Notes on Quantum Computing & Information

Introduction to Quantum Mechanics I Spring 2024

Jim Napolitano
March 24, 2024

Introduction

A "computer" stores and manipulates information as "bits," which are two-dimensional sets of zero and one, that is $\{0, 1\}$. The computer can be represented by a function $f(x)$ that takes a set x of n bits and maps it onto a set y of m bits. That is

$$f(x) : x = \{0, 1\}^n \mapsto y = \{0, 1\}^m$$

Every time we execute the function $f(x)$, we have completed one operation. Generally we refer to executing one operation as a "step."

A classical digital computer (i.e. an ordinary computer in today's language) stores the ones and zeros in transistorized circuits. Information is stored in arrays of eight-bit units called "bytes." Operations are performed on the bytes using electrical signals driven at some clock speed, typically upwards of 1 GHz (a billion ticks per second) on a modern laptop computer. Programs are written in some high level language which is translated into machine commands that manipulate bytes and produce the output.

Once upon a time, "computers" were not digital devices. Instead, they were analog devices that did things like take derivatives and integrals of analog signals, and you could wire up such an "analog computer" to solve a particular problem. They were very popular, but as digital computers became more powerful, analog computers all but disappeared.

A "quantum computer" uses quantum mechanics to store and manipulate information. It is closer in concept to an analog computer than to a digital computer, and one needs to "wire up", so to speak, the components of a quantum computer to solve a specific problem. The key to quantum computing is that the answer to some problems can be reached in far fewer steps in a quantum computer than in a classical digital computer. Sometimes the factor is exponential, for example in Shor's Algorithm.

These notes represent my own attempt to understand some of the details of quantum information and quantum computing. I haven't annotated things very thoroughly and there are likely various mistakes throughout. I hope they are helpful nevertheless.

Qubits

A "quantum bit", or "quibit", is a any quantum mechanical system with two states, typically (or always?) the eigenstates of some Hamiltonian. In state space, we can write the two qubit states as $|0\rangle$ or $|1\rangle$, in analogy with the binary digit, or "bit", in a classical digital computer. Many other notations are possible, though. A common form is $|\uparrow\rangle$ and $|\downarrow\rangle$, because a convenient physical way to think about a qubit is as the spin states of an electron, proton, or other spin-1/2 system. In this case, we can manipulate qubits using magnetic fields to rotate the spin state.

A different physical manifestation for a qubit is the "transmon qubit", an electrical circuit that is a superconducting nonlinear oscillator, where $|0\rangle$ or $|1\rangle$ are the ground and first excited state of the oscillator. The state of this qubit is manipulated using radio frequency signals. Here is a reference:

https://arxiv.org/pdf/2106.11352.pdf

IBM has a program of building quantum computers based on the transmon qubit.

In order to build a quantum computer, we assemble a collection of n qubits into a state

$$|x\rangle = |1011\cdots01\rangle \qquad \text{etc}\ldots$$

where x just represents any one of the $N = 2^n$ possible combinations of the n qubits. In order to build a functional quantum computer, it is critical that all of the qubits form a single, coherent quantum mechanical system. In more colloquial language, the wave functions of all of the qubits need to overlap with each other.

Now you can see the potential of a quantum computer relative to a classical digital computer. The state of a quantum computer at any one time can be written as

$$|\alpha\rangle = \sum_{x=1}^{N} c_x |x\rangle$$

In other words, we can specify this quantum mechanical state using a classical digital computer by specifying the $N = 2^n$ complex values of c_x. For $n = 100$, a value achieved by IBM's latest quantum computer, this means you would need to specify

$$2^{100} = \left(2^{10}\right)^{10} = (1\text{K})^{10} \approx 10^{30} = 1 \text{ Million} \times 1 \text{ Trillion} \times 1 \text{ Trillion}$$

complex numbers. On my desk at home, I have a 1 Tb backup disk, enough storage for about 0.1 Trillion complex numbers. In other words, I would need ten million trillion of such disks just to store the information at any one time in a 100 qubit quantum computer.

Representations of Qubits

We represent qubits as column vectors the same way that we make representations of any other quantum mechanical state, now using $|0\rangle$ and $|1\rangle$ for the basis. That is, we represent a state vector $|\alpha\rangle$ as

$$|\alpha\rangle \doteq \begin{bmatrix} \langle 0|\alpha\rangle \\ \langle 1|\alpha\rangle \end{bmatrix} \qquad \text{e.g.} \qquad |0\rangle \doteq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{and} \qquad |1\rangle \doteq \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

where we remember that $|0\rangle$ and $|1\rangle$ are orthogonal because they are eigenstates of some Hermitian operator, and we assume that all state kets are normalized.

Multiple qubits are represented by tensor products of column vectors. To see how this works, let's write out the state $|\alpha\rangle \otimes |\beta\rangle$ for a two-qubit system. (I will resist writing this state as $|\alpha\beta\rangle$ for clarity in this discussion.) The four complex numbers we need in order to represent this state are

$$\begin{aligned}
(\langle 0| \otimes \langle 0|)\,(|\alpha\rangle \otimes |\beta\rangle) &= \langle 0|\alpha\rangle\langle 0|\beta\rangle && \text{(1a)}\\
(\langle 0| \otimes \langle 1|)\,(|\alpha\rangle \otimes |\beta\rangle) &= \langle 0|\alpha\rangle\langle 1|\beta\rangle && \text{(1b)}\\
(\langle 1| \otimes \langle 0|)\,(|\alpha\rangle \otimes |\beta\rangle) &= \langle 1|\alpha\rangle\langle 0|\beta\rangle && \text{(1c)}\\
(\langle 1| \otimes \langle 1|)\,(|\alpha\rangle \otimes |\beta\rangle) &= \langle 1|\alpha\rangle\langle 1|\beta\rangle && \text{(1d)}
\end{aligned}$$

Just to be concise, write $a = \langle 0|\alpha\rangle$, $b = \langle 1|\alpha\rangle$, $c = \langle 0|\beta\rangle$, and $d = \langle 0|\beta\rangle$. Then Equations (1) become

$$|\alpha\rangle \otimes |\beta\rangle \doteq \begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} \\ b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}$$

so that, assuming the two-dimensional vectors are normalized,

$$\underline{\underline{\alpha}}^\dagger\,\underline{\underline{\alpha}} = |ac|^2 + |ad|^2 + |bc|^2 + |bd|^2 = |a|^2(|c|^2 + |d|^2) + |b|^2(|c|^2 + |d|^2) = |a|^2 + |b|^2 = 1$$

the four-dimensional tensor product is also normalized. This means the two-qubit states are

$$|00\rangle \doteq \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad |01\rangle \doteq \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad |10\rangle \doteq \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad |11\rangle \doteq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

It is worth pointing out that MATHEMATICA provides the function TensorProduct for calculating tensor products of vectors (and matrices).

Some History

The huge amount of information that you can store in quantum computer as compared to a classical computer has been known since the birth of quantum mechanics, of course, but serious work in quantum computing was started in the 1960's. Important early contributors include Paul Benioff, Richard Feynman, and David Deutsch, among many others.

Research in quantum computing was a primarily academic exercise until 1994 when Peter Shor came up with Shor's Algorithm for breaking a number into its prime factors. This was a huge advance because at that time, computer communications were encrypted using something known as the RSA algorithm, which relied on it being exponentially more difficult to factorize a large number as the number of digits increased. Shor's algorithm represented a literally exponential speed up of this problem solution on a quantum computer. Encryption is critical to many walks of life, including national security and banking, and many resources were consequently brought to bear on this research. This has lead to a number of breakthroughs in the past three decades, including a recent solution to the Ising model in two dimensions on a quantum computer by researchers at IBM:

https://www.nature.com/articles/s41586-023-06096-3

Quantum Gates

We manipulate qubits mathematically using "quantum gates" which, in essence, perform rotations and phase changes on qubits. Recall that the rotation operator in a two-dimensional Hilbert space is represented in the $|\pm\hat{\mathbf{z}}\rangle$ basis as

$$\underline{\underline{D}} = \underline{\underline{1}}\cos\frac{\phi}{2} - i\underline{\underline{\vec{\sigma}}}\cdot\hat{\mathbf{n}}\sin\frac{\phi}{2}$$

and we can create any gate we want by the right choices of ϕ and $\hat{\mathbf{n}}$, or combinations of different rotations.

Sometimes a gate will introduce a phase, but if the gate is applied to all qubits, it will be irrelevant in the end. For example, a simple NOT gate is effected using σ_x, that is

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

and $\underline{\underline{D}}_x(\pi) = -i\underline{\underline{\sigma}}_x$, but we can ignore the $-i$ if we want to explicitly use a rotation to describe gate.

Of course, we can build a classical computer NOT gate out of transistors. On the other hand, there is a gate that can only be achieved quantum mechanically. I call this the "square-root-of-a-NOT" gate because if you apply the identical gate twice, you get a NOT gate. It is pretty obvious that this can be achieved by a 90° rotation about the x- or y-axis. Choosing the former gives

$$\sqrt{\text{NOT}} = \underline{\underline{D}}_x\left(\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\left(\underline{\underline{1}} - i\underline{\underline{\sigma}}_x\right) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix}$$

in which case

$$\left(\sqrt{\text{NOT}}\right)^2 = \frac{1}{2}\left(\underline{\underline{1}} - i\underline{\underline{\sigma}}_x\right)^2 = \frac{1}{2}\left(\underline{\underline{1}} - 2i\underline{\underline{\sigma}}_x - \underline{\underline{1}}\right) = -i\underline{\underline{\sigma}}_x = \underline{\underline{D}}_x(\pi)$$

A particularly useful quantum computing gate is the Hadamard gate, represented as

$$\underline{\underline{H}} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

which takes the qubit $|0\rangle$ into $|+\rangle \equiv (|0\rangle + |1\rangle)/\sqrt{2}$ and $|1\rangle$ into $|-\rangle \equiv (|0\rangle - |1\rangle)/\sqrt{2}$, that is

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \text{and} \qquad \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

We can realize a Hadamard gate physically by a $\pi/2$ rotation about the y-axis followed by a π rotation about the x-axis, that is

$$\underline{\underline{D}}_x(\pi)\,\underline{\underline{D}}_y\left(\frac{\pi}{2}\right) = \left[-i\underline{\underline{\sigma}}_x\right]\left[\frac{1}{\sqrt{2}}\underline{\underline{1}} - \frac{i}{\sqrt{2}}\underline{\underline{\sigma}}_y\right] = -\frac{i}{\sqrt{2}}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = (-i)\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

where we don't care about the overall phase factor $-i = \exp(-i\pi/2)$.

Two-Qubit Quantum Gates

We can build gates that act on two coherent qubits using 4×4 matrices. This includes using the tensor product of two single-qubit gates to make two-qubit gate, using the same approach that we used to build two-qubit column vectors. That is, we build a "super matrix" by putting the second 2×2 matrix in front of each element of the first matrix, namely

$$\underline{\underline{A}} \otimes \underline{\underline{B}} = \begin{bmatrix} A_{11}\underline{\underline{B}} & A_{12}\underline{\underline{B}} \\ A_{21}\underline{\underline{B}} & A_{22}\underline{\underline{B}} \end{bmatrix}$$

This is not hard to prove. We just go back to basics and build on what we learned about two-qubit representations. That is, using the same notation we used above for two-qubits,

$$\begin{aligned}(A \otimes B)(|\alpha\rangle \otimes \beta\rangle) &= (A|\alpha\rangle) \otimes (B|\beta\rangle) \doteq \underline{\underline{A}}\begin{bmatrix} a \\ b \end{bmatrix} \otimes \underline{\underline{B}}\begin{bmatrix} c \\ d \end{bmatrix} \\ &= \begin{bmatrix} A_{11}\underline{\underline{B}}\begin{bmatrix} ac \\ ad \end{bmatrix} + A_{12}\underline{\underline{B}}\begin{bmatrix} bc \\ bd \end{bmatrix} \\ A_{21}\underline{\underline{B}}\begin{bmatrix} ac \\ ad \end{bmatrix} + A_{22}\underline{\underline{B}}\begin{bmatrix} bc \\ bd \end{bmatrix} \end{bmatrix} = \begin{bmatrix} A_{11}\underline{\underline{B}} & A_{12}\underline{\underline{B}} \\ A_{21}\underline{\underline{B}} & A_{22}\underline{\underline{B}} \end{bmatrix}\begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}\end{aligned}$$

Now let's look at some examples. One particularly useful example of a two-qubit NOT gate is the Controlled NOT, or CNOT, gate. Call the first qubit the *control* qubit and the second the *target* qubit. Then the action of the CNOT gate is to flip the target qubit if the control bit is $|1\rangle$, and do nothing if the control bit is $|0\rangle$. I claim that the CNOT is given by

$$\underline{\underline{CNOT}} = \begin{bmatrix} \underline{\underline{1}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{\sigma}}_x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Let's check that this works. For the control bit equal to $|1\rangle$, we have

$$CNOT|10\rangle \doteq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \doteq |11\rangle \tag{2}$$

$$\text{and} \qquad CNOT|11\rangle \doteq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \doteq |10\rangle \tag{3}$$

and indeed, the target qubit is flipped in each case (and the control bit is left unchanged). For the control bit equal to $|0\rangle$, we have

$$CNOT|00\rangle \doteq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \doteq |00\rangle \tag{4}$$

$$\text{and} \qquad CNOT|01\rangle \doteq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \doteq |01\rangle \tag{5}$$

and this time the target bit is not flipped.

We can also use the tensor product of two single-qubit gates to form two-qubit gates. For example

$$\underline{\underline{\sigma}}_x^{\otimes 2} \equiv \underline{\underline{\sigma}}_x \otimes \underline{\underline{\sigma}}_x = \begin{bmatrix} 0\underline{\underline{\sigma}}_x & 1\underline{\underline{\sigma}}_x \\ 1\underline{\underline{\sigma}}_x & 0\underline{\underline{\sigma}}_x \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

I will leave it as an exercise to show that this gate does what it is supposed to do.

Grover's Search Algorithm

If you are searching for a specific element through an unordered list of N items, then you can expect on average that it will take you something like $N/2$ tries before you hit the right answer. That is, you might get very luck and hit the right answer on the first try, or you might be very unlucky and it takes the full N tries, but on average you will be somewhere in between.

The point is that if you increase the number N, then you expect the number of tries, and the length of time it takes to do the search, to increase like N. Soon after Peter Shor came up with Shor's Algorithm, his colleague Lov Grover came up with a search algorithm on a quantum computer that would scale like $\sqrt{N}$. Here is Grover's original paper:

https://arxiv.org/abs/quant-ph/9605043

If you are searching through, say, a trillion numbers, then this would represent a speed up over a classical computer by a factor of a million. Unlike Shor's Algorithm, which cracks RSA and therefore has driven digital cryptography to other schemes, Grover's Algorithm in principle would have everyday practical use if it can be realized on a large quantum computer. (You need to figure out how to wire up the black box known as an "oracle", though. See below for some details and comments.)

First we will go through Grover's Algorithm step by step for $N = 4 = 2^2$, that is, for an $n = 2$ qubit system. Then we will generalize to an arbitrary number of qubits.

Example: N=4

Let's first calculate the number of operations, on average, it takes to find s specific element among four values. The probability of finding it on the first try is $1/4$. The probability of finding it on the second try, after not finding on the first try, is $3/4 \times 1/3 = 1/4$. The probability of finding on the third try, after not finding it on the first two tries, is $1/2 \times 1/2 = 1/4$. If you don't find it on the first three tries, then you find on the fourth try with certainty, and you'll get to that point with probability $1/4$. Therefore, the average number of times you need to try is

$$\frac{1}{4} \times 1 + \frac{1}{4} \times 2 + \frac{1}{4} \times 3 + \frac{1}{4} \times 4 = 2.5$$

(The Quantum Information textbook gets 2.25, but I don't understand their argument.)

We will see that Grover's Algorithm can find the right answer with only one compute step. In fact, it finds the right answer with 100% probability. (For arbitrary N, we will see that you need to iterate the algorithm until you are at a point where the probability of getting the right answer is "large.")

Here's how Grover's Algorithm works. I will be using a mixture of notations, abstract kets or matrix representations, interchangeably, when it is handier to use one of the other. With $N = 4 = 2^n$ with $n = 2$, we need a system with only two qubits. Label the four indices using these two qubits in the standard fashion, namely

$$|00\rangle \equiv |0\rangle \otimes |0\rangle \qquad |10\rangle \equiv |1\rangle \otimes |0\rangle \qquad |01\rangle \equiv |0\rangle \otimes |1\rangle \qquad |11\rangle \equiv |1\rangle \otimes |1\rangle$$

It will of course be useful to represent the qubit states as the standard two-dimensional column vectors, that is

$$|0\rangle \doteq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{and} \qquad |1\rangle \doteq \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Our job is to find out which of the four pairs of qubits is the "right answer." We start by having the system initialized to the fully entangled state

$$|I\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|11\rangle \doteq \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

so that the probability of measuring the system to be in any state $|i\rangle$ is $P_I(i) = |\langle i|I\rangle|^2 = 1/4$. We can easily get to this state by preparing the system in $|00\rangle$ and then applying a Hadamard gate to each of the two qubits. That is

$$H|0\rangle \doteq \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \doteq \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

so that applying two Hadamard gates to the state $|00\rangle$ means

$$\begin{aligned} (H \otimes H)|00\rangle &= H|0\rangle \otimes H|0\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \\ &= \frac{1}{2}\left(|0\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle + |0\rangle \otimes |1\rangle + |1\rangle \otimes |1\rangle\right) = |I\rangle \end{aligned}$$

The oracle is a "black box" that knows the right answer. It will query the quantum state and load the results into an ancillary qubit $|y\rangle$. We prepare $|y\rangle = |1\rangle$ and also put it through a Hadamard gate, that is

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} \qquad \text{i.e.} \qquad H|1\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)$$

The result of all this so far is that the state of the quantum computer, that is the two "computational" qubits plus the ancillary qubit, ready for query by the oracle, is

$$|I\rangle \otimes \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) = \frac{1}{2}\left(|00\rangle + |10\rangle + |01\rangle + |11\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)$$

In computer-speak, we might refer to these two parts of the state as "registers."

Now is the time for the compute operation, or "step." Each of the four states $|x\rangle$ where $x = 00$, 01, 10, or 11, represents the index of the four-element list we are searching for the element x_0. There is some function $f(x)$, called the *oracle*, for which $f(x) = 0$ if $x \neq x_0$, and $f(x_0) = 1$. The *oracle query* performs an exclusive OR, aka XOR or $\oplus$, between $f(x)$ and the ancillary qubit after the Hadamard gate, that is

$$\begin{aligned} |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle &= |x\rangle\frac{1}{\sqrt{2}}(|0\oplus 0\rangle - |1\oplus 0\rangle) = |x\rangle\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \qquad \text{for} \qquad x \neq x_0 \\ \text{and} \qquad &= |x\rangle\frac{1}{\sqrt{2}}(|0\oplus 1\rangle - |1\oplus 1\rangle) = |x\rangle\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) \qquad \text{for} \qquad x = x_0 \end{aligned}$$

In other words, the oracle switches the sign on the piece of the input state that corresponds to the index we are searching for.

Note that an XOR can be accomplished by a CNOT gate. That is, the target qubit becomes $|1\rangle$ if either the control qubit or target qubit are $|1\rangle$, but not both. If both qubits are $|1\rangle$, then the target bit is $|0\rangle$. This is precisely the XOR function.

There is something perplexing about the oracle, of course. It seems to need to know the right answer. In fact, how one would actually realize a quantum circuit that is the oracle is not clear. Thanks to Dean Lee for pointing me to this discussion:

https://cstheory.stackexchange.com/questions/38538/oracle-construction-for-grovers-algorithm

I guess you can think of the oracle as someone who knows the right answer, and your job is to figure out what they know. Whether or not this means you can actually build a quantum computer to execute Grover's Algorithm to search through a trillion numbers, well, that remains to be seen.

To illustrate what comes next, let's assume that $|x_0\rangle = |10\rangle$. (This is just for illustration. It won't make any difference which of the four components corresponds to the correct index.) Then, after the oracle query, the state of the quantum computer is

$$|\mathcal{O}\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \qquad \text{where} \qquad |\mathcal{O}\rangle = \frac{1}{2}(|00\rangle - |10\rangle + |01\rangle + |11\rangle) \doteq \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \equiv \underline{\mathcal{O}}$$

The oracle query has "pushed a minus sign" onto the component of the state vector that is the index to the correct answer. This doesn't look like it has achieved anything, though, since the probability of getting the result of any one of the four measurements is still 1/4.

However, as we will see now, we can use quantum gates to turn the phase difference of the second term into an amplitude enhancement. This can be done with a matrix $\underline{\underline{D}}$ whose

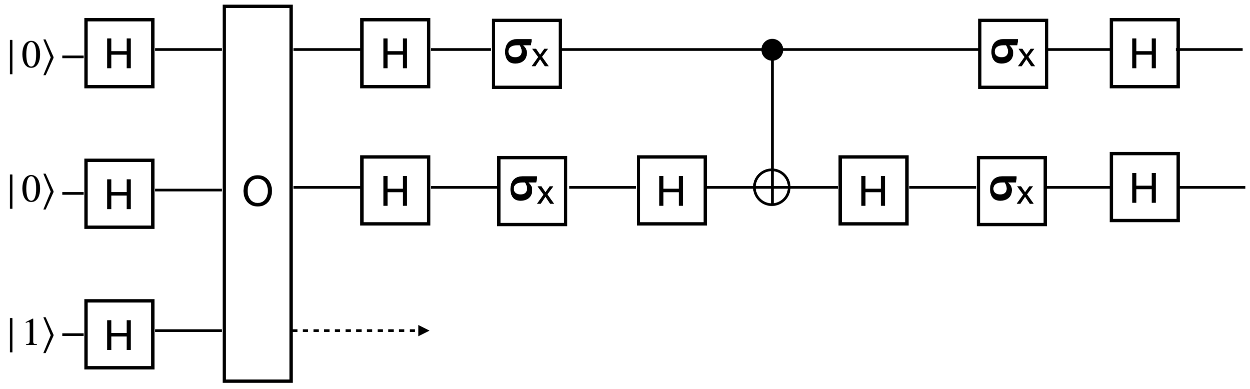

Figure 1: A quantum computer that executes Grover's search algorithm for $N = 4$, that is, two quibits.

elements are $D_{ij} = -\delta_{ij} + 2/2^n$. In our case, for $n = 2$, this is

$$\underline{\underline{D}} = \frac{1}{2}\begin{bmatrix} .-1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \qquad \text{so} \qquad \underline{\underline{D}}\,\underline{\mathcal{O}} = \frac{1}{4}\begin{bmatrix} 0 \\ 4 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

and the system is now in a state where the probability of projecting onto the correct answer is unity. In other words, if we measure the state of the quantum computer at this point, then we will identify the marked state with 100% certainty.

Now let's fill in the last detail, namely how to realize the matrix $\underline{\underline{D}}$. This can be done using a series of Hadamard and σ_x gates. Figure 1 shows how to do this diagrammatically, along with the steps needed to prepare the initial state and carry out the oracle query. Let's take this part of the diagram step by step and see how it in fact builds $\underline{\underline{D}}$. We claim that

$$\begin{aligned} \underline{\underline{D}} &= \left(\underline{\underline{H}} \otimes \underline{\underline{H}}\right) \underline{\underline{D}}' \left(\underline{\underline{H}} \otimes \underline{\underline{H}}\right) \\ \text{where} \qquad \underline{\underline{D}}' &= \left(\underline{\underline{\sigma}}_x \otimes \underline{\underline{\sigma}}_x\right) \left(\underline{\underline{1}} \otimes \underline{\underline{H}}\right) (\mathsf{CNOT}) \left(\underline{\underline{1}} \otimes \underline{\underline{H}}\right) \left(\underline{\underline{\sigma}}_x \otimes \underline{\underline{\sigma}}_x\right) \end{aligned}$$

Recall that the tensor product of two 2×2 matrices $\underline{\underline{A}}$ and $\underline{\underline{B}}$ is

$$\underline{\underline{A}} \otimes \underline{\underline{B}} = \begin{bmatrix} A_{11}\underline{\underline{B}} & A_{12}\underline{\underline{B}} \\ A_{21}\underline{\underline{B}} & A_{22}\underline{\underline{B}} \end{bmatrix}$$

and the two-qubit CNOT gate is

$$\mathsf{CNOT} = \begin{bmatrix} \underline{\underline{1}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \underline{\underline{\sigma}}_x \end{bmatrix}$$

It's easy to work all this out with MATHEMATICA. See the accompanying notebook to find

$$\underline{\underline{H}}\otimes\underline{\underline{H}} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad \underline{\underline{1}}\otimes\underline{\underline{H}} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \quad \underline{\underline{\sigma}}_x\otimes\underline{\underline{\sigma}}_x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

which up to an overall phase gives

$$\underline{\underline{D}}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

and the desired result for $\underline{\underline{D}}$.

Searching among N items

Now let's see how Grover's Algorithm works for a list of length N. We'll find that, unlike in the case for $N = 4$, we do not end up in a state where a measurement of the quantum computer gives the target index x_0 with certainty. Instead, we have to iterate the oracle until we get to the point where a measurement yields x_0 with high probability. We will see a "geometric" argument that the number of iterations needed is on the order of $\sqrt{N}$. You can also show that the probability that you don't measure the right answer is on the order of $1/\sqrt{N}$, but we won't prove this.

We write $N = 2^n$ for convenience, so that we know that we need n qubits, and then we start the same way, with all n qubits in the $|0\rangle$ state and an additional ancillary bit in the $|1\rangle$ state. Applying $n+1$ Hadamard gates to this state of the computer we get

$$(H^{\otimes n}\otimes H)(|00\cdots 0\rangle|\otimes 1\rangle) = \frac{1}{\sqrt{2^n}}\sum_{x=1}^{2^n}|x\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \tag{6}$$

which creates an equal superposition of all of the index states and the ancillary qubit in a state with the all-important minus sign.

Now it's time for the oracle query $|x\rangle|y\rangle \to |x\rangle|y\oplus f(x)\rangle$, where $f(x)$ is the same as before, namely $f(x) = 1$ for $x = x_0$, the target index, and $f(x) = 0$ for $x \neq x_0$. In exactly the same way that it worked for the $N = 4$ case, there is a sign change on the target index. We can write the action of the oracle as

$$\begin{aligned} \mathcal{O}\left\{\frac{1}{\sqrt{2^n}}\sum_{x=1}^{2^n}|x\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)\right\} &= \frac{1}{\sqrt{2^n}}\sum_{x=1}^{2^n}|x\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\oplus f(x)\rangle - |1\oplus f(x)\rangle\right) \\ &= \frac{1}{\sqrt{2^n}}\sum_{x=1}^{2^n}(-1)^{f(x)}|x\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \end{aligned}$$

and, once again, the result is that a minus sign is "kicked back" to the target index state.

For the $N = 4$ case, this relative minus sign was converted into a relative amplitude enhancement of unity using a collection of quantum gates. This won't happen for larger N, and we will instead have to iterate until this relative amplitude enhancement is large enough so that it is highly probable that we identify the target index state.

We nevertheless proceed in the same way. In order to generalize the 4×4 matrix $\underline{\underline{D}}'$ from the two-qubit example, we see that it is the representation of the operator

$$D' = -1 + 2|\mathbf{0}\rangle\langle\mathbf{0}|$$

where $|\mathbf{0}\rangle$ is the n-qubit state with all qubits $|0\rangle$. Therefore define the *Grover iteration* as

$$G = D\mathcal{O} \qquad \text{where} \qquad D = H^{\otimes n}(-1 + 2|\mathbf{0}\rangle\langle\mathbf{0}|)H^{\otimes n}$$

and consider what repeated operation of this operator does to the state (6).

Grover's originally paper proceeds methodically to explain how the algorithm converges through a series of short theorems and lemmas. However, it is not hard to understand the algorithm geometrically, when we realize that

$$\mathcal{O} = 1 - 2|x_0\rangle\langle x_0| \equiv R_{|x_0^\perp\rangle}$$

which is, effectively, a "reflection" about the "plane" perpendicular to $|x_0\rangle$. In other words, if we imagine a two dimensional space spanned by the orthogonal vectors $\{|x_0\rangle, |x_0^\perp\rangle\}$, then for an arbitrary vector $|\alpha\rangle = a|x_0\rangle + b|x_0^\perp\rangle$, we have

$$\mathcal{O}|\alpha\rangle = -a|x_0\rangle + b|x_0^\perp\rangle$$

Now we know that

$$H^{\otimes n}|00\cdots 0\rangle = H^{\otimes n}|\mathbf{0}\rangle = \frac{1}{\sqrt{2^n}}\sum_{x=1}^{2^n}|x\rangle \equiv |S\rangle$$

that is, the equal superposition state. We also know that H is Hermitian, as well as unitary, so $H^2 = 1$ and

$$D = H^{\otimes n}(-1 + 2|\mathbf{0}\rangle\langle\mathbf{0}|)H^{\otimes n} = -H^{2^{\otimes n}} + 2H^{\otimes n}|\mathbf{0}\rangle\langle\mathbf{0}|H^{\otimes n} = -(1 - 2|S\rangle\langle S|) \equiv -R_{|S^\perp\rangle}$$

In other words, the Grover iteration $G = D\mathcal{O}$ is a reflection about the plane perpendicular to $|x_0\rangle$ followed by a reflection about the plane perpendicular to $|S\rangle$, with a factor of (-1).

We can remove the factor of (-1), however, by realizing that a reflection about $|S^\perp\rangle$ is the negative of the reflection about $|S\rangle$. This is easy to see, since

$$R_{|S^\perp\rangle}\left(a|S\rangle + b|S^\perp\rangle\right) = -a|S\rangle + b|S^\perp\rangle = -\left(a|S\rangle - b|S^\perp\rangle\right) = -R_{|S\rangle}\left(a|S\rangle + b|S^\perp\rangle\right)$$

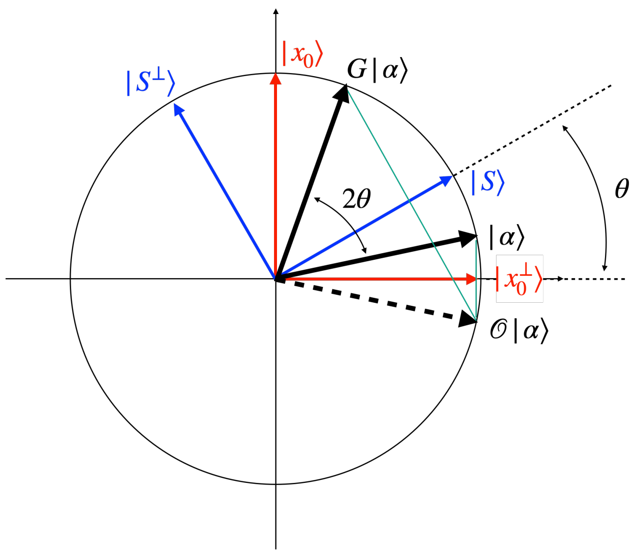

Figure 2: The result $G|\alpha\rangle$ of a Grover iteration on a state $|\alpha\rangle$, pictured as a reflection about the $|x_0^\perp\rangle$ axis followed by a reflection about the $|S\rangle$ axis. The thin green lines are meant to clarify the two reflections. The initial state of the computer is the equal-superposition state $|S\rangle$ and the target index state is $|x_0\rangle$. The angles are discussed in the text.

for an arbitrary vector $a|S\rangle + b|S^\perp\rangle$. Therefore, we can write the Grover iteration as the product of two reflections, namely

$$G = R_{|S\rangle} R_{|x_0^\perp\rangle}$$

The Grover iteration is shown graphically in Figure 2. Let θ be the angle between $|S\rangle$ and $|x_0^\perp\rangle$, that is, the angle between $|S\rangle$ and (the target index) $|x_0\rangle$ is $\pi/2 - \theta$. If the angle between some generic state $|\alpha\rangle$ and $|x_0^\perp\rangle$ is ϕ (not pictured), then the angle between $|\alpha\rangle$ and $\mathcal{O}|\alpha\rangle = R_{|x_0^\perp\rangle}|\alpha\rangle$ is 2ϕ, and the angle between $\mathcal{O}|\alpha\rangle$ and $G|\alpha\rangle = R_{|S\rangle}\mathcal{O}|\alpha\rangle$ is $2(\theta + \phi)$. Therefore, the angle between $|\alpha\rangle$ and $G|\alpha\rangle$ is $2(\theta + \phi) - 2\phi = 2\theta$, as shown in Figure 2.

(The PQCI textbook states explicitly that all of these vectors would like in the same plane, but I don't think I see how to prove that.)

In other words, the geometric effect of the Grover iteration operator G is to rotate a generic state $|\alpha\rangle$ by 2θ. We also have the relationship between the initial state direction and the target state direction as

$$|S\rangle = \sin\theta|x_0\rangle + \cos\theta|x_0^\perp\rangle$$

Therefore, each Grover iteration adds 2θ to this angle, that is

$$G^j|S\rangle = \sin(2j+1)\theta|x_0\rangle + \cos(2j+1)\theta|x_0^\perp\rangle$$

The goal is to find the smallest integer j so that $(2j + 1)\theta$ is as close as possible to $\pi/2$.

Written in terms of the "nearest integer" function, this means

$$j = \mathsf{NInt}\left(\frac{\pi}{4\theta} - \frac{1}{2}\right)$$

Since we are starting from the equal-superposition state, the coefficients of every basis vector is $1/\sqrt{N}$, and

$$\sin\theta = \frac{1}{\sqrt{N}} \approx \theta \qquad \text{for} \qquad N \gg 1$$

and the number of iterations we need to get there is

$$j = \mathsf{NInt}\left(\frac{\pi}{4}\sqrt{N} - \frac{1}{2}\right) = \mathcal{O}(\sqrt{N})$$

This is the basic advantage of Grover's algorithm, that you can sort through N numbers in a time proportional to $\sqrt{N}$, instead of N as with a classical search algorithm.

A nice exercise is to simulate Grover's algorithm on a classical computer with as many qubits as you can easily handle – which in fact turns out to give a significantly large number $N = 2^n$ – and watch the qubit coefficients converge on the target search index.