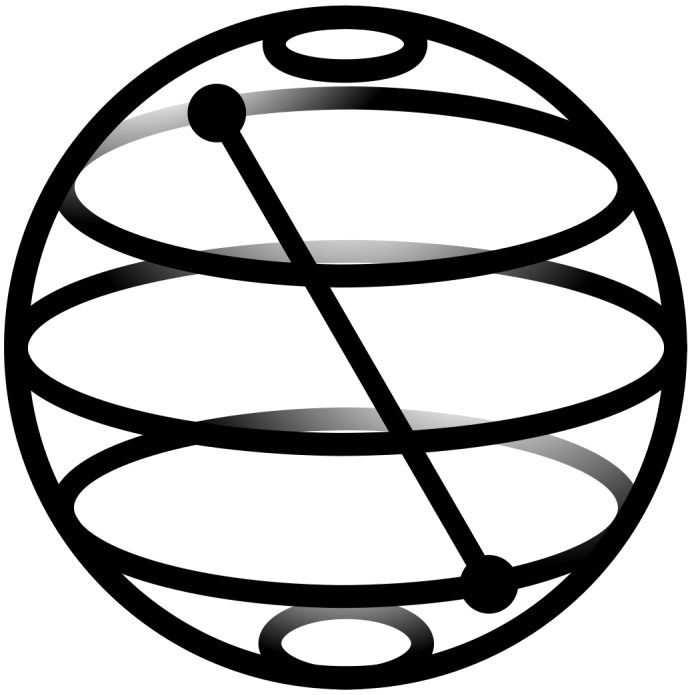
Introduction to Quantum Programming

Contents



1 Quantum Programming

1.1 Libraries

Quantum programming can be done using several programming libraries, we are going to use Qiskit which is a Python library for quantum computing. With Qiskit we can run our code on IBM real quantum computers. It can be installed with the command: `pip install qiskit`. Note that you may want to install matplotlib to visualize your quantum circuits.

First we need to do the necessary imports:

```
import numpy as np
import random
import matplotlib.pyplot as plt
from qiskit import *
from qiskit.circuit import *
from qiskit.extensions import *
from qiskit.circuit.library import *
from qiskit.extensions.simulator.snapshot import snapshot
from qiskit.quantum_info import Statevector
from qiskit.quantum_info.operators import Operator
from qiskit.extensions.simulator.snapshot import snapshot
from qiskit.visualization import plot_state_city, plot_bloch_multivector
from qiskit.visualization import plot_state_paulivec, plot_state_hinton
from qiskit.visualization import plot_state_qsphere
from qiskit.tools.visualization import plot_histogram
from qiskit.quantum_info import random_unitary
from qiskit.tools.monitor import job_monitor
```

1.2 Introduction

1.2.1 Quantum circuit

A circuit in Qiskit in composed of quantum and classical registers. The quantum registers are used to perform the quantum computations, and the classical ones to store the classical outcome of the measurements. In Qiskit a register is always initialize in state $|0\rangle$.

```
n = 3
m = n
# Quantum register of n qubits
q = QuantumRegister(n, name='q')
# Classical register of m bits
c = ClassicalRegister(m, name='c')
# Quantum circuit
qc = QuantumCircuit(q,c)
# Apply X on qubit 0
qc.x(q[0])
# Apply H on qubit 0
qc.h(q[0])
# Apply a CNOT on qubit 0 controlled by qubit 2
qc.cnot(q[2],q[0])
# Put a barrier between the operations
qc.barrier()
# Measure of the register q, the results are stored in c
qc.measure(q,c)
# Drawing the circuit
qc.draw('mpl')
# Note that the display style of the circuit can be modified with:
# qc.draw('mpl',style'bw'), qc.draw('mpl',style'iqx') or qc.draw()
```

Once a quantum circuit is created, one can run it to get some outcomes.

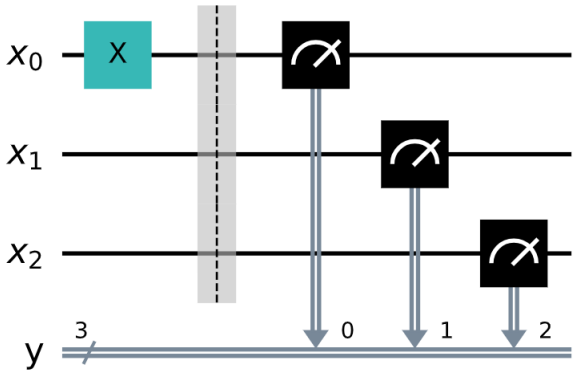
```
backend = BasicAer.get_backend('qasm_simulator')
job = execute(qc,backend,shots=1024)
res = dict(job.result().get_counts(qc))

# Plot the results of the measurements
plot_histogram(res)
```

The variable `res` is a dictionnary that contains the results of the measurements. The keys are the bit string measured and the values corresponds to the number of times this string has been measured. Hence, the dictionnary `{'00':134,'01':237,'10':511,'11':118}` indicates the we measured the state $|00\rangle$ 134 times, $|01\rangle$ 237 times, $|10\rangle$ 511 times and $|11\rangle$ 118 times.

It is also possible to only measure some qubits of the system:

```
# Measure of the qubit 0 of the register q
qc.measure(q[0],c[0])
```

The top wire corresponds to the qubit encoding the least significant bit. Let $|x_2x_1x_0\rangle = |000\rangle$ be a quantum state, we want to apply a NOT gate on $|x_0\rangle$, its Qiskit circuit is:



The outcome of the measurement is 001 with probability 1.

1.2.2 Sub-circuit

It is possible to transform a quantum circuit into a unitary gate, one can see it as a subroutine. Let's create a simple circuit:

```
q = QuantumRegister(2,name='q')
qc = QuantumCircuit(q)
qc.cnot(q[0],q[1])
qc.h(q)
```

We transform it into a unitary gate:

```
gate = qc.to_gate(label='gate')
```

We can now add it as a subroutine in other quantum circuits:

```
q = QuantumRegister(4,name='q')
qc = QuantumCircuit(q)
qc.x(q)
# We apply the gate on the qubits 1 and 2 of the new circuit
qc.append(gate,[q[1],q[2]])
# We apply the same gate on the qubits 1 and 2 of the new circuit but controlled by qubit 0
qc.append(gate.control(),[q[0],q[1],q[2]])
qc.draw('mpl')
```

Then, we can decompose the boxes we just created:

```
qc.decompose().draw('mpl')
```

We can compute the inverse, power or control gate easily:

```
q = QuantumRegister(5,name='q')
qc = QuantumCircuit(q)
# We apply the gate^2 on the qubits 1 and 2
qc.append(gate.power(2),[q[1],q[2]])
# We apply the inverse of the gate on the qubits 1 and 2
qc.append(gate.inverse(),[q[1],q[2]])
# We apply the gate on the qubits 2 and 3 controlled by qubits 0 and 1
qc.append(gate.control().control(),[q[0],q[1],q[2],q[3]])
qc.draw('mpl')
```

1.2.3 From unitary to gate

We can convert a unitary matrix into a quantum gate, let's create the gate $U = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/7} \end{bmatrix}$:

```
q = QuantumRegister(2,name='q')
qc = QuantumCircuit(q)
U = np.array([[1,0],[0,np.exp(1j*np.pi/7)]])
op = Operator(U)
qc.unitary(op,[q[0]],label='U')
qc.draw('mpl')
```

1.2.4 Run on IBM quantum computer

To run your code on IBM real quantum computer, you first have to create an account on IBM Quantum Experience to get your API Key. Once it is done, you can run the following code:

```
# List the simulators and the Quantum Computers available : provider.backends()
IBMQ.save_account('API KEY', overwrite=True)
IBMQ.load_account()
provider = IBMQ.get_provider(hub = 'ibm-q')
device = provider.get_backend('ibmq_manila')
job = execute(qc ,backend = device,shots = 1024)
# Entering the queue
job_monitor(job)
device_result = job.result()
real_counts = device_result.get_counts(qc)
plot_histogram(real_counts)
```

1.2.5 Usefull functions

One is encouraged to use the following function to be more efficient:

```
# Function to run a quantum circuit
def run(qc, n_shots=1024, plot=True):
    backend = BasicAer.get_backend('qasm_simulator')
    job = execute(qc,backend,shots=n_shots)
    res = dict(job.result().get_counts(qc))
    if plot:
        return plot_histogram(res)
    else:
        return res
```

```
# Function to plot the Bloch Sphere
def bloch(qc):
    backend = BasicAer.get_backend('statevector_simulator')
    result = backend.run(transpile(qc, backend)).result()
    psi = result.get_statevector(qc)
    return plot_bloch_multivector(psi)
```

1.3 First quantum circuits

1. Plot the Bloch Sphere representation of the states $|0\rangle$,$|1\rangle$, $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Here is the code for $|0\rangle$:

```
q = QuantumRegister(1)
qc = QuantumCircuit(q)
bloch(qc)
```

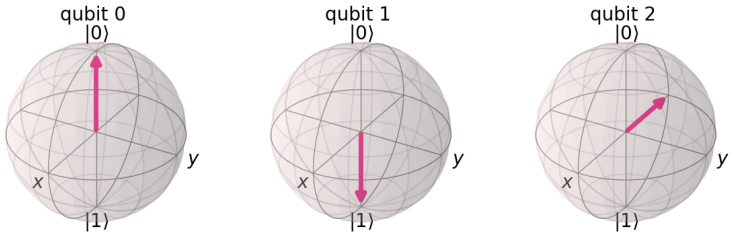
2. Do the same with the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)$.

3. Lastly, do it for the state $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Can you explain what is happening ?

4. Design a quantum circuit that creates the state $|\psi\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$, run it and plot the measurement results.

5. Design a quantum circuit that creates the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$, run it and plot the measurement results.

6. Design a quantum circuit that creates the state $|\psi\rangle = \frac{i}{\sqrt{2}}(|010\rangle - |110\rangle)$. Plot its Bloch Sphere representation, you should obtain this:



7. Using the function `random_unitary`, create a random state $|\phi\rangle$. Design a quantum circuit working on the state $|\phi\rangle \otimes |0\rangle$ that swaps these two qubits into:

$$|\psi\rangle \otimes |0\rangle \mapsto |0\rangle \otimes |\psi\rangle$$

Plot its Bloch Sphere representation before and after to check the result of the computation.

8. One wishes to design a quantum circuit that performs the following transformation on n qubits:

$$\sum_{k=0}^{2^n-1} \alpha_k |k\rangle \mapsto \sum_{k=0}^{2^n-1} \alpha_k |2^n - 1 - k\rangle$$

Note that it should work for any $(\alpha_0, \ldots, \alpha_{2^n-1}) \in \mathbb{C}^{2^n}$, not only for $\alpha_k = \frac{1}{\sqrt{2^n}}$. This transformation is just a permutation of the amplitudes α_k over the basis states, creating some kind of "mirror quantum state". Let's take an example with $n = 2$ qubits, in such a case this mapping acts as:

$$\alpha_0 |00\rangle \mapsto \alpha_0 |11\rangle$$
$$\alpha_1 |01\rangle \mapsto \alpha_1 |10\rangle$$
$$\alpha_2 |10\rangle \mapsto \alpha_2 |01\rangle$$
$$\alpha_3 |11\rangle \mapsto \alpha_3 |00\rangle$$

   (a) Make a list `unitaries` of length n that contains random 2×2 unitaries.

   (b) Design a quantum circuit `qc1` that creates the state $\sum_{k=0}^{2^n-1} \alpha_k |k\rangle$, where the α_k are that of `unitaries`.

   (c) Design a quantum circuit `qc2` that creates the state $\sum_{k=0}^{2^n-1} \alpha_k |2^n - 1 - k\rangle$, where the α_k are that of `unitaries`.

   (d) Run the quantum circuits and plot their results, both plots should mirror one another.
   It is possible to plot results from 2 quantum circuit at once with:
   `plot_histogram([run(qc1, plot=False),run(qc2, plot=False)])`

1.4 Cloning qubits ?

In quantum computing it is not possible to copy an arbitrary qubit state $\alpha|0\rangle+\beta|1\rangle$ due to the No-Cloning theorem. Hence, such a transformation is impossible:

$$|\psi\rangle|0\rangle \mapsto |\psi\rangle|\psi\rangle$$

However, it does not mean that any kind of copy is impossible, let's try some experiments. Unknown quantum states can be prepared using the function `random_unitary`.

1. Given a quantum state $|a\rangle$ with $a = 0, 1$ not being in any superposition ($|0\rangle$ or $|1\rangle$), design a quantum circuit that performs the following computation:

   $$|0\rangle \otimes |0\rangle \mapsto |0\rangle \otimes |0\rangle$$
   $$|1\rangle \otimes |0\rangle \mapsto |1\rangle \otimes |1\rangle$$

   That is:

   $$|a\rangle \otimes |0\rangle \mapsto |a\rangle \otimes |a\rangle$$

2. Can you still find a circuit to do it when the second qubit is initialized to $|1\rangle$?

   $$|a\rangle \otimes |1\rangle \mapsto |a\rangle \otimes |a\rangle$$

3. Given a quantum state $|a = a_n \dots a_1\rangle = |a_n\rangle \otimes \dots \otimes |a_1\rangle$ with $|a_i\rangle = |0\rangle$ or $|1\rangle$, design a quantum circuit that performs the following computation:

   $$|a_n \dots a_1\rangle \otimes |0\rangle^{\otimes n} \mapsto |a_n \dots a_1\rangle \otimes |a_n \dots a_1\rangle$$

4. Given two arbitrary unknown quantum states $|\psi\rangle$ and $|\phi\rangle$, design a quantum circuit that does the following transformation:

   $$|0\rangle \otimes |\psi\rangle \otimes |\phi\rangle \mapsto \alpha|0\rangle \otimes |\psi\rangle \otimes |\phi\rangle + \beta|1\rangle \otimes |\phi\rangle \otimes |\psi\rangle$$

   for $(\alpha, \beta) \in \mathbb{C}^2$.

5. One wishes to apply the quantum gates Z and Y on an arbitrary state $|\psi\rangle$ in the following way:

   $$|\psi\rangle \otimes |0\rangle \otimes |0\rangle \mapsto Z|\psi\rangle \otimes |0\rangle \otimes \frac{1}{\sqrt{2}}|0\rangle + |0\rangle \otimes Y|\psi\rangle \otimes \frac{1}{\sqrt{2}}|1\rangle$$

   Design a quantum circuit that implements this computation.

   Hint: the control-Z and control-Y gates can be used with `qc.cz(qubit_control,qubit_target)` *and* `qc.cy(qubit_control,qubit_target)`.

6. **Bonus**: One wants to do the following computation with a quantum circuit:

   $$\sum_{k=0}^{2^n-1} \alpha_k |k\rangle \otimes |0\rangle^{\otimes n} \mapsto \sum_{k=0}^{2^n-1} \alpha_k |k\rangle \otimes |k\rangle$$

   In other worlds, one wishes to copy the bit value of the first register to the second, where both are of size n. Note that this operation is not a copy of unknown quantum state as we did not copy the phase.

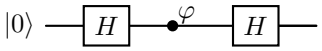
   (a) A naive method to perform this computation has a time complexity of $O(n)$. Implement it.

   (b) Try to make an efficient circuit that does the same computation with a time complexity of $1 + \lfloor \log_2(n) \rfloor = O(\log_2(n))$.

   (c) Plot the time complexity evolution for the naive and efficient methods for $n \in [1 \dots 50]$.

1.5 Single qubit interference

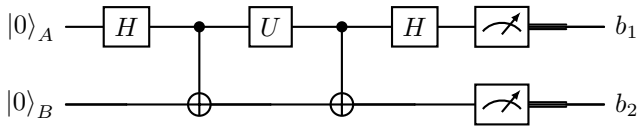
Interferences play a major role in quantum computation, by going into the Fourier basis (applying a Hadamard on a single qubit) and applying a phase it is possible to create an interesting interference scheme as depending on the value of the phase, the outcome of the measurement will change.



1. Implement the single qubit interference circuit with the phase gate $P_\varphi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}$.
2. Plot the probability of measuring $|0\rangle$ as a function of φ.

1.6 Superdense coding

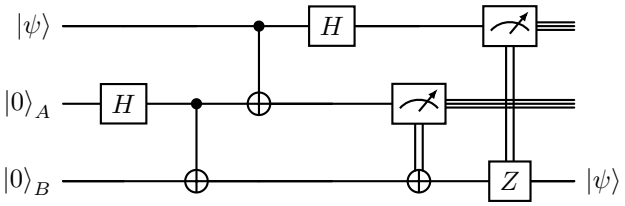
The Superdense coding protocol corresponds to the following quantum circuit:



Note that the black box U corresponds to the operations performed by Alice depending on the informations she wants to send. Write a function `Superdense_Coding(b2,b1)` that takes the bits of information Alice wants to send to Bob as inputs, and implements the corresponding Superdense coding protocol.

1.7 Quantum Teleportation

The Quantum Teleportation protocol is described with the quantum circuit:



1. Implement the Quantum Teleporation protocol.
2. Plot the Bloch Sphere of the 3 qubits system before and after teleportation. Do not forget to generate a random quantum state for the state $|\psi\rangle$ to teleport. Note that the state before teleportation is $|\psi\rangle \otimes \frac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle)$.

1.8 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is the quantum analog of the Fourier Transform, it is used as a subroutine in many quantum algorithms. It acts on a quantum state of n qubits ($N = 2^n$)

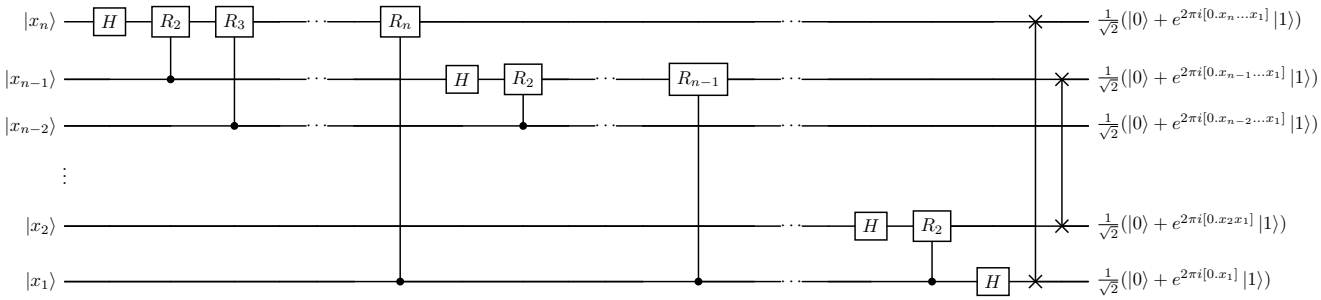
$$|x\rangle = \sum_{k=0}^{N-1} x_k |k\rangle$$

and maps it to

$$QFT\,|x\rangle = \sum_{k=0}^{N-1} y_k |k\rangle \text{ with } y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i \frac{jk}{N}}$$

The QFT is a transform between the computational basis and the Fourier basis. The computational basis encodes information in the $|k\rangle$ states unlike the Fourier basis that does it in the phases. Fore more informations, one is encouraged to click here.

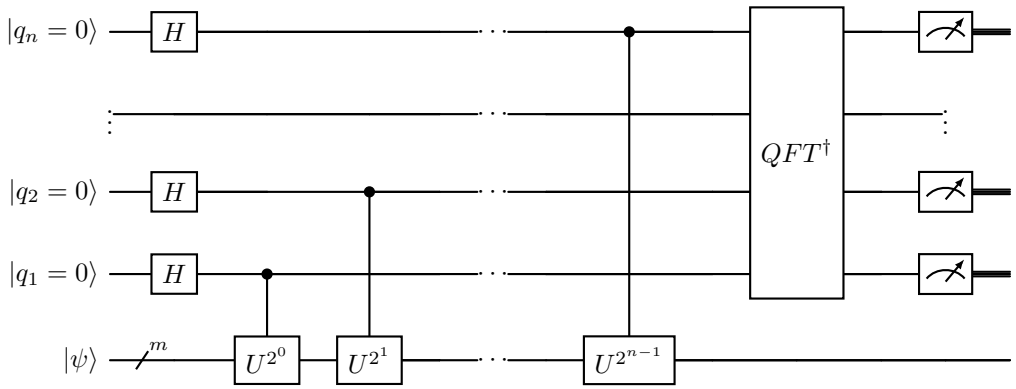
The quantum circuit implementing the QFT is shown bellow with $R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$:



1. Write a function `QFT(n,swaps=True)` that takes the number of qubits n and a boolean `swaps` as inputs, which implements the QFT. The boolean `swaps` indicates if one should include the swap step at the end of the circuit. The controlled-rotations can be implemented with U1Gate or by creating the gate with a numpy array.

1.9 Quantum Phase Estimation

Quantum Phase Estimation (QPE) is a very usefull subroutine for finding the eigenvalues of a unitary U. As it is unitary, its eigenvalue are of the form $e^{2i\pi\varphi}$. Using QPE, one can find the phase φ as the outcome of the measurements is $\varphi 2^n$ with high probability. We recall that if $|\psi\rangle$ is an eigenvector of U, then $U|\psi\rangle = e^{2i\pi\varphi}|\psi\rangle$ for $\varphi \in \mathbb{R}$. It uses two quantum registers, the eigenvector $|\psi\rangle$ (acting on m qubits) and a working register of n qubits initialized to $|0\rangle$. Here is the quantum circuit implementation of QPE:



1. Implement the QPE circuit with $m = 1$ by writing a function `QPE(n,psi,U)` that takes the number of qubits n initialized to $|0\rangle$, `psi` a unitary to initialize the state of the bottom qubit, and a unitary U as inputs.

2. To check if your implementation is correct, we are going to estimate the value of π using QPE. Let $|\psi\rangle$ be an eigenvector of U, therefore $U|\psi\rangle = e^{2i\pi\varphi}|\psi\rangle$. Using QPE we can easily get φ since $\varphi = \frac{\delta}{2^n}$, where δ is the outcome of the measurement. Let's pick U such that $U = \begin{bmatrix} 1 & 0 \\ 0 & e^i \end{bmatrix}$. Hence, we have $U|1\rangle = e^i|1\rangle$, with $|1\rangle$ and e^i being eigenvector and eigenvalue of U. We recall that any eigenvalue of a unitary matrix can be written as $e^{2i\pi\varphi}$, thus $2\pi\varphi = 1$, i.e. $\pi = \frac{1}{2\varphi}$.
   The protocol is summarized bellow:

   - Compute the QPE circuit with $|\psi\rangle = |1\rangle$ and $U = \begin{bmatrix} 1 & 0 \\ 0 & e^i \end{bmatrix}$
   - Get the outcome δ by measuring the n qubit register

- Compute $\pi = \frac{1}{2\frac{\delta}{2^n}}$

(a) Write a function `estimate_pi(n)` that uses QPE to return an estimation of π.

(b) Plot the quantum estimation of π as a function of the number of qubits n with $n \in [2, \dots, 15]$.

1.10 Quantum Addition: Draper Adder

With a quantum computer it is possible to perform additions by using the QFT. Given two quantum register of n qubits each, $|a\rangle$ and $|b\rangle$, we wish to compute the sum of them and to store the result in $|b\rangle$. Let's see how this algorithm works. Initially the state of the system is:

$$|\psi\rangle = |a\rangle \otimes |b\rangle$$

After applying the QFT on the second register it becomes:

$$\begin{aligned}|\psi\rangle &= |a\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.b_n \dots b_1)}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.b_{n-1} \dots b_1)}|1\rangle) \otimes \cdots \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.b_1)}|1\rangle) \\ &= |a\rangle \otimes |\phi_n(b)\rangle \otimes |\phi_{n-1}(b)\rangle \otimes \cdots \otimes |\phi_1(b)\rangle\end{aligned}$$

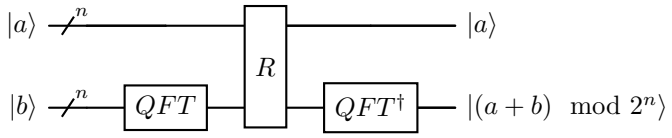
Then, we perform some rotations on $|b\rangle$ controlled by $|a\rangle$:

$$\begin{aligned}|\psi\rangle &= |a\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.a_n \dots a_1 + 0.b_n \dots b_1)}|1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.a_{n-1} \dots a_1 + 0.b_{n-1} \dots b_1)}|1\rangle) \otimes \cdots \otimes \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i(0.a_1 + 0.b_1)}|1\rangle) \\ &= |a\rangle \otimes |\phi_n(a+b)\rangle \otimes |\phi_{n-1}(a+b)\rangle \otimes \cdots \otimes |\phi_1(a+b)\rangle\end{aligned}$$

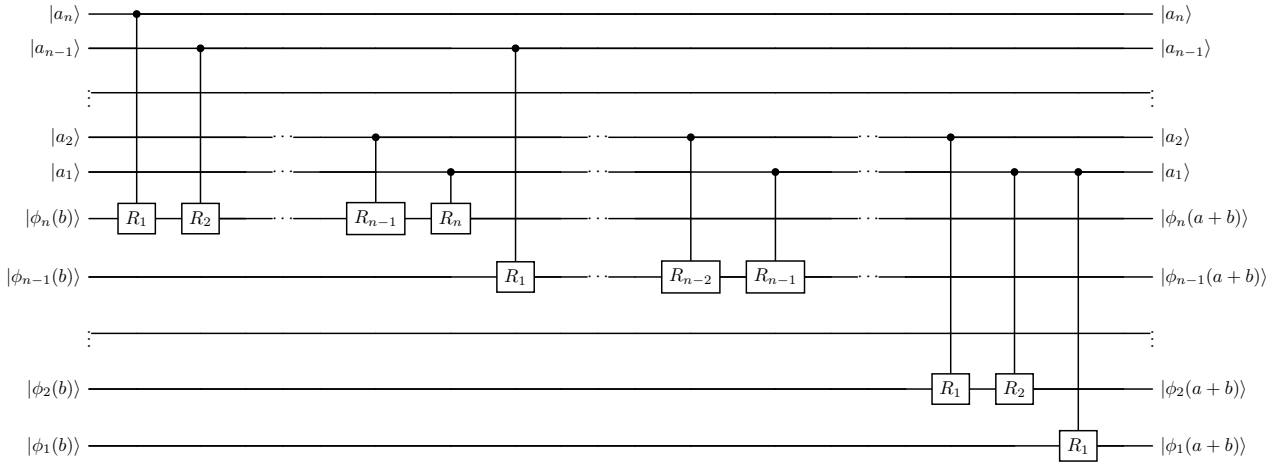
Finally, as the information is now encoded in the phases, we go back to the computational basis by applying the inverse QFT on $|b\rangle$:

$$\begin{aligned}|\psi\rangle &= |a\rangle \otimes |(a+b)_n\rangle \otimes |(a+b)_{n-1}\rangle \otimes \cdots \otimes |(a+b)_1\rangle \\ &= |a\rangle \otimes |(a+b) \mod 2^n\rangle\end{aligned}$$

The quantum circuit representation of the algorithm is shown bellow:



The controlled-rotations R step is shown bellow with $R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix}$:



Implement the Draper Adder. The controlled-rotations can be implemented with U1Gate or by creating the gate with a numpy array. **To make it work you must not do the swaps in the QFT.** Therefore, you have to set `swaps=False` when calling the `QFT` function. We will do the implementation in two steps;

- The initialization of the registers $|a\rangle$ and $|b\rangle$.
- The implementation of the Draper Adder.

1. Write a function `Draper_Adder(x,y)` that takes two integers x and y as inputs and compute their sum using the Draper Adder. The first quantum register $|a\rangle$ is going to store the value of x and $|b\rangle$ the value of y. Start by initializing the two quantum registers correctly with the corresponding NOT gates. We recall that $|x = a_n \dots a_1\rangle$ and $|y = b_n \dots b_1\rangle$ where the index 1 corresponds to the least significant qubit. Then, implement the Draper Adder with no swaps for the QFT steps. Note that your function should work for any arbitrary non negative integers values of x and y.

   You can use the line `bin(i)[2:].zfill(n)` to get the binary string representation of integer i on n bits, setting n to 0 gives the binary representation of i with the minimum number of bits. Recall that in Python the first element $s[0]$ of a bit string s is the leftmost bit, you can reverse a string s with `reversed(s)` or `s = s[::-1]`.

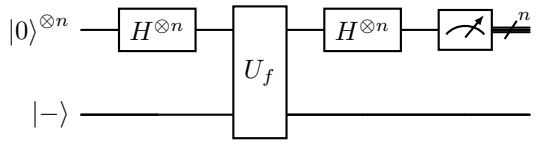
You can verify the correctness of your implementation with the function:

```
# Function to verify if the Draper Adder is correctly implemented
def verification(x,y):
    n = max(len(bin(x)[2:].zfill(0)),len(bin(y)[2:].zfill(0)))
    outcome = run(Draper_Adder(x,y),plot=False)
    return bin((x+y)%2**n)[2:].zfill(n) == list(outcome.keys())[0]
```

1.11 Bernstein-Vazirani Algorithm

In the Bernstein-Vazirani problem, one has to guess a secret number encoded on n bits. Classicaly the most efficient method would require n trials. Indeed, one would first do the logical operation AND between 1 and the first bit, if this bit is 1 the result would be 1, otherwise 0. Repeating this procedure for the n bits, one is able to guess the secret number with a time complexity $O(n)$. However, with a quantum computer one can solve this problem with only 1 trial using Bernstein-Vazirani Algorithm, hence with a constant complexity $O(1)$. Let's formalize the problem.

Given a function $f : \{0,1\}^n \to \{0,1\}$ acting as an oracle where $f(x)$ is a scalar product between x and the secret bit string $s \in \{0,1\}^n$ modulo 2, with $f(x) = x \cdot s = x_1 s_1 \oplus \dots \oplus x_n s_n$. The problem is to find the value of s. The quantum circuit implementing Bernstein-Vazirani Algorithm is shown bellow, where $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$:



Let's compute the effect of the algorithm on an input state. We start with the initial state and we only focus on the n qubits of input:

$$|\psi\rangle = |0\rangle^{\otimes n}$$

After applying the Hadamard tower on the first n qubits, we obtain:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$$

The oracle U_f acts on $|x\rangle$ as $U_f |x\rangle = (-1)^{f(x)} |x\rangle$. The superposition becomes:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} |x\rangle$$

Reapplying the Hadamard tower, we get:

$$\begin{aligned} |\psi\rangle &= \frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)+x\cdot y} |y\rangle \\ &= |s\rangle \end{aligned}$$

The final state is $|s\rangle$ because for a given value of y we have:

$$\begin{aligned} |\psi\rangle &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)+x\cdot y} \\ &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot s+x\cdot y} \\ &= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot (s\oplus y)} \\ &= 1 \text{ if } s\oplus y = 0, 0 \text{ otherwise} \end{aligned}$$

As $s \oplus y = 0$ only if $s = y$, the only non zero amplitude is that of $|s\rangle$.

1. Write a function that generates a random bit string of length n.
2. Let's implement the oracle U_f. To do so we are going to use sequence of CNOT gates. Given a secret bit string $s = s_n \dots s_1$, one aims to apply a CNOT gate on the last qubit $|-\rangle$ controlled by $|x_i\rangle$ if $s_i = 1$. Recall that in Python the first element $s[0]$ of a bit string s is the leftmost bit, you can reverse a string s with `reversed(s)` or `s = s[::-1]`. As we do not use this convention in our encoding, do not forget to reverse the bit string s before entering the loop.
3. Merging everything together, write a function `Bernstein_Vazirani(n)` that takes a number n as input and implements the Bernstein-Vazirani Algorithm.
4. Run your code on IBM real quantum device.