El Algoritmo de Shor

Explicación del algoritmo e implementación con 2n+3 qubits

Autor:
David Castaño Bandín
SCBI (Universidad de Málaga)

Co-autores:
Alejandro Cano
UNICAN
Javier Sánchez
Cénits
Alejandro Cano
BSC-CNS

Revisores:
Javier Mas Solé
Universidad de Santiago de Compostela
Andrés Gonzalez Tajo
CESGA

27/03/2023

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia

Índice

1. Introducción.

1.1. Criptografía y factorización.

El algoritmo de Shor es uno de los algoritmos de computación cuántica más conocidos debido a que es mejor en su tarea que cualquier algoritmo de computación clásica conocido hasta la fecha. Además, resuelve un problema que tiene una aplicación práctica directa: factorizar un número

Para entender porque este algoritmo es tan importante debemos hablar primero de criptografía, en concreto, del encriptado de transmisiones a través de Internet mediante el método de clave pública y clave privada (**criptografía asimétrica**). La explicación conceptual de esta forma de encriptar es simple. Cuando quieres, por ejemplo, acceder a la aplicación de tú banco tienes que ingresar las claves de acceso (DNI y pin), estas se mandan por Internet hasta la sede de tu banco, el cual verifica que son correctas y te da acceso. El problema radica precisamente en que la conexión se hace mediante Internet, con lo cual el mensaje con las claves de acceso puede ser interceptado. La solución a este problema es que el mensaje que el cliente envía esté encriptado y que solo tu banco pueda desencriptar el mensaje.

El método de encriptación más usado en Internet es la ya mencionada criptografía asimétrica. En este tipo de encriptado el receptor del mensaje (en nuestro ejemplo, el banco) genera dos claves dependientes entre sí, una la publicará al exterior (clave pública) y otra solo la conocerá él (clave privada). Si un receptor quisiera recibir un mensaje encriptado, bastaría con que publicase su clave pública de forma que cualquiera que quiera mandarle un mensaje, pueda usarla para encriptar el mismo. Sin embargo, la clave privada solo es conocida por el receptor del mensaje, y se usa para desencriptar. Puede decirse que la clave pública es como un candado y la clave privada es la llave. Cualquiera puede cerrar el candado, pero solo el que tiene la llave puede abrirlo. (Aquí podemos poner un poco más como se encripta/desencripta de forma sencilla...)

El punto importante aquí es que, la clave privada son dos números primos (de gran tamaño, cientos de cifras), y la clave pública es la multiplicación de estos dos números. La solidez de este método de cifrado (RSA) radica en el hecho de que si tenemos dos números primos multiplicarlos es muy fácil, pero si tenemos la multiplicación de los mismos (la clave pública) hallar cuales son los dos números con los que se construyó (factorizar el número en sus elementos primos) es extremadamente difícil. Como es esperable, cuanto más larga es la clave, más tiempo se tarda en factorizarla. El problema radica específicamente en que el tiempo que se requiere crece **exponencialmente** con el número de bits. Para las longitudes de clave que se manejan actualmente, incluso con los mejores superordenadores se tardarían cientos o miles de años en hallar los factores.

La tremenda potencia y aplicabilidad del algoritmo de Shor es que convierte este problema de complejidad exponencial en el número de bits para un computador clásico, en un problema de complejidad polinómica para un computador cuántico. Es decir, con el algoritmo de Shor el tiempo requerido para factorizar un número crece **polinómicamente** con el número de cifras del número. De esta forma, si se llega a tener un ordenador cuántico con suficientes qubits como para aplicar este algoritmo a números de la longitud de clave que se usa actualmente, se podrían factorizar y hallar la clave privada en un tiempo razonable para la escala humana. El algoritmo de Shor tiene el potencial de romper la criptografía asimétrica y hacer vulnerables las comunicaciones a través de la red, pero estamos muy lejos de tener un ordenador cuántico capar de implementarlo a la escala requerida. Se estima que se necesitarían del orden del millón de qubits, mientras que actualmente (año 2022) los ordenadores cuánticos más grandes andan por el orden de cientos de qubits.

1.2. Algoritmo de factorización.

El algoritmo de Shor se basa en el hecho de poder reducir un problema de factorización a uno de **period (o order) finding (hallar el periodo -orden- de una función)**. Antes de hablar de nada cuántico, vamos a ver como sería la estructura general de un algoritmo de factorización de esta forma, tal y como se describe en el Nielsen-Chuang [1], comentando en que punto entra la computación cuántica para acelerarlo.

Lo primero es introducir la noción de **números coprimos**: dos números a y b son coprimos si su máximo común divisor es 1, esto es, $\gcd(a, b) = 1$. Es decir, dos números coprimos solo comparten como divisor común el 1.

Los pasos reducir un problema de factorización en uno de period finding son los siguientes. Sea N el número que queremos factorizas

1. Si N es par, devolver el factor 2.
2. Determinar si $N = p^b$ para los enteros $p \geq 1$ y $b \geq 2$, y si es así, devolver el factor p (puede hacerse en un tiempo polinómico).
3. Elegir un número entero aleatorio a tal que $1 < a \leq N - 1$. Usando el algoritmo de Euclides, determinar si $\gcd(a, N) > 1$. Si lo es, devolver el factor $\gcd(a, N)$.
4. Si $\gcd(a, N) = 1$, calculamos el periodo r de la función $f(x) = a^x \bmod N$.
5. Si r es impar o r es par pero $a^{r/2} \bmod N = -1$, volvemos al paso 3. Sino, calculamos $\gcd(a^{r/2} - 1, N)$ y $\gcd(a^{r/2} + 1, N)$. Probamos a ver si uno de estos dos es un factor no-trivial de N, y devolvemos el mismo si lo es.

Todos los pasos de este algoritmo, excepto el **paso 4**, se pueden implementar en un ordenador clásico y resolverse en un tiempo polinómico. Esto es debido a que para calcular el máximo común divisor puede usar el Algoritmo de Euclides [2], el cual resuelve el problema en un **tiempo polinómico** (se puede calcular en un tiempo razonable).

El paso complicado y que, por lo menos hasta la fecha, no hay ninguna forma de implementarlo en un tiempo polinómico (se implementa en un **tiempo exponencial**) en un ordenador clásico es el **paso 4**, hallar el periodo de la función. Sin embargo, este paso puede implementarse un ordenador cuántico en un tiempo polinómico. Tenemos pues que la forma óptima de factorizar un número consiste en implementar los pasos 1, 2, 3 y 5 en un ordenador clásico, y el paso 4 en un ordenador cuántico.

1.3. Explicación cualitativa

Vamos a intentar entender de forma cualitativa porqué calculando el periodo de una función se pueden hallar los factores de un número. En la sección 1.4 veremos un poco más en detalle las afirmaciones que se hacen en esta sección.

La función que nos interesa es la siguiente

$$f(x) = a^x \bmod N \tag{1.1}$$

donde a y N son enteros positivos mayores que 1, siendo además a ¡ N y no teniendo factores comunes (es decir, cumpliéndose $\gcd(a, N) = 1$). La operación ($z \bmod N$) a lo que se refiere es a quedarnos con el **resto** de dividir el número que z por N. Esta función se denomina **exponenciales moduladas**, se encaja dentro de la **aritmética modular** y si se cumplen las condiciones anteriores esta función

es periódica. Denominaremos r al valor del periodo de la función $f(x)$, es decir, r es el mínimo valor entero para que se cumple:

$$f(x+r) = f(x).$$

Este se puede calcular mediante un circuito cuántico.

Una vez se tiene el periodo r, si este es par (sino hay que probar con otro valor de a) se pueden calcular los factores de N. Esto es debido a que

$$a^r \bmod N = 1$$

con lo cual

$$(a^r - 1) \bmod N = 0$$

Con lo cual, N debe ser un divisor de $a^r - 1$. Si r es par (sino hay que probar con otro valor de a), podemos escribir:

$$a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)$$

Entonces tenemos una alta probabilidad de que el **máximo común divisor** de N y $a^{r/2}-1$ o $a^{r/2}+1$ sea un factor propio de N.

1.4. Formalismo matemático

Veamos un poco el formalismo matemático detrás de las afirmaciones de la sección anterior.

1.4.1. Periodicidad de $f(x)$

Demostrar que, dada la condición $\gcd(a, N) = 1$, la función $f(x) = a^x \bmod N$ es periódica no es fácil, pues se necesitan plantear varios teoremas y la explicación se hace árida (puede verse el Appendix 4 de [1]). Aquí vamos a ver una explicación más simple partiendo del siguiente resultado (sin demostrarlos):

- Dada la función $f(x) = a^x \bmod N$, si se cumple que $\gcd(a, N) = 1$, tenemos que para algún valor entero $z > 0$ se cumple $f(z) = a^z \bmod N = 1$.

Vemos a ver ahora que este el menor valor $z > 0$ para el cual se cumple $f(z) = a^z \bmod N = 1$ será el periodo de la función. Denominaremos a este valor r, es decir, r será el primer valor (mayor que cero) para el cual se cumple $f(r) = 1$. Tenemos que

$$a^0 = 1 \rightarrow f(0) = a^0 \bmod N = 1 = f(r).$$

En el momento en el que llegamos a un exponente r tal que $a^r \bmod N = 1$ podemos pues escribir

$$a^r = \alpha N + 1$$

con lo cual

$$f(r+z) = a^{r+z} \bmod N = a^r a^z \bmod N = (\alpha N + 1)a^z \bmod N = \tag{1.2}$$

$$= \alpha N a^z \bmod N + a^z \bmod N = a^z \bmod N = f(z) \tag{1.3}$$

Hemos visto pues que $f(x)$ es periódica.

1.4.2. Factores de N a partir del periodo r

Para entender como pasar del periodo r de nuestra función a tener los factores de N nos hace falta conocer un par de teoremas, ambos presentes en [1].

Teorema 5.2 del Nielsen-Chuang: Supongamos que N es un número compuesto de L bits, y x es una solución no trivial de la ecuación $a^2 \bmod N$ en el rango $1 \leq a \leq N$, esto es, ni $x \bmod N = 1$ ni $x \bmod N = N - 1 \bmod N = -1 \bmod N$. Entonces, uno de $\gcd(x-1, N)$ y $\gcd(x+1, N)$ es un factor no trivial de N que se puede calcular usando $\mathcal{O}(L^3)$ operaciones.

Demostración: Ya que $x^2 \bmod N = 1 \rightarrow (x^2 - 1) \bmod N = 0$, debe de cumplirse que N divida a $(x^2-1) = (x+1)(x-1)$, con lo cual N debe de tener un factor común con $(x+1)$ o con $(x-1)$. Como por suposición tenemos que $1 < x < N - 1$, con lo cual $x - 1 < x + 1 < N$, de lo cual podemos ver que el factor común no puede ser el propio N. Usando el Algoritmo de Euclides [2] podemos calcular $\gcd(x-1)$ y $\gcd(x+1)$, y con lo cual obtener un factor no trivial de N, usando $\mathcal{O}$ operaciones.

Teorema 5.3 del Nielsen-Chuang: Supongamos $N = p_1^{\alpha_1} \dots p_m^{\alpha_m}$ es la descomposición en factores primos de un entero impar positivo. Sea x un número entero elegido uniformemente al azar, sujeto a la restricción $1 \leq x \leq N - 1$ y x coprimo de N. Sea r el periodo de $x \bmod N$. Entonces

$$p(r \text{ es impar y } x^{r/2} \bmod N = -1) \geq 1 - \frac{1}{2^m}$$

esto es, la probabilidad de hallar un r impar y que cumpla $x^{r/2} \bmod N = -1) \geq 1$ es mayor que 1 - $1/2^m$.

Nota

En nuestro caso el teorema 5.2 se aplica con $x = a^r$ y el teorema 5.3 con $x = a$.

Los teoremas 5.2 y 5.3 pueden combinarse para dar un algoritmo que, con alta probabilidad, devuelve un factor no trivial de cualquier compuesto N . Todos los pasos del algoritmo pueden realizarse de forma eficiente en un ordenador clásico, excepto (por lo que se sabe hoy en día) una "subrutina"de búsqueda de periodo que utiliza el algoritmo. Repitiendo el procedimiento podemos encontrar una factorización prima completa de N.

1.5. Hallar el periodo en un ordenador cuántico

Como hemos comentado, el paso 4 descrito en la sección 1.2 (buscar el periodo de $f(x)$) se puede implementar en un ordenador cuántico. Para ello lo que se hace es reducir el problema de la búsqueda de periodo a un problema de **Estimación de Fase Cuántica (Quantum Phase Estimation)**, que a su vez usa la **Transformada de Fourier Cuántica (Quantum Fourier Transform)**.

En la sección 2 hablaremos de la transformada de Fourier Cuántica, el la sección 3 del algoritmo de estimación de fase cuántica para pasar a ver en las siguientes secciones implementaciones del algoritmo de Shor.

2. Transformada de Fourier Cuántica (Quantum Fourier Transform - QFT)

Veremos aquí simplemente la fórmula general de la QFT aplicada a un conjunto de qubits (pues es lo único que necesitaremos para entender la Estimación de Fases Cuántica) y el circuito que la implementa. Para más información sobre la QFT, puede verse, por ejemplo, la explicación del [3].

2.1. Fórmula

Dado un estado de n qubits $|x\rangle = |x_1 x_2 \dots x_n\rangle$ donde x_1 es el bit más significativo, la transformada de Fourier cuántica QFT a n quibits es de la forma:

$$\begin{aligned} QFT\,|x\rangle &= \frac{1}{2^{n/2}} \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n}\,|y\rangle \qquad \text{donde } |y\rangle = |y_1 y_2 \dots\rangle\,,\ y/2^n = \sum_{k=1}^{n} y_k/2^k \\ &= \frac{1}{2^{n/2}} \left(|0\rangle + e^{\frac{2\pi i}{2}x}\,|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^2}x}\,|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}\,|1\rangle\right) \otimes \left(|0\rangle + e^{\frac{2\pi i}{2^n}x}\,|1\rangle\right) \end{aligned}$$

La transformada de Fourier cuántica inversa de un estado de n-qubit $|y\rangle$ solo difiere de la transformada directa en el signo de la exponencial:

$$QFT^{-1}\,|y\rangle = \frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} e^{-2\pi i y x/2^n}\,|x\rangle$$

2.2. Circuito

En la Fig. 1 se ve el circuito que implementa la trasformada de Fourier cuántica

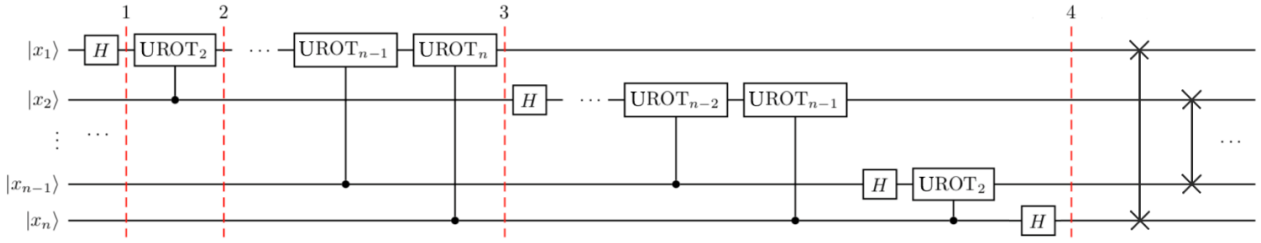


Figura 1: Implementación de la QFT (con el convenio de Qiskit)

Vemos que se aplican dos tipos de puertas, las puertas de Hadamard y unas puertas $UROT_k$. Vamos llamar a la versiones controladas de estas últimas como $CROT_k$, es decir

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix}$$

Esta puerta $UROT_k$ es una puerta de rotación de la forma

$$UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{bmatrix}$$

Es decir, la expresión completa de la puerta $CROT_k$ es

$$CROT_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{bmatrix}$$

Nota importante

Véase que este circuito sigue el convenio más extendido en la literatura, es decir, aquel en el que el **bit más significativo es el de más arriba** y el bit menos significativo el de más abajo. **Qiskit sigue el convenio contrario**. Esto es, dado un estado

$$|x\rangle = |a_{n-1}a_{n-2}\dots a_1 a_0\rangle$$

que podemos escribirlo en la base decimal como

$$x = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_0 2^0$$

tenemos que al escribir los circuitos

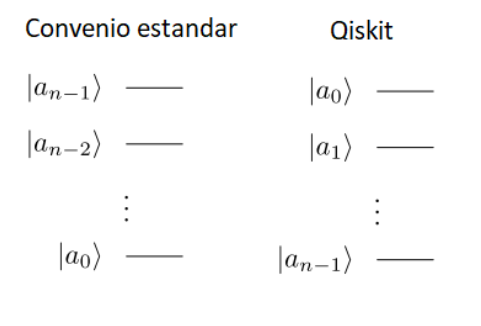


Figura 2: Convenio estándar y convenio de Qiskit

En este caso, para escribir el circuito en el convenio de qiskit lo que habría que hacer es invertir los qubits que controlan las puertas U, es decir, aplicar $U^{2^{t-1}}$ controlada por el último qubit, $U^{2^{t-2}}$ controlada el penúltimo qubit, ... y U^{2^0} controlada por el primer qubit.

3. Estimación de fase cuántica (Quantum Phase Estimation)

La estimación de fase cuántica es una pieza fundamental de algoritmos más complejos (como es el caso que estamos viendo del algoritmo de Shor).

Este algoritmo sirve para calcular los autovalores de un **operador unitario**. Como sabemos, los operadores unitarios son los únicos que podemos aplicar a un estado cuántico (son las puertas en los circuitos), pues son los únicos que preservan la normalización de los estados cuánticos, es decir, que las probabilidades sumen la unidad. Esto se manifiesta en que los autovalores de los operadores unitarios tiene módulo 1, es decir, son **fases**. Dado un operador unitario U y un autovector $|\psi\rangle$ del mismo, tenemos:

$$U\,|\psi\rangle = e^{2\pi i\theta}\,|\psi\rangle$$

El algoritmo lo que hace es estimar el valor de θ

3.1. Circuito

En la Fig. 3 podemos ver el circuito que implementa el algoritmo de estimación de fases. Podemos ver que el circuito consta de tres parte:

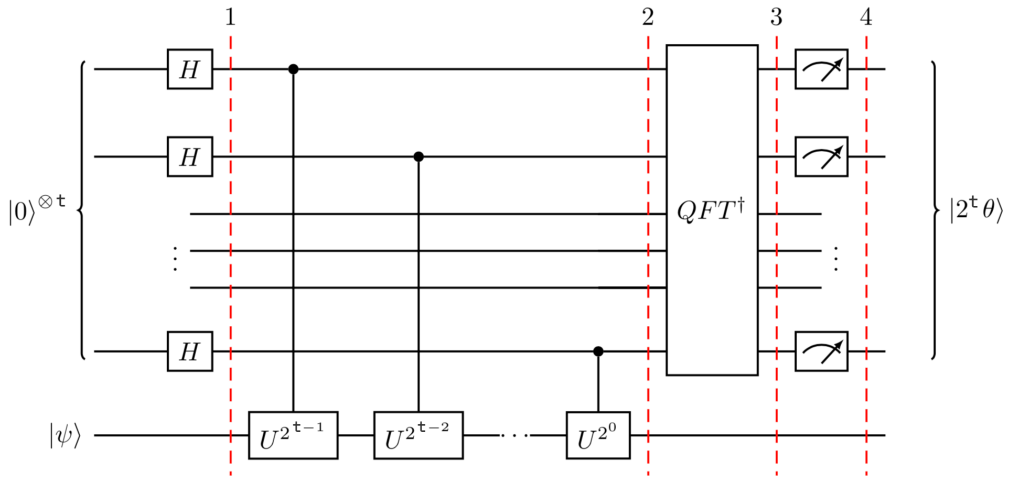


Figura 3: Implementación del algoritmo de estimación de fase cuántica (en el convenio estándar)

3.2. Formulación matemática

Veamos paso a paso que hace el circuito anterior para estimar la fase θ.

1. **Inicialización**: Por un lado tenemos un registro de qubits que forman un autoestado $|\psi\rangle$ del operador U. Por otro lado tenemos un conjunto de $\boldsymbol{t}$ **qubits** que forman un **registro de conteo** donde al final del circuito tendremos almacenado el valor $2^t\theta$:

$$|\psi_0\rangle = |0\rangle^{\otimes t}\,|\psi\rangle$$

2. **Superposición**: Aplicamos una operación de puerta de Hadamard n-bit al registro de conteo:

$$|\psi_1\rangle = \frac{1}{2^{t/2}} \left(|0\rangle + |1\rangle\right)^{\otimes t} |\psi\rangle$$

3. **Operaciones unitarias controladas**: Aplicamos sucesivas veces el operador controlado CU, es decir, aplicar U en el registro objetivo solo si el qubit controlador está en el estado $|1\rangle$. En concreto, aplicamos 2^j veces U en el registro $|\psi\rangle$ controlado por los qubits del registro de conteo. El número de veces 2^j que aplicamos U depende de que qubit es el controlador (si es el bit más significativo tenemos $j = t-1$, para el siguiente $j = t-2$, ..., hasta llegar al bit menos significativo en el cual $j = 0$).

   Como U es unitaria y $|\psi\rangle$ es autovector de U, aplicar 2^j veces U se traduce en:

$$U^{2^j} |\psi\rangle = U^{2^j-1} U |\psi\rangle = U^{2^j-1} e^{2\pi i \theta} |\psi\rangle = \cdots = e^{2\pi i 2^j \theta} |\psi\rangle$$

   Usando la relación

$$CU\left[(|0\rangle + |1\rangle) \otimes |\psi\rangle\right] = |0\rangle \otimes |\psi\rangle + |1\rangle \otimes e^{2\pi i\theta} |\psi\rangle = \left(|0\rangle + e^{2\pi i\theta} |1\rangle\right) \otimes |\psi\rangle$$

   Llegamos a

$$\begin{aligned} |\psi_2\rangle &= \frac{1}{2^{t/2}} \left(|0\rangle + e^{\mathbf{2\pi i}\theta 2^{t-1}}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{\mathbf{2\pi i}\theta 2^1}|1\rangle\right) \otimes \left(|0\rangle + e^{\mathbf{2\pi i}\theta 2^0}|1\rangle\right) \otimes |\psi\rangle \\ &= \frac{1}{2^{n/2}} \sum_{y=0}^{2^t-1} e^{\mathbf{2\pi i}\theta y} |y\rangle \otimes |\psi\rangle \end{aligned}$$

> **Nota**
>
> Podemos ver que si por el registro de conteo entra un estado $|z\rangle = |z_{n-1}, z_{n-2}, \ldots, z_0\rangle$, la salida antes de aplicar la QFT^{-1} es de la forma
>
> $$|z\rangle|\psi\rangle \to |z\rangle U^{z_{t-1}2^{t-1}} U^{z_{t-2}2^{t-2}} \ldots U^{z_0 2^0} |\psi\rangle = |z\rangle U^z |\psi\rangle$$
>
> Es decir, lo que se hace es aplicar z veces el operador U sobre el estado $|\psi\rangle$.

4. **Transformada de Fourier inversa**: Se nos fijamos en detalle vemos que la expresión anterior es igual al resultado de aplicar la transformada de Fourier cuántica (vista anteriormente) un estado de t bits —x¿

$$(QFT\,|x\rangle) \otimes |\psi\rangle = \frac{1}{2^{t/2}} \sum_{y=0}^{2^t-1} e^{2\pi i x y/2^t} |y\rangle \otimes |\psi\rangle$$

   si tomamos $x = 2^t\theta$.

   Lo que podemos hacer es aplicar la transformada de Fourier inversa en el registro de conteo para obtener el valor $2^t\theta$

$$|\psi_2\rangle = \frac{1}{2^{t/2}} \sum_{y=0}^{2^t-1} e^{\mathbf{2\pi i}\theta y}|y\rangle \otimes |\psi\rangle \xrightarrow{QFT_t^{-1}} \frac{1}{2^{t/2}} \sum_{y=0}^{2^t-1} e^{\mathbf{2\pi i}\theta y} \left(QFT^{-1}|y\rangle\right) \otimes |\psi\rangle \Rightarrow$$

$$|\psi_3\rangle = \frac{1}{2^t} \sum_{x=0}^{2^t-1} \sum_{y=0}^{2^t-1} e^{-\frac{2\pi i k}{2^t}(x-2^t\theta)}|x\rangle \otimes |\psi\rangle$$

5. **Medida**. Vemos que la expresión anterior está picada entorno a $x = 2^t\theta$. Para el caso en que $2^t\theta$ es un número entero, medir en la base computacional nos da la fase en el registro de conteo con alta probabilidad:

   $$|\psi\rangle = |2^t\theta\rangle \otimes |\psi\rangle$$

   Para el caso en el que $2^t\theta$ no sea un entero, puede mostrarse que la expresión anterior sigue picada cerca de $x = 2^t\theta$ con una probabilidad mayor del 40

4. Algoritmo de Shor

El algoritmo de Shor se basa en el algoritmo de **Estimación de Fase Cuántica (Quantum Phase Estimation)**, que a su vez usa la **Transformada de Fourier Cuántica (Quantum Fourier Transform)**. El algoritmo de Shor lo que hace es convertir el problema de la factorización de un número en un problema de **encontrar el periodo de una función**, el cual puede ser implementado en un tiempo polinómico.

Para factorizar un número N básicamente el algoritmo de Shor lo que hace calcular el periodo de una función (periódica) de la forma

$$f(x) = a^x \text{mod} N$$

donde a y N son enteros positivos mayores que 1, siendo además $a < N$ y no teniendo factores comunes. La operación (z mod$N$) a lo que se refiere es a quedarnos con el **resto** de dividir el número que z por N. El periodo r de esta función se calcula mediante el algoritmo de estimación de fase cuántica. Una vez se tiene el periodo r, si este es par (sino hay que probar con otro valor de a) se pueden calcular los factores de N ya que existe una alta probabilidad de que el máximo común divisor de N y $a^{r/2} - 1$ o $a^{r/2} + 1$ sea un factor propio de N

4.1. Hallar del periodo de una función (Period Finding)

4.1.1. La función

Como comentamos en la introducción, lo que queremos es hallar el periodo de la función

$$f(x) = a^x \text{mod } N$$

donde a y N son enteros positivos mayores que 1, siendo además $a < N$ y no teniendo factores comunes. La operación (z mod$N$) a lo que se refiere es a quedarnos con el **resto** de dividir el número que z por N. A este tipo de funciones se las denomina **exponenciales moduladas**.

Denominaremos r al valor del periodo de la función $f(x)$, es decir, r es el mínimo valor entero para que se cumple:

$$f(x + r) = f(x)$$

En la Fig. 4 vemos un ejemplo de este tipo de funciones con $a = 3$ y $N = 35$. Vemos que para este caso el periodo es $r = 12$.

4.1.2. Solución: Estimación de fase de un operador U

El algoritmo de Shor se basa en implementar el algoritmo de estimación de fases al operador unitario

$$U|y\rangle \equiv |ay \,\text{mod} N\rangle$$

Al aplicar sucesivas veces el operador U sobre el estado $|1\rangle$ vamos obteniendo los valores de $f(x)$ con $x \in \mathbb{N}$, esto es,

$$U^x|1\rangle = |f(x)\rangle$$

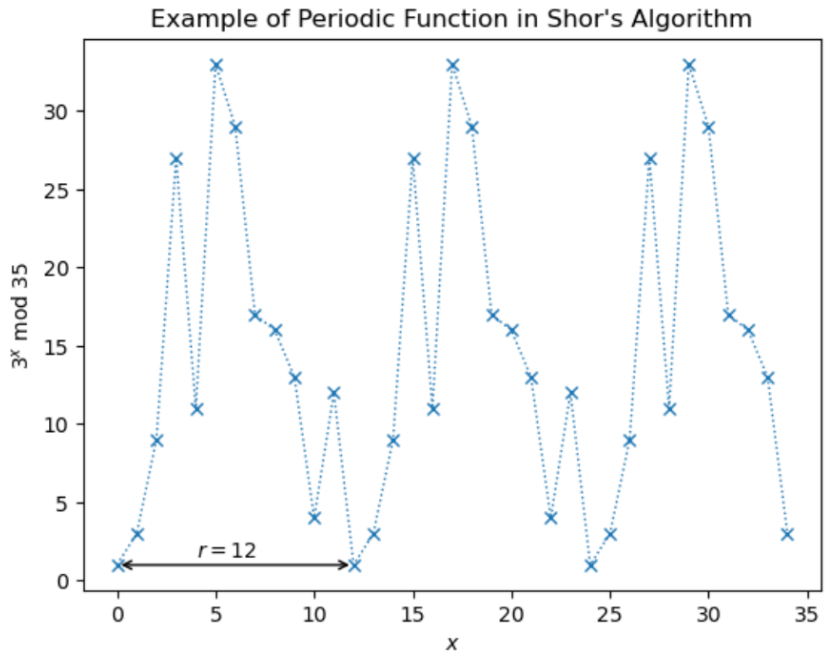


Figura 4: Gráfica de los primeros valores de la función periódica $f(x) = a^x \bmod N$ con $a = 3$ y $N = 15$. Véase que las lineas puntuadas que unen las cruces son solo por estética.

Por ejemplo, para el caso que vimos en la gráfica anterior ($a = 3$ y $N = 35$) tenemos

$$\begin{aligned} U^0|1\rangle &= |1\rangle \\ U|1\rangle &= |3\rangle \\ U^2|1\rangle &= |9\rangle \\ &\vdots \\ U^{r-1}|1\rangle &= |12\rangle \\ U^r|1\rangle &= |1\rangle \end{aligned}$$

(Recordemos que dado un estado de n qubits $|x\rangle$ tenemos que $|x\rangle = |x_1 x_2 \dots x_n\rangle$ donde x_1 es el bit más significativo.)

Como podemos ver, aplicar una vez más el operador U significa pasar de un número al siguiente de la lista periódica. Veámoslo explícitamente:

$$\begin{aligned} U(U^0|1\rangle) = U(|1\rangle) &= |3\rangle \\ U(U|1\rangle) = U(|3\rangle) &= |9\rangle \\ U(U^2|1\rangle) = U(|9\rangle) &= |27\rangle \\ &\vdots \\ U(U^{r-1}|1\rangle) = U(|12\rangle) &= |1\rangle \\ U(U^r|1\rangle) = U(|1\rangle) &= |3\rangle \end{aligned}$$

Con esto se entiende fácilmente que la superposición equiprobable de todos los estados es un autoestado

del operador U con autovalor 1:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \bmod N\rangle, \qquad \text{donde} \quad U|u_0\rangle = |u_0\rangle$$

Ejemplo

Ejemplo: caso con $a = 3$ y $N = 35$

$$\begin{aligned} U|u_0\rangle &= U\left[\frac{1}{\sqrt{12}}\left(|1\rangle + |3\rangle + |9\rangle + \ldots |4\rangle + |12\rangle\right)\right] = \\ &= \frac{1}{\sqrt{12}}\left(U|1\rangle + U|3\rangle + U|9\rangle + \ldots U|4\rangle + U|12\rangle\right) = \\ &= \frac{1}{\sqrt{12}}\left(|3\rangle + |9\rangle + |27\rangle + \ldots |12\rangle + |1\rangle\right) = \\ &= |u_0\rangle \end{aligned}$$

Un autoestado de autovalor 1 no nos es muy interesante a la hora de aplicar el algoritmo de estimación fase. Otro conjunto de autoestados mucho más interesantes son aquellos de la forma:

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\mathbf{2\pi i} k \frac{s}{r}} |a^k \bmod N\rangle, \qquad \text{donde} \quad U|u_s\rangle = e^{\mathbf{2\pi i} \frac{s}{r}} |u_s\rangle$$

donde $0 \leq s \leq r-1$. Si ahora aplicamos el algoritmo de estimación de fase cuántica a uno de estos autoestados $|u_s\rangle$, lo que obtendremos en el registro de conteo es $|2^n s/r\rangle$. De aquí podemos extraer el valor de r. Sin embargo, para preparar el estado $|u_s\rangle$ tenemos que conocer r, es decir, lo que queremos calcular.

Un solución elegante y fácil de implementar es darnos cuenta de que la suma de todos estos estados $|u_s\rangle$ nos da el estado $|1\rangle$, esto es,

$$\frac{1}{r} \sum_{s=0}^{r-1} |u_s\rangle = \frac{1}{r}\left(|u_0\rangle + |u_1\rangle + \cdots + |u_{r-1}\rangle\right) = |1\rangle$$

Si ahora aplicamos el algoritmo de estimación de fase cuántico (QPS) al estado —1¿ (un estado fácilmente implementable) obtenemos una superposición equiprobable de estados de la forma $|2^n s/r\rangle$, es decir:

$$|0\rangle|1\rangle \xrightarrow{QPS} \frac{1}{\sqrt{r}}\left(\left|2^t \frac{1}{r}\right\rangle + \left|2^t \frac{2}{r}\right\rangle + \cdots + \left|2^t \frac{r-1}{r}\right\rangle\right)|1\rangle$$

donde t es el número de qubits del registro de conteo. Usando el las fracciones continuas [4] podemos calcular r a partir de los cocientes s/r. En Fig. 5 podemos ver el circuito (en el orden de Qiskit para los qubits) que implementa la estimación de fase cuántica

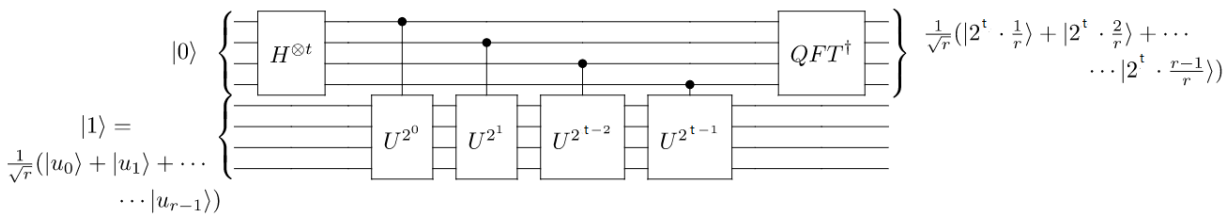


Figura 5: Implementación del algoritmo de estimación de fase cuántica (en el convenio de Qiskit).

Nota

Denominamos ***n* al número de qubits que necesitamos para codificar el número *N*** (que queremos factorizar) en un registro cuántico. Para aplicar el algoritmo de Shor se suelen usar ***t* = 2*n* qubits en el registro de conteo**.

4.2. Implementación (ad hoc) en Qiskit para $N = 15$

En esta sección vamos a ver una implementación un poco *ad hoc* del algoritmo de Shor para factorizar el número 15. Con lo de *ad hoc* a lo que nos referimos es a que el oráculo que representa la exponencial modulada del operador U está construido específicamente para el caso de factorizar 15 y no se va a explicar como funciona. El objetivo de esta sección es ver como tratamos los resultados del circuito, aplicando el método de las fracciones continuas [4] para obtener r y no tanto ver como se construye el oráculo.

En concreto, el caso que vamos a resolver es el de $a = 7$ y $N = 15$. Es fácil ver que para este caso tenemos un periodo de $r = 4$, donde los cuatro posibles estados son $|1\rangle$, $|7\rangle$, $|4\rangle$ y $|13\rangle$.

Nota

Un **oráculo** es, por decirlo así, la forma que se usa en computación cuántica para denominar a las funciones de las cuales queremos extraer información haciendo llamadas a las mismas. El oráculo es una función que desconoces cómo está hecha, sólo sabes lo que hace. Como a los oráculos griegos, sólo te está permitido interrogarles, pero no preguntarles cómo saben que esa es la respuesta. Al final los oráculos se representan como secciones del circuito que se dedica a aplicar un operador U. En nuestro caso, por ejemplo, llamamos oráculo a la función que construye el operador de la exponencial modulada $U^{2^j}|y\rangle = |a^{2^j} y \bmod N\rangle$.

4.2.1. Caso con muchos shots (ineficiente pero didáctico)

Como el número elegido para la factorización es pequeño (necesitamos pocos qubits) vamos a lanzar varios shots al circuito para ver todos los posibles resultado. Veremos que esto es en realidad una perdida de tiempo, pues no nos hace falta tener toda la distribución. Es simplemente un caso pedagógico para entender como lidiar con los resultados del circuito. Veamos pues el código (el código se he tomado del **textbook de Qiskit** [5]

Oráculo:

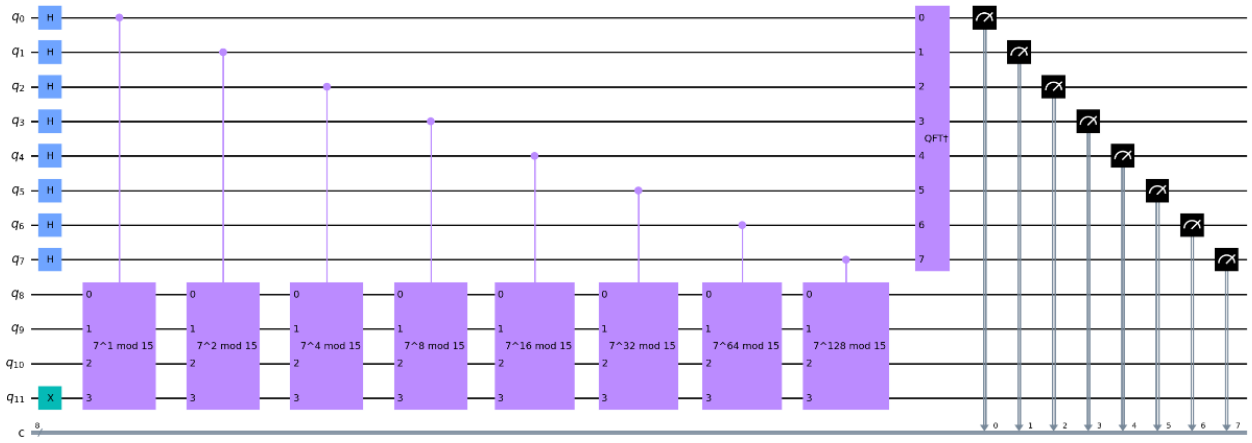
```
# Oracle: Modular Exponentiation U^{2^j} |y> = |a^{2^j}y mod N>
def c_amod15(a, power): # It use 4
    """Controlled multiplication by a mod 15"""
    if a not in [2,4,7,8,11,13]:
```

```
        raise ValueError("'a' must be 2,4,7,8,11 or 13")
    U = QuantumCircuit(4)
    for iteration in range(power):
        if a in [2,13]:
            U.swap(0,1)
            U.swap(1,2)
            U.swap(2,3)
        if a in [7,8]:
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
        if a in [4, 11]:
            U.swap(1,3)
            U.swap(0,2)
        if a in [7,11,13]:
            for q in range(4):
                U.x(q)
    U = U.to_gate()
    U.name = "%i^%i mod 15" % (a, power)
    c_U = U.control()
    return c_U

# Inverse Quantum Fourier Transformation of n qubits
def qft_dagger(n):
    """n-qubit QFTdagger the first n qubits in circ"""
    qc = QuantumCircuit(n)
    # Don't forget the Swaps!
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
    for j in range(n):
        for m in range(j):
            qc.cp(-np.pi/float(2**(j-m)), m, j)
        qc.h(j)
    qc.name = "inversQFT"
    return qc
```

Circuito:

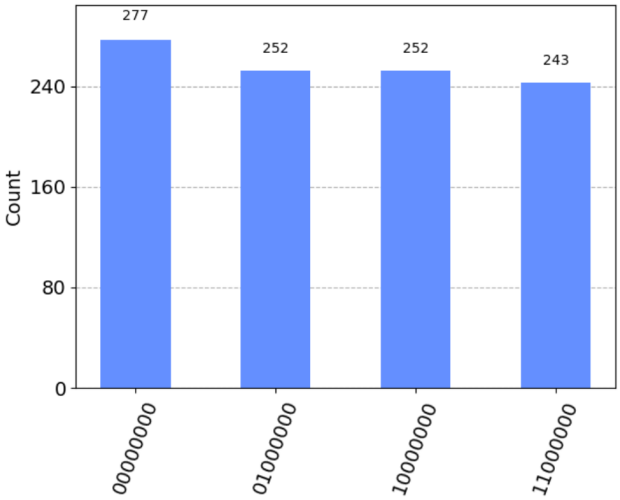
```
# Specify variables
n_count = 8 # number of counting qubits
a = 7

# Create QuantumCircuit with n_count counting qubits
# plus 4 qubits for U to act on
qc = QuantumCircuit(n_count + 4, n_count)

# Initialize counting qubits
# in state |+>
for q in range(n_count):
    qc.h(q)

# And auxiliary register in state |1>
qc.x(3+n_count)

# Do controlled-U operations
for q in range(n_count):
    qc.append(c_amod15(a, 2**q),
```

```
            [q] + [i+n_count for i in range(4)])

# Do inverse-QFT
qc.append(qft_dagger(n_count), range(n_count))

# Measure circuit
qc.measure(range(n_count), range(n_count))
qc.draw(output = 'mpl', fold=-1) # -1 means 'do not fold'
```



Simulación:

```
aer_sim = Aer.get_backend('aer_simulator')
t_qc = transpile(qc, aer_sim)
results = aer_sim.run(t_qc).result()
counts = results.get_counts()
plot_histogram(counts)
```



Como ya comentamos, el periodo para el caso elegido sabemos de antemano que es $r = 4$. Podemos ver que efectivamente, al lanzar varios shots al circuito (ejecutarlo varias veces) y ver todos los resultados tenemos 4 posibilidades, correspondientes a las 4 fases s/r con $0 \leq s \leq r$. Esto no es lo habitual al

aplicar el algoritmo de Shor, pues cuantos más grande es el número que queremos factorizar, más qubits necesitamos, más complejo se vuelve el circuito y más shots necesitamos para reconstruir la distribución de probabilidad.

Lo óptimo es lanzar un shot, ver que valor de r obtenemos con el resultado y intentar hallar los divisores de N con este r. Si no funciona, lanzamos otro shot y probamos a calcular otra vez r con el nuevo resultado. Repetimos el proceso hasta encontrar los factores. Esto será lo que haremos en la sección 4.2.2.

Nota

Se trata de una simulación ideal, sin ruido. En un ordenador cuántico de verdad tendríamos más resultados con probabilidades bajas y estos 4 sobresaliendo.

Para obtener los 4 valores de r correspondientes a los 4 resultados que acabamos de obtener, primero debemos recordar que estos resultados son los valores $2^n s/r$ para $0 \leq s \leq r$, con s siendo un número entero. Lo primero que debemos hacer es pasar nuestros resultados de binario a decimal, después dividir por 2^n.

Una vez hecho esto tendremos números reales entre 0 y 1. Debemos aproximar estos números por una fracción. Para ello podemos usar la función de python Fraction() de python. Como sabemos que el periodo de la función a^x mod N no puede mayor que N, podemos usar Fraction().limit_denominator(N) para limitar el valor del denominador. Tenemos pues:

```
rows, measured_phases = [], []
for output in counts:
    decimal = int(output, 2) # Convert (base 2) string to decimal
    phase = decimal/(2**n_count) # Find corresponding eigenvalue
    frac = Fraction(phase).limit_denominator(15)
    # Add these values to the rows in our table:
    rows.append([f"{output}(bin) = {decimal:>3}(dec)",
                 f"{phase:.2f}",
                 f"{frac.numerator}/{frac.denominator}",
                 frac.denominator])
# Print the rows in a table
headers=["Register Output", " Phase ", " Fraction ", "Guess for r"]
df = pd.DataFrame(rows, columns=headers)
print(df)
```

```
             Register Output   Phase     Fraction    Guess for r
 0  01000000(bin) =  64(dec)      0.25          1/4            4
 1  10000000(bin) = 128(dec)      0.50          1/2            2
 2  11000000(bin) = 192(dec)      0.75          3/4            4
 3  00000000(bin) =   0(dec)      0.00          0/1            1
```

Vemos varías cosas interesantes en este resultado. Primero, vemos que la opción predominante para el periodo es $r = 4$, que como sabemos, es correcta (puede verse que para $a = 7$ y $N = 15$ las cuatro opciones posibles son $|1\rangle$, $|7\rangle$, $|4\rangle$ y $|13\rangle$). Vemos también que el algoritmo de Shor puede fallar. En concreto, falla cuando tenemos la fase del estado con $s = 0$ y cuando s y r tienen un factor común (en este caso tenemos que para $s = 2 \rightarrow 2/4 \rightarrow 1/2$).

Tenemos pues, que a parte de obtener valores erróneos de r por el ruido (no presente en nuestra simulación ideal), el propio algoritmo de Shor nos puede dar valores de r erróneos.

Ahora solo faltaría aplicar lo comentado en la sección 1.4.2 para calcular los factores.

4.2.2. Caso shot a shot (óptimo)

Después de la explicación didáctica de la sección anterior, vamos ahora a ver como hacer una implementación más útil. Para ello, en vez de lanzar varios shot al circuito, coger todos los resultados (o los más probables si tenemos ruido) y calcular los valores de r, lo que vamos hacer es ir shot a shot, calculando r con cada resultado que nos de el circuito, hasta hallar los factores de N. Generalmente necesitaremos muchos menos shots, así que será más rápido.

```
# Number to be factored
N=15
```

Elegimos un número aleatorio, a, entre 1 y $N-1$

```
# Choose a random number, a, between 1 and N-1
np.random.seed(1) # This is to make sure we get reproduceable results
a = randint(2, 15)
print('a = ', a)
```

Verificamos que a no es un factor de no trivial de N

```
#Next we quickly check it isn't already a non-trivial factor of N
from math import gcd # greatest common divisor
gcd(a, N)
```

nota

Véase que el máximo común divisor se calcula usando la función gcd() del módulo [math](https://docs.python.org/3/library/math.html) de python. Esto es, se calcula clásicamente (sin ningún circuito cuántico), pues como ya comentamos, el algoritmo de Euclides se implementa de forma eficiente en un ordenado clásico.

Ahora definimos una función que aplique el algoritmo de estimación de fases para fase s/r donde

$$a^r \bmod N = 1$$

donde s es un entero aleatorio entre 0 y $N-1$.

```
def qpe_amod15(a):
    n_count = 8
    qc = QuantumCircuit(4+n_count, n_count)
    for q in range(n_count):
        qc.h(q)    # Initialize counting qubits in state |+>
    qc.x(3+n_count) # And auxiliary register in state |1>
    for q in range(n_count): # Do controlled-U operations
        qc.append(c_amod15(a, 2**q),
                  [q] + [i+n_count for i in range(4)])
    qc.append(qft_dagger(n_count), range(n_count)) # Do inverse-QFT
    qc.measure(range(n_count), range(n_count))
    # Simulate Results
    aer_sim = Aer.get_backend('aer_simulator')
    # Setting memory=True below allows us to see a list of each sequential reading
```

```
    t_qc = transpile(qc, aer_sim)
    result = aer_sim.run(t_qc, shots=1, memory=True).result() # One shot
    readings = result.get_memory()
    print("Register Reading: " + readings[0])
    phase = int(readings[0],2)/(2**n_count)
    print("Corresponding Phase: %f" % phase)
    return phase
```

Ejecutamos esta función hasta hallar los factores (por ejemplo, con un bucle while):

```
factor_found = False
attempt = 0
while not factor_found:
    attempt += 1
    print("\nAttempt %i:" % attempt)
    phase = qpe_amod15(a) # Phase = s/r
    frac = Fraction(phase).limit_denominator(N) # Denominator should (hopefully!) tell us r
    r = frac.denominator
    print("Result: r = %i" % r)
    if phase != 0:
        # Guesses for factors are gcd(x^{r/2} +-1 , 15)
        guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
        print("Guessed Factors: %i and %i" % (guesses[0], guesses[1]))
        for guess in guesses:
            if guess not in [1,N] and (N % guess) == 0: # Check to see if guess is a factor
                print("*** Non-trivial factor found: %i ***" % guess)
                factor_found = True
```

```
Attempt 1:
Register Reading: 01000000
Corresponding Phase: 0.250000
Result: r = 4
Guessed Factors: 3 and 5
*** Non-trivial factor found: 3 ***
*** Non-trivial factor found: 5 ***
```

5. Implementación con 2n+3 qubit

En esta sección vamos a ver una implementación eficiente del algoritmo de Shor, siguiendo el paper [6], donde lo que se intenta es minimizar el número de qubits lo máximo posible.

Denominamos n al número de qubits que necesitamos para codificar el número N (que queremos factorizar) en un registro cuántico. Para aplicar el algoritmo de Shor se suelen usar $t = 2n$ qubits en el registro de conteo.

5.1. La idea

Esta sección puede ser un poco árida si es la primera vez que se ve esta implementación, pues está enfocada para ser re-leída una vez se ha mirado más a fondo la implementación. Veamos paso a paso como es esta implementación:

1. **Algoritmo cuántico de suma (ver sección 5.2.1)**: Como vamos a ir viendo en las siguientes secciones, esta implementación se basa en **sumar**. En concreto, se parte de la implementación del algoritmo cuántico de suma de Draper [7], que podemos ver en la Fig. 6

   Este algoritmo suma dos registros cuánticos a y $\phi(b)$, donde $|\phi(x)\rangle$ hace referencia a la transformada de Fourier del registro x:

   $$|\phi(x)\rangle = QFT\,|x\rangle.$$

   Lo que vamos a ver es como, partiendo de esta implementación del algoritmo de suma podemos construir la exponencial modulada.

2. **Valor clásico + registro cuántico, puerta $\boldsymbol{\phi ADD(a)}$ (ver sección 5.2.2)**: Nosotros queremos llegar a calcular el periodo de la función $f(x) = a^x \bmod N$, donde a es un valor que fijamos al principio del algoritmo de Shor. Como precisamente este valor es fijo, la primera simplificación para reducir el número de qubits es prescindir de los qubits que codifican a en el algoritmo de suma y tomar a como un valor clásico. Definimos así la puerta $\boldsymbol{\phi ADD(a)}$ (ver Fig. 7), que **suma un valor clásico $\boldsymbol{a}$ a la un registro cuántico que codifica el valor $\boldsymbol{b}$** :

   $$\phi ADD(a)\,|\phi(b)\rangle = |\phi(a+b)\rangle,$$

   donde $|\phi(x)\rangle$ hace referencia a la transformada de Fourier del registro x:

   $$|\phi(x)\rangle = QFT\,|x\rangle.$$

   Podemos además, definir la inversa de esta puerta, es decir, una puerta de resta. La acción de esta última se recoge en la Fig. 8 (**véase que la puerta de suma tiene la barra negra a la derecha y la de resta a la izquierda**)

3. **Suma modulada, puerta $\boldsymbol{\phi ADD(a)MOD(N)}$ (ver sección 5.2.3)**: Un vez definida esta puerta, podemos usarla para construir una puerta doble controlada de **suma modulada $\boldsymbol{\phi ADD(a)MOD(N)}$** (ver Fig. 9) tal que

   $$\phi ADD(a)MOD(N)|\phi(b)\rangle = |\phi\left((a+b) \bmod N\right)\rangle = QFT\,|(a+b) \bmod N\rangle$$

4. **Multiplicación modulada, puerta $\boldsymbol{CMULT(a)MOD(N)}$ (ver sección 5.2.5)**: Ahora, podemos usar esta puerta doble controlada de suma modulada para construir una puerta controlada de **multiplicación modulada $\boldsymbol{CMULT(a)MOD(N)}$** (ver Fig. 10) tal que

   $$CMULT(a)MOD(N)\,|c\rangle|x\rangle|b\rangle = |c\rangle|x\rangle|(b+ax) \bmod N\rangle, \qquad \text{si } c = 1$$

5. **Puerta controlada C-U_a (ver sección 5.2.6)**: Juntando esta puerta con una puerta **SWAP** (pues la multiplicación modulada nos sale en el registro de b, no el de x) y tomando $b = 0$ (una ancilla), podemos finalmente construir una puerta controlada C-U_a (ver Fig. 11) tal que :

$$C\text{-}U_a\,|c\rangle|x\rangle = |c\rangle|(a \cdot x) \bmod N\rangle, \qquad \text{si } c = 1$$

6. **Exponencial modulada, puerta C-U_{a^s} (ver sección 5.2.7)**: Ahora, para implementar el algoritmo de Shor lo que se hace es tomar $|x\rangle = |1\rangle$ y implementar puertas C-U_{a^s}, donde $s = 2^0, 2^1, ..., 2^{2n-1}$, pues puede verse que

$$C\text{-}U_{a^s} = (C\text{-}U_a)^s$$

7. **Circuito final con $4n + 2$ qubits, sin la simplificación del registro de conteo (ver sección 5.2.8)**: El circuito para la implementación del algoritmo de Shor sería el de la Fig. 12.

8. **Circuito final con $2n + 3$. El truco de un qubit de control (ver sección 5.2.9)**: Nos faltaría implementar la simplificación del registro de conteo, donde se pasa de $2n$ qubits en el mismo a 1, llegando a tener el circuito de la Fig. 16.

5.2. Explicación desgranada

En las siguientes subsecciones vamos ir explicando poco a poco los pasos mencionado en la sección 5.1.

5.2.1. Algoritmo cuántico de suma

Como vamos a ir viendo en las siguientes secciones, esta implementación se basa en **sumar**. En concreto, se parte de la implementación del **algoritmo cuántico de suma de Draper** [7], que podemos ver en la Fig. 6

Nota importante

Véase que la puerta que la puerta "Conditional Phase Shif" de la Fig. 6 es la puerta $CROT_k$ de la sección 2.2. Véase también que estas puertas no son más que puertas $P(\phi)$ (o P_ϕ) controladas con $\phi_k = 2\pi i/2^k$.

Este algoritmo suma los valores a y b. Las entradas del circuito de suma son el $\boldsymbol{n}$ **qubits** representando el número a y $\boldsymbol{n}$ **qubits** que contienen la transformada de Fourier de otro número b, denotada como $\phi(b)$, es decir,

$$|\phi(b)\rangle = QFT\,|b\rangle.$$

El registro que codifica el número a no cambia, mientras que registro que codifica $\phi(b)$ pasa a albergar la suma de $a+b$ en el espacio de Fourier, $\phi(a+b)$. Haciendo la transformada inversa se puede recuperar el valor $a + b$:

$$QFT^{-1}|\phi(a + b)\rangle = |a + b\rangle$$

Lo que vamos a ver es como, partiendo de esta implementación del algoritmo de suma podemos construir la exponencial modulada.

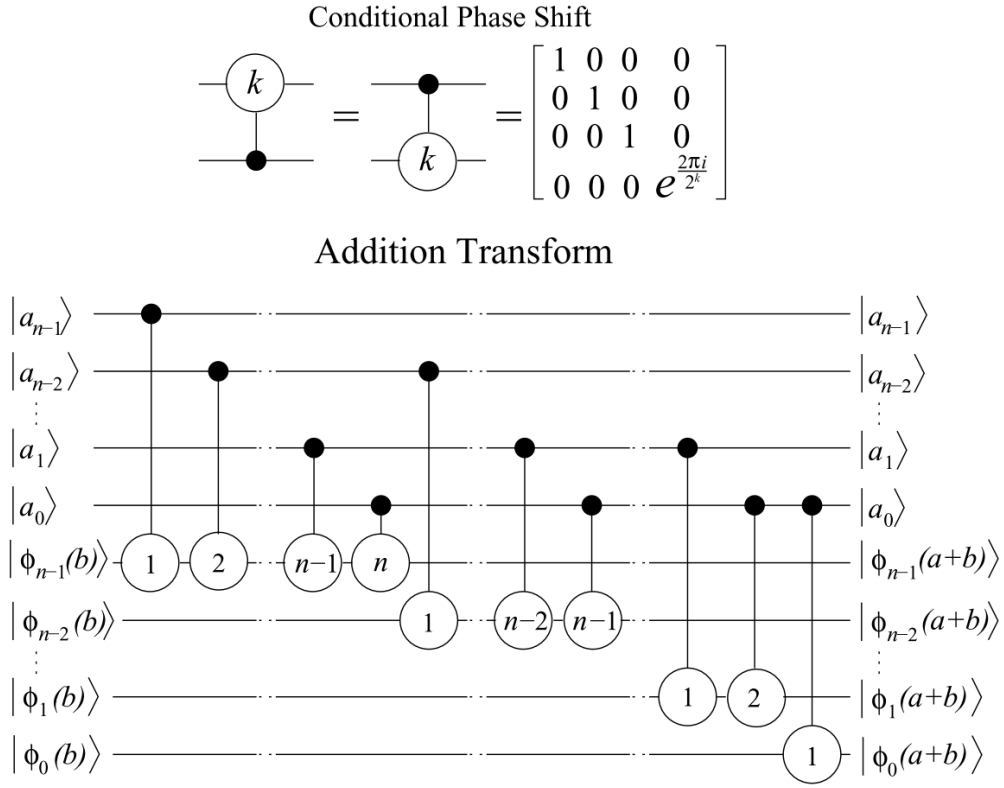


Figura 6: Algoritmo cuántico de suma de Draper.

Nota

No perdamos el objetivo de vista. Nosotros queremos calcular el periodo de la función $f(x) = a^x \text{mod} N$, donde $a < N$ (tomamos además $b < N$). Al tener un $\text{mod}N$ sabemos que el esta función puede devolver valores mayores que N. Tenemos pues que los $\boldsymbol{n}$ **qubits** que dijimos que usamos para codificar a y b son **el número de qubits que nos hace falta para codificar N**

5.2.2. Valor clásico + registro cuántico (puerta $\phi ADD(a)$)

Nosotros queremos llegar a calcular el periodo de la función $f(x) = a^x \text{ mod} N$, donde a es un valor fijo menor que N.

Como precisamente este valor es fijo, no nos vemos en la necesidad de codificarlo usando un registro cuántico.Podemos pues sustituir los qubits que codifican a por bits clásicos. Las puertas controladas pasan entonces a ser puertas controladas clásicamente. Además, como sabemos de antemano el valor de a, podemos precalcular el producto de las puertas sobre cada qubit, aplicando así solo una puerta por qubit (reducimos la profundidad del circuito).

Definimos así la puerta $\boldsymbol{\phi ADD(a)}$ (ver Fig. 7), que **suma un valor clásico $\boldsymbol{a}$ a la un registro cuántico que codifica el valor $\boldsymbol{b}$**. La entrada de esta puerta es la transformada de Fourier del registro b, es decir, $\phi(b)$, y la salida es la transformada de Fourier de la suma, $\phi(b)$:

$$\phi ADD(a)\,|\phi(b)\rangle = |\phi(a+b)\rangle,$$

Comentamos antes que el número de qubits n que usamos para codificar a y b es número de qubits que nos hacen falta para codificar N (ya que a, b ¡ N). Puede darse el caso de que al hacer la suma

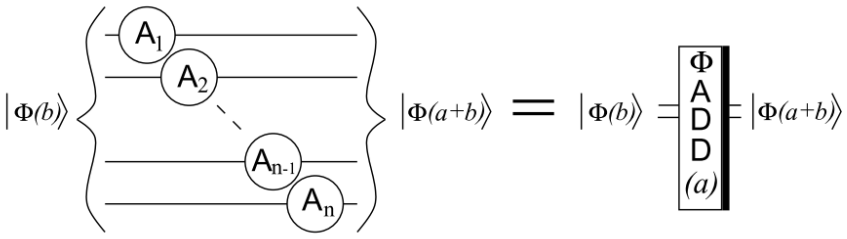


Figura 7: Puerta $\phi ADD(a)$.

tengamos un valor mayor que N, es decir, $a+b > N$. Podríamos tener entonces un número mayor que el número más grande que podemos codificar con los n qubits de los que partimos. Esto se denomina **overflow**. Para evitar esto, lo que podemos hacer es añadir un qubit extra al registro que contiene $\phi(b)$. Tenemos pues que **$\phi(b)$ es de forma efectiva la transformada de Fourier de un registro de $n+1$ qubits que contiene un número de n bits**. Tenemos pues que, antes de la suma, **el bit más significativos de la transformada de Fourier inversa del registro $\phi(b)$ es siempre $|0\rangle$**:

$$\text{el bit más significativo de } \; QFT^{-1}|\phi(b)\rangle = |b\rangle \; \text{ es siempre } \; |0\rangle$$

Podemos además, definir la inversa de esta puerta, es decir, una puerta de resta. La acción de esta última se recoge en la Fig. 8 (**véase que la puerta de suma tiene la barra negra a la derecha y la de resta a la izquierda**). En esta figura p **es un número n bit** y **g un número $n+1$ bit**.

Véase que tenemos resultados diferentes si $g \geq p$ o $g < p$. Precisamente, podemos usar esto para saber cual de los dos números es mayor:

- Si después de la resta el bit más significativo es $|0\rangle$, estamos en el caso de $g \geq p$ (**solo si $g-p$ es un número n bit**).
- Si después de la resta el bit más significativo es $|1\rangle$, estamos en el caso de $p > g$.

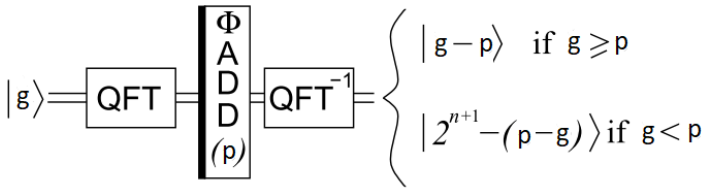


Figura 8: Puerta $\phi ADD^{-1}(a)$, es decir, la puerta inversa de $\phi ADD(a)$, una puerta de resta.

Nota

Aplicaremos este método con:

- $p = N$ y $g = b,\ a+b$ (parte 1 en la Fig. 9)
- $p = a$ y $g = b,\ a+b,\ a+b-N$ (parte 2 en la Fig. 9)

En ambos casos tenemos que p es un número n bit y si $g \geq p$ tenemos $g-p < N$, así que $g-p$ es n bit. Podemos pues aplicar el criterio del bit más significativo para saber que número es más grande.

Nota

Si estamos en el caso $g < p$ tenemos

$$\phi ADD^{-1}(p)\,|\phi(g)\rangle = |\phi(2^{n+1} - (p-g))\rangle$$

Si ahora sumamos otra vez p tenemos:

$$\phi ADD(p)\,|\phi(2^{n+1} - (p-g))\rangle = |\phi(2^{n+1} - (p-g)) + p\rangle = |\phi(2^{n+1} + g))\rangle$$

Como $2^{n+1} + g$ es mayor que el valor máximo que podemos almacenar con $n+1$ qubits, tenemos overflow:

$$\phi ADD(p)\,|\phi(2^{n+1} - (p-g))\rangle = |\phi(2^{n+1} + g))\rangle = |\phi(g)\rangle$$

Con lo cual, efectivamente, $\phi ADD(p)$ y $\phi ADD^{-1}(p)$ son una la inversa de la otra.

5.2.3. Suma modulada (puerta $\phi ADD(a)MOD(N)$)

Un vez definida la puerta de suma $\phi ADD(a)$, podemos usarla para construir una puerta de **suma modulada $\boldsymbol{\phi ADD(a)MOD(N)}$** (ver Fig. 10). Esta puerta suma $a + b$ y le resta N si $a + b \geq N$. Las entradas de la misma son $\phi(b)$ con $b < N$ y un valor clásico $a < N$.

$$\phi ADD(a)MOD(N)|\phi(b)\rangle = |\phi\left((a+b)\ \mathrm{mod} N\right)\rangle = QFT\,|(a+b)\ \mathrm{mod} N\rangle$$

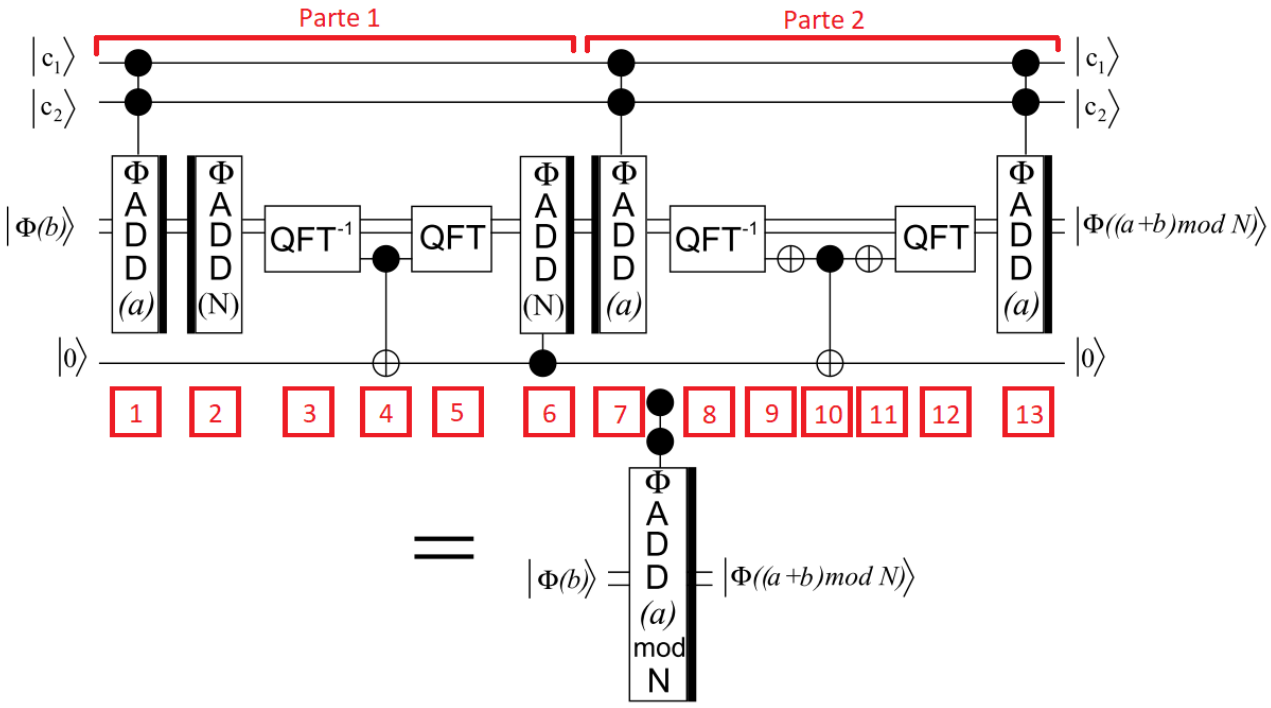


Figura 9: Puerta $\phi ADD(a)MOD(N)$.

En la Fig. 9 vemos que se han añadido dos qubits de control ($|c_1\rangle$, $|c_2\rangle$) para futuros usos. La puerta solo se activa si $c_1 = c_2 = 1$. Se han numerado las puertas (cuadros rojos) para facilitar la explicación.

Nota importante

El paper [6] del que se han sacado las figuras mezcla los dos convenios de ordenación de los qubits en los circuitos. En la Fig. 9 se usa **el mismo que usa Qiskit**, donde el bit más significativo es el de abajo. Por ese motivo las CNOT que se aplican en el qubit de abajo del todo son controladas por el último qubit el registro $|\phi(b)\rangle$.

Como podemos ver en la imagen, el circuito que implementa esta puerta tiene dos parte:

- Parte 1: Calcula $(a+b)$ mod$N$.
- Parte 2: Vuelve a poner en el estado $|0\rangle$ el qubit ancila (el qubit de abajo del todo en la imagen).

Nota

Un qubit **ancila** es un qubit auxiliar que se usa para hacer un calculo intermedio pero que no forma parte de la solución. Estos qubit hay que volver a ponerlos en el estado inicial.

Nota importante

El qubit ancila **no es qubit extra del registro** $|\phi(b)\rangle$. Tenemos pues los 2 qubits de control, los $n+1$ de la entrada $|\phi(b)\rangle$ y el qubit de la ancila, con lo cual usamos $n+4$ qubits.

Veamos todos los posibles casos que nos podemos encontrar el aplicar el circuito de la Fig 9.

5.2.3.1. Caso con $c_1 = c_2 = 1$

Partimos del estado $|\phi(b)\rangle$.

- **Puerta 1**: Después de aplicar la puerta 1 (puerta $\phi ADD(a)$) tenemos el estado $|\phi(a+b)\rangle$.
- **Puertas 2, 3, 4 y 5**: Lo que vamos a hacer con estas puestas es ver si estamos en el caso de $a+b \geq N$ o $a+b < N$. Para ello, aplicamos una puerta $\phi ADD^{-1}(N)$ (puerta 2), con lo cual tenemos:

  1. El estado $|\phi(a+b-N)\rangle$ si $a+b \geq N$.
  2. El estado $|\phi(2^{n+1}-(a+b-N))\rangle$ si $a+b < N$.

  Las puertas 3, 4 y 5 lo que hacen es poner el qubit ancila a 1 si el bit más significativo es 1, es decir, si estamos en el segundo caso.

- **Puerta 6**: Esta puerta solo se activa si la ancila es 1, es decir, si estamos en el segundo caso del paso anterior $(a+b<N)$. Esta puerta lo que hace es sumar N, deshaciendo la resta de la puerta 2 si estamos en el caso $a+b<N$, es decir, si no teníamos que haber restado N. Después de esta puerta ya tenemos el resultado de la suma modulada. Solo nos queda limpiar la ancila.
- **Puerta 7**: Partimos del valor $(a+b)$ mod$N$. Con esta puerta sumamos a, con lo cual tenemos dos casos.

  3. Si $(a+b) > N \quad \Rightarrow \quad (a+b)\ \text{mod}N = a+b-N < a \quad \Rightarrow$

     $$\Rightarrow \quad \phi ADD^{-1}(a)\,|\phi((a+b)\ \text{mod}N)\rangle = |\phi(2^{n+1}-(N-b))\rangle$$

  4. Si $(a+b) < N \quad \Rightarrow \quad (a+b)\ \text{mod}N = a+b > a \quad \Rightarrow$

     $$\Rightarrow \quad \phi ADD^{-1}(a)\,|\phi((a+b)\ \text{mod}N)\rangle = |\phi(b)\rangle$$

- **Puertas 8, 9, 10, 11, 12**: Estas puertas lo que hacen es cambiar la ancila si el bit más significativo es cero. Vemos que este es el caso 4 del paso anterior, es decir, cuando $(a+b) < N$. Si nos fijamos en cuando aplicamos las puertas 2, 3, 4 y 5, esto corresponde al caso 2, justo aquel en el que cambiamos la ancila. Es decir, si hubiéramos cambiado la ancila, ahora la habríamos vuelto a poner a cero.
- **Puerta 13**: Deshace el cambio producido por la puerta 7.

5.2.4. Caso con $c_1 = 0$ y/o $c_2 = 0$.

En estos casos la puerta $\phi ADD(a)MOD(N)$ deja invariante la entrada. Partimos del estado $|\phi(b)\rangle$.

- **Puerta 1**: No se aplica.
- **Puerta 2**: Restamos N, con lo cual pasamos a tener $\phi\left(2^{n+1} - (N-b)\right)$.
- **Puertas 2, 3, 4 y 5**: Como El bit más significativo en este caso es siempre 1, estas puertas ponen la ancila a 1.
- **Puerta 6**: Como la ancila es 1, esta puerta se activa y deshace los cambios de la puerta 2. Pasamos pues a tener el estado inicial $|\phi(b)\rangle$ pero con la ancila a 1.
- **Puerta 7**: No se aplica.
- **Puertas 8, 9, 10, 11, 12**: Como comentamos antes, estas puertas cambian la ancila si el estado que entra tiene el bit más significativo a cero. Como tenemos el estado $|\phi(b)\rangle$, este es nuestro caso así que se vuelve a cambiar la ancila. Pasamos pues a tener el estado inicial, sin ningún cambio.
- **Puerta 13**: No se aplica.

5.2.5. Multiplicación modulada (puerta $CMULT(a)MOD(N)$)

El siguiente paso es usar la puerta $\phi ADD(a)MOD(N)$ para construir una puerta controlada de multiplicación modulada que denominaremos como $CMULT(a)MOD(N)$. La entrada de esta puerta serán tres registros $|c\rangle|x\rangle|b\rangle$, donde $|c\rangle$ es un qubit controlador:

- $CMULT(a)MOD(N)\,|c\rangle|x\rangle|b\rangle = |c\rangle|x\rangle|(b+ax)\ \mathrm{mod}N\rangle, \qquad$ si $c = 1$
- $CMULT(a)MOD(N)\,|c\rangle|x\rangle|b\rangle = |c\rangle|x\rangle|b\rangle, \qquad$ si $c = 0$

Para implementar esta puerta recurrimos a las puertas $\phi ADD(a)MOD(N)$ de la sección anterior y a la identidad

$$\begin{aligned}(ax)\mathrm{mod}N &= (2^0 ax_0 + 2^1 ax_1 + \cdots + 2^{n-1} ax_{n-1})\mathrm{mod}N \\ &= \{\ldots[(2^0 ax_0)\mathrm{mod}N + 2^1 ax_1]\mathrm{mod}N + \cdots + 2^{n-1} ax_{n-1}\}\mathrm{mod}N\end{aligned}$$

Es fácil entender esta identidad: es lo mismo sumar todos los términos y finalmente tomar el módulo de la suma completa que tomar el módulo del primer termino, sumárselo al segundo término, volver a tomar el módulo, etc. Vemos que esto se puede implementar aplicando aplicando primero una puerta $\phi ADD(2^0 a)MOD(N)$ sobre $|b\rangle$ controlada por $|c\rangle$ y $|x_0\rangle$, después una puerta $\phi ADD(2^1 a)MOD(N)$ sobre el resultado de la anterior controlada por $|c\rangle$ y $|x_1\rangle$, etc. Con lo cual, **solo necesitamos aplicar n puertas doblemente controladas** $\phi ADD(2^i a)MOD(N)$ con $0 \leq i < N$. Podemos ver esta implementación en la Fig. 10.

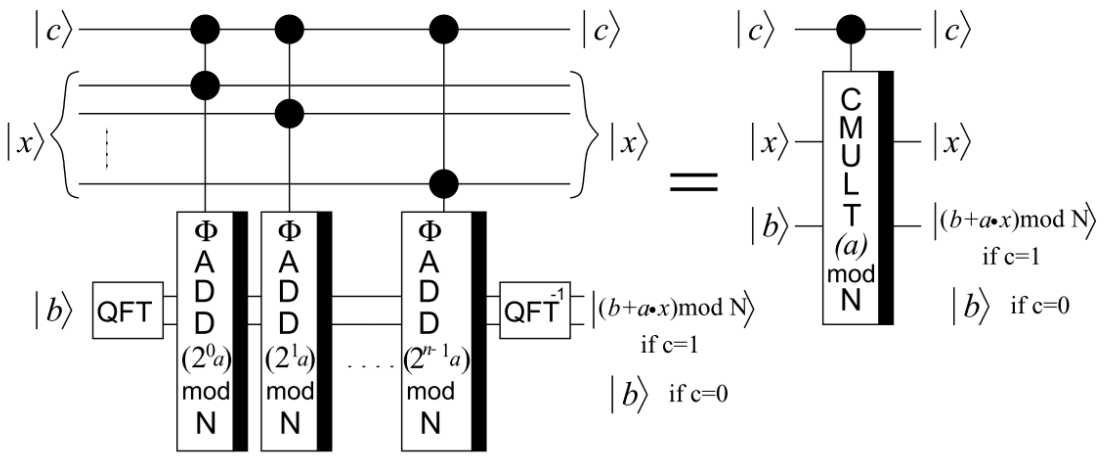


Figura 10: Puerta $CMULT(a)MOD(N)$.

Nota importante

El paper [6] del que se han sacado las figuras mezcla los dos convenios de ordenación de los qubits en los circuitos. En la Fig. 10 se usa **el mismo que usa Qiskit**, donde el bit más significativo es el de abajo. Por ese motivo las CNOT que se aplican en el qubit de abajo del todo son controladas por el último qubit el registro $|\phi(b)\rangle$.

5.2.6. Puerta controlada C-U_a

En la sección anterior vimos como construir una puerta controlada que aplica la operación

$$|x\rangle|b\rangle \quad \rightarrow \quad |x\rangle|(b+ax) \text{ mod}N\rangle$$

Pero esto no es lo que queremos, nosotros queremos una puerta controlada que nos lleve el estado $|x\rangle$ al estado $|(ax) \text{ mod}\rangle$. Lo que podemos hacer para solventar esto es lo siguiente:

- Aplicar primero una puerta $CMULT(a)MOD(N)$ sobre el estado $|c\rangle|x\rangle|0\rangle$, con lo cual obtenemos el estado $|c\rangle|x\rangle|(ax) \text{ mod}N\rangle$.
- A continuación, si $|c\rangle = |1\rangle$ aplicamos puertas SWAP controladas para cambiar los registros $|x\rangle$ y $|(b+ax) \text{ mod}N\rangle$, con lo cual pasamos a tener el estado $|c\rangle|(ax) \text{ mod}N\rangle|x\rangle$. Solo necesitamos aplicar puertas controladas SWAP a n qubits, no a $n+1$, ya que el qubit más significativo de $(ax) \text{ mod}N$ es siempre 0, ya que es el qubit extra que incluimos para evitar overflow en las puertas $\phi ADD(a)$.
- Finalmente, aplicamos la inversa de la puerta controlada $CMULT(a^{-1})MOD(N)$, donde a^{-1} es el inverso de a modulo N. Este valor se calcula clásicamente en tiempo polinómico usando el algoritmo de Euclides y tenemos asegurado que siempre existe ya que $gcd(a, N) = 1$. En resumen, si la entrada de esta puerta es el estado $|c\rangle|x\rangle|b\rangle$ con $|c\rangle = |1\rangle$ tenemos:

  $$[CMULT(a^{-1})MOD(N)]^{-1}|c\rangle|x\rangle|b\rangle = |c\rangle|x\rangle|(b-a^{-1}x\rangle$$

  En nuestro caso tenemos que el estado de entrada es $|c\rangle|(ax) \text{ mod}N\rangle|x\rangle$, con lo cual:

  $$[CMULT(a^{-1})MOD(N)]^{-1}|c\rangle|(ax) \text{ mod}N\rangle|x\rangle = |c\rangle|(ax) \text{ mod}N\rangle|(x-a^{-1}ax) \text{ mod}N\rangle = \\ = |c\rangle|(ax) \text{ mod}N\rangle|0\rangle$$

Denominaremos al conjunto de aplicar estas tres puertas controladas la **puerta controlada U_a**, es decir, $C\text{-}U_a$. En resumen, esta puerta lo que hace es tomar como entrada $|c\rangle|x\rangle|0\rangle$ y devolver $|c\rangle|(ax) \bmod N\rangle|0\rangle$ si $c = 1$:

$$C\text{-}U_a|c\rangle|x\rangle|0\rangle = |c\rangle|(ax) \bmod N\rangle|0\rangle \qquad \text{si } c = 1$$

Si $c = 0$, aplica la identidad.

Para resumir, si $|c\rangle = |1\rangle$ los pasos que hace la puerta $C\text{-}U_a$ son los siguientes (ver Fig. 11)

$$\begin{aligned} |x\rangle|0\rangle \quad &\rightarrow \quad |x\rangle|(ax) \bmod N\rangle \quad \rightarrow \quad |(ax) \bmod N\rangle|x\rangle \quad \rightarrow \\ \rightarrow \quad &|(ax) \bmod N\rangle|(x - a^{-1}ax) \bmod N\rangle = |(ax) \bmod N\rangle|0\rangle \end{aligned}$$

Si nos fijamos, el último registro, al estar a $|0\rangle$ al inicio y al final, podemos considerarlo parte de la puerta $C\text{-}U_a$ (una ancila):

$$C\text{-}U_a|x\rangle|0\rangle = |(ax) \bmod N\rangle|0\rangle \quad \Rightarrow \quad C\text{-}U_a|x\rangle = |(ax) \bmod N\rangle$$

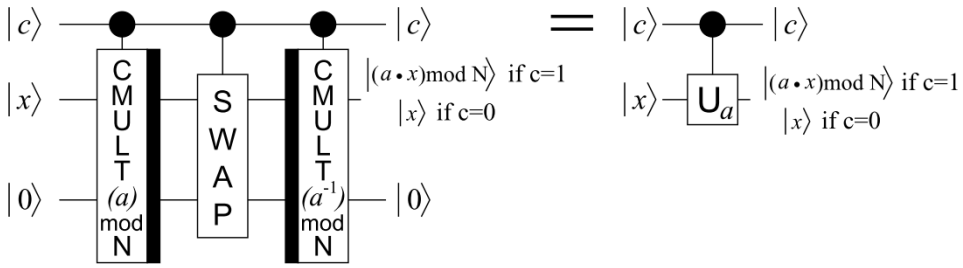


Figura 11: Puerta $C\text{-}U_a$

5.2.7. Exponencial modulada (puerta $C\text{-}U_{a^s}$)

Una vez construida la puerta $C\text{-}U_a$ uno podría pensar que para aplicar la exponencial modulada lo que hay que hacer es aplicar varias veces esta puerta, es decir:

$$(C\text{-}U_a)^s\,|x\rangle = |(a^s x) \bmod N\rangle$$

Aunque esta implementación es posible, tenemos la opción de hacer una mucho más optima. Para ello nos servimos de la propiedad

$$(a^s x) \bmod N = \underbrace{\{\dots[a(ax) \bmod N] \bmod N \dots\} \bmod N}_{s \text{ veces}} = [x(a^s) \bmod N] \bmod N$$

En vez de aplicar s veces la puerta $C\text{-}U_a$ podemos aplicar una sola vez la puerta $C\text{-}U_{a^s}$ donde el subíndice a^s hace referencia que le pasamos a la puerta el valor $a^s \bmod N$ (este se calcula clásicamente)

$$C\text{-}U_{a^s} = (C\text{-}U_a)^s$$

5.2.8. Circuito final con $4n+2$ qubits (sin la simplificación del registro de conteo)

Solo nos queda ver el circuito con la implementación completa del algoritmo de Shor, representado en la Fig. 12. (Recordemos que n es el número de qubits que necesitamos para codificar n y que en el registro de conteo necesitamos $2n$ qubits.)

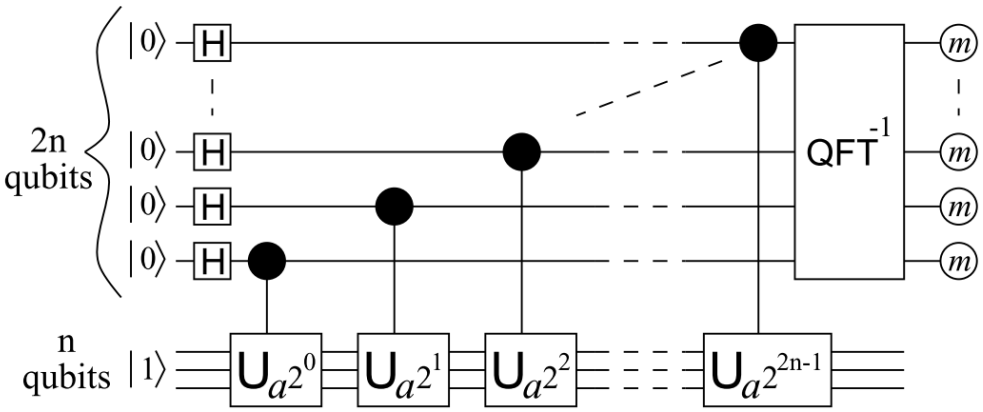


Figura 12: Circuito final con $4n+2$ qubits (sin la simplificación del registro de conteo)

Esta implementación usa $4n+2$ qubits:

- $2n$ qubits en el registro de conteo
- n qubits para el estado $|1\rangle$ (este es, el estado $|x\rangle$ de las secciones 5.2.5 a la 5.2.6.
- $n+2$ qubits para las ancilas:
  - $n+1$ qubits para el estado $|b\rangle = |0\rangle$ (ver Figs. 10 y 11).
  - 1 qubit para la ancila de la puerta $\phi ADD(a)MOD(N)$ (ver Fig. 9).

Nota importante

El paper [6] del que se han sacado las figuras mezcla los dos convenios de ordenación de los qubits en los circuitos. En la Fig. 12 se usa el **convenio estándar**, donde el bit más significativo es el de arriba. Por ese motivo, el qubit que controla la puerta U_{a^0} es el último, la que controla la puerta $U_{a^{2n-1}}$ es el primero, ...

5.2.9. Circuito final con $2n+3$. Algoritmo de estimación iterativa de fase (IPE)

Nos faltaría implementar la simplificación del registro de conteo, donde se pasa de $2n$ qubits en el mismo a 1 (ver Fig. 16). Esta simplificación consiste en usar una versión mejorada del Algoritmo de Estimación de Fases Cuántico (QPE) denominado **Algoritmo de Estimación Iterativa de Fase** o Iterative Phase Estimation (IPE) Algorithm. Vamos a explicar como pasar del QPE al IPE en nuestro circuito. Para ello, vamos a ejemplificar el desarrollo con el circuito de 4 qubits en el registro de conteo de la Fig. 13. Las puertas de colores del final no son más que la transformada de Fourier inversa. Vemos que al ser la inversa lo que hay que hacer es invertir el orden de las puertas del QFT normal (ver Fig. 17 y Fig. 1).

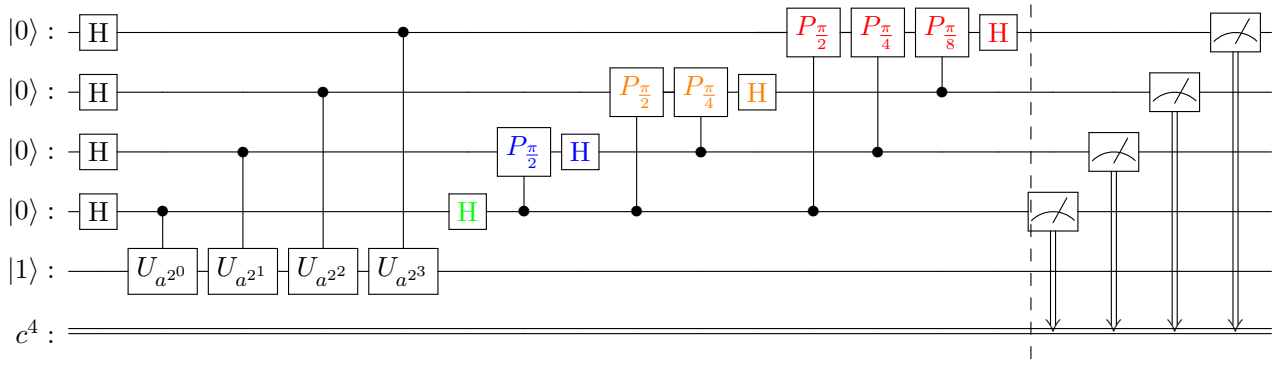


Figura 13: Ejemplo del algoritmo de Shor con 4 qubits. Las puertas de colores del final no son más que la transformada de Fourier inversa.

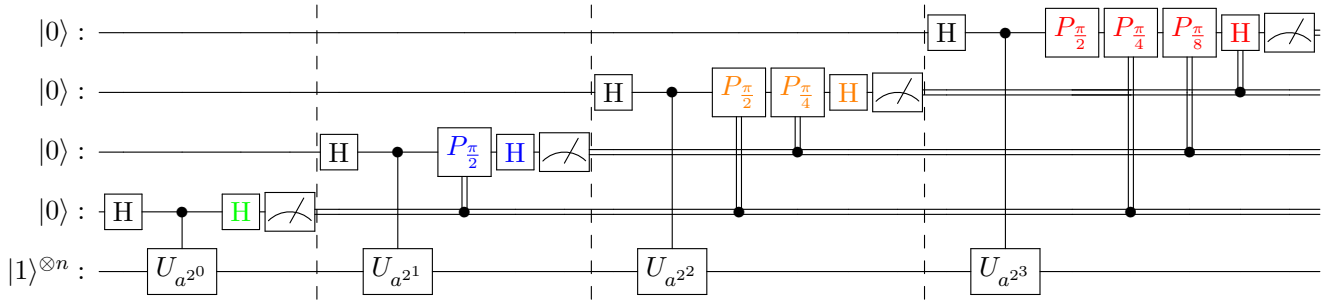


Figura 14: Ejemplo del algoritmo de Shor con 4 qubits usando IPE. Las puertas de colores no son más que la transformada de Fourier inversa.

Nota importante

Véase que la puerta que la puerta "Conditional Phase Shif" de la Fig. 6 es la puerta $CROT_k$ de la sección 2.2. Véase también que estas puertas no son más que puertas $P(\phi)$ (o P_ϕ) controladas con $\phi_k = 2\pi i/2^k$.

Si nos fijamos, vemos que una vez que un qubit controla su respectiva puerta $U_{a^{2^k}}$ sobre este ya no aplican (ni controla) más puertas hasta llegar a las de la QFT^{-1}. Podemos pues llevar a cabo sin ningún problema la reordenación de las puertas de color de la Fig. 14. Vemos sin embargo que hay otros dos cambios significativos:

- En vez de poner los 4 bit clásicos (en los que se almacenan las medidas) en una linea a parte, se han puesto a continuación del medidor. Esto es simplemente para no añadir 4 lineas más al circuito.
- El gran cambio que introducimos en este circuito es el hecho de controlar puertas con bits clásicos.

En el circuito de la Fig. 14 ya se han colocado las puertas para que se vea bien que estas se pueden aplicar de forma secuencial qubit por qubit. Es decir, primero se aplican las puertas sobre el qubit de abajo del todo (en el registro de conteo) y se mide. Después se va a por siguiente qubit y se mide, y así sucesivamente. La gracia es que, una vez que se mide un qubit, el valor de esta medida se va usar para controlar puertas que se aplican en los siguientes qubit. La gracia aquí es que, como ya comentamos,

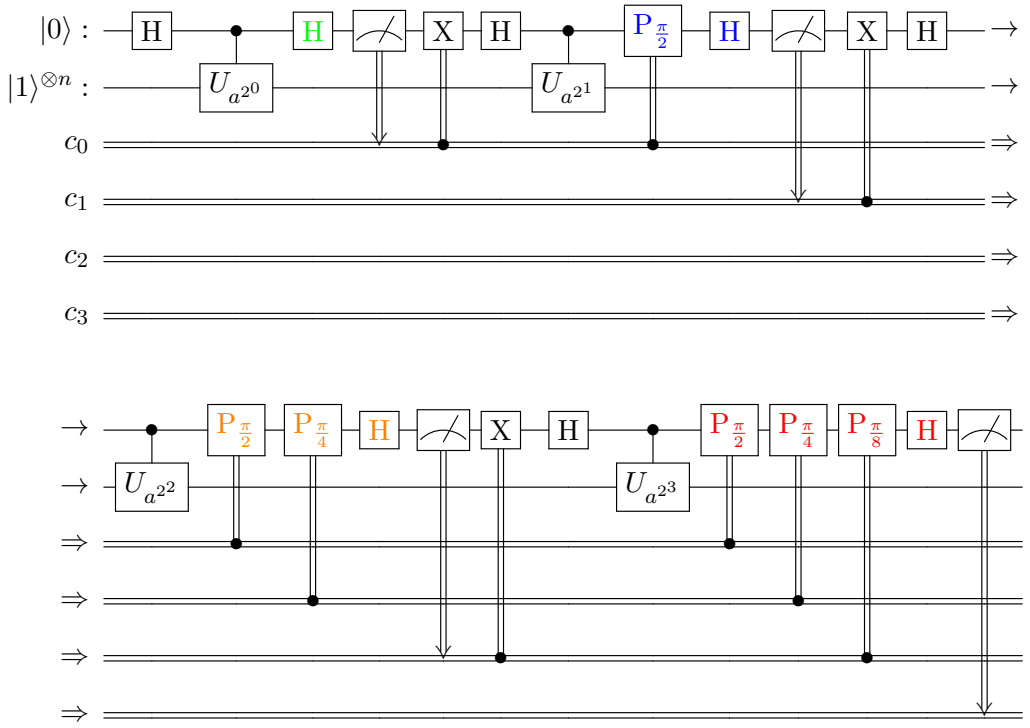


Figura 15: Circuito final con $2n + 3$ qubits (ejemplo con 4 qubits). Las m_k se refiere a medidas, las X^{m_k} se refiere a aplicar la puerta X controlada por las medida anterior y las R_k se refiere a aplicar las puertas P_ϕ de la QFT controladas por las medias anteriores.

estas medidas se almacenan en bits clásicos, con lo cual, una vez que se ha medido un qubit, este deja de ser necesario. De la misma forma, al ir aplicando las puertas de forma secuencial qubit por qubit, mientras se aplican las puertas a los qubits anteriores, lo qubits siguientes también son ïnutiles".

Siguiendo estos argumentos, podemos ver que en realidad, solo nos hace falta un qubit en el registro de conteo. Esto es lo que podemos ver en la Fig. 15. Como vemos, tenemos solo un qubit en el registro de conteo. Lo que se hace es primero aplicar sobre este qubit las puertas que aplicaríamos sobre el último qubit del registro de conteo y después medir. Almacenamos esta medida en un bit clásico. Una vez medido, podemos usar el valor del bit clásico para controlar una puerta X. De esta forma, lo que hacemos es devolver el qubit al estado inicial (el estado $|0\rangle$). Una ver que volvemos a tener el qubit en su estado de partida y la media del mismo a buen recaudo, podemos pasar a aplicar sobre este qubit las puertas que aplicariamos sobre el siguiente qubit del registro de conteo (una de ellas controlada por el bit clásico anterior) y medirlo, almacenando su valor en un segundo bit clásico. Nuevamente, usamos una puerta X controlada por este segundo bit clásico para devolver el qubit al estado inicial. Y así, sucesivamente. Para más detalles, pueden verse las referencias [8], [9] y [10]

Extrapolando al caso de $2n$ qubits en el registro de conteo (el caso del algoritmo de Shor), nuestro circuito final sería el de la Fig. 16, donde las m_k se refiere a medidas, las X^{m_k} se refiere a aplicar la puerta X controlada por las medida anterior y las R_k se refiere a aplicar las puertas P_ϕ de la QFT controladas por las medias anteriores.

5.3. Implementación aproximada de la QFT

Ya vimos en la sección 2.2 la implementación de la trasformada de Fourier cuántica. En la Fig. 17 presentamos otra vez la misma implementación pero con la notación de puertas que estuvimos usando

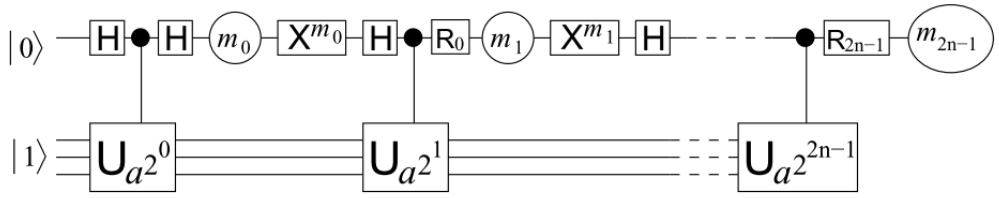


Figura 16: Circuito final con $2n + 3$ qubits.

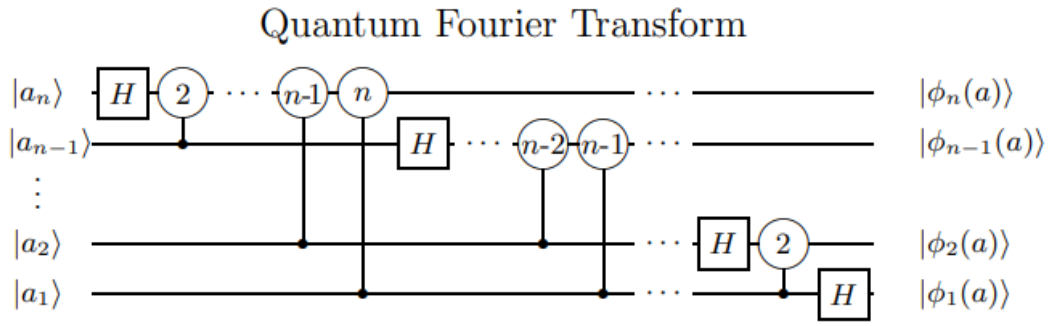


Figura 17: Implementación de la trasformada de Fourier (exacta).

en esta sección.

Nota importante

Véase que las diferencias entre la Fig. 17 y la Fig. 1 son simplemente el cambio de notación de las puertas $UROT_k$ y que faltan las puertas SWAP del final. Esto es simplemente porque en la Fig. 1 se usaba el convenio de Qiskit para la ordenación de los qubits, mientras que en la Fig. 17 se usa el convenio estándar.

Esta es la implementación exacta de la QFT que, como vemos la Fig. 17, cuando la aplicamos a n qubits necesitamos del orden de $\mathcal{O}(n^2)$ operaciones/puertas (en concreto, son $\frac{1}{2}n(n+1)$ operaciones).

Recordemos la expresión matricial de las puertas $CROT_k$

$$CROT_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \exp\left(\frac{2\pi i}{2^k}\right) \end{bmatrix}$$

Se puede apreciar que cuando k crece mucho, las puertas $CROT_k$ se aproximan a la Identidad. Esto hace que podamos prescindir de las puertas con un valor k mayor que un cierto umbral k_{max} y aplicar así una versión **aproximada** (con menos operaciones) de la trasformada de Fourier. Puede demostrarse (ver [11]) que el error introducido por ignorar todas las puertas con un valor $k > k_{max}$ es proporcional a $n2^{-k_{max}}$. Podemos pues tomar k_{max} del orden de $\mathcal{O}(\log_2 n)$, pasando de tener del orden de $\mathcal{O}(n^2)$ operaciones (puertas) a $\mathcal{O}(n \log_2(n))$ operaciones [en concreto $\frac{1}{2}(2n - \log_2 n)(\log_2 n - 1)$ operaciones].

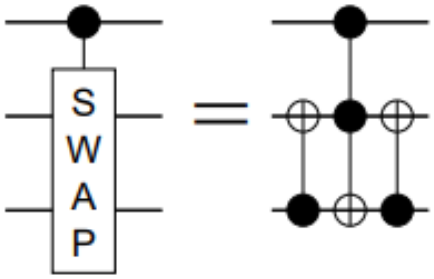


Figura 18: Puerta SWAP

5.4. Implementación de las SWAP controladas

En la Fig. 18 podemos ver la puerta $SWAP$ controlada. Esta es una puerta que admite tres qubits (uno de control y dos de operación). Tenemos dos puertas $CNOT$ que rodean a una puerta *Toffoli*. La función de esta puerta es intercambiar el valor de los dos qubits de operación si el qubit de control está activado. Como vemos, para intercambiar dos qubits necesitamos 3 puertas, con lo cual las puertas necesarias para aplicar una puerta $SWAP$ controlada a n qubit necesitamos $\mathcal{O}(n)$ puertas.

Vemos que la puertas $SWAP$ controlada no es las que la puerta $SWAP$ normal (tres puertas $CNOT$) donde la puerta $CNOT$ central se controla, convirtiéndola en una puerta *Toffoli*.

Referencias

[1] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[2] "Wikipedia: Euclidean algorithm." https://en.wikipedia.org/wiki/Euclidean_algorithm.

[3] "Textbook ibm: Quantum fourier transform." https://learn.qiskit.org/course/ch-algorithms/quantum-fourier-transform.

[4] "Wikipedia: Continued fraction." https://en.wikipedia.org/wiki/Continued_fraction.

[5] "Textbook ibm: Shor's algorithm." https://learn.qiskit.org/course/ch-algorithms/shors-algorithm.

[6] S. Beauregard, "Circuit for shor's algorithm using 2n+3 qubits," *arXiv preprint quant-ph/0205095*, 2002.

[7] T. G. Draper, "Addition on a quantum computer," 2000.

[8] M. Mosca and A. Ekert, "The hidden subgroup problem and eigenvalue estimation on a quantum computer," 1999.

[9] C. Zalka, "Fast versions of shor's quantum factoring algorithm," 1998.

[10] S. Parker and M. B. Plenio, "Efficient factorization with a single pure qubit and $\log N$ mixed qubits," *Phys. Rev. Lett.*, vol. 85, pp. 3049–3052, Oct 2000.

[11] D. Coppersmith, "An approximate fourier transform useful in quantum factoring," 2002.

[12] "Textbook ibm: Quantum phase estimation." https://learn.qiskit.org/course/ch-algorithms/quantum-phase-estimation.