CPEN 400Q / EECE 571Q Lecture 10
The quantum Fourier transform

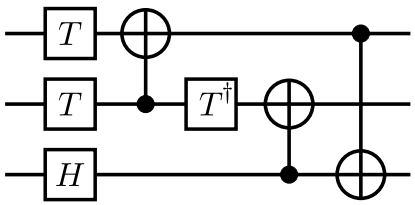
Thursday 10 February 2022

Announcements

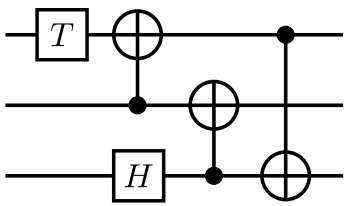
- Assignment 2 due Monday 14 Feb 23:59
- Project group / topic selection due Tuesday
- PennyLane v0.21 released this week: we will switch Tuesday, *after* Assignment 2 is due

Last time

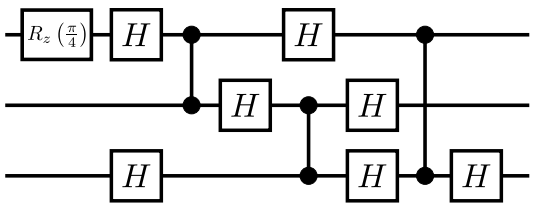
We saw an overview of the quantum compilation process.



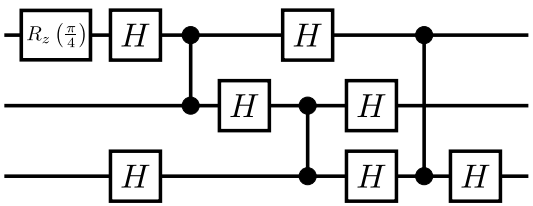
Synthesis



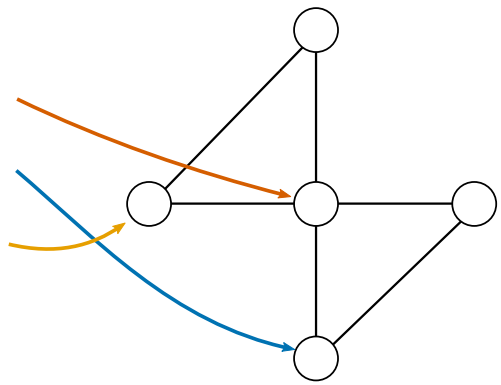
Optimization

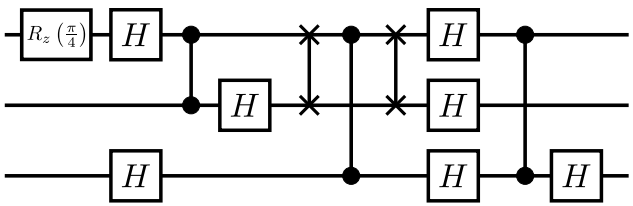


Transpilation



Placement





Routing

Last time

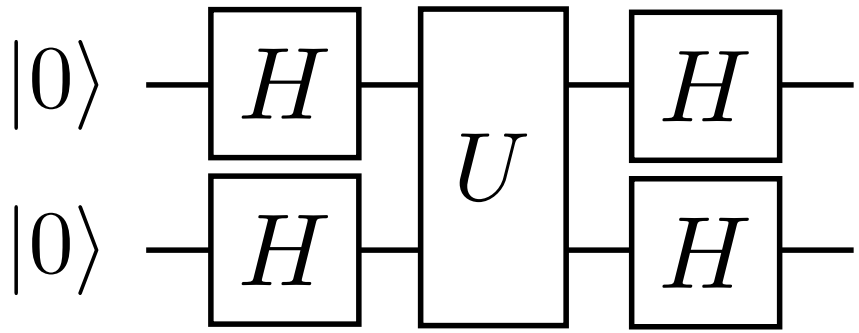
We wrote and applied quantum transforms in PennyLane to manipulate circuits.

```
@qml.qfunc_transform
def x_to_hzh(tape):
    for op in tape.operations:
        if op.name == ``PauliX''
            qml.Hadamard(wires=op.wires)
            qml.PauliZ(wires=op.wires)
            qml.Hadamard(wires=op.wires)
        else:
            qml.apply(op)

    for m im tape.measurements:
        qml.apply(op)
```

Quiz 4 sample solution

Synthesize U, then fuse all single-qubit gates.




```
@qml.qnode(dev)
@qml.transforms.single_qubit_fusion()
@qml.transforms.unitary_to_rot
def qnode():
    qml.Hadamard(wires=0)
    qml.Hadamard(wires=1)
    qml.QubitUnitary(two_qubit_unitary, wires=[0, 1])
    qml.Hadamard(wires=0)
    qml.Hadamard(wires=1)
    return qml.state()
```

Learning outcomes

- Express floating-point values in fractional binary representation
- Describe the behaviour of the quantum Fourier transform
- Implement the quantum Fourier transform in PennyLane

Today there will be lots of MATH.

Motivation

Over the next few lectures, we are going to build up to an implementation of **Shor's algorithm**, which can factor numbers efficiently on a quantum computer.

Shor's algorithm performs a period finding step which requries a subroutine called **quantum phase estimation** (QPE).

QPE requires a subroutine called the **quantum Fourier transform**.

Fourier series

Often, we express functions in a *polynomial basis* using different powers of x:

If our function has some special properties, this is not the only choice of basis.

Fourier series

If a function is 2π-periodic[1], we can represent it as a **Fourier series** using sin and cos as basis functions:

How are sin and cos basis functions??

[1] More formally, it must satisfy *Dirichlet conditions*: single-valued, finite number of optima, finite number of discontinuities, $\int_{-\pi}^{\pi} |f(x)|dx < \infty$.

sin and cos are orthogonal w.r.t. an "inner product" computed by integration over a period.

There are similar relationships between two sin or two cos:

	$m \neq n$	$m = n \neq 0$	$m = n = 0$
$\frac{1}{2\pi} \int_{-\pi}^{\pi} \sin mx \sin nx dx$	0	0.5	0
$\frac{1}{2\pi} \int_{-\pi}^{\pi} \cos mx \cos nx dx$	0	0.5	1

This fact can be used to compute the values of individual coefficients...

Example: compute a_3

$$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}(a_n \cos(nx) + b_n \sin(nx))$$

Multiply both sides by the relevant basis function:

Then integrate...

Fourier series

Discrete Fourier series

We can also represent sin and cos as complex exponentials:

Then, we can re-write our function as a Fourier series in an even more compact way.

Fourier series

Writing

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$$

expresses $f(x)$ in the basis of complex exponential functions.

Clearer to see how these are orthogonal... check value of

Case 1: $m = n$

Case 2: $m \neq n$

Converting to the Fourier basis

We can write functions in two different bases...

In our study of quantum computing so far:

- we are working in finite-sized spaces
- we've talked about how we can perform *basis rotations* to convert between orthonormal bases

Can we do the same here?

The discrete Fourier transform

The discrete Fourier transform (DFT) does the job:

$$DFT = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \bar{\omega} & \bar{\omega}^2 & \cdots & \bar{\omega}^{N-1} \\ 1 & \bar{\omega}^2 & \bar{\omega}^4 & \cdots & \bar{\omega}^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \bar{\omega}^{N-1} & \bar{\omega}^{2(N-1)} & \cdots & \bar{\omega}^{(N-1)(N-1)} \end{pmatrix}$$

where $\bar{\omega} = e^{-2\pi i/N}$.

Note: sometimes there is no prefactor, or a prefactor of $1/N$; depends how inverse is defined.

The discrete Fourier transform

The DFT (and the fast Fourier transform which implements it efficiently) are standard tools in digital signal processing to convert between time and frequency domain.

Given a signal $x[n]$ in the time domain, the DFT computes

The discrete Fourier transform

The inverse DFT computes

where $\omega = e^{2\pi i/N} = \bar{\omega}^{-1}$

The DFT matrix is invertible; its matrix is also unitary (possibly up to a prefactor). Seems like a good candidate for a quantum computer...

Quantum Fourier transform

The quantum Fourier transform (QFT) is the quantum analog of the **inverse DFT**.

Let $|x\rangle$ be an n-qubit computational basis state, $N = 2^n$.

The QFT sends

We are sending individual computational basis states to another basis, which is made up of linear combinations of computational basis states with complex exponential coefficients.

Quantum Fourier transform

The QFT is a unitary operation with the following action on the basis states:

Check that this works...

Quantum Fourier transform

As a matrix, it looks a lot like the DFT:

$$QFT = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

But... can we implement this unitary efficiently? How do we *synthesize* a circuit for it? It looks like it would be very messy.

Quantum Fourier transform

Let's start with some special cases... suppose $n = 1$ ($N = 2$).

Here, $e^{2\pi i/2} = e^{i\pi} = -1$, so

Look familiar?

Quantum Fourier transform

Suppose $n = 2$ ($N = 4$).

Here $\omega = i$, and $\omega^2 = -1$, $\omega^4 = 1$. So

Not so familiar.

Quantum Fourier transform

But, if we apply a SWAP operation, familiar things show up...

$$QFT = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \\ 1 & -i & -1 & i \end{pmatrix}$$

The top blocks are H, and the bottom are HS. Can show that the following circuit implements this QFT:

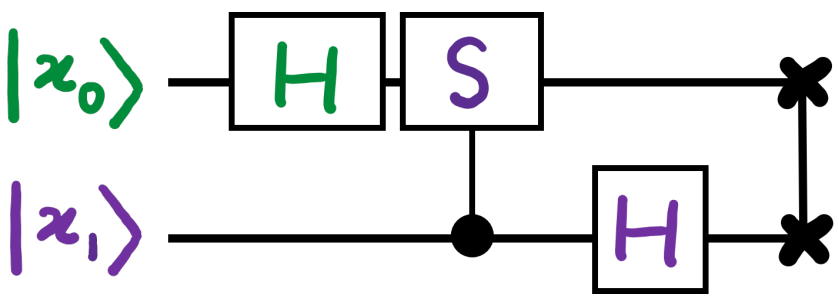
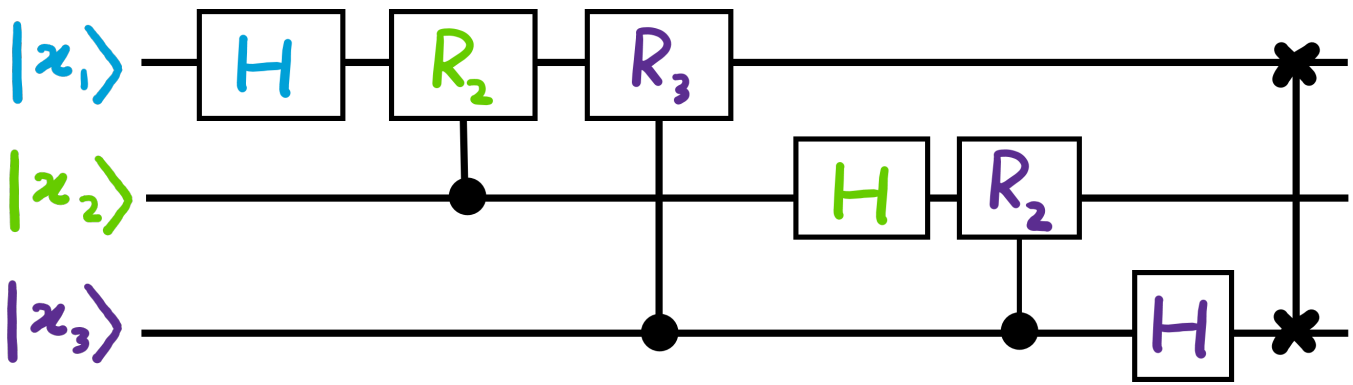


Image credit: Xanadu Quantum Codebook node F.2

Quantum Fourier transform

We can do the same for $n = 3$ ($N = 8$) but now things are getting nasty... but you can show that the structure of the circuit that implements it is

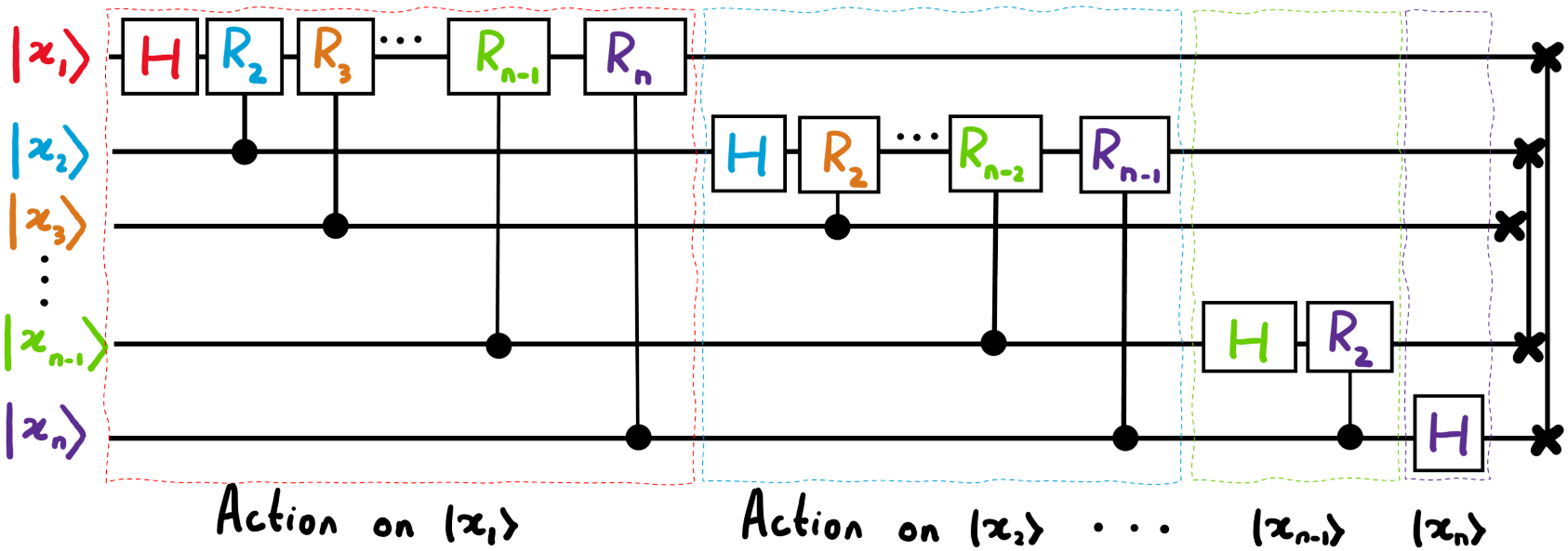


Here, $R_2 = S$ and $R_3 = T$.

Image credit: Xanadu Quantum Codebook node F.3

Quantum Fourier transform

This is what the circuit looks like in general:



Let's understand why this works.

Image credit: Xanadu Quantum Codebook node F.3

A circuit for the QFT

Consider the expression

$$|x\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{xk} |k\rangle$$

Here x and k are represented as integers.

They are n-qubit computational basis states so they also have binary equivalents $|x\rangle = |x_1 \cdots x_n\rangle$, $|k\rangle = |k_1 \cdots k_n\rangle$:

and similarly for k.

A circuit for the QFT

Recall that $\omega = e^{2\pi i/N}$.

We are working with

$$\omega^{xk} = e^{2\pi i x(k/N)}$$

with $N = 2^n$.

We can write a fraction $k/2^n$ in a 'decimal version' of binary:

Binary notation for decimal numbers

Example: let $k = 0.11010$.

The numerical value of this is:

This seems like a very convoluted way to write decimal numbers; this is going to become **very** hideous but I promise it is going somewhere.

A circuit for the QFT

Using the fractional decimal expression for k/N, we will work through the specification of the Fourier transform and see how we can reshuffle and *factor* the output state to get something that will make clear a circuit. Brace yourselves.

A circuit for the QFT

(keeping the last equation from the previous slide)

A circuit for the QFT

(keeping the last equation from the previous slide)

A circuit for the QFT

So...

$$|x\rangle \rightarrow \frac{\left(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.x_1\cdots x_n}|1\rangle\right)}{\sqrt{N}}$$

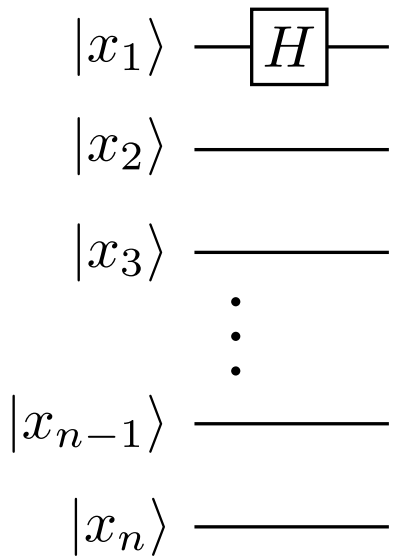
Believe it or not, this form reveals to us how we can design a circuit that creates this state!

A circuit for the QFT

Starting with the state

$$|x\rangle = |x_1 \cdots x_n\rangle,$$

apply a Hadamard to qubit 1:

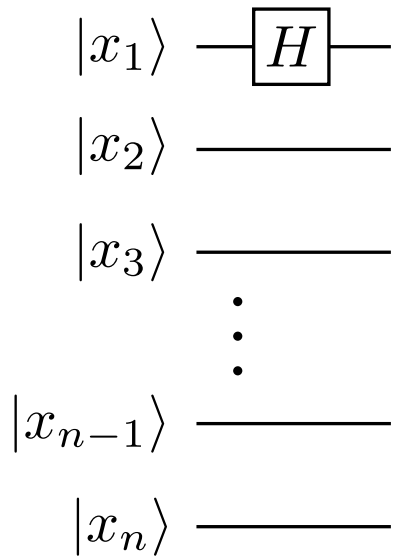


A circuit for the QFT

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1}|1\rangle\right)|x_2 \cdots x_n\rangle$$

If $x_1 = 0$, $e^0 = 1$ and we get the $|+\rangle$ state.

If $x_1 = 1$, $e^{2\pi i(1/2)} = e^{\pi i} = -1$ and we get the $|-\rangle$ state.



A circuit for the QFT

We are trying to make a state that looks like this:

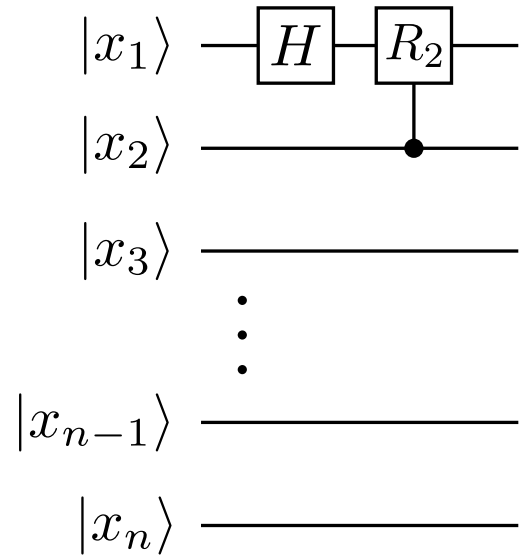
$$|x\rangle \rightarrow \frac{\left(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.x_1\cdots x_n}|1\rangle\right)}{\sqrt{N}}$$

Every qubit has a different *phase* on the $|1\rangle$ state. We are going to need some way of creating this.

We define the gate:

A circuit for the QFT

Now let's apply a controlled R_2 gate from qubit 2 to qubit 1

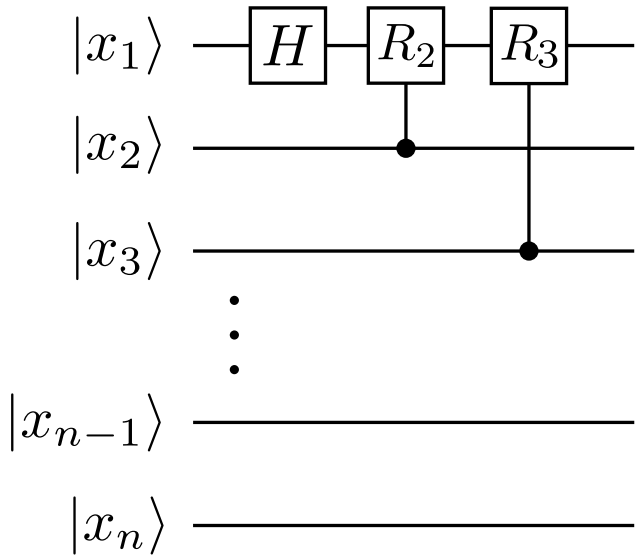


The first qubit picks up a phase:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1}|1\rangle\right)|x_2 \cdots x_n\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1 x_2}|1\rangle\right)|x_2 \cdots x_n\rangle$$

A circuit for the QFT

Now let's apply a controlled R_3 gate from qubit 3 to qubit 1



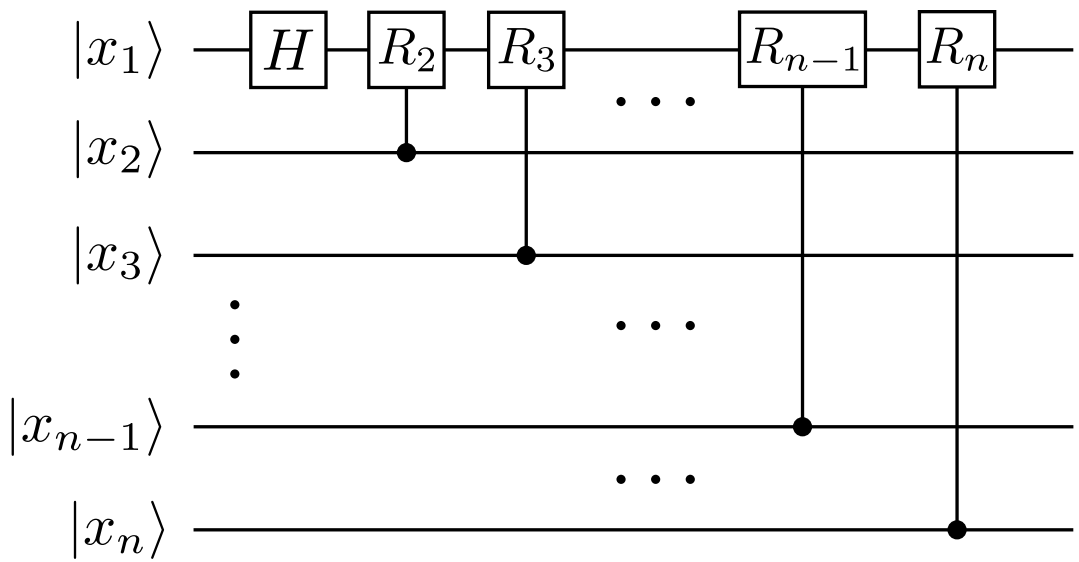
The first qubit picks up another phase:

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1x_2}|1\rangle\right)|x_2\cdots x_n\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1x_2x_3}|1\rangle\right)|x_2\cdots x_n\rangle$$

A circuit for the QFT

We can apply a controlled R_4 from the fourth qubit, etc. up to the n-th qubit to get

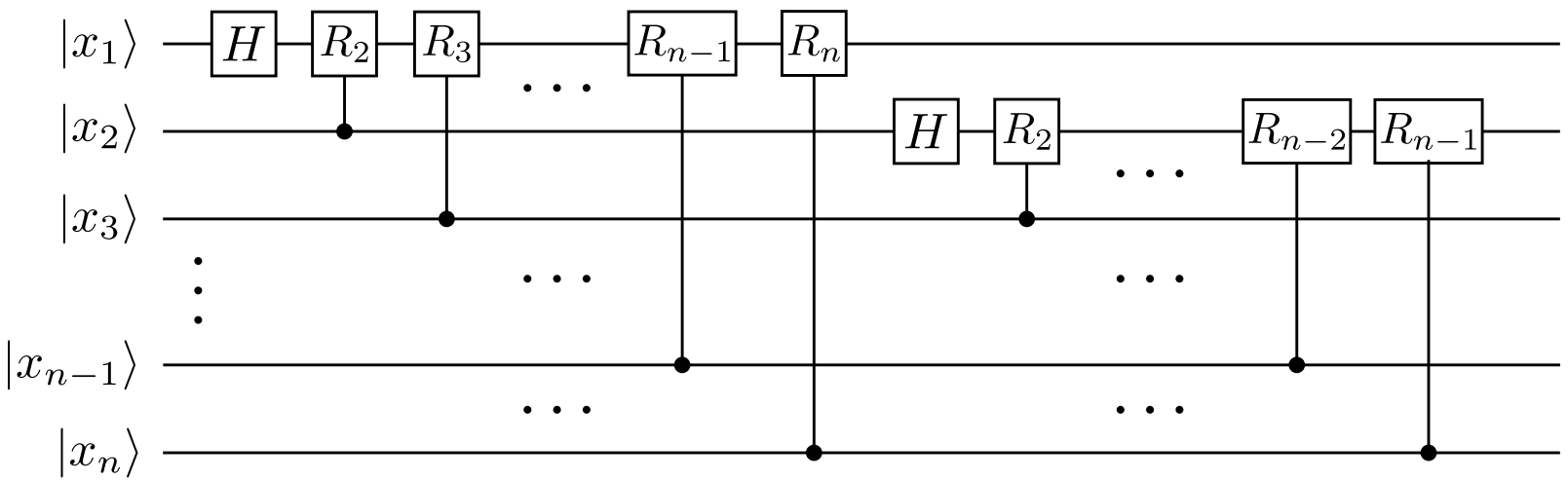
$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.x_1 x_2 \cdots x_n}|1\rangle\right)|x_2 \cdots x_n\rangle$$



A circuit for the QFT

Next, ignore the first qubit and do the same thing with the second qubit: apply H, and then controlled rotations from every qubit from 3 to n to get

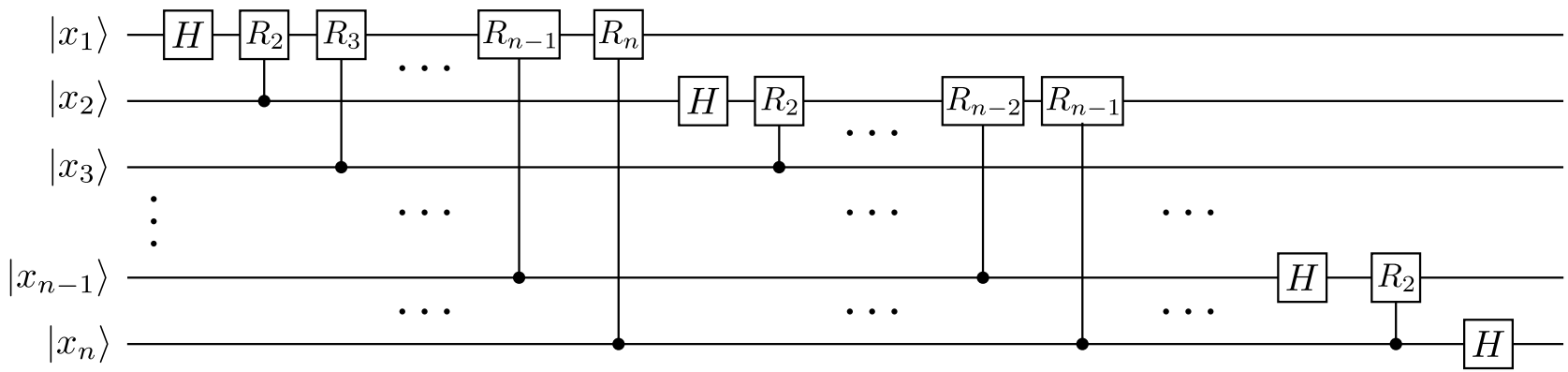
$$\frac{1}{\sqrt{2}^2}\left(|0\rangle + e^{2\pi i 0.x_1x_2\cdots x_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.x_2\cdots x_n}|1\rangle\right)|x_3\cdots x_n\rangle$$



A circuit for the QFT

If we do this for all qubits, we eventually get that big ugly state from earlier:

$$|x\rangle \rightarrow \frac{\left(|0\rangle + e^{2\pi i 0.x_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.x_{n-1}x_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.x_1\cdots x_n}|1\rangle\right)}{\sqrt{N}}$$

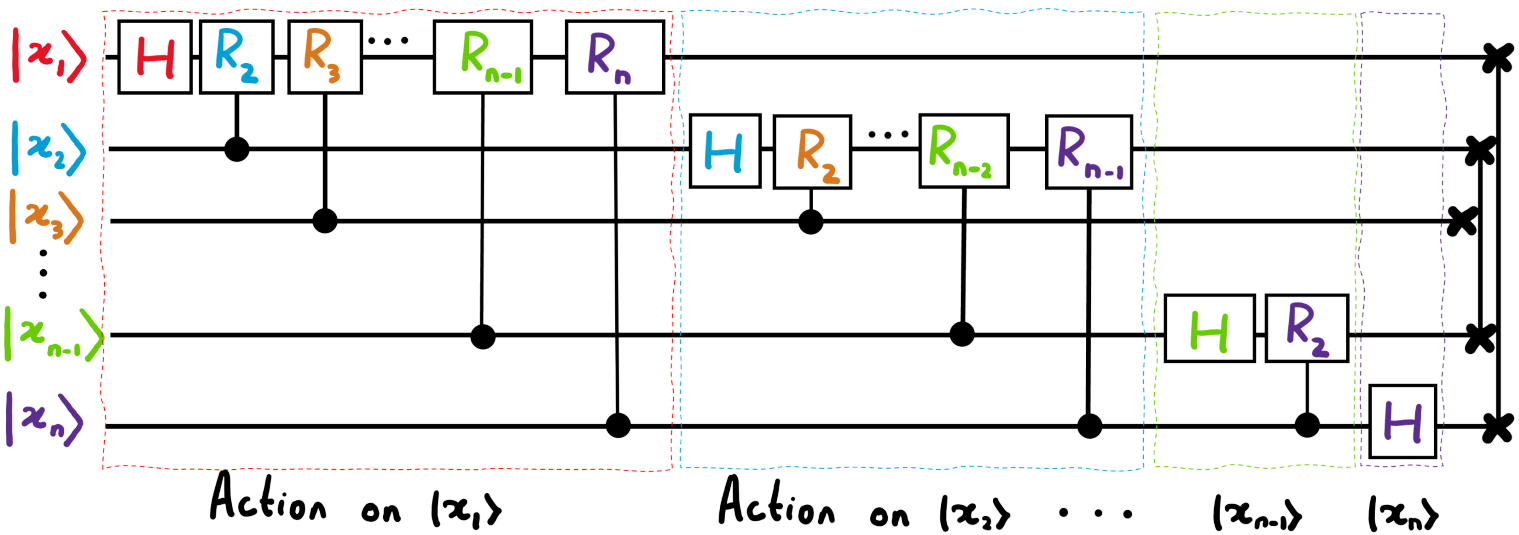


(though note that the order of the qubits is backwards - this is easily fixed with some SWAP gates)

Quantum Fourier transform

So the QFT can be implemented using:

- n Hadamard gates
- $n(n-1)/2$ controlled rotations
- $\lfloor n/2 \rfloor$ SWAP gates if you care about the order



The number of gates is *polynomial in n*, so this can be implemented efficiently on a quantum computer! Let's try it...

Image credit: Xanadu Quantum Codebook node F.3

Next time

Content:

- Quantum phase estimation

Action items:

1. Finish Assignment 2 (due Monday 23:59)
2. E-mail me your project team and paper selection by Tuesday

Recommended reading:

- Codebook nodes F.1-F.3
- Nielsen & Chuang 5.1