Today's outline - February 21, 2023

- Fast Fourier transform
- Quantum Fourier Transform
- Overview of Shor's algorithm

Reading Assignment: Reiffel: 8.3–8.4 Wong: 7.10

Exam #1 Tuesday, February 28, 2023
Covers Chapters HW 01–04

Homework Assignment #05:
due Thursday, March 02, 2023

Discrete Fourier transform

The quantum Fourier transform is an important building block for many quantum algorithms

In order to develop the efficient implementation of the quantum Fourier transform, it is useful to start with the classical discrete and fast Fourier transforms

The discrete Fourier transform (DFT) is a linear transformation which takes a discrete column vector $a(k)$ to a column vector of Fourier coefficients, $A(x)$, where $0 \leq k, x \leq N-1$

$$A(x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a(k) e^{2\pi i k x / N}$$

The DFT operator is an $N \times N$ matrix with elements

$$F_{xk} = \frac{1}{\sqrt{N}} e^{2\pi i k x / N}$$

Assume $a(k) = e^{-2\pi i u k / N}$ is a function of frequency $u < N$ which evenly divides N

Computing the Fourier coefficients,

$$A(x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a(k) e^{2\pi i k x / N} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-2\pi i u k / N} e^{2\pi i k x / N} = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i k (x-u) / N}$$

All are zero except for when $x - u = 0 \mod N$ so the only term which survives is $A(u)$

DFT examples

Start with the definition of the DFT and suppose that $a = \{1, 2, 3, 4\}$, $n = 2$, and $N = 4$

$$A_x = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a_k e^{2\pi i k x / N}$$

$$A_0 = \tfrac{1}{2}\left(a_0 + a_1 + a_2 + a_3\right) = \tfrac{1}{2}(1 + 2 + 3 + 4) = 5$$

$$A_1 = \tfrac{1}{2}\left(a_0 + a_1 e^{i\pi/2} + a_2 e^{i\pi} + a_3 e^{i3\pi/2}\right) = \tfrac{1}{2}(1 + 2i - 3 - 4i) = \tfrac{1}{2}(-2 - 2i) = -(1 + i)$$

$$A_2 = \tfrac{1}{2}\left(a_0 + a_1 e^{i\pi} + a_2 e^{i2\pi} + a_3 e^{i3\pi}\right) = (1 - 2 + 3 - 4) = -1$$

$$A_3 = \tfrac{1}{2}\left(a_0 + a_1 e^{i3\pi/2} + a_2 e^{i3\pi} + a_3 e^{i9\pi/2}\right) = \tfrac{1}{2}(1 - 2i - 3 + 4i) = -(1 - i)$$

But if $u = 2$ and $a_k = e^{-2\pi i u k / N}$, we have

$$A_0 = \tfrac{1}{2}\left(1 + e^{-i\pi} + e^{-i2\pi} + e^{-i3\pi}\right) = (1 - 1 + 1 - 1) = 0$$

$$A_1 = \tfrac{1}{2}\left(1 + e^{-i\pi} e^{i\pi/2} + e^{-i2\pi} e^{i\pi} + e^{-i3\pi} e^{i3\pi/2}\right) = \tfrac{1}{2}(1 - i - 1 + i) = 0$$

$$A_2 = \tfrac{1}{2}\left(1 + e^{-i\pi} e^{i\pi} + e^{-i2\pi} e^{i2\pi} + e^{-i3\pi} e^{i3\pi}\right) = \tfrac{1}{2}(1 + 1 + 1 + 1 + 1) = 2$$

$$A_3 = \tfrac{1}{2}\left(1 + e^{-i\pi} e^{i3\pi/2} + e^{-i2\pi} e^{i3\pi} + e^{-i3\pi} e^{i9\pi/2}\right) = \tfrac{1}{2}(1 - i - 1 + i) = 0$$

DFT examples (cont.)

Suppose instead that $a = \{0, 2, 4, 2, 0, 2, 4, 2\}$, $n = 3$, and $N = 8$

$$A_x = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a_k e^{2\pi i k x / N}$$

$$A_0 = \tfrac{1}{\sqrt{8}}(0 + 2 + 4 + 2 + 0 + 2 + 4 + 2) = \tfrac{16}{\sqrt{8}} = 2\sqrt{8}$$

$$A_1 = \tfrac{1}{\sqrt{8}}\left[0 + 2e^{i\pi/4} + 4e^{i\pi/2} + 2e^{i3\pi/4} + 0e^{i\pi} + 2e^{i5\pi/4} + 4e^{i3\pi/2} + 2e^{i7\pi/4}\right]$$
$$= \tfrac{1}{\sqrt{8}}\left[(\sqrt{2} + i\sqrt{2}) + 4i + (-\sqrt{2} + i\sqrt{2}) + (-\sqrt{2} - i\sqrt{2}) - 4i + (\sqrt{2} - i\sqrt{2})\right] = 0$$

$$A_2 = \tfrac{1}{\sqrt{8}}\left[0 + 2e^{i\pi/2} + 4e^{i\pi} + 2e^{i3\pi/2} + 0e^{i2\pi} + 2e^{i5\pi/2} + 4e^{i3\pi} + 2e^{i7\pi/2}\right]$$
$$= \tfrac{1}{\sqrt{8}}\left[2\cancel{i} - 4 - 2\cancel{i} + 2\cancel{i} - 4 - 2\cancel{i}\right] = -\tfrac{8}{\sqrt{8}} = -\sqrt{8}$$

$$A_3 = \tfrac{1}{\sqrt{8}}\left[0 + 2e^{i3\pi/4} + 4e^{i3\pi/2} + 2e^{i9\pi/4} + 0e^{i3\pi} + 2e^{i15\pi/4} + 4e^{i9\pi/2} + 2e^{i21\pi/4}\right]$$
$$= \tfrac{1}{\sqrt{8}}\left[(-\sqrt{2} + i\sqrt{2}) - 4i + (\sqrt{2} + i\sqrt{2}) + (\sqrt{2} - i\sqrt{2}) + 4i + (-\sqrt{2} - i\sqrt{2})\right] = 0$$

DFT examples (cont.)

$$A_4 = \frac{1}{\sqrt{8}}\left[0 + 2e^{i\pi} + 4e^{i2\pi} + 2e^{i3\pi} + 0e^{i4\pi} + 2e^{i5\pi} + 4e^{i6\pi} + 2e^{i7\pi}\right]$$
$$= [-2 + 4 - 2 - 2 + 4 - 2] = 0$$

$$A_5 = \frac{1}{\sqrt{8}}\left[0 + 2e^{i5\pi/4} + 4e^{i5\pi/2} + 2e^{i15\pi/4} + 0e^{i5\pi} + 2e^{i25\pi/4} + 4e^{i15\pi/2} + 2e^{i35\pi/4}\right]$$
$$= \frac{1}{\sqrt{8}}\left[(\sqrt{2} + i\sqrt{2}) + 4i + (\sqrt{2} - i\sqrt{2}) + (-\sqrt{2} - i\sqrt{2}) - 4i + (-\sqrt{2} + i\sqrt{2})\right] = 0$$

$$A_6 = \frac{1}{\sqrt{8}}\left[0 + 2e^{i3\pi/2} + 4e^{i3\pi} + 2e^{i9\pi/2} + 0e^{i6\pi} + 2e^{i15\pi/2} + 4e^{i9\pi} + 2e^{i21\pi/2}\right]$$
$$= \frac{1}{\sqrt{8}}\left[-2i - 4 + 2i - 2i - 4 + 2i\right] = -\frac{8}{\sqrt{8}} = -\sqrt{8}$$

$$A_7 = \frac{1}{\sqrt{8}}\left[0 + 2e^{i7\pi/4} + 4e^{i7\pi/2} + 2e^{i21\pi/4} + 0e^{i7\pi} + 2e^{i35\pi/4} + 4e^{i21\pi/2} + 2e^{i49\pi/4}\right]$$
$$= \frac{1}{\sqrt{8}}\left[(\sqrt{2} - i\sqrt{2}) - 4i + (-\sqrt{2} - i\sqrt{2}) + (+\sqrt{2} - i\sqrt{2}) + 4i + (+\sqrt{2} + i\sqrt{2})\right] = 0$$

Fast Fourier transform

The fast Fourier transform (FFT) is an efficient implementation for $N = 2^n$, called $F^{(n)}$

The implementation involves recursive decomposition of $F^{(n)}$ in terms of Fourier transforms of lower powers of 2

If $\omega_{(n)} = e^{2\pi i/N}$ is the N^{th} root of unity, the elements of the Fourier transform matrix are

$$F^{(n)}_{xy} = \omega^{xy}_{(n)}$$

where $x, y \in \{0, \ldots, N-1\}$

Let $F^{(k)}$, $I^{(k)}$, $R^{(k)}$ be the 2^k-dimensional Fourier transform matrix, identity matrix, and a permutation matrix defined by

$$R^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

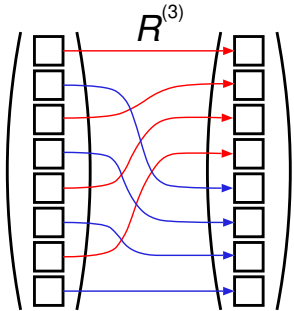
$$R^{(k)}_{xy} = \begin{cases} 1 & \text{for } 2x = y \\ 1 & \text{for } 2x - 2^k + 1 = y \\ 0 & \text{otherwise} \end{cases}$$

$i = 0; \quad y = 2x = 0$

Fast Fourier transform

The $R^{(k)}$ matrix performs a shuffle transform on a column vector

Using $F^{(k)}$, $I^{(k)}$, $R^{(k)}$ plus $D^{(k)}$, a diagonal matrix with entries $\omega^0_{(k+1)}, \ldots, \omega^{2^k-1}_{(k+1)}$ it is possible to solve for $F^{(k)}$ recursively



$$F^{(k)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k-1)} & D^{(k-1)} \\ I^{(k-1)} & -D^{(k-1)} \end{pmatrix} \begin{pmatrix} F^{(k-1)} & 0 \\ 0 & F^{(k-1)} \end{pmatrix} R^{(k)}$$

The $R^{(k)}$ operator serves to reorder the data vector into odd and even elements and multiplication by block diagonal matrices is very efficient

Computing $F^{(k)}$ becomes computing 2 $F^{(k-1)}$, then 4 $F^{(k-2)}$, and so on until 2^{k-1} $F^{(1)}$ matrices and $O(nN)$ computation

Quantum Fourier transform

The quantum Fourier transform (QFT) like the FFT assumes $N = 2^n$ and that the amplitudes $a_x \equiv a(x)$ of the superposition state $|\psi\rangle$ are the function to be transformed

$$|\psi\rangle = \sum_{x=0}^{N-1} a(x)|x\rangle \longrightarrow \sum_{x=0}^{N-1} A(x)|x\rangle$$

The QFT does not require an output register as the output quantum state contains the Fourier transform in its complex amplitudes

If the initial state is such that the amplitudes are a periodic function with period $r = 2^m$, the resultant Ax would be zero unless $x = j\frac{N}{r}$ with $j = 0, 1, \ldots, \frac{N}{r} - 1$

When $r \neq 2^m$, the QFT will produce an approximate solution with higher probability coefficients for states with integers near multiples of $\frac{N}{r}$

As the base for the QFT, $N = 2^n$ is increased, the approximation improves

The QFT is exponentially faster, $[O(n^2)]$, than the discrete $[O(N^2)]$, and the fast $[O(N \log N)]$ transforms

Quantum Fourier transform

For $N = 2^n$, the quantum Fourier transform acting on $|k\rangle$ is defined as

$$U_F^{(n)}|k\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i k x/N}|x\rangle$$

For $N = 1$ the quantum Fourier transform is identical to the Hadamard transform

$$U_F^{(1)}|0\rangle = \frac{1}{\sqrt{2}} \sum_{x=0}^{1} e^{0}|x\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \qquad U_F^{(1)}|1\rangle = \frac{1}{\sqrt{2}} \sum_{x=0}^{1} e^{\pi i x}|x\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The recursive decomposition of the fast Fourier transform is used to compute the $N = 2^n$ transform

$$U_F^{(k+1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix} \begin{pmatrix} U_F^{(k)} & 0 \\ 0 & U_F^{(k)} \end{pmatrix} R^{(k+1)}$$

All matrices are unitary and can be implemented efficiently on a quantum computer

QFT implementation

$$U_F^{(k+1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix} \begin{pmatrix} U_F^{(k)} & 0 \\ 0 & U_F^{(k)} \end{pmatrix} R^{(k+1)}$$

The implementation starts with the rotation $R^{(k+1)}$

$$R^{(k+1)} = \sum_{i=0}^{2^k-1} |i\rangle\langle 2i| + |i+2^k\rangle\langle 2i+1|$$

This can be accomplished by a permutation of the $k+1$ qubits, resulting in just the kind of shuffling needed

Only $k-1$ swap operations are needed to perform this permutation

Next is the QFT transformation matrix U_F^{k+1}, which is implemented by recursively applying the QFT to qubits 0 to k

$$\begin{array}{ccc} 000 & \rightarrow & 000 \\ 001 & \rightarrow & 010 \\ 010 & \rightarrow & 100 \\ 011 & \rightarrow & 110 \\ 100 & \rightarrow & 001 \\ 101 & \rightarrow & 011 \\ 110 & \rightarrow & 101 \\ 111 & \rightarrow & 111 \end{array}$$

$$\begin{pmatrix} U_F^{(k)} & 0 \\ 0 & U_F^{(k)} \end{pmatrix} = I \otimes U_F^{(k)}$$

QFT implementation

$$U_F^{(k+1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix} \begin{pmatrix} U_F^{(k)} & 0 \\ 0 & U_F^{(k)} \end{pmatrix} R^{(k+1)}$$

The diagonal matrix of phase shifts, $D^{(k)}$ is decomposed recursively, where $\omega_{(k+1)} = e^{2\pi i/2^{k+1}}$

$$D^{(k)} = D^{(k-1)} \otimes \begin{pmatrix} 1 & 0 \\ 0 & \omega_{(k+1)} \end{pmatrix}$$

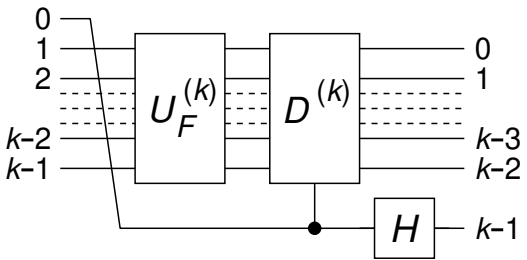
This recursive decomposition applies a phase rotation of ω_{j+1} to the j^{th} qubit for $1 \leq j \leq k$ and can be implemented using k single-qubit gates

Thus only k gates are necessary for the implementation of the controlled $D^{(k)}$

$$\begin{aligned} \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix} &= \tfrac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle\right) \langle 0| \otimes I^{(k)} + \tfrac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle\right) \langle 1| \otimes D^{(k)} \\ &= \left(H|0\rangle\langle 0|\right) \otimes I^{(k)} + \left(H|1\rangle\langle 1|\right) \otimes D^{(k)} \\ &= \left(H \otimes I^{(k)}\right) \left(|0\rangle\langle 0| \otimes I^{(k)} + |1\rangle\langle 1| \otimes D^{(k)}\right) \end{aligned}$$

QFT implementation

One possible recursive circuit for QFT is



The recursive circuit for $U_F^{(k+1)}$ can be implemented as

define $QFT|x[1]\rangle = H|x\rangle$

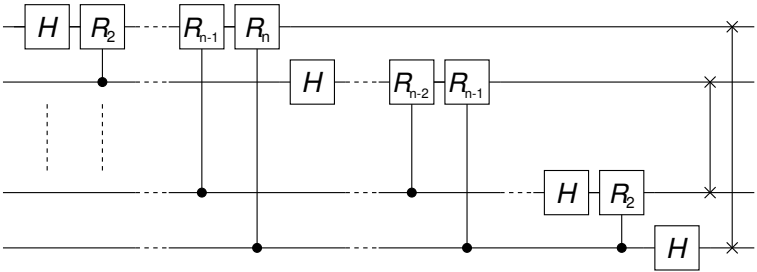
$QFT|x[n]\rangle =$

1. $Swap|x_0\rangle|x_1 \cdots x_{n-1}\rangle$
2. $QFT|x_0 \cdots x_{n-2}\rangle$
3. $|x_{n-1}\rangle$ **control** $D^{(n-1)}|x_0 \cdots x_{n-2}\rangle$
4. $H|x_{n-1}\rangle$

$D^{(k)}$ and $R^{(k)}$ can be implemented with $O(k)$ gates and the k^{th} step in the recursion adds $O(K)$ gates to the implementation of $U_F^{(n)}$, overall $U_F^{(n)}$ takes $O(n^2)$ gates to implement which exponentially faster than the $O(n2^n)$ for a classical FFT

QFT example

A more straightforward circuit provides insight to the recursive one just described



$$R_n = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^n} \end{pmatrix}$$

Starting with the high order qubit at the top, the Hadamard transform is followed by controlled rotations from each of the other $N-1$ qubits

The next qubit is transformed the same way using the $N-2$ lower order qubits, and so on until the last qubit which only has a Hadamard gate applied

At the end, all the qubits need to be swapped to recover the proper order

Quirk implementations

Recalling that with Quirk, the top qubit is the lowest order qubit a 6-qubit Fourier Transform is

https://tinyurl.com/mr3k997a

A more compact treatment performs the swap first

https://tinyurl.com/3nf5aax5

Classical period-finding

In 1994 Shor developed an algorithm for factoring integers which coupled with the quantum Fourier transform threatened to crack the standard encryption algorithms of the time

The factoring algorithm relies finding the period of a function $f(k)$

The order of an integer $a \bmod M$ is the smallest integer r such that $r > 0$ and $a^r = 1 \bmod M$

If the two integers a and M are relatively prime (i.e. they share no prime factors) then r exists and the order of a is finite

Consider the function $f(k)$
Since $a^r = 1 \bmod M$ we can write and r is the period of $f(k)$

$$f(k) = a^k \bmod M = a^{k+r} \bmod M$$

For example, take $a = 5$ and $M = 13$

r	a^r	$a^r \bmod M$
1	5	5
2	25	12
3	125	8
4	625	1

Thus $r = 4$ is the period of the function $f(k) = a^k = 5^k$