WS3

November 17, 2024

```
[ ]: from quantum.computer import QuantumComputer
     %display latex
```

1 1. Simon algorithm

Exercise 1.1: implement the Simon's circuit: write a function `simonCircuit(n, m, f)` that takes as input a function $f : \{0,1\}^n \to X$ and outputs the outcome of the measure of the Simon circuit. (Here, we assume that elements of X are encoded by integers written on m bits, *e.g.* $X = \{0, 1, \ldots, 2^m - 1\}$.)

```
[ ]: def simonCircuit(n, m, f):
         ### write your code here
```

Exercice 1.2: implement Simon's algorithm: write a function `simon(n, m, f)` that takes as input a function $f : \{0,1\}^n \to X$ with the promise that there exists `a` such that $f(x) = f(y)$ iff $y \in \{x, x+a\}$, and outputs a.

```
[ ]: def simon(n, m, f):
         """
         Return `a` such that `f(x) = f(y)` iff `y = x` or `y = x + a`.
         """
         ### write your code here
```

Exercise 1.3: write tests demonstrating that your implementation works

```
[ ]: ### write your tests here
```

2 2. Shor algorithm

2.1 Controlled phase shift gate

Unfortunately, controlled phase shift gates are not builtin functions of our quantum computer.

One can nevertheless implement it in soft as follows:

```
[ ]: def controlledPhaseShift(QC, c, x, angle):
         r"""
         INPUT:
```

```
    - ``QC'' -- the quantum computer on which the gate is applied

    - ``c'' -- the controlled register

    - ``x'' -- the register on which the phase shift gate acts

    - ``angle`` -- the angle of rotation
    """
    QC.phase_shift(c, angle/2)
    QC.phase_shift(x, angle/2)
    QC.CX(c, x)
    QC.phase_shift(x, -angle/2)
    QC.CX(c, x)
```

Exercise 2.1: prove that the function `controlledPhaseShift` works correctly

2.2 Quantum Fourier transform

Let n be a positive integer. We recall that the Quantum Fourier transform is the gate QFT_n acting by:

$$\mathrm{QFT}_n |x\rangle = \frac{1}{2^{n/2}} \cdot \sum_{y=0}^{2^n-1} \zeta_n^{xy} |y\rangle$$

with $\zeta_n = \exp\left(\frac{2i\pi}{2^n}\right)$

Exercise 2.2: implement a function `QFT(QC, reg)` which applies the quantum Fourier transform to the register `reg` of the quantum computer `QC`.

```
[ ]: def QFT(QC, reg):
         ### write your code here
```

2.3 Shor circuit

Exercise 2.3: write a function `shorCircuit(n, m, f)` that takes as input integers n and m together with a function $f : \mathbb{Z} \to X$ (where elements of X are encoded by integers written on m bits, *e.g.* $X = \{0, 1, \dots, 2^m - 1\}$) and outputs the outcome of the Shor circuit (normalized as a rational number between 0 and 1) corresponding to these values.

```
[ ]: def shorCircuit(n, m, f):
         ### write your code here
```

Exercise 2.4: below is a simplified version of the Shor circuit avoiding the use of controlled shift gates; draw the corresponding circuit and prove that it is equivalent to the classical Shor circuit.

```
[ ]: def simplifiedShorCircuit(n, m, f):
         QC = QuantumComputer()
         x = QC.malloc(n)
         y = QC.malloc(m)
```

```
    QC.hadamard(x)
    QC.apply(f, x, y)
    angle = 0
    outcome = 0
    for i in range(n):
        QC.phase_shift(x[n-1-i], angle)
        QC.hadamard(x[n-1-i])
        v = QC.measure(x[n-1-i])
        angle = angle/2 + (pi/2)*v
        outcome += v / 2**(n-i)
    return outcome
```

We now would like to observe the behavior of Shor's circuit. For this, we use the following code which runs Shor circuit with the function `f` a bunch of times and draws an histogram of the outcomes.

[ ]:
```
def statistics(n, m, f, repeat=100):
    outcomes = [simplifiedShorCircuit(n, m, f) for _ in range(repeat)]
    return histogram(outcomes, bins=2^n, range=[0,1])
```

Exercise 2.5: run the function `statistics` with the functions $x \mapsto x \bmod r$ with $r = 2, 3, 4, 5, 8$ and comment the results you observe.

[ ]:
```
# The function x |-> x mod r
def modr(r):
    def f(x):
        return x % r
    return f
```

[ ]:
```
### write you tests here
```

2.4 Finding periods

The function `continued_fractions(x)` computes the continued fraction of a real number x

[ ]:
```
CF = continued_fraction(pi)
CF
```

The convergents of x (that are the rational approximations of x) are given by the method `convergents`. Precisely the call `CF.convergents()` returns the (lazy) list of convergents (this list is finite if x rational, infinite otherwise).

[ ]:
```
CF.convergents()       # the (lazy) list of convergents of pi
```

[ ]:
```
CF.convergents()[0]    # first approximation of pi
```

[ ]:
```
CF.convergents()[1]    # second approximation of pi
```

Exercise 2.6: write a function `findPeriod(n, m, f)` that returns the period of `f`

[ ]:
```
def findPeriod(n, m, f):
    ### write your code here
```

[ ]:
```
findPeriod(6, 3, modr(7))   # should return 7
```

[ ]:
```
findPeriod(7, 4, modr(11))  # should return 11
```

[ ]:
```
def powermod(a, n):
    def f(x):
        return (a^x) % n
    return f
findPeriod(7, 4, powermod(2, 11))  # should return 10 (which is the order of 2␣
 ↪modulo 11)
```