Implementation of Grover's Algorithm with Qiskit tool

Raimondas Čiegis

Department of Mathematical Modelling, email: rc@vgtu.lt

September 22, 2026

Let's remind iterative Grover's algorithm

$$G = Z_f \, Z_A.$$

In Qiskit, operations are performed from left to right, unlike traditional matrix calculations done in Matlab.

Here operator Z_f changes the signs of the amplitudes if the oracle function has the value $f(x) = 1$

$$Z_f : \quad |x\rangle \quad \rightarrow \quad (-1)^{f(x)} \, |x\rangle \, ,$$

Z_A performs an inversion transformation with respect to the mean value

$$|\psi\rangle = \sum_{x=0}^{N-1} \alpha_x \, |x\rangle \, , \quad A = \frac{1}{N} \sum_{x=0}^{N-1} \alpha_x ,$$

$$Z_A \, |\psi\rangle = \sum_{x=0}^{N-1} (2A - \alpha_x) \, |x\rangle \, .$$

We start the laboratory work:

We start the laboratory work:

First, it is important to write all operators using standard quantum gates.

We start the laboratory work:

First, it is important to write all operators using standard quantum gates.

In Lecture 5, we performed calculations using classical logic gates, but in quantum computers we can use only quantum logic gates.

This is an integral part of implementing any quantum algorithm.

We start the laboratory work:

First, it is important to write all operators using standard quantum gates.

In Lecture 5, we performed calculations using classical logic gates, but in quantum computers we can use only quantum logic gates.

This is an integral part of implementing any quantum algorithm.

Let me remind that in Lecture 5 we showed that the inversion transformation with respect to the mean value can also be written as follows (all operators are unitary):

$$Z_A = H^{\otimes n}\, Z_{OR}\, H^{\otimes n}, \quad n = \log N,$$

$$Z_{OR}: \quad |x\rangle \;\rightarrow \begin{cases} |x\rangle, & x = 0^n, \\ -|x\rangle, & x \neq 0^n. \end{cases}$$

Let's consider the first practical problem. We have two qubits q_0 and q_1 (i.e. $|q_1, q_0\rangle$), such a system can be in four states

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle.$$

Let's consider the first practical problem. We have two qubits q_0 and q_1 (i.e. $|q_1, q_0\rangle$), such a system can be in four states

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle.$$

Using Grover's search algorithm, let's find the element "11"

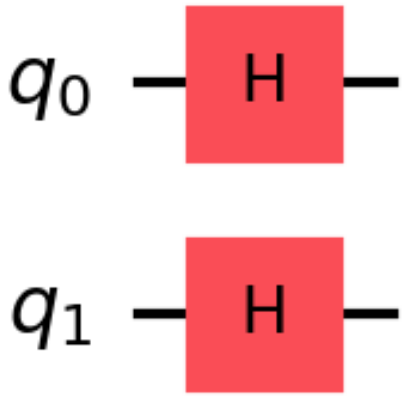
Let's consider the first practical problem. We have two qubits q_0 and q_1 (i.e. $|q_1, q_0\rangle$), such a system can be in four states

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle.$$

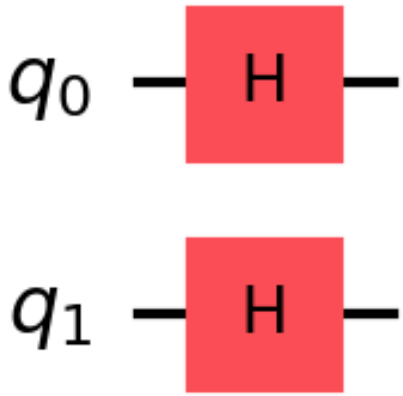
Using Grover's search algorithm, let's find the element "11"

```
#Required imports

from qiskit import QuantumCircuit

from qiskit import transpile

circuit = QuantumCircuit(2)
circuit.h(0)
circuit.h(1)
```

Visualization of circuit construction



Visualization of circuit construction

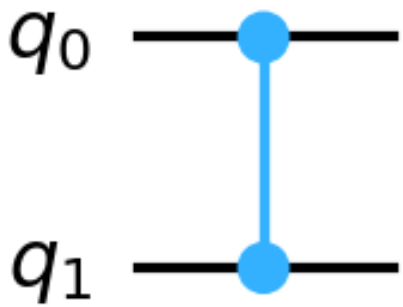


We obtain the following initial state of the two-qubit chain: a superposition of four base states with equal coefficients

$$|\psi\rangle = \frac{1}{2}\sum_{x=0}^{3}|x\rangle = \frac{1}{2}\big(\,|00\rangle + |01\rangle + |10\rangle + |11\rangle\,\big).$$

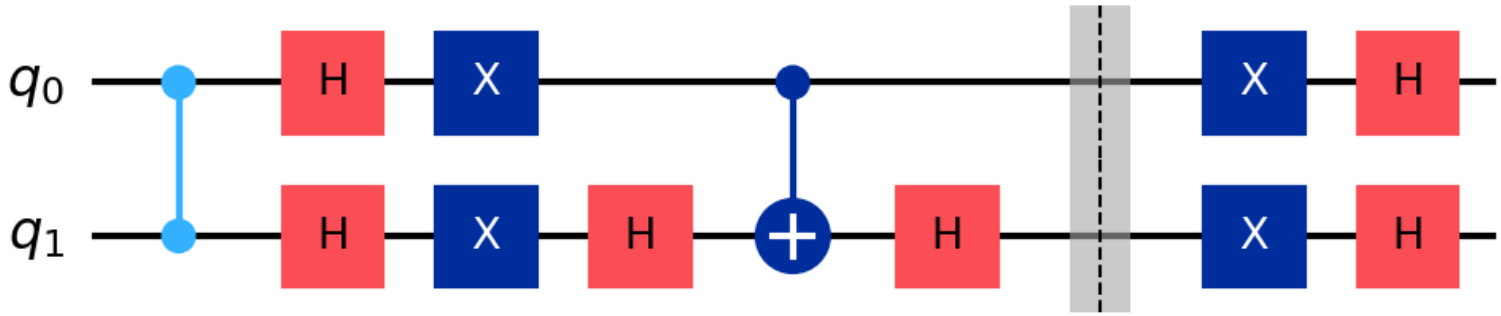
#We generate an oracle operator Z_f that recognizes the state "11".
#We describe it as a separate quantum circuit that uses
#the same two qubits q_0, q_1:

```
oracle = QuantumCircuit(2)
oracle.cz(0, 1)
```



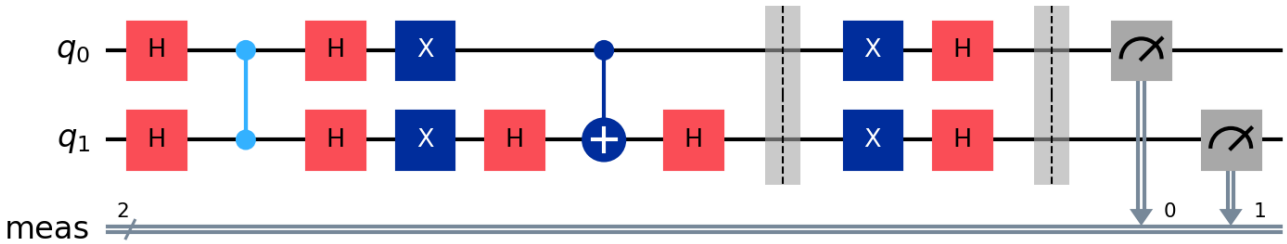
#Generate Grover's operator: add Z_A transformation:

```
grover = oracle
grover.h(0)
grover.h(1)
grover.x(0)
grover.x(1)
grover.h(1)
grover.cx(0, 1)
grover.h(1)
grover.x(0)
grover.x(1)
grover.h(0)
grover.h(1)
```


q_0
q_1
H
H
X
X
H
H
X
X
H
H

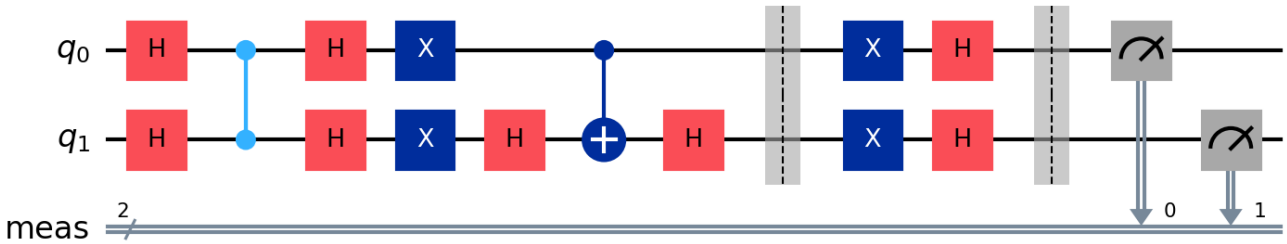
#We perform one Grover iteration and measure the result:

circuit.measure_all()



#We perform one Grover iteration and measure the result:

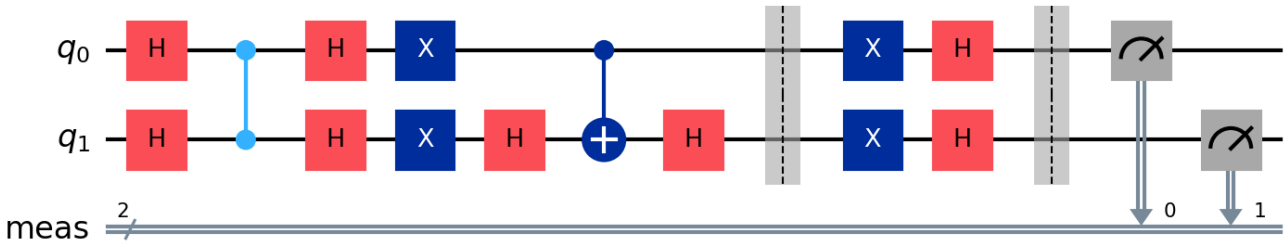
circuit.measure_all()



simulator = AerSimulator()
circuit2 = transpile(circuit, simulator)
job = simulator.run(circuit2, shots=200)
result = job.result()
counts = result.get_counts(circuit2)
print(counts)

#We perform one Grover iteration and measure the result:

circuit.measure_all()



simulator = AerSimulator()
circuit2 = transpile(circuit, simulator)
job = simulator.run(circuit2, shots=200)
result = job.result()
counts = result.get_counts(circuit2)
print(counts)

Results:

{'11': 200}

Do two iterations of the Grover algorithm:

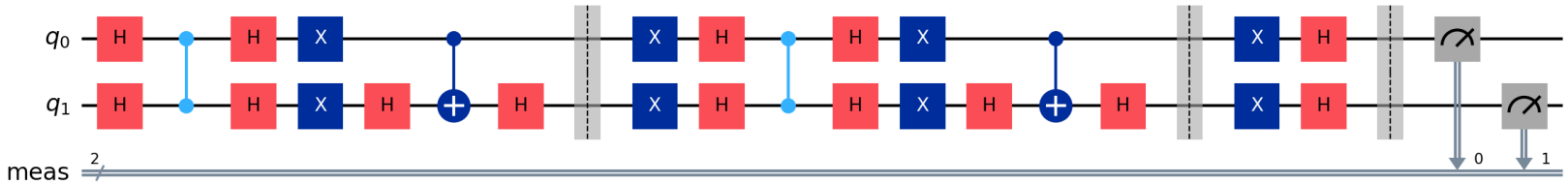
Add the second iteration to the circuit:

circuit = circuit.compose(grover)

Do two iterations of the Grover algorithm:

Add the second iteration to the circuit:

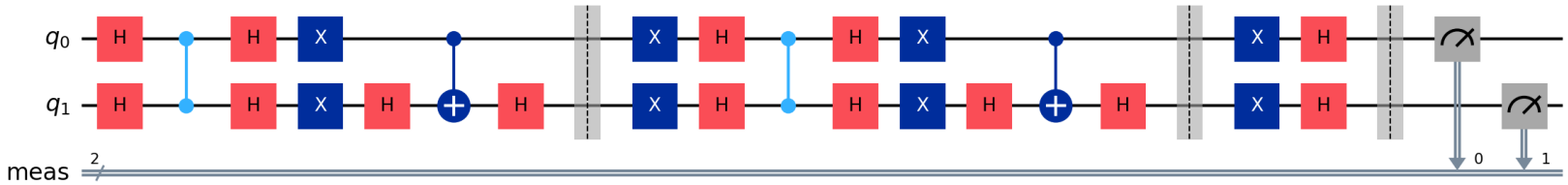
circuit = circuit.compose(grover)



Do two iterations of the Grover algorithm:

Add the second iteration to the circuit:

circuit = circuit.compose(grover)



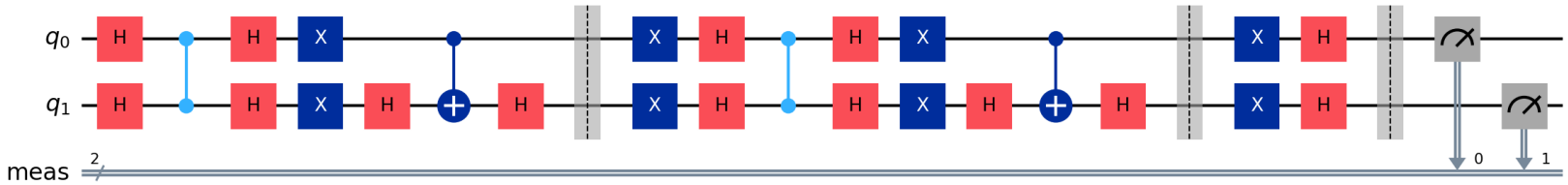
Results:

{'11': 44, '10': 52, '00': 44, '01': 60}

Do two iterations of the Grover algorithm:

Add the second iteration to the circuit:

circuit = circuit.compose(grover)



Results:

{'11': 44, '10': 52, '00': 44, '01': 60}

Remind that from theoretical estimates it also follows that one iteration is optimal for two qubits.

Let's change our target and make a search for the element "01".

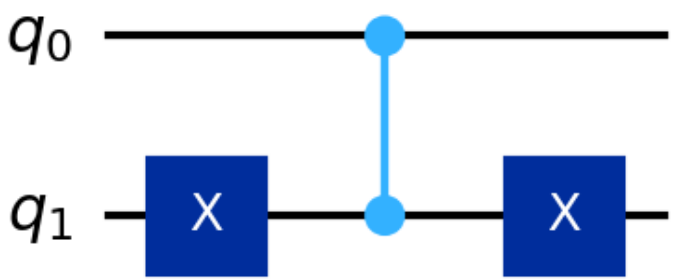
It is enough to change only the oracle part.

oracle.x(1)
oracle.cz(0, 1)
oracle.x(1)

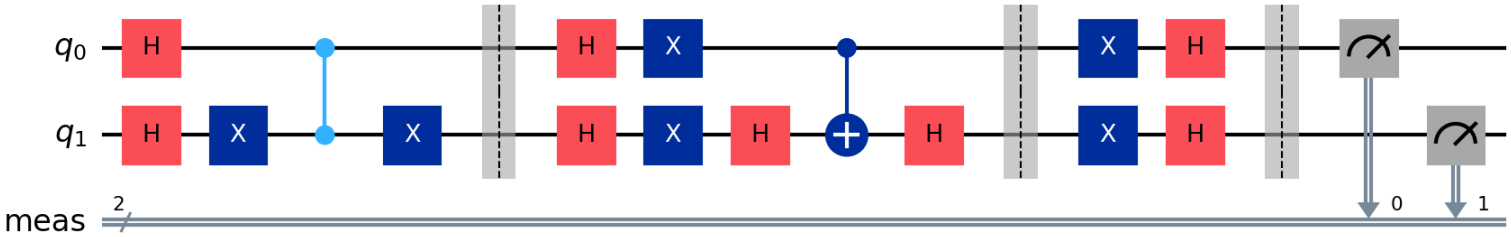
Let's change our target and make a search for the element "01".

It is enough to change only the oracle part.

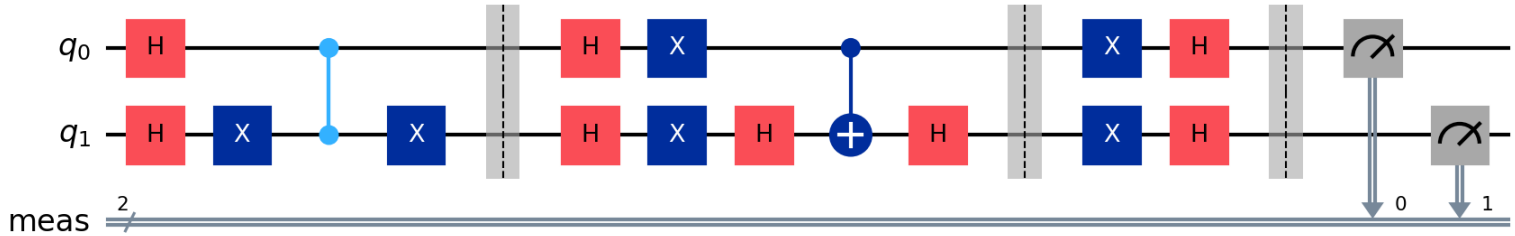
oracle.x(1)
oracle.cz(0, 1)
oracle.x(1)



We perform one iteration of Grover's algorithm



We perform one iteration of Grover's algorithm



We obtain the following results from a series of experiments:

{'01': 200}

Next we take a system of three qubits and generate a superposition of eight quantum states:

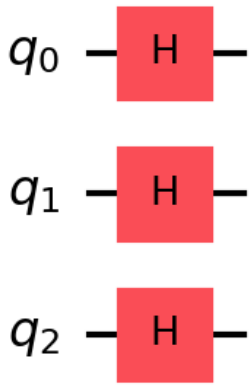
$$|\psi\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle).$$

Next we take a system of three qubits and generate a superposition of eight quantum states:

$$|\psi\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle).$$

Initial distribution is generated by the following circuit

```
init = QuantumCircuit(3)
init.h(0)
init.h(1)
init.h(2)
```

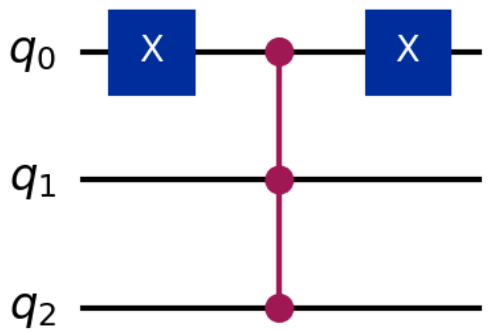



```
circuit = QuantumCircuit(3)
circuit = circuit.compose(init)
```

```
# Define the Oracle to find a state "110"
oracle = QuantumCircuit(3)
oracle.x(0)
oracle.ccz(0, 1, 2)
oracle.x(0)
```

Define the Oracle to find a state "110"

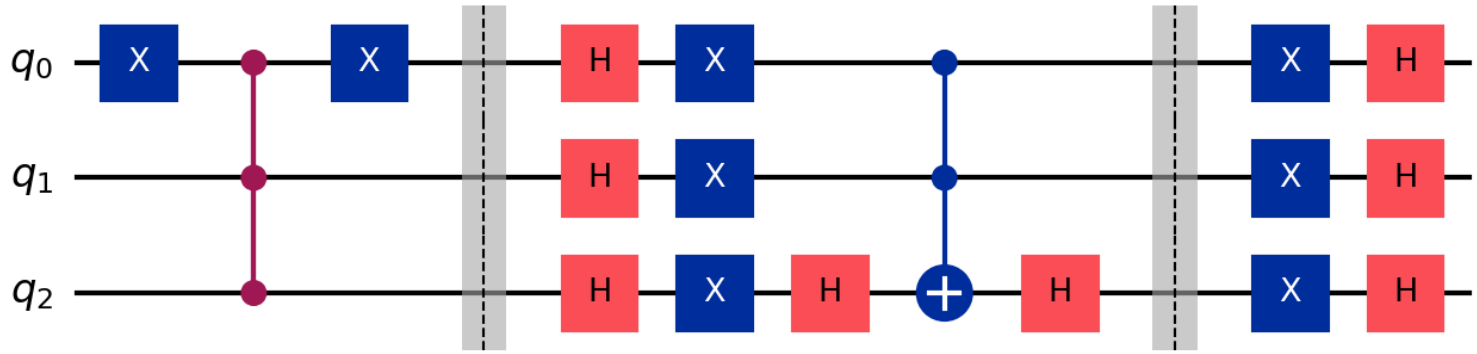
```
oracle = QuantumCircuit(3)
oracle.x(0)
oracle.ccz(0, 1, 2)
oracle.x(0)
```



```
# Generate Grover's operator:
grover = oracle

grover.barrier()
grover.h(0)
grover.h(1)
grover.h(2)
grover.x(0)
grover.x(1)
grover.x(2)
grover.h(2)
grover.ccx(0, 1, 2)
grover.h(2)
```

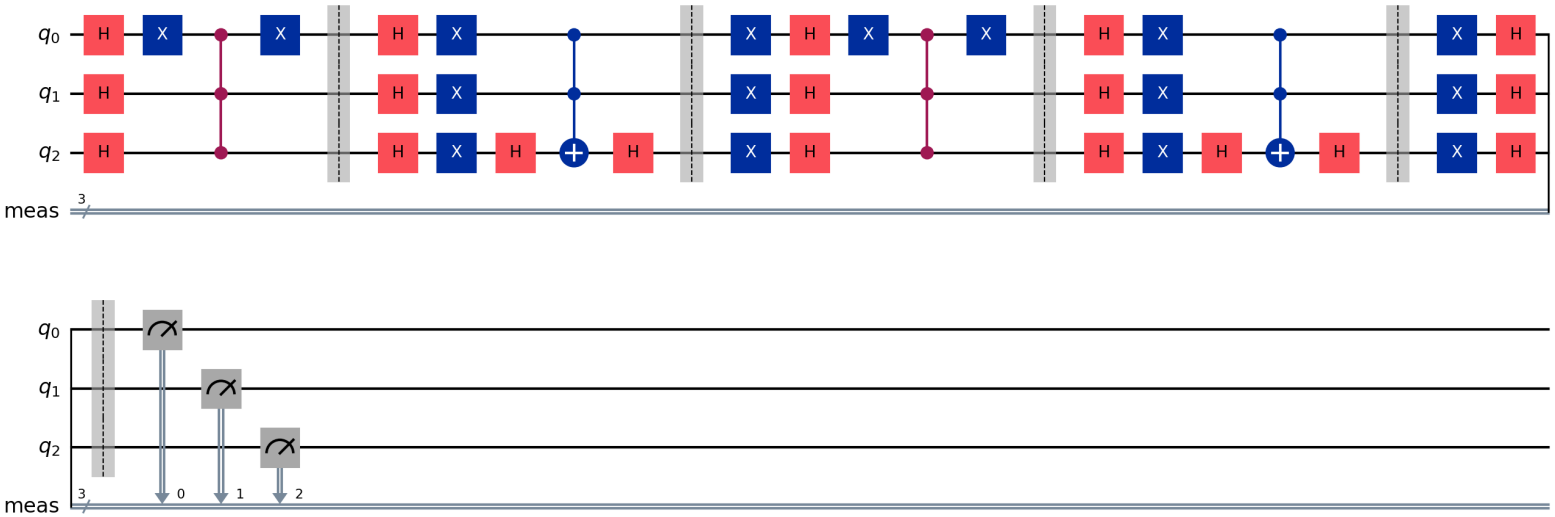
```
grover.barrier()
grover.x(0)
grover.x(1)
grover.x(2)
grover.h(0)
grover.h(1)
grover.h(2)
```

```
grover.barrier()
grover.x(0)
grover.x(1)
grover.x(2)
grover.h(0)
grover.h(1)
grover.h(2)
```



```
#Generate a circuit with two Grover's iterations
#and make measurements
circuit = circuit.compose(grover)
circuit = circuit.compose(grover)
circuit.measure_all()
```

```
#Generate a circuit with two Grover's iterations
#and make measurements
circuit = circuit.compose(grover)
circuit = circuit.compose(grover)
circuit.measure_all()
```



Results of two experimental series of 100 trials each:

{'111': 1, '101': 1, '000': 1, '100': 1, '001': 1, '110': 95 }

{'100': 1, '001': 1, '010': 1, '101': 1, '011': 1, '111': 2, '110': 93 }