HW11

June 2, 2023

1 Exercise Set 11

1.1 Mohaddeseh Mozaffari

[ ]:
```
import cirq
from cirq import X, H, Z, inverse, CX, I
```

2 Q1:

2.1 Figure 1)

[ ]:
```
def edge_check(a, b, c):
    yield CX(qq[a], qq[c])
    yield CX(qq[b], qq[c])

def oracle():
    # check 0-1 edge and store at 4th qubit
    yield edge_check(0, 1, 4)
    # check 0-2 edge and store at 5th qubit
    yield edge_check(0, 2, 5)
    # check 0-3 edge and store at 6th qubit
    yield edge_check(0, 3, 6)
    # check 1-3 edge and store at 7th qubit
    yield edge_check(1, 3, 7)

    # check all edge qubits
    yield X(qq[8]).controlled_by(*(qq[4:8]))

def oracle_computation():
    yield oracle()
    yield Z(qq[8])
    yield inverse(oracle())

def inversion():
    yield H.on_each(*qq)
    yield X.on_each(*qq)
    yield Z(qq[3]).controlled_by(*(qq[0:3]))
```

```
    yield X.on_each(*qq)
    yield H.on_each(*qq)
```

[ ]:
```
circuit = cirq.Circuit()

qq = cirq.LineQubit.range(9)

circuit.append(H.on_each(*(qq[0:4])))
##################################################


for i in range(2):
    circuit.append(oracle_computation())
    circuit.append(inversion())

# we are only intertested in outputs of first 4 qubits
circuit.append(cirq.measure(*(qq[0:4]), key='result'))
##########################################################################################################
# determine the statistics of the measurements
trials_number = 1000
s = cirq.Simulator()
samples = s.run(circuit, repetitions=trials_number)
result = samples.measurements["result"]

def bitstring(bits):
    return "".join(str(int(b)) for b in bits)

counts = samples.histogram(key="result",fold_func=bitstring)
print(counts)
print(circuit)
```

```
Counter({'1111': 81, '0000': 80, '1100': 72, '1101': 69, '0110': 68, '0101': 66,
'0011': 66, '1001': 65, '0111': 62, '1110': 60, '0001': 56, '1010': 53, '1011':
53, '0010': 50, '0100': 50, '1000': 49})
```

```
0:   H  @   @   @                   @   @   @
   H  X  @  X  H  @   @   @                   @
    @   @   H  X  @  X  H  M('result')

1:   H    @    @                @   @   H
X       @  X  H    @    @               @
@    H   X      @  X  H  M
```

```
2:  H       @             @   H   X
      @ X H       @             @   H
 X            @ X H M
```

```
3:  H           @ @       @   @   H   X
      Z X H           @ @       @   @
 H   X         Z X H M
```

```
4:    X  X          @   @         X       X
  H X X H   X  X          @   @
X       X  H X X H
```

```
5:       X  X       @   @  X          X   H
  X X H        X  X       @   @  X
   X   H  X X H
```

```
6:          X  X   @   @      X  X   H   X
  X H             X  X   @   @      X  X
   H   X  X H
```

```
7:          X   X @   @ X   X   H   X   X
  H               X   X @   @ X   X   H
   X   X  H
```

```
8:                  X Z X H   X   X   H
                        X Z X H   X   X
   H
```

As a result, the graph is not bipartite since all probabilities are almost equal.

2.2 Figure 2)

[ ]:
```
def edge_check(a, b, c):
    yield CX(qq[a], qq[c])
    yield CX(qq[b], qq[c])
```

```
def oracle():
    # check 0-2 edge and store at 4th qubit
    yield edge_check(0, 2, 4)
    # check 0-3 edge and store at 5th qubit
    yield edge_check(0, 3, 5)
    # check 2-1 edge and store at 6th qubit
    yield edge_check(2, 1, 6)
    # check 1-3 edge and store at 7th qubit
    yield edge_check(1, 3, 7)

    # check all edge qubits
    yield X(qq[8]).controlled_by(*(qq[4:8]))

def oracle_computation():
    yield oracle()
    yield Z(qq[8])
    yield inverse(oracle())

def inversion():
    yield H.on_each(*qq)
    yield X.on_each(*qq)
    yield Z(qq[3]).controlled_by(*(qq[0:3]))
    yield X.on_each(*qq)
    yield H.on_each(*qq)
```

[ ]:
```
circuit = cirq.Circuit()

qq = cirq.LineQubit.range(9)

circuit.append(H.on_each(*(qq[0:4])))
################################################


for i in range(2):
    circuit.append(oracle_computation())
    circuit.append(inversion())

# we are only intertested in outputs of first 4 qubits
circuit.append(cirq.measure(*(qq[0:4]), key='result'))
################################################################################################
# determine the statistics of the measurements
trials_number = 1000
s = cirq.Simulator()
samples = s.run(circuit, repetitions=trials_number)
result = samples.measurements["result"]

def bitstring(bits):
```

```
    return "".join(str(int(b)) for b in bits)

counts = samples.histogram(key="result",fold_func=bitstring)
print(counts)
print(circuit)
```

```
Counter({'0011': 482, '1100': 458, '0001': 7, '0111': 7, '1000': 7, '1001': 6,
'0110': 5, '1010': 5, '0010': 4, '1110': 4, '0000': 4, '0101': 3, '1101': 3,
'1011': 2, '0100': 2, '1111': 1})
```

```
0:  H  @  @                        @
  @  H  X  @  X  H  @  @
   @        @  H  X  @  X  H  M('result')


1:  H          @  @           @   @   H  X
       @  X  H          @  @           @
  @   H  X         @  X  H  M


2:  H   @   @                      @  @
  H  X   @  X  H   @   @
     @  @  H  X   @  X  H  M


3:  H       @      @       @  @   H  X
       Z  X  H       @      @       @  @
   H   X           Z  X  H  M


4:    X  X           @   @              X
  X  H  X  X  H   X  X          @   @
       X  X  H  X  X  H


5:       X  X        @   @    X  X   H  X
  X  H           X  X        @   @    X
   X   H  X  X  H


6:          X  X     @   @       X   X  H
```

```
 X   X  H                X  X      @    @
 X    X   H   X   X  H
```

```
7:                X  X  @    @  X   X    H    X   X
 H                          X  X  @    @  X   X
 H    X   X   H
```

```
8:                     X  Z  X  H  X    X    H
                                 X  Z  X  H   X
 X    H
```

The graph is bipartite because the 0011 and 1100 have the highest probabilities.

3 Q2:

3.1 Figure 1)

If color is 0110, We have:

[ ]:
```
circuit = cirq.Circuit()

qs = cirq.LineQubit.range(5)
circuit.append(I(qs[0]))
circuit.append(X(qs[1]))
circuit.append(X(qs[2]))

circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[2], qs[4]))
circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[3], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[3], qs[4]))

circuit.append(cirq.measure(qs[4], key="result"))
```

[ ]:
```
trials_number = 1000
s = cirq.Simulator()
samples = s.run(circuit, repetitions=trials_number)
result = samples.measurements["result"]

def bitstring(bits):
```

```
    return "".join(str(int(b)) for b in bits)

counts = samples.histogram(key="result",fold_func=bitstring)
print(counts)
```

```
Counter({'1': 1000})
```

[ ]:
```
print(circuit)
```

```
0:   I  @    @    @

1:   X    @          @

2:   X        @

3:                @    @

4:     X  X  X  X  X  X  X  X  M('result')
```

If color is 1001, We have:

[ ]:
```
circuit = cirq.Circuit()

qs = cirq.LineQubit.range(5)

circuit.append(X(qs[0]))
circuit.append(I(qs[1]))
circuit.append(X(qs[3]))

circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[2], qs[4]))
circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[3], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[3], qs[4]))

circuit.append(cirq.measure(qs[4], key="result"))
```

[ ]:
```
trials_number = 1000
s = cirq.Simulator()
samples = s.run(circuit, repetitions=trials_number)
result = samples.measurements["result"]

def bitstring(bits):
    return "".join(str(int(b)) for b in bits)
```

```
counts = samples.histogram(key="result",fold_func=bitstring)
print(counts)
```

```
Counter({'1': 1000})
```

[ ]:
```
print(circuit)
```

```
0:   X  @    @    @

1:   I    @          @

2:           @

3:   X             @    @

4:     X  X  X  X  X  X  X  X  M('result')
```

3.2 Figure 2)

If color is 0011, We have:

[ ]:
```
circuit = cirq.Circuit()

qs = cirq.LineQubit.range(5)

circuit.append(I(qs[0]))
circuit.append(I(qs[1]))
circuit.append(X(qs[2]))
circuit.append(X(qs[3]))

circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[2], qs[4]))
circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[3], qs[4]))
circuit.append(CX(qs[2], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[3], qs[4]))

circuit.append(cirq.measure(qs[4], key="result"))
```

[ ]:
```
trials_number = 1000
s = cirq.Simulator()
samples = s.run(circuit, repetitions=trials_number)
result = samples.measurements["result"]

def bitstring(bits):
    return "".join(str(int(b)) for b in bits)
```

```
counts = samples.histogram(key="result",fold_func=bitstring)
print(counts)
```

```
Counter({'0': 1000})
```

[ ]:
```
print(circuit)
```

```
0:   I  @    @

1:   I            @  @

2:   X    @       @

3:   X         @         @

4:     X  X  X  X  X  X  X  X  M('result')
```

If color is 1100, We have:

[ ]:
```
circuit = cirq.Circuit()

qs = cirq.LineQubit.range(5)

circuit.append(X(qs[0]))
circuit.append(X(qs[1]))
circuit.append(I(qs[2]))
circuit.append(I(qs[3]))

circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[2], qs[4]))
circuit.append(CX(qs[0], qs[4]))
circuit.append(CX(qs[3], qs[4]))
circuit.append(CX(qs[2], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[1], qs[4]))
circuit.append(CX(qs[3], qs[4]))

circuit.append(cirq.measure(qs[4], key="result"))
```

[ ]:
```
trials_number = 1000
s = cirq.Simulator()
samples = s.run(circuit, repetitions=trials_number)
result = samples.measurements["result"]

def bitstring(bits):
    return "".join(str(int(b)) for b in bits)
```

```
counts = samples.histogram(key="result",fold_func=bitstring)
print(counts)
```

```
Counter({'0': 1000})
```

[ ]:
```
print(circuit)
```

```
0:   X  @    @

1:   X            @  @

2:   I    @      @

3:   I        @        @

4:     X  X  X  X  X  X  X  X  M('result')
```