CS 758/858: Algorithms

http://www.cs.unh.edu/~ruml/cs758

Quantum Computing

Quantum Algorithms

The two most well-known are:

Shor's Algorithm: factor integers in time $O((\log N)^2(\log\log N)(\log\log\log N))$. exponentially faster than best known classical algorithm ($O(e^{1.9(\log N)^{1/3}(\log\log N)^{2/3}})$). complexity of factoring unknown.

Grover's Algorithm: given $f()$ and y, find x such that $f(x) = y$ when domain is of size N. time $O(\sqrt{N})$ instead of $O(N)$ linear search. these methods are optimal for quantum and classical computers, respectively.

Quantum Computers

quantum supremacy is very recent and still rather theoretical

Shor's Algorithm

Outline

1. turn factoring problem into period-finding
2. solve period finding using quantum algorithm for Fourier analysis

Period Finding

powers of 2: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...
$2^k \mod 15$: 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, ... (period 4)
$2^k \mod 21$: 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, ... (period 6)

Euler (1760s): consider $x^k \mod n$. If $n = p \times q$ with p, q prime, then period will divide $(p-1)(q-1)$.

$n = 15, p = 3, q = 5, (p-1)(q-1) = 8$. 4 divides 8.
$n = 21, p = 3, q = 7, (p-1)(q-1) = 12$. 6 divides 12.

Finding period of $x^k \mod n$ tells us about prime factors of n.

Progress

knowing many random divisors of $(p-1)(q-1)$ (eg, by using different values of x) allows us to learn $(p-1)(q-1)$.

using magic, knowing $(p-1)(q-1)$ will get us p and q (the prime factors).

Period Finding (again)

powers of 2: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...
$2^k \mod 15$: 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, ... (period 4)
$2^k \mod 21$: 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, ... (period 6)

there is $0 < r \leq n$ such that $x^r \mod n = 1$

with prob > 0.25, $(x^{r/2} + 1)(x^{r/2} - 1) = kn$ for some k

both $gcd(x^{r/2} + 1, n)$ and $gcd(x^{r/2} - 1, n)$ will be non-trivial with substantial probability (if not, re-pick x) and both will be factors of n.

Euclid's gcd algorithm (300BC) is $O(b)$

Example: 15

Let's factor 15.

Pick random $x = 11$.
$x^k \mod n = 11^k \mod 15$

$11^k = 1, 11, 121, 1331, 14641, ...$
$11^k \mod 15 = 1, 11, 1, 11, 1, ...$

So period $r = 2$.

$gcd(x^{r/2} + 1, n) = gcd(12, 15) = 3$
$gcd(x^{r/2} - 1, n) = gcd(10, 15) = 5$

So period finding is one way to find factors.

Example: 15

Let's factor 15.

Pick random $x = 11$.
$x^k \mod n = 11^k \mod 15$

$11^k = 1, 11, 121, 1331, 14641, ...$
$11^k \mod 15 = 1, 11, 1, 11, 1, ...$

So period $r = 2$.

$gcd(x^{r/2} + 1, n) = gcd(12, 15) = 3$
$gcd(x^{r/2} - 1, n) = gcd(10, 15) = 5$

So period finding is one way to find factors.

Problem: How to find period? It might be very large, eg, $O(n)$

A Quantum of Solace

n quantum bits can be in 2^n states simultaneously
(like a probability distribution but with complex probabilities?)

create state involving $x^k \mod n$ for many k simultaneously. use repeated squaring to reach high k in $\lg n$ steps.

(Unfortunately, quantum modular exponentiation is slow ($O(b^3)$). This is the bottleneck of the algorithm.)

period is global property of the sequence, so will not get washed away

to detect, use Fourier analysis

Fourier Analysis

Baron Jean Baptiste Joseph Fourier (1822): any function can be represented as a(n infinite) combination of sines

youtube: 'fourier transform', Eugene K, 6:44
youtube: 'spectrogram audacity', exploring, 2:18

YouTube: spectrogram: audacity, Beethoven

Quantum Fourier Transform

quantum superpositions can be manipulated using linear algebra (with unitary matrices). apparently this is enough for Fourier analysis. time complexity is $O(b^2)$

with high probability, amplitudes of values other than the true period will interfere (cancel each other out), leaving the true period as the most probable state to measure. Ie, we are really only sampling the Fourier transform.

Outline

1. turn factoring problem into period-finding
2. solve period finding using quantum algorithm for Fourier analysis

factoring 100 digit numbers in 100^3 time!

time for quantum cryptography!

Church-Turing

Factoring might be first counter-example to 'strong' Church-Turing: super-polynomial gap between Turing machine and quantum computer. Not sure until complexity of classical factoring is known.

References

I recommend Scott Aaronson's talk on YouTube on "Quantum Computing and the Limits of the Efficiently Computable". Starts at 5:30. Quantum at 29:30.

EOLQs

Nope.

Feel free to collaborate on your studying,
and good luck on the final exam!