CS290A, Spring 2005:

Quantum Information & Quantum Computation

Wim van Dam

Engineering 1, Room 5109
vandam@cs

http://www.cs.ucsb.edu/~vandam/teaching/CS290/

Administrivia

- Next week talk by Matthias Steffen on “Nuclear Magnetic Resonance” (NMR) quantum computing.
- Final will be an exam *à la* last week’s Midterm
- This week: quantum Fourier transform, Shor’s algorithms for factoring and discrete logarithms, Grover’s search algorithm.
- Other questions?

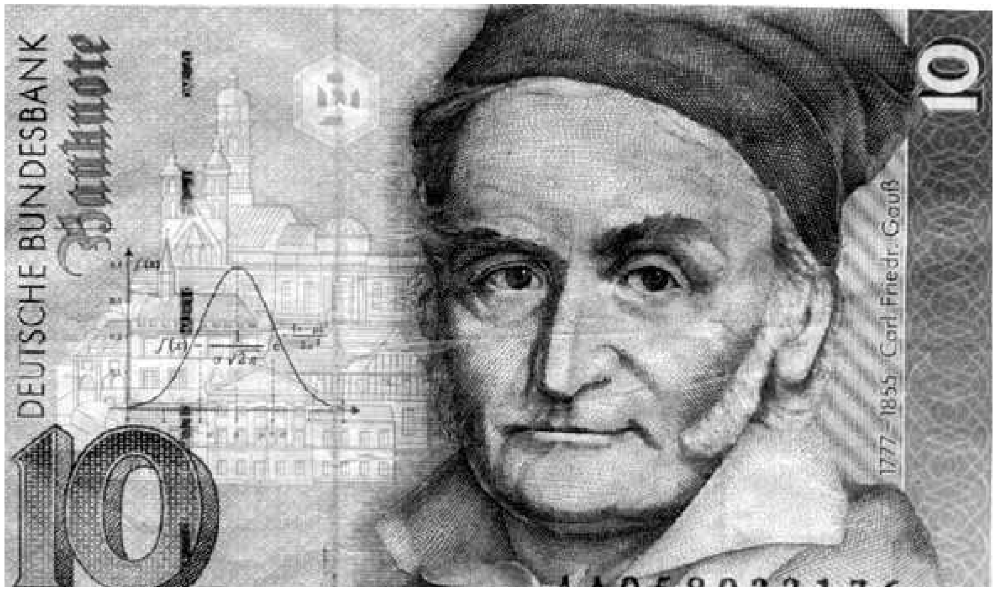
Last Week

- Query complexity: Searching a database of size N can be done with $\Theta(\sqrt{N})$ quantum queries to F.
- This quadratic speed-up is nice, but what we really want is an exponential speed-up.
- No snake oil; we cannot search blindly a database with $O(\log N)$ queries: there is no straightforward way of solving NP-complete problems in polynomial time.
- We are able to solve some problems efficiently with a quantum computer that —as far as we know— require exponential resources with a classical computer.

Primes vs Composite Numbers

"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length… Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated."

— **Carl Friedrich Gauß**,
Disquisitiones Arithmeticæ 1801



Primality Testing

- **Let N be an n-bit integer.**
  Question: Is N prime? (efficient ~ poly(n) operations)

- **Efficient primality testing:**

- **Probabilistic** tests of complexity $O(n^3)$:
  Solovay-Strassen [1977], Miller-Rabin [1976/80]

- The Agarwal-Kayal-Saxena primality test [AKS2002] is a **deterministic** algorithm with running time $O(n^{6+\varepsilon})$

- (Assuming the Riemann hypothesis, the Miller-Rabin algorithm is deterministic as well.)

Factoring Integers

- **Let N be an n-bit integer.**
  Question: What are the prime factors of N?

- Relevant for breaking RSA cryptography

- Best known classical algorithm:
  "Number Field Sieve" [Pollard 1988]
  Time complexity: $\exp(\sqrt[3]{\log(N)} \cdot \sqrt[3]{\log\log(N)^2} \cdot O(1))$

- [**`NFSNET.ORG`**, September 2004]:
  Factorization of a 173 digit number.

Shor’s Factoring Algorithm

[Peter Shor, 1994]:
There exists a quantum algorithm that finds the prime factors of an integer N in time $O((\log N)^3)$

[Chuang et al, 2001]

Experimental implementation for N=15.

To understand Shor’s algorithm we have to look at:

- Quantum Fourier Transform

- Classical Number Theory

Quantum Fourier Transform

Consider the mod N numbers {0,1,2,…,N–1}.
The “Quantum Fourier Transform over $\mathbb{Z}_N$” is defined for each $x \in \{0,1,\dots,N-1\}$ by

$$\boxed{|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i \cdot xy/N} |y\rangle}$$

Hence for each superposition over mod N:

$$\boxed{\sum_{x=0}^{N-1} \alpha_x |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} \alpha_x \cdot e^{2\pi i \cdot xy/N} |y\rangle}$$

Important fact: The QFT can be efficiently implemented in circuit size poly(log(N)) for each N.

Some Small Fourier Transforms

- For N=2,3,4 we have the following transformations:

$$\mathrm{Four}_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (= \mathrm{H})$$

$$\mathrm{Four}_3 = \frac{1}{\sqrt{3}}\begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{pmatrix} \quad \text{with } \omega = e^{2\pi i/3}$$

$$\mathrm{Four}_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

Note unitarity

Properties of Four_N

The definition: $\text{Four}_N : |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i \cdot xy/N} |y\rangle$

Hence: $\langle y | \text{Four}_N | x \rangle = \frac{1}{\sqrt{N}} e^{2\pi i \cdot xy/N}$

and: $\text{Four}_N = \frac{1}{\sqrt{N}} \sum_{x,y=0}^{N-1} e^{2\pi i \cdot xy/N} |y\rangle\langle x|$

The inverse: $\text{Four}_N^{-1} : |y\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} e^{-2\pi i \cdot yz/N} |z\rangle$

Know your phase summations…

Example

- What happens if you apply Four_N twice to $|0\rangle$?

$$|0\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} |y\rangle$$

$$\mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} e^{2\pi i yz/N} |z\rangle$$

$$= \frac{1}{N} \sum_{z=0}^{N-1} \left(\sum_{y=0}^{N-1} e^{2\pi i yz/N} \right) |z\rangle$$

The (summation) is 0 if $z \neq 0$ and N if $z=0$.
Hence the outcome state is $|0\rangle$.

Question: What happens if we apply Four_N twice to a basis state $|x\rangle$ with $0<x<N$?

More About Fourier

- Traditionally, Fourier transforms are used to detect periodic signals (depending on their frequencies).
- In quantum computing we will use the QFT to determine the periodicity of a function F.

- Already interesting by itself, this periodicity finding subroutine can be used to factorize numbers and calculate discrete logarithms over $\mathbb{Z}_N$.

- See later Handouts for more technical details and a description of efficient circuits to implement Four_N.

Periodicity Problem

Consider function F:{0,…,N–1} $\rightarrow$ S

Assume that: F has period r
F is bijective on its period

$$\boxed{F(x) = F(y) \text{ if and only if } x = y \bmod r}$$

Task: determine r (efficiently ~ poly(log N)

Note: This is the kind of global property that quantum computing is useful for.

Periodicity Algorithm (1)

Start with a uniform superposition of x values:

$$\boxed{\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\left|x,0\right\rangle}$$

Calculate the periodic function F for these values:

$$\boxed{\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\left|x,F(x)\right\rangle \quad\approx\quad \frac{1}{\sqrt{N}}\sum_{y=0}^{r-1}\left(\sum_{t=0}^{N/r-1}\left|tr+y\right\rangle\right)\otimes\left|F(y)\right\rangle}$$

“Measure” the rightmost register; assume outcome “F(c)” with $0\leq c<r$ [Cf. Handout 3.]…

Periodicity Algorithm (2)

… this yields the superposition for the left register:

$$\boxed{\sqrt{\frac{r}{N}}\sum_{t=0}^{N/r-1}\left|tr+c\right\rangle}$$

Apply the Fourier transform over $\mathbb{Z}_N$, giving:

$$\boxed{\frac{\sqrt{r}}{N}\sum_{t=0}^{N/r-1}\sum_{j=0}^{N-1}\zeta_N^{j(tr+c)}\left|j\right\rangle = \frac{\sqrt{r}}{N}\sum_{j=0}^{N-1}\zeta_N^{jc}\left(\sum_{t=0}^{N/r-1}\zeta_N^{jtr}\right)\left|j\right\rangle}$$

If j *not* a multiple of N/r, then destructive interference

If j multiple of N/r, then constructive interference

Periodicity Algorithm (3)

Calculating the j-dependent interference:

$$\boxed{\sum_{t=0}^{N/r-1} \zeta_N^{jtr} \approx \frac{N}{r} \quad \text{if } jr \text{ is a multiple of } N}$$

$$\boxed{\sum_{t=0}^{N/r-1} \zeta_N^{jtr} \approx \frac{\zeta_N^{jN}-1}{\zeta_N^{jr}-1} = 0 \quad \text{if } jr \text{ is not a multiple of } N}$$

Hence we have the output state:

$$\boxed{\frac{\sqrt{r}}{N}\sum_{j=0}^{N-1} \zeta_N^{jc}\left(\sum_{t=0}^{N/r-1} \zeta_N^{jtr}\right)\left|j\right\rangle \quad \approx \quad \frac{1}{\sqrt{r}}\sum_{k=0}^{r-1} \zeta_N^{ckN/r}\left|k\cdot N/r\right\rangle}$$

Periodicity Algorithm (4)

With very high probability we will measure a multiple of N/r, where r is the period of the function.

By repeating the procedure several times, we obtain enough information to determine N/r and hence r. (This is not entirely trivial and requires the usage of the “continued fractions method”, but it can be done.)

Being able to find the (hidden) period of a function allows us to solve factoring, discrete logarithms and other (presumed) hard problems.