a

February 8, 2025

[ ]:
```
pip install git+https://github.com/MIROptics/ECC2025.git
```

[13]:
```
import numpy as np
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from ECC2025.testing import test_3a, test_3b, test_3c
```

0.0.1 Algoritmo de Shor

El **algoritmo de Shor** es uno de los algoritmos cuánticos más conocidos. Publicado en 1994 por Peter Shor en su artículo Algorithms for Quantum Computation: Discrete Logarithms and Factoring, su importancia recae en su utilidad para factorizar, ya sea números primos u otras estructuras que poseen estructura similar, y **resolver el problema del logaritmo discreto en un tiempo polinomial**, una mejora considerable del tiempo subexponencial de los mejores algoritmos clásicos. Como los protocolos criptográficos de clave pública más utilizados actualmente (RSA, Diffie-Hellman, Curvas elípticas) están basados en la dificultad de resolver el problema del logaritmo discreto en computadores clásicos, si se llegara poder implementar el algoritmo de Shor en una cantidad considerable de qubits, estos protocolos criptográficos quedarían obsoletos.

La cantidad necesaria aproximada para romper una clave RSA de 2048 bits con el algoritmo de Shor es de 4000 qubits lógicos, una cantidad que sin tener corrección de errores se ve difícil de alcanzar en el corto plazo, aún así las empresas de ciberseguridad y gobiernos están proponiendo cambios y nuevas estandarizaciones para hacer frente a la posibilidad de poder implementarlo.

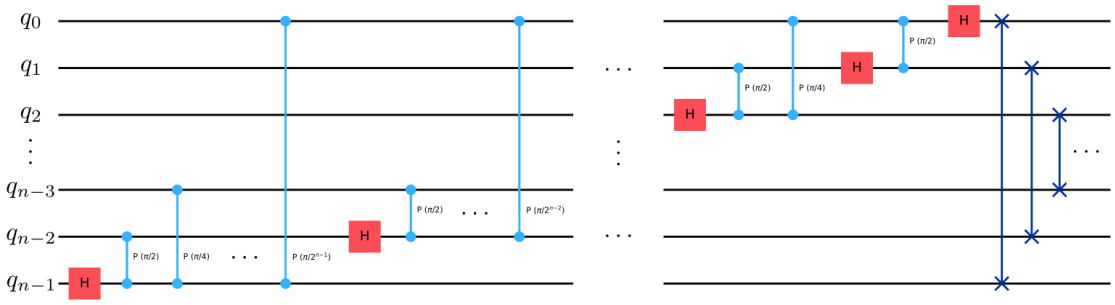
0.0.2 Transformada Cuántica de Fourier

La subrutina más importante del algoritmo de Shor es el algoritmo de estimación de fase, y para implementar este algoritmo, primero necesitamos la **transformada cuántica de Fourier** (QFT, Quantum Fourier Transform). La QFT es la versión cuántica de la **transformada discreta de Fourier**, que actúa sobre los amplitudes de probabilidad de un estado cuántico. Esta operación permite transformar un estado cuántico de tal manera que las frecuencias de las componentes de la superposición puedan ser extraídas con mayor facilidad.

Matemáticamente, la transformada cuántica de Fourier para un estado cuántico $|x\rangle$de$n$qubits es una superposición de los estados$|y\rangle$, dada por la expresión:

$$|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i \frac{xy}{2^n}} |y\rangle, \tag{1}$$

Lo que está representado por el siguiente circuito:



Como nuestro objetivo final es implementar el algoritmo de estimación de fase, comenzaremos creando una función para implementar la transformada cuántica de Fourier.

Desafío: Cree una función llamada `Fourier(n)` cuyo argumento es el número de qubits ny que retorne el circuito cuántico de la transformada de Fourier de$n$ qubits.

[14]:
```
def Fourier(n):
    """
    n : Numero de qubits
    """
    qc = QuantumCircuit(n)

    ## Escribe tu codigo acá ###
    for i in range(n):
        qc.h(n-1-i)
        for j in range(n-1-i):
            qc.cp( np.pi/2**(j+1), n-2-i-j, n-1-i )

    for i in range(n//2):
        qc.swap(i,n-1-i)
    ######

    return qc
```

[15]:
```
qc = Fourier( 4 )
```

[16]:
```
test_3a( Fourier )
```

```
Felicidades, tu solución es correcta!
```

0.0.3 Potencia de una operación unitaria

En concreto, la transformada cuántica de Fourier es utilizada por el algoritmo de estimación de fase para estimar una fase codificada en una **operación unitaria** U. El algoritmo necesita aplicar potencias de Uen la formaU^mpara poder extraer la información de la fase con precisión. Estas potencias permiten construir una superposición que, cuando se combina con la QFT, revela la fase asociada a un autovector de$U$.

Crearemos ahora la función que construye U^mcon unϕ especifico para poder utilizarla en nuestra implementación más adelante.

Desafío: Complete la siguiente función para que implente el circuito de U^m controlada, donde

$$U = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i\phi} \end{pmatrix}, \tag{2}$$

con $\phi = 0.375$.

HINT: Busque la puerta cuántica "control-phase".

[17]:
```
def U_to_n( power ):
    """
    power : potencia a la que elevar U
    """
    qc = QuantumCircuit( 2 )
    ###
    angle = 2*np.pi*0.375

    for i in range(power):
        qc.cp( angle, 0,1 )
    ###
    return qc
```

[18]:
```
test_3b( U_to_n )
```

```
Felicidades, tu solución es correcta!
```

0.0.4 Algoritmo de estimación de fase

Dada una operación unitaria Uy un autovector$|\psi\rangle$, el **algoritmo de estimación de fase** permite estimar con una alta precisión la fase ϕasociada al autovalor de$|\psi\rangle$, es decir ϕ tal que:

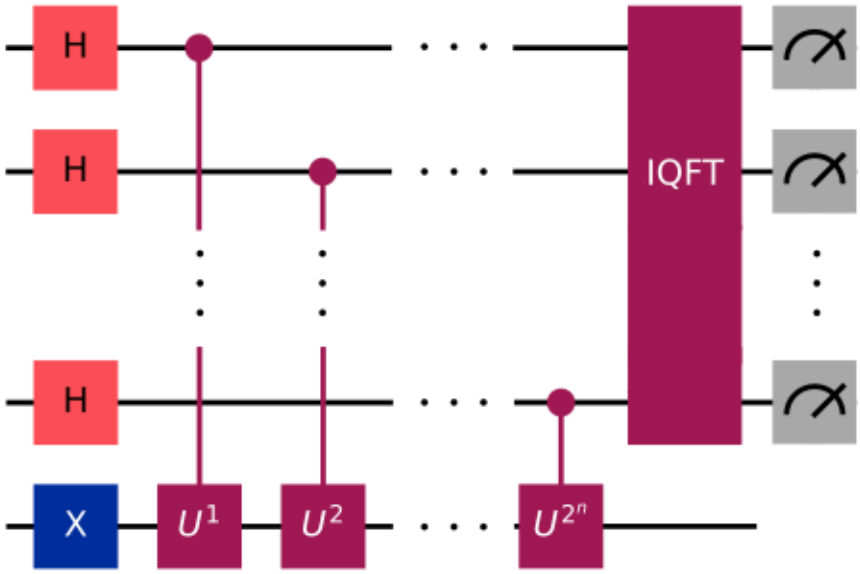
$$U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle. \tag{3}$$

El proceso consiste en:

1. Preparar un sistema de $n$qubits auxiliares en estado de superposición y un qubit en el estado$|\psi\rangle$.
2. Aplicar las potencias de la operación unitaria, U^m, a $|\psi\rangle$ de forma controlada para codificar la fase en los qubits auxiliares.
3. Ejecutar la transformada cuántica de Fourier inversa en los qubits auxiliares.
4. Medir los qubits auxiliares para obtener el resultado binario que nos dará la estimación de la fase ϕ.

Ahora, como anteriormente ya creamos la función para implementar la QFT y U^m, llegó el momento de implementar el algoritmo de estimación de fase para nuestro caso particular de ϕ.

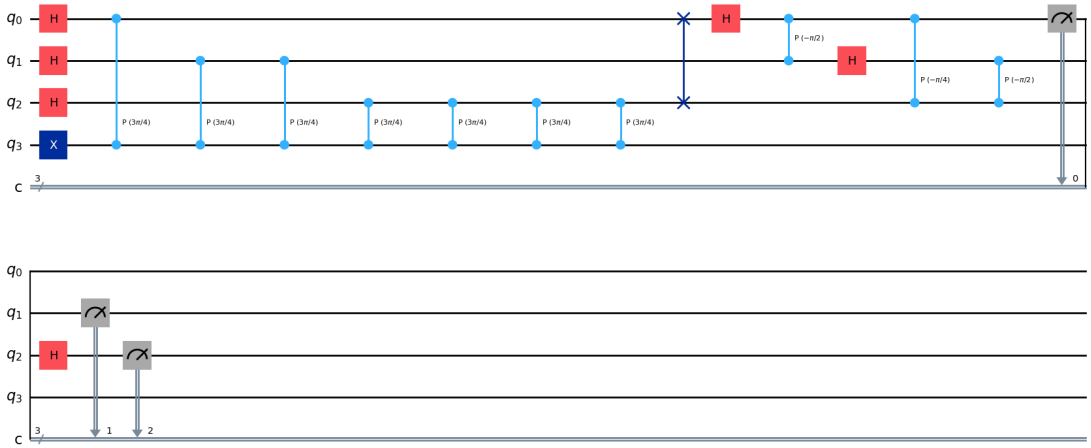
Desafío: Utilice las funciones anteriores para completar la siguiente función que implementa el algoritmo de estimación de fase, es decir, el siguiente circuito. Note que IQFT es la operación inversa de la transformada de Fourier.



[19]:
```
def QuantumPhaseEstimation( n ):
    """
    n : Numero de qubits
    """
    qc = QuantumCircuit( n+1, n )

    ########
    qc.h(range(n))
    qc.x(n)
    for j in range(n):
        qc.compose( U_to_n(2**j), qubits=[j,n], inplace=True )

    qc.compose( Fourier(n).inverse(), qubits=range(n), inplace=True )

    #######
    qc.measure( range(n), range(n) )

    return qc
```

[20]:
```
qc = QuantumPhaseEstimation(3)
qc.draw('mpl')
```

[20]:



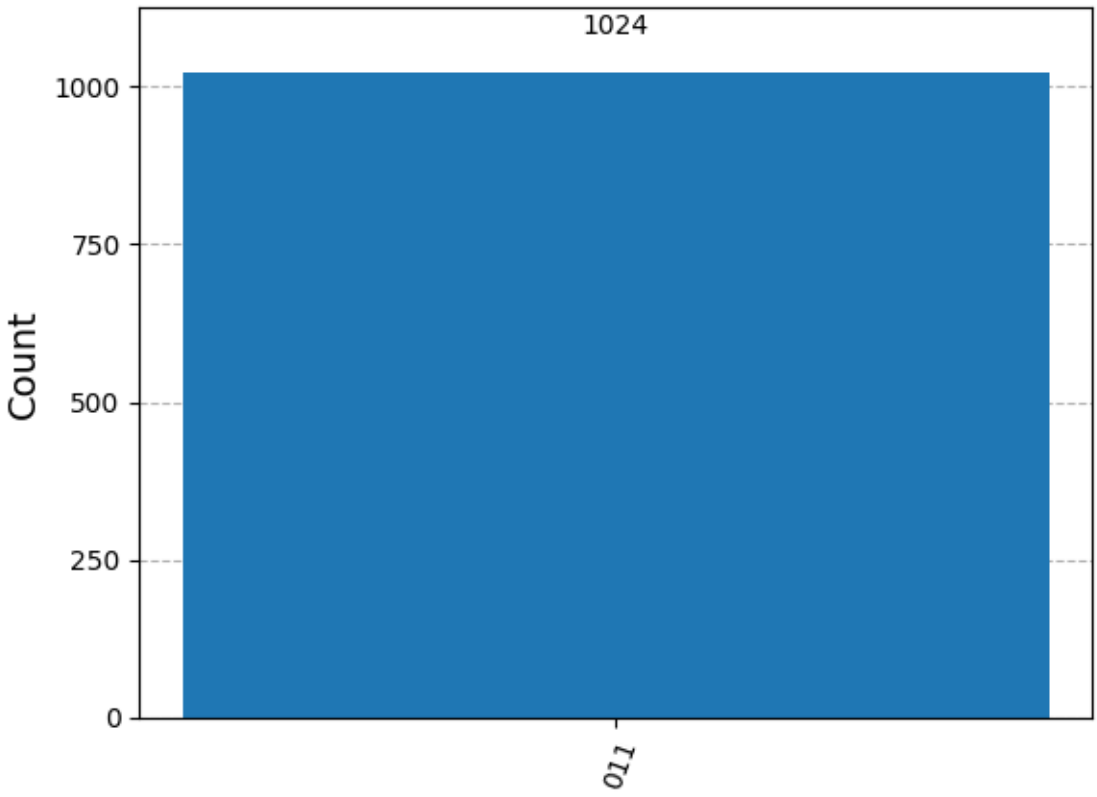
Si sus funciones están correctas, al simular la ejecucción del circuito se debería obtener el resultado $0110\cdots0$con mayor probabilidad, para un número de qubits$n \geq 3$. Esto es porque el ángulo estimado por el algoritmo es

$$\tilde{\phi} = \frac{\text{Int}(0110\cdots0)}{2^n} = \frac{2^{n-2} + 2^{n-3}}{2^n} = 0.375 \tag{4}$$

que es el ϕque utilizamos cuando creamos la función para obtenerU^m.

[21]:
```
backend = AerSimulator()
job = backend.run( qc )
counts = job.result().get_counts()
plot_histogram( counts )
```

[21]:



[22]:
```
test_3c( QuantumPhaseEstimation )
```

```
Felicidades, tu solución es correcta!
```

[ ]:
```

```