UNIVERSITAT DE BARCELONA

Trabajo de final de grado

GRADO DE MATEMÁTICAS

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

Algunas variantes del algoritmo cuántico de Shor

Autor: Juan Cano Pradas

Director: Dr. Luis Victor Dieulefait
Realizado en: Departamento de Matemáticas e Informática

Barcelona, 13 de junio de 2022

Abstract

The aim of this project is to study the Shor's factorization algorithm, as well as some of its variants, both from a theoretical and practical point of view. First, the mathematical foundations on which it is based are presented, as well as the formalization of the notation used in quantum computing. Next, Shor's algorithm and some variants are detailed in order to make it more efficient. Finally, a practical Python implementation of Shor's quantum algorithm is carried out using IBM's Qiskit library and also another implementation of Ekerå's algorithm in SageMath.

Resumen

El objetivo de este trabajo es estudiar el algoritmo de factorización de Shor, así como algunas de sus variantes, tanto desde una vertiente teórica como práctica. En primer lugar, se presentan los fundamentos matemáticos en los cuales se basa, así como la formalización de la notación empleada en la computación cuántica. A continuación, se detalla el algoritmo de Shor y algunas variantes con el objetivo de hacerlo más eficiente. Por último, se lleva a cabo una implementación práctica en Python del algoritmo cuántico de Shor utilizado la librería Qiskit de IBM y también otra implementación en SageMath del algoritmo de Ekerå.

2020 Mathematics Subject Classification. 11A51, 68Q12, 81P68

Agradecimientos

Quiero agradecer al Dr. Luis Victor Dieulefait su implicación y guía a lo largo de todo el trabajo, desde la propuesta del tema hasta los últimos consejos, así como también al hecho de haberme introducido en el apasionante mundo de la computacion cuántica y su relación con la teoría de números. Ha sido un placer poder desarrollar este trabajo bajo su tutorización.

También doy las gracias a todas aquellas personas que, de alguna u otra forma, me han ayudado durante mi etapa universitaria: familiares, compañeros y profesores.

Por último, me gustaría dar las gracias especialmente a mis padres, a quienes agradezco todo el cariño y apoyo que siempre me han mostrado, y a mi hermana Nuria, que me ha ayudado en los momentos más complicados y me ha enseñado la importancia de la constancia y el esfuerzo. Sin vosotros no hubiera sido posible. Muchas gracias de todo corazón.

Índice general

Capítulo 1

Introducción

1.1. ¿Qué es la computación cuántica?

La computación cuántica representa un nuevo paradigma en el mundo de la computación ya que utiliza los principios fundamentales de la mecánica cuántica para realizar los cálculos y operaciones computacionales. Actualmente, el principal motivo de interés y desarrollo de la computación cuántica radica en la posibilidad de realizar de manera eficiente un cierto número de tareas como la factorización en primos, la simulación cuántica o la optimización. Estas operaciones serian inabordables, en cuanto a tamaño y tiempo de ejecución, incluso para las computadoras clásicas más avanzadas.

El poder de la computación cuántica se basa en dos piedras angulares de la mecánica cuántica, a saber, la superposición y el entrelazamiento que resultan del carácter dual de las partículas, con aspectos ondulatorios y, a su vez, corpuscular. Por lo tanto, los ordenadores cuáticos realizan cálculos que utilizan estos principios de la mecánica cuántica a través del lenguaje de los circuitos cuánticos.

Integrados por puertas cuánticas, instrucciones y lógica de control clásica, los circuitos cuánticos permiten expresar aplicaciones y algoritmos matemáticos complejos de manera abstracta que se pueden ejecutar en un ordenador cuántico. Normalmente, se utilizan en conjunto con los recursos informáticos clásicos para poder resolver desde problemas de optimización, química cuántica o física hasta en programas de aprendizaje automático y de redes neuronales para aplicarlo en ámbitos tan diversos como finanzas, climatología o criptografía.

1.2. ¿Qué es la criptografía?

Esta última disciplina, que se ocupa de las técnicas de cifrado o codificado destinadas a alterar las representaciones simbólicas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados, es decir, mantener la confidencialidad y la autenticidad, se podria ver altamente amenazada. Sistemas de encriptación como Diffie-Hellman o RSA,

que se basan en la imposibilidad temporal de poder factorizar ciertos semiprimos gigantescos corren un cierto riesgo de ser explotados debido a los avances en la computación cuántica y los diferentes algoritmos matemáticos que se han descubierto. Esto hace, por ejemplo, que nuestras transacciones en línea con tarjetas de crédito o nuestros mensajes privados enviados a través aplicaciones móviles, queden al descubierto.

1.3. Conociendo el algoritmo cuántico de Shor

En este sentido, algoritmos como el de Shor publicado en 1994[Shob], llamado así por el matemático Peter Shor, juegan un papel fundamental ya que reduce a tiempo polinomial la factorización de enteros. Podría decirse que es el ejemplo más dramático de cómo el paradigma de la computación cuántica ha cambiado nuestra percepción sobre qué problemas deberían considerarse computacionalmente abordables. Este hecho ha motivado el estudio de nuevos algoritmos cuánticos y se han incrementado los esfuerzos para diseñar y construir ordenadores cuánticos. Del mismo modo, también ha acelerado la investigación de nuevos criptosistemas que no se basan en la factorización de enteros, la conocida como criptografía postcuántica.

El algoritmo de Shor ha sido llevado a la práctica experimentalmente por varios equipos para enteros compuestos específicos. El número 15 se factorizó en 3×5 por primera vez en 2001 usando siete qubits de RMN [VSB$^+$], y desde entonces dos equipos lo han implementado usando cuatro qubits de fotones en 2007 [LWL$^+$, LBYP], tres qubits de estado sólido en 2012 [LBC$^+$] y cinco qubits de iones atrapados en 2016 [MNM$^+$]. El número 21 también se ha factorizado en 3×7 en 2012 [MLLL$^+$] usando un fotón qubit y un qutrit (un sistema de tres niveles). No obstante, cabe decir que estas demostraciones experimentales se basan en ciertas optimizaciones del algoritmo de Shor basadas en el conocimiento a priori de los resultados esperados. En general, se necesitan $2 + \frac{3}{2} log_2 N$ qubits [ME] para factorizar cualquier número entero N.

El algoritmo de Shor se compone de tres partes. La primera parte convierte el problema de factorización en un problema de búsqueda de períodos utilizando la teoría de números, que se puede calcular en una computadora clásica. La segunda parte encuentra el período utilizando la transformada cuántica de Fourier y es responsable de la aceleración cuántica del algoritmo. La tercera parte utiliza el período encontrado para calcular los factores.

A lo largo de este trabajo, podremos ir viendo desde los fundamentos de la computación cuántica, descritos en diversas publicaciones y libros [Mer, OCT, NC], hasta aplicaciones reales del algoritmo Shor para factorizar el entero 15. Así como la descripción detallada de cada una de sus partes y cómo podemos hacer para optimizarlo ayudándonos de la teoría de números para encontrar algunas variantes que sean más rápidas computacionalmente. En particular, este trabajo se compone de una parte teórica y de otra parte práctica, que nos muestran lo íntimamente relacionadas que están la informática y las matemáticas.

Capítulo 2

Fundamentos

2.1. Definiciones y propiedades básicas

En el ámbito de la computación clásica, la unidad básica de información es el bit. Un bit solo puede estar en uno de sus dos estados posibles y, por lo tanto, puede implementarse físicamente con un dispositivo de dos estados. Este par de valores se representa comúnmente con 0 y 1. Por otro lado, tenemos un concepto análogo en la computación cuántica: el qubit, abreviatura de bit cuántico, que es una representación matemática de un sistema mecánico-cuántico de dos estados.

Sabemos que los qubits existen en la naturaleza gracias al experimento Stern-Gerlacht, realizado por primera vez por los físicos alemanes Otto Stern (1888-1969) y Walther Gerlach (1889-1979) en 1922 [GS].

Supondremos que tenemos un espacio de Hilbert $\mathbb{C}^2$, con el producto escalar usual. En terminología cuántica, los vectores se escriben de la siguiente forma:

Definición 1. *Los vectores*

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \; y \; |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

se llaman los estados base de un bit cuántico.

Sin embargo, existe una diferencia entre bits y qubits: un qubit también puede estar en un estado distinto de $|0\rangle$ y $|1\rangle$. De hecho, su estado genérico es una combinación lineal sobre los números complejos de ambos estados base.

Definición 2. *Un estado puro de un qubit $|\psi\rangle$ es un vector unitario que es una combinación lineal de los estados de la base,*

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

donde los coeficientes $\alpha, \beta \in \mathbb{C}$ se denominan amplitudes del estado.

Como $|\psi\rangle$ es un vector unitario significa que se cumple la condición $|\alpha|^2 + |\beta|^2 = 1$.

2.1.1. Notación de Dirac y base de Hadamard

El símbolo $|\psi\rangle$, denominado *ket*, describe un estado cuántico. A esta notación se la conoce como bra-ket, o notación de Dirac [Dir]. Alternativamente, también usaremos $\langle\psi|$, llamado *bra*, para describir el conjugado Hermitiano de $|\psi\rangle$.

Por lo tanto, $|0\rangle$ y $|1\rangle$ forman una $\mathbb{C}$-base ortonormal de $\mathbb{C}^2$. A partir de ahora, la base formada por $|0\rangle$ y $|1\rangle$ la denominaremos como base computacional de un qubit. Sin embargo, existen otras bases comúnmente utilizadas para los estados de un bit cuántico. Un ejemplo que nos será útil más adelante es la denominada base de Hadamard, que se define como

$$|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

y

$$|-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Es fácil ver que $|+\rangle$ y $|-\rangle$ también forman una $\mathbb{C}$-base ortonormal de $\mathbb{C}^2$, y que nuestro qubit genérico $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ puede verse como

$$|\psi\rangle = \frac{\alpha+\beta}{\sqrt{2}}|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}|-\rangle.$$

2.1.2. Medición de un qubit

Una de las principales características que hace que una computadora cuántica difiera dramáticamente de su contraparte clásica es el proceso de medir el estado de un bit cuántico. Una medida, también llamada observación, de un estado genérico de un solo qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ es un procedimiento físico que produce un resultado a partir de la base ortonormal, dependiendo de los valores de α y β. Esta dependencia se modela como una distribución de Bernoulli: la probabilidad de que la medida dé como resultado $|0\rangle$ es $|\alpha|^2$ y, obviamente, la probabilidad de que la medida dé como resultado $|1\rangle$ es $|\beta|^2$.

Sin embargo, a diferencia del caso clásico, el proceso de medición inevitablemente perturba el qubit $|\psi\rangle$, obligándolo a colapsar a $|0\rangle$ o $|1\rangle$ y, por lo tanto, generalmente haciendo imposible la tarea de averiguar los valores reales de α y β. Este colapso en $|0\rangle$ o $|1\rangle$ no es determinista ni reversible, y es una característica fundamental de la computación cuántica. Se mostrará más adelante cómo cambiar estas probabilidades sin violar la restricción unitaria.

Resumiendo, uno podría pensar en un qubit como un bit no determinista. Este qubit puede tomar dos valores posibles, pero cuál de ellos realmente se obtiene depende de una distribución de probabilidad. Y, lo más importante, los datos proporcionados por estas probabilidades desaparecen una vez que se observa el qubit.

Cabe destacar que se puede medir un qubit respecto a diferentes bases y que si tenemos múltiples qubits entrelazados el hecho de medir uno hace que los otros también colapsen, perdiendo la información del estado de superposición.

2.1.3. Esfera de Bloch

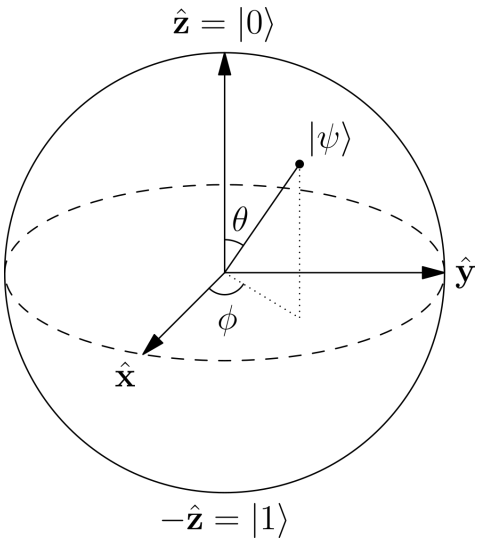


Figura 2.1: Esfera de Bloch. Es la representación geométrica del espacio de estados puros de un sistema cuántico de dos niveles. Su nombre se debe al físico suizo Felix Bloch [Blo][2]

Una posible representación geométrica de los estados de un sistema de un solo qubit es la llamada esfera de Bloch (ver Figura 2.1). Las amplitudes α y β no son interesantes por sí mismas: son sus módulos, que caracterizan la distribución de probabilidad, los que realmente importan. Por lo tanto, dos qubits que presentan la misma distribución son computacionalmente indistinguibles.

Por ello, podemos elegir un representante para todos esos qubits: por ejemplo, podemos forzar que α sea un número real, y se deduce directamente que, dado un qubit $|\psi\rangle$, solo hay un único qubit

$$|\psi_0\rangle = \cos\frac{\theta}{2} + e^{i\varphi}\sin\frac{\theta}{2}$$

donde $\theta \in [0, \pi]$ y $\varphi \in [0, 2\pi)$.

Eligiendo tal representación, vemos que un qubit genérico se puede representar únicamente como un punto (θ, φ) de la 2-esfera unidad, con los polos norte y sur típicamente elegidos para corresponder a los vectores de la base estándar $|0\rangle$ y $|1\rangle$ como se indica en la figura.

[2]Fuente: Glosser.ca (CC BY-SA 3.0)

La esfera de Bloch es útil como una forma de representar las transformaciones de un solo qubit. Como veremos más adelante, las transformaciones más importantes se pueden dividir, esencialmente, en rotaciones en la esfera de Bloch.

2.1.4. Sistemas de múltiples qubits y producto tensorial

Pasar de un solo qubit a un sistema de múltiples qubits obviamente debe implicar considerar un espacio vectorial de Hilbert derivado de la combinación de múltiples copias de $\mathbb{C}^2$. La forma elegida para hacerlo es utilizando el producto tensorial.

Definición 3. *Sean V y W espacios vectoriales de dimensiones n y m respectivamente. El producto tensorial de V y W, denotado por $V \otimes W$, es un espacio vectorial nm-dimensional cuyos elementos son combinaciones lineales de los símbolos $v \otimes w$ que satisfacen las siguientes propiedades:*

- $\alpha(v \otimes w) = (\alpha v) \otimes w = v \otimes (\alpha w)$
- $(v_1 + v_2) \otimes w = (v_1 \otimes w) + (v_2 \otimes w)$
- $v \otimes (w_1 + w_2) = (v \otimes w_1) + (v \otimes w_2)$

donde $\alpha \in \mathbb{C}$, $v, v_1, v_2 \in V$ y $w, w_1, w_2 \in W$.

Una definición relacionada que se usará en la sección 2.2 es el concepto de producto tensorial entre operadores lineales.

Definición 4. *Sean A y B operadores lineales definidos en V y W respectivamente, entonces el operador lineal $A \otimes B$ que opera en $V \otimes W$ se define como*

$$(A \otimes B)(v \otimes w) = Av \otimes Bw$$

donde $v \in V$ y $w \in W$.

Si A y B son matrices $n \times n$ y $m \times m$ respectivamente que corresponden a las representaciones matriciales de los operadores lineales A y B con respecto a la base canónica, el operador lineal AxB (llamado producto tensorial o producto de Kronecker de A y B) tiene la siguiente representación matricial con respecto a la base canónica:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}B & a_{n2}B & \cdots & a_{nn}B \end{bmatrix}$$

Como era de esperar, la representación matricial de $A \otimes B$ tiene dimensión $nm \times nm$. Como ocurre con el producto habitual esta operación no es conmutativa. Una notación común para el producto de Kronecker de l copias de una matriz A es $A^{\otimes l}$.

Ejemplo: Para mostrar cómo se calculan las representaciones matriciales de los productos de Kronecker, sean

$$A = \begin{bmatrix} 1 & -1 \\ -2 & 0 \end{bmatrix} \text{ y } B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

dos operaciones lineales definidas en $\mathbb{R}^2$ y $\mathbb{R}^3$ respectivamente. Entonces, su producto tensorial se calcula de la siguiente manera:

$$A \otimes B = \begin{bmatrix} 1 & -1 \\ -2 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 2 & 0 & 0 & -2 & 0 \\ 0 & 0 & 3 & 0 & 0 & -3 \\ -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 & 0 \\ 0 & 0 & -6 & 0 & 0 & 0 \end{bmatrix}$$

donde $A \otimes B$ es un operador lineal definido en $\mathbb{R}^6$.

Por supuesto, las definiciones anteriores se extienden de forma directa a productos tensoriales finitos de espacios y operadores. En particular, el producto tensorial de vectores unitarios es nuevamente un vector unitario.

Además, el producto tensorial que así hemos definido también puede extenderse a vectores y matrices no cuadradas, y será útil a la hora de calcular los estados base de un sistema cuántico con más de un qubit y representarlo como un vector en $\mathbb{C}^l$, para algun $l \in \mathbb{N}$.

Ejemplo: Si $|0\rangle$ y $|1\rangle$ son los estados básicos de un bit cuántico, el producto tensorial $|1\rangle \otimes |0\rangle$ estará dado por:

$$|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Observación: Como es bien sabido, dadas dos bases para V y W, sean

$$\mathcal{B}_V = \{v_1, ..., v_n\}, \quad \mathcal{B}_W = \{w_1, ..., w_m\},$$

el conjunto

$$\mathcal{B}_{V \otimes W} = \{v_i \otimes w_j \mid 1 \leq i \leq n,\ 1 \leq j \leq m\}$$

es una base de $V \otimes W$.

En nuestro *setup*, las bases correspondientes serán de utilidad para representar números enteros. En una computadora clásica, representamos un entero $a \in \mathbb{Z}_{\geq 0}$ tal que $a < 2^n$ (es decir, tal que se puede describir con n bits) con el sistema numérico de base 2:

$$a = \sum_{l=0}^{n-1} a_l 2^l$$

donde $a_l \in \{0,1\}$ son los dígitos binarios de a. En una computadora cuántica, también podemos representar un entero $a < 2^n$ con n *qubits* de la siguiente manera:

$$|a\rangle_n = |a_{n-1} \cdots a_1 a_0\rangle = \bigotimes_{l=0}^{n-1} |a_l\rangle$$

Así, por ejemplo, el número 29 se puede representar con 5 qubits (ya que $29 < 2^5$) así:

$$|29\rangle_5 = |11101\rangle = |1\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle .$$

De esta forma, los números enteros siempre se representarán por elementos de la base que se obtiene a partir de los productos tensoriales de las bases computacionales de un solo qubit. Esta base, posteriormente, se denominará también base computacional.

Notación: De ahora en adelante, la notación $|\psi\rangle_n$ implicará que estamos describiendo un sistema de n-qubit (con $n \geq 2$) en lugar de uno de un solo qubit, que seguirá siendo indicado con la ausencia de un subíndice. También haremos uso algunas veces de la notación $|uv\rangle$ para describir el producto tensorial $|u\rangle \otimes |v\rangle$ de dos estados básicos. Ahora que sabemos lo que realmente significan $|\psi\rangle_n$ y $|a\rangle_n$, finalmente estamos en condiciones de comenzar a estudiar los posibles estados de un sistema de múltiples qubits que es, esencialmente, un vector unitario en el espacio de Hilbert correspondiente.

Definición 5. *El estado $|\psi\rangle_n$ de un sistema n-qubit genérico es una superposición (es decir, una combinación lineal) de los 2^n estados de la base computacional $|0\rangle_n, |1\rangle_n, \ldots, |2^n - 1\rangle_n$ con módulo 1. En particular,*

$$|\psi\rangle_n = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle_n ,$$

con amplitudes $\alpha_j \in \mathbb{C}$ cumpliendo

$$\sum_{j=0}^{2^n-1} |\alpha_j|^2 = 1.$$

Esto puede verse como una ventaja obvia con respecto a la computación clásica. En una computadora convencional podemos almacenar uno y solo un entero entre 0 y $2^n - 1$ dentro de un registro de n bits, lo que puede verse como una distribución de probabilidad discreta entre todos los enteros posibles donde el entero que hemos almacenado tiene probabilidad 1 y el resto tiene 0. En un registro cuántico, la probabilidad se puede distribuir entre todos los enteros desde 0 hasta $2^n - 1$, en lugar de tener una sola posibilidad a la hora de leer el registro. Más aún, si vamos a simular este comportamiento cuántico con una computadora clásica, necesitaríamos 2^n registros de n bits, en lugar de un único registro de n-qubit como en el caso cuántico. Este es precisamente uno de los beneficios de la computación cuántica a los que se referia Richard Feynman [Fey82].

2.1.5. Entrelazamiento cuántico

Ejemplo: Veamos el caso más simple. Los estados básicos de un sistema de dos qubits son los productos tensoriales de los estados básicos de un sistema de un solo qubit:

$$|0\rangle_2 = |00\rangle = |0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad |1\rangle_2 = |01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

$$|2\rangle_2 = |10\rangle = |1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad |3\rangle_2 = |11\rangle = |1\rangle \otimes |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Y el estado genérico de dos sistemas diferentes de un solo qubit, descritos de forma independiente, se puede representar como

$$|\psi_0\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

y

$$|\psi_1\rangle = \gamma\,|0\rangle + \delta\,|1\rangle = \gamma \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \delta \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \gamma \\ \delta \end{bmatrix},$$

donde $|\alpha|^2 + |\beta|^2 = 1$ y $|\gamma|^2 + |\delta|^2 = 1$. Esto significa que el estado de este sistema de 2 qubits que surge de ellos debe describirse como el producto tensorial de ambos:

$$|\psi_0\rangle \otimes |\psi_1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \otimes \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha\gamma \\ \alpha\delta \\ \beta\gamma \\ \beta\delta \end{bmatrix}.$$

Por otro lado, si queremos describir un sistema genérico de 2 qubits $|\psi\rangle_2$, tendríamos

$$|\psi\rangle_2 = \alpha_0 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \alpha_1 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix},$$

donde

$$|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$$

debe cumplirse (ya que cualquier sistema cuántico debe describirse como un vector unitario).

Cabe destacar que si nuestro sistema genérico de dos qubits descrito por $|\psi\rangle_2$ se va a descomponer en dos estados de un solo qubit (es decir, $|\psi\rangle_2 = |\psi_0\rangle \otimes |\psi_1\rangle$), entonces

$$\alpha_0 = \alpha\gamma, \;\; \alpha_1 = \alpha\delta, \;\; \alpha_2 = \beta\gamma \text{ y } \alpha_3 = \beta\delta.$$

Es fácil ver que se cumple la igualdad $\alpha_0\alpha_3 = \alpha_1\alpha_2$; sin embargo, está claro que esta condición no se cumple necesariamente en un estado genérico de dos qubits.

Esta es la contrapartida matemática de un conocido fenómeno físico llamado **entrelazamiento cuántico** que implica que el estado cuántico de cada una de las partículas de un sistema de dos qubits no puede describirse de forma independiente. Esto nos lleva a la siguiente definición:

Definición 6. *Un estado general de n-qubit $|\psi\rangle_n$ se denomina mixto o entrelazado si no existen n estados de un qubit $|\psi_0\rangle$,..., $|\psi_n\rangle$ tal que*

$$|\psi\rangle_n = |\psi_0\rangle \otimes ... \otimes |\psi_n\rangle .$$

Observación: El entrelazamiento cuántico se observó por primera vez en la naturaleza en 1935, y en un principio se conocía como la paradoja de Einstein-Podolsky-Rosen. Primeramente fue estudiado por el físico teórico Albert Einstein (1879-1955) y sus colegas Boris Podolsky (1896-1966) y Nathan Rosen (1909-1995) [EPR], más tarde por el físico austriaco Erwin Schrödinger (1887– 1961)[Sch]. Richard Jozsa y Noah Linden [JL] debatieron ampliamente el papel y la importancia del entrelazamiento cuántico en los algoritmos que operan en estados puros y en la aceleración computacional cuántica.

En particular, una computadora cuántica que no aprovecha el entrelazamiento no está muy lejos de una computadora clásica. De hecho, el resultado más interesante que vincula el entrelazamiento cuántico y el rendimiento de la computación cuántica sobre la computación clásica es el siguiente:

Teorema 7. (Gottesman–Knill) *Un algoritmo cuántico que comienza en un estado de base computacional y no presenta entrelazamiento cuántico puede simularse en tiempo polinomial mediante una computadora clásica probabilística. .*

Por lo tanto, es precisamente el entrelazamiento cuántico lo que podría dar una ventaja a la computación cuántica, en comparación con la computación clásica. Uno de los muchos desafíos en el lado del hardware es precisamente crear un entorno lo suficientemente estable para el entrelazamiento, que es un fenómeno muy delicado y frágil.

Observación: Análogamente al caso de un solo qubit, observar un sistema de n-qubit inevitablemente interfiere con $|\psi\rangle_n$ y lo impulsa a colapsar en uno de los vectores de la base computacional (es decir, en $|j\rangle_n$ con $0 \leq j < 2^n$). Este colapso nuevamente no es determinista y se rige por la distribución de probabilidad discreta dada por $|\alpha_j|^2$. Por lo tanto, toda la información que pueda haber sido almacenada en las amplitudes α_j se pierde inevitablemente después del proceso de medición.

Ejemplo: A modo de ilustración, supongamos que tenemos el siguiente sistema cuántico de 3 qubits:

$$|\psi\rangle_3 = \frac{1}{2}|1\rangle_3 + \frac{1}{2}|3\rangle_3 + \frac{1}{2}|5\rangle_3 + \frac{1}{2}|7\rangle_3 .$$

Entonces, si medimos este sistema, obtendremos con idéntica probabilidad uno de estos posibles resultados: 1, 3, 5 o 7. Además, es interesante ver el comportamiento de un

sistema cuántico si, en lugar de medir todos los qubits a la vez, los medimos uno por uno. Nuestro sistema cuántico anterior puede verse como

$$|\psi\rangle_3 = \frac{1}{2}|001\rangle + \frac{1}{2}|011\rangle + \frac{1}{2}|101\rangle + \frac{1}{2}|111\rangle .$$

Pero también como

$$|\psi\rangle_3 = \frac{1}{\sqrt{2}}|0\rangle \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big) + \frac{1}{\sqrt{2}}|1\rangle \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big)$$

o como

$$|\psi\rangle_3 = \Big(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\Big) \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big).$$

Si medimos el primer qubit, tenemos la misma probabilidad de obtener 0 o 1. Sin embargo, como la medición colapsa el estado del qubit, los dos qubits restantes se verán obligados a estar en un estado ligado al que hemos obtenido para el primer qubit. Supongamos que al medir el primer qubit, hemos obtenido un 1. Entonces, nuestro sistema de 3 qubits ha colapsado a

$$|\psi\rangle_3 = |1\rangle \otimes \Big(\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle\Big),$$

que también se puede ver como

$$|\psi\rangle_3 = |1\rangle \otimes \Big(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\Big) \otimes |1\rangle .$$

Hay que tener en cuenta que el tercer qubit ya está en uno de los estados de la base computacional, lo que significa que, si lo medimos ahora mismo, obtendremos con certeza el valor 1. El único qubit restante que no está en la base computacional es el segundo. Mirando el estado actual de nuestro sistema, es fácil ver que tenemos la misma probabilidad de obtener 0 o 1 al medirlo, lo que significa que obtendremos 5 o 7.

Observación: Es interesante ver si el resultado que obtengamos de uno de los qubits condicionará los posibles valores para el resto de qubits. Sea

$$|\psi\rangle_2 = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle) + \frac{1}{\sqrt{2}}(|1\rangle \otimes |1\rangle)$$

uno de los cuatro posibles estados Bell [Bel, Mer]. Este estado se compone de dos qubits entrelazados (no se pueden describir como dos estados de un solo qubit). Si medimos el segundo qubit, tenemos la misma probabilidad de obtener 0 o 1. Sin embargo, si medimos primero el primer qubit y obtenemos 1, entonces el estado del segundo qubit colapsará (sin haberlo observado) a 1, ya que el valor 1 para el segundo qubit solo se tensa con el valor 1 del primer qubit. Así, el resultado que obtengamos de un qubit o de un conjunto de qubits puede estar condicionado por el orden en el que procedamos a medir el resto de qubits. Como se verá más adelante, el orden en que elegimos leer los miembros de un registro cuántico es uno de los aspectos más importantes de un algoritmo cuántico.

2.1.6. Producto interno y externo

Otro conjunto de operaciones entre qubits que son importantes son el producto interno y el externo, que pasamos a definir.

Definición 8. *Sea $|\psi_0\rangle_n$ y $|\psi_1\rangle_n$ dos sistemas n-qubit, el producto interno de $|\psi_0\rangle_n$ y $|\psi_1\rangle_n$ es el producto escalar usual, definido por*

$$\langle\psi_0|\psi_1\rangle_n = |\psi_0\rangle_n^* |\psi_1\rangle_n .$$

El producto interno tiene las siguientes propiedades:

- $\langle\psi_0|\psi_1\rangle_n = \langle\psi_1|\psi_0\rangle_n^*$
- $\langle\psi_0| \ (a\,|\psi_1\rangle + b\,|\psi_2\rangle) \ \rangle_n = a\,\langle\psi_0|\psi_1\rangle_n + b\,\langle\psi_0|\psi_2\rangle_n$
- $\langle\psi|\psi\rangle_n = ||\,|\psi\rangle_n\,||^2$

Definición 9. *Sea $|\psi_0\rangle_n$ y $|\psi_1\rangle_n$ dos sistemas n-qubit, el producto externo de $|\psi_0\rangle_n$ y $|\psi_1\rangle_n$ se define como*

$$|\psi_0\rangle\,\langle\psi_1|_n = |\psi_0\rangle_n\,|\psi_1\rangle_n^*$$

Por ejemplo, sean

$$|\psi_0\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \text{ y } |\psi_1\rangle = \gamma\,|0\rangle + \delta\,|1\rangle = \begin{bmatrix} \gamma \\ \delta \end{bmatrix}$$

dos sistemas genéricos de un solo qubit, entonces las representaciones matriciales del producto interno y externo entre $|\psi_0\rangle$ y $|\psi_1\rangle$ se calculan de la siguiente manera:

$$\langle\psi_0|\psi_1\rangle = \begin{bmatrix} \alpha^* & \beta^* \end{bmatrix} \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \alpha^*\gamma + \beta^*\delta, \qquad |\psi_0\rangle\,\langle\psi_1| = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \begin{bmatrix} \gamma^* & \delta^* \end{bmatrix} = \begin{bmatrix} \alpha\gamma^* & \alpha\delta^* \\ \beta\gamma^* & \beta\delta^* \end{bmatrix}.$$

Observación: Como solo consideramos vectores unitarios se cumple $\langle\psi|\psi\rangle = 1$ para cualquier estado cuántico $|\psi\rangle$.

2.2. Circuitos cuánticos

El lenguaje de los circuitos cuánticos es un modelo de computación equivalente a las máquinas cuánticas de Turing o a las computadoras cuánticas universales. Actualmente, se usan los circuitos cuando se trata de describir un algoritmo que se ejecuta en una máquina cuántica y se basa en una secuencia de mediciones de registros y transformaciones discretas. Esto se debe principalmente a que todos sus elementos pueden ser tratados como clásicos, con la única excepción de la información que circula por los cables.

Primeramente, veremos qué tipo de transformaciones se pueden aplicar al estado de un sistema n-qubit. Como un estado cuántico siempre se representa mediante un vector unitario, necesitamos el operador más general que conserve esta propiedad y la dimensión del vector.

Definición 10. *Una matriz $A \in \mathcal{M}_{\mathbb{C}}(n)$ es unitaria si*

$$A^*A = AA^* = I$$

donde I es la matriz identidad y A^* es el adjunto hermitiano de A.

Definición 11. *El grupo unitario de grado n, denotado por $U_{\mathbb{C}}(n)$, es el grupo de $n \times n$ matrices unitarias, con la multiplicación de matrices como operación de grupo.*

Proposición 12. *Sea $A \in U_{\mathbb{C}}(n)$ una matriz unitaria y sea $x \in \mathbb{C}^n$ un vector unitario, entonces $Ax \in \mathbb{C}^n$ también es un vector unitario.*

En este contexto, una transformación unitaria que actúa sobre n-qubits se denomina puerta cuántica de n-qubits y puede representarse mediante una matriz unitaria. A continuación, ampliaremos este concepto y sus implicaciones físicas.

2.2.1. Puertas cuánticas

Definición 13. *Una puerta cuántica que opera en un espacio de un qubit está representada por una matriz unitaria $A \in U_{\mathbb{C}}(2)$. De manera más general, una puerta cuántica que actúa sobre un sistema de n-qubit se representa mediante una matriz unitaria $A \in U_{\mathbb{C}}(2^n)$.*

Cabe remarcar que las puertas cuánticas tienen necesariamente el mismo número de entradas y salidas, a diferencia de las puertas lógicas clásicas. A partir de ahora, todas las puertas cuánticas se representarán con un símbolo en negrita, para distinguirlas de las meras matrices. Mostraremos como ejemplos las puertas cuánticas más utilizadas, y varios resultados que simplifican de forma espectacular la dificultad de implementar físicamente cualquier puerta cuántica.

Definición 14. *La puerta de Hadamard es una puerta de un solo qubit con la siguiente representación matricial:*

$$\boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

que es unitaria.

Observación: La puerta de Hadamard, aplicada a cada uno de los estados elementales, tiene el siguiente efecto:

$$\boldsymbol{H} : |j\rangle \to \frac{1}{\sqrt{2}} \Big(|0\rangle + (-1)^j |1\rangle \Big).$$

Simbología: El diagrama de flujo clásico para algoritmos se reemplaza en la computación cuántica por la representación del circuito, lo que permite vislumbrar el procedimiento con bastante rapidez. Por ejemplo, la representación del circuito de la puerta de Hadamard es

$$|\psi_0\rangle \; — \boxed{\boldsymbol{H}} — \; |\psi_1\rangle$$

que es la abreviatura de $|\psi_1\rangle \leftarrow \boldsymbol{H}\,|\psi_0\rangle$. El paso de medición se escribe como

$$|\eta_0\rangle \; — \boxed{\;\diagup\;} — \; |\eta_1\rangle$$

Observación: Como se verá al describir los algoritmos cuánticos más relevantes, la transformación de Hadamard es una de las puertas cuánticas más importantes. Su importancia radica en el papel que tiene a la hora de generar todos los estados base posibles, todos ellos con la misma amplitud, dentro de un registro cuántico.

Supongamos que tenemos un qubit cuyo estado cuántico es $|\psi_0\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, donde $|\alpha|^2 + |\beta|^2 = 1$. Entonces, el estado de este sistema de un solo qubit después de aplicarle la puerta de Hadamard es:

$$\boldsymbol{H}\,|\psi_0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} \alpha+\beta \\ \alpha-\beta \end{bmatrix} = \frac{\alpha+\beta}{\sqrt{2}}\,|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}\,|1\rangle$$

Supongamos ahora que, en lugar de tener un estado genérico, tenemos el estado base $|\psi_0\rangle = |0\rangle$ en nuestro sistema de un qubit. En ese caso, el resultado de aplicar la puerta de Hadamard será el siguiente:

$$\boldsymbol{H}\,|\psi_0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle = |+\rangle$$

Como se puede apreciar, hemos transformado un estado base, $|0\rangle$, en una combinación lineal de los dos estados base, $|0\rangle$ y $|1\rangle$, con amplitudes idénticas. Si medimos el qubit en este momento, entonces obtendremos con igual probabilidad uno de los dos estados base posibles. Dicho esto, ¿qué sucederá si en lugar de tener un solo estado cuántico, tenemos un sistema cuántico de n-qubits, todos ellos también en su estado base $|0\rangle$?

$$\boldsymbol{H}^{\otimes n}\,|\psi_0\rangle_n = \boldsymbol{H}^{\otimes n}\,|0\rangle_n = \frac{1}{\sqrt{2^n}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}^{\otimes n}\begin{bmatrix} 1 \\ 0 \end{bmatrix}^{\otimes n} = \frac{1}{\sqrt{2^n}}\begin{bmatrix} 1 \\ 1 \end{bmatrix}^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}|j\rangle_n$$

Lo que hemos obtenido es, gracias al entrelazamiento cuántico, una superposición de todos los estados básicos del sistema con idéntica probabilidad. En otras palabras, si medimos nuestro registro n-qubit en este momento, obtendremos un cierto número entero $j \in \{0, \ldots, 2^n - 1\}$ con probabilidad $1/2^n$.

Definición 15. *Las puertas de Pauli son puertas de un solo qubit con las siguientes matrices:*

$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \boldsymbol{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \boldsymbol{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

que son unitarias, y también hermitianas.

Las matrices de Pauli tienen el siguiente efecto sobre los estados base:

$$\begin{aligned} \boldsymbol{X} : |j\rangle &\longmapsto |1 \oplus j\rangle \\ \boldsymbol{Y} : |j\rangle &\longmapsto (-i)^j\,|1 \oplus j\rangle \\ \boldsymbol{Z} : |j\rangle &\longmapsto (-1)^j\,|j\rangle \end{aligned}$$

Las tres puertas cuánticas anteriores llevan el nombre del físico Wolfgang Pauli (1900 – 1958). Las tres, junto con la matriz identidad I, forman una base para el espacio vectorial

de matrices hermitianas de 2×2 (multiplicadas por coeficientes reales).

Las puertas de un solo qubit (es decir, matrices unitarias de 2×2) se pueden describir completamente mediante el siguiente resultado:

Proposición 16. *Sea $A \in U_{\mathbb{C}}(2)$. Entonces, existen los números reales α, β, γ y δ tales que*

$$A = e^{i\alpha} \begin{pmatrix} e^{-i\beta/2} & \\ & e^{i\beta/2} \end{pmatrix} \begin{pmatrix} \cos(\gamma/2) & -\sin(\gamma/2) \\ \sin(\gamma/2) & \cos(\gamma/2) \end{pmatrix} \begin{pmatrix} e^{-i\delta/2} & \\ & e^{-i\delta/2} \end{pmatrix}$$

Observación: Las matrices del tipo

$$\begin{pmatrix} e^{-i\beta/2} & \\ & e^{i\beta/2} \end{pmatrix}$$

se suelen llamar z–rotaciones, ya que su efecto en un qubit, visto en la esfera de Bloch, corresponde precisamente a una rotación de ángulo β alrededor del eje z. Por razones análogas, las matrices del tipo

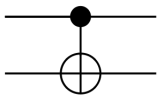
$$\begin{pmatrix} \cos(\gamma/2) & -\sin(\gamma/2) \\ \sin(\gamma/2) & \cos(\gamma/2) \end{pmatrix}$$

se llaman y–rotaciones.

Sin embargo, todas las puertas cuánticas definidas previamente tienen sus limitaciones. De hecho, las puertas cuánticas que son el producto directo de las puertas de un solo qubit no pueden producir entrelazamiento, lo cual es razonable, ya que el entrelazamiento necesita al menos dos qubits para que ocurra.

Definición 17. *La puerta $\boldsymbol{C}_{NOT}$, que significa no-controlado, es una puerta cuántica de dos qubits con la siguiente representación matricial:*

$$\boldsymbol{C}_{NOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Aplicada a un estado base de dos qubits, la puerta $\boldsymbol{C}_{NOT}$ tiene el siguiente efecto:

$$\boldsymbol{C}_{NOT} : |i\rangle \otimes |j\rangle \longmapsto |i\rangle \otimes |i \oplus j\rangle \, .$$

La puerta $\boldsymbol{C}_{NOT}$ es otra de las puertas cuánticas principales, ya que se puede usar para entrelazar y desentrelazar los estados de Bell. De hecho, es la puerta más simple que produce entrelazamiento cuántico. Por ejemplo, sea

$$|\psi\rangle_2 = \frac{1}{\sqrt{2}} \left(|0\rangle_2 + |2\rangle_2 \right)$$

un estado cuántico no mixto (como se puede escribir $|\psi\rangle_2 = |+\rangle \otimes |0\rangle$). Si le aplicamos la puerta $\boldsymbol{C}_{NOT}$, obtenemos

$$|\psi'\rangle_2 = \boldsymbol{C}_{NOT}\left(|\psi\rangle_2\right) = \frac{1}{\sqrt{2}}\Big(\boldsymbol{C}_{NOT}(|0\rangle \otimes |0\rangle) + \boldsymbol{C}_{NOT}(|1\rangle \otimes |0\rangle)\Big) = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle\right),$$

que es uno de los estados de Bell entrelazados.

Teorema 18. *Sea $\boldsymbol{A} \in \mathcal{M}_{\mathbb{C}}(2^n)$ una puerta n-qubit, entonces se puede expresar como un número finito de productos tensoriales de puertas de un bit $M_i \in \mathcal{M}_{\mathbb{C}}(2)$ y la puerta de dos qubits $\boldsymbol{C}_{NOT}$.*

El resultado anterior implica que cada transformación unitaria en un sistema de n-qubits se puede implementar físicamente usando solo puertas de un solo qubit y la puerta $\boldsymbol{C}_{NOT}$. En otras palabras, las puertas de un solo qubit y las puertas $\boldsymbol{C}_{NOT}$ forman un conjunto de puertas universales.

De esta manera, concluimos la explicación de las principales nociones necesarias para la correcta comprensión del resto de este trabajo. Así, un algoritmo que emplee un circuito cuántico consistirá en un conjunto de transformaciones de dos tipos diferentes, mediciones y transformaciones unitarias, a un registro n-qubit.

Capítulo 3

Rompiendo la criptografía RSA

El problema de encontrar el período r de una función f sobre los enteros que es periódica bajo la suma ordinaria, que satisface $f(x) = f(y)$ para x e y distintos si y sólo si x e y difieren en un múltiplo entero de r. Encontrar el período de tal función periódica resulta ser la clave para factorizar productos de grandes números primos. Este es un problema matemático con aplicaciones bastante prácticas.

Uno podría pensar que encontrar el período de tal función periódica debería ser fácil, pero eso es solo porque cuando uno piensa en funciones periódicas tiende a imaginar funciones continuas que varían lentamente (como la función seno) cuyos valores en una pequeña muestra de puntos, dentro un período, pueden dar pistas sobre ese posible período. Pero el tipo de función periódica a tener en cuenta aquí es una función en los enteros cuyos valores dentro de un período r son aleatorios de un entero al siguiente y, por lo tanto, no da ninguna pista sobre el valor de r.

Los algoritmos clásicos más conocidos para encontrar el período r de tal función tardan un tiempo que crece más rápido que cualquier potencia del número n de bits de r (exponencialmente como $n^{1/3}$). Pero en 1994, Peter Shor [Shob, Shoa] descubrió que se puede usar el poder de una computadora cuántica para descubrir ese período r, en un tiempo que escala solo un poco más rápido que n^3.

Debido a que la capacidad de encontrar períodos de manera eficiente, combinada con algunos trucos de teoría de números, permite factorizar eficientemente el producto de dos números primos grandes, da lugar a que el descubrimiento del algoritmo cuántico de Shor sea de gran interés práctico.

El gran esfuerzo computacional que requieren todas las técnicas de factorización clásicas conocidas es el principal motivo por el cual el método de encriptación RSA es seguro [RSA].

Por lo tanto, cualquier ordenador que pueda romper este encriptado RSA de manera eficiente será una enorme amenaza para la seguridad de comunicaciones, tanto militares como comerciales, siendo esta la principal razón de que sea tan interesante la realización de este trabajo.

Así pues, aunque a primera vista parezca que estos sencillos trucos de teoría de números, que subyacen en el método de encriptación RSA, no tengan nada que ver directamente con la computación cuántica, una mirada más profunda a la base del cifrado RSA revela

que el algoritmo de búsqueda de períodos de Shor se puede usar para su desencriptación.

3.1. Preliminares de teoría de números

Las entidades algebraicas básicas detrás del cifrado RSA son los grupos finitos, donde la operación de grupo es la multiplicación módulo algún número entero fijo N. En aritmética módulo-N todos los enteros que difieren en múltiplos de N están identificados, por lo que sólo hay N cantidades distintas, que pueden ser representadas por 0, 1, . . ., $N-1$. Por ejemplo $5 \times 6 \equiv 2 \pmod 7$ ya que $5 \times 6 = 30 = 4 \times 7 + 2$. Se escribe $\equiv \pmod N$ para enfatizar que la igualdad es sólo hasta un múltiplo de N, reservándose = para la igualdad estricta. Sea G_N el conjunto de todos los enteros positivos menores que N (incluido 1) que no tienen factores en común con N. Dado que la factorización en números primos es única, el producto de dos números en G_N (ya sea el ordinario o el producto módulo-N) tampoco tiene factores en común con N, por lo que G_N es cerrado bajo multiplicación módulo N. Si a, b y c están en G_N con $ab \equiv ac \pmod N$, entonces $a(b-c)$ es múltiplo de N, y como a no tiene factores en común con N, debe ser que $b-c$ es un múltiplo de N, entonces $b \equiv c \pmod N$. Así, la operación de multiplicación módulo N por un miembro fijo a de G_N toma valores distintos de G_N en valores distintos, por lo que la operación simplemente permuta los miembros del grupo finito G_N. Dado que 1 pertenece a G_N, debe haber alguna d en G_N que satisfaga $ad = 1$, es decir, a debe tener un inverso multiplicativo en G_N. Por lo tanto, G_N cumple las condiciones de ser un grupo bajo multiplicación módulo-N. Todo miembro a de un grupo finito G se caracteriza por su orden k, el entero más pequeño para el cual (en el caso de G_N)

$$a^k \equiv 1 (mod N) \tag{3.1}$$

Se cumple que el orden de cada miembro de G es un divisor del número de miembros de G (el *orden* de G). Si p es un número primo, entonces el grupo G_p contiene $p-1$ números, ya que ningún entero positivo menor que p tiene factores en común con p. Como $p-1$ es entonces un múltiplo del orden k de cualquier a en G_p, se sigue de (3.1) que cualquier entero a menor que p satisface

$$a^p - 1 \equiv 1 (mod p) \tag{3.2}$$

Esta relación, conocida como *el pequeño teorema de Fermat*, se extiende a enteros a no divisibles por p, ya que cualquier a es de la forma $a = mp + a'$ con m entero y a' menor que p. El cifrado RSA usa una extensión del pequeño teorema de Fermat para un caso caracterizado por *dos* números primos distintos, p y q. Si un entero a es no es divisible ni por p ni por q, entonces ninguna potencia de a es divisible por ningún p o q. Dado que, en particular, a^{q-1} no es divisible por p, concluimos de (3.2) que

$$[a^{q-1}]^{p-1} \equiv 1 (mod p) \tag{3.3}$$

Por la misma razón,

$$[a^{p-1}]^{q-1} \equiv 1 (mod q) \tag{3.4}$$

Las relaciones (3.3) y (3.4) establecen que $a^{(q-1)(p-1)} - 1$ es un múltiplo de p y de q. Dado que p y q son primos distintos, por lo tanto debe ser un múltiplo de pq, y por lo tanto

$$a^{(q-1)(p-1)} \equiv 1(modpq) \tag{3.5}$$

Como una derivación alternativa de (3.5), dado que a no es divisible ni por p ni por q, no tiene factores en común con pq y por lo tanto pertenece a G_{pq}. El número de elementos de G_{pq} es $pq - 1 - (p-1) - (q-1) = (p-1)(q-1)$, ya que hay $pq-1$ enteros menores que pq, entre los que se encuentran $p-1$ múltiplos de q y otros distintos $q-1$ múltiplos de p. Se sigue la ecuación (3.5) porque el orden $(p-1)(q-1)$ de G_{pq} debe ser un múltiplo del orden de a. Obtenemos la versión de (3.5) que es la base para el cifrado RSA tomando cualquier potencia integral s de (3.5) y multiplicando ambos lados por a:

$$a^{1+s(q-1)(p-1)} \equiv a(modpq) \tag{3.6}$$

La relación (3.6) se cumple incluso para números enteros a que son divisibles por p o q. Se cumple trivialmente cuando a es un múltiplo de pq. Y si a es divisible por solo uno de p y q, sea $a = kq$. Como a no es divisible por p ni tampoco es ninguna potencia de a, entonces, el pequeño teorema de Fermat nos dice que $[a^{s(q-1)}]^{p-1} = 1 + np$ para algún número entero n. Al multiplicar ambos lados por a tenemos $a^{1+s(q-1)(p-1)} \equiv a + nap \equiv a + nkqp$, entonces (3.6) se continua manteniendo. Nótese finalmente que si c es un número entero que no tiene ningún factor en común con $(p-1)(q-1)$ entonces c está en $G_{(p-1)(q-1)}$ y por lo tanto tiene una inversa en $G_{(p-1)(q-1)}$; es decir, hay una d en $G_{(p-1)(q-1)}$ que satisface

$$cd \equiv 1(mod(p-1)(q-1)) \tag{3.7}$$

Entonces, para algún entero s,

$$cd = 1 + s(p-1)(q-1) \tag{3.8}$$

En vista de (3.8) y (3.6), cualquier entero a debe satisfacer que

$$a^{cd} \equiv a(modpq) \tag{3.9}$$

Así que si

$$b \equiv a^c(modpq) \tag{3.10}$$

entonces

$$b^d \equiv a(modpq) \tag{3.11}$$

Las igualdades aritméticas elementales resumidas en este apartado constituyen la base completa para el cifrado RSA [RSA].

3.2. Algoritmo RSA

Bob quiere recibir un mensaje de Alice codificado de manera que solo él pueda léerlo. Para hacer esto, elige dos números primos grandes (digamos de 200 dígitos) p y q. Le da a Alice, a través de un canal público, su producto $N = pq$ y c, un número de codificación muy grande que ha elegido de tal manera que no tenga factores en común con $(p-1)(q-1)$. Sin embargo, no revela los valores separados de p ni q y, dada la imposibilidad práctica de factorizar un número de 400 dígitos con las computadoras actualmente disponibles, él está bastante seguro de que ni Alice ni Eve, quien quiere interceptar y descifrar la comunicación, serán capaces de calcular p y q conociendo solo su producto N. Bob, sin embargo, como sabe p y q, y por lo tanto $(p-1)(q-1)$, puede encontrar el inverso multiplicativo d de c mod $(p-1)(q-1)$, que satisface (3.7). Mantiene d estrictamente para sí mismo para su uso en la decodificación.

Alice codifica un mensaje representándolo como una cadena de menos de 400 dígitos usando, por ejemplo, alguna versión de codificación ASCII. Si su mensaje requiere más de 400 dígitos, lo corta en pedazos más pequeños. Ella interpreta cada cadena como un número a menor que N. Usando el número de codificación c y el valor de $N = pq$ que recibió de Bob, calcula $b \equiv a^c (mod pq)$ y se lo envía a Bob a través de un canal público. Cuando recibe b, Bob usa su conocimiento privado de d para calcular $b^d (mod pq)$, cosa que por (3.11) le asegura que es el mensaje original de Alice a.

Si Eve, que escuchaba a escondidas, pudiera encontrar los factores p y q de N, ella podría calcular $(p-1)(q-1)$ y encontrar el entero decodificador d de la codificación disponible públicamente c, de la misma manera que lo hizo Bob. Pero factorizar un número tan grande como N está mucho más allá de las capacidades computacionales clásicas. La búsqueda eficiente de períodos es de interés en esta configuración criptográfica no solo porque conduce directamente a la factorización efectiva, sino también porque puede llevar a Eve directamente a una manera alternativa de decodificar el mensaje de Alice b sin que ella lo sepa o tenga que calcular los factores p y q de N. Así es como funciona:

Eve usa su eficiente máquina de búsqueda de períodos para calcular el orden r del mensaje codificado públicamente disponible de Alice $b = a^c$ en G_{pq}. Ahora el orden r del mensaje codificado de Alice $b = a^c$ en G_{pq} es el mismo que el orden de a. Esto se debe a que el subgrupo de G_{pq} generado por a contiene $a^c = b$, y por lo tanto contiene el subgrupo generado por b; pero el subgrupo generado por b contiene $b^d = a$, y por lo tanto el subgrupo generado por a. Como cada subgrupo contiene al otro, debe ser idéntico. Como el orden de a o b es el número de elementos en el subgrupo que genera, sus órdenes son los mismos. Así que si Eve puede encontrar el orden r del mensaje b de Alice, entonces ella también puede saber el orden del texto original de Alice a.

Como Bob ha elegido c para que no tenga factores en común con $(p-1)(q-1)$, y dado que r divide el orden $(p-1)(q-1)$ de G_{pq}, el entero de codificación c no puede tener factores en común con r. Entonces c es congruente módulo r a un miembro c' de G_r, que tiene un inverso d' en G_r, y d' es también un inverso módulo-r de c:

$$cd' \equiv 1 (mod r) \tag{3.12}$$

Por lo tanto, dado c (que Bob ha anunciado públicamente) y r (que Eve lo puede obtener

con su programa de búsqueda del período a partir del mensaje codificado b de Alice y el valor anunciado públicamente de $N = pq$), es fácil para Eve calcular d' con una computadora clásica, usando, módulo r, la misma extensión del algoritmo de Euclides, como Bob usó para encontrar d, módulo $(p-1)(q-1)$. Entonces se sigue que para algún entero m

$$b^{d'} \equiv a^{cd'} = a^{1+mr} = a(a^r)^m \equiv a(mod pq) \tag{3.13}$$

Por lo tanto, Eve ha usado su habilidad de encontrar el período para decodificar el mensaje encriptado de Alice $b = a^c$, para así revelar el mensaje original a de Alice.

3.3. Transformada cuántica de Fourier

3.3.1. Transformada de Fourier sobre grupos Abelianos finitos

Para el grupo $\mathbb{Z}/N\mathbb{Z}$, el grupo de enteros modulo N con la suma, la *Transformada Cuántica de Fourier* (QFT) es una operación unitaria en $F_{\mathbb{Z}/N\mathbb{Z}}$. Su efecto en un estado de la base $|x\rangle$ para cualquier $x \in \mathbb{Z}/N\mathbb{Z}$ es

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y \in \mathbb{Z}/N\mathbb{Z}} \omega_N^{xy} |y\rangle , \tag{3.14}$$

donde $\omega_N := e^{2\pi i/N}$ denota una N-raíz primitiva de la unidad.

En general, un grupo Abeliano finito G tiene $|G|$ representaciones unidimensionales irreducibles distintas $\psi \in \hat{G}$.

Aclaración: Una representación es *irreducible* si no se puede descomponer como la suma directa de otras dos representaciones. Cualquier representación de un grupo finito G puede escribirse como una suma directa de representaciones irreducibles de G. Salvo isomorfismo, G tiene un número finito de representaciones irreducibles. El símbolo $\hat{G}$ denota un conjunto completo de representaciones irreducibles de G, una para cada tipo de isomorfismo. Estas representaciones $\psi \in \hat{G}$ son funciones $\psi : G \to \mathbb{C}$ con $\psi(a+b) = \psi(a)\psi(b)$ para todo $a, b \in G$. La transformada cuántica de Fourier F_G sobre G actua como

$$|x\rangle \mapsto \frac{1}{\sqrt{|G|}} \sum_{\psi \in \hat{G}} \psi(x) |\psi\rangle \tag{3.15}$$

para cada $x \in G$.

Por ejemplo, el grupo $(\mathbb{Z}/N\mathbb{Z}) \times (\mathbb{Z}/N\mathbb{Z})$ tiene N^2 representaciones irreducibles definidas por $\psi_{y_1,y_2} : (x_1, x_2) \mapsto \omega_N^{x_1y_1+x_2y_2}$ para todo $y_1, y_2 \in \mathbb{Z}/N\mathbb{Z}$; por lo tanto su transformada cuántica de Fourier $F_{(\mathbb{Z}/N\mathbb{Z})\times(\mathbb{Z}/N\mathbb{Z})}$ es

$$|x_1, x_2\rangle \mapsto \frac{1}{N} \sum_{y_1,y_2 \in \mathbb{Z}/N\mathbb{Z}} \omega_N^{x_1y_1+x_2y_2} |y_1, y_2\rangle \tag{3.16}$$

para todo $x_1, x_2 \in \mathbb{Z}/N\mathbb{Z}$. En este ejemplo, $F_{(\mathbb{Z}/N\mathbb{Z})\times(\mathbb{Z}/N\mathbb{Z})}$ se puede escribir como el producto tensorial $F_{\mathbb{Z}/N\mathbb{Z}} \otimes F_{\mathbb{Z}/N\mathbb{Z}}$. En general, de acuerdo con el teorema fundamental de los grupos finitos Abelianos, cualquier grupo abeliano finito G puede expresarse como

un producto directo de subgrupos cíclicos de orden potencia prima, $G \cong (\mathbb{Z}/p_1^{r_1}\mathbb{Z}) \times \cdots \times (\mathbb{Z}/p_k^{r_k}\mathbb{Z})$, y QFT sobre G se puede escribir como el producto tensorial de QFTs $F_{\mathbb{Z}/p_1^{r_1}\mathbb{Z}} \otimes \cdots \otimes F_{\mathbb{Z}/p_k^{r_k}\mathbb{Z}}$.

La transformada de Fourier F_G es útil para explotar la simetría con respecto a G. Se considera el operador P_s que suma $s \in G$, definido por $P_s|x\rangle = |x+s\rangle$ para cualquier $x \in G$. Este operador es diagonal en la base de Fourier:

$$F_G P_s F_G^\dagger = \sum_{\psi \in \hat{G}} \psi(s)|\psi\rangle\langle\psi|. \tag{3.17}$$

Por lo tanto, las mediciones en la base de Fourier producen las mismas estadísticas para un estado puro $|\phi\rangle$ y su desplazamiento $P_s|\phi\rangle$. De manera equivalente, un estado mixto invariable en G está diagonalizado por F_G.

3.3.2. Circuito cuántico eficiente para la QFT sobre $\mathbb{Z}/2^n\mathbb{Z}$

Para usar la transformada de Fourier sobre G como parte de un cálculo cuántico eficiente, debemos implementarla (aproximadamente) mediante un circuito cuántico de tamaño $\text{poly}(\log|G|)$. De hecho, esto se puede hacer para cualquier grupo abeliano finito. En esta sección explicamos una construcción para el caso del grupo $\mathbb{Z}/2^n\mathbb{Z}$.

Transformando de la base de estados $\{|x\rangle : x \in G\}$ a la base $\{|\psi\rangle : \psi \in \hat{G}\}$, la matriz que representa la transformación de Fourier sobre $\mathbb{Z}/N\mathbb{Z}$ es

$$F_{\mathbb{Z}/N\mathbb{Z}} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2N-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2N-2} & \cdots & \omega_N^{(N-1)(N-1)} \end{pmatrix}. \tag{3.18}$$

De manera compacta,

$$F_{\mathbb{Z}/N\mathbb{Z}} = \frac{1}{\sqrt{N}} \sum_{x,y \in \mathbb{Z}/N\mathbb{Z}} \omega_N^{xy} |y\rangle\langle x|, \tag{3.19}$$

donde $|y\rangle$ representa el estado base correspondiente al carácter ψ_y con $\psi_y(x) = \omega_N^{xy}$. Es sencillo verificar que $F_{\mathbb{Z}/N\mathbb{Z}}$ es de hecho una transformación unitaria, es decir, que $F_{\mathbb{Z}/N\mathbb{Z}} F_{\mathbb{Z}/N\mathbb{Z}}^\dagger = F_{\mathbb{Z}/N\mathbb{Z}}^\dagger F_{\mathbb{Z}/N\mathbb{Z}} = 1$.

Supongamos ahora que $N = 2^n$, y representemos el entero $x \in \mathbb{Z}/N\mathbb{Z}$ por n bits $x_0, x_1, \ldots, x_{n-1}$ donde $x = \sum_{j=0}^{n-1} 2^j x_j$. La transformada de Fourier de $|x\rangle$ se puede escribir

como el producto tensorial de n qubits, ya que

$$F_{\mathbb{Z}/2^n\mathbb{Z}} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{y\in\{0,1\}^n} \omega_{2^n}^{x(\sum_{j=0}^{n-1} 2^j y_j)} |y_0, \ldots, y_{n-1}\rangle \tag{3.20}$$

$$= \frac{1}{\sqrt{2^n}} \bigotimes_{j=0}^{n-1} \sum_{y_j\in\{0,1\}} \mathrm{e}^{2\pi \mathrm{i}\, x y_j/2^{n-j}} |y_j\rangle \tag{3.21}$$

$$= \bigotimes_{j=0}^{n-1} \frac{|0\rangle + \mathrm{e}^{2\pi\mathrm{i}\sum_{k=0}^{n-1} 2^{j+k-n} x_k} |1\rangle}{\sqrt{2}} \tag{3.22}$$

$$=: \bigotimes_{j=0}^{n-1} |z_j\rangle\,. \tag{3.23}$$

Ahora, debido a que $\exp(2\pi\mathrm{i}\, 2^s x_k) = 1$ para todos los enteros $s \geq 0$, vemos que el j-ésimo qubit de salida es

$$|z_j\rangle = \frac{1}{\sqrt{2}}(|0\rangle + \mathrm{e}^{2\pi\mathrm{i}(2^{j-n}x_0 + 2^{j+1-n}x_1 + \cdots + 2^{-1}x_{n-1-j})} |1\rangle), \tag{3.24}$$

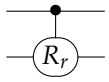
y por lo tanto solo depende de los $n - j$ bits de entrada $x_0, \ldots, x_{n-1-j}$.

Para describir un circuito cuántico que implementa la transformada de Fourier, definimos la rotación de fase de un solo qubit

$$R_r := \begin{pmatrix} 1 & 0 \\ 0 & \mathrm{e}^{2\pi\mathrm{i}/2^r} \end{pmatrix} \quad \simeq \quad \text{—}(R_r)\text{—} \tag{3.25}$$

y la rotación controlada de dos qubits - una puerta que está condicionada a un solo qubit de control y aplica las rotaciones necesarias en nuestro qubit actual:

$$\Lambda(R_r) := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \mathrm{e}^{2\pi\mathrm{i}/2^r} \end{pmatrix} \quad \simeq \quad \tag{3.26}$$



actuando simétricamente sobre a y $b \in \{0, 1\}$ como $\Lambda(R_r) |a, b\rangle = \mathrm{e}^{2\pi\mathrm{i}ab/2^r} |a, b\rangle$. El circuito que se muestra en la Figura 3.1 usa $\binom{n}{2}$ de estas puertas junto con n puertas de Hadamard para implementar exactamente la transformada cuántica de Fourier sobre $\mathbb{Z}/2^n\mathbb{Z}$.

En este circuito hay muchas rotaciones de pequeños ángulos que no afectan significativamente al resultado final. Simplemente omitiendo las puertas $\Lambda(R_r)$ con $r = \Omega(\log n)$, obtenemos un circuito de tamaño $O(n \log n)$ (en lugar de $O(n^2)$ para el circuito original) que implementa la QFT con precisión $1/\operatorname{poly}(n)$.

3.3.3. Estimación de fase y QFT aplicada a un grupo Abeliano finito

Además de ser directamente aplicable a los algoritmos cuánticos, como el algoritmo de Shor, la QFT sobre $\mathbb{Z}/2^n\mathbb{Z}$ proporciona una primitiva de computación cuántica útil llamada *estimación de fase*, abreviado como QPE. En el problema de estimación de fase, tenemos

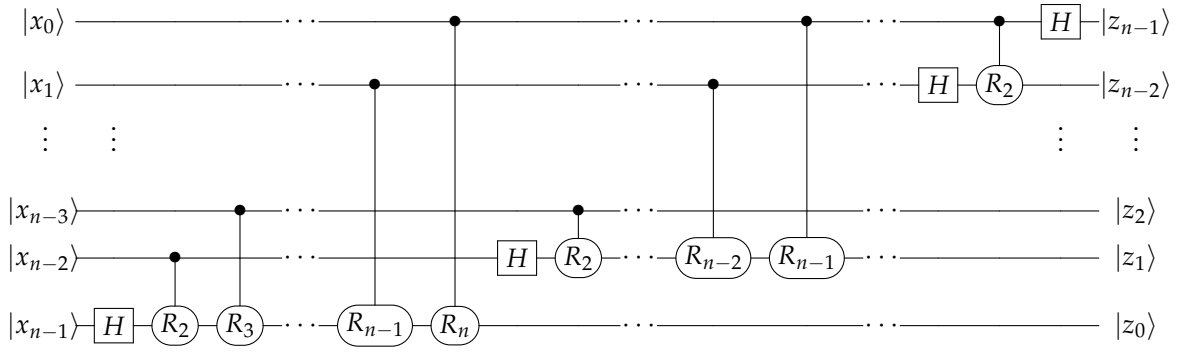


Figura 3.1: Un circuito cuántico eficiente (de tamaño $O(n^2)$) para la transformada cuántica de Fourier sobre $\mathbb{Z}/2^n\mathbb{Z}$. Teniendo en cuenta que el orden de los bits de salida n $z_0, \ldots, z_{n-1}$ se invierte, en comparación con el orden de los n bits de entrada $x_0, \ldots, x_{n-1}$.

un operador unitario U (ya sea como un circuito explícito o como una caja negra que nos permite aplicar una operación controlada de U^x para valores enteros de x). También se nos da un estado $|\phi\rangle$ que promete ser un vector propio de U, a saber, $U|\phi\rangle = e^{i\phi}|\phi\rangle$ para unos $\phi \in \mathbb{R}$. El objetivo es generar una estimación de ϕ con la precisión deseada. (Por supuesto, también podemos aplicar el procedimiento a un estado general $|\psi\rangle$; por linealidad, obtenemos cada valor ϕ con probabilidad $|\langle\phi|\psi\rangle|^2$.) El procedimiento para la estimación de fase es sencillo:

Si la expansión binaria de $\phi/2\pi$ termina después de un máximo de n bits, se garantiza que el resultado será la expansión binaria de $\phi/2\pi$. En general, obtenemos una buena aproximación con alta probabilidad.

La complejidad del algoritmo puede depender de la forma del operador unitario U. Si solo se nos da una caja negra para la puerta U controlada, entonces puede que no haya mejor manera de implementar la operación U^x controlada que realizando una puerta U controlada x veces, de modo que el tiempo de ejecución sea $\Theta(2^n)$.

Una aplicación útil de la estimación de fase es implementar el QFT sobre un grupo cíclico arbitrario $\mathbb{Z}/N\mathbb{Z}$. El circuito presentado en la sección anterior solo funciona cuando N es una potencia de dos (o, con una ligera generalización, una potencia de algún otro número entero fijo).

Nos gustaría realizar la transformación que mapea $|x\rangle \mapsto |\hat{x}\rangle$, donde $|\hat{x}\rangle := F_{\mathbb{Z}/N\mathbb{Z}}|x\rangle$ denota un estado de la base de Fourier. Por linealidad, si la transformación actúa correctamente sobre una base, entonces actúa correctamente sobre todos los estados. Es sencillo realizar la transformación: $|x, 0\rangle \mapsto |x, \hat{x}\rangle$ (que crea una superposición uniforme $\sum_{y\in\mathbb{Z}/N\mathbb{Z}} |y\rangle / \sqrt{N}$ en el segundo registro y aplica el desfase controlado $|x, y\rangle \mapsto \omega_N^{xy} |x, y\rangle$), pero queda por borrar el primer registro).

Consideramos el operador unitario P_1 que suma 1 módulo N, es decir, $P_1|x\rangle = |x + 1\rangle$ para cualquier $x \in \mathbb{Z}/N\mathbb{Z}$. Los estados propios de este operador son precisamente los estados de la base de Fourier $|\hat{x}\rangle$, con valores propios ω_N^x. Por lo tanto, usando la estimación de fase en P_1 (con $n = O(\log N)$ bits de precisión), podemos aproximar la transformación $|\hat{x}, 0\rangle \mapsto |\hat{x}, x\rangle$. Invirtiendo esta operación, podemos borrar $|x\rangle$, dando la QFT deseada. Hay que tener en cuenta que podemos realizar en P_1^x pasos de $\text{poly}(\log N)$ incluso cuan-

Algoritmo 1 Estimación de fase

Input: Estado propio $|\phi\rangle$ (con valor propio $e^{i\phi}$) de un operador unitario dado U.
Problema: Dar una estimación de n bits de ϕ.

1. Preparar el ordenador cuántico en el estado
$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}/2^n\mathbb{Z}} |x\rangle \otimes |\phi\rangle. \tag{3.27}$$
2. Aplicar operador unitario
$$\sum_{x \in \mathbb{Z}/2^n\mathbb{Z}} |x\rangle\langle x| \otimes U^x, \tag{3.28}$$
dado el estado
$$\frac{1}{\sqrt{2^n}} \sum_{x \in \mathbb{Z}/2^n\mathbb{Z}} e^{i\phi x} |x\rangle \otimes |\phi\rangle. \tag{3.29}$$
3. Aplicar la transformada de Fourier inversa en el primer registro, dando
$$\frac{1}{2^n} \sum_{x,y \in \mathbb{Z}/2^n\mathbb{Z}} \omega_{2^n}^{x(\frac{2^n}{2\pi}\phi - y)} |y\rangle \otimes |\phi\rangle. \tag{3.30}$$
4. Medir el primer registro del estado resultante en la base computacional.

do x es exponencialmente grande en $\log N$, por lo que el procedimiento resultante es verdaderamente eficiente.

Dada la transformada de Fourier sobre $\mathbb{Z}/N\mathbb{Z}$, es sencillo implementar QFT sobre un grupo abeliano finito arbitrario usando la descomposición del grupo en factores cíclicos.

Si las puertas se pueden realizar en paralelo, es posible realizar la QFT mucho más rápido, usando solo $O(\log \log N)$ pasos de tiempo.

3.4. Algoritmo de Shor

Dado que el algoritmo cuántico de búsqueda de períodos (period-finding) de Shor siempre se describe como un algoritmo de factorización, concluimos este capítulo observando cómo la búsqueda del período conduce a la factorización. Por lo tanto, nuestro algoritmo cuántico consistira en la estimación de la fase de la función periódica $f(x) = a^x \pmod N$, utilizando la exponenciacion modular y aplicando la Transformada de Fourier inversa, y en encontrar el período r aplicando un algoritmo de fracciones contínuas que a partir de la fase nos devuelva el orden de a módulo N. Aquí consideraremos únicamente el caso relacionado con la encriptación RSA, en el que se quiere factorizar el producto de dos números primos grandes, $N = pq$, aunque la conexión entre la determinación del período y la factorización es más general.

Una vez tenemos una forma de determinar el período (algoritmo 1), lo siguiente que queremos es encontrar los factores primos de $N = pq$. Así pues, elegimos un número alea-

torio a coprimo con N. Las probabilidades de que una a aleatoria resulte ser un múltiplo de p o de q son minúsculas cuando p y q son enormes (en el caso altamente improbable de que a sea un múltiplo de p o q, entonces el algoritmo de Euclides aplicado a a y N dará p o q directamente, y se habrá factorizado N). Usando nuestro algoritmo de búsqueda de períodos, encontraremos el orden de a en G_{pq}: la r más pequeña para cual

$$a^r \equiv 1(modpq) \tag{3.31}$$

Podemos usar esta información para factorizar N si nuestra elección de a fue afortunada.

Supongamos primero que hemos tenido la suerte de obtener una r que es par. Entonces podemos calcular

$$x = a^{r/2}(modpq) \tag{3.32}$$

y teniendo en cuenta que

$$0 \equiv x^2 - 1 \equiv (x-1)(x+1)(modpq) \tag{3.33}$$

Ahora $x - 1 = a^{r/2} - 1$ no es congruente con 0 módulo pq, ya que r es la potencia más pequeña de a congruente con 1. Supongamos además, que por segunda volvemos a tener suerte, y se cumple

$$x + 1 = a^{r/2} + 1 \not\equiv 0(modpq) \tag{3.34}$$

En ese caso, ni $x - 1$ ni $x + 1$ son divisibles por $N = pq$, pero (3.33) nos dice que su producto si lo es. Como p y q son primos esto es posible sólo si uno de ellos, digamos p, divide $x - 1$ y el otro, q , divide $x + 1$. Como los únicos divisores de N son p y q, se deduce que p es el máximo común divisor de N y $x - 1$, mientras q es el máximo común divisor de N y $x + 1$. Por lo tanto, podemos encontrar p o q por aplicación directa del algoritmo de Euclides.

Así que todo se reduce a la probabilidad de que tengamos suerte en el momento de escoger a. En el siguiente lema (ver Lema 19), se puede ver que la probabilidad es de al menos 0.5 de que un número aleatorio a en G_{pq} tenga un orden r que sea par con $a^{r/2} \not\equiv -1(modpq)$. Por lo tanto, solo tendríamos que repetir unas pocas veces la ejecución del algoritmo cuántico de Shor para conseguir una probabilidad de éxito muy alta. En el siguiente capítulo veremos cómo podemos mejorar esto.

Lema 19. *Supongamos que a se elige uniformemente al azar de* $(\mathbb{Z}/N\mathbb{Z})^\times$*, donde N es un número entero impar con al menos dos factores primos distintos. Entonces con probabilidad de al menos* $1/2$*, el orden multiplicativo r de a módulo N es par, y* $a^{r/2} \neq -1$ *(mod N).*

Demostración. Supongamos que $N = p_1^{m_1} \cdots p_k^{m_k}$ es la factorización de N en potencias de $k \geq 2$ primos impares distintos. Por el teorema chino del resto, hay valores únicos $a_i \in \mathbb{Z}/p_i^{m_i}\mathbb{Z}$ tal que $a = a_i \pmod{p_i^{m_i}}$. Sea r_i el orden multiplicativo de a_i módulo $p_i^{m_i}$, y sea 2^{c_i} la mayor potencia de 2 que divide a r_i. Se cumple que si r es impar o si $a^{r/2} = -1 \pmod{N}$, entonces $c_1 = \cdots = c_k$. Como $r = mcm(r_1, \ldots, r_k)$, tenemos $c_1 = \cdots = c_k = 0$ cuando r es impar. Por otro lado, si r es par y $a^{r/2} = -1 \pmod{N}$, entonces para cada

i tenemos $a^{r/2} = -1 \pmod{p_i^{m_i}}$, entonces r_i no divide $r/2$; pero sabemos que r/r_i es un número entero, por lo que debe ser impar, lo que implica que cada r_i tiene el mismo número de potencias de 2 en su descomposición en factores primos.

Ahora afirmamos que la probabilidad de que cualquier c_i dado tome cualquier valor particular es como mucho $1/2$, lo que implica que $\Pr(c_1 = c_2) \leq 1/2$, y sigue la conclusión deseada. Para ver esto, consideramos a elegido uniformemente al azar de $(\mathbb{Z}/N\mathbb{Z})^\times$, o de manera equivalente, cada a_i elegido uniformemente al azar de $(\mathbb{Z}/p_i^{m_i}\mathbb{Z})^\times$. El orden del último grupo es $\varphi(p_i^{m_i}) = (p_i - 1)p_i^{m_i} = 2^{d_i}q_i$ para algún entero positivo d_i y algún entero impar q_i. Él número de $a_i \in (\mathbb{Z}/p_i^{m_i}\mathbb{Z})^\times$ de orden impar es q_i, y el número de a_i con cualquier $c_i \in \{1, \ldots, d_i\}$ en particular es $2^{c_i - 1}q_i$. En particular, el evento de mayor probabilidad es $c_i = d_i$, cosa que sucede con probabilidad de solo $1/2$. □

A continuación explicaremos el algoritmo de Shor para determinar el orden de un entero a módulo N. Por simplicidad, limitaremos la explicación al ejemplo concreto de $N = 15$.

3.4.1. Ejemplo: Factorización de 15

Queremos encontrar los factores primos de 15 usando el algoritmo de factorización de Shor. Primero, el algoritmo elige un entero aleatorio a tal que $1 < a < 15$. Supongamos que obtenemos $a = 13$, que cumple la primera condición: $\text{mcd}(13, 15) = 1$. Ahora es el momento de encontrar el orden multiplicativo r de 13 módulo 15, lo cual se logra con la ayuda de una computadora cuántica.

Calculamos n e inicializamos el sistema cuántico. En este caso, $n = \lceil \log_2 15 \rceil = 4$. Cabe notar que en el caso general, en realidad necesitariamos $2n$ qubits de entrada para el registro $|x\rangle$:

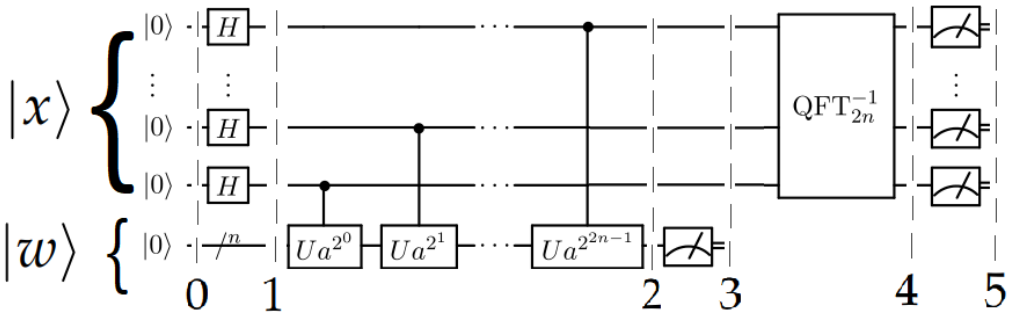


Figura 3.2: Esquema del circuito cuántico que implementa el algoritmo de Shor

Así, nuestros registros cuánticos $|x\rangle_4$ y $|w\rangle_4$ tienen los siguientes estados iniciales:

Paso 0: $\|\psi_0\rangle_{4,4} \leftarrow \|0\rangle_4 \otimes \|0\rangle_4$

A continuación, aplicamos la transformación de Hadamard al primer registro, obteniendo así una superposición de todos los estados básicos en este, todos con amplitudes idénticas. De esta forma, todos los enteros entre 0 y $2^4 - 1$ ahora están *en algún lugar* en el primer registro. Por lo tanto, hemos pasado de tener los qubits inicializados en $|0\rangle$ a tenerlos en el estado $|+\rangle$.

Paso 1: $|\psi_1\rangle_{4,4} \leftarrow (\boldsymbol{H}^{\otimes 4} \otimes \boldsymbol{I}^{\otimes 4}) \left(|\psi_0\rangle_{4,4}\right)$

$$|\psi_1\rangle_{4,4} = \frac{1}{\sqrt{2^4}} \sum_{j=0}^{2^4-1} |j\rangle_4 \otimes |0\rangle_4$$

Posteriormente, aplicamos la puerta cuántica que calcula las potencias de 13 módulo 15, que tiene el siguiente efecto.

Paso 2: $|\psi_2\rangle_{4,4} \leftarrow \boldsymbol{U}_{13,15} \left(|\psi_1\rangle_{4,4}\right)$

$$|\psi_2\rangle_{4,4} = \frac{1}{\sqrt{2^4}} \sum_{j=0}^{2^4-1} \boldsymbol{U}_{13,15} \left(|j\rangle_4 \otimes |0\rangle_4\right) = \frac{1}{4} \sum_{j=0}^{15} |j\rangle_4 \otimes \left|13^j \bmod 15\right\rangle_4$$

Si expandimos la suma, alternativamente podemos expresar el resultado de la siguiente manera (teniendo en cuenta que, a modo de simplificación, se han omitido los subíndices y los operadores del producto tensorial):

$$\begin{aligned} |\psi_2\rangle = \tfrac{1}{\sqrt{2^4}} \big(&|0\rangle|1\rangle + |1\rangle|13\rangle + |2\rangle|4\rangle + |3\rangle|7\rangle \quad + \\ &|4\rangle|1\rangle + |5\rangle|13\rangle + |6\rangle|4\rangle + |7\rangle|7\rangle \quad + \\ &|8\rangle|1\rangle + |9\rangle|13\rangle + |10\rangle|4\rangle + |11\rangle|7\rangle \quad + \\ &|12\rangle|1\rangle + |13\rangle|13\rangle + |14\rangle|4\rangle + |15\rangle|7\rangle \big) \end{aligned}$$

Después de analizar el resultado anterior, podemos observar que los valores del segundo registro son periódicos. Si hacemos factor común, terminamos con el estado:

$$\begin{aligned} |\psi_2\rangle = \tfrac{1}{\sqrt{2^4}} \big(&(|0\rangle + |4\rangle + |8\rangle + |12\rangle)|1\rangle \quad + \\ &(|1\rangle + |5\rangle + |9\rangle + |13\rangle)|13\rangle \quad + \\ &(|2\rangle + |6\rangle + |10\rangle + |14\rangle)|4\rangle \quad + \\ &(|3\rangle + |7\rangle + |11\rangle + |15\rangle)|7\rangle \big) \end{aligned}$$

Gracias a esta representación, es más fácil entender qué sucederá si medimos el segundo registro.

Paso 3: $\tilde{\omega} \leftarrow$ medimos el segundo registro | Es decir, el output del registro $|w\rangle$

$|\psi_3\rangle_4 \leftarrow |\psi_2\rangle_{4,4}$ después de haber medido el segundo registro

Es claro que obtendremos un valor $\tilde{\omega}$ tal que $\tilde{\omega} = 13^{\tilde{j}} \pmod{15}$ para una cierta $\tilde{j} \in \{0, \ldots, 2^4 - 1\}$. Así, $\tilde{\omega} \in \{1, 13, 4, 7\}$ (que son las potencias de 13 módulo 15). Supongamos que al realizar la medición obtenemos $\tilde{\omega} = 7$. El registro colapsará en $|7\rangle$ y todos los demás valores posibles serán destruidos y desaparecerán para siempre, la información sobre el resto de posibles potencias de 13 módulo 15 se perderá. Sin embargo, el primer registro ($|x\rangle_4$) también colapsará en los valores que fueron tensorizados con $|7\rangle$, descartando así los restantes. Lo que nos interesa ver es que todos los estados de base tensorizados con $|7\rangle$ se corresponden con los exponentes $\tilde{j}$ tales que $\tilde{\omega} = 13^{\tilde{j}} \pmod{15}$. Más específicamente:

$$|\psi_3\rangle_4 = \frac{1}{\sqrt{4}}\Big(|3\rangle_4 + |7\rangle_4 + |11\rangle_4 + |15\rangle_4 \Big) = \frac{1}{2}\left(\sum_{a=0}^{3} |4a+3\rangle_4 \right)$$

Nos damos cuenta que ha surgido un patrón, ya que los estados básicos en el primer registro muestran un comportamiento periódico. Este período se corresponde naturalmente con el orden multiplicativo de 13 módulo 15, que resulta ser igual a 4. No obstante, esta información aún sigue oculta para nosotros, ya que aquí solo estamos utilizando algunos conocimientos del problema para proporcionar una explicación matemática del rendimiento del algoritmo en este caso particular. Por otro lado, la constante c=4, que aparece en $\frac{1}{\sqrt{c}}$, es solo una normalización de las amplitudes después del colapso del registro, correspondiente al número total de exponentes que devuelven 7 módulo 15 entre 0 y $2^4 - 1$.

Como se ha explicado anteriormente, para obtener el período que esta en el interior de nuestro sistema cuántico, hay que usar la inversa de la transformada cuántica de Fourier:

Paso 4: $|\psi_4\rangle_4 \leftarrow \boldsymbol{QFT}^{+}{}_4\,(|\psi_3\rangle_4)$

Recordemos que:

$$QFT^{+}\,|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} (e^{\frac{-2\pi i}{N} xy}\,|y\rangle)$$

Así pues, aplicado a cada unos de los posibles estados restantes:

$$QFT^{+}\,|3\rangle_4 = \frac{1}{\sqrt{16}} \sum_{y=0}^{15} (e^{\frac{-2\pi i}{16} 3y}\,|y\rangle) = \frac{1}{4} \sum_{y=0}^{15} (e^{\frac{-\pi i}{8} 3y}\,|y\rangle)$$

$$QFT^{+}\,|7\rangle_4 = \frac{1}{4} \sum_{y=0}^{15} (e^{\frac{-\pi i}{8} 7y}\,|y\rangle)$$

$$QFT^{+}\,|11\rangle_4 = \frac{1}{4} \sum_{y=0}^{15} (e^{\frac{-\pi i}{8} 11y}\,|y\rangle)$$

$$QFT^{+} |15\rangle_4 = \frac{1}{4} \sum_{y=0}^{15} (e^{\frac{-\pi i}{8} 15y} |y\rangle)$$

Juntando todo y resolviendo, obtenemos:

$$|\psi_4\rangle_4 = \frac{1}{8} \sum_{y=0}^{15} [e^{-i\frac{3\pi}{8}y} + e^{-i\frac{7\pi}{8}y} + e^{-i\frac{11\pi}{8}y} + e^{-i\frac{15\pi}{8}y}] |y\rangle = \frac{1}{8} [4 |0\rangle_4 + 4i |4\rangle_4 - 4 |8\rangle_4 - 4i |12\rangle_4]$$

Aquí, podemos apreciar como gracias a la aplicación de la transformada cuántica de Fourier, y su efecto de interferencia, solo los términos $|\tilde{x}\rangle$ con

$$\tilde{x} = j2^n / r$$

tienen una amplitud significativa (pico) para j un entero aleatorio que va de 0 a $r - 1$

Paso 5: $\tilde{x} \leftarrow$ medimos $|\psi_4\rangle_4$! Es decir, el output del registro $|x\rangle$ y obtenemos $\tilde{x}$

Al medir el primer registro, obtenemos $\tilde{x}$, que es un valor no determinista de uno de los cuatro picos. Con una probabilidad razonable, la aproximación de fracción continua de $2^n / \tilde{x}$ será un múltiplo entero del período r. Como podemos obtener, $\tilde{x}$ = $|0\rangle$ o $|4\rangle$ o $|8\rangle$ o $|12\rangle$, con una probabilidad 0,25 para cada uno de los casos, entonces:

Si medimos $|0\rangle$, no nos aporta información y tenemos que repetir el proceso.

Si medimos $|4\rangle$, obtenemos $j\frac{16}{r} = 4$, por lo tanto $r = 4$ si $j = 1$, esto nos da los factores 5 y 3.

Si medimos $|8\rangle$, tenemos $j\frac{16}{r} = 8$, por lo tanto $r = 4$ si $j = 2$, obteniendo el resultado anterior, o si $r = 2$ si $j = 1$, esto nos da una solución parcial ya que obtenemos los factores 3 y 1.

Si medimos $|12\rangle$, obtenemos $j\frac{16}{r} = 12$, por lo tanto $r = 4$ si $j = 3$, esto nos da los factores 5 y 3.

Así pues, parece que funciona correctamente, permitiéndonos extraer los factores de forma exitosa, en $\frac{3}{4}$ de los casos. En el caso de obtener 0, la solución más fácil es volver a repetir el proceso hasta obtener la r adecuada.

Ahora explicaremos formalmente el proceso de fracción continua. Recordamos que

una fracción continua se representa como

$$[a_0; a_1, \ldots, a_K] = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\cdots + \cfrac{1}{a_K}}}}}$$

donde $a_0 \in \mathbb{Z}_{\geq 0}$ y $a_1, \ldots, a_K \in \mathbb{Z}_{>0}$. Sea $[a_0; a_1, \ldots, a_K]$, entonces existe un único $q \in \mathbb{Q}_{>0}$ tal que $q = [a_0; a_1, \ldots, a_K]$. Definimos el k-ésimo iterado de $[a_0; a_1, \ldots, a_K]$, donde $0 \leq k < K$, como

$$q_k = [a_0; a_1 \ldots, a_k].$$

Dado un cierto $q \in \mathbb{Q}_{>0}$, podemos calcular los elementos de su fracción continua correspondiente de la siguiente manera:

$$\begin{aligned} a_0 &\leftarrow \lfloor q \rfloor \\ q_0 &\leftarrow q - a_0 \\ a_{k+1} &\leftarrow \lfloor 1/q_k \rfloor \\ q_{k+1} &\leftarrow 1/q_k - a_{k+1} \end{aligned}$$

Además, cada uno de los iterados q_k se puede expresar como $q_k = b_k/c_k$, donde $\gcd(b_k, c_k) = 1$ y

$$\begin{aligned} b_0 &\leftarrow a_0 \\ c_0 &\leftarrow 1 \\ b_1 &\leftarrow a_0 a_1 + 1 \\ c_1 &\leftarrow a_1 \\ b_{k+2} &\leftarrow a_{k+2} b_{k+1} + b_k \\ c_{k+2} &\leftarrow a_{k+2} c_{k+1} + c_k \end{aligned}$$

Se puede demostrar que podemos obtener el periodo r a partir de $\tilde{x}$ con el siguiente algoritmo: a partir de $k = 1$, calculamos b_k y c_k como se explicó anteriormente a partir de la fracción continua de $q = \tilde{x}/2^n$. Luego comprobamos si

$$x^{c_k} \equiv 1 (mod N).$$

Si la respuesta es afirmativa, hemos obtenido el orden r; si no, lo hacemos de nuevo por $k + 1$.

Queda por ver si $a = 13$ y $r = 4$ cumplen las condiciones finales del algoritmo. Como 4 es un número par y $13^{4/2} + 1 = 5 \not\equiv 0 (mod 15)$, finalmente podemos continuar con el último paso. Como se explicó, podemos calcular ahora dos factores no triviales de 15:

$$d_1 = \gcd(3, 15) = 3$$

y

$$d_2 = \gcd(5, 15) = 5,$$

que de hecho son los únicos factores primos de 15. Así termina el algoritmo de Shor para este caso particular.

$$15 = 3 \times 5$$

Cabe recordar, que en el caso general del algoritmo hacen falta $2n$ qubits en el registro de entrada $|x\rangle$ en vez de n qubits. Esto lo veremos aplicado en la implementación práctica.

Capítulo 4

Factorización eficiente con una sola ejecución de Shor

4.1. Introducción

Ahora, una vez hemos visto como funciona el algoritmo de factorización de Shor, y basándonos en el trabajo de Martín Ekerå et al. [Eke], pasaremos a explicar cómo con una única pasada del algoritmo cuántico de factorización de Shor podemos llegar a obtener todos los factores. Así pues, todos los factores primos de N se pueden recuperar con un coste computacional negligible en un post-procesado en un ordenador clásico. El algoritmo clásico que se necesita para este paso se debe esencialmente a Miller. [Mila]

Por lo tanto, mostraremos que dado el orden de un único elemento escogido uniformemente al azar de $\mathbb{Z}/N\mathbb{Z}$, podemos encontrar, con alta probabilidad y para cualquier entero N, la factorización completa de este N en tiempo polinomial.

4.1.1. Notación

En lo que sigue, usaremos la siguiente notación

$$N = \prod_{i=1}^{n} p_i^{e_i}$$

sea un entero de m bits, con $n \geq 2$ factores primos distintos p_i, con exponentes enteros positivos e_i. Diremos que un algoritmo factoriza N si calcula un factor no trivial de N, y que factoriza completamente N si calcula el conjunto $\{p_1, \ldots, p_n\}$.

Sea ϕ la función ϕ de Euler, tal que $\varphi(n) = |\{m \in \mathbb{N} | m \leq n \wedge \text{mcd}(m, n) = 1\}|$. Sea λ la función de Carmichael, que se define como el menor entero m tal que cumple: $a^m \equiv 1(\text{mod } n)$ para cada número entero a coprimo con n. Y sea $\lambda'(N) = mcm(p_1 - 1, \ldots, p_n - 1)$. Además, sea $\mathbb{Z}_N^\star$ el grupo multiplicativo de $\mathbb{Z}_N$, el anillo de los enteros módulo N, y sean ln y log los logaritmos natural y de base dos, respectivamente. Denotaremos por $[a, b]$ el rango de números enteros desde a hasta b incluyendo los extremos. Y denotaremos por

$g \in Z_N^\star$ el entero aleatorio que en el capítulo anterior hemos denominado a cuyo orden es r.

A lo largo de este capítulo, supondremos que N es impar por razones técnicas de demostración. Esto no implica una pérdida de generalidad: es fácil cumplir con este requisito utilizando la división trivial. De hecho, en general siempre se eliminarían los factores primos pequeños antes de recurrir a algoritmos de factorización más elaborados. Nótese además que la determinación del orden se puede realizar antes de que se aplique la división trivial a N si se desea.

Existen pruebas de primalidad probabilística eficientes, como Miller-Rabin[Mila, Rab], y algoritmos eficientes para reducir potencias perfectas $z = q^e$ a q: una opción simple es probar si $z^{1/d}$ es un número entero para algún $d \in [2, logz]$.

4.2. El algoritmo

Dado el orden r de g, en general podemos adivinar correctamente los órdenes de una gran fracción de los otros elementos de $\mathbb{Z}_N^\star$ con alta probabilidad.

Para ver por qué sucede esto, hay que tener en cuenta que es probable que g tenga un orden tal que $\lambda(N)/r$ sea un producto de tamaño moderado de factores primos pequeños. Por lo tanto, al multiplicar o dividir pequeños factores primos con r, podemos adivinar $\lambda(N)$ y, por extensión, los órdenes de otros elementos en el grupo. En este apartado, sin embargo, vamos un poco más allá:

En particular, en lugar de adivinar el orden de los elementos individuales en $\mathbb{Z}_N^\star$, adivinamos algún múltiplo de $\lambda'(N)$. Además, mostramos que incluso si solo conseguimos adivinar algún múltiplo de un divisor de $\lambda'(N)$, a menudo conseguimos recuperar la factorización completa de N.

A continuación, describimos un algoritmo clásico, esencialmente debido a Miller[Milb], para factorizar completamente N dado un múltiplo de $\lambda'(N)$. Sin embargo, lo hemos modificado ligeramente al agregar un paso en el que intentamos adivinar dicho múltiplo, denotado r', dado el orden r de g.

Además, seleccionamos k elementos del grupo x_j uniformemente al azar de $\mathbb{Z}_N^\star$, para $k \geq 1$ algún parámetro pequeño que se puede seleccionar de manera libre, mientras que Miller itera sobre todos los elementos hasta cierto límite.

Con estas modificaciones, probaremos que el algoritmo probabilístico resultante se ejecuta en tiempo polinomial, con la posible excepción de la llamada a un algoritmo de búsqueda de órdenes en el primer paso, y analizaremos su probabilidad de éxito.

Para ser específicos, el algoritmo resultante primero ejecuta el siguiente procedimiento una vez para encontrar factores no triviales de N: (Ver algoritmo 2)

Algoritmo 2 Variante de Shor aplicando Miller

Input: N (número entero a factorizar); c (constante entera $k \geq 1$).
Problema: Encontrar factores no triviales de N

1. Seleccionar g uniformemente al azar de $\mathbb{Z}_N^\star$.
   Calcular el orden r de g a través de un algoritmo de búsqueda de orden, en nuestro caso Shor.

2. Sea $P(B)$ el conjunto de primos $\leq B$.

   Sea $\eta(q, B)$ el entero más grande tal que $q^{\eta(q,B)} \leq B$.

   Sea $m' = cm$ para alguna constante $c \geq 1$ que puede seleccionarse libremente. (Donde m es la longitud en bits de N).

   Calcular $r' = r \prod_{q \in P(m')} q^{\eta(q,m')}$.

3. Sea $r' = 2^t o$ donde o es impar.

4. Para $j = 1, 2, \ldots, k$ para algún $k \geq 1$ que se puede seleccionar libremente hacer:

   4.1. Seleccionar x_j uniformemente al azar de $\mathbb{Z}_N^\star$.

   4.2. Para $i = 0, 1, \ldots, t$ hacer:

   4.2.1. Calcular $d_{i,j} = mcd(x_j^{2^i o} - 1, N)$.
   Si $1 < d_{i,j} < N$ informar $d_{i,j}$ como un factor no trivial de N.

Luego obtenemos la factorización completa a partir de la $d_{i,j}$ reportada de la siguiente manera:

Se inicializa un conjunto y se le agrega N antes de ejecutar el algoritmo anterior. Para cada factor no trivial reportado, el factor se agrega al conjunto. El conjunto se mantiene reducido, de manera que solo contenga factores coprimos no triviales por pares. Además se comprueba para cada factor del conjunto, si es una potencia perfecta q^e, en cuyo caso q se reporta como un factor no trivial. El algoritmo tiene éxito si el conjunto contiene todos los factores primos distintos de N cuando el algoritmo se detiene.

Cabe recordar de la introducción que existen métodos eficientes para reducir q^e a q, y métodos para probar la primalidad en tiempo polinomial probabilístico.

4.2.1. Notas sobre una implementación eficiente

Cabe notar que el algoritmo tal y como se describe en la Secc.4.2 no está optimizado: más bien, se presenta la base fundamental de este para facilitar su comprensión y análisis.

En una implementación real, sería beneficioso, por ejemplo, realizar el módulo aritmético N' a lo largo del paso 4 del algoritmo, para N' un divisor compuesto de N que carece de factores primos que ya se han encontrado.

Por supuesto, el algoritmo también dejaría de incrementar j tan pronto como se complete la factorización, en lugar de después de k iteraciones, y dejaría de incrementar i tan pronto como $x_j^{2^i o} \equiv 1 (mod N')$ en lugar de continuar hasta t. También, se seleccionaría x_j

y g de $\mathbb{Z}_N^\star \setminus \{1\}$ en lugar de $\mathbb{Z}_N^\star$. En el paso 4.2.1, se calcularía $d_{i,j} = mcd(u_{i,j} - 1, N')$, donde $u_{0,j} = x_j^o (mod N')$, y $u_{i,j} = u_{i-1,j}^2 (mod N')$ para $i \in [1, t]$, para evitar elevar x_j a o repetidamente.

Ideas para posibles optimizaciones

Para acelerar aún más las exponenciaciones, en lugar de elevar cada x_j a una suposición r' calculada previamente para un múltiplo de $\lambda'(N)$, un exponente más pequeño que es un múltiplo del orden de $x_j (mod N')$ se puede calcular especulativamente y usarse en lugar de r'. Para obtener más factores no triviales de cada x_j, cabe la posibilidad de agotar las combinaciones de divisores pequeños del exponente; no solo las potencias de dos que dividen el exponente.

Factores faltantes:

Si un x_j es tal que $w_j = x_j^{N'r'} \not\equiv 1 (mod N')$, falta un factor q igual al orden de w_j (mod N') en la suposición de r' para un múltiplo de $\lambda'(N)$. Si esto lleva al algoritmo a fallar en factorizar completamente N', puede valer la pena intentar calcular el factor faltante:

Las opciones disponibles incluyen la búsqueda de q mediante la exponenciación de todos los números primos hasta cierto límite, lo que es esencialmente análogo a aumentar c, o usar alguna forma de algoritmo de búsqueda de ciclos que no requiera que se conozca un múltiplo de q de antemano.

En resumen, hay una serie de optimizaciones que se pueden aplicar, pero hacerlo más arriba oscurecería el funcionamiento del algoritmo y el análisis que estamos a punto de presentar. Además, no es necesario, ya que el algoritmo descrito ya es muy eficiente.

4.2.2. Notas sobre la búsqueda de órdenes para un múltiplo de N

Hay que tener en cuenta que la llamada de búsqueda de orden en el paso 1 se puede realizar para un múltiplo de N si se desea. Esto solo puede hacer que r crezca en algún múltiplo, lo que a su vez solo puede servir para aumentar la probabilidad de éxito, de la misma manera que hacer crecer r en r' en el paso 2 sirve para aumentar la probabilidad de éxito (consultar la Secc. 4.3). A su vez, esto explica por qué podemos reemplazar N por N' como se describe en la sección anterior, y por qué una restricción a N impar no implica una pérdida de generalidad.

4.2.3. Notas sobre analogías con las obras de Miller, Rabin y Long

La versión original de Miller del algoritmo en la Secc.4.2 es determinista y se ha demostrado que funciona solo asumiendo la validez de la hipótesis de Riemann extendida (ERH), al igual que la prueba de primalidad de Miller en la misma tesis [Mila].

Rabin [Rab] luego convirtió la prueba de primalidad de Miller en un algoritmo probabilístico de tiempo polinomial que es altamente eficiente en la práctica. Casi al mismo tiempo, Long [Lon81], reconociendo las ideas de Flajolet, convirtió el algoritmo de factorización de Miller en un algoritmo probabilístico en tiempo polinomial. Más específicamente, Long establece límites inferiores para la probabilidad de seleccionar aleatoriamente un

elemento $g \in \mathbb{Z}_N^\star$ de orden r como múltiplo de $\lambda'(N)$. Luego da una versión aleatoria del algoritmo de Miller para dividir N en dos factores no triviales dado este múltiplo de $\lambda'(N)$.

Lo anterior está íntimamente relacionado con este caso. Sin embargo, aquí se llevan las cosas un paso más allá al convertir el algoritmo de factorización de Miller en un algoritmo de tiempo polinomial probabilístico eficiente para recuperar la factorización completa de N dado el orden r de un solo elemento g seleccionado uniformemente al azar de $\mathbb{Z}_N^\star$. Limitamos inferiormente la probabilidad de que el algoritmo logre recuperar todos los factores de N dado r. Mostramos que esta probabilidad es muy alta, considerando cuidadosamente los casos donde r es un múltiplo de un divisor de $\lambda'(N)$.

4.2.4. Notas sobre analogías con las obras de Pollard

El algoritmo de Miller se puede considerar como una generalización del algoritmo $p-1$ de Pollard [Pol]: Miller esencialmente ejecuta el algoritmo de Pollard para todos los factores primos p_i en paralelo usando que se conoce un múltiplo de $\lambda'(N) = mcm(p_1 - 1, \ldots, p_n - 1)$. Pollard, suponiendo que no tiene conocimiento previo, utiliza un producto de pequeñas potencias primas hasta cierto límite B-suave en potencias en lugar de $\lambda'(N)$. Esto factoriza p_i de N si $p_i - 1$ es B-suave en potencias, cosa que da nombre al algoritmo de Pollard.

Como solo conocemos r, un múltiplo de algún divisor de $\lambda'(N)$, hacemos crecer r en r' multiplicando un producto de pequeñas potencias primas. Esto está en analogía con el enfoque de Pollard.

4.3. Análisis del algoritmo

Una diferencia clave entre el algoritmo modificado en la [Secc. 4.2] y el algoritmo original de la tesis de Miller es que en el de Ekerå se calcula el orden de un g aleatorio y luego se añade un paso de suposición: Se adivina un r en el paso 2 que esperamos que sea un múltiplo de $p_i - 1$ para todo $i \in [1, n]$, y si no para todos, al menos para todos menos uno de los índices de este intervalo, en cuyo caso el algoritmo seguirá teniendo éxito en la factorización completa de N.

Esto se muestra en el siguiente análisis. Específicamente, se acota inferiormente la probabilidad de éxito y se demustra que el tiempo de ejecución del algoritmo es polinomial. A lo largo del análisis, usaremos la notación implícitamente introducida en el pseudocódigo del algoritmo de la Secc. 4.2.

Además, usamos que, para un g seleccionado uniformemente al azar entre

$$\mathbb{Z}_N^\star \simeq \mathbb{Z}_{p_1}^\star e_1 \times \cdots \times \mathbb{Z}_{p_n}^\star e_n,$$

tenemos que $g \pmod{p_i}$ para $i \in [1, n]$ se seleccionan de forma independiente y uniforme al azar dentro de los grupos cíclicos $\mathbb{Z}_{p_i}^\star$. Lo mismo ocurre con x_j en lugar de g.

Definición 20. *El primo p_i se dice desafortunado si r no es un múltiplo de $p_i - 1$.*

Lema 21. *La probabilidad de que p_i sea desafortunado es como mucho $\log p_i/(m' \log m')$.*

Demostración. Para que p_i sea desafortunado, tiene que existir una potencia prima q^e tal que

(i) $q^e > m'$, ya que q^e divide de otro modo a r',

(ii) q^e divide a $p_i - 1, y$

(iii) g es una potencia q^e (mod p_i), que reduce el orden de g (mod p_i) por un factor q^e.

El número de dichas potencias primas q^e que dividen $p_i - 1$ es como mucho $logp_i/logm'$ por una simple comparación de tamaño, ya que q^e>m' y el producto de las potencias primas en cuestión no puede exceder $p_i - 1$. Para cada una de esas potencias primas, la probabilidad de que g sea una potencia q^e mod p_i es como máximo $1/q^e \leq 1/m'$. El lema sigue tomando el producto de estas dos expresiones. □

Lema 22. *Si a lo sumo un p_i es desafortunado, entonces excepto con probabilidad a lo sumo $2^{-k}\binom{n}{2}$ todos los n factores primos de N serán recuperados por el algoritmo después de k iteraciones.*

Demostración. Para que el algoritmo no encuentre todos los factores primos, deben existir dos factores primos q_1 y q_2 distintos que dividan a N, de modo que para todas las combinaciones de $i \in [0, t]$ y $j \in [1, k]$, ambos factores dividen $x_j^{2^i o} - 1$, o ninguno de ellos divide $x_j^{2^i o} - 1$.

Para ver por qué sucede esto, hay que tener en cuenta que, de lo contrario, los dos factores se dividirán para alguna combinación de i y j en el paso 4.2.1 del algoritmo de la Secc. 4.2, y si esto ocurre por parejas para todos los factores, el algoritmo recuperará todos los factores.

Hay $\binom{n}{2}$ formas de seleccionar dos primos distintos de los n primos distintos que dividen a N. Para cada par, como máximo uno de q_1 y q_2 es desafortunado, según la formulación del lema.

(i) Si q_1 o q_2 son desafortunados:

Sin pérdida de generalidad, digamos que q_1 es afortunado y q_2 es desafortunado.

- El número primo afortunado q_1 divide $x_j^{2^t o} - 1$. Para ver por qué sucede esto, hay que recordar que $x_j \in \mathbb{Z}_N^\star$ y que q_1-1 divide r' ya que q_1 es afortunado, entonces

$$x_j^{2^t o} = x_j^{r'} \equiv 1 (mod q_1).$$

- El primo desafortunado q_2 divide $x_j^{2^t o}$-1 si y solo si $x_j^{2^t o} \equiv 1 (mod \mathrm{q}_2)$.

Para x_j seleccionado uniformemente al azar de $\mathbb{Z}_N^\star$, y q_2 impar, donde recordamos que asumimos que N y, por lo tanto, q_2 eran impares en la introducción, este evento ocurre con una probabilidad máxima de $1/2$.

Para ver por qué ocurre esto, hay que tener en cuenta que dado que q_2 es desafortunado, solo un elemento x_j con un módulo de orden q_2 que se reduce del orden máximo q_2-1 por algún factor que divida q_2-1 puede cumplir la condición. El factor de reducción debe ser al menos dos. De ello se deduce que como máximo la mitad de los elementos de $\mathbb{Z}_N^\star$ pueden cumplir la condición.

Para cada iteración $j \in [1, k]$, la probabilidad de falla es, por lo tanto, como máximo $1/2$.

Dado que hay k iteraciones, la probabilidad total de falla es como máximo 2^{-k}.

(ii) Si tanto q_1 como q_2 son afortunados:

En este caso, tanto q_1 como q_2 dividen $x_j^{2^t o}$-1, ya que $x_j \in \mathbb{Z}_N^\star$, y dado que $r' = 2^t o$ donde q_1-1 y q_2-1 dividen r', entonces

$$x_j^{2^t o} = x_j^{r'} \equiv 1(mod q)$$

para $q \in \{q_1, q_2\}$.

El algoritmo falla si y solo si x_j^o tiene el mismo orden módulo q_1 como q_2.

Para ver por qué esto es así, hay que ver

$$d_{i,j} = mcd(x_j^{2^i o} - 1, N)$$

se calcula en el paso 4.2.1 del algoritmo, para $i \in [0, t]$, y que el primo $q \in \{q_1, q_2\}$ divide $d_{i,j}$ si y solo si $x_j^{2^i o} \equiv 1(mod q)$. Solo si esto ocurre para el mismo i tanto para q_1 como para q_2, q_1 y q_2 no se dividirán, es decir, si x_j^o tiene el mismo orden módulo q_1 como para q_2.

Para analizar la probabilidad de que x_j^o tenga el mismo orden módulo q_1 y q_2, dejemos que 2^{t_1} y 2^{t_2} sean las mayores potencias de dos que dividan q_1-1 y q_2-1, respectivamente. Recordamos además que supusimos que N, y por lo tanto q_1 y q_2, eran impares en la introducción. Esto implica que podemos suponer que $t \geq t_1 \geq t_2 \geq 1$ sin pérdida de generalidad.

Consideramos $x_j^{2^{t_1-1} o}$:

Si $t_1 = t_2$, la probabilidad de que $x_j^{2^{t_1-1} o}$-1 sea divisible por q_1 pero no por q_2 es $1/4$, y viceversa para q_2 y q_1. Por lo tanto, la probabilidad es como mucho $1/2$ de que x_j^o tenga el mismo orden módulo q_1 como q_2.

Si $t_1 > t_2$, la probabilidad de que $x_j^{2^{t_1-1} o}$-1 sea divisible por q_1 es $1/2$, mientras que lo mismo ocurre siempre con q_2. Por lo tanto, la probabilidad es de nuevo como máximo $1/2$ de que x_j^o tenga el mismo orden módulo q_1 y q_2.

Para cada iteración $j \in [1, k]$, la probabilidad es de nuevo como máximo $1/2$.

Dado que hay k iteraciones, la probabilidad total de falla es como máximo 2^{-k}.

El lema se deriva del argumento anterior, ya que hay $\binom{n}{2}$ combinaciones con una probabilidad máxima de 2^{-k} cada una. □

Por definición, se dice que q divide u si y solo si $u \equiv 0(mod q)$. Teniendo en cuenta que esto implica que todos los $q \neq 0$ dividen $u = 0$. Esta situación surge en la demostración anterior del Lema 22.

Lema 23. *Al menos dos números primos son desafortunados con probabilidad como máximo*

$$\frac{1}{2c^2 log^2 cm}$$

Demostración. Las tres condiciones usadas para establecer el límite superior en la demostración del Lema 21 son independientes para dos primos p_i distintos. Por lo tanto, por la demostración del Lema 21, tenemos que la probabilidad de que al menos dos números primos sean desafortunados está acotada por arriba por

$$\sum_{(i_1,i_2)\in S} \frac{logp_{i_1}}{m'logm'} \frac{logp_{i_2}}{m'logm'} \leq \frac{1}{2(m'logm')^2} (\sum_{i=1}^{n} logp_i)^2 \leq \frac{1}{2c^2log^2cm}$$

donde usamos que $\sum_{i=1}^{n} logp_i \leq logN \leq m$ y $m' = cm$, y donde S es el conjunto de todos los pares $(i_1, i_2) \in [1, n]^2$ tal que el producto $p_{i_1} \times p_{i_2}$ es distinto, y por lo que se sigue el lema. □

4.3.1. Análisis del tiempo de ejecución

Proposición 24. *Se cumple que log r'= O(m).*

Demostración. Por el teorema de los números primos, hay O(m'/ln m') primos menores que m'. Como r<N tenemos log r<m. Además, como cada potencia prima q^e en r'/r es menor que m', tenemos

$$log(r) \leq log(r) + O(m'/ln(m'))log(m') = O(m)$$

como $m' = cm$ para alguna constante $c \geq 1$, por lo que sigue la proposición. □

Teorema 25. *El algoritmo de factorización, con la posible excepción de la llamada única de búsqueda de orden, factoriza completamente N en tiempo polinomial, excepto con probabilidad como máximo*

$$2^{-k} \binom{n}{2} \frac{1}{2c^2log^2cm}$$

donde n es el número de factores primos distintos de N, m es la longitud de bits de N, $c \geq 1$ es una constante que puede seleccionarse libremente y k es el número de iteraciones realizadas en el posprocesamiento clásico.

Demostración. Es fácil ver que la parte del algoritmo que no busca el orden se ejecuta en un tiempo polinomial en m, ya que todos los números enteros tienen una longitud $O(m)$, incluido en particular r' según proposición 24. El teorema se sigue del análisis de la Secc. 4.3, sumando el límite superior de la probabilidad de que ocurra una falla cuando, como máximo, un primo tiene es desafortunado en el Lema 22, y de la probabilidad de que al menos dos primos sean desafortunados en el Lema 23. □

Por el teorema principal anterior, el algoritmo tendrá éxito en la factorización completa de N, si la constante c se selecciona de modo que

$$\frac{1}{2c^2log^2cm}$$

sea lo suficientemente pequeño, y si 2^{-k} para k, el número de iteraciones es lo suficientemente pequeño en relación con $\binom{n}{2}$ para n el número de factores primos distintos de N. El último requisito es fácil de cumplir: se escoge $k \geq 2logn - 1 + \tau$ para algún τ positivo. Entonces $2^{-k}\binom{n}{2}) \leq 2^{-\tau}$.

La complejidad temporal del algoritmo está dominada por k exponenciaciones de un entero módulo N a un exponente de longitud $O(m)$ bits, y por la necesidad de probar los factores identificados para la primalidad. Esto es de hecho muy eficiente.

Hay que tener en cuenta que este análisis de la probabilidad de éxito del algoritmo es un análisis del peor de los casos. En la práctica, la probabilidad real de éxito del algoritmo es mayor. Además, nada en estos argumentos requiere estrictamente que c sea una constante: podemos hacer que c sea una función de m para aumentar aún más la probabilidad de éxito a expensas de trabajar con exponentes de $O(c(m)m)$ bits.

Capítulo 5

Implementación práctica

Una vez vistas con detalle las bases teóricas sobre las que se fundamentan tanto el algoritmo cuántico de Shor como su variante que factoriza N completamente con una sola pasada, conviene ponerlos en práctica para ver de primera mano como se comportan. Así pues, a lo largo de esta sección se explicará el proceso de implementación, las principales decisiones de diseño y las conclusiones experimentales que se extraen. Más concretamente, el objetivo principal detrás de este apartado no es otro que transformar los algoritmos y circuitos cuánticos descritos en los capítulos anteriores en un programa funcional que sea capaz de factorizar un entero N.

En referencia a los programas realizados, se corresponden a dos *Jupyter notebooks*, uno para la parte cuántica del algoritmo de Shor y otro para la variante de Ekerå. Ambos códigos se encuentran al final de este trabajo en los apéndices A y B, respectivamente. Además se ha realizado el *script* correspondiente para cada uno de los *notebook*.

El código en cuestión también se encuentra disponible para su descarga y correcta visualización en el siguiente repositorio público de *GitHub* : Algoritmo de Shor y variante de Ekerå en el cuál también se ha añadido un README en el que se explica todo el algoritmo de Shor detalladamente así como una aplicación para el caso N=21 y a=2.

5.1. Entorno de programación

En primer lugar, es importante especificar que la parte del código correspondiente a la obra de Ekerå [Eke] ha sido desarrollado mediante el software libre *SageMath*, en su versión 9.2, que es un sistema algebraico computacional escrito en lenguaje *Python* y que se construye sobre múltiples paquetes como *NumPy*, *Maxima* o *R*. Su elección radica en que *SageMath* destaca por tener un amplio abanico de funcionalidades referentes al álgebra abstracta. De hecho, en la actualidad *SageMath* es empleado en multitud de publicaciones matemáticas, tanto en el campo de la criptografía como en muchos otros. Además, su sintaxis sencilla e intuitiva facilita la comprensión y lectura del código, lo que casa perfectamente con el hecho de implementar un código claro y comprensible. En caso de querer

descargarlo o consultar una guía detallada es necesario visitar [Dev21b].

La otra parte ha sido desarrollada en *Python*, en su versión 3.8.3, haciendo uso de la librería *Qiskit* de IBM, en su versión 0.36.1, así como el módulo *qiskit-aer*, en su versión 0.10.4, para poder realizar la construcción, visualización y simulación del circuito cuántico del algoritmo de Shor. *Qiskit* es un kit de desarrollo de software (SDK) creado por IBM para trabajar con computadoras cuánticas a nivel de circuitos, pulsos y algoritmos. Proporciona herramientas para crear y manipular programas cuánticos y ejecutarlos en dispositivos cuánticos como el *IBM Quantum Experience* o en simuladores en local.

5.2. Implementación del Algoritmo de Shor con Qskit

Para la realización del circuito cuántico para factorizar 15 se ha seguido el tutorial facilitado por la propia IBM [Dev21a] (Tutorial Shor):

Primero hemos mostrado el gráfico de la función periódica $f(x) = a^x \pmod N$ para $N = 15$ y $a = 13$. Teniendo en cuenta que las líneas entre los puntos son para ayudar a ver la periodicidad y no representan los valores intermedios entre estos puntos x.

A continuación hemos definido la función *c_amod15(a, power)* que devuelve la puerta U-controlada para a, tantes veces como power. Esta función es la encargada de la exponenciación modular y solo funciona para los valores $a \in [2, 4, 7, 8, 11, 13]$.

Luego hemos implementado la inversa de la transformada de Fourier (que hemos denominado *qtf_inversa*) y hemos creado el circuito cuántico con 8 qubits para el primer registro y 4 qubits auxiliares para la exponenciación modular.

Entonces, hemos realizado la simulación, con el *AerSimulator* y hemos obtenido la distribución de los qubits finales después de la QPE. Para que quedase toda esta parte de la estimación de fase recogida en una misma función hemos definido *qpe_amod15(a)*.

Por último, hemos aplicado las fracciones continuas para encontrar el periodo r y hemos realizado la parte de factorización del algoritmo de Shor, básicamente con el algoritmo de Euclides que ya nos viene implementado en la función *gcd*, así pues hemos encontrado los valores 3 y 5, como esperábamos, en caso de no obtener este resultado automáticamente se vuelve a ejecutar.

Además, también se ha programado el circuito con varias implementaciones alternativas, siguiendo la formulación descrita en el trabajo de [MNM$^+$], que demuestra una realización razonablemente y escalable del algoritmo de Shor para $N = 15$. Esta se basa en optimizaciones significativas gracias al conocimiento a priori de los resultados esperados. A continuación se muestra el siguiente diagrama 5.1, con varios circuitos cuánticos para factorizar $N = 15$.

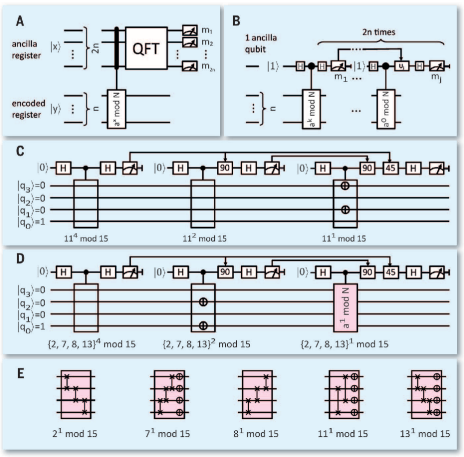


Figura 5.1: Diagrama con diferentes implementaciones de circuitos cuánticos para la factorización de $N = 15$. Usando un enfoque de un libro genérico (**A**) comparado con el enfoque de Kitaev [Kit] (**B**) para una base genérica a. (**C**) La implementación real para factorizar 15 con $a = 11$, optimizado para el estado de entrada única correspondiente. Aquí q_i corresponde al qubit respectivo en el registro computacional. (**D**) Aproximación de Kitaev al algoritmo de Shor para las bases $2, 7, 8, 13$. Aquí, la aplicación optimizada del primer multiplicador es idéntico en los cuatro casos, y el último multiplicador se implementa con multiplicadores modulares completos, como se muestra en (**E**). En todos los casos, el único qubit QFT se usa tres veces, lo que, junto con los cuatro qubits en el registro de cálculo, hace que haya siete qubits efectivos. (**E**) Diagramas de circuito de los multiplicadores modulares de la forma $a \pmod N$ para bases $a = 2, 7, 8, 11, 13$. Fuente: [MNM$^+$]

Hay que tener en cuenta que no podemos ejecutar esta versión del algoritmo de Shor en un dispositivo *IBM Quantum Experience* en este momento, ya que actualmente carecemos de la capacidad de realizar mediciones anticipadas y restablecer qubits. Por lo tanto, solo hemos construido los circuitos para ejecutarlos en los simuladores.

5.3. Algoritmo que factoriza completamente N

A continuación, se detalla brevemente la estrategia llevada a cabo de cara a la creación del código, que es idéntico en el *script* y en el *notebook*. Para ver una explicación más precisa, extensa y línea a línea, se recomienda leer el código comentado que se puede encontrar en el apéndice B o en el repositorio de *GitHub* citado previamente. A modo de ejemplo, en el *notebook* se muestran los resultados de una ejecución completa del código para los casos $N = 7148826542566201483125740477929151540249503556645029 13 \times 4047792915154024950355664502913$ y $N = 714882654256620148312574047792915154024950355664$.

En cuanto a la programación del algoritmo 2 se han seguido los siguientes pasos:

- Primero se ha creado la clase auxiliar *ColeccionFactores* para guardar los factores no triviales de N en forma reducida, que contiene dos listas, una para los factores

de N y otra para los factores primos encontrados. Esta clase contiene las funciones *constructor*, *add*, que añade los factores de N en la lista correspondiente dependiendo de si es primo o no, *esta_completa*, que nos dice si todos los factores primos han sido encontrados, y *imprimir_estados*, que nos muestra la información que contiene cada una de las listas.

- Por último, se ha creado la función *factorizar_completamente(N, c = 1, k = None)*, que aplica el algoritmo 2 punto por punto. Se ha decidido implementar la versión básica, es decir, sin ninguna de las optimizaciones comentadas. La función factorizar_completamente tiene como entrada los parámetros N (entero a factorizar), la constante c, que es más grande o igual que 1, y el número de iteraciones k, que es opcional, y nos devuelve el conjunto de todos los factores primos distintos que dividen a N. Lo primero que hacemos es comprobar que los parámetros entrados sean correctos y a continuación se definen las siguiente funciones auxiliares:

  - *orden(N)* que dado N, calcula un g aleatoriamente de $Z_N^\star$, con el método *IntegerModRing(N).random_element()* y nos da su orden r, utilizando el método *multiplicative_order()*. Ambas funciones están ya implementadas en *SageMath*.
  - *productorio_potencias_primas(B)* que nos sirve para construir el producto de q^e, para q primos $\leq$ B y e el exponente mas grande tal que $q^e \leq B$ para una cota B.
  - *exponente_t(x)* que calcula el exponente t máximo tal que $x = 2^t o$ para un o impar, es decir, nos calcula la multiplicidad del factor 2 del entero x.

  Ahora aplicamos el algoritmo 2 en si:

  - Paso 1: r = *orden(N)*
  - Paso 2: Construimos el producto de factores primos $q^e < cm$ y lo multiplicamos por r, donde m = *N.nbits()* es el número de bits de N y c la constante pasada por parámetro. Por lo tanto para calcular r' hacemos rp = *productorio_potencias_primas(c * m) * r*
  - Paso 3: calculamos t llamando a la función t = *exponente_t(rp)* y posteriormente dividimos esta r' (rp en el programa) por 2^t para obtener o.
  - Paso 4: Finalmente aplicamos el paso de iterar para $j = 0, 1, \ldots, k$ y para $i = 0, 1, \ldots, t$ e ir añadiendo, con *F.add(d)*, los factores $d_{i,j}$ de N a la colección F, que hemos instanciado como F = *ColeccionFactores(N)*, si cumplen la condición descrita de $d_{i,j} = mcd(x_j^{2^i o} - 1, N)$ si $1 < d_{i,j} < N$ para un x_j seleccionado uniformemente al azar de $\mathbb{Z}_N^\star$. En caso de que se haya especificado un límite de iteraciones k, se para el algoritmo en cuanto se llega a esta iteración k y se devuelve una excepción.

  Finalmente, como ya hemos comentado, se devuelven los factores primos de N con la llamada a *F.primos_encontrados*.

Capítulo 6

Conclusiones

A partir de lo visto a lo largo del trabajo se puede concluir que la variante del algoritmo cuántico de Shor que se describe en el trabajo de M. Ekerå [Eke] es una alternativa viable y eficiente al hecho de haber de ejecutar varias veces la parte cuántica de la estimación de fase y puede convertirse en una herramienta útil para romper la criptografía RSA.

Entrando en detalles, el estudio del algoritmo de Shor nos ha permitido presentar y desarrollar algunos de los principales resultados de la computación cuántica. Desde la definición de qubit y de las diferentes puertas cuánticas hasta poder definir un circuito que nos permita realizar la exponenciación modular o aplicar la transformada de Fourier. Todos estos conceptos han sido utilizados en la estimación cuántica de fase para obtener el período que nos permita factorizar un entero N, siendo esta la esencia del algoritmo de Shor. Todos estos conocimientos, más el aporte del trabajo de Miller[Mila], han sido esenciales para entender en profundidad el funcionamiento del algoritmo y sus posibles optimizaciones.

Enlazándolo con esto, el análisis del algoritmo de Shor ha sido dividido en dos vertientes muy diferenciadas, la teórica y la práctica. En cuanto a la parte teórica, se han enunciado y demostrado varios lemas y teoremas, así como las definiciones necesarias, que nos han permitido su completa comprensión. A nivel práctico, se ha implementado una versión del algoritmo de Shor usando la libreria *Qskit* de IBM para poder simular el circuito cuántico que nos ha permitido factorizar el número 15. Además, se ha implementado el algoritmo descrito en el trabajo de Ekerå usando SageMath para poder encontrar todos los factores primos de un número compuesto N.

Para finalizar, se proponen dos líneas de investigación que dan continuidad a este trabajo. Por un lado, existe la posibilidad de mejorar la eficiencia del algoritmo de Ekerå con diferentes optimizaciones, muchas de ellas presentes en su propio artículo [Eke]. Finalmente, la última propuesta se corresponde con el análisis en profundidad de otros estudios relacionados con el cuello de botella que representa la exponenciación modular y el hecho de que se disponga de más qubits reales.

Bibliografía

[Bel] J. S. Bell, *On the einstein podolsky rosen paradox*, no. 3, 195–200.

[Blo] F. Bloch, *Nuclear induction*, no. 7, 460–474.

[Dev21a] IBM Developers, *Shor's algorithm tutorial*, `https://community.qiskit.org/textbook/ch-algorithms/shor.html`, 2021.

[Dev21b] The Sage Developers, *Sagemath, the sage mathematics software system (version 9.2)*, `https://www.sagemath.org`, 2021.

[Dir] P. A. M. Dirac, *A new notation for quantum mechanics*, no. 3, 416–418, Edition: 2008/10/24 Publisher: Cambridge University Press.

[Eke] Martin Ekerå, *On completely factoring any integer efficiently in a single run of an order finding algorithm*, type: article.

[EPR] A. Einstein, B. Podolsky, and N. Rosen, *Can quantum-mechanical description of physical reality be considered complete?*, no. 10, 777–780.

[Fey82] Richard P. Feynman, *Simulating physics with computers*, no. 6, 467–488.

[GS] Walther Gerlach and Otto Stern, *Der experimentelle nachweis der richtungsquantelung im magnetfeld*, no. 1, 349–352.

[JL] Richard Jozsa and Noah Linden, *On the role of entanglement in quantum-computational speed-up*, no. 2036, 2011–2032.

[Kit] A. Yu Kitaev, *Quantum measurements and the abelian stabilizer problem*, Number: arXiv:quant-ph/9511026.

[LBC$^+$] Erik Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O'Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and John M. Martinis, *Computing prime factors with a josephson phase qubit quantum processor*, no. 10, 719–723.

[LBYP] Chao-Yang Lu, Daniel E. Browne, Tao Yang, and Jian-Wei Pan, *Demonstration of a compiled version of shor's quantum factoring algorithm using photonic qubits*, no. 25, 250504.

[Lon81] Douglas L Long, *Random equivalence of factorization and computation of orders.*

[LWL+] B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. V. James, A. Gilchrist, and A. G. White, *Experimental demonstration of a compiled version of shor's algorithm with quantum entanglement*, no. 25, 250505.

[ME] Michele Mosca and Artur Ekert, *The hidden subgroup problem and eigenvalue estimation on a quantum computer*, Quantum Computing and Quantum Communications (Colin P. Williams, ed.), vol. 1509, Springer Berlin Heidelberg, Series Title: Lecture Notes in Computer Science, pp. 174–188.

[Mer] N. David Mermin, *Quantum computer science: an introduction*, Cambridge University Press, OCLC: ocn137221653.

[Mila] G. L. Miller, *Riemann's hypothesis and tests for primality*.

[Milb] Gary L. Miller, *Riemann's hypothesis and tests for primality*, no. 3, 300–317.

[MLLL+] E. Martín-López, A. Laing, T. Lawson, R. Alvarez, X. . Q. Zhou, and J. L. O'Brien, *Experimental realization of shor's quantum factoring algorithm using qubit recycling*.

[MNM+] Monz Thomas, Nigg Daniel, Martinez Esteban A., Brandl Matthias F., Schindler Philipp, Rines Richard, Wang Shannon X., Chuang Isaac L., and Blatt Rainer, *Realization of a scalable shor algorithm*, no. 6277, 1068–1070, Publisher: American Association for the Advancement of Science.

[NC] Michael A. Nielsen and Isaac L. Chuang, *Quantum computation and quantum information*, 10th anniversary ed ed., Cambridge University Press.

[OCT] J. Ossorio-Castillo and José M. Tornero, *Quantum computing from a mathematical perspective: a description of the quantum circuit model*, type: article.

[Pol] J. M. Pollard, *Theorems of factorization and primality testing*.

[Rab] M. O. Rabin, *Probabilistic algorithm for testing primality*.

[RSA] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*.

[Sch] E. Schrödinger, *Discussion of probability relations between separated systems*, no. 4, 555–563, Edition: 2008/10/24 Publisher: Cambridge University Press.

[Shoa] Peter W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, no. 5, 1484–1509, _eprint: https://doi.org/10.1137/S0097539795293172.

[Shob] P.W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Proceedings 35th Annual Symposium on Foundations of Computer Science, IEEE Comput. Soc. Press, pp. 124–134.

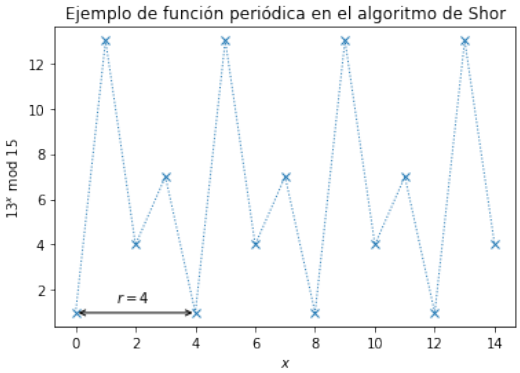
[VSB+] Lieven M. K. Vandersypen, Matthias Steffen, Gregory Breyta, Costantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang, *Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance*, no. 6866, 883–887.

Apéndice A

Código algoritmo cuántico de Shor N=15

```
N = 15
a = 13

# Calculamos las x e yque vamos a dibujar
xvals = [i for i in range(N)]
yvals = [np.mod(a**x, N) for x in xvals]

# Usamos matplotlib para mostrarlo graficamente
fig, ax = plt.subplots()
ax.plot(xvals, yvals, linewidth=1, linestyle='dotted', marker='x')
ax.set(xlabel='$x$', ylabel='$%i^x$ mod $%i$' % (a, N),
       title="Ejemplo de funcion periodica en el algoritmo de Shor")
try: # Dibujamos el periodo r en el grafico
    r = yvals[1:].index(1) +1
    plt.annotate('', xy=(0,1), xytext=(r,1), arrowprops=dict(arrowstyle='<->'))
    plt.annotate('$r=%i$' % r, xy=(r/3,1.5))
except ValueError:
    print('No se pudo encontrar el periodo, comprobar que a < N y que no tenga factores comunes con N.')
```

Output:



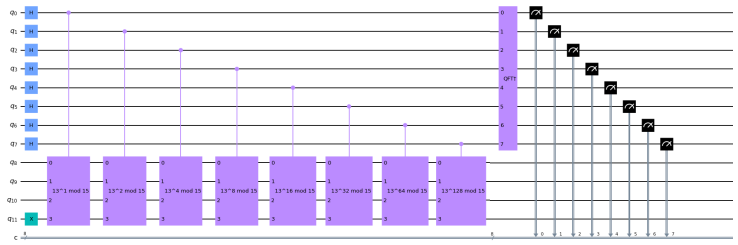
```
#La funcion c_amod15 devuelve la puerta U-controlada para a, tantes veces como 'power'.
#Esta funcion es especifica pora la operacion mod15 y asi esta implementada en el ordenador cuantico de IBM
#Solo funciona para los valores a = [2,4,7,8,11,13]
def c_amod15(a, power):
    """Multiplicacion-controlada para a mod 15"""
    if a not in [2,4,7,8,11,13]:
        raise ValueError("'a' debe ser 2,4,7,8,11 o 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
```

```
#Funcion que crea el circuito de la inversa de la transformada cuantica de Fourier
def qft_inversa(n):
    """n-qubit QFT-inversa para los primeros n qubits del circuito"""
    qc = QuantumCircuit(n)
    # Hay que realizar los 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 = "†QFT"
    return qc
```

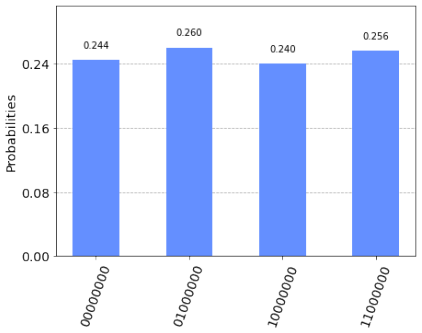
```
# Creamos el circuito cuantico (QuantumCircuit) con n_count qubits para el primer registro |x>
# mas 4 qubits para el segundo registro |w> que actuar en U
n_count = 8  # numero de qubits necesarios para factorizar 15

qc = QuantumCircuit(n_count + 4, n_count)

# Inicializamos los n_count qubits al estado |+> aplicando la  puerta de Hadamard
for q in range(n_count):
    qc.h(q) #Aplicamos la puerta de Hadamard a cada qubit

# Definimos un registro auxiliar en el estado |1>
qc.x(3+n_count)

# Aplicamos la operacion de U-controlada (que realizara la exponenciacion modular)
for q in range(n_count):
    qc.append(c_amod15(a, 2**q),
              [q] + [i+n_count for i in range(4)])

# Aplicamos la inversa-QFT a los 8 qubits del primer registro |x>
qc.append(qft_inversa(n_count), range(n_count))

# Medimos los n_count qubits
qc.measure(range(n_count), range(n_count))
qc.draw(mtl) #Dibujamos el circuito
```

Output:



```
#Realizamos la simulacion del circuito anterior
aer_sim = Aer.get_backend('aer_simulator') #Usamos el AerSimulator
t_qc = transpile(qc, aer_sim) #Pasamos el ciruito al simulador
qobj = assemble(t_qc) #Lo compilamos
results = aer_sim.run(qobj).result() #Hacemos la simulacion y guardamos los resultados
counts = results.get_counts() #Obtenemos la distribucion de los qubits finales despues del QPE
plot_histogram(counts) #Hacemos la grafica con las probabilidades
```

Output:




```
filas, fases_medidas = [], []
for output in counts:
    decimal = int(output, 2)  # Convertimos (base 2) a decimal
    fase = decimal/(2**n_count)  # Encontramos el valor propio
    fases_medidas.append(fase)
    frac = Fraction(fase).limit_denominator(N)
    # Agragamos los valores por filas en la tabla
    filas.append([f"{output}(bin) = {decimal:>3}(dec)",
                  f"{decimal}/{2**n_count} = {fase:.2f}", f"{frac.numerator}/{frac.denominator}", frac.denominator])
# Imprimimos la tabla
headers=["Registro de salida", "Fase", "Fraccion", "Posible r"]
df = pd.DataFrame(filas, columns=headers)
print(df)
```

Output:

```
         Registro de salida              Fase Fraccion  Posible r
0  11000000(bin) = 192(dec)  192/256 = 0.75      3/4          4
1  10000000(bin) = 128(dec)  128/256 = 0.50      1/2          2
2  00000000(bin) =   0(dec)    0/256 = 0.00      0/1          1
3  01000000(bin) =  64(dec)   64/256 = 0.25      1/4          4
```

```
#Circuito que recopila lo visto anteriorme para hacer la estimacion de fase
#Le pasamos el parametro de entrada a que es el numero del cual queremos saber su orden mod 15, devuelve la fase
def qpe_amod15(a):
    n_count = 8 #8 bits para el registro de entrada |x>
    qc = QuantumCircuit(4+n_count, n_count) #Creamos el circuito cuantico con |x> de 8 qubits y |w> de 4 qubits auxiliar
    for q in range(n_count):
        qc.h(q)       # Inicializamos los 8 qubits de |x> al estado |+> con puertas de Hadamard
    qc.x(3+n_count) # Ponemos el registro auxiliar en el estado |1>
    for q in range(n_count): # Aplicamos las puertas U-controladas de la exponenciacion modular con modulo 15
        qc.append(c_amod15(a, 2**q),
                  [q] + [i+n_count for i in range(4)])
    qc.append(qft_inversa(n_count), range(n_count)) # Aplicamos la QFT inversa
    qc.measure(range(n_count), range(n_count)) #Medimos los qubits finales del registro |x>
    # Simulamos los resultados obtenidos
    aer_sim = Aer.get_backend('aer_simulator')
    t_qc = transpile(qc, aer_sim)
    qobj = assemble(t_qc, shots=1)
    # Poniendo memory=True nos permite ver una lista secuencial de cada resultado
    result = aer_sim.run(qobj, memory=True).result()
    lecturas = result.get_memory()
    print("Lectura del registro: " + lecturas[0])
    fase = int(lecturas[0],2)/(2**n_count) #Calculamos una de las posibles fases
    print("Fase correspondiente: %f" % fase)
    return fase
```

```
#Algoritmo de Shor para recuperar los factores de N=15. Si no los encontramos volvemos a realizar la ejecucion
factor_encontrado = False
intento = 0
while not factor_encontrado:
    intento += 1
    print("\nIntento %i:" % intento)
    fase = qpe_amod15(a) # Fase = s/r
    #El denominador despues de hacer las fracciones continuas nos permite encontrar r, puede ser r o un divisor, por lo tanto no siempre funciona
    frac = Fraction(fase).limit_denominator(N)
    r = frac.denominator
    print("Resultado: r = %i" % r)
    if fase != 0:
        # Los posibles factores son gcd(x^{r/2} ±1 , 15)
        posibles_factores = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
        print("Factores posibles: %i and %i" % (posibles_factores[0], posibles_factores[1]))
        for factor in posibles_factores:
            if factor not in [1,N] and (N % factor) == 0: # Comprobamos si el posible factor es realmente un factor primo de N no trivial
                print("*** Factor no trivial encontrado: %i ***" % factor)
                factor_encontrado = True
```

Output:

```
Intento 1:
```

```
Lectura del registro: 10000000
Fase correspondiente: 0.500000
Resultado: r = 2
Factores posibles: 3 and 1
*** Factor no trivial encontrado: 3 ***
```

```
#Alternativa de implementracion de QPE
#qc = circuito cuantico, qr = registro cuantico, cr = registro clasico, a = 2, 7, 8, 11 or 13
def circuit_amod15(qc,qr,cr,a):
    if a == 2:
        qc.cswap(qr[4],qr[3],qr[2])
        qc.cswap(qr[4],qr[2],qr[1])
        qc.cswap(qr[4],qr[1],qr[0])
    elif a == 7:
        qc.cswap(qr[4],qr[1],qr[0])
        qc.cswap(qr[4],qr[2],qr[1])
        qc.cswap(qr[4],qr[3],qr[2])
        qc.cx(qr[4],qr[3])
        qc.cx(qr[4],qr[2])
        qc.cx(qr[4],qr[1])
        qc.cx(qr[4],qr[0])
    elif a == 8:
        qc.cswap(qr[4],qr[1],qr[0])
        qc.cswap(qr[4],qr[2],qr[1])
        qc.cswap(qr[4],qr[3],qr[2])
    elif a == 11:
        qc.cswap(qr[4],qr[2],qr[0])
        qc.cswap(qr[4],qr[3],qr[1])
        qc.cx(qr[4],qr[3])
        qc.cx(qr[4],qr[2])
        qc.cx(qr[4],qr[1])
        qc.cx(qr[4],qr[0])
    elif a == 13:
        qc.cswap(qr[4],qr[3],qr[2])
        qc.cswap(qr[4],qr[2],qr[1])
        qc.cswap(qr[4],qr[1],qr[0])
        qc.cx(qr[4],qr[3])
        qc.cx(qr[4],qr[2])
        qc.cx(qr[4],qr[1])
        qc.cx(qr[4],qr[0])
```

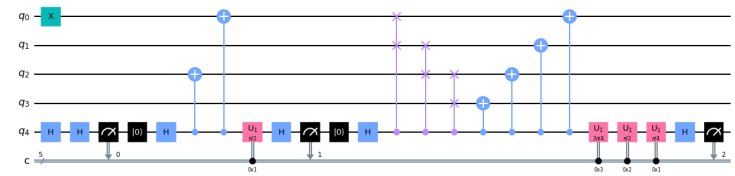
```
def circuit_aperiod15(qc,qr,cr,a):
    if a == 11:
        circuit_11period15(qc,qr,cr)
        return

    # Inicializamos q[0] a |1>
    qc.x(qr[0])

    # Aplicamos a**4 mod 15
    qc.h(qr[4])
    #   identidad controlada en los otros 4 qubits, que es lo mismo que no hacer nada
    qc.h(qr[4])
    #   medimos
    qc.measure(qr[4],cr[0])
    #   reinicializamos q[4] a |0>
    qc.reset(qr[4])

    # Aplicamos a**2 mod 15
    qc.h(qr[4])
    #   Unitaria controlada
    qc.cx(qr[4],qr[2])
    qc.cx(qr[4],qr[0])
    #   avanzamos
    qc.u1(math.pi/2.,qr[4]).c_if(cr, 1)
    qc.h(qr[4])
    #   medimos
    qc.measure(qr[4],cr[1])
    #   reinicializamos q[4] a |0>
    qc.reset(qr[4])

    # Aplicamos a mod 15
    qc.h(qr[4])
    #   Unitaria controlada
    circuit_amod15(qc,qr,cr,a)
    #   avanzamos
    qc.u1(3.*math.pi/4.,qr[4]).c_if(cr, 3)
    qc.u1(math.pi/2.,qr[4]).c_if(cr, 2)
    qc.u1(math.pi/4.,qr[4]).c_if(cr, 1)
    qc.h(qr[4])
    #   medimos
    qc.measure(qr[4],cr[2])
```

```
def circuit_11period15(qc,qr,cr):
    # Aplicamos a**4 mod 15
    qc.x(qr[0])

    # Aplicamos a**4 mod 15
    qc.h(qr[4])
```

```
    #  identidad controlada en los otros 4 qubits, que es lo mismo que no hacer nada
    qc.h(qr[4])
    #   medimos
    qc.measure(qr[4],cr[0])
    #    reinicializamos q[4] a |0>
    qc.reset(qr[4])

    # Apply a**2 mod 15
    qc.h(qr[4])
    #   identidad controlada en los otros 4 qubits, que es lo mismo que no hacer nada
    #   avanzamos
    qc.u1(math.pi/2.,qr[4]).c_if(cr, 1)
    qc.h(qr[4])
    #   medimos
    qc.measure(qr[4],cr[1])
    #    reinicializamos q[4] a |0>
    qc.reset(qr[4])

    # Aplicamos 11 mod 15
    qc.h(qr[4])
    #   controlled unitary.
    qc.cx(qr[4],qr[3])
    qc.cx(qr[4],qr[1])
    #   avanzamos
    qc.u1(3.*math.pi/4.,qr[4]).c_if(cr, 3)
    qc.u1(math.pi/2.,qr[4]).c_if(cr, 2)
    qc.u1(math.pi/4.,qr[4]).c_if(cr, 1)
    qc.h(qr[4])
    #   medimos
    qc.measure(qr[4],cr[2])
```

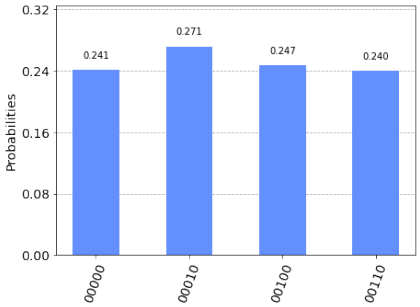
```
#caso con a=7
q = QuantumRegister(5, 'q')
c = ClassicalRegister(5, 'c')

shor = QuantumCircuit(q, c)
circuit_aperiod15(shor,q,c,7)
shor.draw(output='mpl')
```

Output:



```
backend = BasicAer.get_backend('qasm_simulator')
sim_job = execute([shor], backend)
sim_result = sim_job.result()
sim_data = sim_result.get_counts(shor)
plot_histogram(sim_data)
#vemos que todos los estados son equiprobables con probabilidad 0.25
```

Output:



```
#n_counts =4
n_counts =4
filas, fases_medidas = [], []
for output in sim_data:
    decimal = int(output, 2)  # Convertimos (base 2) a decimal
    fase = decimal/(2**3)  # Encontramos el valor propio usamos solo 4 qubits de entrada n_counts =4
    fases_medidas.append(fase)
    frac = Fraction(fase).limit_denominator(N)
    # Agragamos los valores por filas en la tabla
```

```
    filas.append([f"{output}(bin) = {decimal:>3}(dec)",
                  f"{decimal}/{2**n_counts} = {fase:.2f}", f"{frac.numerator}/{frac.denominator}", frac.denominator])
# Imprimimos la tabla
headers=["Registro de salida", "Fase", "Fraccion", "Posible r"]
df = pd.DataFrame(filas, columns=headers)
print(df)

Output:
      Registro de salida          Fase Fraccion  Posible r
0  00000(bin) =    0(dec)  0/16 = 0.00      0/1          1
1  00100(bin) =    4(dec)  4/16 = 0.50      1/2          2
2  00010(bin) =    2(dec)  2/16 = 0.25      1/4          4
3  00110(bin) =    6(dec)  6/16 = 0.75      3/4          4
-------------------------------------------------------------------------------------------
```

Apéndice B

Código variante de Shor de Ekerå

```
# Clase auxiliar para guardar los factores no triviales de N en forma reducida.
class ColeccionFactores:
    def __init__(self, N):
        # El entero N a factorizar
        self.N = N

        # El conjunto de factores encontrado hasta ahora, reducido para que todos los factores del conjunto
        # sean coprimos por pares entre si. Esta propiedad se aplica mediante add().
        self.factores_encontrados = set()

        # El conjunto de factores primos encontrados hasta ahora; un subconjunto de factores_encontrados
        self.primos_encontrados = set()

        # El residuo; el producto de los factores coprimos por pares compuestos en el
        # coleccion, o uno si no hay factores compuestos en la coleccion.
        self.residuo = 1

        # Agrega N como factor.
        self.add(N)

    # Comprueba si todos los factores primos se han encontrado.
    def esta_completa(self):
        return self.residuo == 1

    # Agrega un factor a la coleccion.
    def add(self, d):
        # Verificamos que el factor no sea trivial y que aun no se haya encontrado.
        if (d == 1) or (d in self.factores_encontrados):
            return

        # Comprobamos si d comparte un factor con cualquiera de los factores encontrados.
        D = 1

        for f in self.factores_encontrados:
            D = gcd(f, d)

            if D != 1:
                break

        if D != 1:
            # Si es asi, eliminamos f, dividimos f y d, y agregamos los factores resultantes.
            self.factores_encontrados.remove(f)
            if f not in self.primos_encontrados:
                # Tambien eliminamos f del residuo cuando eliminamos f de la coleccion.
                self.residuo /= f

            f /= D
            d /= D

            self.add(D);

            if f != 1:
                self.add(f)

            if d != 1:
                self.add(d);
        else:
            # Comprobamos si d es una potencia perfecta y, de ser asi, reducimos d.
            (d, _) = ZZ(d).perfect_power();
```

```
            # Agregamos d a los factores encontrados
            self.factores_encontrados.add(d);

            # Comprobamos si d es primo, y si es asi lo agregamos
            resultado = d.is_prime(proof = False);

            if resultado:
                self.primos_encontrados.add(d);
            else:
                # Si d no es primo, multiplicamos d por el residuo.
                self.residuo *= d;

    # Imprimimos la informacion de los conjuntos
    def imprimir_estado(self):
        print("Factores encontrados:", len(self.factores_encontrados));
        print("Pirmos encontrados:", len(self.primos_encontrados));

        factores_encontrados = list(self.factores_encontrados);
        factores_encontrados.sort();

        for i in range(len(factores_encontrados)):
            print(" Factor " + str(i) + ":", factores_encontrados[i]);
        print("");


# ------------------------------------------------------------------------
# Funcion que factoriza completamente N (nos da todos sus factores primos sin multiplicidad)
#
# El parametro c esta definido en el trabajo, es una constante mas grande o igual que 1. El parametro k
# no necesita especificarse explicitamente: por defecto, tantas iteraciones k como sean necesarias para
# factorizar completamente el entero N.
#
# Si se quiere, se puede especificar k. Si el numero de
# iteraciones realizadas excede k, entonces se detiene el algoritmo
#
# Esta funcion devuelve el conjunto de todos los factores primos distintos que dividen a N.

def factorizar_completamente(N, c = 1, k = None):

    # Comprobaciones de los parametros
    if (N < 2) or (c < 1):
        raise Exception("Error: Parameteros incorrectos.")

    # Funcion de soporte para construir el producto de q^e, para q primos <= B y
    # e el exponente mas grande tal que q^e <= B para una cota B.
    def productorio_potencias_primas(B):
        factor = 1
        for q in prime_range(B + 1):
            e = 1
            while q^(e + 1) <= B:
                e += 1
            factor *= q^e

        return factor

    # Funcion de soporte para calcular t tal que x = 2^t * o para o impar.
    def exponente_t(x):
        if x == 0:
            return 0

        t = 0
        while (x % 2) == 0:
            t += 1
            x /= 2

        return t

    #Funcion auxiliar que dado N, calcula un g aleatoriamente de Z_N^* y nos da su orden
    def orden(N):
        while True:
            g = IntegerModRing(N).random_element() #Seleccionamos g aleatoriamente de Z_N^*
            if (g == 1):
                continue
            if gcd(g.lift(), N) == 1:
                break
        return g.multiplicative_order() #Devolvemos el orden de g (Esto es lo que nos daria Shor)

    # Paso 1: Seleccionamos g aleatoriamente de Z_N^* y calculamos su orden
    r = orden(N)

    # Paso 2: Construimos el producto de factores primos q^e < cm y lo multiplicamos por r.
    # Obtenemos r' que denominamos rp

    m = N.nbits() # m es el numero de bits de N
    rp = productorio_potencias_primas(c * m) * r #Calculamos r'

    # Paso 3: Sea rp = 2^t o para o impar.
    t = exponente_t(rp) #calculamos t
```

```
o = rp / 2^t #obtenemos o

# Definimos el conjunto de coprimos a pares y agregamos N.
F = ColeccionFactores(N);

# Paso 4: Para j = 1, 2, ... hasta k donde k puede estar o no acotada.
j = 0;
while True:
    # Imprimimos el estado actual para cada iteracion
    print("Iteracion:", j);
    F.imprimir_estado();

    # Comprobamos si ya hemos acabado
    if F.esta_completa():
        break;

    # Incrementamos j para la siguiente iteracion.
    j += 1;

    # Comprobamos que j > k, si k se ha especificado, y en ese caso
    # devolvemos una excepcion.
    if (k != None) and (j > k):
        raise Exception("Error: Se ha superado el limite de iteraciones.");

    # Paso 4.1: Seleccionamos x uniformemente al azar de Z_N^*.
    x=0
    while x == 0:
        x = IntegerModRing(N).random_element();
    x = IntegerModRing(N)(x) #Para aplicar exponenciacion/aritmetica modular

    #4.2 Para cada i = 0, ..., t hacemos:
    for i in range(0, t + 1):
        xj = x^((2^i) * o);
        # Paso 4.2.1 para i = 0, 1, ..., t:
        d = gcd((xj - 1).lift(), N);
        if 1 < d < N:
            F.add(d);

return F.primos_encontrados;
```