Trabajo Fin de Grado

Computación Cuántica: el Algoritmo de Shor.

Santiago Iglesias González

29 de Julio, 2022

UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

GRADO DE MATEMÁTICAS

Trabajo Fin de Grado

Computación Cuántica: el Algoritmo de Shor.

Santiago Iglesias González

Julio, 2022

UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Conocimiento: Álgebra
Título: Computación Cuántica: el Algoritmo de Shor.
Breve descripción del contenido
En este trabajo se hará una introducción a la computación cuántica explotando la analogía entre las funciones booleanas y las transformaciones unitarias en un espacio de Hilbert. Como aplicación se dará una descripción del algoritmo de factorización en primos de número enteros desarrollado por Shor.
Recomendaciones
Haber superado las materias de Área de Álgebra del Grado en Matemáticas. Tener conocimientos de los algoritmos clásicos y de su formalización en términos de funciones booleanas.
Otras observaciones

Índice

Resumen

En este trabajo se presentan los conceptos preliminares y la explicación de uno de los algoritmos fundamentales de la computación cuántica: el algoritmo de Shor, que consiste en la factorización de un número entero en sus factores primos. Para ello, recordaremos conceptos básicos de álgebra lineal, así como la representación de números naturales como números binarios. También hablaremos de la importancia que presentan los números primos en el algoritmo de Shor mediante la presentación y explicación del sistema de encriptación RSA. Una vez vistos estos conceptos, explicaremos las particularidades presentes en las funciones cuánticas, así como su representación en forma de circuitos cuánticos, que nos servirán para entender la Transformada Cuántica de Fourier (QFT), una de las funciones principales de nuestro algoritmo. Finalizaremos el documento explicando paso a paso el algoritmo de Shor, ilustrándolo con un ejemplo sencillo y comentando cuestiones relacionadas con la eficiencia del mismo.

Abstract

In this work we are going to show the preliminary concepts and the description of one of the fundamental algorithms in quantum computation: Shor's algorithm, which gives a factorization of an integer number into its prime factors. To do this, we will recall basic linear algebra concepts, as well as the representation of natural numbers as binary ones. We will also discuss the importance of prime numbers in Shor's algorithm by presenting and explaining the encryptation system called RSA. After seeing these concepts, we will explain some particularities of quantum functions, as well as their representation as quantum circuits, which will help us to understand the quantum Fourier transform (QFT), one of the main functions of our algorithm. We will finish the document by explaining Shor's algorithm step by step, illustrating it with a simple example and discussing issues related to the algorithm efficiency.

Introducción

Una de las principales aplicaciones de las matemáticas ha sido la aplicación de la teoría de números a los sistemas encargados de la seguridad informática. Los sistemas de cifrado actuales están basados en operaciones matemáticas. Un ejemplo muy importante es el algoritmo de cifrado RSA. La idea básica detrás del algoritmo es la dificultad para encontrar los dos factores primos en los que podemos factorizar un número entero (que puede ser de muchas cifras). El proceso de encontrar los factores primos de un número entero es una problema de una gran complejidad computacional en ausencia de información adicional sobre los mismos. Otro problema añadido a esta cuestión es que los algoritmos actuales más eficientes presentan un rendimiento de orden de complejidad exponencial, lo que implica un proceso de factorización mucho más lento y costoso a medida que aumenta el tamaño del número que queremos factorizar.

La computación clásica está basada en el empleo de operadores lógicos que se entremezclan para crear funciones lógicas más complejas. Dichos operadores pueden ser implementados mediante la electrónica de los semiconductores, formando la base de la computación digital actual. Además, cabe recordar que esta tecnología es la responsable de la presente revolución en los campos de la computación y las comunicaciones.

Recordemos también que la informática está basada en el empleo de funciones binarias, cuyos argumentos son combinaciones de ceros y unos. De esta manera, podemos aplicar operadores lineales sobre estos elementos (que no son otra cosa que la expresión equivalente de los operadores lógicos) de modo que el álgebra lineal aplicada sobre cuerpos finitos nos aporta un modelo útil y con una fuerte base matemática para la construcción de algoritmos.

En esta línea de pensamiento, *Paul Benioff* propuso en 1980 un modelo cuántico de la máquina de Turing propuesta 56 años antes por el matemático y precursor de la informática moderna, *Alan Turing*. Análogamente a la expresión matricial de operaciones binarias, podemos formalizar también las principales operaciones lógicas en las que se basa la computación cuántica mediante el empleo de matrices unitarias complejas.

En este contexto, el matemático estadounidense *Peter Shor* propuso en el año 1994 un algoritmo (que lleva su propio apellido) que utilizando técnicas características de la computación cuántica consigue reducir a un orden de complejidad polinómico el proceso de factorización de números enteros comentado anteriormente que en la actualidad presenta un orden de complejidad exponencial. Por esta y otras contribuciones, *Peter Shor* recibió en el congreso internacional de Matemáticas celebrado en Berlín en el año 1998 el premio *Nevanlinna* otorgado por la IMU (Unión Matemática Internacional).
El algoritmo de Shor se basa en un uso de la Transformada cuántica de Fourier, operador que podemos describir en términos de funciones cuánticas básicas. De esta manera, el objetivo de este trabajo no es otro que comentar las principales cuestiones realcionadas con la descripción y aplicación práctica de este reconocido algoritmo.
Para ello, hemos organizado el contenido del trabajo en una serie de capítulos cuyo contenido procederemos a describir a continuación:

1. **Capítulo 1: Cuestiones preliminares**. En este capítulo hablaremos de como *representar números naturales como cadenas binarias* (y viceversa) y las operaciones que podemos realizar con ellas, además de introducir también los conceptos relacionados con *el orden de complejidad de algoritmos*.

2. **Capítulo 2: Preliminares de álgebra lineal**. Expondremos en este capítulo las *definiciones y resultados de álgebra lineal* que conforman la base para la parte clásica del algoritmo de Shor, *fijando notaciones* e incluyendo también conceptos de *espacios complejos*, importantes también en la computación cuántica.

3. **Capítulo 3: Sistema RSA: claves pública y privada**. Comentaremos aquí las cuestiones relacionadas con el *sistema de encriptación RSA*. Para ello, introduciremos definiciones de gran importancia para nuestro algoritmo como las *congruencias*. Expondremos también el *enunciado y demostración* del resultado 3.4 de gran importancia en el capítulo, así como un *ejemplo de aplicación* de dicho algoritmo para afianzar los conceptos presentados previamente.

4. **Capítulo 4: Funciones binarias, bits cuánticos y factibilidad**. En este capítulo presentaremos los principales conceptos relacionados con las *funciones binarias* y *las funciones cuánticas*, así como el concepto de *factibilidad* en cada uno de estos dos grupos de funciones. Por último, nos centraremos en la *representación* de estas funciones como *circuito de puertas cuánticas* y definiremos los dos principales tipos de matrices cuánticas que utilizaremos en el algoritmo: *las matrices de Hadamard* (en la sección 4.4.2) y las *matrices* $\boldsymbol{U}_1(\alpha)$ (en la sección 4.4.3).

5. **Capítulo 5: Transformada Cuántica de Fourier (QFT)**. Definiremos y hablaremos de las propiedades y la *representación como circuito cuántico* (con su correspondiente demostración) de una de las funciones más importantes y versátiles de la computación cuántica: *la Transformada Cuántica de Fourier* (QFT). Para terminar con los contenidos de este capítulo presentaremos la aplicación fundamental de la QFT en el algoritmo de Shor: *determinar el periodo de una función* f. Para ello, incluiremos la *definición* de *función periódica* y a continuación enunciaremos y demostraremos el Lema 5.6, fundamental en dicho proceso.

6. **Capítulo 6: El algoritmo de Shor**. Comentaremos finalmente los pasos en los que se divide tanto *la parte clásica* como *la parte cuántica* del algoritmo de Shor. Para ello, definiremos los conceptos claves siguientes: *la raíz cuadrada no trivial módulo (N)* y *el orden r de un entero m módulo (N)*; además de enunciar y demostrar los dos resultados en los que se sustenta toda la teoría del algoritmo: el Lema 6.5 y el Lema 6.7. Finalizaremos la primera sección de este último capítulo presentando un *ejemplo de aplicación del algoritmo* para afianzar los conceptos previamente comentados. Para terminar con el capítulo, comentaremos las cuestiones relacionadas con la principal aplicación real del algoritmo: la *criptografía*; y también analizaremos las razones que explican su *mejor desempeño en cuanto a orden de complejidad en la ejecución* respecto a cualquier algoritmo clásico.

Capítulo 1

Cuestiones preliminares

1.1. Números naturales y cadenas binarias

En este capítulo, hablaremos de los números naturales y su representación como cadenas binarias de unos y ceros, fundamentales en la computación debido al establecimiento de roles contrarios en la informática: encendido/apagado o cargado/descargado, donde cada uno de estos estados es representado por un bit diferente (0 para el apagado y 1 para el encendido).
Recordemos que una **cadena binaria** es un conjunto de bits cuyos elementos solo pueden tomar dos valores: 0 o 1. Con estas cadenas podemos realizar operaciones simples: el tamaño de una cadena es el número de bits presentes en la misma, y si **x** e **y** son cadenas binarias, entonces **xy** es la concatenación de ambas cadenas.

Ejemplo 1.1. Si **x=01** e **y=11101**, entonces el resultado de concatenar ambas cadenas es el siguiente: **xy=0111101**.

Esta operación de concatenación la podemos entender como una operación interna para las cadenas binarias, pero en este caso no utilizaremos un símbolo explícitamente. Por otro lado, hablaremos del **producto interior binario**: si x e y son cadenas de longitud m; entonces $x \bullet y$ representa dicho producto y está definido de la siguiente manera:

$$x \bullet y = x_1 y_1 \oplus \cdots \oplus x_m y_m$$

En este caso, el símbolo $\oplus$ denota el OR exclusivo (únicamente es verdadero cuando ambos valores difieren; es decir, cuando uno es 0 y el otro es 1 o viceversa) [9].
Una vez comentadas estas primeras cuestiones, lo siguiente que debemos conocer es cómo pasar de un número natural a una cadena binaria y viceversa, ya que en ciertos casos será más útil utilizar una forma que otra. Esta correspondencia es muy importante en la computación, y más concretamente en los algoritmos cuánticos.

A continuación, mostramos los procesos que nos permiten pasar de una forma a otra:
Un número $m \in \mathbb{N}$ admite una única representación en forma de cadena binaria:

$$m = 2^{n-1}x_{n-1} + \cdots + 2x_1 + x_0,$$

donde cada x_i, con $i \in \{1, \ldots, n-1\}$, es un bit (0 o 1) y en el que $x_{n-1} \neq 0$.

Ejemplo 1.2. El número **9** se representaría en binario como **1001** ($2^3 \cdot 1 + 2^2 \cdot 0 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 0 + 0 + 1 = 9$).

En el otro sentido, podemos emplear la cadena binaria

$$x_{n-1} \ \ldots \ x_1 \, x_0$$

para denotar el número natural

$$2^{n-1}x_{n-1} + \cdots + 2x_1 + x_0.$$

De esta manera, la cadena **101010** representa el número natural $2^5 + 2^3 + 2^1 = 32 + 8 + 2 = \mathbf{42}$.
Surge la necesidad de fijar dichas representaciones de tal manera que sean únicas. Para ello, es necesario conocer la longitud de las cadenas; ya que en otro caso, no sabríamos, por ejemplo, ¿que cadena representaría al número 0: si 0 o 00 o 000 y así en adelante? Para resolver este problema, lo que podemos hacer es especificar la correspondencia unívocamente entre los números naturales y las cadenas de longitud n, que son de la forma $\{0,1\}^n$. Si hacemos esto, vemos que ahora el número natural 0 se corresponde de manera única con la cadena binaria $0 \cdots 0$, que presenta n ceros.
Además, las cadenas binarias se pueden usar para representar los diferentes subconjuntos de un conjunto. Para ilustrarlo, veamos el siguiente ejemplo:

Ejemplo 1.3. Denotemos por $A = \{1, 2, 3\}$ nuestro conjunto. Entonces, 000 se corresponde con el conjunto vacío, 011 se asocia con el subconjunto $B = \{2, 3\}$, 110 con el subconjunto $C = \{1, 2\}$, 111 con el conjunto A y así sucesivamente.

1.2. Notación asintótica

Una cuestión muy importante a tener en cuenta en los algoritmos es el orden de complejidad, que representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia del mismo [5]. A continuación, introduciremos un par de definiciones relacionadas con el orden de complejidad de los algoritmos:

Definición 1.4. Dos funciones $s(n)$ y $t(n)$ definidas en $\mathbb{N}$ son **asintóticamente equivalentes** si $\lim_{n\to\infty} \frac{s(n)}{t(n)} = 1$.

Definición 1.5. Dadas dos funciones $s, t : \mathbb{N} \longrightarrow \mathbb{R}$ tenemos que:

- $s(n) = O(t(n))$ si existen constantes $c, d \in \mathbb{N}$ tales que, $\forall n$,

$$s(n) \leq c \cdot t(n) + d.$$

- $s(n) = \Omega(t(n))$ si $t(n) = O(s(n))$.
- $s(n) = \Theta(t(n))$ si $s(n) = O(t(n))$ y $s(n) = \Omega(t(n))$,

siendo $O(t(n))$ el límite superior asintótico; que es una medida de la mayor cantidad de tiempo que podría tardar el algoritmo en completarse, $\Omega(t(n))$ el límite inferior asintótico, que es una medida de la menor cantidad de tiempo que podría tardar el algoritmo en completarse y por último, $\Theta(t(n))$ representa el tiempo real de ejecución del algoritmo [8].
De esta forma, estas definiciones serán utilizadas más adelante para estudiar el orden de complejidad de los algoritmos y procesos involucrados en nuestro estudio.

Capítulo 2

Preliminares de álgebra lineal

En este capítulo fijaremos notaciones y recordaremos resultados y definiciones de álgebra lineal que utilizaremos posteriormente.

2.1. Espacios de Hilbert

En primer lugar, recordemos que un **espacio de Hilbert real** es un espacio euclidiano, que se denota por $\mathbb{H}_N$ siendo N la dimensión de dicho espacio y cuyos elementos $\boldsymbol{a}$ son vectores reales de dimensión N, que se representan de la siguiente manera:

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \\ \vdots \\ \boldsymbol{a}_N \end{bmatrix}$$

En él, están definidas las siguientes operaciones:

- **Suma de vectores**: si $\boldsymbol{a}$ y $\boldsymbol{b}$ son vectores del espacio; su suma $\boldsymbol{a}+\boldsymbol{b}$ también pertenece al espacio y está definida como:

$$\boldsymbol{a}+\boldsymbol{b} = \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_N \end{bmatrix} + \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1 + \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{a}_N + \boldsymbol{b}_N \end{bmatrix}$$

- **El producto** de un vector $\boldsymbol{a} \in \mathbb{H}_N$ y un escalar $c \in \mathbb{R}$, es el vector $c\boldsymbol{a}$

$$c \cdot \boldsymbol{a} = c \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_N \end{bmatrix} = \begin{bmatrix} c \cdot \boldsymbol{a}_1 \\ \vdots \\ c \cdot \boldsymbol{a}_N \end{bmatrix}.$$

- **El producto escalar** de dos vectores $\boldsymbol{a}$ y $\boldsymbol{b} \in \mathbb{H}_N$ que se suponen escogidos de una base ortonormal es

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^t \cdot \boldsymbol{b} = \sum_{i=1}^{N} \boldsymbol{a}_i \boldsymbol{b}_i.$$

- La norma de un vector $\boldsymbol{a} \in \mathbb{H}_N$ se define como:

$$\|\boldsymbol{a}\| = \left| \sum_k \boldsymbol{a}_k^2 \right|^{\frac{1}{2}} = \sqrt{\boldsymbol{a}^t \cdot \boldsymbol{a}}.$$

En el caso de que nuestro espacio sea únicamente de dos dimensiones, la norma del vector $\boldsymbol{a}$

$$\boldsymbol{a} = \begin{bmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \end{bmatrix}$$

es la longitud del vector en el plano: $\sqrt{\boldsymbol{a}_1^2 + \boldsymbol{a}_2^2}$. Por último, recordemos que un **vector unitario** es aquel cuya norma es 1.

2.2. Producto: usual, tensorial y vectorial

El **producto cartesiano** $\mathbb{H}_1 \times \mathbb{H}_2$ entre un espacio de Hilbert de dimensión m, que denotaremos por $\mathbb{H}_1$ con otro espacio de Hilbert de dimensión n, denotado por $\mathbb{H}_2$ es el espacio de Hilbert de dimensión $m+n$ obtenido concatenando vectores del espacio $\mathbb{H}_1$ con vectores del espacio $\mathbb{H}_2$. De modo más preciso, si $\xi_1 = \{e_1, \ldots, e_m\}$ es una base ortonormal de $\mathbb{H}_1$ y $\xi_2 = \{e_1^{'}, \ldots, e_n^{'}\}$ es una base ortonormal de $\mathbb{H}_2$, los elementos de $\mathbb{H}_1 \times \mathbb{H}_2$ están identificados en la base ortonormal $\xi_{12} = \{e_1, \ldots, e_m, e_1^{'}, \ldots, e_n^{'}\}$ por vectores de $\mathbb{R}^{n+m}$.
Por otro lado, el **producto tensorial** de ambos espacios, denotado por $\mathbb{H}_1 \otimes \mathbb{H}_2$ es un espacio de Hilbert de dimensión $n \cdot m$ con base ortonormal asociada $\xi_1 \otimes \xi_2 = \{e_i \otimes e_j^{'},\ i \in \{1, \ldots, m\},\ j \in \{1, \ldots, n\}\}$.
El **producto tensorial** de dos vectores $\boldsymbol{a} \in \mathbb{H}_1$, $\boldsymbol{b} \in \mathbb{H}_2$ es el vector $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ determinado en coordenadas respecto a la base $\xi_1 \otimes \xi_2$ por la expresión

$$\boldsymbol{c}_{ij} = \boldsymbol{a}_i \boldsymbol{b}_j$$

a partir de las coordenadas de $\boldsymbol{a}$ en ξ_1 y las de $\boldsymbol{b}$ en ξ_2. Por tanto, utilizando la base $\xi_1 \otimes \xi_2$ el espacio $\mathbb{H}_1 \times \mathbb{H}_2$ se identifica con $\mathbb{R}^{nm}$.

Estos vectores se utilizan en computación cuántica para denotar estados cuánticos puros y se dividen en dos grupos:

- Vectores **separables**: son aquellos vectores que son el resultado del producto tensorial realizado sobre otros dos vectores. Ejemplos: los vectores $\boldsymbol{e}_{00}$ o $\boldsymbol{e}_{11}$.
- Vectores **entrelazados**: son aquellos vectores que no se pueden escribir como el producto tensorial de otros dos vectores. Ejemplos: la combinación lineal de los vectores anteriores $\frac{1}{\sqrt{2}}(\boldsymbol{e}_{00} + \boldsymbol{e}_{11})$ es entrelazado.

Así, los vectores que forman la base del espacio $\mathbb{H}_1 \otimes \mathbb{H}_2$ son **separables**, pero esto no es cierto en general para las combinaciones lineales de los mismos.
Por último, recordemos también que el **producto escalar** de dos vectores reales $\boldsymbol{a} \in \mathbb{R}^m$ y $\boldsymbol{b} \in \mathbb{R}^m$ viene dado por:

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{k=1}^{m} \boldsymbol{a}_k \boldsymbol{b}_k.$$

2.3. Matrices

Las **matrices** representan operadores lineales en los espacios de Hilbert. Podemos sumarlas, multiplicarlas y aplicarlas sobre vectores. Recordemos también la notación básica de matrices y su aplicación sobre vectores con el siguiente ejemplo. Asumiremos sin mencionarlo que las matrices tienen las dimensiones adecuadas para realizar las operaciones indicadas.

Ejemplo 2.1.

$$\boldsymbol{U}\boldsymbol{V}\boldsymbol{a} = \boldsymbol{b}$$

Esto significa que en primer lugar aplicamos la matriz $\boldsymbol{V}$ sobre el vector $\boldsymbol{a}$ cuyo resultado es un vector al que a continuación se le aplica la matriz $\boldsymbol{U}$ dando como resultado el vector $\boldsymbol{b}$. Por otro lado, la matriz $\boldsymbol{I}_N$ denota la matriz identidad de dimensión $N \times N$, y cualquier matriz se representará como $\boldsymbol{U}[ij]$, la entrada correspondiente a la fila i y columna j de la matriz $\boldsymbol{U}$. Recordemos también que en un lenguaje matricial diremos que una aplicación es **lineal** si verifica las siguientes propiedades:

- $\boldsymbol{U}(\alpha \cdot \boldsymbol{a}) = \alpha \boldsymbol{U}\boldsymbol{a}$.
- $\boldsymbol{U}(\boldsymbol{a} + \boldsymbol{b}) = \boldsymbol{U}\boldsymbol{a} + \boldsymbol{U}\boldsymbol{b}$.

Además, diremos también que si $\boldsymbol{U}$ es una matriz, entonces $\boldsymbol{U}^k$ denota la **k-ésima potencia** de la matriz $\boldsymbol{U}$. Recordemos también diferentes tipos de matrices muy utilizadas en diferentes contextos.

Definición 2.2. Denotaremos por $\boldsymbol{U}^T$ a la **matriz traspuesta** de $\boldsymbol{U}$, es decir

$$\boldsymbol{U}^T[i,j] = \boldsymbol{U}[j,i]$$

El **producto escalar** de dos vectores de $\mathbb{H}_N$ se obtiene respecto a una base ortonormal mediante la expresión matricial

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^t \boldsymbol{b}.$$

Definición 2.3. Diremos que $\boldsymbol{U}$ es una matriz **inversible** si existe una matriz $\boldsymbol{V}$ que verifica:

$$\boldsymbol{V}\boldsymbol{U} = \boldsymbol{U}\boldsymbol{V} = \boldsymbol{I}$$

Tal matriz $\boldsymbol{V}$ si existe, entonces es única y se denota por $\boldsymbol{V} = \boldsymbol{U}^{-1}$.

Definición 2.4. Diremos que $\boldsymbol{U}$ es una **matriz unitaria** si verifica que:

$$\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$$

es decir, si su inversa coincide con su traspuesta.

Definición 2.5. Diremos que $\boldsymbol{U}$ es una **matriz de permutación** si es una matriz cuadrada y que presenta todo ceros en cada fila y columna excepto un único 1. *Todas* las matrices de permutación son unitarias. Las matrices unitarias preservan la distancia euclidiana, como veremos en el siguiente lema [1]:

Lema 2.6. *Si $\boldsymbol{U}$ es una matriz unitaria real y $\boldsymbol{a}$ es un vector real, entonces $\|\boldsymbol{U}\boldsymbol{a}\| = \|\boldsymbol{a}\|$.*

Demostración. El resultado se obtiene:

$$\|\boldsymbol{U}\boldsymbol{a}\|^2 = \langle \boldsymbol{U}\boldsymbol{a}, \boldsymbol{U}\boldsymbol{a} \rangle = (\boldsymbol{U}\boldsymbol{a})^t \boldsymbol{U}\boldsymbol{a} = \boldsymbol{a}^t\boldsymbol{U}^t\boldsymbol{U}\boldsymbol{a} = \langle \boldsymbol{a}, \boldsymbol{a} \rangle = \|\boldsymbol{a}\|^2 \tag{2.1}$$

□

2.4. Espacios complejos y producto escalar

Muchos algoritmos cuánticos (en especial el algoritmo base de este documento, el de Shor) presentan una parte de computación clásica, basada en las nociones de álgebra vistas, pero también presenta una parte de computación cuántica, basada en espacios vectoriales complejos. Como en los espacios reales, en los espacios complejos utilizaremos matrices y vectores complejos para representar la información, cuyas operaciones sobre ambas son similares al caso real excepto la *transposición* y el *producto escalar*. Para poder entender ambas operaciones, es necesario definir la operación **conjugación**: si $z = x + iy$, con $z \in \mathbb{C}$, $x, y \in \mathbb{R}$ e $i = \sqrt{-1}$, entonces el **conjugado** de z (se denota por $\overline{z}$) es $x - iy$.

Definición 2.7. Sea $\boldsymbol{U}$ una matriz compleja de dimensiones $n \times n$. Entonces, $\boldsymbol{V}$ es la matriz **adjunta** de $\boldsymbol{U}$ si:

$$\boldsymbol{V}[i,j] = \overline{\boldsymbol{U}[j,i]}$$

es decir, $\boldsymbol{V}$ es la matriz *conjugada* y *traspuesta* de $\boldsymbol{U}$. Se denota por $\boldsymbol{V} = \boldsymbol{U}^*$.

Definición 2.8. Una matriz compleja $\boldsymbol{U}$ de dimensiones $n \times n$ es una matriz **unitaria** si verifica que

$$\boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{I}$$

es decir, si la matriz *adjunta* de $\boldsymbol{U}$ coincide con su *inversa*.

Recordemos la definición del *producto escalar hermitiano* y sus propiedades, además de señalar también sus diferencias respecto al caso real. El producto entre los vectores $\boldsymbol{a}$ y $\boldsymbol{b} \in \mathbb{C}$ está definido de la siguiente manera

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{k=0}^{m} \overline{\boldsymbol{a}_k} \boldsymbol{b}_k \tag{2.2}$$

y determinado por las siguientes propiedades:

- $\langle \boldsymbol{a} + \boldsymbol{b}, \boldsymbol{c} \rangle = \langle \boldsymbol{a}, \boldsymbol{c} \rangle + \langle \boldsymbol{b}, \boldsymbol{c} \rangle,\ \forall \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{C}$.
- $\langle \alpha \boldsymbol{a}, \boldsymbol{b} \rangle = \overline{\alpha} \langle \boldsymbol{a}, \boldsymbol{b} \rangle,\ \forall \alpha \in \mathbb{C}$ y $\forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{C}$.
- $\langle \boldsymbol{b}, \boldsymbol{a} \rangle = \overline{\langle \boldsymbol{a}, \boldsymbol{b} \rangle}, \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{C}$.
- $\langle \boldsymbol{a}, \boldsymbol{a} \rangle > 0,\ \forall \boldsymbol{a} \in \mathbb{C}$ con $\boldsymbol{a} \neq 0$.

En otras palabras, un producto escalar hermitiano es una forma bilineal simétrica con forma cuadrática real y definida positiva. Esta definición generaliza el caso real, ya que también cumple la definición 2.2, de modo que para nuestros propósitos tiene sentido unificar en la misma noción los espacios de Hilbert reales y complejos.

Para el caso de espacios complejos el siguiente lema se demuestra de la misma manera que el lema 2.6 anterior [1]:

Lema 2.9. *Si* $\boldsymbol{U}$ *es una matriz unitaria compleja y* $\boldsymbol{a}$ *es un vector complejo, entonces* $\|\boldsymbol{Ua}\| = \|\boldsymbol{a}\|$.

Recordemos también que en espacios complejos podemos realizar el producto tensorial de matrices de dimensiones diferentes: si $\boldsymbol{U}$ es una matriz compleja de dimensión $m \times n$ y $\boldsymbol{V}$ es otra matriz compleja de dimensión $r \times s$, entonces $\boldsymbol{W} = \boldsymbol{U} \otimes \boldsymbol{V}$ es la matriz de dimensión $mr \times ns$ que aplicada sobre los vectores complejos de la forma: $\boldsymbol{c}_{ij} = \boldsymbol{a}_i \boldsymbol{b}_j$ presenta el siguiente resultado:

$$\boldsymbol{W}\boldsymbol{c}_{ij} = (\boldsymbol{U}\boldsymbol{a}_i)\boldsymbol{V}\boldsymbol{b}_j$$

Además, esta operación está bien definida debido a que cada vector $\boldsymbol{d}$ de dimensión rs (sea o no separable) se puede escribir como la combinación lineal de vectores de la base y de esta manera al aplicar la acción $\boldsymbol{Wd}$ ésta se realiza sobre las mismas combinaciones lineales utilizadas para definir al vector $\boldsymbol{d}$. Esto es:

$$\boldsymbol{d} = \sum_{i=1}^{r} \sum_{j=1}^{s} \boldsymbol{d}_{ij} \boldsymbol{e}_i \otimes \boldsymbol{e}_j,$$

entonces:

$$\boldsymbol{Wd} = \sum_{i=1}^{r} \sum_{j=1}^{s} \boldsymbol{d}_{ij} (\boldsymbol{U}\boldsymbol{e}_i) \otimes (\boldsymbol{V}\boldsymbol{e}_j)$$

Capítulo 3

Sistema RSA: claves pública y privada

Otro aspecto fundamental para poner en contexto el algoritmo de Shor es conocer el **sistema de cifrado RSA**. ¿Qué relación guarda dicho sistema de cifrado con nuestro algoritmo? El objetivo final de nuestro algoritmo es descomponer un número en factores primos, que a su vez son utilizados en el sistema RSA para poder cifrar y descifrar mensajes. Como veremos más adelante, una de las principales aplicaciones del algoritmo de Shor es la criptografía y supone una gran mejora respecto a la criptografía actual; debido a que consigue factorizar números en tiempo polinomial ($O(n)$) por la utilización de de un algoritmo cuántico, superando con creces el orden de complejidad exponencial ($O(n^m)$) que presenta este problema de factorización de enteros en el caso real y que supone la base de la utilidad práctica del algoritmo RSA. De esta manera, es necesario conocer los detalles presentes en el sistema RSA para poder comprender la importancia que tienen los números primos en la criptografía actual, y por tanto en el algoritmo de Shor.
Pasemos entonces a comentar las particularidades del sistema RSA. Partiremos de un intento de comunicación entre dos usuarios, A y B. A (emisor) desea enviar un mensaje a B (receptor) sin que nadie más se entere, pero en el medio de la transmisión hay usuarios indeseados (como C) que desean enterarse también del mensaje. Tanto A como B como C conocen la clave pública, pero únicamente B (el receptor) conoce la clave privada. De esta manera, pongamos el siguiente ejemplo:

Ejemplo 3.1. A quiere enviar el mensaje "Hola" a B. Para ello, cifra el mensaje con su clave pública para convertirlo en "&j@s", y tanto B como C reciben este mensaje encriptado "&j@s". Sin embargo, únicamente B (dado que posee la clave privada) puede decodificar el mensaje, y por tanto obtener el mensaje original "Hola".

De esta forma, es necesaria una combinación del par clave pública-privada para poder encriptar y desencriptar un mensaje. Pero podemos preguntarnos, ¿si yo realmente puedo encriptar dicho mensaje con la clave pública y sé el proceso que he realizado, debería saber desencriptar también dicho mensaje, no? Es una pregunta habitual que nos podemos hacer en esta situación, pero resulta que no es cierto. Para darnos cuenta, pensemos en el siguiente ejemplo; cuyas condiciones serán las que utilizaremos para explicar el algoritmo utilizado en RSA:

Ejemplo 3.2. Supongamos que nuestro mensaje es un vector real $\boldsymbol{s}$ de tamaño 6, y que la clave pública es una matriz $\boldsymbol{M}$ de dimensión 6×6. Por tanto, la operación necesaria para poder encriptar nuestro mensaje será multiplicar dicho vector $\boldsymbol{s}$ por nuestra matriz $\boldsymbol{M}$ y de esta manera obtendremos un vector resultante $\boldsymbol{t}$ de dimensión 6. Pues bien, para poder desencriptar y volver a obtener el mensaje original no podemos volver a aplicar la matriz $\boldsymbol{M}$ sobre nuestro vector encriptado $\boldsymbol{t}$, sino que lo que tenemos que hacer para poder desencriptar el mensaje es calcular la matriz inversa de $\boldsymbol{M}$ ($\boldsymbol{M}^{-1}$) y aplicar dicha matriz al vector $\boldsymbol{t}$, para obtener finalmente nuestro vector mensaje de partida $\boldsymbol{s}$. De esta manera, en este ejemplo el mensaje sería el vector $\boldsymbol{s}$, la clave pública la matriz $\boldsymbol{M}$ y la clave privada sería la matriz $\boldsymbol{M^{-1}}$.

Así, vemos como se ilustra claramente la idea de disponer de dos claves pública y privada para poder encriptar/desencriptar un mensaje. Una vez entendido dicho ejemplo, pasemos a comentar los pasos en los que se divide el algoritmo RSA:

1. Buscamos en $\mathbb{N}$ dos números primos distintos p y q suficientemente grandes.
2. Calculamos n y θ ($n = p \cdot q$ y $\theta = (p-1) \cdot (q-1)$).
3. Elegimos $e \in \mathbb{Z}$, $e > 1$ coprimo con θ y tal que $e < \theta$. Este paso es muy importante, ya que en caso de que ambos valores no fueran coprimos; no existiría el inverso de e módulo θ.
4. Obtenemos d como el inverso de e módulo θ ($d \cdot e = 1 (\mathrm{mod}\, \theta)$). Véase el lema 3.4.
5. Para encriptar, el texto encriptado es: $c = m^e (\mathrm{mod}\ n)$, correspondiendo m al mensaje que queremos codificar. Además, al par $(e,\ n)$ se le conoce como la **clave pública** de nuestro proceso.
6. Para desencriptar, aplicamos: $m = c^d (\mathrm{mod}\ n)$, donde al par (d, n) se le conoce como la **clave privada** de nuestro proceso.

Antes de presentar el enunciado y demostración del lema 3.4, recordemos brevemente que significaba dado un número entero $n > 0$ que dos números enteros sean
congruentes módulo n.

Definición 3.3. Dado $n > 0$, diremos que dos números enteros a, b son **congruentes módulo** n ($a = b(\mathrm{mod}\, n)$) con $n \in \mathbb{N}$ si el resto de dividir a entre n y b entre n coinciden.

Lema 3.4. *Sea $e \in \mathbb{N}$ tal que e es coprimo con θ. Entonces, $\exists\, d \in \mathbb{N}$ que satisface:*

$$e \cdot d = 1(mod\, \theta).$$

Demostración. Para demostrar este resultado, empezaremos utilizando el *teorema de Bezout*, que determina el máximo común divisor de dos números $a, b \in \mathbb{N}$ y una solución $x, y \in \mathbb{Z}$ de la ecuación $ax + by = \mathrm{mcd}(a, b)$. Así, tomemos en nuestro caso $e = a$ y $\theta = b$. Entonces, existen $x, y \in \mathbb{Z}$ tales que:

$$ex + \theta y = 1, \tag{3.1}$$

ya que por hipótesis, $\mathrm{mcd}(a, \theta) = 1$.
A su vez, esta expresión anterior implica que

$$ex = 1 - \theta y,$$

y que por tanto

$$ex = 1(\mathrm{mod}\, \theta).$$

Si $x > 0$, entonces escogemos $d = x$; mientras que si $x < 0$ (el caso x=0 está excluido por 3.1), seleccionaremos un $l \in \mathbb{N}$ tal que $x + l\theta > 0$ y así $d = x + l\theta \in \mathbb{N}$. Para ambos casos tenemos que $d \in \mathbb{N}$ y

$$ed = 1(\mathrm{mod}\, \theta),$$

tal y como queríamos demostrar. □

Una vez demostrado este lema, pasaremos a demostrar porque podemos recuperar nuestro mensaje una vez hemos comprobado que se cumplen las condiciones descritas previamente. Recordemos que $p, q \in \mathbb{N}$ son primos distintos, $n = p \cdot q$, $\theta = (p-1) \cdot (q-1)$ y que $e > 1$ es un entero coprimo con θ. Partamos de que nos llega el mensaje encriptado c, que recordemos que era $c = m^e(\mathrm{mod}\, n)$. A continuación, elevamos en ambos lados de la igualdad y así: $c^d = m^{ed}(\mathrm{mod}\, n)$. Por tanto, si conseguimos demostrar que $m^{ed} = m(\mathrm{mod}\, n)$ habremos terminado, ya que tendríamos que:

$$c^d = m^{ed} = m(\mathrm{mod}\, n).$$

Por definición, sabemos que $e \cdot d = 1(\mathrm{mod}\, \theta)$, que es lo mismo que decir que $e \cdot d = k\theta + 1$ para cierto $k \in \mathbb{Z}$. Es decir, queremos demostrar que

$$m^{k\theta+1} = m(\mathrm{mod}\, n),$$

(contando que $m \neq 0$, ya que en caso contrario ya se habría cumplido la igualdad) o lo que es lo mismo (dividiendo por m en ambos lados de la igualdad):

$$m^{k\theta} = 1(\mathrm{mod}\, n)$$

Llegados a este punto, para poder continuar con la demostración necesitaremos utilizar la función de Euler (φ). En este caso, como n es el producto de dos números primos distintos p y q, entonces $\varphi(n) = (p-1)(q-1)$, que es justo el valor de θ. Volvamos entonces a la demostración.
Por tanto, será lo mismo demostrar $m^{k\theta} = 1(\mathrm{mod}\, n)$ que $m^{k\varphi(n)} = 1(\mathrm{mod}\, n)$. Esta última identidad es consecuencia de que si $m^{\varphi(n)}$ fuese congruente con 1 módulo n, dado que:

$$m^{k\varphi(n)} = \underbrace{m^{\varphi(n)} \cdot m^{\varphi(n)} \cdot \ldots m^{\varphi(n)}}_{k \text{ veces}} = 1 \cdot 1 \cdot \ldots 1 = 1(\mathrm{mod}\, n)$$

Pero esto es exactamente lo que afirma el teorema de Euler, cuyo enunciado es que si dos números m y n son coprimos, entonces $m^{\varphi(n)} = 1(\mathrm{mod}\, n)$, y por tanto queda demostrado que podemos recuperar el mensaje original a partir del mensaje encriptado.

Para ilustrar el proceso descrito anteriormente, pongamos un ejemplo donde veamos todos los pasos anteriores:

Ejemplo 3.5.

- Supongamos $p = 11$ y $q = 23$ dos números primos.
- Calculamos entonces $n = 11 \cdot 23 = 253$ y $\theta = (p-1) \cdot (q-1) = 220$.
- Seleccionamos $e = 3$ un número coprimo con θ y menor que él.
- Calculamos ahora $d \in \mathbb{N}$ tal que $d \cdot e = 1(\mathrm{mod}\, \theta)$. En nuestro caso, obtenemos $d = 147$.
- Una vez obtenidos estos valores y sabiendo que el mensaje es $m = 2$, el mensaje encriptado será $c = m^e(\mathrm{mod}\, n) \implies c = 2^3 = 8(\mathrm{mod}\, 253)$, y el par $(e, n) = (3, 253)$ es la **clave pública**.
- Por último, desencriptamos el mensaje aplicando que $m = c^d(\mathrm{mod}\, n) \implies m = 8^{147} = 2(\mathrm{mod}\, 253)$, y el par $(d, n) = (147, 253)$ es la **clave privada**.

Capítulo 4

Funciones binarias, bits cuánticos y factibilidad

4.1. Funciones binarias

Antes de comentar los contenidos de esta sección recordemos la identificación entre espacios de Hilbert de dimensión finita:

$$(\mathbb{Z}_2, +, \cdot) \longrightarrow (\{0,1\}, \textbf{XOR}, \textbf{AND}),$$

que establece una correspondencia entre el cuerpo $\mathbb{Z}_2$ con las operaciones suma y producto usuales y el conjunto $\{0,1\}$ junto con las operaciones **XOR** (equivalente a la suma) y **AND** (equivalente a la multiplicación). En este documento utilizaremos la segunda forma por ser utilizada en la Ingeniería Informática, y por tanto más conocida y adecuada para las cuestiones de las que hablaremos.

4.1.1. Conceptos generales

Una **función binaria** f es una correspondencia entre $\{0,1\}^n$ y $\{0,1\}^m$, con $m, n \in \mathbb{N}$. De esta manera, considerando $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$, tenemos que los x_i con $i \in \{1, \ldots, n\}$ son los datos de entrada en la función; mientras que los y_j con $j \in \{1, \ldots, m\}$ son los datos de salida de la función.
Estas funciones son muy útiles, dado que nos permiten representar los valores básicos de las tablas de verdad, cadenas binarias e incluso (tal y como vimos en la sección 1.1) representar también números y otras estructuras. La mayoría de funciones de este tipo están definidas únicamente para 1 o 2 dimensiones; como la función **NOT** ($f(0) = 1$ y $f(1) = 0$), el **AND** binario ($f(0,0) = 0$,

$f(1,0) = 0$, $f(0,1) = 0$ y $f(1,1) = 1$), la **OR** ($f(0,0) = 0$, $f(1,0) = 1$, $f(0,1) = 1$ y $f(1,1) = 1$) o la **XOR** ($f(0,0) = 0$, $f(1,0) = 1$, $f(0,1) = 1$ y $f(1,1) = 0$). A continuación mostraremos las 3 últimas operaciones comentadas en el caso de estar trabajando en un espacio de dimensión mayor que 2. En ese caso, definimos

- **AND**: $f(x_1, \ldots, x_n) = 1 \Leftrightarrow$ si $x_i = 1$, $\forall i \in \{1, \ldots, n\}$.
- **OR**: $f(x_1, \ldots, x_n) = 1 \Leftrightarrow$ si $\exists i \in \{1, \ldots, n\}$ tal que $x_i = 1$.
- **XOR**: $f(x_1, \ldots, x_n) = 1 \Leftrightarrow$ si el número de índices $i \in \{1, \ldots, n\}$ tales que $x_i = 1$ es impar.

De hecho, estas operaciones binarias se pueden aplicar en pares de cadenas bit a bit. Para ilustrarlo, si x e y son cadenas binarias de dimensión n, entonces $x \oplus y$ es igual a:

$$z = (x_1 \oplus y_1, \ldots, x_n \oplus y_n)$$

Igual que en el ejemplo anterior, podemos definir también las operaciones **AND** y **OR** bit a bit para el caso de dos cadenas binarias de igual longitud. Es importante recordar que **no** son lo mismo que las funciones descritas anteriormente, no son operaciones en n dimensiones, si no que son la aplicación de n operaciones binarias, dónde cada operación de las anteriores conecta el i-ésimo bit de x con el i-ésimo bit de y para cada i. De esta manera, el producto interior binario se define aplicando primero la función **AND** bit a bit y después a cada uno de esos resultados se les aplica la función **XOR** tal y como se muestra a continuación:

$$x \bullet y = \mathbf{XOR}(x_1 \wedge y_1, \ldots, x_n \wedge y_n)$$

En casos como el de la operación $x \oplus y$, diremos que tenemos un circuito de n puertas binarias que computan la función binaria $f : \{0,1\}^r \longrightarrow \{0,1\}^n$, siendo $r = 2n$ y f definida por $f(x, y) = x \oplus y$. Una vez llegados a este punto, es importante comentar que dichas funciones binarias se representan mediante puertas lógicas, y estas se interconectan formando circuitos para obtener la salida deseada según los parámetros de entrada. Es lógico pensar que a mayor número de puertas en el circuito, mayor esfuerzo deberá ser realizado por el circuito para poder computar el resultado requerido y por tanto, mayor tiempo necesitará para poder calcular el valor final. Además, dichos circuitos deben de cumplir la siguiente propiedad: únicamente se pueden utilizar operaciones básicas y éstas solamente se pueden aplicar a valores calculados previamente. De esta manera, podemos construir circuitos que computen cualquier función que deseemos. El problema radica en la eficiencia que podemos conseguir para poder crear dichos circuitos. Relacionado con esta cuestión, hablaremos a continuación de las nociones de factibilidad de las funciones binarias, que nos permitirán afirmar de manera objetiva el grado en el que una función binaria puede ser computada eficientemente o no.

4.1.2. Funciones binarias factibles

Existen distintos niveles de complejidad en los que podemos encuadrar a las funciones binarias. Dichos niveles pueden ser definidos gracias a la Ingeniería Informática: por ejemplo, las funciones **AND** y **OR** son computables en un número lineal de pasos, y por tanto su orden de complejidad es $O(n)$, mientras que existen otras cuyo orden en exponencial, esto es, $O(n^m)$ con $m \in \mathbb{N}$. Para visualizar esta circunstancia de la diferencia entre los órdenes de complejidad, consideremos la tabla de verdad para la operación **XOR** exclusiva binaria:

$x\ y$	$x \oplus y$
0 0	0
0 1	1
1 0	1
1 1	0

Cuadro 4.1: Tabla de verdad de la operación XOR-exclusiva binaria

La tabla representa la salida que produce la función OR-exclusiva según los datos de entrada utilizados por la función. En general, una función binaria $f(x_1, \ldots, x_n)$ se define mediante una tabla de verdad que tiene de tamaño 2^n filas (una por cada posible combinación de los datos de entrada). Como es lógico, la dificultad radica en que a medida que aumenta el número de datos de entrada; el número de filas presentes en la tabla de verdad aumenta de manera exponencial. De esta manera, la representación de funciones binarias como tabla de verdad siempre es posible, pero no *factible*; debido a que las tablas asociadas a funciones con un gran de número de entradas serían demasiado largas para poder representarlas.
Como últimos contenidos de esta sección, comentaremos los conceptos de familia $[f_i] = \{f_1, \ldots, f_r\}$ de funciones binarias y la factibilidad de las mismas. En primer lugar, cada familia $[f_i]$ de cadenas binarias representa el conjunto formado por cada una de las funciones f_i que toman n_i datos de entrada. De esta manera, la familia $[f_i]$ determina una única función que actúa sobre cadenas de todas las longitudes consideradas; esto es: $f : \{0,1\}^n \longrightarrow \{0,1\}^n$, con $n = n_1 + \cdots + n_r$ generalizando también la dimensión de la salida generada por la función considerada. Una vez comentada esta cuestión, podemos indicar ya la definición de *factibilidad*:

Definición 4.1. Una familia de funciones $[f_i] = \{f_1, \ldots, f_r\}$ es **factible** si cada una de las funciones f_i es computable mediante circuitos de puertas lógicas de orden de complejidad $n^{O(1)}$.

4.2. Representación cuántica de argumentos binarios

En primer lugar, denotaremos por $|x\rangle = |x_n x_{n-1} \cdots x_0\rangle = |x_n\rangle \otimes |x_{n-1}\rangle \otimes \cdots \otimes |x_0\rangle$ al qubit x siendo $x_i \in \{0, 1\}$ con $i \in \{1, \cdots, n\}$ las coordenadas binarias de un número binario x. Ahora, sea $N = 2^n$. Entonces, cada vector básico en un espacio de Hilbert de dimensión N se corresponde con una cadena binaria de dimensión n. El **esquema de codificación estándar** asigna a cada índice $j \in \{0, \ldots, n-1\}$ la cadena binaria de n bits que representa dicho índice en notación binaria, añadiendo ceros al principio si es necesario. De esta manera, para poder representar una función binaria cualquiera ($y = f(x_1, \ldots, x_n)$) necesitaremos $n+1$ coordenadas binarias, lo que implica $2N = 2 \cdot 2^n = 2^{n+1}$ coordenadas matriciales. Realmente, lo que estamos computando es la función:

$$F(x_1, \ldots, x_n, z) = (x_1, \ldots, x_n, z \oplus f(x_1, \ldots, x_n))$$

Una propiedad importante de esta función es que es *invertible*, y además la inversa es ella misma. Veámoslo:

$$\begin{aligned} F(F(x_1, \ldots, x_n, z)) &= F(x_1, \ldots, x_n, z \oplus y) \\ &= (x_1, \ldots, x_n, (z \oplus y) \oplus y) = (x_1, \ldots, x_n, z) \end{aligned} \tag{4.1}$$

La segunda propiedad importante es que tenemos una matriz de permutación $\boldsymbol{P}_f$ de dimensión $2N \times 2N$ que se describe fácilmente: el único 1 de cada fila $x_1 x_2 \ldots x_n z$ está en la columna $x_1 x_2 \ldots x_n b$, donde $b = z \oplus f(x_1, \ldots, x_n)$. Sin embargo, este es el caso en el que la salida de la función f es solamente un bit. Generalmente, si la función f presenta m valores de salida $(y_1, \ldots, y_m)$, entonces la función F se aplica de la siguiente manera:

$$F(x_1, \ldots, x_n, z_1, \ldots, z_m) = (x_1, \ldots, x_n, z_1 \oplus y_1, \ldots, z_m \oplus y_m)$$

Por lo tanto, la matriz $\boldsymbol{P}_f$ sigue siendo una matriz de permutación, pero ahora tiene dimensión $2^{n+m} \times 2^{n+m}$.

Por otro lado, es importante comentar también lo que sucede en los casos en los que necesitamos h "bits de ayuda" para poder calcular la función f. Con este esquema, utilizando el sistema de computación clásica de "filas y columnas", toda nuestra información está contenida en $n' = n+m+h$ filas con la entrada x situada en la primera columna. De esta forma, cada fila representa un **qubit**, abreviatura de *bit cuántico*; mientras que las h filas de ayuda se conocen como qubits auxiliares. Estos qubits auxiliares se utilizan porque en la computación cuántica es necesario igualar la dimensión del conjunto de datos de entrada al de los datos de salida.

Como es lógico, escribir una matriz de dimensión $2^{n'} \times 2^{n'}$ solo para una permutación no es factible. Esta razón es suficiente para pensar utilizar los operadores $\boldsymbol{P}_f$ como piezas de código. Realmente, estos qubits son coordenadas de las cadenas binarias que representan los índices de estos programas. Dichas cadenas tienen dimensión n', y sus índices $1, \ldots, n'$ es lo que denotamos por **coordenadas cuánticas**. Por lo tanto, nuestro siguiente paso será determinar cuáles son las operaciones factibles dentro de la computación cuántica; que es lo que trataremos a continuación.

4.3. Factibilidad cuántica

Un algoritmo cuántico lo que hace es aplicar una serie de matrices unitarias a un vector inicial. Ante la pregunta de si es posible aplicar cualquier matriz unitaria que deseemos, la respuesta obviamente es no; debido a que los algoritmos cuánticos están diseñados para ser eficientes, y es por esta razón por la que debe de haber una restricción en las matrices que pensamos utilizar. Antes de seguir, recordemos ciertas particularidades de las matrices $\boldsymbol{P}_f$:

- El diseño de $\boldsymbol{P}_f$ no depende de la complejidad de la función binaria f.
- Aunque empleemos operaciones simples (como por ejemplo la $\mathbf{AND}(x_1, x_2, \dots, x_n)$, dicha matriz seguirá siendo demasiado grande para poderla computar de manera factible.

Por lo tanto, esta circunstancia nos permite plantear varias dudas: ¿Cómo podemos distinguir *operaciones básicas factibles*? ¿Qué debemos usar como variables de nuestras funciones? ¿Si tenemos un vector de tamaño 2^n, cuántas variables necesitaremos? La respuesta a estas preguntas se basa en darse cuenta que si conseguimos mantener el número k de argumentos de cualquier operación como una constante, entonces 2^k permanecerá constante. Así, podremos utilizar matrices de dimensión $2^k \times 2^k$ que aplicaremos sobre solo algunos argumentos. Estos argumentos **no** deben confundirse con las coordenadas $0, \dots, N-1$ de los espacios de Hilbert. Por otro lado, las coordenadas cuánticas empiezan siendo nombradas como $x_1, x_2, \dots, x_n$ igual que las cadenas binarias de entrada y se extienden para representar también los bits de salida de la función y los *bits acompañantes* que ya comentamos anteriormente.
Una vez entendidas estas cuestiones, la noción de factibilidad en matrices unitarias no es más que la extensión natural de la misma aplicada sobre circuitos binarios. Cualquier matriz unitaria $\boldsymbol{B}$ de dimensión $2^k \times 2^k$, siendo k constante y $k \leq 3$ es **factible**. Este tipo de matrices operan en cualquier subconjunto de k coordenadas cuánticas, dejando sin tocar el resto de $n' - k$ coordenadas.
A continuación, supongamos que $\boldsymbol{U}$ es una matriz unitaria cualquiera de dimensión $N \times N$. Entonces, diremos que es *factible* si existe una manera sencilla de crearla empleando matrices básicas, que son el resultado de realizar el producto tensorial entre una matriz $\boldsymbol{B}$ y matrices identidad aplicadas sobre las demás coordenadas cuánticas. Técnicamente, estamos suponiendo que $\boldsymbol{U}$ pertenece a la familia infinita $[\mathcal{U}_n]$ parametrizada por $\boldsymbol{n}$, siendo cada matriz $\boldsymbol{U} \in \mathcal{U}_n$ constructible mediante $n^{O(1)}$ matrices básicas.
Una vez comentado lo anterior, mostraremos a continuación la definición de que una *computación* cuántica sea factible:

Definición 4.2. Una computación cuántica $\boldsymbol{C}$ en s qubits es factible si:

$$\boldsymbol{C} = \boldsymbol{U}_t \boldsymbol{U}_{t-1} \dots \boldsymbol{U}_1,$$

donde cada $\boldsymbol{U}_i$ con $i \in \{1, \dots, t\}$ es una operación factible, y donde s y t están acotados por un polinomio de grado el número n de qubits de entrada.

4.4. Circuitos y matrices cuánticas : Hadamard y $U_1(\alpha)$

4.4.1. Introducción

Los circuitos cuánticos permiten representar el proceso de aplicar una serie de transformaciones cuánticas a un conjunto de datos de entrada para generar un conjunto de datos de salida. Dichos circuitos se pueden representar mediante matrices unitarias y las transformaciones que se llevan a cabo son las denominadas *puertas cuánticas*. Un ejemplo de un circuito de este tipo se presenta en la siguiente imagen:

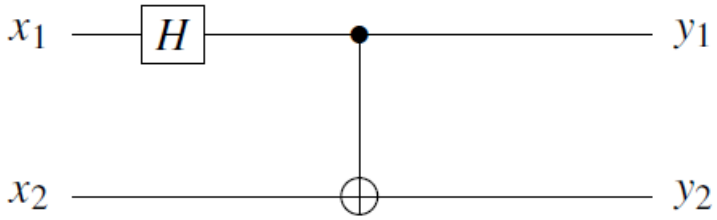

Figura 4.1: Ejemplo de circuito cuántico

De esta manera, podemos observar como se representa un circuito cuántico: las lineas horizontales (similares a las de un pentagrama) representan las entradas $x_1, \dots, x_n$ (en nuestro ejemplo $n = 2$). Por otro lado, el recuadro con la H representa una de las puertas cuánticas (en este caso la de Hadamard, que veremos en la siguiente subsección), y la línea vertical representa el origen (o control) en el punto negro situado en la línea 1 y el objetivo en la segunda de una puerta cuántica (en este caso la $\boldsymbol{CNOT}$) que no se muestra explícitamente. Así, para obtener el resultado tras aplicar el circuito a los bits de entrada se realiza la composición de las puertas cuánticas involucradas, de la misma manera que se hace un producto de matrices. En nuestro ejemplo, el resultado se obtendría al realizar el producto de matrices $\boldsymbol{CNOT} \cdot \boldsymbol{H}$.

Una vez indicado como funcionan y se escriben los circuitos cuánticos, pasaremos a comentar en las dos siguientes subsecciones los dos tipos de puertas cuánticas que tenemos que saber para poder comprender los pasos previos al algoritmo de Shor (debido a que van a ser usadas durante el proceso de determinar el periodo de nuestra función tras emplear la Transformada Cuántica de Fourier). Estas puertas son las denominadas puertas (o matrices) de **Hadamard** y las matrices $\boldsymbol{U_1(\alpha)}$.
A continuación, pasemos a comentar cada una de ellas:

4.4.2. Matrices de Hadamard

Consideraremos en nuestro caso que $N = 2^n$ para cierto $n \in \mathbb{N}$. Así:

Definición 4.3. La matriz de **Hadamard** $\boldsymbol{H}_N$ de orden N está definida recursivamente por $\boldsymbol{H}_2 = \boldsymbol{H}$ y para $N \geq 4$:

$$\boldsymbol{H}_N := \boldsymbol{H}_{N/2} \otimes \boldsymbol{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} \boldsymbol{H}_{N/2} & \boldsymbol{H}_{N/2} \\ \boldsymbol{H}_{N/2} & -\boldsymbol{H}_{N/2} \end{bmatrix}$$

También podemos utilizar $\boldsymbol{H}_1 = \begin{bmatrix} 1 \end{bmatrix}$ como base y de esta manera tendríamos:

$$\boldsymbol{H}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} \boldsymbol{H}_1 & \boldsymbol{H}_1 \\ \boldsymbol{H}_1 & -\boldsymbol{H}_1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \boldsymbol{H}_4 = \frac{1}{\sqrt{2}} \begin{bmatrix} \boldsymbol{H}_2 & \boldsymbol{H}_2 \\ \boldsymbol{H}_2 & -\boldsymbol{H}_2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Esta definición recursiva nos muestra las características más importantes de este tipo de matrices. Por ejemplo, implica que en general; $\boldsymbol{H}_N = \frac{1}{\sqrt{N}} \boldsymbol{A}$, donde $\boldsymbol{A}$ es una matriz cuyas únicas entradas son $+1$ y -1. Por otro lado, este tipo de matrices solo existen si su orden es 1, 2 o un múltiplo de 4. Veamos por qué:

Demostración. Dos filas cualquiera de una matriz de Hadamard de dimensión $n \times n$ son ortogonales. Para una matriz formada por 1 y -1, significa que dos filas cualquiera difieren exactamente en la mitad de sus valores, lo cual es imposible cuando n es un número impar. Cuando $n = 2 (\mathrm{mod}\, 4)$, dos filas que son ortogonales a una tercera fila no pueden ser ortogonales entre sí. Por tanto, juntando todo lo anterior: podemos afirmar que una matriz de Hadamard $n \times n$ solo puede existir si $n \in \{1, 2, 4k\}$, siendo $k \in \mathbb{N}$. □

4.4.3. Matrices $U_1(\alpha)$

Definición 4.4. La matriz $\boldsymbol{U_1(\alpha)}$ realiza un *control* sobre una rotación en el ángulo especificado (α). Así, podemos definir este tipo de matrices de la siguiente manera:

$$\boldsymbol{U_1(\alpha)}(x) = \begin{cases} |0\rangle, \text{si } x = |0\rangle \\ e^{i\alpha}|1\rangle, \text{si } x = |1\rangle \end{cases}$$

Para el caso de un qubit de entrada de dimensión 2 ($|x\rangle = |x_1 x_0\rangle$), esta matriz es de la forma:

$$\boldsymbol{U_1(\alpha)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{bmatrix} \tag{4.2}$$

Por tanto, un *control* en este caso no es más que la identidad si la entrada es el qubit 0 o aplicar una rotación de ángulo α si la entrada es el qubit 1.

Capítulo 5

Transformada Cuántica de Fourier (QFT)

Pasamos entonces a comentar los detalles detrás de la Transformada Cuántica de Fourier (QFT, siglas de *Quantum Fourier Transform*). En primer lugar, *definiremos* este tipo de matrices. Después, veremos su *representación como un circuito cuántico de puertas lógicas* y en la última sección de este capítulo explicaremos el principal uso que tiene dicha transformada en el algoritmo de Shor: *calcular el periodo de una función.*

5.1. Definición y principales propiedades

Definición 5.1. Sea $N = 2^n$ y sea $\omega = e^{2\pi i/N}$, denominada habitualmente como la **raíz principal N-ésima de la unidad**. Entonces, la matriz QFT ($\boldsymbol{F}_N$) de dimensión $N \times N$ es la matriz

$$\boldsymbol{F}_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \dots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{N-2} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{N-3} \\ \vdots & & & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{N-2} & \omega^{N-3} & \dots & \omega \end{bmatrix}, \tag{5.1}$$

es decir, la matriz que tiene en el lugar (i, j) el valor $\boldsymbol{F}_N[i, j] = \omega^{ij \bmod N}$.
A continuación, mostraremos el enunciado y la demostración de una de las propiedades fundamentales de este tipo de matrices.

Proposición 5.2. *Las matrices QFT son **unitarias**.*

Demostración. Veamos que $\boldsymbol{F}\boldsymbol{F}^* = \boldsymbol{F}^*\boldsymbol{F} = \boldsymbol{I}$. Así, para un elemento $\boldsymbol{F}_{ij}$ de $\boldsymbol{F}\boldsymbol{F}^*$, tenemos que

$$\boldsymbol{F}_{ij} = \frac{1}{N}\sum_{k=0}^{N-1} \omega^{jk}\omega^{-rk}.$$

En el caso en el que $j = r$ (elementos de la diagonal) , tendremos que sus elementos ($\boldsymbol{F}_{ii}$) serán de la forma

$$\boldsymbol{F}_{ii} = \frac{1}{N}\sum_{k=0}^{N-1} \omega^{ki}\omega^{-ki} = \frac{1}{N}\sum_{k=0}^{N-1} \omega^{ki-ki} = \frac{1}{N}\sum_{k=0}^{N-1} 1 = \frac{1}{N}\cdot N = 1, \forall i \in \{0,\dots,N-1\}$$

Por tanto, si a continuación conseguimos ver que el resto de elementos $\boldsymbol{F}_{ij} = 0$ con $i \neq j$ habremos concluido. Entonces, puesto que si $i \neq j$ entonces $\omega_0 = \omega^{j-i} \neq 1$ es otra raíz N-ésima de la unidad. Por tanto

$$\boldsymbol{F}_{ij} = \frac{1}{N}\sum_{k=0}^{N-1} \omega_0^k = \frac{1-\omega_0^N}{1-\omega_0} = 0.$$

□

5.2. Transformación a circuito cuántico

El objetivo principal de aplicar la Transformada Cuántica de Fourier es transformar la base computacional en la base de Fourier, entendiendo como base computacional aquella que permite formar los números en binario, mientras que la base de Fourier trabaja con estados en superposición. Para ello, es necesario convertir la expresión anterior 5.1 en un circuito cuántico, así que pasemos a ver dicho proceso.

Antes de empezar, recordemos que $N = 2^n$ y $\omega = e^{2\pi i/N}$. También, recordemos que el qubit $|x\rangle$ se puede representar mediante $|x_{n-1}\, x_{n-2}\, \dots x_1\, x_0\rangle$ y que a su vez esta expresión es equivalente a $|x_{n-1}\rangle \otimes |x_{n-2}\rangle \otimes \dots |x_{n1}\rangle \otimes |x_0\rangle$. Recordemos también que todo número natural x presenta su correspondiente expresión en número binario como

$$x = \sum_{k=0}^{n-1} x_k \cdot 2^k \tag{5.2}$$

Partimos entonces de nuestra expresión que transforma el qubit $|x\rangle$ en $\frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\omega^{xy}|y\rangle$.
Haciendo operaciones, tendremos que se da la siguiente igualdad (que demostraremos a continuación):

$$\frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\omega^{xy}|y\rangle = \underbrace{(|0\rangle + e^{2\pi i x/2}|1\rangle)}_{x_{n-1}} \otimes \underbrace{(|0\rangle + e^{2\pi i x/2^2}|1\rangle)}_{x_{n-2}} \otimes \cdots \otimes \underbrace{(|0\rangle + e^{2\pi i x/2^n}|1\rangle)}_{x_0} \tag{5.3}$$

Una vez llegado a este punto, lo que nos interesa es trabajar con las coordenadas binarias del número x y no con él directamente. Por tanto, aplicando 5.2, tendremos que la expresión 5.3 es equivalente a:

$$(|0\rangle+e^{2\pi i(\frac{x_0}{2}+\cancel{x_1}+\cancel{2x_2}+\cancel{\ldots})}|1\rangle)\otimes(|0\rangle+e^{2\pi i(\frac{x_0}{4}+\frac{x_1}{2}+\cancel{x_2}+\cancel{\ldots})}|1\rangle)\otimes\cdots\otimes(|0\rangle+e^{2\pi i(\frac{x_0}{2^n}+\frac{x_1}{2^{n-1}}+\frac{x_2}{2^{n-2}}+\cdots+\frac{x_{n-1}}{2})}|1\rangle). \tag{5.4}$$

Como podemos ver, el último qubit de la expresión anterior 5.4 presenta la interacción de todos los demás; ya que en el resto de casos podemos prescindir de los elementos de la forma $e^{2m\pi i}$ con $m \in \{1,\ldots,n\}$, ya que siempre van a ser 1, y por tanto solo tendremos en cuenta los elementos de la forma $\frac{x_k}{2^j}$ con $k \in \{0,\ldots,n-1\}$ y $j \in \{1,\ldots,n\}$.
Hemos visto como podemos escribir nuestra expresión de la Transformada Cuántica de Fourier en una expresión equivalente más manejable para poder representarla como un circuito de puertas cuánticas. Sin embargo, antes de visualizar y explicar como sería nuestro circuito; nos quedó pendiente demostrar el por qué se da la igualdad 5.3. Veámoslo:

Demostración. Escribiendo y como su correspondiente forma en número binario, tenemos que

$$(*) = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\omega^{xy}|y\rangle = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}e^{2\pi ix\sum_{k=0}^{n-1}\frac{y_k\cdot 2^k}{2^n}}|y\rangle, \tag{5.5}$$

además (utilizando que $e^{a+b}=e^a\cdot e^b$)

$$(*) = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}e^{2\pi ix\sum_{k=0}^{n-1}\frac{y_k\cdot 2^k}{2^n}}|y\rangle = \frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\prod_{k=0}^{n-1}e^{2\pi ix\frac{y_k}{2^{n-k}}}|y\rangle. \tag{5.6}$$

A continuación, recordemos que por definición: $|y\rangle = |y_{n-1}y_{n-2}\ldots y_0\rangle$. Por tanto

$$\sum_{y=0}^{N-1}|y\rangle = \sum_{y_{n-1}=0}^{1}\sum_{y_{n-2}=0}^{1}\cdots\sum_{y_0=0}^{1}|y_{n-1}y_{n-2}\ldots y_0\rangle$$

De esta manera, el cambio que realizaremos a continuación es el siguiente

$$\begin{aligned}(*) = &\sum_{y_{n-1}=0}^{1}\cdots\sum_{y_0=0}^{1}\frac{1}{\sqrt{N}}\prod_{k=0}^{n-1}e^{2\pi ix\frac{y_k}{2^{n-k}}}|y_{n-1}\ldots y_0\rangle = \\ &\sum_{y_{n-2}=0}^{1}\cdots\sum_{y_0=0}^{1}(\frac{1}{\sqrt{N}}\prod_{k=0}^{n-2}e^{2\pi ix\frac{y_k}{2^{n-k}}}|0\,y_{n-2}\ldots y_0\rangle + \frac{1}{\sqrt{N}}\prod_{k=0}^{n-2}e^{2\pi ix\frac{y_k}{2^{n-k}}}e^{2\pi ix/2}|1\,y_{n-2}\ldots y_0\rangle).\end{aligned} \tag{5.7}$$

La segunda igualdad de 5.7 se verifica porque estamos escribiendo esta suma de forma explícita, primero $y_{n-1}=0$ y luego $y_{n-1}=1$. Si nos fijamos también y recordando que $|ab\rangle = |a\rangle\otimes|b\rangle$, podemos extraer factor común.

Por tanto, la expresión 5.7 se puede escribir de la siguiente manera:

$$(*) = (|0\rangle + e^{\pi i x}|1\rangle) \otimes \underbrace{\sum_{y_{n-2}=0}^{1} \cdots \sum_{y_0=0}^{1} \prod_{y_0=0}^{n-2} e^{2\pi i x \frac{y_k}{2^{n-k}}} |y_{n-2} \dots y_0\rangle}_{\alpha_1} \tag{5.8}$$

Llegados a este punto, podemos darnos cuenta de que la expresión α_1 anterior es exactamente igual a la expresión inicial 5.3 pero con un qubit menos, por lo que si repetimos hasta el final el proceso presentado en las ecuaciones 5.7 y 5.8, obtendremos:

$$(*) = (|0\rangle + e^{2\pi i x/2}|1\rangle) \otimes (|0\rangle + e^{2\pi i x/2^2}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i x/2^n}|1\rangle) \tag{5.9}$$

tal y como queríamos demostrar. □

Tras demostrar los pasos necesarios para poder transformar nuestra expresión inicial en una más manejable, pasamos a mostrar la conversión de la Transformada Cuántica de Fourier como circuito cuántico. Para eso, debemos comentar que utilizaremos las **matrices de Hadamard** y las $\boldsymbol{U_1(\alpha)}$, que ya vimos en las subsecciones 4.4.2 y 4.4.3. Recordar también que se cumple la siguiente igualdad:

$$\frac{1}{\sqrt{2}}(|0\rangle + e^{\pi i x_k}|1\rangle) = \boldsymbol{H}|x_k\rangle, \tag{5.10}$$

siendo $\boldsymbol{H}$ la matriz de Hadamard de dimensión $N \times N$. Para ilustrar esta transformación y poder visualizarla sobre un circuito cuántico factible, supondremos 4 bits de entrada (x_0, x_1, x_2 y x_3), y los correspondientes 4 bits de salida (recordemos que en computación cuántica la dimensión de los datos de entrada y la de los de salida deben de coincidir). Así, el circuito correspondiente a la expresión:

$$\frac{1}{\sqrt{N}} \underbrace{(|0\rangle + e^{2\pi i(\frac{x_0}{2})}|1\rangle)}_{x_3} \otimes \underbrace{(|0\rangle + e^{2\pi i(\frac{x_0}{2^2}+\frac{x_1}{2})}|1\rangle)}_{x_2} \otimes \underbrace{(|0\rangle + e^{2\pi i(\frac{x_0}{2^3}+\frac{x_1}{2^2}+\frac{x_2}{2})}|1\rangle)}_{x_1} \otimes \underbrace{(|0\rangle + e^{2\pi i(\frac{x_0}{2^4}+\frac{x_1}{2^3}+\frac{x_2}{2^2}+\frac{x_3}{2})}|1\rangle)}_{x_0} \tag{5.11}$$

será el siguiente:

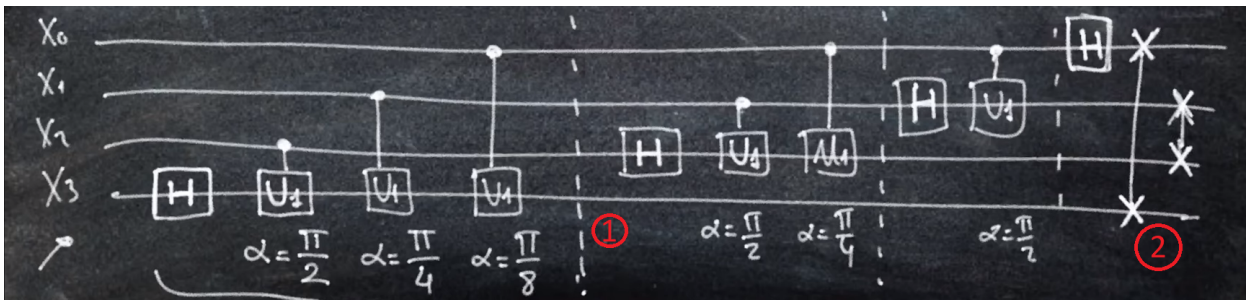

Figura 5.1: Circuito cuántico de la QFT de dimensión 4

Antes de pasar con la siguiente sección, en la que explicaremos la principal aplicación de la **QFT** para el Algoritmo de Shor, el cálculo del periodo de una función, explicaremos brevemente por qué se construye así el circuito explicando los pasos 1 y 2 llevados a cabo (que se muestran en la imagen 5.1):

1. Centrémonos en la expresión de x_0. Vemos que el término x_3 presente en el exponente está dividido por 2. Entonces, este 2 se cancelaría con el 2 presente en $2\pi i$ y de esta manera obtendríamos que dicha expresión es igual a aplicar la matriz de Hadamard al qubit $|x_3\rangle$ tal y como hemos visto en 5.10. Por tanto, debemos situar una puerta $\boldsymbol{H}$ como primer elemento en la línea de x_3. El siguiente paso sería determinar la puerta para el qubit x_2 dentro de la expresión de x_0. Así, en este caso debemos aplicar la puerta $\boldsymbol{U_1(\alpha)}$ sobre el qubit x_2, siendo en este caso la rotación correspondiente al ángulo $\alpha = \frac{\pi}{2}$. De manera análoga, y puesto que la expresión del exponente de e en el término correspondiente a y_3 es $2\pi i(\frac{x_0}{2^4} + \frac{x_1}{2^3} + \frac{x_2}{2^2} + \frac{x_3}{2})$, aplicaremos también una puerta $\boldsymbol{U_1(\alpha)}$ sobre x_1 y x_0, con $\alpha = \frac{\pi}{4}$ y $\alpha = \frac{\pi}{8}$ respectivamente. De manera análoga, repetimos el proceso para cada uno de los qubits de salida restantes (y_2, y_1 e y_0).

2. Una vez incluidas todas las puertas necesarias y con sus correspondientes valores α para el caso de las puertas $\boldsymbol{U_1(\alpha)}$, debemos darnos cuentas que tenemos los valores cambiados: en x_0 estamos almacenando la información que querríamos almacenar en x_3 y viceversa y lo mismo para x_1 y x_2. Por tanto, el paso final es realizar un *SWAP* o intercambio entre los qubits (representados mediante líenas verticales y X), para que de esta manera se almacene la información adecuada en cada uno.

Acabamos de ver un ejemplo de como convertir nuestra **QFT** en un circuito cuántico, explicando la inclusión de las puertas y como actúan sobre los qubits de entrada. Pese a que en la realidad esta función está ya implementada y que para utilizarla únicamente se incluye la librería correspondiente y después se ejecuta, es útil dar una explicación detallada ya que aporta una visión y un nivel de detalle muy importante para poder comprender en profundidad como funciona la Transformada Cuántica de Fourier.

5.3. Calcular el periodo de una función

En esta sección, veremos una aplicación práctica de la QFT. Pero antes de empezar con los conceptos de esta sección, recordaremos brevemente la noción de función periódica.

Definición 5.3. Una función periódica es una función $f : \mathbb{Z} \longrightarrow \mathbb{Z}/n\mathbb{Z}$ tal que $\forall x \in \mathrm{Dom}(f)$, $f(x) = f(x + k \cdot p)$, con $k \in \mathbb{Z}$ y donde $p \in \mathbb{R}$, es el periodo de nuestra función.

Ejemplo 5.4. Las funciones trigonométricas, cuyo periodo es 2π, son funciones periódicas.

De esta manera, en el caso de las funciones trigonométricas tendríamos infinitos puntos en los que podríamos obtener el valor del periodo de estas funciones, debido a que son funciones continuas. Ante la pregunta de si en nuestro caso es posible evaluar infinitos puntos, la respuesta es no, ya que estamos trabajando en un circuito con una función que trabaja con un número finito de qubits, de tal manera que si trabajamos con 3 qubits de entrada únicamente podemos representar números en binario del 0 al 7 o si usamos 4 qubits del 0 al 15, por poner un par de ejemplos. El número total de elementos que podremos representar lo denominaremos M. Por tanto, nuestro objetivo será encontrar el valor del periodo de nuestra función entre el conjunto de valores disponibles M.
Es importante también comentar que presentaremos un caso más sencillo realizando unas ciertas y fuertes suposiciones, para esclarecer la idea que hay detrás que es la misma. Así, evitaremos aumentar la complejidad en la explicación de manera innecesaria.
De esta manera, daremos las dos siguientes cuestiones por supuestas:

1. Una función f es periódica si, $f(x) = f(y) \Leftrightarrow y = x + k \cdot p$, siendo p el periodo de la función y $k \in \mathbb{Z}$.

2. El conjunto de valores disponibles M será un múltiplo del periodo.

Tras estas consideraciones, pasamos a mostrar la base del proceso que debemos realizar para encontrar dicho periodo. A dicho proceso se le conoce como realizar **mediciones parciales**: si tenemos dos qubits, medimos únicamente uno de ellos para ganar información sobre el segundo evitando que colapse, dado que no lo estamos midiendo. Recordemos también que en la computación cuántica es necesario conservar las dimensiones; partiremos del qubit $|x\,0\rangle$ y tras aplicar la función correspondiente lo llevaremos a $|x\,f(x)\rangle$. Veamos entonces un ejemplo de mediciones parciales:

Ejemplo 5.5. Supongamos que $M = 5$ y que tenemos una función f que funciona de la siguiente manera:

- $|0\,0\rangle \rightarrow |0\,4\rangle$
- $|1\,0\rangle \rightarrow |1\,3\rangle$
- $|2\,0\rangle \rightarrow |2\,1\rangle$
- $|3\,0\rangle \rightarrow |3\,3\rangle$
- $|4\,0\rangle \rightarrow |4\,2\rangle$

Entonces, si medimos el primer qubit en cada uno de los datos de salida de la función; y por ejemplo medimos un 2, sabemos que el segundo qubit de ese par ha de ser 1. ¿Por qué? Porque la única combinación posible de datos de salida de la función cuyo primer qubit sea un 2 es aquella en la que el segundo qubit es un 1. De forma análoga, supongamos que ahora medimos el segundo qubit. Si medimos en este caso un 3, estamos en un caso diferente al anterior; ya que hay dos valores del primer qubit en los que su pareja es un 3. En estos casos, el valor del primer qubit es un **estado en superposición** cuya expresión es de la forma: $\frac{1}{\sqrt{2}}(|1\rangle + |3\rangle)$.

Observando el proceso anterior, podemos ver como en los estados en superposición se sigue un **patrón** de la forma $\frac{1}{\sqrt{n}}|x_0+k\cdot p\rangle$, en donde la inclusión del factor $\frac{1}{\sqrt{n}}$ se debe a la necesidad de trabajar con un vector unitario. Nuestra intención es poder expresar dichos estados únicamente como $|k \cdot p\rangle$, que si nos fijamos es similar a la expresión anterior excepto por la adición del término x_0. Para eliminar dicho término de la expresión anterior y poder llegar a la expresión que deseamos utilizaremos la **QFT** vista en las secciones 5.1 y 5.2 de este capítulo. Empleando la **QFT**, determinaremos múltiplos de M/p, pero después podremos despejar y calcular p, que es nuestro objetivo desde el principio.
De esta manera, nuestro objetivo ahora será aplicar la **QFT** para poder demostrar:

Lema 5.6. *Dado M el número total de elementos y $|x_0 + k \cdot p\rangle$ la expresión de estados en superposición sin el factor multiplicativo, tendremos que:*

$$\sqrt{\frac{p}{M}} \sum_{k=0}^{\frac{M}{p}-1} |x_0 + k \cdot p\rangle \xrightarrow{QFT} \frac{1}{\sqrt{p}} \sum_{k=0}^{p-1} |k \cdot \frac{M}{p}\rangle \phi_k. \tag{5.12}$$

aplicando la Transformada Cuántica de Fourier.

Antes de comentar la demostración del lema, explicaremos el enunciado:

1. Partimos de $\frac{M}{p}$ elementos, y es por esta razón por la que el factor que multiplica antes de la suma es $\frac{1}{\sqrt{\frac{M}{p}}} = \sqrt{\frac{p}{M}}$, pero tras aplicar la QFT, pasamos a tener p elementos (y por tanto, en este caso el factor multiplicativo de la suma es $\frac{1}{\sqrt{p}}$).

2. El término ϕ_k es una fase global que no afecta a la medición final, y que por tanto podemos prescindir de él.

3. Si realizamos una medición antes de aplicar la QFT, podemos obtener cualquier valor debido al término x_0 que aparece sumando. Sin embargo, tras aplicar la QFT nos aseguramos obtener únicamente múltiplos de $\frac{M}{p}$.

Sin más dilación, pasemos entonces a ver la demostración:

Demostración. Antes de profundizar en la demostración, recordemos que:

1. M es un múltiplo del periodo p.

2. QFT $|x\rangle = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega^{xy} |y\rangle$.

Por lo tanto, aplicando la definición de la QFT a nuestro estado:

$$\sqrt{\frac{p}{M}} \sum_{k=0}^{\frac{M}{p}-1} |x_0+k\cdot p\rangle \xrightarrow{\text{QFT}} \sqrt{\frac{p}{M}} \sum_{k=0}^{\frac{M}{p}-1} (\frac{1}{\sqrt{M}} \sum_{y=0}^{M-1} e^{2\pi i \frac{(x_0+kp)y}{M}} |y\rangle) = \frac{\sqrt{p}}{M} \sum_{y=0}^{M-1} (\sum_{k=0}^{\frac{M}{p}-1} e^{2\pi i \frac{x_0 y}{M}} \cdot e^{2\pi i \frac{k\cdot p\cdot y}{M}}) |y\rangle) \tag{5.13}$$

Llegados a este punto, debemos separar en dos casos:
En el primero de ellos, tendremos que si y es múltiplo de $\frac{M}{p}$, entonces $e^{2\pi i \frac{k\cdot p\cdot y}{M}} = 1$. Además, tendremos que el coeficiente asociado a ese valor "y" será $\frac{\sqrt{p}}{M}(\frac{M}{p} \cdot e^{2\pi i \frac{x_0 y}{M}}) = \frac{1}{\sqrt{p}}\phi_y$
(lo denotamos por ϕ_y, que es una fase global que no afecta a la probabilidad de medir ese qubit).
Por otro lado, sabemos que dentro del conjunto M existen p múltiplos de $\frac{M}{p}$. Entonces, cuando nuestro "y" recorra todos esos valores, estará cumpliendo la condición anterior de ser múltiplo de $\frac{M}{p}$ p veces, y la de no ser múltiplo en el resto de casos. Por tanto, y dado que los p coeficientes de estos estados son $\frac{1}{\sqrt{p}}$, tendremos que la probabilidad total será 1, por lo siguiente:

$$\underbrace{|\frac{1}{\sqrt{p}}|^2 + \cdots + |\frac{1}{\sqrt{p}}|^2}_{p \text{ veces}} = 1.$$

Esto quiere decir que el coeficiente de todos aquellos elementos que no sean múltiplos de $\frac{M}{p}$ será 0, llegando así a la solución deseada:

$$\sqrt{\frac{p}{M}} \sum_{k=0}^{\frac{M}{p}-1} |x_0 + k\cdot p\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{p}} \sum_{k=0}^{p-1} |k \cdot \frac{M}{p}\rangle \phi_k. \tag{5.14}$$

□

Lo que acabamos de demostrar nos permite afirmar que cuando tengamos un estado en superposición y lo midamos, siempre vamos a obtener un múltiplo de $\frac{M}{p}$. Una vez hecho esto, ejecutaremos el circuito varias veces, obteniendo un valor múltiplo de $\frac{M}{p}$ diferente (o no) en cada una de ellas. De esta manera, lo único que tenemos que hacer para determinar dicho periodo es calcular el **máximo común divisor** de estos valores calculados previamente y despejar p de la siguiente expresión (considerando que x_i con $i \in \{0, \ldots, n\}$ son los múltiplos de $\frac{M}{p}$ encontrados tras ejecutar el circuito):

$$m.c.d(x_i) = \frac{M}{p} \Longrightarrow p = \frac{M}{m.c.d(x_i)} \tag{5.15}$$

Antes de pasar con el capítulo en el que explicaremos los pasos que presenta el algoritmo de Shor, debemos mostrar el circuito que debemos utilizar para poder obtener dicho periodo. Un ejemplo con 3 qubits de entrada aparece representado en la siguiente imagen:

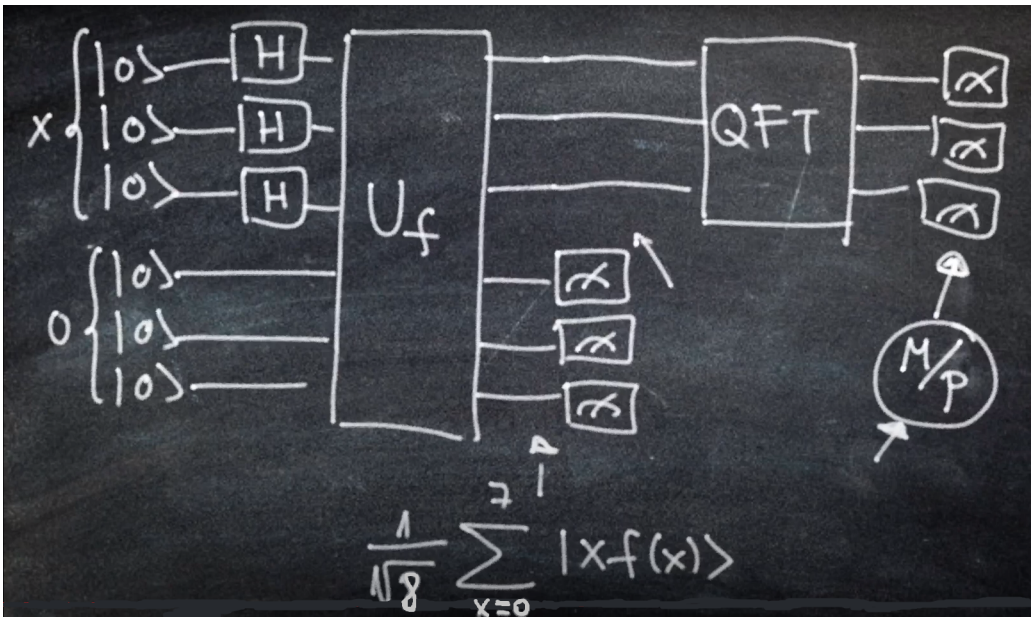

Figura 5.2: Circuito cuántico para 3 qubits de entrada para determinar el periodo

Como podemos ver en la imagen 5.2, partimos de 3 qubits de entrada y los correspondientes 3 qubit 0 para equivaler las dimensiones. A continuación, creamos la superposición de estados aplicando una puerta de Hadamard a cada qubit $x_0, x_1 y x_2$ involucrado, y después aplicamos nuestra puerta cuántica $\boldsymbol{U}_f$ asociada a la función f a todos los pares $|x_j 0\rangle$, con $j \in \{0, 1, 2\}$.
A continuación, medimos la superposición de estados generada; cuya interacción en este caso es de 8 elementos (ya que $2^3 = 8$), y así podemos representar 8 números (del 0 al 7) en binario.

Una vez hecha esta medición para conseguir que el primer qubit colapse hacia donde nos interese (no utilizaremos el resultado de esta primera medición), aplicaremos a $x_0, x_1 y x_2$ la QFT y una vez aplicada, medimos la salida generada por esta función. Es *esta* medición la que nos dará siempre un múltiplo de $\frac{M}{p}$, y al obtener varios valores de estos múltiplos; podremos aplicar el máximo común divisor y despejando p como en la expresión 5.15, obtendremos el periodo de la función; que es lo que queríamos desde un principio.

Capítulo 6

El algoritmo de Shor

Llegados al último capítulo del documento y tras haber comentado todos los preliminares necesarios para poder entender el algoritmo de Shor, explicitaremos los pasos en los que se divide el algoritmo y también presentaremos algunos aspectos importantes del mismo: división en parte clásica y cuántica, y eficacia respecto a la computación actual.

6.1. Descripción del algoritmo: pasos y ejemplo de aplicación

El algoritmo en sí se divide en dos partes: una **parte cuántica**, que es la que hemos visto sobre todo en este documento, que consiste en encontrar el periodo de una función, y después una **parte clásica**, que emplea el resultado del periodo calculado en la anterior fase para poder descomponer un número en factores primos, objetivo final de nuestro algoritmo.
La parte cuántica es fundamental, ya que aunque se pueda determinar el periodo de una función mediante un método clásico; este sería mucho más costoso, porque habría que evaluar demasiados puntos para poder llevar a cabo dicho cálculo y convertiría el algoritmo en impráctico.
Antes de empezar con la explicación, debemos conocer un par de conceptos que serán empleados en el algoritmo. El primero de ellos es la llamada **raíz cuadrada no trivial módulo** (N), que definiremos a continuación:

Definición 6.1. Diremos que $x \in \mathbb{N}$ es una **raíz cuadrada no trivial módulo** (N) si $x^2 = 1 (\mathrm{mod}\, N)$ y $x \in (2, N-2)$

Ejemplo 6.2. 5 es una raíz cuadrada no trivial módulo (8), ya que el resto de dividir $5^2 = 25$ entre 8 es 1 y además $5 \in (2, 6)$.

El otro concepto que definiremos es el denominado **orden de un natural módulo** (N):

Definición 6.3. El **orden** de $m \in \mathbb{N}$ módulo (N) es **el menor entero** $r \in \mathbb{Z}$ tal que $m^r = 1(\text{mod}\, N)$.

Ejemplo 6.4. 10 es el orden de 7 módulo (11); ya que $7^{10} = 1(\text{mod}\, 11)$ y $7^j \neq 1(11)$ con $j \in \{1, \ldots, 9\}$.

Tras haber comentado estos dos conceptos, pasamos a ver los resultados en los que se sustenta fundamentalmente la teoría aplicada en nuestro algoritmo.

Lema 6.5. *Dado un número N producto de 2 números primos distintos p y q, encontrar la raíz cuadrada no trivial módulo (N) es equivalente a encontrar p y q.*

Demostración. Sea $N = p \cdot q$. Recordemos que por definición, si x es una raíz cuadrada no trivial módulo (N) esto quiere decir que $x^2 = 1(\text{mod}\, N)$ y además que $x \in (2, N-2)$.
Así:

$$x^2 = 1(\text{mod}\, N) \Longrightarrow x^2 - 1 = 0(\text{mod}\, N) \Longrightarrow (x-1)\cdot(x+1) = 0(\text{mod}\, N) \Longrightarrow (x-1)\cdot(x+1) = k\cdot N. \tag{6.1}$$

Entonces, y recordando que $N = p \cdot q$, nuestro objetivo será demostrar que p es un factor de $(x-1)$ y que q lo es de $(x+1)$ o bien que p es un factor de $(x+1)$ y q lo es de $(x-1)$.
Para ver esto, utilizaremos que $x \in (2, N-2)$ y así $x - 1 < N$ y $x + 1 < N$. Por tanto, ni $x-1$ ni $x+1$ pueden ser múltiplos de N; ya que para ser múltiplos deben ser mayores o iguales que N. Como N es producto de dos factores primos, la única opción posible es que uno de los factores, supongamos p sea factor de $(x-1)$ y que el otro q lo sea de $(x+1)$.
Por último, calculando el mcd de cada uno de los términos $x+1$ y $x-1$ respecto de N, se tiene que $\text{mcd}(x-1, N) = p$ y que $\text{mcd}(x+1, N) = q$. □

De esta manera, si somos capaces de calcular la raíz cuadrada no trivial módulo (N), también podremos calcular los factores primos a emplear en la factorización utilizada en el algoritmo de Shor. ¿Cómo podemos encontrar dicha raíz cuadrada no trivial x? Para ello, debemos darnos cuenta que la siguiente función $f(i) = x^i(\text{mod}\, N)$ con $i \in \mathbb{N}$ es periódica, donde su periodo es exactamente **el orden de x módulo (N)**, cuya definición vimos antes.
De esta forma, la idea detrás del algoritmo se basa en escoger un valor dentro del intervalo $(2, N-2)$, construimos la función $f(i) = x^i(\text{mod}\, N)$ y calculamos el periodo. Si el periodo obtenido es 2, ya hemos terminado; puesto que si el periodo de esa función es 2 hemos encontrado una raíz cuadrada no trivial, tal y como queríamos.

En general, realmente nos vale con encontrar *cualquier periodo par*, ya que si por ejemplo tenemos un periodo $p = 6$; es decir, nuestra función es $x^6 (\mathrm{mod}\, N)$, realmente esta expresión es equivalente a $(x^3)^2 (\mathrm{mod}\, N)$, y de esta forma nuestra raíz cuadrada no trivial sería x^3.
Por último, solo nos queda saber como debemos de escoger este valor dentro del intervalo $(2, N-2)$ para poder obtener un periodo par. Pero, a priori esta decisión es azarosa, ¿cómo podemos asegurarnos de escoger un valor adecuado? Pues aquí es donde entra en juego el segundo resultado que utilizaremos, pero antes de verlo enunciaremos y demostraremos un lema que es necesario para poder demostrar nuestro resultado.

Lema 6.6. *Sea $N \in \mathbb{N}$ impar. Entonces, al menos la mitad de los elementos de $(\mathbb{Z}/N\mathbb{Z})^\times$ tiene orden par.*

Demostración. Supongamos que el orden r de x es impar. Entonces,

$$(-x)^r = (-1)^r x^r = (-1)^r = -1 (\mathrm{mod}\, N).$$

Por tanto, $-x$ debe tener orden $2r$, que es par. De esta manera, al menos la mitad de los elementos de $(\mathbb{Z}/N\mathbb{Z})^\times$ tiene orden par. □

Ahora sí, pasamos enunciar y demostrar el segundo resultado fundamental para el algoritmo de Shor:

Lema 6.7. *Sea $N \in \mathbb{N}$ impar con descomposición en factores primos $N = p_1^{\alpha_1} \cdots p_m^{\alpha_m}$. Sea además un número al azar $x \in (2, N-2)$ coprimo con N y r su orden. Entonces, la probabilidad de que al mismo tiempo r sea par y $x^{\frac{r}{2}} \neq -1 (mod\, N))$ es mayor o igual que $1 - (\frac{1}{2})^m$*

Demostración. Por el *Teorema chinés de los restos*, escoger un $x \in (\mathbb{Z}/N\mathbb{Z})^\times$ al azar es equivalente a escoger $x_i \in (\mathbb{Z}/p_i^{e_i}\mathbb{Z})^\times$ para cada p_i de manera aleatoria, con $i \in \{1, \ldots, m\}$ tal que $x_i = x (\mathrm{mod}\, N)$. Sea ahora r el orden de x y sea r_i el orden de cada x_i. En particular, $x^{r/2} \neq 1 (\mathrm{mod}\, N)$. De esta manera, lo que queremos ver es lo mostrado en el enunciado, que la probabilidad de que r sea par o que $x^{r/2} = 1 (\mathrm{mod}\, N)$ es mayor o igual que $1 - (\frac{1}{2})^m$.
Como los primos $p_1, \ldots, p_m$ son distintos, se tiene que $r = \mathrm{mcm}\ (r_1, r_2, \ldots, r_m)$, donde mcm denota el mínimo común múltiplo.
Supongamos que r es impar. Entonces, esto solo sucederá si y solo si cada uno de los r_i son impares. Aplicando el Lema 6.6 anterior, sabemos que cada r_i es impar con una probabilidad de al menos $\frac{1}{2}$. Entonces, r es impar con una probabilidad de al menos $(\frac{1}{2})^m$.

Ahora, supongamos que r es par. En este caso aún debemos comprobar si no se da el caso en el que $x^{r/2} = \pm 1 (\mathrm{mod}\, N)$. Aplicando el *Teorema chinés de los restos*, esta situación solo pasa cuando $x^{r/2} = \pm 1 (\mathrm{mod}\, p_i^{e^i}), \forall\, p_i$. La probabilidad de escoger un x tal que cumpla alguno de los dos casos anteriores es $2 \cdot 2^{-m} = 2^{-m+1}$, y combinando ambas probabilidades obtenemos una probabilidad de éxito de al menos $(1 - 2^{-k})(1 - 2^{-k+1}) \geq 1 - 3 \cdot 2^{-k}$. □

De esta manera, escogiendo pocos números podemos asegurar con una probabilidad elevada que alguno de ellos presentará un periodo par.
Para ilustrar todo lo explicado en este capítulo hasta ahora, proponemos el siguiente ejemplo de aplicación del algoritmo de Shor:

Ejemplo 6.8. Trataremos de descomponer el número 33 en factores primos. Para ello, seleccionamos en primer lugar un número al azar en el intervalo $(2, 31)$; por ejemplo, el número 10. Ahora, vemos que $10^2 = 100 = 1 (\mathrm{mod}\, 33)$; y por tanto 10 es una raíz cuadrada no trivial módulo (33). Por lo tanto, aplicando el lema 6.5, tendremos que los factores primos de 33 serán el $\mathrm{mcd}(10 - 1, 33) = \mathrm{mcd}(9, 33) = 3$ por un lado y el $\mathrm{mcd}(10 + 1, 33) = \mathrm{mcd}(11, 33) = 11$ por el otro. Por tanto, los factores primos de 33 son 3 y 11.

Ya hemos visto todos los pasos necesarios para poder explicar de manera completa el algoritmo de Shor y un ejemplo sencillo de aplicación del mismo. Para terminar, en la última sección comentaremos brevemente cuestiones referidas a la eficiencia del algoritmo.

6.2. Principal aplicación y eficiencia del algoritmo

Como vimos en el capítulo 3, el **sistema RSA** constituye una de las bases en las que se sustenta la criptografía actual, y vimos en su caso la importancia de los números primos. De esta manera, la principal aplicación del algoritmo de Shor no es otra que la criptografía. Este algoritmo es tan poderoso que si se consiguiera implementar en un ordenador cuántico podría terminar con la criptografía actual; debido a que la factorización de primos necesaria para poder determinar las claves pública y privada presenta orden de complejidad exponencial empleando la criptografía clásica; mientras que el orden de complejidad empleando el algoritmo de Shor y la computación cuántica es polinómico, siendo radicalmente superior en velocidad al caso clásico. De esta manera, este algoritmo conseguiría en muchísimo menos tiempo encontrar la factorización en primos de cada número resultado de la encriptación correspondiente, de tal manera que los sistemas actuales de contraseñas y seguridad serían muy vulnerables a esta solución. Este aumento de rapidez respecto al caso clásico se debe al empleo del algoritmo cuántico basado en los capítulos 4 y 5, que permiten encontrar el periodo de la función evaluada en muchos menos intentos que en el caso clásico, que al necesitar muchos más puntos dificulta y ralentiza la labor de dicho algoritmo.

Pese al principal inconveniente que acabamos de comentar respecto a la brecha de seguridad que se puede generar si se consigue desarrollar este algoritmo; viene también acompañado de buenas noticias: el algoritmo de Shor puede suponer una nueva base para un sistema de encriptación cuántica mucho más seguro que el actual y que permita mantener la seguridad y la privacidad de los usuarios en la red en cualquiera de sus ámbitos.

Aún nos queda mucho por aprender de la computación cuántica, pero estoy seguro de que de la misma forma en la que se ha descubierto y estudiado este algoritmo, aparecerán muchos más que nos permitirá a los usuarios navegar de la manera más cómoda y segura posible en la nueva era de la computación cuántica que se avecina en los próximos años.

Bibliografía

[1] Lipton, Richard J. y Regan, Kenneth W.: *Quantum Algorithms via Linear Algebra*, The MIT Press, 1ª Ed. (2014).

[2] Scherer, Wolfgang: *Mathematics of Quantum Computing: An Introduction*, Springer, 1ª Ed. (2019).

[3] Shor, Peter W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, *SIAM J. Comput.*, **26**(5), 1997, 1484–1509.

Referencias webgráficas:

[4] Guillermo Alonso. *Playlist de vídeos de los pasos necesarios para entender el algoritmo de Shor.* `https://www.youtube.com/playlist?list=PLhYoqmIacCv9IryISV7HhsXucZAUL2WGb`. Visitada el 20 de julio del 2022.

[5] José Vicente Álvarez Bravo. *Definición de orden de complejidad de un algoritmo.* Enlace: `https://www2.infor.uva.es/~jvalvarez/docencia/tema5.pdf`. Visitado el 2 de julio del 2022.

[6] Tomás Domínguez. *Matrices de Hadamard.* Enlace: `https://idus.us.es/bitstream/handle/11441/43427/Matrices%20de%20Hadamard.pdf?sequence=1&isAllowed=y`. Visitado el 15 de julio del 2022.

[7] Fang Xi Lin. *Shor's Algorithm and the Quantum Fourier Transform.* Enlace: `https://cpb-us-w2.wpmucdn.com/u.osu.edu/dist/7/36891/files/2019/05/Fangxi-LinMcGillShorsQuantumFactoringDiscreteLogAlgorithms.pdf`. Visitado el 20 de julio de 2022.

[8] Abhishek Yadav, traducido por Acervo Lima. *Diferencias entre los diferentes órdenes de complejidad de un algoritmo.* Enlace: `https://es.acervolima.com/diferencia-entre-big-oh-big-omega-y-big-theta/`. Visitado el 2 de julio del 2022.

[9] Varios autores. *Definición del OR-exclusivo*. Enlace: `https://en.wikipedia.org/wiki/Exclusive_or`. Visitado el 2 de julio del 2022.

[10] Varios autores. *El algoritmo de Shor*. Enlace: `https://es.wikipedia.org/wiki/Algoritmo_de_Shor`. Visitado el 20 de julio del 2022.