Lecture 17: Thurs March 23

People often want to know where the true power of quantum computing comes from.

- Is it the ability of amplitudes to interfere with one another?
- Is it that entanglement gives us 2^n amplitudes to work with?

But that's sort of like dropping your keys and asking "what made them fall?"

- Is it their proximity to the Earth?
- Is it the curvature in space-time?

You could come up with all sorts of answers that are perfectly valid.

Last time Tom demonstrated the existence of universal gate sets.

It's worth mentioning that we don't have a criteria to characterize which sets of gates are universal and which aren't. Not many people in the field care about figuring out this particular open problem, since "we have universal gate sets that work, so just roll with it," but it would be nice to know, and you should go figure it out anyways.

It seems like our rules for universal gate sets are just avoiding *certain* bad cases. Do we have formal proof that they work?

Yes. There's a paper from the 90s by Yaoyun Shi on the subject, but it's out of scope for this class.

In designing quantum algorithms, we're ultimately looking to minimize the number of gates required to implement them. That problem turns out to be insanely hard for reasons that have nothing to do with quantum mechanics.

"What's the smallest circuit that solves Boolean satisfiability?"
is a similarly hard problem, for reasons related to P vs NP.

So people design quantum algorithms that center around query complexity. This abstracts away part of the problem by saying:

"There's some Boolean function f: $\{0,1\}^n \longrightarrow \{0,1\}$ and we're trying to learn something about f."

You might want to learn:

Is there some input x where $f(x) = 0$?

Is there some symmetry in the solution?

Etc.

More importantly, we want to know how many queries it takes to solve such a problem.

In this model we abstract out the cost of gates that don't do queries.

To be precise, we map queries as

$$|x, a\rangle \rightarrow |x, a \oplus f(x)\rangle$$ since the transformation must be unitary.

But it can also be thought of as

$$|x\rangle \rightarrow (-1)^{f(x)}|x\rangle$$

Before we jump into a few quantum algorithms, it's worth asking,
"Why do we care about this model? You're debating how you'd phrase your wishes if you found a genie. Who cares?"

You can think of a black box as basically a huge input. Querying $f(i)$ means looking up the i^{th} number in the string.

That allows us to break a problem down to "if you want to do an unordered (or ordered) search, how many queries do you need?"
This is much more reasonable to compute that the alternative.

Another way to think about it:

Imagine I'm writing code, and I have a subroutine that computes $f(x)$. How many times do I need to call the subroutine to find some information about f?

Under the assumption that we can only learn information about f by looking at it's output.

There's a technical question we need to answer…
Suppose we know that there exists a fast algorithm to implement f. Could you then implement the black box behavior $|x, a\rangle \rightarrow |x, a \oplus f(x)\rangle$?

Keep in mind that quantum circuits have to be reversible. We're essentially asking what constraints arise from that.

We know f must be injective.

What if f was a one-way injective function? i.e. there's a unitary C such that $C|x\rangle = |f(x)\rangle$
The problem with this is that a small circuit for C implies that there's a small circuit for C^{-1}, which would imply that there's a small circuit for $f(x)$, such that $C^{-1}||f(x)\rangle = |x\rangle$.

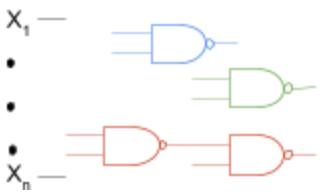
It's worth noting that even though quantum computing can break a few supposedly one-way functions, like finding prime factors, it doesn't provide evidence against the existence of *any* one-way functions.

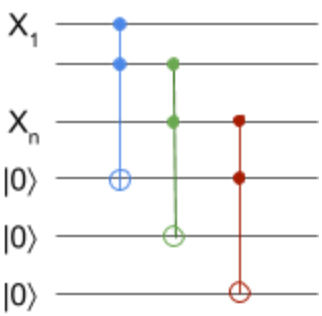
Mapping $f(x)$ and erasing x is much harder for a computer that necessitates reversible circuits. You *could* do $|x, 0\rangle \rightarrow |x, f(x)\rangle$ which doesn't let you reverse f.

To invert f you would need to know that x is $|x,0\rangle = C^{-1}|x, f(x)\rangle$.

Tom showed that a reversible circuit can always simulate a non-reversible circuit, since Toffoli can simulate NAND. However, in reversible computing erasing is expensive.

Imagine a classical circuit (without loss of generality, let's say it's a cluster of NAND gates).





You could simulate this as a reversible circuit by having each NAND replaced with a suitable Toffoli. The problem with this is that you'd get all sorts of undesired results in the intermediate bits—the technical name for this is ***garbage***. Yes, really.
A truly universal algorithm must produce no garbage, because garbage can prevent the desired interference pattern from showing up.

Garbage is the bane of quantum computing, because the point of

quantum algorithms is to create these patterns of interference.

For example, what's the difference between having $\frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and having $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$, but treating the second qubit as garbage?

The garbage creates unwanted entanglement. Looking at a part of the superposition basically turns it into a mixed state.

Suppose you have a circuit to compute f. How do we get a circuit that maps $\sum_x \alpha_x |x, 0\rangle \rightarrow \sum_x \alpha_x |x, f(x)\rangle$ without all the garbage? In the 70s, Bennett invented a trick for this called…

Uncomputing

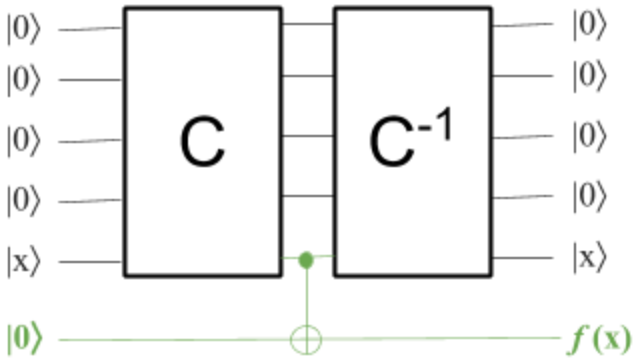
Let's say I have some circuit that maps

$C |x, 0, \ldots, 0\rangle = |x, gar(x), f(x)\rangle$.

First, run the circuit, C.

Then cNOT x. (make a copy of it in a safe place)

Then run the inverse circuit, C^{-1}.



The reason we can copy x in spite of the No Cloning Theorem is that we're assuming that there's a classical answer. This wont work if the output is a general quantum state.

This justifies the quantum query model because if we can compute f at all, then we do have the ability to map $|x, a\rangle = |x, f(x)\rangle$.

With that out of the way, we're ready to talk about some quantum algorithms.

Deutsch's Algorithm

computes the parity of two bits with one query. (the parity of n bits would require n/2 queries).

It basically involves making a state like $\frac{1}{\sqrt{2}}\left((-1)^{f(x)}|0\rangle + (-1)^{f(x)}|1\rangle\right)$ and querying it in the $|0\rangle,|1\rangle$ basis. It uses the *phase kickback trick* to measure phase change.

The basic idea of the phase kickback trick is that we have a quantum oracle that does $\sum \alpha_x |x, y\rangle \rightarrow \sum \alpha_x |x, y \oplus f(x)\rangle$ but we'd rather get a final state in the form $\sum \alpha_x (-1)^{f(x)}|x\rangle$. To accomplish this we put $|-\rangle$ in second register. U_f gives us $\frac{|0\rangle|-\rangle - |1\rangle|-\rangle}{\sqrt{2}}$ and we can interchange $|0\rangle$ - $|1\rangle$ and $|1\rangle$ - $|0\rangle$.

There's a generalization of this, called…

The Deutsch-Jozsa Algorithm

Assume a black box computes $f: \{0,1\}^n \rightarrow \{0,1\}$, and that f is either:

- a constant function — All outputs are 0 or all outputs are 1
- a balanced function — Same number of 0's and 1's in output

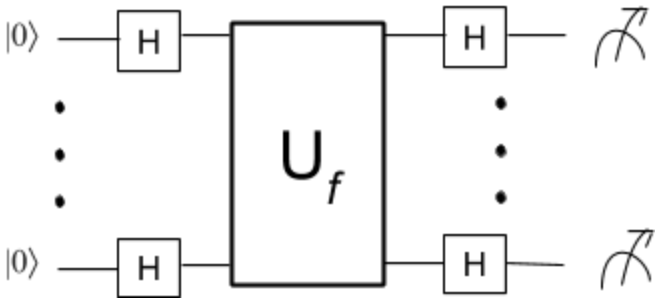
The problem is to decide out which.

Classically, you could look at $2^{n-1}+1$ cases of the function. If all inputs match, then the function is constant. You can improve this through random sampling. On average, you'd need about 5 or 6 queries to get an answer with a sufficiently small probability of error.

We'll see how a quantum algorithm can solve this perfectly with only one query.

Truth is, this isn't a hard classical problem, and so we wont get that big of a speed up. This is why, initially, people didn't care about quantum computing. They figured all advantages would be in the same vein.

Here's the quantum circuit for it:



You'll begin to notice that some patterns appear a lot in quantum algorithms.

- you start by putting everything in superposition
- Then query *f*, mapping each *x* to $(-1)^{f(x)}|x\rangle$
- Then measure (from the superposition) the information that we want to know.

If you can't figure out what's next in a quantum algorithm, a round of Hadamards is always a good guess.

So we want to know the probability of getting back the state $|00\ldots0\rangle$.

Let's call the circuit C. We can compute $|\langle 00\ldots0|\ C\ |00\ldots0\rangle|^2$

What's the final amplitude of the C state?

Well H maps $|0\rangle \rightarrow |+\rangle$ and $|1\rangle \rightarrow |-\rangle$.

For an arbitrary $|x\rangle$ it'll map $|x\rangle \rightarrow \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}}$

and H maps a given string $|x_1, \ldots, x_n\rangle \rightarrow \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}} \otimes \ldots \otimes \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}}$.

After the oracle is applied, you get.

$$= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$$

Note: x·y is their inner product. Pick up a phase if $x_i = y_i = 1$.

So keeping track of each basis state individually, you get

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}} (-1)^{f(x)} |x\rangle$$

And after another Hadamard

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle$$

A shortcut to simplify this is to ask, "What is the amplitude when $y = |00\ldots0\rangle$?"

It would be $\frac{1}{2^n} \sum_{x \in \{0,1\}} (-1)^{f(x)}$

What does the value of this have to do with *f* being constant?
If *f* is constant, then this is either 1 or -1.
If *f* is balanced, then this is 0.

The first problem we'll see next time is

The Bernstein-Vazirani Problem

Given a black box function $f: \{0,1\}^n \rightarrow \{0,1\}$
and a promise that $f(x) = s \cdot x \pmod 2$ for some secret string $s \in \{0,1\}^n$

The problem is to find *s*.

Classically, you could get an answer one bit at a time by querying $f(1000) = s_1$, $f(0100) = s_2$, $f(0010) = s_3$, $f(0001) = s_4$

But there's no algorithm that can do better, since each query can only provide one bit of information.

The Bernstein-Vazirani Algorithm, however, can solve this quantumly with only one query.