Lecture 7 – Quantum algorithms 2

Last time we looked at the quantum algorithms for the Deutsch-Jozsa, Bernstein-Vazirani and Simon problems.

In this lecture we'll cover one of the most important q. algorithms: Shor's algorithm for factoring.

Outline

- Period finding and order finding
- The quantum Fourier transform (QFT)
- Factoring as an order finding problem
- Shor's algorithm

Period finding and order finding

The period finding problem

In: $f: \{0,\dots d-1\} \to \{0,\dots d-1\}$

f is periodic of period $k > 0$ $(k < \sqrt{d})$

$f(x) = f(x+k)$

↳ addition is mod d

Out: k

Note that this problem is similar to Simon's problem

For Simon's we had $f: \{0,1\}^n \to \{0,1\}^n$

$f(x) = f(x \oplus s)$ and we had to find s.

For much of the same reasons as with Simon's problem any classical algorithm for period finding requires $\Omega(\sqrt{k})$ queries to f. Typically $k \sim \sqrt{d}$ so if d is large

problem is classically intractable. We would like an algorithm that runs in time $O(\text{poly}(\log(d)))$.
We will see that this is possible quantumly.

Before that, let's consider a related problem

The order finding problem

In: $g \in G$, where G is a finite group (for which we have a presentation)

Out: smallest $k > 0$ s.t $g^k = e$
↳ identity elem of G

k is referred to as the order of g.

E.g.: $G = (\{0,1,2,3,4\}, +\bmod 5)$
(usually denoted $\mathbb{Z}/n\mathbb{Z}$)

$g = 3$

$g^1 = 3 \quad g^2 = 3+3 \bmod 5 = 1 \quad g^3 = 4$

$g^4 = 2 \quad g^5 = 0$

$G = (\{1, 3, 7, 9\}, \cdot \bmod 10\}$

(usually denoted $(\mathbb{Z}/n\mathbb{Z})^\times$)

$g = 7$

$g^1 = 7,\ g^2 = 7 \cdot 7 \bmod 10 = 9,\ g^3 = 3$

$g^4 = 1$

$G = (\{1, 5, 7, 11\}, \cdot \bmod 12\}$

$g = 5$

$g^1 = 5,\ g^2 = 1$

Order finding reduces to period finding

(order finding $\leq$ period finding)

Let $f(x) = g^x$

Note that if g has order k, then $f(x) = f(x+k)$

$\Rightarrow f$ is periodic with period k

Finding $k \Rightarrow$ we find order of g.

Again this problem seems hard classically for general groups.

But there is an efficient quantum algorithm inspired by Simon...

Quantum Fourier Transform (QFT)

The QFT is just the unitary version of the DFT (discrete Fourier transform) over $\mathbb{Z}_{2^n}$, where n is the number of qubits.

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

($x \cdot y$: interpreted as integers)

where w is the $N = 2^n$ root of unity $w = e^{\frac{2\pi i}{N}}$

Why is QFT unitary?

First look at the matrix entries

$$QFT(x,y) = w^{x\cdot y} \rightarrow \text{symmetric matrix}$$

$$QFT^{+}(x,y) = (w^{*})^{x\cdot y}, \quad w^{*} = e^{-\frac{2\pi i}{N}} = w^{-1}$$

$$(QFT\cdot QFT^{+})(x,z) = \frac{1}{2^n}\sum_{y} w^{-yz}\cdot w^{xy}$$

$$(QFT\cdot QFT^{+})(x,x) = \frac{1}{2^n}\sum_{y} 1 = 1$$

$$(QFT\cdot QFT^{+})(x,z) = \frac{1}{2^n}\sum_{y} w^{y(x-z)} = \quad x\neq z$$

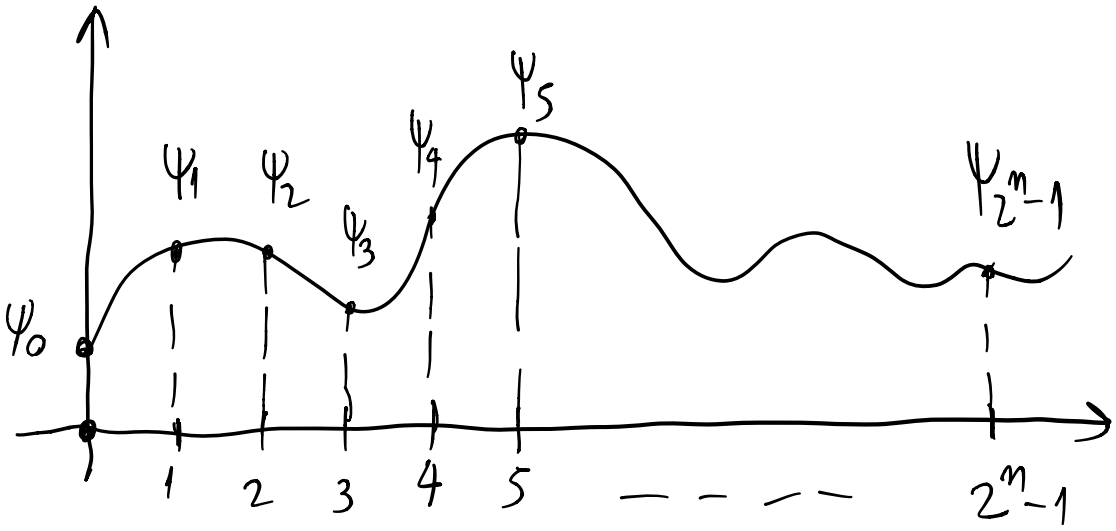
$$= \frac{1}{2^n}\sum_{y}\left(w^{x-z}\right)^{y} = 0$$

So QFT is unitary. What is its interpretation?

Suppose $|\psi\rangle = \sum_{x} \psi_x |x\rangle$

We can think of this vector as a list of discrete

samples from same function.



The QFT is applying the DFT to this set of points.

DFT decomposes a function (signal) into a linear combination of simple periodic functions (sines and cosines or $e^{i\varphi}$)

If the function is periodic, then the Fourier transform will have high weight on components proportional to the period.

Take a very simple periodic function

$f(x) = 1$ for $x = x_0 + r \cdot k$, for some $x_0, r \geq 0$

$f(x) = 0$ for all other x values.

What does this look like?

These are our ψ_i values, the amplitudes of the state on which we apply QFT

So as a state we have $|\psi\rangle = \sum_r |x_0 + r\cdot k\rangle$

(suitably normalized)

Let's apply QFT to this state

$$\text{QFT}\,|\psi\rangle = \sum_r \text{QFT}\,|x_0 + r\cdot k\rangle$$

$$= \sum_r \sum_y w^{y\cdot(x_0+r\cdot k)}\,|y\rangle$$

So the amplitude of any particular y value is

$$\sim \left| \sum_r w^{y\cdot(x_0 + r\cdot k)} \right| = \left| w^{y x_0} \cdot \sum_r w^{y\cdot r k} \right| =$$

$$= \left| \sum_r \left(w^{y\cdot k}\right)^r \right|$$

Note that this is large when $w^{y\cdot k} \sim 1$

Since $w = e^{\frac{2\pi i}{N}}$, this translates to $y\cdot k \sim m\cdot N$ for some $m \geq 0$

So $y \sim \frac{m\cdot N}{k}$

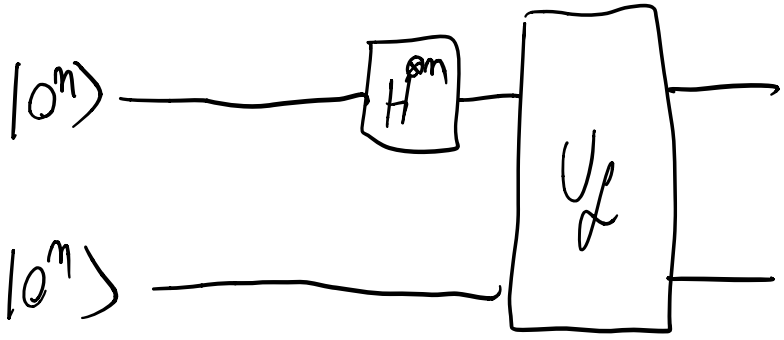
Thus, multiples of $\frac{N}{k}$ have high amplitude (in absolute value). If we view $\frac{1}{k}$ as a frequency of our function, we see that we are getting multiples of the "fundamental" frequency (the frequency of the function we Fourier transformed)

This is the key to quantum period finding!

Given some periodic f, of period k, our goal is to prepare $|x\rangle + |x+k\rangle + |x+2k\rangle + \cdots$

How? Like in Simon's algorithm!

First prepare $\sum_{x=0}^{N-1} |x\rangle|f(x)\rangle$ using



Then measure second register. We'll get some value $|f(x_0)\rangle$ and so top register collapses to

$$\sum_r |x + r \cdot k\rangle$$

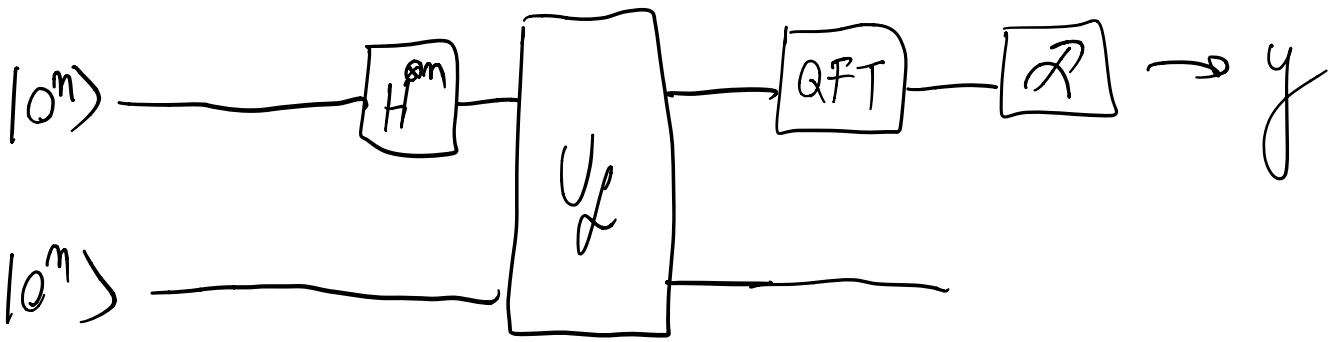
as desired.

We then apply QFT to the top register. Is there an efficient decomposition of QFT in terms of the

{H, T, CNOT} gate set? Yes, though we won't cover it here. It's possible to implement QFT with $O(n^2)$ (in fact less) gates from the universal set.

As we saw, if we apply QFT and measure, we get y values such that $y \sim \frac{m \cdot N}{k}$, for some $m \geq 0$.

So our circuit is



Say we do this twice and get $y_1 \sim \frac{m_1 N}{k}$, $y_2 \sim \frac{m_2 N}{k}$

Note that $\frac{y_1}{N} \sim \frac{m_1}{k}$; $\frac{y_2}{N} \sim \frac{m_2}{k}$.

Can use a procedure called continued fraction expansion to get closest $\frac{a_1}{b_1}$ to $\frac{m_1}{k}$ and $\frac{a_2}{b_2}$ to $\frac{m_2}{k}$

with $\gcd(a_1, b_1) = 1$, $\gcd(a_2, b_2) = 1$ and $a_1, a_2, b_1, b_2 < N$.

Since (m_1, k) and (m_2, k) are not necessarily co-prime, it need not be that either b_1 or b_2 is k. However, with good probability $\text{lcm}(b_1, b_2) = k$.

If we repeat this a couple of times (constant number of times), we find k!

This is the quantum period finding algorithm.

From the reduction order finding $\leq$ period finding, we also solve order finding.

Finally, to solve factoring, we need factoring $\leq$ order finding.

This is the last ingredient for Shor's algorithm!

Factoring

In: $N = p \cdot q$, p, q - prime

Out: (p, q)

(there are versions that work even for more general composites)

How hard is factoring classically?

Naive algorithm

```
for r in range(2, √N):
    if gcd(r, N) > 1
        return (gcd(r, N), N/gcd(r, N))
```

Worst case runtime is $\sqrt{N} = 2^{n/2}$

Better algorithm - Pollard Rho; runtime $\sqrt[4]{N} = 2^{n/4}$

Best known algorithm - General Number Field Sieve (GNFS)

Runtime $2^{O(\sqrt[3]{n})}$ - subexponential time

GNFS is much better than previous approaches, but still inefficient.

No known poly-time algorithm.

Cryptographic protocols like RSA rely on this fact to achieve security!

Factoring $\leq$ Order finding (Miller '75)

$N = p \cdot q$

Consider $G = (\mathbb{Z}/N\cdot\mathbb{Z})^{\times}$

$|G| = (p-1)\cdot(q-1)$

1. Pick random g in G as follows

Pick random g in range $(2, N-1)$

if $\gcd(g, N) > 1 \Rightarrow$ we found p and q and we stop. This is unlikely!

$\Rightarrow$ g is likely co-prime with N

2. Use order finding to compute k s.t

$$g^k = 1 \bmod N$$

3. If k is odd return to 1.
(this happens with prob $\sim 1/2$)

4. For even k, $k = 2 \cdot s$, we have

$$g^{2s} = 1 \bmod N$$

$$\Rightarrow g^{2 \cdot s} - 1 = 0 \bmod N$$

$$\Rightarrow (g^s - 1)(g^s + 1) = 0 \bmod N$$

$\Rightarrow$ either $g^s - 1$ or $g^s + 1$ is a multiple of N

(happens with constant probability). In this case, return to 1.

Otherwise, it must be that

$$\gcd(g^s - 1, N) = p \text{ or } q$$
$$\gcd(g^s + 1, N) = q \text{ or } p$$

In this case we are done!

gcd - classically efficient with Euclid's algorithm

$g^x \bmod N$ - classically efficient with repeated squaring.

Only problematic step is order finding. Classically, this seems hard, but quantumly we saw that it can be performed efficiently.

We just need an efficient implementation of U_f,

$$U_f |x\rangle|y\rangle = |x\rangle|y + g^x \bmod N\rangle$$

(or $U_f |x\rangle|y\rangle = |x\rangle|g^x \cdot y \bmod N\rangle$)

For any fixed k, we can do $|y\rangle \to |y \cdot g^k \bmod N\rangle$ with repeated squaring.

```
repeatSquaring(g, k):
    if k == 0
        return 1
    if k % 2 == 1
        g · repeatedSquaring(g, k-1);
    res = repeatedSquaring(g, k/2)
    return res^2
```

Can do this quantumly with Toffoli gates, since this is just a classical procedure. Runtime is $O(\log k)$

But what about $|x\rangle|y\rangle \to |x\rangle|g^x \cdot y \bmod N\rangle$

Note that $x = x^{(1)} \cdot x^{(2)} \cdot \ldots \cdot x^{(n)}$ – binary representation

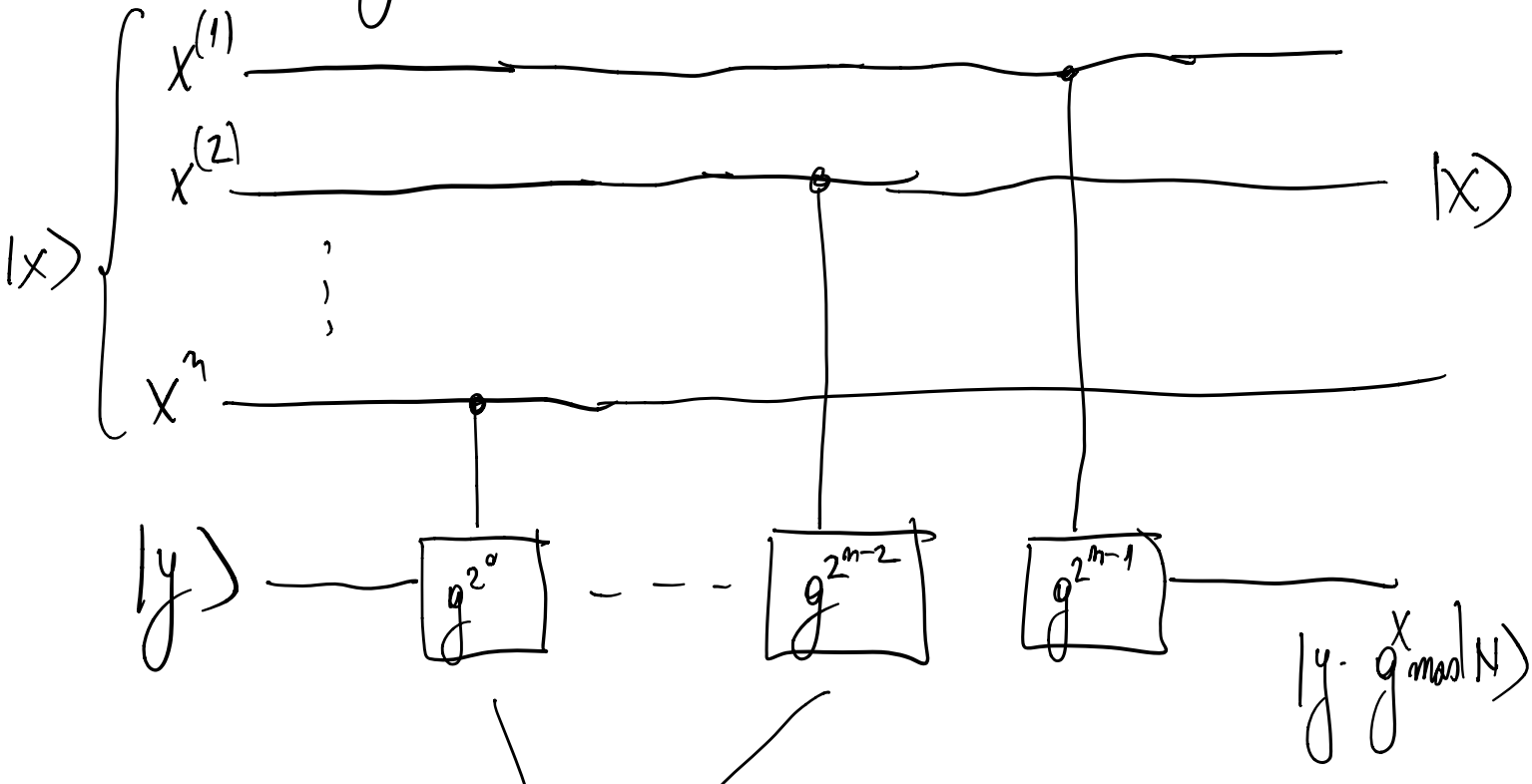
So $x = 2^0 \cdot x^{(n)} + 2^1 \cdot x^{(n-1)} + \ldots + 2^{n-1} \cdot x^{(1)}$

$$g^x = g^{2^0 \cdot x^{(n)} + 2^1 \cdot x^{(n-1)} + \ldots + 2^{n-1} \cdot x^{(1)}} =$$

$$= g^{2^0 \cdot x^{(n)}} \cdot g^{2^1 \cdot x^{(n-1)}} \cdot \ldots \cdot g^{2^{n-1} \cdot x^{(1)}}$$

But we can do g^{2^k} with repeated squaring

So for g^x we have



These gates perform $|y\rangle \to |y \cdot g^{2^k} \bmod N\rangle$

Putting everything together, we get Shor's algorithm!

Quantum order finding can solve other interesting problems efficiently as well (like computing the discrete logarithm), but that's a topic for another time...