13.4 Bernstein-Vazirani

The Bernstein-Vazirani algorithm is given black box access to a function $f : \{0,1\}^n \to \{0,1\}$ that we know is in the form

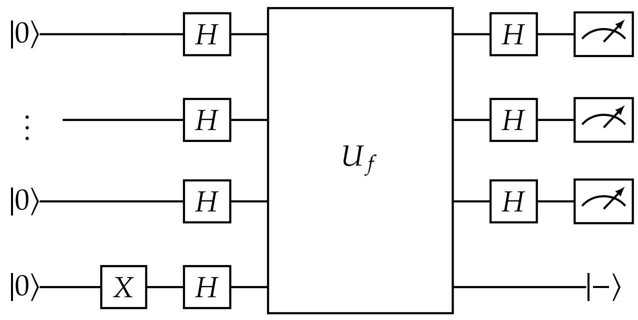
$$f_s(x) = x \cdot s \pmod 2 \tag{105}$$

for some mystery string $s \in \{0,1\}^n$. The goal of this algorithm is to figure out what s is.

Question 129. Let's consider an example where $n = 5$ and the secret string is $s = 10110$. What is $f(11101)$?

Question 130. What is a strategy we can use using a classical computer to decide what s is? What is the optimal query complexity classically?

Here is the circuit for the Bernstein-Vazirani algorithm:



Question 131. What is the state of the algorithm before the query to U_f?

Question 132. What is the state of the algorithm after the query to U_f?

Question 133. What is the state of the algorithm after the second layer of H gates?

Bernstein and Vazirani chose this problem since there is a way to have all the amplitudes for $y \neq s$ interfere destructively to become 0, while the amplitudes for s all "point in the same direction" and interfere constructively to become 1. We have found a way to achieve a linear query complexity speed up using a quantum algorithm, but can we do even better? Are there setting where we can achieve exponential speed up?

13.5 Simon's Algorithm

In this problem, we will consider a function with an n bit input and an n bit output. The function $f : \{0,1\}^n \to \{0,1\}^n$ will encode a secret string s in the following way.

$$f(x) = f(y) \iff x \oplus y = s \iff x \oplus s = y \tag{106}$$

As we have been doing, given blackbox access to this function, the goal is to find s.

Question 134. Let f be a function that takes an n bit input and satisfies the property above. What is the size of the domain of this function? What is the size of the range of this function?

To determine what s is, we need to find a pair x and y such that $f(x) = f(y)$, and then take the sum module 2 of these strings to recover s.

Question 135. How many queries will we need classically in the worst case to determine s?

What if we use a randomized classical algorithm? In this case, we can show that we will require approximately $\sqrt{2^n} = 2^{n/2}$ queries. Let's try to prove this together. To prove this, we will use a general version of the Birthday Paradox.

Suppose we have a set of items, each with a uniformly random tag from $\{1, 2, \ldots, T\}$. How many samples do we need to collect before we have at least two items with the same tag with probability greater than $1/2$?

Question 136. What is the probability that a random pair of items have matching tags?

Question 137. Suppose we have chosen m items so far. How many different ways can we pair two items from this set (the tags do not have to match)?

Question 138. Determine how many items m we have to choose until the probability that there is a collision is over $1/2$.

Now suppose that this randomized algorithm queries the function using t bit strings, x_1, x_2, $\ldots$, x_t.

- If we find a pair such that $f(x_i) = f(x_j)$, then we are done.
- If none of these $x_i's$ are matches, then we know that $s \neq x_i \oplus x_j$ for all i, j pairs. In other words, we have ruled out $\begin{pmatrix} t \\ 2 \end{pmatrix} \sim \frac{t^2}{2}$ possibilities, and all other choices are equally likely. In the worst case, we need to find t such that the number of items we rule out equals all possible inputs.

We can conclude then, that classically we need at least $\Omega(2^{n/2})$ queries.

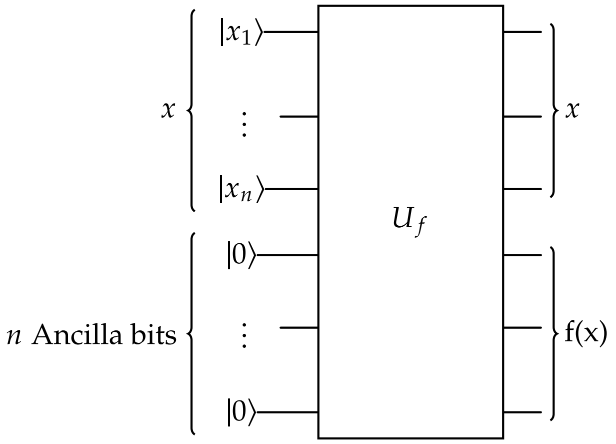
How can we solve this problem using a quantum computer? The unitary encoding the function would act as follows:

$$U_f(\vec{x},\vec{0}) = (\vec{x},\vec{0}\oplus f(x)) \tag{107}$$

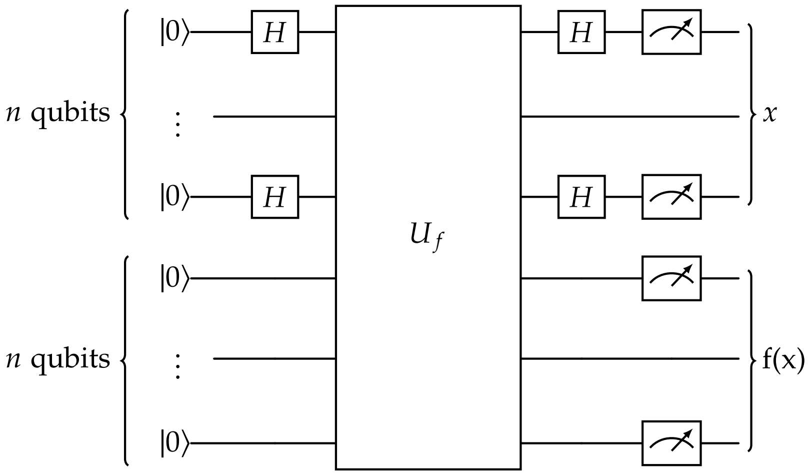
or in ket notation

$$U_f\,|x\rangle\,|0\rangle = |x\rangle\,|f(x)\rangle\,. \tag{108}$$

As a circuit, they would look like the following.



Now the circuit for solving Simons problem is just a small addition to the above circuit and is drawn below.



The neat thing about this algorithm is that we don't actually care about what our measurement result is, but just the interference pattern that is created. Let's go through the circuit to see what we mean by this.

Question 139. What is the state of the algorithm before the U_f gate?

Question 140. What is the state of the algorithm after the U_f gate?

It turns out that we can measure the last n qubits before applying the H gates on the first n qubits. The output distribution will be the same in either case!

Question 141. Suppose that upon measuring the second register at this stage, we get the result $|w\rangle$. What is the state of the first register?

It would be great if we could have multiple copies of the above state, because then we can directly measure the first register to recover all the relevant states. The problem is that if we rerun this experiment, it is extremely unlikely (how unlikely?) that we measure $|w\rangle$ again!

Instead, what this circuit is doing is measuring in the H basis by applying the H gates.

Question 142. What is the state of the superposition after the final Hadamard gates?

Question 143. Suppose we measured the first register and observe some random $|z\rangle$. What can we say about the coefficient of such a $|z\rangle$?

This can be analyzed using modular arithmetic:

$$x \cdot z \mod 2 = y \cdot z \mod 2 \tag{109}$$

$$(x - y) \cdot z \mod 2 = 0. \tag{110}$$

When working in binary, $x - y$ is equivalent to $x \oplus y$. Therefore what we get is that for the string z we recovered,

$$(x \oplus y) \cdot z = s \cdot z = 0. \tag{111}$$

So in 1 run of Simon's algorithm we found a random z that is orthogonal to s!

The measurement yields a random z such that $s \cdot z \equiv 0 \pmod 2$. We can repeat this $O(n)$ times to get a set of linearly independent strings who are all orthogonal to s. Once we have this, we can use Gaussian elimination (mod 2) to find s in $O(n^3)$ time.

$$\begin{bmatrix} --- & z_1 & --- \\ --- & z_2 & --- \\ & \vdots & \\ --- & z_m & --- \end{bmatrix} \cdot \begin{bmatrix} | \\ s \\ | \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \tag{112}$$

This gives us a polynomial time quantum algorithm to find s, whereas classically the best we could do was still exponential.

13.6 Query Based Algorithm Wrap Up

We've looked at several algorithms in this strange query model, which achieves speedups in a non-standard way. You may be suspicious that we are sweeping too many details under the rug, and for that you would be correct. To actually *implement* Simon's algorithm, you need an actual circuit to compute f, and when given to the actual circuit (as opposed to a black-box oracle), classical algorithms can exploit the details of the circuit to significantly reduce the number of queries.

Unfortunately, because of this reason these algorithms we have seen so far are not actually very practical for finding ways to speed up our computations. However, they provided valuable practice using some tools that will be useful for analyzing other quantum algorithms. Furthermore, I hope it gave you a peek into the workflow of a computer science researcher, and some ways that we try to separate the power of classical and quantum computing. It is not perfect, but it provides some concrete examples and intuition behind why quantum computers may excel at certain tasks over classical computers.