Resources: IBM and qiskit

- https://qiskit.org/
- https://qiskit.org/aqua
- https://quantumexperience.ng.bluemix.net/qx/editor
Resources: cirq and Forest

Cirq

- https://github.com/quantumlib/Cirq

Forest

- https://github.com/rigetti/pyquil
- pyquil.readthedocs.io/en/latest

See

https://quantum-computing.ibm.com/support
Example: The Deutsch-Jozsa algorithm

- One of the first algorithms to demonstrate usefulness of QC

Problem: given a function \( f \) from \( \{0, 1\} \) to itself determine whether \( f \) is a constant function.

- The function is constant when \( f(x) \equiv 0 \ \forall x \) or \( f(x) \equiv 1 \ \forall x \) (\( \forall = \) for all). It is balanced otherwise.

- Here are all possible 2-bit functions:
  - Constant: \( f_0, f_1 \), balanced: \( f_x, f_{\bar{x}} \)

- Normally we need 2 evaluations to solve the problem [one eval. = querying one qubit]

- Can do it with one - with quantum computing

- For \( n \) bit functions would classically need \( n \) evals. QC: one
The Deutsch-Jozsa algorithm

First: \( f \) is not injective - so cannot tell \( x \) from \( f(x) \). It is not reversible. Make it reversible with a trick

Define 'Oracle':

\[
U_f(|x\rangle|y\rangle) := |x\rangle|y \oplus f(x)\rangle
\]

Here: \( \oplus \) == addition mod 2 == XOR operation

Show that \( U_f \circ U_f = I \) (where: \( \circ \) = composition)

From above exercise we see that \( U_f \) is now reversible (even though \( f \) may not be)

Consider \( U_f \) as a function of the 2 qubits \( x \) and \( y \)

Show that when \( f = f_0 \) then \( U_f \) is the identity

Show: when \( f = f_1 \) then \( U_f \) does an XOR on the 2nd qubit
When \( f = f_x \) then \( U_f \) does the CNOT operation:

**Case** \( f = f_x \)

Control = \( x \), Target = \( y \)

\[
U_f(\left| xy \right>) = \left| 00 \right> \left| 01 \right> \left| 10 \right> \left| 11 \right>
\]

When \( f = f \bar{x} \) then \( U_f \) does the operation:

**Case** \( f = f \bar{x} \)

\[
U_f(\left| xy \right>) = \left| 01 \right> \left| 00 \right> \left| 10 \right> \left| 11 \right>
\]

Note: all second bits are flipped from case \( f_x \) above - therefore:

- This is a CNOT operation followed by a NOT (X) on 2nd qubit.

Show that for a given \( f \), \( U_f \) (a 2 qubit operator) is linear and that it is unitary. What is its matrix representation for each of the 4 functions \( f_0, f_1, f_x, f \bar{x} \)?

- Deutsch-Jozsa algorithm based on exploiting superposed states

- Take second qubit as \( \left| - \right> = \frac{1}{\sqrt{2}} \left( \left| 0 \right> - \left| 1 \right> \right) \) and apply oracle.
\[ U_f |x\rangle |–\rangle = U_f |x\rangle \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \]
\[ = |x\rangle \frac{1}{\sqrt{2}} (|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle) \]
\[ = |x\rangle \frac{1}{\sqrt{2}} (|f(x)\rangle - |\bar{f}(x)\rangle) \]
\[ = (-1)^{f(x)} |x\rangle |–\rangle \]

Known as the **phase kick-back trick** – value of the function reflected in phase.

**Q:** If we observe the first qubit on output: to what operation is the oracle equivalent for \( f_0, f_1, f_x, f_{\bar{x}} \)?

**A:**

<table>
<thead>
<tr>
<th>( f_0 )</th>
<th>( f_1 )</th>
<th>( f_x )</th>
<th>( f_{\bar{x}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I )</td>
<td>(-I)</td>
<td>( Z )</td>
<td>(-Z)</td>
</tr>
</tbody>
</table>
One more transform: Exploit the relation $HZH = X$. Apply $H$ to $x$ before and after $U_f$. Let $x = |0\rangle$ (top qubit).

- If $f$ is either $f_0$ or $f_1$ we observe $|0\rangle$
- If $f$ is either $f_x$ or $f_{\bar{x}}$ we observe a $|1\rangle$

**Done!**

**DIAGRAM**

![Diagram](image_url)
Cirq codes

Resources:

- See [https://github.com/quantumlib/cirq](https://github.com/quantumlib/cirq)
- Also: the Cirq workshop bootcamp repository (google search it)

Cirq provides a toolkit (a ‘framework’) for simulating quantum algorithms.

- Written in python. Implements all the gates we have seen and more.
- The following illustration shows a simple example
import cirq
q0 = cirq.NamedQubit("q0")
q1 = cirq.NamedQubit("q1")
q2 = cirq.NamedQubit("q2")
ops = [cirq.X(q0), cirq.H(q1), cirq.CNOT(q1, q2), cirq.X(q1), cirq.CZ(q0, q1)]
circuit = cirq.Circuit(*ops)
print(circuit)

**Output:**

![Circuit Diagram]
A longer example showing many of the gates

```python
import cirq
import numpy as np
q0, q1, q2 = cirq.LineQubit.range(3)
ops = [cirq.X(q0),
    cirq.Y(q1),
    cirq.Z(q2),
    cirq.CZ(q0, q1),
    cirq.CNOT(q1, q2),
    cirq.H(q0),
    cirq.T(q1),
    cirq.S(q2),
    cirq.CCZ(q0, q1, q2),
    cirq.SWAP(q0, q1),
    cirq.CSWAP(q0, q1, q2),
    cirq.CCX(q0, q1, q2),
    cirq.ISWAP(q0, q1),
    cirq.Rx(0.5 * np.pi)(q0),
    cirq.Ry(.5 * np.pi)(q1),
    cirq.Rz(0.5 * np.pi)(q2),
    (cirq.X**0.5)(q0)]
print(cirq.Circuit(*ops))
print(cirq.unitary(cirq.CNOT))
print(cirq.unitary(cirq.CZ))
```
A few commands to loot at:

- `cirq.X(q0)`: gate X at q0.
- `cirq.LineQubit.range(p)`: create a line of qubits .. or
- `cirq.GridQubit.range(p,q)`: create a grid of qubits ..
- `print(cirq.Circuit(*ops))`: prints circuit
Quantum Fourier Transform

- QFT is at the core of the Shor algorithm
- Main idea of QFT: Exploit product decomposition. Recall:

\[ x = [x_0, x_1, \cdots, x_{N-1}]^T \]
is transformed to \( y \) with:

\[ y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2i\pi jk/N} \]

Therefore:

\[ |j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2i\pi jk/N} |k\rangle \quad (\ast) \]

- Suppose that \( N = 2^n \). Write any \( k \) in its binary representation:

\[ k = k_1 2^{n-1} + k_2 2^{n-2} + \cdots + k_n 2^0 = \sum_{l=1}^{n} k_l 2^{n-l} \]
Drop the scaling term $\frac{1}{\sqrt{N}}$ in (*) and set that $N = 2^n$. Then:

$$
\sum_{k=0}^{2^n-1} e^{2i\pi jk/2^n} |k\rangle = \sum_{k=0}^{2^n-1} e^{2i\pi j \sum_{l=1}^{n} k_l 2^{-l}} |k_1...k_n\rangle
$$

$$
= \sum_{k_1=0}^{1} \sum_{k_2=0}^{1} \ldots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2i\pi j k_l 2^{-l}} |k_l\rangle
$$

$$
= \bigotimes_{l=1}^{n} \left[ \sum_{k_l=0}^{1} e^{2i\pi j k_l 2^{-l}} |k_l\rangle \right]
$$

$$
= \bigotimes_{l=1}^{n} \left[ |0\rangle + e^{2i\pi j 2^{-l}} |1\rangle \right]
$$
Write \( j = \sum_{m=1}^{n} j_m 2^{n-m} \). Since \( e^{2i\pi \times \text{integer}} = 1 \) then

\[
e^{2i\pi j 2^{-l}} = e^{2i\pi \sum_{m=1}^{n} j_m 2^{n-m} 2^{-l}} = e^{2i\pi \sum_{m=1}^{n} j_m 2^{n-l-m}} = e^{2i\pi \sum_{m=n-l+1}^{n} j_m 2^{n-l-m}} = e^{2i\pi 0 \cdot j_{n-l+1} j_{n-l+2} \cdots j_n}
\]

In the end:

\[
\frac{1}{2^{n/2}} \sum_{k=0}^{2^{n-1}} e^{2i\pi j_k / 2^n} |k\rangle =
\frac{(|0\rangle + e^{2i\pi 0 \cdot j_n} |1\rangle) (|0\rangle + e^{2i\pi 0 \cdot j_{n-1} j_n} |1\rangle) \cdots (|0\rangle + e^{2i\pi 0 \cdot j_1 j_2 \cdots j_n} |1\rangle)}{2^{n/2}}
\]

Let \( R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2i\pi / 2^k} \end{pmatrix} \)
Here is a diagram for a 4-cubit QFT

\[ |j_1\rangle \xrightarrow{H} |0\rangle + e^{2i\pi 0. j_2 j_3 j_4} |1\rangle \]

\[ |j_2\rangle \xrightarrow{H} |0\rangle + e^{2i\pi 0. j_2 j_3 j_4} |1\rangle \]

\[ |j_3\rangle \xrightarrow{H} |0\rangle + e^{2i\pi 0. j_3 j_4} |1\rangle \]

\[ |j_4\rangle \xrightarrow{H} |0\rangle + e^{2i\pi 0. j_4} |1\rangle \]
L. K. Glover

On the future of QC:

Will quantum computers ever grow into their software? How long will it take them to blossom into the powerful calculating engines that theory predicts they could be? I would not dare to guess, but I advise all would-be forecasters to remember these words, from a discussion of the Electronic Numerical Integrator and Calculator (ENIAC) in the March 1949 issue of Popular Mechanics: Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and weigh only 1.5 tons.