_z$. Thus, the above circuit is a controlled σ_z, or a C-Z.

We can make this substitution in our original circuit, to get the equivalent circuit

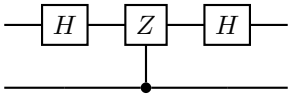


However, a C-Z with the first qubit as the control and the second qubit as the target is the same as a C-Z with the second qubit as the control and the first qubit as the target. You can see this by noticing that the action on the two qubits is symmetric—a phase of -1 is applied if and only if the state is $|11\rangle$. You can also see this by matrix

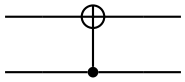
multiplication. You can compute the action of the gate with the qubits swapped using SWAP · C-Z · SWAP = C-Z where

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \text{C-Z} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

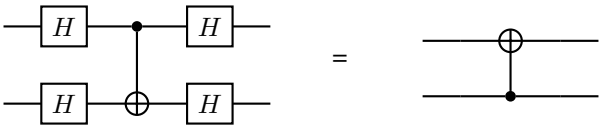
Thus, the circuit is equivalent to



But we've just computed that this is the same as:



So now we've shown that putting two H gate before and after a CNOT reverses the direction of the CNOT:



This is actually an example of a general principle in quantum mechanics: if system A has some effect on system B, then system B will also have an effect on system A. The effect of system B on system A is called "back-action". One consequence of this is that you cannot measure a quantum system without also affecting the quantum system being measured; in a standard von Neumann measurement, the system being measured is projected onto one of a set of quantum states, or a set of subspaces. However, there are more general kinds of measurement, that we won't discuss in this class, and the principle that there will always be a back-action still applies.

Notes 8.370/18.435 Fall 2022

Lecture 19 Prof. Peter Shor

In class, I Ialked a little bit about Fourier series and the discrete Fourier transform, to give some motivation for the quantum Fourier transform. In the interest of getting these lecture notes out on time, I'm not going to put these into the notes right now (I actually did the same thing last year). I may come back and revise it.

The quantum Fourier transform will be very useful in a number of quantum algorithms, which we will be covering in class. Namely, we will use them in the phase estimation, the factoring, and the discrete logarithm algorithms.

The quantum Fourier transform is very similar to the discrete Fourier transform. The Fast Fourier Transform (FFT) is an algorithm for computing the discrete Fourier transform, and if you've seen the FFT (and you remember it), you will realize that it shares many elements with the quantum Fouriet transform. In fact, the discrete Fourier transform takes the amplitudes of the input to the quanutm Fourier transform to the amplitudes of the output.

The quantum Fourier transform takes input $|j\rangle$ to

$$|j\rangle \longrightarrow \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} e^{2\pi i\, jk/n} |k\rangle \; .$$

The inverse transform takes

$$|k\rangle \longrightarrow \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} e^{-2\pi i\, kj/n} |j\rangle \; .$$

In matrix form, this is

$$M_{FT} = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{n-2} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{n-3} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{n-2} & \omega^{n-3} & \dots & \omega \end{pmatrix}$$

where $\omega = e^{2\pi i/n}$. Note that $M^\dagger = M^{-1}$ is just M with ω replaced by ω^{-1}.

The reason the quantum Fourier transform works is that

$$\sum_{k=0}^{n-1} \omega^{k\ell} = \begin{cases} n & \text{if } \ell \text{ is a multiple of } n \\ 0 & \text{otherwise} \end{cases} . \tag{1}$$

This is not hard to prove. If ℓ is a multiple of n, all the terms in the sum are 1, and the sum is n. If ℓ is not a multiple of n, we have a geometric series, and the sum is

$$\sum_{k=0}^{n-1} \omega^{k\ell} = \frac{\left(\omega^\ell\right)^n - 1}{\omega^\ell - 1} = 0,$$

because $\omega^n = 1$. Note that Eq. (1) shows that M is unitary.

In this lecture, we will be taking $n = 2^L$, as the quantum Fourier transform is particularly easy to implement in this case. The quantum Fourier transform presented above is an L-qubit gate. For large L, experimental physicists cannot hope to build an apparatus that takes L qubits and makes them interact so as to execute an arbitrary L-qubit gate.[1] So to implement multi-qubit unitary transformations efficiently, we need to break them into a sequence of 1- and 2-qubit gates. How can we break the quantum Fourier transform into 1- and 2-qubit gates? It turns out that we only need to use two kinds of gates, the first is a Hadamard gate and the second is a controlled R_j gate, where j = 2, 3, 4, ..., L. Here $R_j = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^j} \end{pmatrix}$.

What we will do is represent $|j\rangle$ and $|k\rangle$ as sequences of qubits. Let $j = j_1 j_2 j_3 \dots j_L$ in binary, and let $k = k_1 k_2 k_3 \dots k_L$. Thus,

$$\frac{j}{n} = \sum_{s=1}^{L} \frac{1}{2^s} j_s \qquad \text{and} \qquad \frac{k}{n} = \sum_{t=1}^{L} \frac{1}{2^t} k_t$$

Recall that the quantum Fourier transform takes

$$|j\rangle \longrightarrow \frac{1}{\sqrt{2^L}} \sum_k e^{2\pi i j k/2^L} |k\rangle \tag{2}$$

We will use the binary decimal notation

$$0.a_1 a_2 a_3 \dots = \frac{1}{2} a_1 + \frac{1}{4} a_2 + \frac{1}{8} a_3 + \dots$$

and rewrite Eq. 2 as

$$|j_1\rangle |j_2\rangle \dots |j_L\rangle \rightarrow \frac{1}{\sqrt{2^L}} (|0\rangle + e^{2\pi i 0.j_L} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{L-1} j_L} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_L} |1\rangle)$$

There are two things to show. First, we will show how this rewriting lets us find a circuit for the QFT, and second, why we are allowed to rewrite it like this.

How can we use this to find a circuit? First, let's divide the $\frac{1}{\sqrt{2^L}}$ normalization constant equally among the terms on the righthand side. Now, let's think about the first term

$$\frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.j_n} |1\rangle).$$

Here $0.j_n$ is either 0 or $1/2$, so this exponential is either 1 or -1. We thus have that if $|j_n\rangle = |0\rangle$, this term is $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and if $|j_n\rangle = |1\rangle$, this term is $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Now, we know a gate that implements this transform on $|j_n\rangle$ — the Hadamard gate. It turns out that this Hadamard gate needs to be the last thing we do in our circuit. Let's think about the second term next. What we need to implement is

$$|j_{n-1}\rangle \rightarrow (|0\rangle + e^{2\pi i 0.j_{n-1} j_n} |1\rangle)$$

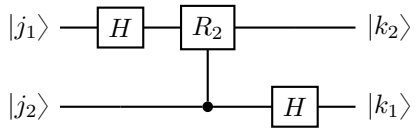
[1] Note that the same thing holds for classical computers ... we implement L-bit Boolean functions not by building special transistors for them, but by breaking them down into NOT, AND, and OR gates, so we only ever have to build 2-bit gates.

How can we implement this? The first thing we do is take

$$|j_{n-1}\rangle \to (|0\rangle + e^{2\pi i 0.j_{n-1}} |1\rangle)$$

We've nearly got it. After applying this, what we need to do is if both $|j_n\rangle = |1\rangle$ and this qubit we've just transformed (which will end up being $|k_2\rangle$, is $|1\rangle$, then we need to multiply the phase by $e^{2\pi i/4} = i$. We can do this—it's just the gate R_2. But note that we need to have the qubit $|j_n\rangle$ available for this, which is why we need to process $|j_n\rangle$ last.

We've worked out how to implement the first two terms. The circuit for this is



where R_2 is the gate $\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$. Note how we need to process $|j_1\rangle$ first so that we have $|j_2\rangle$ available when we're processing $|j_1\rangle$.

Now, we can explain how to produce an arbitrary term in the above product. This is

$$|j_s\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_s j_{s+1} \ldots j_L} |1\rangle)$$

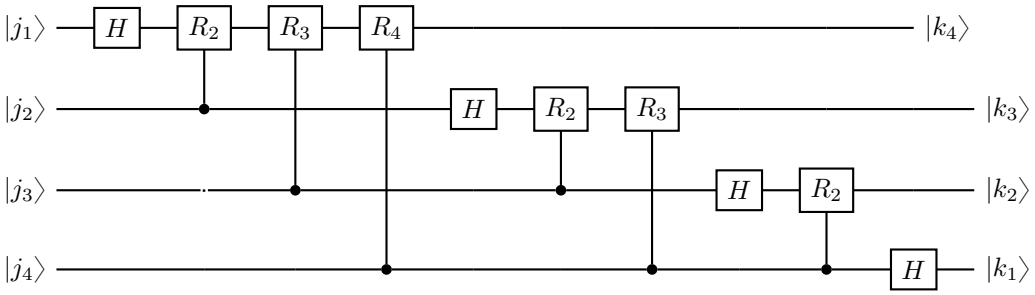
We first apply a Hadamard gate to take

$$|j_s\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_s} |1\rangle)$$

We then apply C-R_2, C-R_3, C-R_4, C-R_{L-s+1} gates between this qubit and the qubits $j_{s+1}, j_{s+2}, \ldots, j_L$ to take

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_s} |1\rangle) \to \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_s j_{s+1} \ldots j_L} |1\rangle)$$

This gives us the circuit, which we give here for $L = 4$ It should be easy enough for you to see the generalization to arbitrary L, but if you want to see it explicitly, the circuit for the general case is given in the textbook (p. 219).



Note that the output qubits come out in reverse order as the input qubits.

Note also that reversing the circuit gives exactly the same gates, (the circuit is symmetric if you flip top and bottom and right and left, because the $|k_t\rangle$ qubits appear in the opposite order as the $|j_s\rangle$ qubits), except that the complex conjugate is applied. This shows that the inverse quantum Fourier transform is the complex conjugate of the quantum Fourier transform, something we already worked out.

There is one final thing I want to say. When people first saw the quantum Fourier transform circuit, some of them raised the objection that the C-R_ℓ gate is so close to the identity that it couldn't possibly be doing anything, so there must be something wrong with the circuit. In fact, they were right about the first part of this—the C-R_ℓ gate hardly changes the state at all. What this means is that you can get a very good approximate Fourier transform by just ignoring the C-R_ℓ gates for moderately large ℓ. This lets you do approximate quantum Fourier transform with many fewer gates.

Notes 8.370/18.435 Fall 2022

Lecture 20 Prof. Peter Shor

Last time, I explained how to construct the circuit for the quantum Fourier transform, but I left one piece out. Today, we finish that piece.

Today, we will also look at one of the applications of the quantum Fourier transform: phase estimation. Later, we will use this to show how to factor and take discrete logs on a quantum computer, rather than using the quantum Fourier transform directly (which was the way these algorithms were originally discovered).

Recall that the quantum Fourier transform takes

$$|j\rangle \longrightarrow \frac{1}{2^{L/2}} \sum_{k=0}^{2^L-1} e^{2\pi i jk/2^L} |k\rangle \tag{1}$$

To derive the circut for the QFT, we needed to rewrite this as

$$|j_1\rangle |j_2\rangle \dots |j_L\rangle \rightarrow \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{\sqrt{2^L}},$$

where j_1 is the most significant bit of j and j_n is the 1's bit of j, where

$$0.j_1 j_2 \dots j_n = \tfrac{1}{2} j_1 + \tfrac{1}{4} j_2 + \tfrac{1}{8} j_3 + \dots + \tfrac{1}{2^n} j_n,$$

and where the qubits on the righthand side are $|k_1\rangle$, $|k_2\rangle$, ..., $|k_n\rangle$.

How do we derive this expression? We start by plugging in the binary expressions for j and k into Eq. 1:

$$\begin{aligned} |j_1\rangle |j_2\rangle \dots |j_L\rangle \longrightarrow & \frac{1}{2^{L/2}} \sum_{k=0}^{2^L-1} e^{2\pi i 2^L \left(\sum_{s=1}^{L} \frac{1}{2^s} j_s\right) 2^L \left(\sum_{t=1}^{L} \frac{1}{2^t} k_t\right)/2^L} |k_1\rangle |k_2\rangle \dots |k_L\rangle \\ = & \frac{1}{2^{L/2}} \sum_{k=0}^{2^L-1} e^{2\pi i \sum_{t=1}^{L} \left(k_t 2^{L-t} \left(\sum_{s=1}^{L} \frac{1}{2^s} j_s\right)\right)} |k_1\rangle |k_2\rangle \dots |k_L\rangle \end{aligned}$$

We next express the sum on the righthand side as a tensor product:

$$|j_1\rangle |j_2\rangle \dots |j_L\rangle \longrightarrow \frac{1}{2^{L/2}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \dots \sum_{k_L=0}^{1} \bigotimes_{t=1}^{L} \left(e^{2\pi i k_t 2^{L-t} \sum_{s=1}^{L} \frac{1}{2^s} j_s} |k_t\rangle \right)$$

Now, because the exponential of an integer times $2\pi i$ is 1, we can start the sum in the exponent, $2^{L-t} \sum_{s=1}^{L} \frac{1}{2^s} j_s$, with $s = t$ rather than $s = 1$. But

$$e^{2\pi i 2^{L-t} \sum_{s=t}^{L} \frac{1}{2^s} j_s} = e^{2\pi i 0.j_t j_{t+1} \dots j_L}$$

So this gives

$$|j_1\rangle |j_2\rangle \dots |j_L\rangle \longrightarrow \frac{1}{2^{L/2}} \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \dots \sum_{k_L=0}^{1} \bigotimes_{t=1}^{L} \left(e^{2\pi i k_t 0.j_t j_{t+1} \dots j_L} |k_t\rangle \right).$$

However, the value of the term for $|k_t\rangle$ in this expression depends only on whether $|k_t\rangle = |0\rangle$ or $|k_t\rangle = |1\rangle$. We can thus use the distributive law of tensor products over sums to rewrite it as

$$|j_1\rangle\,|j_2\rangle\ldots|j_L\rangle \longrightarrow \frac{1}{2^{L/2}}\bigotimes_{t=1}^{L}\left(|0\rangle + e^{2\pi i 0.j_t j_{t+1}\ldots j_L}\,|1\rangle\right),$$

which is what we were trying to derive.

What is quantum phase estimation? Suppose we have a unitary operator U and an eigenvector $|v_\theta\rangle$ of this operator. The phase estimation problem is to give an approximation to the eigenvalue associated with $|v_\theta\rangle$, that is, to approximate the θ such that

$$U\,|v_\theta\rangle = e^{i\theta}\,|v_\theta\rangle\,.$$

The phase estimation algorithm does not give very accurate estimates of θ unless you can take high powers of U. Thus, we will assume that we have some kind of circuit that will compute U^{2^ℓ} in polynomial time in ℓ. Is this a reasonable assumption? There are cases for which it is. Suppose the unitary takes a number $s \pmod p$ and multiplies it by a number $g \pmod p$, so

$$U\,|s \,(\mathrm{mod}\ p)\rangle = |gs \,(\mathrm{mod}\ p)\rangle$$

Then U^k simply multiplies a number s by $g^k \pmod p$,

$$U^k\,|s \,(\mathrm{mod}\ p)\rangle = \left|g^k s \,(\mathrm{mod}\ p)\right\rangle,$$

and if we know $g^k \pmod p$, U^k is essentially no harder to implement than U. We will use this unitary transformation in the factoring algorithm, and we will discuss it in more detail then.

What is the basic idea of the algorithm? What we will do is create the state

$$\frac{1}{2^{L/2}}\sum_{k=0}^{2^L-1} e^{ik\theta}\,|k\rangle \tag{2}$$

Now, recall that the quantum Fourier transform maps

$$|j\rangle \longrightarrow \frac{1}{2^{L/2}}\sum_{k=0}^{2^L-1} e^{2\pi ijk/2^L}\,|k\rangle$$

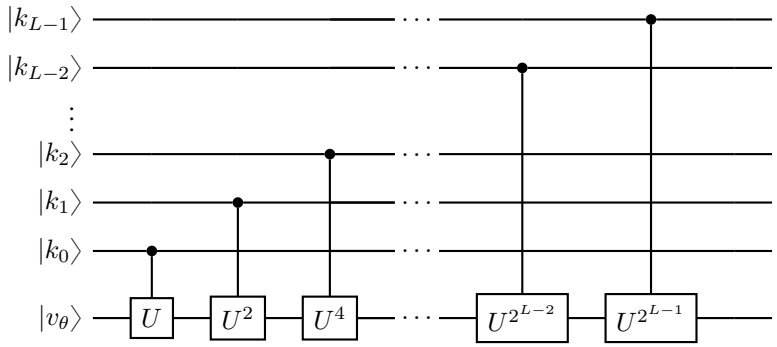
And since on the left-hand side, the states $|j\rangle$ form a basis, so do the states on the rigith-hand-side. But if $\theta = 2\pi m/2^L$ for some integer m, it's one of these basis states, and thus to identify θ, all we need to do is measure in that basis. How do we do that? We take the inverse Fourier transform, which takes

$$\frac{1}{2^{L/2}}\sum_{k=0}^{2^L-1} e^{2\pi ijk/2^L}\,|k\rangle$$

to $|j\rangle$, and then measure in the standard basis. Thus, if we get j, we know that $\theta = 2\pi j/2^L$.

To complete this sketch of the algorithm, we need to do two things. First, we need to show how to create the state (2). Second, we need to show that the algorithm works even if θ is not an integer multiple of $2\pi/2^L$.

We first explain how to create the state (2). Let's assume that we know how to implement the transformation U^{2^k} efficiently (in polynomial time, if we want the entire algorithm to be polynomial-time). Now, consider the following quantum circuit.



If the input is $k = k_{L-1}k_{L-2}\ldots k_1k_0$ in binary, this circuit applies $U^{k_0+2k_1+4k_2+\ldots+2^{L-1}k_{L-1}}$ to $|v_\theta\rangle$, which is the same as applying $U^k|v_\theta\rangle = e^{ik\theta}$, we have the output of $|k\rangle e^{i\theta k}$. Thus, to get our desired state

$$\frac{1}{2^{L/2}}\sum_{k=0}^{2^L-1} e^{ik\theta}|k\rangle|v_\theta\rangle, \tag{3}$$

we simply need to input

$$\frac{1}{2^{L/2}}\sum_{k=0}^{2^L-1} |k\rangle|v_\theta\rangle$$

into the circuit above. We can do this by putting a $|+\rangle$ state into each of the first L quantum wires.

Next, we will compute what happens when we take the above state and apply the inverse Fourier transform to it, even in the case where θ is not an integer multiple of $2\pi/2^L$. The inverse Fourier transform is

$$|k\rangle \longrightarrow \frac{1}{2^{L/2}}\sum_{j=0}^{2^L-1} e^{-2\pi ijk/2^L}|j\rangle,$$

so when we plug it into (3), we get

$$\frac{1}{2^L}\sum_{k=0}^{2^L-1} e^{ik\theta}\sum_{j=0}^{2^L-1} e^{-2\pi ijk/2^L}|j\rangle = \frac{1}{2^L}\sum_{j=0}^{2^L-1}|j\rangle\left(\sum_{k=0}^{2^L-1} e^{ik(\theta-2\pi j/2^L)}\right)$$

The last piece is a geometric sum, so we can use the formula for geometric sums, and show that the probability of seeing $|j\rangle$ is

$$\left| \frac{1}{2^L} \sum_{k=0}^{2^L-1} e^{ik(\theta - 2\pi j/2^L)} \right|^2 = \frac{1}{4^L} \left| \frac{1 - e^{i(2^L\theta - 2\pi j)}}{1 - e^{i(\theta - 2\pi j/2^L)}} \right|^2 .$$

We now show that the value of j we obtain will give us a very good approximation of θ. There are more accurate ways of doing this than the one we use, but this one is fairly simple. will bound the numerator by 2, and approximate the denominator by $i(\theta - 2\pi j/2^L)$. Let j' be the value (not necessarily an integer) which would give the right value of θ, that is the j' that makes $2\pi j'/2^L = \theta$. We see that the probability of seeing some specific j with $j > j' + \alpha$ or $j < j' - \alpha$ is at most around

$$\frac{1}{4^L} \left| \frac{2}{2\pi(j' - j)/2^L} \right|^2 \leqslant \frac{1}{|\pi\alpha|^2}.$$

This shows that j is very tightly concentrated around $2^L\theta/(2\pi)$, and thus we can get a good estimate of the phase θ.

By looking at the phase estimation algorithm, you can see that if you input an eigenvector of the transformation U, the phase estimation circuit does not change this eigenvector. This will be quite important in our later applications of phase estimation.

Notes 8.370/18.435 Fall 2022

Lectures 21 Prof. Peter Shor

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 in general?. 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 $A \not\equiv \pm B \pmod N$. (Here $x \equiv y \pmod N$ means that $y - x$ is a multiple of 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 $A - B$ using the Euclidean algorithm, which we will explain 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 $f(x) \equiv a^x \pmod N$. 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 almost certainly 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 $a^{10} \equiv 1 \pmod{33}$ so $(a^5 + 1)(a^5 - 1) \equiv 0 \pmod{33}$ Unfortunately, it doesn't work this time because we chose the wrong value of a: since $2^5 + 1 \equiv 33$, 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 $a^{10} \equiv 1$. But now $a^5 \equiv 23$, 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)

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

What happens when we apply U_a? We get

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

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 $d/2^{2L}$ to 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.

Notes 8.370/18.435 Fall 2022

Lecture 22 Prof. Peter Shor

In this lecture, we talked about the number theory that we needed for the factoring algorithm..

1 The Euclidean Algorithm and the Extended Euclidean Algorithm

Let's recall how we found the factors of N. To make the exposition easier, we will assume that N is a product of two primes, $N = PQ$ in these notes, but the factoring algorithm works fine in the general case when more than two primes divide N.

Recall that in order to factor, we found the period of the sequence

$$g, g^2 \pmod N, g^3 \pmod N, \ldots.$$

If the period of this sequence is r, then we must have $g^r \equiv 1 \pmod N$. Why? Because there are at most $N-2$ different values in this sequence, we must have $g^a = g^{r+a} \pmod N$ for some a. But then, multiplying by $g^{-a} \pmod N$, we get $1 = g^r \pmod N$.

Now that we have $g^r \equiv 1 \pmod N$, if r is even, we can factor this expression to get

$$(g^{r/2} - 1)(g^{r/2+1}) \equiv 0 \pmod N;$$

We have two numbers multiplying to a multiple of N. If neither of them is a multiple of N, then we have P must divide one of the numbers and Q the other. Let's try to factor 33 in this way. Suppose we take $g = 2$. We see that $2^{10} \equiv 1 \pmod{33}$, so we get $(2^5 - 1)(2^5 + 1) \equiv 0 \pmod 3$. Unfortunately, this doesn't give us a factor, because $2^5 + 1 \equiv 33$.

So let's take $g = 5$. We then have (again) that $5^{10} \equiv 1 \pmod N$, so $(5^5 - 1)(5^5 + 1) \equiv 0 \pmod{33}$. We an compute that $5^5 \pmod{33} = 23$, so this gives $(23 - 1)(23 + 1) \equiv 0 \pmod{33}$. And this time it worked! 22 contains the factor 11 and 24 contains the factor 3. How do we recover 3 from 24 and 33. We use the Euclidean algorithm for finding the greatest common divisor of two numbers. How do we implement this? One standard way is to put the two numbers we start with in a row, with the larger first. We then repeatedly move the number in the right column to the left column, and replace the number in the right column by the remainder we get when dividing these two numbers. For example, to find gcd$(24, 9)$,

$$\begin{array}{rr} 33 & 24 \\ 24 & 9 \\ 9 & 6 \\ 6 & 3 \end{array}$$

Here, in the first step, we divide 33 by 24, and get remainder 9. In the second step, we divide 24 by 9 and get remainder 6, and so on. If something divides both of the

first numbers, it will divide all the other numbers in our array. We keep decreasing the size of the numbers, so eventually we will reach the greatest common divisor of the numbers.

Now, let's look again at our algorithm. We needed to find the period of the unitary map $|\, y\rangle \to |\, gy \pmod N\rangle$. How can we implement this map.

Recall from our discussion of reversible classical computation, that if we have a classical circuit taking y to $gy \pmod N$ and a circuit take $gy \pmod N$ to y, we can find a reversible circuit whose input is y (along with some workbits whose initial values are 0 and whose output is $gy \pmod N$, where the values of the workbits have been returned to 0. Finding a circuit that takes y to $gy \pmod N$ is easy—it's just multiplication. But how do we go the other way? What we need to do is find $g^{-1} \pmod N$ and then use a circuit for multiplication that takes x to $g^{-1}x \pmod N$. So the only hard part of this is finding $g^{-1} \pmod N$. For this, we use something called the extended Euclidean algorithm.

As an example, let's find $5^{-1} \pmod{33}$. The first thing we do is use the Euclidean algorithm to find the greatest common divisor of 5 and 33. Recall that 5^{-1} only exists if this $\gcd(5, 33) = 1$. What we do is divide 33 by 5 to get the remainder 3, and then repeat with these two numbers—we divide 5 by 3 to get the remainder 2:

$$\begin{array}{cc} 33 & 5 \\ 5 & 3 \\ 3 & 2 \\ 2 & 1 \end{array} \qquad \begin{array}{l} 33 - 6 \cdot 5 = 3 \\ 5 - 1 \cdot 3 = 2 \\ 3 - 1 \cdot 2 = 1 \end{array}$$

Our next goal is to find two integers s and t such that $s \cdot 3 + t \cdot 5 = 1$. What we do is start from the last row and work backwards. In the second to last row, we have that $1 \cdot 3 - 1 \cdot 2 = 1$. What we do is plug in the expression for 2 in the second last row of this array to get So $1 \cdot 3 - 1 \cdot (5 - 3) = 1$, and simplifying this gives $2 \cdot 3 - 1 \cdot 5 = 1$. Now, we plug in the expression for 3 in the first row of our array, giving $2 \cdot (33 - 6 \cdot 5) - 1 \cdot 5 = 1$. This simplifies to $2 \cdot 33 - 13 \cdot 5 = 1$. But from this expression, we can find the inverse of $5 \pmod{33}$. Since the first term is a multiple of 33, we have $-13 \cdot 5 = 1 \pmod N$, which gives $5^{-1} = 33 - 13 = 20$.

In general, to implement the extended Euclidean algorithm, we start at the last row given by the Euclidean algorithm and work backwards. Let's say the first row of the Euclidean algorithm is r_1, r_2, the second row r_2, r_3, and so forth. For each row, we get an equation $r_j - q_j r_{j+1} = r_{j+2}$. Now, lets say we have found two integers s and t such that

$$s r_{j+1} + t r_{j+2} = 1$$

We plug in our formula for r_{j+2} into this equation to get

$$s r_{j+1} + t(r_j - q_j r_{j+1}) = 1.$$

Simplifying this will give us s' and t' so that $s' r_j + t' r_{j+1} = 1$. when we reach the top row, we have $s r_1 + t r_2 = 1$. This means that $t = r_2^{-1} \pmod{r_1}$.

2 Continued Fractions

First, we're going to go through an example to show how the continued fraction algorithm works. After that, we will prove some properties of it.

Let's use 33 as the example number we want to factor. We first pick a number g and find the period of $g^x \pmod{33}$. Recall that if we chose $g = 5$, the period was 10. We will need to choose $L \approx 2 \log N$ in the phase estimation algorithm. What the phase estimation algorithm finds is an estimate of the eigenvalue of $e^{2\pi i k/10}$ for some k, let's say $k = 3$. The phase estimation algorithm then returns a number of the form $d/2^L$that is close to $3/10$. Let's choose $2^L = 2048$. Then the approximation for $3/10$ would have a denominator of 2048. Let's say this approximation is $615/2048$. How do we recover $3/10$ from $615/2048$?

What we do is use *continued fractions*. A continued fraction is a number of the form

$$\cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{\ddots}}}}}$$

Taking the continued fraction of $615/2048$, we find that

$$a_1, a_2, a_3, a_4, a_5, \ldots = 3, 3, 33, 1, 5.$$

How did we find this? We start by dividing 2048 by 615, and get 3, with remainder 203 This shows us that

$$\frac{615}{2048} = \cfrac{1}{3 + \cfrac{203}{615}}$$

and so forth.

We now proceed by finding the continued fraction of $\frac{203}{615}$. Since $615 = 3 \cdot 203 + 9$, this gives

$$\cfrac{1}{3 + \cfrac{203}{615}} = \cfrac{1}{3 + \cfrac{1}{3 + \cfrac{6}{203}}}$$

Continuing this process, we get

$$\begin{aligned}
\frac{615}{2048} &= \frac{1}{3+\dfrac{203}{615}} \\
&= \frac{1}{3+\dfrac{1}{3+\dfrac{6}{203}}} \\
&= \frac{1}{3+\dfrac{1}{3+\dfrac{1}{33+\dfrac{5}{6}}}} \\
&= \frac{1}{3+\dfrac{1}{3+\dfrac{1}{33+\dfrac{1}{1+\dfrac{1}{5}}}}}
\end{aligned}$$

The property of continued fractions that we will be using is that all the best approximations of a real number R by a rational number are the convergents of the continued fractions for the number. What are the convergents? With the example above, the first few convergents are:

$$\begin{aligned}
\frac{1}{3} &= \frac{1}{3} = 0.3333, \\
\frac{1}{3+\dfrac{1}{3}} &= \frac{3}{10} = 0.3, \\
\frac{1}{3+\dfrac{1}{3+\dfrac{1}{33}}} &= \frac{100}{333} = 0.3003003,
\end{aligned}$$

You can see that these values keep getting closer to $615/2048 = .300293$. The third convergent has a denominator of 333, which is clearly too large when we're factoring 33, so the right convergent to choose is the second one, which has 10 as the denominator, and which gives us the correct factorization.

The remaining thing to do is to show that all the close approximations to a fraction are convergents of its continued fraction. We will show:

Theorem 1 *if* $|R - \frac{p}{q}| \leq \frac{1}{2q^2}$*, then* $\frac{p}{q}$ *is one of the convergents of* r*.*

How do we show this theorem? We will first prove a lemma:

Lemma 1 *Suppose* $\frac{p}{q} < R < \frac{p'}{q'}$ *and* $pq' = 1 + p'q$ *With these conditions if* $q < q'$*, then* $\frac{p}{q}$ *is one of the convergents of R, and if* $q' < q$*, then* $\frac{p'}{q'}$ *is one of the convergents of R.*

Let's take as an example $R = 615/2048 \approx .30030293$. We have

$$\frac{3}{10} = 0.3 < 615/2048 = 0.30030293 < \frac{10}{33} = 0.30303.$$

We can easily check that $3 \cdot 33 + 1 = 10 \cdot 10$ This shows that $\frac{3}{10}$ is a convergent. ($\frac{33}{100}$ is not, although it is something called a *semiconvergent*).

Proof of Lemma:

First, let's look at the continued fractions for $\frac{p}{q}$ and $\frac{p'}{q'}$. I claim that they cannot be of the forms

$$\frac{p}{q} = \cfrac{1}{a + \cfrac{1}{b + \cfrac{1}{c + \dots}}}$$

and

$$\frac{p'}{q'} = \cfrac{1}{a' + \cfrac{1}{b' + \cfrac{1}{c' + \dots}}}$$

with $a > a'$. Suppose they were. Then $\frac{p}{q} = \frac{1}{a+\epsilon} < \frac{1}{a}$ and $\frac{p'}{q'} = \frac{1}{a-\epsilon} > \frac{1}{a}$. So the fraction $\frac{1}{a}$ would have to be between $\frac{p}{q}$ and $\frac{p'}{q'}$, and would have a lower denominator than either, and it would be impossible for $\frac{p'}{q'} - \frac{p}{q} = \frac{1}{qq'}$. Thus, p' and q' both must start with $a = a'$. Now, let's consider the fractions

$$\frac{p}{q} = \frac{1}{a + \frac{r}{s}} \quad \text{and} \quad \frac{p'}{q'} = \frac{1}{a + \frac{r'}{s'}},$$

where

$$\frac{r}{s} = \cfrac{1}{b + \cfrac{1}{c + \dots}}$$

and

$$\frac{r'}{s'} = \cfrac{1}{b' + \cfrac{1}{c' + \dots}}$$

We will show $p'q - pq' = 1$ if and only if $r's - rs' = 1$.

First, we calculate

$$\frac{p}{q} = \frac{r}{ar+s} \quad \text{and} \quad \frac{p'}{q'} = \frac{r'}{ar'+s'}\,.$$

Now, all these fractions must be in lowest terms, so

$$1 = p'q - pq' = r'(ar+s) - r(ar'+s') = r's - rs'.$$

What this shows is that if the two continued fractions

$$\frac{p}{q} = \cfrac{1}{a+\cfrac{1}{b+\cfrac{1}{c+\ldots}}} \quad \text{and} \quad \frac{p'}{q'} = \cfrac{1}{a'+\cfrac{1}{b'+\cfrac{1}{c'+\ldots}}},$$

satisfy $|\frac{p'}{q'} - \frac{p}{q}| = \frac{1}{qq'}$, then the continued fractions

$$\frac{r}{s} = \cfrac{1}{b+\cfrac{1}{c+\ldots}} \quad \text{and} \quad \frac{r'}{s'} = \cfrac{1}{b'+\cfrac{1}{c'+\ldots}},$$

satisfy $|\frac{r}{s} - \frac{r'}{s'}| = \frac{1}{ss'}$. We can in this way keep removing the first terms of the continued fractions and preserve the relation between the remaining terms. When can this process end? It can only end when one of the two continued fractions has been reduced to the form $\frac{1}{a}$. At this point, the other continued fraction must look like

$$\cfrac{1}{a+\cfrac{1}{b+\ldots}},$$

so the first continued fraction is a convergent of the second one. And since R is sandwiched between them, the first continued fraction must also be a convergent of R.

We now use the lemma to prove the theorem. Suppose that $\frac{p}{q} < R$ (the case of $\frac{p'}{q'} > R$ is completely analogous) and that $R - \frac{p}{q} < \frac{1}{2q^2}$. Now, $\frac{p}{q}$ must be the closest fraction to R with denominator at most q, because the closest two fractions with denominator less than or equal to q can be to each other is $\frac{1}{q(q-1)}$. There must also be a smallest fraction larger than $\frac{p}{q}$ with denominator at most q. Call this fraction $\frac{p'}{q'}$. Because there are no fractions between $\frac{p}{q}$ and $\frac{p'}{q'}$ with denominator at most q, we must have $pq' + 1 = p'q$. And we must have

$$\frac{p}{q} < R < \frac{p'}{q'}$$

Let's consider the fraction $\frac{p+p'}{q+q'}$. We have

$$\begin{aligned}\frac{p+p'}{q+q'} - \frac{p}{q} &= \frac{q(p+p') - p(q+q')}{q(q+q')} \\ &= \frac{p'q - q'p}{q(q+q')} \\ &= \frac{1}{q(q+q')} > \frac{1}{2q^2}\end{aligned}$$

This is larger than the distance between $\frac{p}{q}$ and R, so R must be between $\frac{p}{q}$ and $\frac{p+p'}{q+q'}$. And clearly the denominator q is less than the denominator $q+q'$. This shows that $\frac{p}{q}$ satisfies the conditions of the Lemma to be a convergent of R, and we have proved Theorem 1.

3 The Chinese Remainder Theorem

Recall that I said that the probability of finding an r such that $\gcd(a^{r/2} \pm 1, N)$ gave you a factor was at least $\frac{1}{2}$. Why is this true? You need the Chinese remainder theorem to prove this.

What is the Chinese remainder theorem? It says that if you have a product $N = PQ$, and if P and Q sre relatively prime, then there is a one-to-one correspondence between numbers modulo N and pairs of numbers modulo P and Q. That is, we have a correspondence between

$$x \text{mod } N \longleftrightarrow \big(x \text{mod } P, x \text{mod } Q\big)$$

Let's take $77 = 7 \cdot 11$ as an example. Suppose we have the number 60 mod 77. We want the pair (x, y) corresponding with 77. We find this pair by finding the remainder when 60 is divided by 7 and 11, respectively. Thus $53 \leftrightarrow (4, 6)$. There is a polynomial-time algorithm to go the other way; that is, from the pair $(2, 8)$, it is possible to find 30 mod 77, but we won't cover this calculation in these notes.

How does r for some a depend on a mod P and a mod Q? It turns out that 3 is a multiplicative generator modulo 7 and 2 is a multiplicative generator mod 11. Let's use this to make a table of the numbers modulo 7 and 11. For a number x, we let $r_7(x)$ and r_{11} be the smallest power to which we have to raise x to get 1.

power of 3	$x$mod 7	r_7	power of 2	$x$mod 11	r_{11}
3^1	3	6	2^1	2	10
3^2	2	3	2^2	4	5
3^3	6	2	2^3	8	10
3^4	4	3	2^4	5	5
3^5	5	6	2^5	10	2
3^6	1	1	2^6	9	5
			2^7	7	10
			2^8	3	5
			2^9	6	10
			2^{10}	1	1

You can see from a little thought that if g is a generator of the multiplicative group mod P, then for $a = g^x$, $r_P = P - 1$ if and only if $\gcd(x, P-1) = 1$.

How do we combine the r_7 and r_{11} to get r_{77}. For a number to be 1 mod 77, it has to be 1 both mod 7 and mod 11. Thus, we need to take the least common multiple (lcm) of r_7 and r_{11}. For example, let's look at the number $53 \longleftrightarrow (4, 6)$. From the table, we need 4^3 to make it 1 mod 11 and 6^{10} to make it 1 mod 11. Thus, if we choose $a = 53$, we get $r = \mathrm{lcm}(3, 10) = 30$. So what is 53^{15}. It corresponds to $4^{15} \equiv (4^3)^5 \equiv 1 \pmod 7$ and $6^{15} \equiv 6^{10}6^5 \equiv 6^5 \equiv -1 \pmod{11}$. Thus, $53^{15} \neq \pm 1$, and will it give us a factor.

If we had chosen $a = 30 \longleftrightarrow (2, 8)$, we would get $r_7 = 3$ and $r_{11} = 5$. Then, $r = 15$ so it is not even, and thus it doesn't work.

How about $a = 6 \longrightarrow (6, 6)$? Then we get $r_7 = 2$ and $r_{11} = 10$. Thus, $r = 10$. We can see from the table that $6^{10} \equiv 1$ mod 11, so 6^5 mod 11 is a square root of 1. There is only one square root of 1 modulo any prime P, so $6^5 \equiv 1$ mod 11. Similarlly, $6^5 \equiv -1$ mod 7, so $6^5 \equiv -1$ mod 77, and we don't find a factor.

What are the conditions for giving a factor for a general $N = PQ$, with P and Q prime? r cannot be odd, and $a^{r/2}$ cannot be -1. We have r is odd if and only if $r_P(a)$ and $r_Q(a)$ are odd, so 2 doesn't divide either $r_P(a)$ and $r_Q(a)$. Put another way, the largest power of 2 dividing $r_P(a)$ (and $r_Q(a)$) is 0.

Now, if $a^{r/2} = -1$ modulo both P and Q, it must be the case that the same largest power of 2 divides both r_P and r_Q. To see this, consider an example. if $r_P(a) = 4s$ and $r_Q(a) = 2t$, where s and t are odd, then $r = \mathrm{lcm}(r_P, r_Q) = 4\mathrm{lcm}(s, t)$. And $a^{2\mathrm{lcm}(r_P, r_Q} \equiv -1$ mod P but $a^{2\mathrm{lcm}(r_P, r_Q} \equiv 1$ mod P. So an a will result in a factor if and only if the largest powers of 2 dividing r_P and r_Q are different.

Now consider an arbitrary P and a generator g for it. For $a = g^x$ mod P, if x is odd, then the largest power of 2 dividing $P - 1$ is the largest power of two dividing $r_P(a)$, and if g is even, the largest power of 2 dividing $r_P(a)$ is smaller than the largest power of 2 dividing $P - 1$. Thus, if a is chosen at random, the largest powers of 2 dividing $r_P(a)$ and $r_Q(a)$ are the same with probability at most $\frac{1}{2}$, and we see that a random a works with probability at least $\frac{1}{2}$.

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

$(\mathrm{mod}\ P)$ and g to find x, and then use this to obtain $g^{xy}\ (\mathrm{mod}\ P)$. However, we assume that discrete logs are hard. While we cannot formally prove that finding $g^{xy}\ (\mathrm{mod}\ P)$ from the three values g, $g^x\ (\mathrm{mod}\ P)$, and $g^y\ (\mathrm{mod}\ P)$ 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 $g^x\ (\mathrm{mod}\ P)$? One way is to find $g^2\ (\mathrm{mod}\ P)$, $g^4\ (\mathrm{mod}\ P)$, $g^8\ (\mathrm{mod}\ P)$, 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: $x = x_{L-1}x_{L-2}\ldots x_1x_0$.

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

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:

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

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 $g^{-1}\ (\mathrm{mod}\ P)$, and $gg^{-1} = 1$. For example, if P is 31 and g is 3. then g^{-1} is 21, because $3\cdot 21 = 63 \equiv 1\ (\mathrm{mod}\ 31)$. We explained how to compute g^{-1} later in a previous set of lecture notes. The inverse of the unitary U_g is:

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

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:

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

and we can compute $g^{2^k}\ (\mathrm{mod}\ P)$ 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

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

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

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

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

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 $1, g, g^2, \ldots, g^{P-2}$ (mod P) must be the numbers $1, 2, \ldots, P-1$ 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 $e^{-2\pi ik/(P-1)}$. Let's say the estimate is θ_k. The phase estimation takes

$$|\,v_k\rangle \rightarrow |\,v_k\rangle\,|\,\theta_k\rangle \ ,$$

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 $\theta_k = \frac{d}{2^L}$:

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

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?

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

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

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.

Notes 8.370/18.435 Fall 2022

Lecture 24 Prof. Peter Shor

Today, we are talking about Grover's search algorithm. This will let you search a possible space of size N for a solution in $\sqrt{N}$ time.

Suppose you have an equation, and you want to know whether it has a solution in some finite search space of possible solutions. To make this more concrete, suppose you were looking at the equation $x^3 + y^3 + z^3 = 42$, where x, y, and z are integers. Number theorists have studied this type of equation, and equations like this often have small solutions when you replace 42 with a different number, like 34:

$$3^3 + 2^3 + (-1)^3 = 34$$
$$5^3 + (-4)^3 + (-3)^3 = 34.$$

However, nobody had discovered three cubes that added to 42 until a few years ago, when Andrew Booker (a math professor at U. Bristol) and Drew Sutherland (a math professor at MIT) used a large amount of computer time to find three 17-digit numbers whose cubes added up to 42.

How would you solve this problem with brute force search? You would take two numbers, x, and y, and see whether x^3+y^3 had an integer cube root. This would require searching around 10^{34} potential solutions before you found the right 17-digit numbers. Using Grover's algorithm would only require around the square root of that, so the time it would take to check around 10^{17} numbers on a classical computer. Booker and Sutherland, of course, didn't search 10^{34} potential solutions ... they used an algorithm that was much cleverer than that, and they still needed a million hours of computer time (run on computers that would otherwise have been idle). But a quantum computer could have reduced the time take substantially without any cleverness at all. Furthermore, quite often you can combine clever classical techniques with ideas from Grover's algorithm to reduce the computational time still further than plain Grover search.

So how does Grover's algorithm work? We are given a phase oracle that will tell us whether something is a solution:

$$O_p \,|\, x\rangle = \begin{cases} -\,|\, x\rangle & \text{if x is a solution,} \\ \;\;\,|\, x\rangle & \text{otherwise.} \end{cases}$$

If you have a circuit that checks whether something is a solution, it is fairly straightforward to implement this oracle. You choose a qubit that you set to $|\, 0\rangle$ if the input is not a solution and $|\, 1\rangle$ if the input is a solution, you then apply σ_z to this qubit, and finally you uncompute everything so you are left only with the input $\pm\,|\, x\rangle$.

We will encode the elements of the search space by the numbers 0, 1, 2, 3, ..., $N-1$. We can assume that N is a power of 2 by adding dummy elements of the search space ... the running time of the algorithm isn't greatly affected by rounding N up to a power of 2. Grover's algorithm works by starting in the superposition of every item in the search space:

$$|\, \psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N-1} |\, i\rangle \,.$$

It then applies the following four steps, which we will call a *Grover iteration*, repeatedly until we are done. We will calculate the exact number of times we need to apply the Grover iteration later.

The Grover iteration is:

1. Apply the oracle O_p,
2. Apply the Hadamard transform $H^{\otimes n}$
3. Apply the unitary transformation $2\,|\,0\rangle\langle 0\,| - I$,
4. Apply the Hadamard transform $H^{\otimes n}$.

The third step, $2\,|\,0\rangle\langle 0\,| - I$, is unitary and easy to implement. To see why, consider the case of two qubits. The transformation is

$$2\,|\,0\rangle\langle 0\,| - I = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

So it applies a -1 phase if the state is not $|\,0\rangle$ and a 1 phase if it is $|\,0\rangle$. To do this, you check whether it is $|\,0\rangle$, and if not you apply a -1 phase. This transformation can be constructed out of Toffoli gates, σ_x gates and σ_z gates (try this as an exercise).

Let's look at the last three steps. Let $|\,\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |\,i\rangle$. Then, because $H^{\otimes n}\,|\,0\rangle = |\,\psi\rangle$,

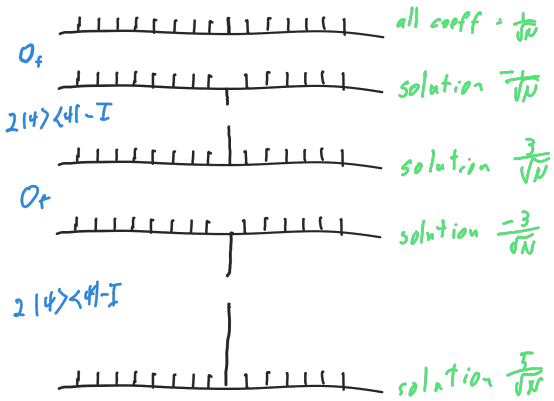
$$H^{\otimes n}(2\,|\,0\rangle\langle 0\,| - I)H^{\otimes n} = 2\,|\,\psi\rangle\langle\psi\,| - I.$$

What does $2\,|\,\psi\rangle\langle\psi\,| - I$ do? It reflects all the amplitudes around their average value. The operator $\frac{1}{N}\sum_{i=0}^{N-1} \langle i\,|$ computes the average value of the amplitude, and the operator $\sum_{i=0}^{N-1} |\,i\rangle$ applies this value to every state. And together, we have

$$\left(\sum_{i=0}^{N-1} |\,i\rangle\right)\left(\frac{1}{N}\sum_{i=0}^{N-1} \langle i\,|\right) = |\,\psi\rangle\langle\psi\,|\,.$$

Why does this work, and how many times do we have to apply the iteration? We will start by giving some intuition by explaining Grover's original analysis, and then describe a different way of analyzing the algorithm which comes up with a more accurate answer, and gives a different intuition for why it works.

The O_p steps reflect each of the amplitudes around 0, and the other three steps of the Grover iteration reflect them around the average amplitude. Let's see what happens:



The marked state starts off with amplitude $\frac{1}{\sqrt{N}}$. The first reflection around the x-axis takes it to $-\frac{1}{\sqrt{N}}$. At this point, the average will still be almost exactly $\frac{1}{\sqrt{N}}$, so the reflection around the average takes it to $\frac{3}{\sqrt{N}}$. After two more steps, it is $\frac{5}{\sqrt{N}}$. And after the kth Grover iteration, as long as the average amplitude stays around $\frac{1}{\sqrt{N}}$, the marked state will have amplitude $\frac{2k+1}{\sqrt{N}}$. So it seems reasonable that after $O(\sqrt{N})$ steps, almost all the amplitude will be in the marked state, and we will have probability nearly 1 of finding it.

This is the analysis that Grover used to discover his algorithm. The textbook uses a more accurate analysis. We now explain it. Suppose now we have M marked states.

Let $|\,\alpha\rangle$ be the equal superposition of all unmarked states:

$$|\,\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{x \text{ unmarked}} |\,x\rangle\,,$$

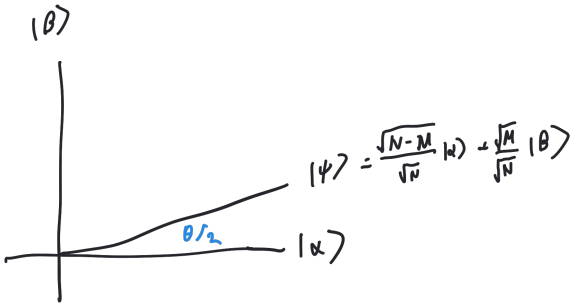
and let $|\beta\rangle$ be the equal superposition of all marked states:

$$|\,\beta\rangle = \frac{1}{\sqrt{M}} \sum_{x \text{ marked}} |\,x\rangle\,.$$

We have $|\,\psi\rangle$ is the equal superposition of all states, so

$$|\,\psi\rangle = \frac{\sqrt{N-M}}{\sqrt{N}}\,|\,\alpha\rangle + \frac{\sqrt{M}}{\sqrt{N}}\,|\,\beta\rangle\,.$$

The Grover iteration treats all marked states alike, and all unmarked states alike. It starts in a superposition of $|\,\alpha\rangle$ and $|\,\beta\rangle$, so during the algorithm, the computational state will always remain in the subspace generated by $|\,\alpha\rangle$ and $|\,\beta\rangle$. We can thus picture it as acting in a plane:



Let $\frac{\theta}{2}$ be the angle between $|\,\alpha\rangle$ and $|\,\psi\rangle$. By trigonometry,

$$\sin\left(\frac{\theta}{2}\right) = \frac{\sqrt{M}}{\sqrt{N}},$$

so $\theta \approx 2\frac{\sqrt{M}}{\sqrt{N}}$ if $M \ll N$.

What do O_p and $2\,|\,\psi\rangle\langle\psi\,| - I$ do to the state? The operation O_p takes $|\,\alpha\rangle$ to $|\,\alpha\rangle$, because it acts as the identity on unmarked states, and $|\,\beta\rangle$ to $-\,|\,\beta\rangle$, because it applies a phase of -1 to marked states. The operation $2\,|\,\psi\rangle\langle\psi\,| - I$ takes $|\,\psi\rangle$ to $|\,\psi\rangle$, because

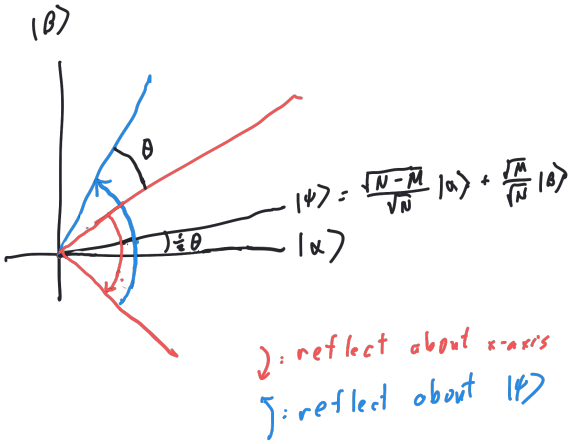
$$\left(2\,|\,\psi\rangle\langle\psi\,| - I\right)|\,\psi\rangle = 2\,|\,\psi\rangle - |\,\psi\rangle = |\,\psi\rangle\,.$$

and if $\left|\,\bar{\psi}\right\rangle$ is the state in the $|\,\alpha\rangle$, $|\,\beta\rangle$ plane orthogonal to $|\,\psi\rangle$, then

$$\left(2\,|\,\psi\rangle\langle\psi\,| - I\right)\left|\,\bar{\psi}\right\rangle = -\left|\,\bar{\psi}\right\rangle.$$

These two facts mean that $2\,|\,\psi\rangle - I$ is a reflection of the plane around the line through 0 and $|\,\psi\rangle$.

What is the product of two reflections in the plane?



If the two reflections are about lines that intersect at an angle of ϕ radians, then their product is a rotation around the intersection by an angle of 2ϕ radians. Thus, each Grover iteration rotates the state by approximately $2\frac{\theta}{2} = \theta \approx 2\frac{\sqrt{M}}{\sqrt{N}}$ radians. We would like the state to be rotated to the axis $|\,\beta\rangle$, which is at an angle of approximately $\pi/2$ from the original state $|\,\phi\rangle$. This will take roughly

$$\frac{\frac{\pi}{2}}{2\frac{\sqrt{M}}{\sqrt{N}}} \approx \frac{\pi}{4}\frac{\sqrt{N}}{\sqrt{M}},$$

So Grover's algorithm will find a marked state in approximately $\frac{\pi}{4}\frac{\sqrt{N}}{\sqrt{M}}$ iterations.

What do we do if we don't know M? If you have a vector at a random angle, and measure it in the $|\,\beta\rangle, |\,\alpha\rangle$ basis, then $\frac{1}{2}$ of the time you will get $|\,\beta\rangle$. That is, if you run Grover's algorithm for a random number of iterations, then as long as this number of iterations is sufficiently large, you have a $\frac{1}{2}$ chance of getting a marked state. Thus, one method for finding a marked state in $O(\frac{\sqrt{N}}{\sqrt{M}})$ iterations is to run it for a random number between 5 and 10 iterations, then a random number between 10 and 20 iterations, then a random number between 20 and 40 iterations, then a random number between 40 and 80, and so on. Eventually, not long after you reach roughly $\frac{\sqrt{N}}{\sqrt{M}}$ itertions, you are very likely to find a marked state, and you haven't taken more than $O(\frac{\sqrt{N}}{\sqrt{M}})$ iterations total.

Notes 8.370/18.435 Fall 2022

Lecture 25 Prof. Peter Shor

In this lecture, we will give a lower bound that shows that Grover's algorithm is nearly optimal. In particular, we assume that you have a quantum state space with basis $\{|1\rangle, |2\rangle, \ldots |N\rangle\}$ and an oracle function O_x such that

$$O_x |y\rangle = \begin{cases} -|y\rangle & \text{if } y = x \\ |y\rangle & \text{if } y \neq x\,. \end{cases}$$

We will show that any algorithm which always returns the output $|x\rangle$ must call the oracle on the order of $\sqrt{N}$ times. We first show that this is true if the algorithm is required to give the answer $|x\rangle$ with probability 1, and then show that the proof still works if we just require the answer with probability at least $1 - \epsilon$.

One amazing thing about this proof is that it was developed independently from Grover's algorithm, starting with the motivation of whether you can show that quantum computers can't solve NP-complete problems (although it was not published until after Grover's paper, and the original paper mentions Grover).

We model our algorithm as follows. First, by the principle of deferred measurement (see homework), we can assume that we postpone all the measurements in our algorithm until the end. The algorithm will then consist of unitary operations and calls to the oracle. We will define U_j so that the algorithm alternates calling the oracle O_x and implementing the unitary U_k. Suppose the algorithm requires t calls to the oracle in the worst case. Then, we define the algorithm to be:

$$|\psi_t^x\rangle = U_t O_x U_{t-1} O_x U_{t-2} \ldots U_1 O_x |\psi_0\rangle$$

If not all runs of the algorithm use the same number of calls to the oracle, we can always add more dummy calls to the oracle that in essence do nothing.

Now, suppose we are able to identify x with probability 1 after this algorithm. Then it must be the case that all the $|\psi_t^x\rangle$ are orthonormal. We will bound how quickly they are able to become distinct.

Define the state of the quantum computer after k oracle calls to be

$$|\psi_k^x\rangle = U_k O_x U_{k-1} O_x U_{k-2} \ldots U_1 O_x |\psi_0\rangle$$

We will define a measure of how quickly the states $|\psi_k^x\rangle$ are diverging from each other. In order to do this, let

$$|\psi_t\rangle = U_t U_{t-1} U_{t-2} \ldots U_1 |\psi_0\rangle\,,$$

so $|\psi_t\rangle$ is what we get when we run the algorithm with no oracle calls (or equivalently, when there are no solution states). Now, define

$$D_k = \sum_x \big|\, |\psi_k^x\rangle - |\psi_k\rangle \,\big|^2.$$

We will use D_k as a measure of how far the states $|\psi_k^x\rangle$ are from each other. In other to get a bound on the number of oracle calls we need, we will need to prove two things:

1. an upper bound on D_k,

2. a proof that if D_t is too small, then we cannot distinguish the $|\psi_x\rangle$ from each other well.

We first prove the upper bound for D_k.

We have

$$\begin{aligned} D_{k+1} &= \sum_x \left| U_{k+1} O_x \left|\psi_k^x\right\rangle - U_{k+1} \left|\psi_k\right\rangle \right|^2 \\ &= \sum_x \left| O_x(\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle) + (O_x - I)\left|\psi_k\right\rangle \right|^2 \\ &\leq \sum_x \left| O_x(\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle)\right|^2 + \left|(O_x - I)\left|\psi_k\right\rangle\right|^2 + 2\left|(O_x - I)\left|\psi_k\right\rangle\right| \cdot \left|O_x(\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle)\right|, \end{aligned}$$

where we have added and subtracted the term $O_x \left|\psi_k\right\rangle$, and used the inequality $|a+b|^2 \leq |a|^2 + |b|^2 + 2|b||a|$.

We will deal with the three terms in this inequality in order. For the first term, because unitary transformations preserve length,

$$\sum_x \left| O_x(\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle)\right|^2 = \sum_x \left| \left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle \right|^2 = D_k$$

For the second term, we have $(O_x - I)\left|\psi_k\right\rangle = -2\left\langle x|\psi_k\right\rangle \left|x\right\rangle$ because $O_x \left|y\right\rangle = \left|y\right\rangle$ if $y \neq x$ and $O_x \left|x\right\rangle = -2\left|x\right\rangle$. Thus,

$$\sum_x \left|(O_x - I)\left|\psi_k\right\rangle\right|^2 = \sum_x \left|2\left\langle x|\psi_k\right\rangle\right|^2 = 4;$$

because $\{\left|x\right\rangle\}$ is a basis. Finally, we have

$$2\sum_x \left|(O_x - I)\left|\psi_k\right\rangle\right| \cdot \left|O_x(\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle)\right| = 2\sum_x \left|\left\langle x|\psi_k\right\rangle\right| 2 \cdot \left|\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle)\right|.$$

Now, we use the Cauchy-Schwarz inequality, $v \cdot w \leq |v||w|$, on this term. Thus,

$$\begin{aligned} 2\sum_x \left|2\left\langle x|\psi_k\right\rangle\right| \cdot \left|\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle\right| &\leq 4\sqrt{\sum_x \left|\left\langle x|\psi_k\right\rangle\right|^2}\sqrt{\sum_x \left|\left|\psi_k^x\right\rangle - \left|\psi_k\right\rangle\right|^2} \\ &= 4\sqrt{D_k}\,. \end{aligned}$$

We thus have, substituting into the equation for D_{k+1},

$$D_{k+1} \leq D_k + 4\sqrt{D_k} + 4$$

We show by induction that this gives $D_k \leq 4k^2$. It's clear that $D_0 = 0$. Now, assume that this equation holds true for D_k. We have

$$D_{k+1} \leq D_k + 4\sqrt{D_k} + 4 \leq 4k^2 + 8k + 4 = 4(k+1)^2,$$

and we are done.

The next thing we need to do is show that D_t has to be large in order to identify x. We will assume that we find x with certainty; you can modify the proof so that this is not necessary. If we find x with certainty, then the $|\psi_t^x\rangle$ are othonormal for all N values of x. We will then show that $D_t \geq 2N - 2\sqrt{N}$.

Let's look at the equivalent problem where we have $|e_1\rangle$, $|e_2\rangle$, $|e_3\rangle$, ..., $|e_N\rangle$, and we want to find the vector $|v\rangle$ that minimizes $\sum_{i=1}^N |\,|e_i\rangle - |v\rangle\,|^2$. It turns out that $|v\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^N |e_i\rangle$, and one can calculate that for this value of $|v\rangle$, $\sum_{i=1}^N |\,|e_i\rangle - |v\rangle\,|^2 = 2N - 2\sqrt{N}$. Since $D_t \leq 2t^2$, this shows that $2t^2 \geq 2N - 2\sqrt{N}$, which implies that $t > \sqrt{N} - 1$.

We now show that this is indeed the minimum. We have that

$$|\,|e_i\rangle - |v\rangle\,|^2 = \langle e_i|e_i\rangle + \langle v|v\rangle + \langle e_i|v\rangle + \langle v|e_i\rangle \geq 2 - 2|\,\langle e_i|v\rangle\,|$$

so

$$\sum_{i=1}^N |\,|e_i\rangle - |v\rangle\,|^2 \geq 2N - 2\sum_i |\,\langle e_i|v\rangle\,|.$$

By Cauchy-Schwarz,

$$\sum_{i=1}^N |\,\langle e_i|v\rangle\,| \cdot 1 \leq \sqrt{\sum_{i=1}^N |\,\langle e_i|v\rangle\,|^2}\sqrt{\sum_{i=1}^N 1} = \sqrt{N}$$

Thus, we have

$$\sum_{i=1}^N |\,|e_i\rangle - |v\rangle\,|^2 \geq 2N - 2\sqrt{N},$$

and we are done.

In fact, you can show that even if you only require the algorithm to succeed half the time, it still must take order $\sqrt{N}$ steps; you can see the proof of this in the Nielsen and Chuang.

Notes 8.370/18.435 Fall 2022

Lecture 26 Prof. Peter Shor

In this lecture, we discuss an algorithm for simulating quantum dynamics. Before we can discuss the algorithm, we need to explain how continuous quamtum systems evolve.

There are two main differences between continuous quantum mechanics and the quantum mechanics of the circuit model.

1. Time is continuous (at least, most physicists believe it is up to the Planck scale, which is around 10^{-44} seconds).

2. Everything evolves at once, unlike the circuit model, where the gates operate on various qubits one at a time.

In the circuit model, we had discrete time. So $\psi_{t+1} = U_t\psi_t$. In actual quantum mechanics, we have the Schrödinger equation, which is the differential equation

$$\frac{d}{dt}\,|\,\psi(t)\rangle = \frac{-i}{\hbar} H\,|\,\psi(t)\rangle\,.$$

Here, H is the Hamiltonian, which gives the energy of the system. In the rest of this lecture, we will choose are units so that $\hbar = 1$, so we don't have all these $\hbar$s floating around.

Let's look at the equation more closely. The left side, $\frac{d}{dt}\,|\,\psi(t)\rangle$, has units of sec^{-1}, or Hertz. The right side, $H\,|\,\psi(t)\rangle$ has units of energy. We thus need a constant to relate them. This contant turns out to be $\hbar \approx 1.05 \times 10^{-34}$ Joule-sec. The reason that we do not generally observe macroscopic quantum effects is that this constant is so small.

How can we derive the Schrödinger equation? We know quantum mechanics is linear, so let's assume that the dynamics is given by a linear first-order differential equation. We then get

$$\frac{d}{dt}\,|\,\psi(t)\rangle = A\,|\,\psi(t)\rangle\,. \tag{1}$$

Now, let's differentiate $\langle\psi(t)|\psi(t)\rangle$. We then have

$$\begin{aligned}\frac{d}{dt}\langle\psi(t)|\psi(t)\rangle &= \left(\frac{d}{dt}\,\langle\psi(t)\,|\right)\,|\,\psi(t)\rangle + \langle\psi(t)\,|\left(\frac{d}{dt}\,|\,\psi(t)\rangle\right)\\ &= \langle\psi(t)\,|\,A^\dagger\,|\,\psi(t)\rangle + \langle\psi(t)\,|\,A\,|\,\psi(t)\rangle = \langle\psi(t)\,|\,(A^\dagger + A)\,|\,\psi t\rangle\ ,\end{aligned}$$

where the first equality comes from the product formula for differentiation, and the second from plugging in Eq. 1.

However, we know that $\langle\psi(t)|\psi(t)\rangle = 1$, and thus its derivative must be 0, so the expression above must be 0, no matter what $|\,\psi(t)\rangle$ is. The only way this could happen is for $A^\dagger + A = 0$. Thus, $A^\dagger = -A$. If we let $H = iA$, this makes H real and gives the differential equation

$$\frac{d}{dt}\,|\,\psi(t)\rangle = -iH\,|\,\psi(t)\rangle\,,$$

which is the Schrödinger equation with $\hbar = 1$. Solving this equation gives

$$|\,\psi(t)\rangle = e^{-iHt}\,|\,\psi(0)\rangle\,.$$

To simulate quantum dynamics, we have to simulate the real world, which behaves quite differently than our gate model, because we have continuous time and because everything evolves simultaneously. There are some other differences. (For example the real world evolves according to quantum field theory, where there is no non-local action. Two electrons repel each other not because there is action at a distance, but because they are coninually exchanging photons.) However, in this course we will stick with quantum mechanics, which is much less complicated.

Let's now discuss the Schrödinger equation some more, and give some examples. The equation (setting $\hbar = 1$) was

$$\frac{d}{dt}\,|\,\psi(t)\rangle = -iH\,|\,\psi(t)\rangle\,.$$

Recall that H is a Hermitian operator that gives the energy of the system $|\,\psi\rangle$. The solution of the Schrödinger equation is

$$|\,\psi(t)\rangle = e^{-iHt}\,|\,\psi(0)\rangle$$

We can expand this equation in a Taylor series which is reasonably accurate for small t:

$$e^{-iHt} = I - iHt - \frac{1}{2}H^2t^2 + \ldots$$

There is another way of looking at the solutions of the Schrödinger equation.

For our first example, let's look at a qubit. Any 2×2 complex matrix can be expressed as

$$v_x\sigma_x + v_y\sigma_y + v_z\sigma_z + v_I\sigma_I$$

However, we can disregard the last term. Consider what happens if we have a Hamiltonian of vI. The energy of a state $|\,\psi\rangle$ is $\langle\psi\,|\,vI\,|\,\psi\rangle = v$, and the evolution is

$$|\,\psi(t)\rangle = e^{-vt}\,|\,\psi_0\rangle\,.$$

Because a global phase doesn't affect the state, this state is essentially $|\,\phi_0\rangle$. I won't work out the details here, but similarly adding a term proportional to I just adds a global phase, which can be ignored. So the absolute energy of a system is not important for quantum evolution; the only thing that mastters is the relative energy of the states of the system. Thus, we can assume that this Hamiltonian is $\vec{v}\cdot\vec{\sigma}$, where $\vec{\sigma} = \{\sigma_x, \sigma_y, \sigma_z\}$

Now, let's look at how we might simulate the dynamics of a qubit evolving under a Hamiltonian. Consider two times, t and $t + \Delta t$. We have that

$$\begin{aligned}
|\,\psi(t+\Delta t)\rangle &= e^{iH\Delta t}\,|\,\psi(t)\rangle \\
&\approx (I + iH\Delta t)\,|\,\psi(t)\rangle \\
&= |\,\psi(t)\rangle + i\Delta t H\,|\,\psi(t)\rangle
\end{aligned}$$

This equation is quite easy to simulate, and for small t, it should work fairly well.

Another thing we can do is write $|\psi_0\rangle$ in the basis of eigenstates of H. Let $|v_E\rangle$ be the eigenstate of energy E, and let

$$|\psi_0\rangle = \sum_E c_E |v_E\rangle .$$

Then

$$|\psi_t\rangle = \sum_E c_E e^{-iEt} |v_E\rangle$$

Exercise: using these ideas, show that when we have the Hamiltonian $\vec{v}\cdot\vec{\sigma}$, the state $|\psi\rangle$ rotates around the vector $\vec{v}$.

Now, let's look at another Hamiltonian; specifically, one called the transverse Ising model. This is a Hamiltonian on a *spin chain* — we have qubits at sites $0, 1, \ldots, N$, and the Hamiltonian is

$$\sum_{j=0}^{N-1} \sigma_z^j \sigma_z^{j+1} - g \sum_{j=0}^{N} \sigma_x^j,$$

where σ_x^j is a Pauli matrix acting on site j.

This is a *local Hamiltonian*; this means that the Hamiltonian is a sum of terms, each of which acts only on a small (in this case, two) number of qubits. Here, the Hamiltonian can be expressed as $H = A + B$, where the local terms in A all commute (they are all composed of σ_z's, which all commute) and the local terms in B (composed of σ_x's) all commute. We will give the Trotter-Suzuki method, which can be used to simulate such Hamiltoniaans.

The first thing to realize is that if all the local terms H_j of the Hamiltonian commuted, there would be no problem—we could simulate them all sequentially; because they commute, applying the unitaries e^{-iH_j} in any order would give the right answer.

We will show that by alternately applying the A Hamiltonian and the B Hamiltonian, we can achieve a good approximation of the $A + B$ Hamiltonian.

First, let's compare applying the $A+B$ Hamiltonian for time t with applying the A Hamiltonian and then the B Hamiltonian for time t. Using the Taylor approximation, we get

$$\begin{aligned} e^{-i(A+B)t} &\approx I - i(A+B)t - (A+B)^2 t^2 + O(t^3) \\ &= I - i(A+B)t - \frac{1}{2}(A^2 + B^2 + AB + BA)t^2 + O(t^3) \end{aligned}$$

and

$$\begin{aligned} e^{-iAt}e^{-iBt} &\approx \big(I - iAt - \frac{1}{2}A^2t^2 + O(t^3)\big)\big(I - iBt - \frac{1}{2}B^2t^2 + O(t^3)\big) \\ &= I - i(A+B)t - \frac{1}{2}(A^2 + B^2 + 2AB)t^2 + O(t^3) \end{aligned}$$

We can then see that the commutator

$$[e^{-(A+B)t}, e^{-iAt}e^{-iBt}] = -\frac{1}{2}[A,B]t^2 + O(t^3).$$

For small t, this is small.

What we will do is compute the evolution

$$\left(e^{-iA\frac{t}{L}}e^{-iB\frac{t}{L}}\right)^L \approx \left(e^{-i(A+B)\frac{t}{L}}\right)^L$$

Intuitively, each of the L terms of this product differs from the correct evolution by order $\left(\frac{t}{L}\right)^2$. When we multiply all L terms together, this increases the difference by a factor of L, so the approximation is good to $O\left(\frac{t^2}{L}\right)$. To make this rigorous, we first need to decide hwo we will measure the difference, and then show that our intuition above holds.

What we use to measure the difference is the operator norm. This norm is

$$\|R\| = \max_{|\,\psi\rangle}\{|R\,|\,\psi\rangle\,| : |\,|\,\psi\rangle\,| = 1\}$$

If we want to approximate R by S, we measure how good our approximation is by the matrix $\|R - S\|$.

We will need several properties of the operator norm. These are

$$\|R + S\| \leq \|R\| + \|S\|;$$

For U unitary,

$$\|UR\| = \|AR\| = \|A\|;$$

and

$$\|R_1R_2\ldots R_m - S_1S_2\ldots S_m\| \leq \sum_{j=1}^{M}\|R_j - S_j\|$$

The last of these properties shows that

$$\|e^{-i(A+B)t} - e^{-At}e^{-Bt}\| \leq L\left\|\left(e^{-i(A+B)\frac{t}{L}} - e^{-A\frac{t}{L}}e^{-B\frac{t}{L}}\right)\right\|$$

Thus, if we divide the time t into L subintervals of length $\frac{t}{L}$, we can make the error of order $\frac{t^2}{L}$. By making L large enough, we can make the approximation as accurate as desired. We thus have that to get the error down to ϵ, we need to set $\epsilon = O\left(\frac{t^2}{L}\right)$, or $L = O\left(\frac{t^2}{\epsilon}\right)$.

There are also better Trotter-Suzuki formulas which work with higher order. I am only going to give the second-order one in this lecture (they get quite a bit more complicated as the orders get higher. Here, instead of repeating the evolution $e^{iAt}e^{iBt}$, we repeat the evolution

$$e^{-\frac{1}{2}iAt}e^{-iBt}e^{-\frac{1}{2}iAt}$$

Expanding the exponentials in Taylor series, we have

$$e^{-\frac{1}{2}iAt}e^{-iBt}e^{-\frac{1}{2}iAt} = (I - \frac{1}{2}iA - \frac{1}{8}A^2\ldots)(I - iB - \frac{1}{2}B^2\ldots)(I - \frac{1}{2}iA - \frac{1}{8}A^2\ldots)$$

Expanding and collecting terms, we find that

$$e^{\frac{1}{2}iAt}e^{iBt}e^{\frac{1}{2}iAt} = I - \frac{1}{2}i(A+B) - \frac{1}{2}(A^2 + AB + BA + B^2)t^2 + O(t^3),$$

which agrees with $e^{-i(A+B)}$ up to an error of $O(t^3)$.

Now, if we divide the time up into L intervals of size $\frac{t}{L}$, we see that

$$e^{-\frac{1}{2}iA\frac{t}{L}}e^{-iB\frac{t}{L}}e^{-\frac{1}{2}iA\frac{t}{L}}$$

is accurate to order $L\left(\frac{t}{L}\right)^3$. So to get accuracy of ϵ, we need to set $\epsilon = O\left(\frac{t^3}{L^2}\right)$, i.e., make L of order $\frac{t^{3/2}}{\epsilon^{1/2}}$.

Notes 8.370/18.435 Fall 2022

Lecture 27 Prof. Peter Shor

Today, we start our unit on quantum error-correcting codes.

I'm going to start with a digression into classical information theory. In 1948, Claude Shannon published a paper, "A Mathematical Theory of Communication", which started the field of information theory. In it, he showed that noisy communication channels had a *capacity*, and he derived *Shannon's formula* for the capacity of a channel. There is a protocol such that you can send information over a channel at a rate less then its capacity, in the limit as the length of the message goes to infinity, and almost always succeed. On the other hand, if you try to send information over a channel at a rate larger than the capacity, it is overwhelmingly likely that the message the receiver gets has errors—i.e., it is different from the message that the sender transmitted.

Claude Shannon's theory was not constructive. While he showed that an algorithm existed that would succeed in transmitting information at nearly the channel capacity existed, it was a randomized construction that didn't explicitly produce such an algorithm. Worse, the algorithm itself would take exponential time to implement in the most straightforward way. It wasn't until forty-five years after Shannon's paper that somebody experimentally found an efficient coding and decoding procedure that came close to the channel capacity (*turbo codes*), and not until 60 years later until a coding method was found that provably approached Shannon's bound in the limit of long messages (these were *polar codes*).

However, only two years after Shannon published his paper, Richard Hamming discovered the first error-correcting codes, and in the next few decades, computer technology advanced greatly and error correcting codes were developed to the point where they could be used in practice for the reliable transmission of information over noisy channels. Hamming codes are one of a class of codes called *linear codes*, and these have nice properties. They are the class of codes that is most often used in practice, and they have received extensive study.

We will look at the code that Hamming discovered much more closely in future classes, but I want to explain a little bit about how it works right now. You take a four-bit message, and encode it into a seven-bit message in such a way that even if one of the seven bits is wrong, you can still recover the original four bits. The encoding is done by multiplying the message m by a generator matrix G, where

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

The decoding is done by multiplying by a matrix H, where

$$H = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

We have $GH = 0 (\mathrm{mod}\, 2)$. This is because $G = \begin{pmatrix} I_4 & S \end{pmatrix}$ and $H = \begin{pmatrix} S \\ I_3 \end{pmatrix}$, so $GH = 2S = 0$, where S is the 4×3 matrix

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

Suppose that there is a one-bit error in the received transmission r. Then

$$r = mG + e$$

and

$$rH = mGH + eH = eH,$$

so $rH = eH$ is independent of the message, and tells you what the error is. It is called the *syndrome* (because you can use it to diagnose the error). Computing the error e from the syndrome eH is computationally difficult problem in general; the hard part of designing error-correcting codes is finding codes where there is an efficient algorithm going from the syndrome to the error.

Before Hamming, the only error-correcting codes engineers knew about were *repetition codes*. In these, you just repeat the message bit k times. If $k = 2t + 1$, the code can correct t errors. These codes can be put into the framework of linear codes described above. For a repetition code3, the generator matrix is

$$G = [1, 1, 1, 1, \ldots, 1].$$

Let's look at the three-bit repetition more carefully. Assume that each bit has a p probability of having an error, and a $1 - p$ probability of being correct. Then there will be an error in an encoded bit if and only if at least two of the encoding bits have errors, so instead of a probability p of having an error, the probability is $3p^2(1 - p) + p^3$, which is an improvement as long as $p < \frac{1}{2}$, and is around $3p^2$ if p is small.

Today, we will look at the quantum analog of repetition codes. Since it is impossible to clone a qubit, you can't actually implement a repetition code $|\psi\rangle \to |\psi\rangle\,|\psi\rangle\,|\psi\rangle$. One thing you could do instead is use the unitary:

$$U\,|0\rangle\,|00\rangle = |000\rangle$$
$$U\,|1\rangle\,|00\rangle = |111\rangle$$

Here, we first adjoint the qubits $|00\rangle$ to the qubit we want to encode and then perform the unitary.

This is a three-qubit code that protects against bit-flip errors. It was first investigated by Asher Peres in 1985. The code maps

$$\alpha\,|0\rangle + \beta\,|1\rangle \to \alpha\,|000\rangle + \beta\,|111\rangle\,,$$

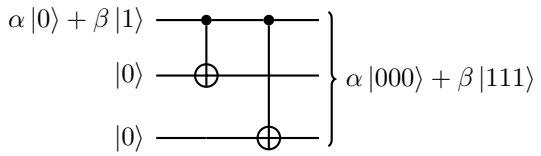
so you can see it does not perform quantum cloning; if you cloned the original qubit, you would get $\big(\alpha\,|0\rangle + \beta\,|1\rangle\big)^{\otimes 3}$.

This code can correct one bit-flip, or σ_x, error. How does this work? We measure the answer to the question "which bit is different?" More specifically, we project the three qubits onto one of the following four subspaces:

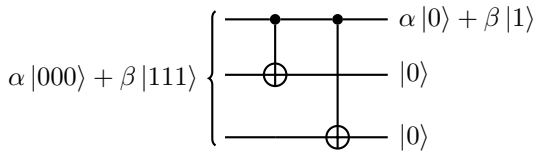
$$\begin{aligned}
&|000\rangle\langle 000| + |111\rangle\langle 111| \\
&|100\rangle\langle 100| + |011\rangle\langle 011| \\
&|010\rangle\langle 010| + |101\rangle\langle 101| \\
&|001\rangle\langle 001| + |110\rangle\langle 110|\,.
\end{aligned}$$

Once we know which subspace are in, we can correct the error. For example, if the state was projected onto the third subspace above, we would apply a σ_x to the second qubit to correct it.

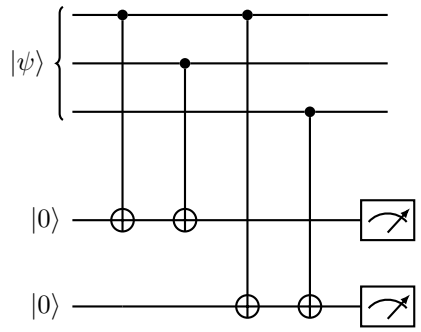
How do we encode a qubit into this code? We use the following circuit:



How do we decode? We basically reverse the circuit above, which gives exactly the same circuit since the two CNOT gates commute with each other:



How do we correct errors? We use the following quantum circuit, that projects onto one of the four subspaces described above:



The two measurement results here are called the *syndrome* and tell us what the error is.

If the two measurements are both $|0\rangle$, then we know that $|\psi\rangle$ was in the code subspace. If the first one is $|1\rangle$ and the second is $|0\rangle$, we know that the first and third qubits are qual, but the first and second qubits are different. This means that the second qubit must be different. Thus, if there is only one error, it is on the second qubit, and we can correct the state by applying σ_x to the second qubit. Similarly, if we get the measurement results $(0, 1)$, we know the third qubit is different, and if the measurement results are $(1, 1)$, we know the first qubit is different.

For bit-flip errors, the probabilities work exactly the same way as they did for the classical three-bit code. If the probabililty of a bit-flip error on a single qubit is p, then the probability of a bit-flip error on the encoded qubit is $3p^2 + p^3$, which is approximately $3p^2$ for small p.

But what about phase-flip errors? What happens if we apply a σ_z to one of the three qubits in the code? We will call an encoded $|0\rangle$ and $|1\rangle$ a *logical* $|0\rangle$ and $|1\rangle$, and we will represent them by

$$|0\rangle_L = |000\rangle \qquad\qquad |1\rangle_L = |111\rangle$$

If we apply a phase-flip error in any of three three encoding qubits, it will take $|0\rangle$ to $|0\rangle$ and $|1\rangle$ to $-|1\rangle$. That is, it will apply a phase error to the logical qubit. So if the probability of a phase error on a single qubit is p, the probability of a phase error on the encoded qubit is $3p + p^3$, or approximately $3p$ for small p.

So we can reduce the error rate for bit flip errors at the cost of increasing the error rate for phase flip errors. Is there anything we can do to protect phase-flip errors?

There is. Recall that a Hadamard gate took a σ_x to a σ_z and vice versa. That is, it takes a bit flip error to a phase flip error and vice versa. Thus, we can find a code that interchanges the role of bit-flip and phase-flip errors by applying a Hadamard gate to each of our qubits. This code is simply

$$|0\rangle \to |+++\rangle$$
$$|1\rangle \to |---\rangle\,.$$

There is an equivalent way to represent this code, namely:

$$|0\rangle_L \to \frac{1}{2}\big(\,|000\rangle + |011\rangle + |101\rangle + |110\rangle\,\big),$$
$$|1\rangle_L \to \frac{1}{2}\big(\,|100\rangle + |010\rangle + |001\rangle + |111\rangle\,\big).$$

We obtain this alternate representation by applying a Hadamard gate to the encoded qubit before encoding it. This code encodes a $|0\rangle$ as the superposition of all states with an even number of 0's, and a $|1\rangle$ as the superposition of all states with an odd number of 1s. It protects against phase-flip errors, but any bit-flip error in an encoding qubit results in a bit-flip error on the logical qubit, so bit-flip errors are roughly three times as likely as on unencoded qubits. This can be seen directly from the fact that a bit-flip error takes a bit string with an odd number of 1's to a bit string with an even number of 1's.

So now we have a choice: we can protect against one bit-flip error, but only by making phase-flip errors more likely, or we can protect against one phase-flip errors but only by making bit-flip errors more likely. Is there any way around this problem?

It turns out there is. The answer comes from a technique of classical coding theory: you *concatenate* the two codes. This means you first encode using one code, and then you encode using the other. Let's see how this works.

$$|0\rangle \to |+++\rangle \to (|000\rangle + |111\rangle)^{\otimes 3}$$
$$|1\rangle \to |---\rangle \to (|000\rangle - |111\rangle)^{\otimes 3}$$

Now, what happens? if you have a bit-flip error, it gets corrected immediately by the inner code. If you have a phase-flip error, the inner code turns this into a phase-flip on the logical qubit of the inner code, which gets corrected by the outer code. Thus, any single σ_x or σ_z can be corrected.

How about σ_y errors? They can be corrected as well. A σ_y error can be viewed as a σ_x error and a σ_z error acting on the same qubit, since $\sigma_y = i\sigma_x\sigma_z$. Thus, any single σ_y error can be corrected as well—the inner code will correct the σ_x component and the outer code the σ_z component.

But what about more general errors? It turns out that the 9-qubit code given in this lecture can correct these as well, as long as they are restricted to one qubit. We will see this in the next lecture.

Notes 8.370/18.435 Fall 2022

Lecture 28 Prof. Peter Shor

We continue our discussion of the nine-qubit code.

Recall that last time, we started our discussion with the three-qubit bit flip correcting code. This was based on the classical repetition code, that just repeats every bit three times. This code takes

$$|0\rangle \to |000\rangle,$$
$$|1\rangle \to |111\rangle.$$

This is not a cloning transformation, because $\alpha|0\rangle + \beta|1\rangle \to \alpha|000\rangle + \beta|111\rangle$. This code corrected one bit-flip error, but made phase-flip errors more likely. We then used the fact that $H\sigma_x H = \sigma_z$ to get a three-qubit phase-flip correcting code. This code is

$$|0\rangle \to |+++\rangle,$$
$$|1\rangle \to |---\rangle.$$

or in the $\{|0\rangle, |1\rangle\}$ basis,

$$|0\rangle \to \frac{1}{\sqrt{8}}(|0\rangle + |1\rangle)^{\otimes 3},$$
$$|1\rangle \to \frac{1}{\sqrt{8}}(|0\rangle - |1\rangle)^{\otimes 3}.$$

A bit-flip error (σ_x) on any qubit results in a phase flip error on the encoded state: It will change one of the $(|0\rangle - |1\rangle)$ terms to $(|1\rangle - |0\rangle)$, which changes a $|1\rangle_L$ to a $-|1\rangle_L$, and leaves an encoded $|0\rangle$ the same.

On the other hand, this will correct any phase-flip (σ_z) error on a single qubit. Why is this true? It's because the eight states

$$|0\rangle_L, \quad \sigma_z^{(1)}|0\rangle_L, \quad \sigma_z^{(2)}|0\rangle_L, \quad \sigma_z^{(3)}|0\rangle_L$$
$$|1\rangle_L, \quad \sigma_z^{(1)}|1\rangle_L, \quad \sigma_z^{(2)}|1\rangle_L, \quad \sigma_z^{(3)}|1\rangle_L$$

are all orthogonal, where $\sigma_z^{(j)}$ denotes a σ_z error on qubit j.

To correct the state, we project it onto one of the four subspaces:

$$|0_L\rangle\langle 0_L| + |1_L\rangle\langle 1_L|$$

and

$$\sigma_z^{(j)}\big(|0_L\rangle\langle 0_L| + |1_L\rangle\langle 1_L|\big)\sigma_z^j,$$

for $j = 1, 2, 3$.

We combined these two codes by *concatenating* them. This means first encoding by using the phase-flip code (it would work just as well if we used the bit-flip code

first, but this is the standard way to do it) and then encoding each of the qubits in the resulting three-qubit code by the bit-flip code:

$$|\,0\rangle \rightarrow |\,+\rangle^{\otimes 3} \rightarrow \frac{1}{\sqrt{8}}\big(\,|\,000\rangle + |\,111\rangle\,\big)^{\otimes 3},$$
$$|\,1\rangle \rightarrow |\,-\rangle^{\otimes 3} \rightarrow \frac{1}{\sqrt{8}}\big(\,|\,000\rangle - |\,111\rangle\,\big)^{\otimes 3}.$$

This results in a nine-qubit code. It corrects both bit and phase errors. This nine-qubit code can correct one Pauli error on any qubit. One σ_x error is corrected by the bit-flip correcting code. One σ_z error passes through the bit-flip correcting code to apply a σ_z error to one of the groups of three qubits, which then gets corrected by the phase-error correcting code. And a σ_y error can be thought of as both a σ_x and a σ_z error on the same qubit, since $\sigma_y = i\sigma_x\sigma_z$, so the bit-flip correcting code corrects the σ_x error and the phase-flip correcting code corrects the σ_z error..

But what about arbitrary errors? You can have arbitrary unitary errors, or you can have a measurement on qubits, or you can have a more general type of quantum transformation that we haven't covered in this class (but which you will see if you take 8.371/18.436). It turns out that this code will correct them, as well. This is because of the following theorem:

Theorem 1 *Any quantum error-correcting code, which corrects t or fewer Pauli errors (σ_x, σ_y, and σ_z errors) on a subset of t or fewer qubits will also correct an arbitrary quantum operation which is applied to at most t qubits.*

That the 9-qubit code will correct any arbitrary single-qubit error follows from the above theorem with $t = 1$.

How do we prove this theorem? We will first show why it works by looking at an example on the three-qubit phase-flip error–correcting code, and then prove it.

Let's consider a qubit $\alpha\,|\,0\rangle + \beta\,|\,1\rangle$ encoded in the three -qubit phase-flip correcting code:

$$\alpha\,|\,0\rangle + \beta\,|\,1\rangle \rightarrow \alpha\,|\,0\rangle_L + \beta\,|\,1\rangle_L$$

What happens when we apply the phase error $\begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix}$ to the second qubit of it? We have

$$\begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta - i\sin\theta & 0 \\ 0 & \cos\theta + i\sin\theta \end{pmatrix} = \cos\theta I - i\sin\theta\sigma_z$$

So we get the state

$$\cos\theta\big(\alpha\,|\,0_L\rangle + \beta\,|\,1_L\rangle\,\big) - i\sin\theta\,\sigma_z^{(2)}\big(\alpha\,|\,0_L\rangle + \beta\,|\,1_L\rangle\,\big)$$

When we measure which qubit is in error, we get that there is no error with probability $\cos^2\theta$ and that there is a σ_z in qubit 2 with probability $\sin^2\theta$. And in fact, we collapse the state, so after the measurement, this will indeed be the case. When we correct the σ_z error on qubit 2, this restores the original state.

Why did this happen? The reason is that the error matrix has a decomposition in I and σ_z:

$$\begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix} = (\cos\theta)I - i(\sin\theta)\sigma_z\,.$$

When you apply it, you get a superposition of applying the identity matrix with amplitude $\cos\theta$ and the σ_z matrix with amplitude $-i\sin\theta$. Now, when you perform the error correction protocol, you measure the error, and find out that there was no error with probability $\cos^2\theta$ and a $\sigma_z^{(2)}$ error with probability $\sin^2\theta$. However, after the quantum state has collapsed, this is indeed the situation.

So how do we prove Theorem 1? We prove that if an error-correcting code can correct errors described by matrices M_1, M_2, M_3, ..., M_k, then it can correct errors described by any linear combination of these. Then we show that any error on t qubits is a linear combination of Pauli errors on t qubits.

The first step is just an application of the linearity of quantum mechanics. Consider an error correcting circuit. Then we can apply the principle of delayed measurement to postpone any measurements until the end. Here, instead of measuring the error classically and applying Pauli matrices (say) to correct it, you measure the error coherently and then use controlled Pauli gates to correct the error. This gives us a unitary which takes

$$M_i\,|\,\psi\rangle\,\big|\,0^l\big\rangle \longrightarrow |\,\psi\rangle\,|\,D_i\rangle$$

where $|\,D_i\rangle$ is essentially a description of the error. Then, for an error F, if we have $F = \sum_i M_i$, we can correct it. This is because the error correction circuit takes

$$F\,|\,\psi\rangle\,\big|\,0^k\big\rangle = \sum_i \alpha_i M_i\,|\,\psi\rangle\,\big|\,0^k\big\rangle \longrightarrow |\,\psi\rangle\sum_i \alpha_i\,|\,D_i\rangle\,.$$

This calculation also shows how error correction gets around the Heisenberg Uncertainty Principle, which says that if you measure a quantum state, you disturb it. What error correction does is measure the error without measuring the encoded quantum state. This lets you correct the error without measuring the quantum state.

Finally, let me address the question of what happens if you have a small error on every qubit. For example, suppose you have n qubits, and each qubit has an independent error where the error on the ith qubit is

$$F_i = (1-\epsilon_i)I + \delta_{x,i}\sigma_x^{(i)} + \delta_{y,i}\sigma_y^{(i)} + \delta_{z,i}\sigma_z^{(i)}.$$

What you do is expand the tensor product $\bigotimes_i F_i$. If the δ's are small enough, then most of the amplitude of this product will be in terms which have relatively few Pauli errors, so if you can correct (say) any tensor product of fewer than $n/100$ Pauli errors, then if the δ's are small enough, nearly all the time, when you measure the error there will be Pauli errors on fewer than $n/100$ qubits, and the probability that you make an error that is too large to be corrected will be exponentially small.

Notes 8.370/18.435 Fall 2022

Lecture 29 Prof. Peter Shor

Today, we're talking about the seven-qubit Hamming code. This Hamming code is probably the simplest CSS code; it's certainly the shortest CSS code which corrects one error. In lecture, I tried to present both the Hamming code and CSS codes at the same time, but thinking about my presentation, I think it would work better to do the quantum Hamming code first and the more general CSS code construction later, so I'm doing that in these notes.

First, we need to explain the classical Hamming code. The codewords of the Hamming code are the binary linear combinations of the generator matrix G:

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

There are 2^4 of these combinations, since there are 4 rows.

We consider a code C to be the set of codewords of the code. There are many different generator matrices that give essentially the same code.

The Hamming code encodes four bits into seven bits. To encode, we take a four-bit message m and multiply G by it to get the corresponding codeword:

$$c = mG\,.$$

We've been working with quantum mechanics, where you use left multiplication, so it may seem strange to be using right multiplication (the vector is on the left and the matrix is on the right). However, this is the standard convention in coding theory, and if I try to reverse it I suspect I would get very confused.

How do you decode it? Choose some set of columns that are linearly independent. Let's choose the first four columns. The matrix G restricted to these four columns is

$$\hat{G} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Because the first four columns are linearly independent, the matrix $\hat{G}$ has an inverse (mod 2), that we will call $\hat{G}^{-1}$. You can find it using Gaussian elimination. In this case,

$$\hat{G}^{-1} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

and $\hat{G}^{-1}\hat{G} = I$ (mod 2).

To decode, take $\hat{c}$, the first four entries of the codeword, and apply $\hat{G}^{-1}$ to it. This gives m because

$$\hat{G}^{-1}\hat{c} = \hat{G}^{-1}\hat{G}m = Im = m$$

The Hamming code is just the simplest non-trivial example of a large class of classical error-correcting codes, called *linear codes*. In these codes, the codewords are binary linear combinations of the rows of some generator matrix G. In this lecture, we will talk about classical linear codes, and then discuss the 7-qubit quantum Hamming code, which encodes one qubit into seven qubits and corrects one error. In the next lecture, we will talk about quantum CSS codes, which are the simplest generalization of classical linear codes to quantum codes. A broader generalization, *stabilizer codes*, exists, but we won't discuss them in these notes.

How do we correct errors for the Hamming code? We use the *parity check* matrix H, which is a generator matrix for the dual space of the code. If you have a vector space V, then the dual space is

$$V^{\perp} = \{w | w \cdot v = 0 \; \forall v \in V\}.$$

If you've used to working with vector spaces over $\mathbb{R}$ or $\mathbb{C}$, one thing that is confusing about binary vector spaces is that the dual can overlap with the original, so some vectors can be in both V and $V^{\perp}$. However, even though this is true, it is still the case that $\dim V + \dim V^{\perp} = n$, as happens in vector spaces over $\mathbb{R}$.

Why can the Hamming correct one error? It's because the minimum non-zero codeword has Hamming weight 3. Here, the *Hamming weight* of a codeword is the number of non-zero coordinates of a codeword (for binary codewords, the number of 1's). The *Hamming distance* $d_H(c_1, c_2)$ between two codewords c_1 and c_2 is the Hamming weight of $c_1 - c_2$, in other words, the number of coordinates where c_1 and c_2 differ.

We now prove

Theorem 1. *Suppose that the minimum non-zero weight of a codeword in code C is d. Then the code C can correct $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ errors and detect $d-1$ errors.*

Proof: We claim that the a word w cannot be within Hamming distance t of two different codewords c_1 and c_2. Suppose it was, then

$$d_H(c_1, c_2) \leq d_H(c_1, w) + d_H(w, c_2) \leq 2 \left\lfloor \frac{d-1}{2} \right\rfloor \leq d - 1,$$

but $c_1 - c_2$ is a codeword, and has Hamming weight $d_H(c_1, c_2)$, which must be at least d. Since we just showed that $d_H(c_1, c_2) < d$, this is a contradiction. Thus, for any word, there is at most one codeword within distance t of it. So if a codeword has t or fewer errors, we can correct it to the unique codeword within Hamming distance t.

If a codeword has fewer than $d-1$ errors, then the errors cannot take it to another codeword, so we can detect that there is an error (although we cannot necessarily correct it to the original codeword). $\square$.

The dual of a code C , is $C^{\perp}$, the set of words that are perpendicular to every codeword in C. That is,

$$C^{\perp} = \{x | x \cdot c = 0 \forall c \in C\}.$$

A generator matrix for $C^\perp$ is the transpose of a parity check matrix for C. We typically use the variable G to denote a generator matrix and H to denote a parity check matrix.

A parity check matrix for the Hamming code is H^T, where

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Note that the rows of H are the same as the first three rows of G, so for the Hamming code, $C^\perp \subset C$. Since all the rows of H are perpendicular to the rows of G, the parity check matrix has the property that $GH^T = 0$, which is important for error correction.

The Hamming code is called a $(7, 4, 3)$ code because it maps 4 bits into 7 bits and has minimum weight 3. The code whose generator matrix is H is a $(7, 3, 4)$ code. In fact, every non-zero codeword has Hamming weight 4; this is not hard to see. And knowing this, when we add the last row of G, we get a codeword where every word except the zero codeword and the all-ones codeword has Hamming weight either 3 or 4.

How do we correct an error in the Hamming code? The Hamming code can correct one error, which we will represent as a vector such as $e = (0, 0, 1, 0, 0, 0, 0)$. If the error is in a single bit, then only one coordinate of e will be 1 and the rest will be 0; here, the error is in the third bit. Now, let us suppose we receive a coded message with an error. This received message can be represented as $r = mG + e$. To correct the error, we multiply by the matrix H^T:

$$rH^T = (mG + e)H^T = eH^T,$$

since $GH^T = 0$. The string eH^T is called the *syndrome*, because this is what we use to diagnose the error. Since e contains a single 1 in (say) the k'th position, eH will be the k'th row of H^T.

Let's do an example.

Suppose we receive the noisy codeword $(1, 0, 0, 1, 0, 1, 0)$. How do we find the error? We multiply by H^T, so

$$\Big(1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0\Big) \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \Big(0,\ 1,\ 1\Big)$$

Now, $rH^T = \big(0,\ 1,\ 1\big)$, which is the third row of H^T. Thus, the third bit of our received codeword is in error. and the correct codeword is $\big(1,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0\big)$. Linear algebra shows that this is the encoding of message $\big(1,\ 0,\ 1,\ 0\big)$

Recall that for the Hamming code, $C^\perp \subset C$. A code with $C \subseteq C^\perp$ is called *weakly self-dual*, and these codes are important for building quantum error-correcting codes.

So what is the quantum Hamming code? It will encode one qubit into seven qubits. The logical $|0\rangle$ is the superposition of all eight of the codewords in H:

$$|0\rangle_L = \frac{1}{\sqrt{8}} \sum_{c \in H} |c\rangle \, .$$

The logical $|1\rangle$ is the superposition of the eight codewords that are in G and not in H. These are the codewords of H XORed with the all 1's vector:

$$|1\rangle_L = \frac{1}{\sqrt{8}} \sum_{c \in H} |c \oplus 1111111\rangle \, .$$

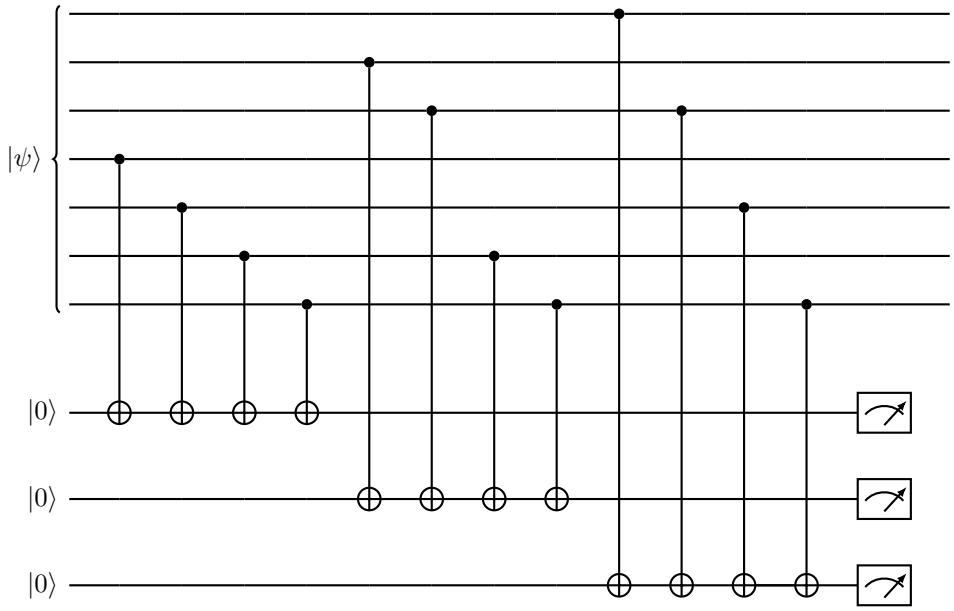
.

Why does this correct an arbitrary one-bit error? To show this, it turns out that we only need to show that it corrects a single σ_x error and a single σ_z error. Because this code treats σ_x errors and σ_z errors separately, it will automatically correct a single σ_y error, and from this we can deduce that it will correct an arbitrary single-qubit error. (This was proven in the lecture notes on the nine-qubit code.)

Why does it correct one bit-flip error? This state is a superposition of elements of the classical Hamming code, and we can apply the classical Hamming error correction algorithm to each of these elements so as to correct a single bit-flip error. We do this by computing the syndrome. If b_i is the i'th bit, then the syndrome is:

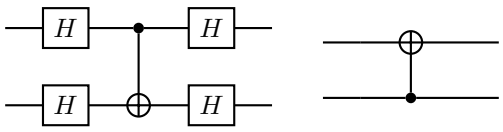
$$(b_4 \oplus b_5 \oplus b_6 \oplus b_7, \;\; b_2 \oplus b_3 \oplus b_6 \oplus b_7, \;\; b_1 \oplus b_3 \oplus b_5 \oplus b_7).$$

This can be computed by the following quantum circuit:



Once the syndrome has been computed, the error needs to be determined and then corrected. The determination of the error from the syndrome can be done classically.

Why does the Hamming code correct one phase-flip error? Recall that the Hadamard gate H takes bit-flip errors to phase-flip errors and vice versa. So we need to compute what happens when we apply a Hadamard gate to every qubit of the code. It turns out that we get the same quantum code back. Thus, the circuit for measuring the syndrome for phase flips is the same as the circuit for measuring the syndrome for bit flips, except that we put seven Hadamard gates on the qubits in the quantum code at the front of the circuit, and seven more at the back of the circuit. Alternatively, we can just use the identity



to turn all the CNOT gates upside-down.

We haven't actually done the calculations to show that $H^{\otimes 7}$ gives the same quantum code back. Let's see what happens when we apply the Hadamard transform to the quantum code. We have

$$|0\rangle_L = \frac{1}{\sqrt{8}} \sum_{x \in H} |x\rangle .$$

Applying the Hadamard transform gives

$$H^{\otimes 7} |0\rangle_L = \frac{1}{\sqrt{8}} \frac{1}{\sqrt{2^7}} \sum_{x \in H} \sum_{y \in Z_2^7} (-1)^{x \cdot y} |y\rangle .$$

Now, we need to figure out what $\sum_{x \in H} (-1)^{x \cdot y |y\rangle}$ gives. If $y \in G$, then since every element of G is orthogonal to every element of H, all the phases are $+1$, and we get $8 |y\rangle$. If $y \notin G$, then half of the phases will be -1 and half will be $+1$, so they cancel out. Thus,

$$H^{\otimes 7} |0\rangle_L = \frac{8}{\sqrt{8}} \frac{1}{\sqrt{2^7}} \sum_{y \in G} |y\rangle = \frac{1}{4} \left(\sum_{y \in H} |y\rangle + \sum_{y \in G \setminus H} |y\rangle \right) = \frac{1}{\sqrt{2}} \big(|0\rangle_L + |1\rangle_L \big)$$

What about $H^{\otimes 7} |1\rangle_L$? Here,

$$H^{\otimes 7} |1\rangle_L = \frac{1}{\sqrt{8}} \frac{1}{\sqrt{2^7}} \sum x \in G \setminus H \sum_{y \in Z_2^7} (-1)^{x \cdot y} |y\rangle .$$

If $y \notin G$, then the sum is 0 for the same reason. If $y \in H$, then the sum is $8 |y\rangle$, and if $y \in G \setminus H$, the sum is $-8 |y\rangle$. This is just because the inner product of two vectors in $G \setminus H$ is 1. Thus, we get

$$H^{\otimes 7} |1\rangle_L = \frac{1}{\sqrt{2}} \big(|0\rangle_L - |1\rangle_L \big)$$

So both $H^{\otimes 7} |0\rangle_L$ and $H^{\otimes 7} |1\rangle_L$ are states in our quantum Hamming code, and thus bit-flip errors in $H^{\otimes 7} |0\rangle_L$ and $H^{\otimes 7} |1\rangle_L$ can be corrected by the error correction procedure. This means that phase-flip errors of $|0\rangle_L$ and $|1\rangle_L$ can be corrected by applying $H^{\otimes 7}$, applying the bit-flip error correction procedure, and applying $H^{\otimes 7}$ again, and we have shown that the quantum Hamming code is an error-correcting code that can correct any single-bit error.

There is one last thing I want to say about the quantum Hamming code. How do we apply a logical σ_x? You can turn any codeword in H into a codeword in $G \setminus H$ and vice versa by adding the vector 1111111 (actually, adding any codeword in $G \setminus H$ will do). So the operation corresponding to a logical $|0\rangle$ is just $\sigma_x^{\otimes 7}$.

How do you apply a logical σ_z? It turns out that you can apply a $\sigma_z^{\otimes 7}$. One way to see this is using the fact that a Hadamard get applied to each qubit applies a Hadamard gate to the logical qubit, and similarly a σ_x gate applied to each qubit also applies a σ_x gate to the logical qubit. But you can also see it by realizing that $\sigma_z^{\otimes 7}$ applies a $+1$ phase to any $|y\rangle$ for $y \in H$ and a -1 phase to any $|y\rangle$ for $y \in G \setminus H$.

Notes 8.370/18.435 Fall 2022

Lecture 30 Prof. Peter Shor

Last time we talked about the quantum Hamming code. This was based on the classical Hamming code. We have two matrices, G and H, with

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The quantum error correcting code they generated mapped $|0\rangle$ to

$$|0\rangle_L = \frac{1}{\sqrt{8}} \sum_{x \in H} |x\rangle \,,$$

and mapped $|1\rangle$ to

$$|1\rangle_L = \frac{1}{\sqrt{8}} \sum_{x \in H} |x \oplus 1111111\rangle \ .$$

Recall that a classical binary linear code is a subspace of $\mathbb{Z}_2^n$. If the minimum non-zero codeword has Hamming weight d, then the code is said to have distance d. Recall that a code with distance d can correct up to $t = \frac{d-1}{2}$ errors. A code that maps k bits into n bits and has distance d is called an $[n, k, d]$ code. For example, the Hamming code with generator matrix G is a $[7, 4, 3]$ code, and the code with generator matrix H is a $[7, 3, 4]$ code.

A CSS code is a quantum error-correcting code that is derived from two classical codes, C_1 and C_2, with $C_2 \subseteq C_1$. We will call this code CSS($C_1 : C_2$). If C_1 can correct t_1 errors and $C_2^\perp$, the dual code to C_2, can correct t_2 errors, then CSS($C_1 : C_2$) can correct t_1 bit-flip errors and t_2 phase-flip errors.

Let me remark that one can consider a quantum error-correcting code to either be a collection of codewords or the subspace generated by these words. Since quantum error-correcting codes must correct superpositions of codewords as well as the codewords themselves, they must correct every quantum state in the subspace. For CSS codes, it doesn't really matter which of these perspectives you use, but for some other quantum error-correcting codes, the subspace view is a more intuitive way to think about it.

Before we give the definition of a quantum CSS code, we need to define a few terms in classical error correcting codes. If $C_2 \subseteq C_1$, a *coset* of C_2 in C_1 is the set of vectors

$$x + C_2 = \{y | y = x + c,\ c \in C_2\}$$

for some $x \in C_1$. Two cosets $x_1 + C_2$ and $x_2 + C_2$ either are equal or are disjoint, and every element $x \in C_1$ is a member of some coset, so the number of cosets is $\frac{|C_1|}{|C_2|}$, i.e., the number of elements of C_1 divided by the number of elements of C_2.

Why are two cosets either equal or disjoint? Suppose that $y \in x + C_2$ Then $y = x + d$ for some $d \in C_2$. Now, if we add d to any other element in C_2, we still have an element in C_2, so

$$y{+}C_2 = \{z|z = y{+}c,\ c \in C_2\} = \{z|z = x{+}d{+}c,\ c \in C_2\} = \{z|z = x{+}c,\ c \in C_2\} = x{+}C_2.$$

For example, in the Hamming code, there are two cosets of H in G,

$$H \quad \text{and} \quad H + 1111111\,.$$

Now, we can give the codewords of the CSS code associated with two n-bit classical codes C_1 and C_2 with $C_2 \subset C_1$. The codewords of this code will be the associated with the cosets of C_2 in C_1. For each coset $x + C_2$, we have the codeword

$$|x + C_2\rangle = \frac{1}{|C_2|^{1/2}} \sum_{c \in C_2} |x + c\rangle\,.$$

The number of codewords is the number of cosets of C_2 in C_1, which is $\frac{|C_1|}{|C_2|}$. The dimension of the code subspace is

$$\dim \mathrm{CSS}(C_1 : C_2) = \log_2 \frac{|C_1|}{|C_2|} = \dim C_1 - \dim C_2$$

Why does CSS(C_1, C_2) correct t_1 bit-flip errors? We know that there is a classical error-correction procedure that will correct t_1 or fewer errors in any codeword c_1 of C_1. Now, the codewords of CSS(C_1, C_2) are composed of superposition of quantum states $|x + c\rangle$, each of which is a codeword of C_1. Thus, applying this classical error-correction procedure in quantum superposition will correct up to t_1 bit-flip errors in the quantum code.

We now need to explain why the dual code will correct t_2 phase errors. Recall that the dual of a subspace W of a vector space V was

$$W^\perp = \{x \in V | x \cdot w = 0\ \forall w \in W\}$$

If you're used to working with real and complex vector spaces, a non-intuitive fact about binary vector spaces is that the original space and its dual can overlap. For example, with the Hamming code above, $H = G^\perp \subset G$. Howerver, it is still true that if $W^\perp$ is the dual of W in vector space V, then

$$\dim V = \dim W + \dim W^\perp\,.$$

We won't prove it in these notes, but there is a proof of this along the lines of the Gaussian elimination manipulations we did in our discussion of Simon's algorithm.

Let's apply the Hadamard transform to the codewords $|x + C_2\rangle$. It turns out that what we get are superpositions of the codewords of the code CSS($C_2^\perp : C_1^\perp$). Before we prove this, let me make a little side comment about why this makes sense. because $C_2 \subseteq C_1$, any vector that is perpendicular to everything in C_1 is also perpendicular

to everything in C_2, so $C_1^\perp \subseteq C_2^\perp$. Further, $\dim C_1^\perp = n - \dim C_1$ and $\dim C_2^\perp = n - \dim C_2$, so

$$\dim \mathrm{CSS}(C_1 : C_2) = \dim \mathrm{CSS}(C_2^\perp : C_1^\perp)\,.$$

Now, let's show that the Hadamard transform of a codeword $|x + C_2\rangle$ of CSS($C_1 : C_2$) is a superposition of codewords of CSS($C_2^\perp : C_1^\perp$). Taking the Hadamard transform, we find:

$$\begin{aligned} H^{\otimes n} |x + C_2\rangle &= \frac{1}{\sqrt{2^{\dim C_2}}} H^{\otimes n} \sum_{c_2 \in C_2} |c_2 + x\rangle \\ &= \frac{1}{\sqrt{2^{\dim C_2}}} \frac{1}{\sqrt{2^n}} \sum_{y \in Z_2^n} \sum_{c_2 \in C_2} (-1)^{(c_2+x)\cdot y} |y\rangle \\ &= \frac{1}{\sqrt{2^{\dim C_2}}} \frac{1}{\sqrt{2^n}} \sum_{y \in Z_2^n} (-1)^{x\cdot y} \sum_{c_2 \in C_2} (-1)^{c_2 \cdot y} |y\rangle \end{aligned}$$

Now I claim that

$$\sum_{c_2 \in C_2} (-1)^{c_2 \cdot y} = \begin{cases} |C_2| & y \in C_2^\perp \\ 0 & y \notin C_2^\perp \end{cases}.$$

Why? if $y \in C_2^\perp$, all the elements of the sum are 1, so we get $|C_2$. If not, then there is some $d \in C_2$ such that $(-1)^{d\cdot yi} = -1$, and we can pair the elements of C_2 into pairs $(c_2, c_2 + d)$. Since $(-1(^{c_2} + (-1)^{c_2+d} = 0$, the sum for each of these pairs is 0, so the whole sum is 0. Thus, we have

$$\begin{aligned} H^{\otimes n} |x + C_2\rangle &= \frac{1}{\sqrt{2^{\dim C_2}}} \frac{1}{\sqrt{2^n}} \sum_{y \in C_2^\perp} (-1)^{x\cdot y} |C_2|\, |y\rangle \\ &= \frac{\sqrt{2^{\dim C_2}}}{\sqrt{2^n}} \sum_{y \in C_2^\perp} (-1)^{x\cdot y} |y\rangle \\ &= \frac{1}{\sqrt{2^{\dim C_2^\perp}}} \sum_{y \in C_2^\perp} (-1)^{x\cdot y} |y\rangle \end{aligned}$$

Now, we are nearly done. We claim that if y_1 and y_2 are in the same coset of $C_1^\perp$ in $C_2^\perp$, then $(-1)^{x\cdot y_1} = (-1)^{x\cdot y_2}$. Why is this? Because $y_1 - y_2 \in C_1^\perp$ and $x \in C_1$, so $x \cdot (y_1 - y_2) = 0$. So now let R be a set of representatives of the cosets of $C_1^\perp$ in $C_2^\perp$; this means there is one element of each coset in R. We can group the sum over y above into elements from each coset of $C_1^\perp$ in $C_2^\perp$. This gives

$$\begin{aligned} H^{\otimes n} |x + C_2\rangle &= \frac{1}{\sqrt{2^{\dim C_2^\perp}}} \sum_{y \in R} (-1)^{x\cdot y} \sum_{z \in y + C_1^\perp} |z\rangle \\ &= \frac{\sqrt{2^{\dim C_1^\perp}}}{\sqrt{2^{\dim C_2^\perp}}} \sum_{y \in R} (-1)^{x\cdot y} \left|y + C_1^\perp\right\rangle, \end{aligned}$$

showing that the Hadamard transform of a codeword in CSS($C_1 : C_2$) is a superposition of codewords in CSS($C_2^\perp : C_1^\perp$), as we wanted.

The last thing I want to talk about in these notes is shifting a code. We will need this definition for the proof of the security of BB84, which we present in the next lecture. We can take a CSS code and shift it in bit space or in phase space. A codeword for a code shifted by s in bit space is

$$|x + C_2\rangle = \frac{1}{|C_2|^{1/2}} \sum_{c \in C_2} |s + x + c\rangle$$

and one shifted by t in phase space is

$$|x + C_2\rangle = \frac{1}{|C_2|^{1/2}} \sum_{c \in C_2} (-1)^{t \cdot (x+c)} |x + c\rangle$$

And of course, you can shift in phase space and bit space (the order makes a difference) to get

$$|x + C_2\rangle = \frac{1}{|C_2|^{1/2}} \sum_{c \in C_2} (-1)^{t \cdot (x+c)} |s + x + c\rangle$$

All of these shifted error correcting codes can correct the same number of errors as the unshifted ones. What we will need to know for the proof of security of quantum key distribution is that if you shift in both bit space and phase space by a random vector, and average over the resulting codes, you get the completely uniform distribution — the density matrix is I_{2^n}. I'm not going to prove that in these lecture notes, but if you want to prove it, it would make a good exercise to make sure you understand CSS codes well.

Notes 8.370/18.435 Fall 2021

Lecture 31 Prof. Peter Shor

Today we will explain the BB84 key distribution protocol, and give the proof of its security based on quantum error-correcting codes.

First, I'm going to say a few things about key distribution protocols and cryptography in general. Then, I'll explain the BB84 key distribution protocol. Next, I'll explain a key distribution protocol based on quantum error correcting codes, which has a fairly straightforward proof that it is secure. Finally, I'll explain why the two protocols are essentially equivalent, so if the QECC-based protocol is secure, then BB84 is.

BB84 is a key distribution protocol; it's named after its inventors, Charlie Bennett and Gilles Brassard, and the year it was invented. It was based on some ideas that Stephen Wiesner had in 1969. Wiesner wrote the paper up and sent it to a journal, at which point it was rejected, and he gave up. The paper was eventually published in 1983 when Charlie Bennett sent it to a theoretical computer science newsletter whose editor he knew.

What is a key distribution protocol? We've seen one before in this class—the Diffie-Hellman key distribution protocol, which is based on the hardness of the discrete log problem. The basic idea is that two participants, Alice and Bob, want to agree on a secret key that an eavesdropper, Eve, does not know. We assume that Alice and Bob start the protocol without any secret information in common, so they have to agree on a secret key using a public channel. Classically, the only way to do this is to base the protocol on a hard problem, which we assume that the eavesdropper cannot solve. Quantum mechanically, however, we don't need to make any hardness assumptions—BB84 is secure as long as the laws of quantum mechanics hold.

We will be making some assumptions about the communication channels we use in the protocol, so first we're going to give the reasons for these assumptions. One thing that you have to guard against is the man-in-the-middle attack. Suppose that Eve cuts the channels, and inserts herself in the middle

$$\text{Alice} \longleftrightarrow \text{Eve} \longleftrightarrow \text{Bob}$$

Now, when Eve talks to Alice she pretends to be Bob, and when she talks to Bob, she pretends to be Alice. Since Alice and Bob don't know any secret identifying information about each other, they cannot uncover her impersonation, and she learns everything they say to each other.

Our assumptions will be that Alice and Bob have a classical channel that is not spoofable—that is, Eve cannot take over the channel and pretend to be Bob, and a quantum channel on which Eve is allowed to do anything that is consistent with the laws of physics. Since Eve has complete control of the quantum channel, she can cut it and prevent Alice and Bob from communicating at all. So what we will require from the protocol is that it is very unlikely that Alice and Bob think they've successfully shared a secret key while Eve has more than an exponentially small amount of information about that secret key.

The advantage that quantum key distribution has over classical key distribution protocols is that classical key distribution protocols must rely on some hard problem, and

we have no guarantee that a very clever mathematician won't come up with a way to solve (say) the discrete log problem in polynomial time tomorrow. The disadvantage of quantum key distribution is that you need a quantum channel between the two participants. This is possible over an optical fiber. There are lots of these already in the ground, but there are extra requirements for them to be used for key distribution—you can't have an amplifier on the optical fiber between the two people who want to use the BB84 protocol, because amplifiers destroy quantum coherence. With current technology, this limits the distance that quantum key distribution can be used on optical fiber to a few hundred kilometers (and good luck finding an optical fiber that long that's already in place which doesn't have amplifiers on it; if you want to use quantum key distribution over existing optical fibers, your distance limitations will be substantially shorter).

1 BB84

So what is the BB84 protocol? We will give the original BB84 protocol; there have been many different quantum key distribution protocols proposed since them, and some of them have substantial practical advantages over BB84. We first explain how it works when the state preparations and the measurements are perfect, and the channel between Alice and Bob is noiseless.

1. Alice sends Bob a sequence of qubits that she has prepared in one of the four states $|0\rangle, |1\rangle, |+\rangle, |-\rangle$:

Alice prepares	$\|0\rangle$	$\|1\rangle$	$\|+\rangle$	$\|+\rangle$	$\|0\rangle$	$\|-\rangle$	$\|-\rangle$	$\|1\rangle$	$\|1\rangle$	$\|0\rangle$	$\|0\rangle$

2. Bob measures the qubits he gets in a random basis:

Bob measures	$0/1$	$0/1$	$0/1$	$+/-$	$+/-$	$0/1$	$+/-$	$0/1$	$+/-$	$0/1$	$0/1$
and gets	$\|0\rangle$	$\|1\rangle$	$\|1\rangle$	$\|+\rangle$	$\|+\rangle$	$\|1\rangle$	$\|-\rangle$	$\|1\rangle$	$\|+\rangle$	$\|0\rangle$	$\|0\rangle$

3. At this point, Alice announces her basis, and Bob tells Alice which ones agree. They discard the measurement results for bases that disagree.

Alice′s basis	$0/1$	$0/1$	$+/-$	$+/-$	$0/1$	$+/-$	$+/-$	$0/1$	$0/1$	$0/1$	$0/1$
places they agree	$\|0\rangle$	$\|1\rangle$		$\|+\rangle$			$\|-\rangle$	$\|1\rangle$		$\|0\rangle$	$\|0\rangle$

4. Now, Alice (or Bob) announces a random sample of the qubits to use to check whether they agree. If they do, they know that Eve couldn't have been measuring many of the qubits. They turn the remaining qubits into a secret key, using (say) the mapping of $|0\rangle$ and $|+\rangle$ to 0 and $|1\rangle$ and $|-\rangle$ to 1:

check qubits		?		?					?	
Alice		1		+					0	
Bob		1		+					0	
secret key	0					1	1			0

The reason that this protocol works is that if Eve tries to measure a qubit, she doesn't know whether to measure in the $0/1$ basis or the $+/-$ basis. If she chooses the wrong basis, then she will disturb the quantum state that Alice sent, and Alice and Bob will notice that some of their check bits disagree.

But what if their channel is noisy? Some of Alice's and Bob's string of bits will disagree anyway, so how can they tell whether Eve is eavesdropping? How can they get a string of bits that they agree on after that? And even if they do, how can they ensure that Eve doesn't have any information about this secret key.

The first problem is solved by using error correcting codes. Suppose Alice and Bob have strings a and b of length m. Because they tested their check bits, they know that they expect around ϵm of their bits to differ, where ϵ is relatively small. Now, Alice chooses an error correcting code C of length m that will correct $\epsilon' m$ bits, where $\epsilon' > \epsilon$, so that even accounting for random fluctuations in the noise, the number of places where Alice and Bob's bits differ is less than $\epsilon' m$ with high probability. Alice then chooses a random codeword $c \in C$, and sends

$$a + c$$

to Bob. Bob takes this message and subtracts b from it to get $a - b + c$. This is a string that differs from the codeword c in fewer than $\epsilon' m$ positions, so Bob can apply error correction and get c. Alice and Bob then share c, and Eve does not know what c is. Why not? Because a was essentially random, $a + c$ is also random. Since this is the only information Eve sees about a and c, she should not have any information on what c is. (You should note that this is not a rigorous proof; it took a decade and a half after BB84 was proposed to get a rigorous proof that it was secure.)

Finally, it's possible that after this protocol, Eve has some information about c. To fix this, Alice and Bob choose a hash function f that maps m bits into ℓ bits where $\ell < m$, If this hash function is sufficiently random, and ℓ is sufficiently shorter than m, then a theorem from classical cryptography says that the information that Eve has about $f(c)$ is much less than the information Eve has about c. In this protocol, to make the proof work, we will assume that f is a linear function, so $f(c) = Mc$ for some binary matrix M.

2 The adapted Lo-Chao protocol

We now give a protocol for which the security proof is relatively simple. The difference between this and the original BB84 protocol is that for this one, Bob needs quantum memory to store all the qubits he receives in, and thus it is not currently practical, and it's not going to be as cheap to implement as BB84 in the foreseeable future. However, we will then show that these protocols are equivalent in that if somebody can break BB84, they can also break this protocol. The protocol is based on one published by Lo and Chau.

The idea behind this protocol is that if Alice and Bob share perfect EPR pairs, then measuring them in the $0/1$ basis will give them a secret key. Eve can never determine the outcomes of their measurements; since Alice and Bob have perfect entanglement, then Eve cannot be entangled with their state, and Eve's state will remain uncorrelated

with Alice and Bob's secret key. There is a theorem called "monogamy of entanglement" which says that the more you are entangled with one system, the less entanglement you can have with any other system. This theorem gives intuition for why this QKD protocol works.

So what is the Lo-Chau protocol?

1. Alice prepares n EPR pairs.
2. Alice chooses a CSS code CSS($C_1 : C_2$) and a translate of it by s in bit space and t in phase space.
3. Alice encodes half of each EPR pair with this code, randomly intersperses test bits which are equally likely to be in one of the four bases $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, and sends this string of qubits to Bob.
4. Bob puts everything into his quantum memory.
5. Alice announces the code and the strings s and t it was translated by, which bits were test bits, which bits were code bits, and the values of the test bits.
6. Bob checks the test bits to determine the error rate. He then decodes the EPR pairs, and Alice and Bob measure each EPR pair in the $0/1$ basis to obtains a secret key.

The first thing to note is that because Alice sends a random translate of CSS(C_1, C_2), and because the test qubits are equally likely to be in any of the four states, the density matrix that Eve sees is completely random; i.e., is the identity matrix. Thus, Eve cannot tell which of the qubits are code qubits and which are test qubits, so the noise rate she induces on the test qubits will also with high probability be induced on the code qubits.

Now, because the rate of noise on the test bits is sufficiently low, the probability that the CSS code does not transmit the correct state is ϵ, where ϵ can be made exponentially small. The state that Alice and Bob share after the transmission is then

$$\sqrt{1-\epsilon}\,|\,\phi\rangle^{\otimes n} + \sqrt{\epsilon}\,|\,E\rangle$$

where $|\,\phi\rangle$ is an EPR pair and $|\,E\rangle$ is an arbitrary error state.

So how much information can Eve find about the EPR pairs in this scenario? Suppose that Alice and Bob got together and measured their state. Then with probability $1-\epsilon$, they would have $|\,\phi\rangle^{\otimes n}$ and with probability ϵ, they would have something else. Eve can have information about their secret key only with probability ϵ. If ϵ is exponentially small, Eve has an exponentially small amount of information about the key (To state this formally and prove it would require explaining more information theory than I want to do right now.) Thus, the Lo-Chau protocol works.

3 The equivalence of the protocols

Now, let's show that these protocols are equivalent. In the second protocol, we can assume that Alice measures her half of the EPR pairs before she encodes the quantum

state, because the operation of measuring commutes with the operation of encoding and sending the other half of the EPR pairs. Thus, we can assume that the quantum state that she sends is a random string of n classical bits which is encoded in the CSS code $\mathrm{CSS}(C_1 : C_2)$.

So what happens when Alice encodes a random string of bits to encode. What she is essentially doing is choosing a random coset $x + C_2$ and encoding it. But choosing a random coset x is exactly the same as choosing a random bit string y and taking the coset $y + C_2$. When it's encoded by the shifted CSS code, it will look like

$$\frac{1}{|C_2|^{1/2}} \sum_{c_2 \in C_2} (-1)^{t\cdot(y+c_2)} \,|\, s + y + c_2 \rangle$$

For the secret key, Bob needs to find the coset of C_2 that this belongs to. He can find the coset by measuring this string in the $|\, 0\rangle, |\, 1\rangle$ basis, and subtracting s to get $y + c_2$. Now, note that Bob doesn't actually need t to find this coset, so we can assume that Alice never sends him t. If Alice doesn't send him t, the density matrix of her message when you take the average over t is

$$\frac{1}{|C_2|} \left(\sum_{c_2 \in C_2} (-1)^{t\cdot(y+c_2)} \,|\, s + y + c_2 \rangle \right) \left(\sum_{c_2 \in C_2} (-1)^{t\cdot(y+c_2)} \,\langle s + y + c_2 \,|\, \right) = \frac{1}{C_2} \sum_{c_2 \in C_2} |\, s + y + c_2 \rangle\langle s + y + c_2 \,|\,,$$

which is the same as Alice taking a random c_2 and adding it to $s + y$.

Because Bob and Alice are communicating over a noisy channel, Bob actually gets $s+y+c_2+e$, where e is some error. He then needs to subtract s and apply the classical decoding procedure to get $y + c_2$.

So let's compare the protocols. In both cases, test qubits are interspersed with the code qubits. In BB84, Alice sends Bob a on the quantum channel. Bob receives $b = a + e$. Alice then sends Bob $a + c_1$ on the classical channel. Bob subtracts these two quantities to get $c_1 + e$ and decodes it to the codeword $c_1 \in c_1$.

In the second protocol, Alice sends Bob $s + y + c_2$, where $y + c_2 \in C_1$. Bob receives $s+y+c_2+e$. Alice then sends Bob s. Bob now subtracts these two quantities to obtain $y + c_2 + e$ and decodes it to get a codeword of C_1.

These two protocols look a lot alike. In both cases, Alice sends Bob a random string on the quantum channel. Then she sends to him over the classical channel a string that differs from the first sting by a random codeword of the code C_1. The secret message (before privacy amplification) is the codeword of C_1. Thus, by equating $a = s+y+c_2$ and $a + c_1 = s$, we can show that the two protocols are completely equivalent.

Now, let's deal with the privacy amplification. In the BB84 protocol, Alice and Bob get the codeword $c \in C_1$ that they will use to derive their the secret key. They then apply a linear hash function to the codeword to amplify the privacy. If you think about it, the only thing that matters about the linear hash functions is which codewords in c_1 get mapped to the same string $f(c_1)$. Because f is a linear hash function, i.e., $f(x) = Mx$ for some matrix M, the codewords that get mapped to 0 (call these codewords

S) are a linear subspace of C_1, and the codewords that get mapped to any other value are a coset of S in C_1. Thus, we can take S to be C_2 — a linear code is just a linear subspace, and random codes (like those generated by random matrices M) are highly likely to be good error correcting codes.

We have thus shown that BB84 is equivalent to our other key distribution protocol based on Lo and Chau's ideas, so BB84 is secure.